5 Concluding the Analysis

We choose $s = 1$ Recall that $s' = s$ (see Section 4.2). By Lemma 3 we have $sd = 2$, $gp = ld = 34$, $c = 36$. On taking equality in Equations 7, 10, 13, 14, 20, 21, we obtain $hd = 36$, $d = 432$, $b = 214$, $c' = 144$, $a = 2 \cdot 216^2 = 93,212$ (incidentally, these values satisfy Equations 6, 8 and 9) and $q + 4 \leq 2^{21}$; by Equation 4 it suffices to set $e = 22$. For each block we need to pay for one global insertion and $\log n - e$ local insertions (note that the cost of the first $e - 1$ local insertions is covered by the charge for the global insertion). Equations 15–19 and 22–25 are not used here. From Lemma 17, we conclude:

Theorem 1 *The number of rotations for sorting a* $\log n$*-block sequence is bounded by* $2500n + O(n/\log n)$*. (Recall that the number of rotations dominates the overall cost of the sort.)*

6 Acknowledgements

We thank Rajamani Sundar for his very careful reading of the paper which uncovered several serious omissions in an earlier draft. We also thank S. Muthukrishnan for his help and advice regarding figure drawing.

References

[C93] R. Cole. On the Dynamic Finger Conjecture for splay trees. Part II: The proof. Submitted for publication.

[Luc88a] J.M. Lucas. *Arbitrary splitting in splay trees.* Technical Report No. DCS-TR-234, Computer Science Department, Rutgers University, 1988.

[Luc88b] J.M. Lucas. *Canonical forms for competitive binary search tree algorithms.* Technical Report No. DCS-TR-250, Computer Science Department, Rutgers University, 1988.

[ST85] D.D. Sleator, R.E. Tarjan. *Self-adjusting binary search trees.* JACM, 3(1985), 652–686.

[STT86] D.D. Sleator, R.E. Tarjan, W.P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. In *Proceedings Eighteenth Symposium on Theory of Computing*, 1986, 122–135.

[T85] R.E. Tarjan. *Sequential access in splay trees takes linear time.* Combinatorica, 5(4), 1985, 367–378.

[Su89] R. Sundar. Twists, turns, cascades, deque conjecture, and scanning theorem. In *Proceedings Thirtieth Annual Symposium on Foundations of Computer Science*, 1989, 555–559.

[W86] R. Wilber. Lower bounds for accessing binary search trees with rotations. In *Proceedings Twenty Seventh Symposium on Foundations of Computer Science*, 1986, 61–69.

following the rotation, u is no longer on L_{new}. If rg is heavy on L_{new}, then by Invariant 14(ii)a applied to L and L_{new}, there are no heavy nodes of L_{new} in rg's left subtree nor in u's left subtree; so, again, following the rotation u is no longer on L_{new}. A similar argument applies if rg is the right guard of L_{new}.

4.5.2 Removing Debits

We show how to restore Invariants 1–4 and 6–7. For each node that was on a newest lazy tree L, but is not on a lazy tree derived from L, we charge the removal of its debit, if any, to the node's potential associated with lazy tree L. So we are only concerned with nodes for which the age of their newest lazy tree is unchanged. For such nodes, the invariants are all restored exactly as in Section 4.3.

Restoring Invariants 1–4 result in the same maximum charge of two small debits and three large debits to a promoted node; this holds regardless of how many lazy trees the promoted node belonged to. Debit removal charged to the node relinquishing the debit is treated as before; this takes care of Invariant 6 as well. It remains to consider Invariant 7.

For a promoted node to be charged as node z of the argument in Section 4.3, it must be a heavy node on the left (resp. right) extreme path of its right (resp. left) lazy tree prior to the split. Likewise, for a node to be charged as node y, it must be a heavy node on its right (resp. left) lazy tree prior to the split and not on the left (resp. right) extreme path. So as a promoted node, a node is charged no more than before. The same node may receive additional charges, but as light nodes on other newer lazy trees; these charges are paid for by the potentials associated with the node as a light node in these newer lazy trees. So as before, Equations 20 and 21 suffice to enable Invariants 1–4 and 6–7 to be restored.

Again, for the more general case, to restore Invariant 11 Equations 22–24 suffice.

To satisfy any further restrictions obeying Property 3, we need to cover a charge of up to $\alpha \cdot md$ for each lazy tree created by the split. For each such lazy tree, the charge is given to its root. However, this cannot be applied solely to the charge made to promoted nodes, as several new lazy trees may share a common root. Instead, the charge is made to each node, either as a light node or for the oldest lazy tree, as a heavy node. This requires modifying Equation 23 to Equation 24 as before, and modifying Equation 22 as follows:

$$c' \geq 4md + \alpha \cdot md \tag{25}$$

4.5.3 Summary

The presence of multiple levels of lazy trees does not alter the previous analysis of a global insertion, stated in Lemma 10 (except that the constraint of Equation 25 is introduced). The analysis of local insertions is also unaffected, for the discussion of Section 4.4 carries over unchanged; so Lemma 6 also continues to hold. In summary, we have shown:

Lemma 17 *The cost of a local access is $c + s + 1$ and the cost of a global access is $e(3 \log n + 1)gp + 5gp \log n$ units.*

In addition, we note that Lemma 11 continues to hold with Equation 22 replaced by Equation 25 (but now in Lemma 11(ii), "block" is interpreted to mean the newest lazy tree older than the lazy tree in question, to which nodes on the left path belong).

Lemma 16 *Let w be a node on lazy tree L promoted in Case $2'$ or $2.1'$. Let w' be a promoted node in v's left subtree on an older lazy tree L_{old}, if any. Then*

(i) w' is a right descendant of w.

(ii) $lazy_rank(w) \geq g_rank(w')$.

(iii) $lazy_rank(w) + \frac{1}{gp} reserve(w) + jump(v) \geq g_rank(v)$.

Proof. The proof of (i) is very similar to the proof of Lemma 14. (ii) follows from Corollary 1(i). The proof of (iii) uses the argument of Case 2 in Section 4.3. •

Case $3'$. The inserted item is to the right of the lazy tree root, r.

The new lazy tree rooted at r must be provided with a right guard. The method followed is identical to that used in Case $2'$ for providing the new lazy tree rooted at v with a right guard. Lemma 16 applies here too.

It remains to analyze the cost of the split operations. Clearly, for the nodes on the right (resp. left) access path, the cost of the promotions is at most $gp \cdot \log n$, a total of $2gp \cdot \log n$ units for the two access paths. Lemmas 15 and 16 imply that for each promoted node on the right (resp. left) access path the associated promoted nodes form a chain descending to the left (resp. right); let $x_1, x_2, \cdots, x_k$ be a maximal path of such nodes, in descending order. Lemmas 15 and 16 show that $lazy_rank(x_i) \geq g_rank(x_{i+1})$, for $i = 1, \cdots, k-1$ and $lazy_rank(x_k) + \frac{1}{gp} reserve(x_k) + jump(v) \geq g_rank(v) \geq g_rank(x_1)$. (Note that the applications of Lemmas 15 and 16 will be interleaved if there is an interleaved sequence of successively older left and right lazy trees.) We conclude that a further $2gp \cdot \log n$ units of potential pay for the promotion of the nodes x_i.

So, as before, the cost of the split operation is at most $4gp \log n$ and Lemma 8 continues to hold.

Lemma 8 *The promotions in a global insertion cost at most $4gp \log n$ units, where for each split lazy tree there is a charge of $2gp$ times the increase in global rank along the right split path and of $2gp$ times the increase in global rank along the left split path.*

It should be clear that Invariants 14–17 are maintained over the course of these promotions. When the promotions are complete there are no pseudo-global nodes remaining, so at this point Invariants 14–16 will have been restored.

Finally, we need to consider the effect of the guard restoration process. We claim that if the guard restoration process is applied to lazy tree L, for each of the promoted nodes, the newest lazy tree to which it belonged was L. Then it is easy to see that Invariants 14–16 continue to hold. To show the claim we demonstrate one case; the others are similar. Suppose that rg, the right guard of L, is in a couple with heavy node u of L (see Figure 30). Suppose

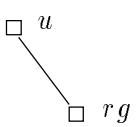


Figure 30: Guard restoration

that u is also on a newer lazy tree L_{new}. Then u is light on L_{new}. If rg is light on L_{new}, then

some older lazy tree, L_{old}. then $\tilde{v}$ must be on the left path of L_{old} (by Lemma 13), so $\tilde{v}$ was already promoted. $\tilde{v}$ will become the right guard of a new lazy tree formed from L. If a left descendant, w, of $\tilde{v}$, on the left path of L has already been promoted (there can be at most one such node) then w becomes the root of this new lazy tree. If not, w is defined to be the first heavy node on the left path of L which is a proper descendant of $\tilde{v}$; it is promoted. Finally, if w has a non-empty right subtree in L, the right guard restoration process is applied to $\tilde{v}$. Finally, suppose that $\tilde{v}$ was not heavy on any older lazy tree. Let v be the nearest descendant node on the left path of L which is heavy or pseudo-heavy. v is promoted if it is not already promoted. It is then handled in the same way as node $\tilde{v}$ earlier in this case.

The charging for the promotions is analyzed shortly. The following lemmas are helpful.

Lemma 14 *Let v be a traversed node on lazy tree L. Suppose that w' is a node in v's right subtree, where v is promoted in Case $1'$ or $1.1'$; in addition, suppose that v and w' are on an older lazy tree L_{old}. Let $w_1, \cdots, w_k$ be the nodes on L promoted by the right offset promotions. Then w' is a proper left descendant of w_k.*

Proof. Note that as w' is on L_{old}, an older lazy tree including v, w' must be in the range (v, w_k), by Invariant 14(ii)a. Suppose, for a contradiction, that w' has an ancestor x on the path from v to w_1 in the range (v, w_1). See Figure 29. (If this is not the case then w' is a

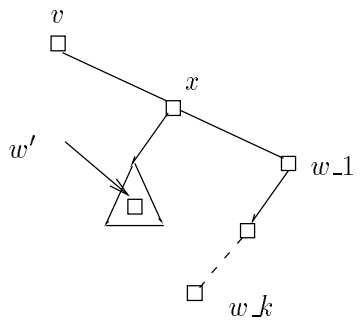


Figure 29: Proof of Lemma 14

proper left descendant of w_k.) x is global as both w' and w_1 are global. As x is a global node, by Invariant 16, x is heavy on some tree L' older than L (it is not heavy on L as it was not promoted). Let x' be the first node on the path from v to x, in the range $(v, x]$, on L', which is heavy on L', or which is already promoted. Then x' is promoted in whichever of cases $1', 1.1', 2', 2.1'$ applies to L'. But if x' is promoted then w_1 would not be promoted, a contradiction. •

The next lemma is shown by the same argument as used in Case 1 in Section 4.3.

Lemma 15 *Let $w_1, \cdots, w_k$ be the chain of nodes promoted on lazy tree L by the right offset promotions for node v. In addition, let w' be a promoted node, if any, in v's right subtree on an older lazy tree L_{old}. $lazy_rank(w_i) \geq g_rank(w_{i+1})$, for $i = 1, \cdots, k-1$; similarly, $lazy_rank(w_k) \geq g_rank(w')$. Finally, $lazy_rank(w_k) + jump(v) \geq g_rank(v)$.*

In addition, we note:

to w_1, in the range (v', w_1) has been promoted. Again, if the right offset promotions are not performed, a new lazy tree is created exactly as in Case $1'$.

If the left guard, lg, is separated from the remainder of L (through being accessed from its right child), and, v', the leftmost heavy node in L, is traversed, then the promotions described in the previous paragraph are performed; but if lg is separated from the remainder of L without v' being traversed (which can arise only if lg is the root of the large lazy block tree) then the new left guard for L is provided by the promoted node, if any, nearest to the root, r, of L, in the range (lg, r), on the path from lg to r. (See Figure 27.) If there is no such promoted

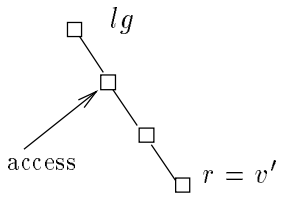


Figure 27: Separating the left guard

node, as in Case $1.1'$, the right offset promotions for r are performed.

Again, we need to provide the new lazy tree rooted at v with a right guard. As in Case 2 of Section 4.3, let x be the rightmost heavy descendant of v in the imaginary tree associated with its lazy block tree. If $x \neq v$ we proceed as in Case 2 of Section 4.3. If $x = v$ we proceed as follows. Let w' be v's left child in the lazy block tree for L (if v does not have such a child then the lazy block tree rooted at v will comprise only one node and so can be ignored henceforth). Let w be the first node on the path from v to w', in the range (w', v), which has already been promoted, if any (there will be at most one promoted node on this path). If there is no such node, set $w = w'$ and promote it. w becomes the root of a new lazy tree formed from L with right guard v. Finally, if w has a non-empty right subtree in L, the right guard restoration process is applied to v.

Case 2.1′. $\tilde{v}$ is a light node on the left path of L and $\tilde{v}$ is accessed from its right child (a node which is not on L). See Figure 28.

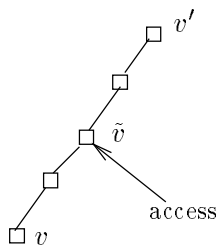


Figure 28: Case $2.1'$

v' is defined to be the nearest heavy or pseudo-heavy ancestor of $\tilde{v}$; v' must be on the left external path of L. v' is handled as in Case $2'$, above. Suppose $\tilde{v}$ is a heavy node on

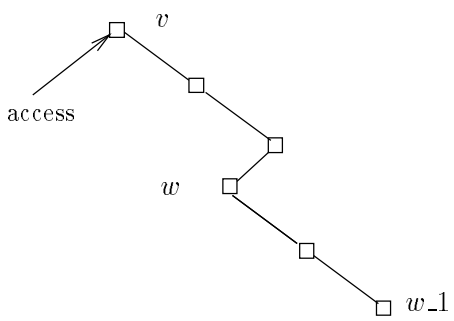


Figure 25: Case 1′

range (v, w_1) has been promoted. If such a node has been promoted, there will be one such node; let it be denoted w. Then w becomes a root of a lazy tree formed from L; its left subtree is empty and its right subtree contains exactly the heavy nodes of L in v's right subtree. v provides the left guard for this new lazy tree. The right guard for the lazy tree rooted at w is obtained in the same way as w_1's right guard in Case 1.

Case 1.1′. See Figure 26. v' is a light node of L, which is not on the left path of L, and v' is

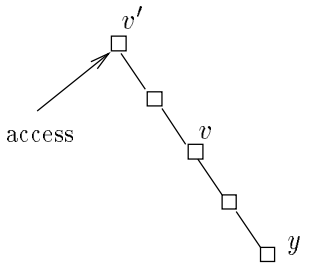


Figure 26: Case 1.1′

accessed from its left child, which is a node not on L.

Suppose v' is a heavy node on some older lazy tree, L_{old}; then v' is on the right path of L_{old} (by Lemma 13) and so v' was already promoted. Suppose that v' is straddled by L's heavy nodes x and y, where y is a descendant of v'. Let v be the promoted node nearest to y on the path from v' to y, in the range $[v', y)$, if any. If $v \neq v'$, v is made the root of a lazy tree formed from L; its left guard is the nearest node on the path from v' to v, in the range $[v', v)$ which has been promoted (in fact, this must be v'). Its right guard is handled as in Case 1. While if $v = v'$ or if v' was not heavy on any older lazy tree, then the right offset promotions for $v = v'$ are performed.

Case 2′. v is a heavy node on the left extreme path and it is accessed from its right child, u.

v' is defined to be the nearest heavy or pseudo-heavy ancestor of $\tilde{v}$; v' must be on the left external path of L. v and v' are promoted as in Case 2 of Section 4.3. As in Case 1′, the *right offset promotions* for v' are performed exactly when no global node on the path from v'

By inspection plus induction, Invariants 14–16 are true on creation of a lazy tree L_{new}; also, they remain true as the extreme paths of the lazy trees are traversed.

We need to reconsider the analysis of splits. Again, in turn, we consider the promotions, the consequential removal of debits and the recreation of the complete lazy tree potential for the lazy trees created by the split. Actually, the arguments concerning Invariants 8–10, 12–13 are unchanged from Section 4.3 and so we will not discuss further the recreation of the complete lazy tree potential.

4.5.1 The Promotions

For each split lazy tree, our goal in a split is to promote the same nodes as in Section 4.3 (remember, this needs to be interpreted symmetrically for left lazy trees); however, this may prove too expensive because of the recursive containment of lazy trees. So, sometimes, instead of promoting a node v, heavy on lazy tree L, we will promote a node w which is light on L but heavy on an older lazy tree, and such that w is an ancestor of v.

We consider the promotions to be performed on one lazy tree at a time, oldest first. Invariants 14 and 15 are maintained, but modified as follows.

Suppose the promotions have been applied to L_{old} but not to L_{new} (see Figure 24). All the

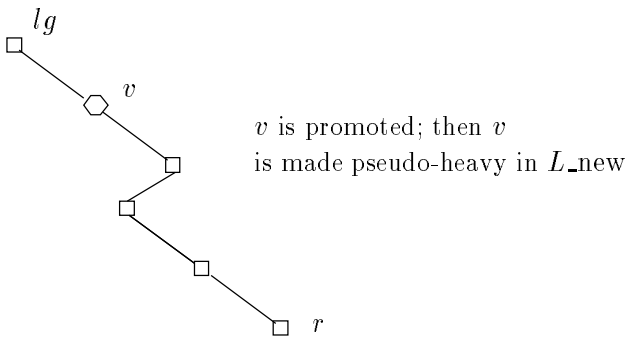


Figure 24: Pseudo-heavy nodes

promoted nodes which are on L_{new} are made into pseudo_heavy nodes of L_{new}. In addition, let lg be the left guard of L_{new} and r its root. Suppose that lg is an ancestor of r. Let v be a node on the path from lg to r in the range (lg, r). If v is promoted, then v also becomes a pseudo-heavy node of L_{new}. A similar rule applies with respect to the right guard of L_{new}.

Invariant 17 *Invariant 14(ii)(a) hold with respect to the heavy and pseudo_heavy nodes of L_{new}, as do Invariant 15(i) and 16.*

Next, we explain how the previous promotion procedure is modified. Consider the promotions on right lazy tree L.

Case 1′. See Figure 25. v is a heavy node in the lazy block tree, which is not on the left extreme path. In addition, v is accessed from its left child, u.

v is promoted as in Case 1 of Section 4.3. w_1 is defined as in this Case 1. The right offset promotions for v are performed exactly when no global node on the path from v to w_1, in the

Lemma 13 *Let w be on lazy tree L_{old}. Suppose w is also on a newer lazy tree L_{new}. Then w is on an extreme path of L_{old}.*

Proof. Invariant 14(ii)a applies to L_{new} and L_{old}. Let u and v be the items of L_{new} straddling the root of L_{old}, as in the statement of the invariant. Since w is on L_{new}, w is on the path from u to v in the range (u, v) (if $u < v$) or (v, u) (if $v < u$). If w does not lie on an extreme path of L_{old}, then some item of L_{old} must lie outside the range (u, v) (or (v, u)), which contradicts Invariant 14(ii)a. •

Invariant 15 *Let L be a lazy tree.*

(i) The nodes of L, apart from its root, are all heavy or light on L.

(ii) Apart possibly for its root, all of L's heavy nodes carry their lazy potential.

(iii) Each of L's light global nodes is a heavy node in some older lazy tree.

(iv) Each node, v, heavy on some lazy tree L, is a light node of all the newer lazy trees to which it belongs.

Invariant 15 implies Requirement 1(ii) and 1(iii), above.

Before giving the next invariant, a few definitions are helpful. Let L be a lazy tree and let u be a node of its large lazy block tree. v is an *L-neighbor* of u if v is also a node of the large lazy block tree and u and v enclose no other node of the large lazy block tree. Suppose that u is an ancestor of its L-neighbor v; *the (u, v)-neighbor path* comprises those items on the path from u to v in the splay tree which are in the range (u, v), if $u < v$, or (v, u), if $v < u$.

Invariant 16 *See Figure 23. Let L be a lazy tree. Let u and v be L-neighbors, with u the*

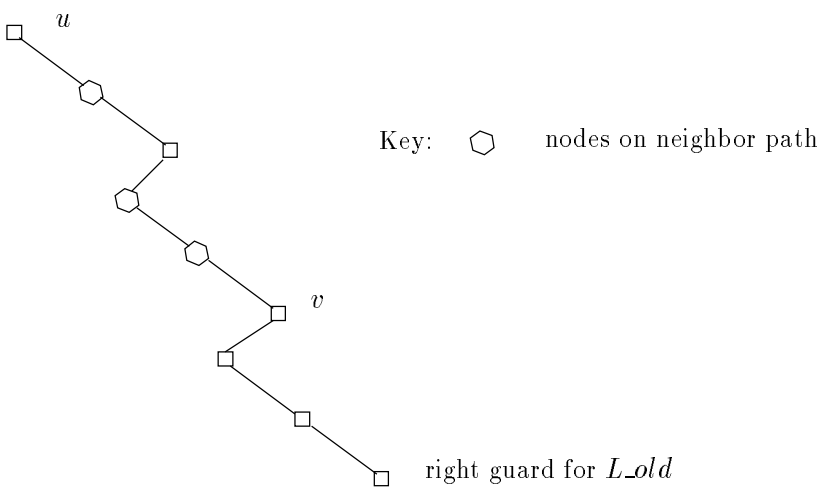


Figure 23: A (u, v)-neighbor path

ancestor of v. Let N denote the (u, v)-neighbor path. If N includes a global node, there is a lazy tree L_{old}, older than L, rooted at u, such that every node in N is on L_{old}. Further suppose that u is a left (resp. right) ancestor of v. Then the right (resp. left) guard for L_{old} is either v or a left (resp. right) descendant of v.

Requirement 1. Suppose u is traversed in the current insertion. Let L be the newest lazy tree to which u belongs.

(i) Let v be an ancestor of u on L. Then L is also the newest lazy tree to which v belongs.

(ii) u is either the root of L, or a heavy or light node of L; if a heavy node it carries a lazy potential defined with respect to L.

(iii) If u is a heavy node of an older lazy tree, then it is a light node of L.

Items (i) and (ii) enable the traversal of node u to be treated as part of the traversal of an extreme path of L; the previous analysis continues to apply. However, a new issue arises because u may also belong to older lazy trees. We maintain u's potentials with respect to each such lazy tree. Maintaining the l-potentials and l-black potentials might appear problematic, for it may involve the spending of k-spares, $k \geq 1$. But we note that each node, u, is heavy in at most one lazy tree, and this is the only lazy tree with respect to which u carries l-potentials and l-black potentials and to whose heavy nodes it transfers spares (for in all newer lazy trees that contain u, u is light). So there is only one lazy tree with whose l-potentials and l-black potentials u is concerned, and this is the only lazy tree to which u contributes spares or from which u draws spares. Thus the l-potentials and l-black potentials can be maintained as before. Otherwise, the analysis of an extreme path traversal is unchanged, apart possibly for the removal of debits following the maintaining of Invariant 13. We discuss this at the same time as we show how to remove debits following the new split operation.

The following invariant characterizes the overlap of lazy trees.

Invariant 14 *(i) Let L_a and L_b be two lazy trees of the same age. Then the two open intervals defined, respectively, by the guards of L_a and of L_b are disjoint.*

(ii) Let L_{old} be a lazy tree and let r be its root. Let L_{new} be another, newer, lazy tree. Then

(a) If r lies strictly between the guards of L_{new} then the guards of L_{old} lie between the guards of L_{new}. In addition, let u and v be the items in the large lazy block tree for L_{new} straddling r (r may or may not be an item in this large lazy block tree). Then L_{old} plus its guards lies between u and v. (Recall that the large lazy block tree for lazy tree L comprises the lazy block tree for L plus the guards for L.)

(b) If r is strictly outside the closed interval defined by the guards of L_{new} then the open intervals defined by the guards of L_{old} and L_{new}, respectively, are disjoint.

(c) Suppose that r is the right (resp. left) guard for L_{new}. Let d_{new} be the rightmost (resp. leftmost) item in the lazy block tree for L_{new}. Then the left (resp. right) guard of L_{old} is either equal to or to the right (resp. left) of d_{new}.

Thus, in some sense, an older lazy tree is either contained in a newer lazy tree or is disjoint from it. In the next two lemmas we show several consequences of this invariant.

Lemma 12 *Requirement 1(i) holds.*

Proof. Let u and v be as in Requirement 1(i). Suppose that v belongs to lazy tree L_{new}, newer than L. Then L and L_{new} obey Invariant Invariant 14(ii)a. By assumption, only an extreme path of L_{new} is traversed; so the inserted item lies outside the range spanned by the guards of L_{new}; without loss of generality, suppose that the inserted item is to the right of the right guard of L_{new}. By Invariant 14(ii)a, all of L is to the left of L_{new}'s right guard; so the only items of L that are traversed must also be on L_{new}. •

4.4 Lazy Trees and Local Accesses

We need to reconsider the analysis of local accesses to take the presence of lazy trees into account (see Remark 2). Note that in such an access only the left paths of lazy trees can be traversed. In fact, only two modifications are needed. First, we need to consider couples comprising two heavy nodes. But these are analyzed as in a global insertion: They have $0 \leq s + 1$ amortized cost, for as noted in Remark 3, no spares are spent on paying for changes to the j-potentials and/or j-black potentials. Second, we may need to modify the guards of some lazy trees so as to maintain Invariant 13. As in Section 4.3, this does not occasion a change to the analysis of the insert operation. So the results of Lemma 6 continue to apply.

4.5 Multiple Level Lazy Trees

It is convenient to refer to the lazy trees encountered so far in the paper as *right lazy trees*. Analogous lazy trees, called *left lazy trees*, in which the roles of left and right are interchanged are considered in this section. While they are not needed to prove the result of this paper, they lead to a more general result in this section, without adding significantly to the complexity of the current section.

Because a local access path may include nodes of a current lazy tree we may seek to make the root of a lazy tree a heavy node carrying a lazy potential in a new lazy tree. We therefore generalize the form of the lazy trees. Now, a "block," or rather metablock, in a lazy tree may itself be another lazy tree.

In order to distinguish the ages of the different lazy trees, we number the blocks, in insertion order, by $1, 2, \cdots$. A lazy tree is labeled by the number of its corresponding creating block. When a lazy tree is split its parts keep the same age label.

Let L be a lazy tree; its skeleton plus its lazy block tree comprise the nodes on L. The nodes on L are also said to *belong* to L. A node may be on an extreme path of several lazy trees. For each lazy tree to which a node belongs it carries a separate potential. However, a node may carry only one debit, as before. Invariants 1–4 and 6–7 should be interpreted with respect to the newest lazy tree to which the node carrying the debit belongs.

We define a new lazy tree, L_{new}, to *contain* an old lazy tree, L_{old}, if the root, r, of L_{old} is on L_{new}. We write $L_{old} \subset L_{new}$. If there is no tree L_{mid} with $L_{old} \subset L_{mid} \subset L_{new}$, then L_{old} is treated as a metablock of L_{new}. The root of L_{old} is treated as the root of this metablock; the root of L_{old} is a heavy node on L_{new}. All other nodes on L_{old} are light nodes of this metablock, unless they are not even on L_{new}. This matters when defining the potentials for nodes on L_{new}. Suppose that L_{old} is contained in L_{new}; then, apart perhaps for its root, any node on L_{old} that is also on L_{new} must be a light node on L_{new}; in fact, only the root and nodes on an extreme path of L_{old} can be on L_{new}, but it need not be the case that even all these nodes are on L_{new}.

In addition, each guard of a lazy tree may be the root of another lazy tree, rather than being the root of a block. More precisely, suppose that g is the right (resp. left) guard for lazy tree L on creation of L, and L_{old} is the newest lazy tree rooted at g at this time (L_{old} is older than L). Then the root of L_{old} remains the right (resp. left) guard for L until one or both of L and L_{old} are split. Note that the root of L_{old} may at some point become the root of a newer lazy tree, L_{new}; however, the root of L_{new} does not take over the guard role for L.

In order to carry over the previous analysis of an extreme path traversal, we require the following properties concerning lazy trees to apply.

4.3.4 The Invariants Defined for the Split Operation

We note that the split operation has been defined so as to maintain Invariant 13. It remains to consider the effect of the split on Invariant 12; but the only effect of a split is to reduce the size and hence weight of a lazy tree; so this Invariant also continues to hold.

4.3.5 Summary and Generalization

We have shown (see Lemma 8 and Equation 5):

Lemma 10 *The cost of a global access is at most* $e \cdot (3 \log n + 1)gp + 5 \log n \cdot gp$ *units.*

In a later paper we will again use lazy trees which will have heavy and light nodes. The heavy nodes will have the same lazy and reserve potentials as here. It is useful to summarize and slightly generalize the results of this section.

As already noted, the nodes of the lazy trees in the later paper will still have small, large and lazy debits which obey the present invariants, namely Invariants 1–4 and 6–7. Other additional debits satisfying Invariant 11 may be present. We need to show how to maintain this invariant following a lazy tree split. It is readily seen that the method for restoring the invariants concerning lazy debits suffice to restore this invariant also, so it suffices to replace hd by md in Equations 20 and 21, yielding:

$$c' \geq \max\{4md, ld + 3md\} = 4md \tag{22}$$

$$b \geq 3md + 2ld + 2sd + gp \tag{23}$$

The debits may be restricted beyond the invariants of this paper. So long as these new constraints satisfy the following property:

Property 3 *The restoration of any further invariant concerning the debits requires the removal of at most* α *additional debits from each lazy tree created by the split,* α *a constant.*

By Property 2 there are at least as many promoted nodes as there are lazy trees created by the split. So the sets of α additional debits can be charged α to each promoted node. modifying Equation 23 as follows

$$b \geq 3md + 2ld + 2sd + gp + \alpha \cdot md \tag{24}$$

suffices to ensure all the invariants can be restored.

We have shown:

Lemma 11 *If a lazy tree comprises light and heavy nodes where the heavy nodes carry lazy potentials as specified in Invariant 12, then each lazy tree resulting from a split satisfies:*

(i) Invariant 12.

(ii) If the nodes on its left path are new to a left path, then they are all light nodes in the same block.

(iii) Assuming the debits obey Invariants 1–4 and 6–7, and 11, and assuming Equations 22 and 24 hold, then these Invariants can be restored following a split.

Clearly, we also need:

Property 4 *The potential must be partitionable following a split; i.e., each lazy tree created by the split must be provided with an appropriate potential (eg. a lazy complete tree potential).*

promoted root of x's new lazy tree. In this case, the removal of x's lazy debit is charged to z.

Note that a promoted node may be charged for the removal of three lazy debits (as node y) or one lazy debit (as node z) but not both.

Case 3. x is the right child of its parent z, which was on the left path of the lazy tree prior to the split. Then z pays for the removal of the lazy debit from x. So z may have to pay for the removal of up to two lazy debits (the first such debit was removed in Case 2).

Each light node in a block that becomes either ordinary or newly leftmost is charged for the removal of at most four lazy debits (from itself and the nodes for which it restored Invariant 7) or one small or large debit and three lazy debits (removed from the same nodes). This is charged to the reduction of at least c' in the node's potential. So it suffices to have:

$$c' \geq \max\{4hd, ld + 3hd\} = 4hd \tag{20}$$

Each global node is charged for the removal of at most three lazy debits (from the nodes for which it restored Invariant 7), two small debits and two large debits; it may also have to pay a segment charge. So it suffices to have:

$$b \geq 3hd + 2ld + 2sd + gp \tag{21}$$

Lemma 9 *Following a split, Invariants 1–4 and 6–7 concerning the debits can be restored provided Equations 20 and 21 hold.*

4.3.3 Recreating the lazy complete tree potential

We continue by showing how to reestablish Invariants 8–10. To restore Invariant 8, we proceed essentially as in the original creation of active layers. Let h be the maximum creation_height of any node in the normal form of the new lazy tree rooted at u, apart from its root (the creation_heights are those defined with respect to the original tree; they are not redefined with respect to the new lazy tree). Suppose there is no l-active node for some $l \leq h$. Then, in the corresponding normal tree, the lowest node v on the right path whose span includes l becomes l-active (these new active layers are then translated back into the lazy tree at hand). Below, we show that Invariants 8–10 hold once more (incidentally, this implicitly shows that the rule for creating new l-active nodes is well defined).

Clearly Invariant 10 still holds for if $d_l(v)$ increases $id_l(v)$ is reduced by an equal amount. Next we consider Invariant 9. In each new lazy tree the black status of the nodes is unchanged; also, by Property 1, the heavy nodes on the new left paths were also previously on left paths. So Invariant 9 still holds.

Next, we show Invariant 8 is reestablished by the creation of new active layers. First, we consider the situation prior to the creation of new active layers. Clearly, Invariant 8 (iii) and (v) still hold; (iv) holds likewise, since the symmetric order of the nodes in the normal form of each new lazy tree is unchanged. We note that each new lazy tree has a span of active layers, possibly empty, of the form $(i, h]$, $0 \leq i \leq h$, where h is the largest creation_height (as provided initially) of any node in the normal form of the new lazy tree, apart from the root. By Invariant 8 (v), (iii) and (iv), the right path in each new normal lazy tree, from top to bottom, contains a sequence, possibly empty, of active nodes with decreasing creation_heights. It is now readily seen that the rule for creating new active layers restores Invariant 8(i)–(ii).

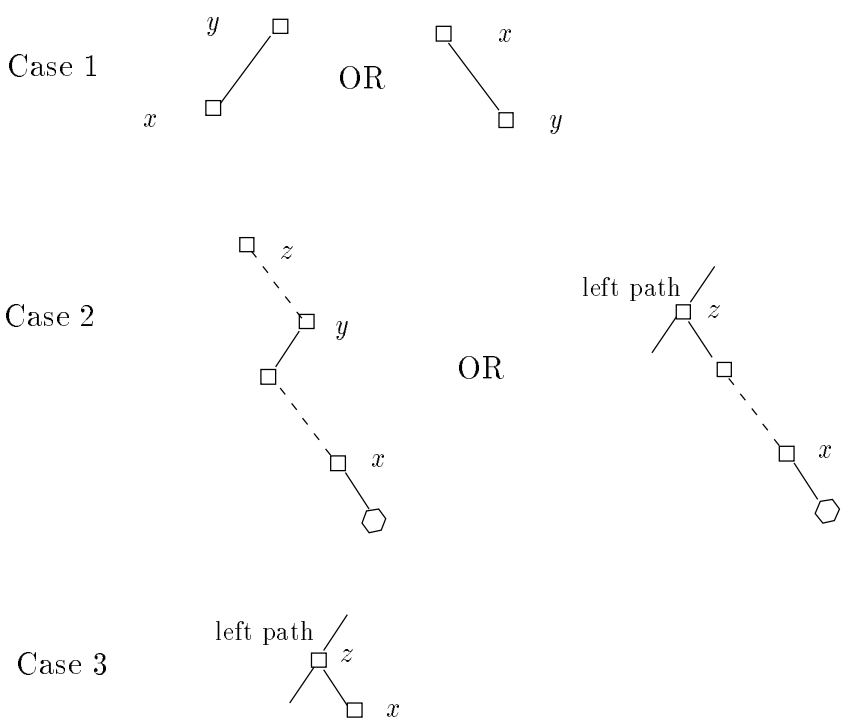


Figure 22: Invariant 7

Property 2 *The number of promoted nodes from each old lazy tree L is at least the number of lazy trees into which L is split.*

4.3.2 Debits and their Invariants

We need to consider the effect of the split and (from the beginning of Section 4.3) of the updates to the guards on Invariants 1–4 and 6–7. We show how to reestablish these invariants by removing debits on the extreme paths of the new lazy trees and on the left and right paths of blocks whose roots cease to be part of the lazy tree.

Invariant 1. The small or large debit, if any, is removed from each promoted node. This is charged to the promoted node. Also, each light node that ceases to be an extreme path node, and hence also ceases to be on a lazy tree, pays for the removal of its small or large debit, if any.

Invariant 2. For each promoted node, the large debits, if any, are removed from its parent and child (if any) on the extreme path. This is charged to the promoted node. (Note that the reestablishment of Invariant 1 has already removed the large debit, if any, from itself.)

Invariant 3. The small debit, if any, is removed from each extreme path node. This is charged to the promoted node.

Invariant 4. For each promoted node u, the small debit, if any, is removed from the corresponding node w, if present. This is charged to the promoted node.

Note that reestablishing Invariants 1–4 requires each promoted node to pay for the removal of at most two large debits and two small debits, or one large and three small debits, or five small debits. The maximum charge is achieved with two large and two small debits.

Invariant 6. The lazy debit, if any, is removed from each light node whose block is no longer part of a lazy tree. This includes those light nodes whose block roots become guards for a new lazy tree. We call such blocks *ordinary blocks*. This is charged to the node itself. For each block that becomes the leftmost block in a new lazy tree we reduce the potential of each light node from $2c'$ to c'. This pays for the removal of lazy debits from these nodes. Any other light node on a new left path must have been on the old left path and so already satisfied the invariant.

Invariant 7. Any node violating Invariant 7 must be on a new right path. There are three ways for a light node x on a new right path to still violate Invariant 7.

Case 1. x is adjacent to node y in block, B_y, where B_y either has become the leftmost block in a new lazy tree or has become an ordinary block. The removal of x's lazy debit is charged to node y.

Case 2. See Figure 22. x's right child is in the same lazy tree but is heavy. Let z be the root of x's block (z is a heavy node also). Let y be the nearest right ancestor of x. If y is between x and z, y must be in a block, B_y, where B_y either has become the leftmost block in a new lazy tree or has become an ordinary block (this follows from Property 1). In addition, for each such node y, there is at most one node x of Case 2, namely the first node, on the right path descending from y's left child, to have a heavy child. The removal of x's lazy debit is charged to node y.

If z had not been on the left extreme path of the lazy tree prior to the split then there is such a node y, for if not, prior to the split, x would have been on the right path descending from z, and by Invariant 7, would not be carrying a lazy debit.

So if there was no such node y, then z had been on the left path of the lazy tree prior to the split and x was on the right path descending from z in the splay tree. Also z must be the

(i) $x \neq v$. Temporarily, x becomes the right guard for the new lazy tree rooted at v. In the imaginary tree, x's right subtree contains no nodes from its large lazy block tree. So by Invariant 12(ii), the lazy potential for x plus its reserve potential is at least its global potential; thus the promotion can be paid for at no extra charge. Now, the right guard restoration process is applied to x. This has zero amortized cost.
(ii) $x = v$. Let w be v's left child in its new lazy block tree (if v does not have such a child, then the new lazy tree rooted at v comprises only one node and can be treated as an ordinary block henceforth). w is promoted, becoming the root of a new lazy tree while v becomes its right guard. Next, if w's right subtree in the large lazy block tree is non-empty, the right guard restoration process is applied to v. w's promotion is paid for as follows. w adds its reserve potential to its lazy potential; w's modified lazy potential is at least gp times the global rank of v's right child in the imaginary tree (for by Invariant 12(ii), this modified lazy potential is at least $gp \cdot lazy_rank(r_n(v))$ and $r_n(v)$ must be to the right of the accessed item; so $lazy_weight(r_n(v))$ includes the weight of all of v's right subtree in the imaginary tree; now recall that $lazy_rank(r_n(v)) \geq \log lazy_weight(r_n(v)))$. Hence $gp \cdot jump(v)$ suffices to pay for this promotion.

Overall, Case 2 occasions a charge of $2gp \cdot jump(v) + gp \cdot jump(v')$. Note that v is on the left split path and v' is on the right split path.
Case 2.1. $\tilde{v}$ is a light node on the left extreme path; it is accessed from its right child, which is not in the lazy tree.
Let v' be the nearest heavy ancestor of v in the lazy tree. v' is treated as in Case 2. Let v be the nearest heavy descendant of $\tilde{v}$ on the left extreme path. v is promoted. v's promotion costs at most $gp \cdot jump(\tilde{v})$ (strictly speaking $jump(\tilde{v})$ was not defined; it is the jump in g_rank on accessing node $\tilde{v}$), which is at least $g_rank(v)$. As in Case 2, we need to provide the new lazy tree rooted at v with a right guard. This is handled as in Case 2.

Hence, overall, Case 2.1 occasions a charge of $2gp \cdot jump(\tilde{v}) + gp \cdot jump(v')$. Note that v is on the left split path and v' is on the right split path.
Case 3. The inserted item is to the right of the lazy tree root, r.
Then we need to provide the new lazy tree rooted at r with a new right guard. The method followed is identical to that used in Case 2 for providing the new lazy tree rooted at v with a right guard. So Case 3 occasions a charge of $gp \cdot jump(r)$. Note that r is on the left split path. These promotions are performed even if only the right guard, rg, in the lazy tree is separated from the rest of the lazy tree (through being accessed from its left child). This applies even if rg is the root of the large lazy tree.

We have shown:

Lemma 8 *The promotions in a global insertion cost at most $4gp \log n$ units, where for each split lazy tree there is a charge of $2gp$ times the increase in global rank along the right split path and of $2gp$ times the increase in global rank along the left split path.*

The following properties of the new lazy trees are helpful; they are readily confirmed by inspection of the split procedure.

Property 1 *The left path of each lazy tree created by the split has one of the following two forms;*

(i) It is a subpath (possibly complete) of the left path of the lazy tree before the split.

(ii) It comprises light nodes from just one block.

henceforth. We call the promotions described in this paragraph the *right offset promotions* for v.

So Case 1 occasions charges of $2gp \cdot jump(v)$. Recall that v is a node on the right split path.

Case 1.1. v' is a light node which is not on the left extreme path. In addition, v' is accessed from its left child, which is not in the lazy tree.

The right offset promotions for v' are performed (where w_1 is now defined to be the nearest heavy descendant of v'). Let v be the root of the block containing v'. As we will see, v is promoted in Case 2 or Case 3. Thus the cost of Case 1.1 is at most $gp \cdot jump(v)$.

Case 2. v is a heavy node on the left extreme path and it is accessed from its right child, u.

Then v is promoted. v becomes the root of a new lazy tree. v's promotion costs at most $gp \cdot jump(v)$ (apply Corollary 2(i)).

We need to provide a new left guard for the remaining portion of the old lazy tree (See Figure 21.) Let v' be v's parent in the lazy block tree. We perform the right offset promotions

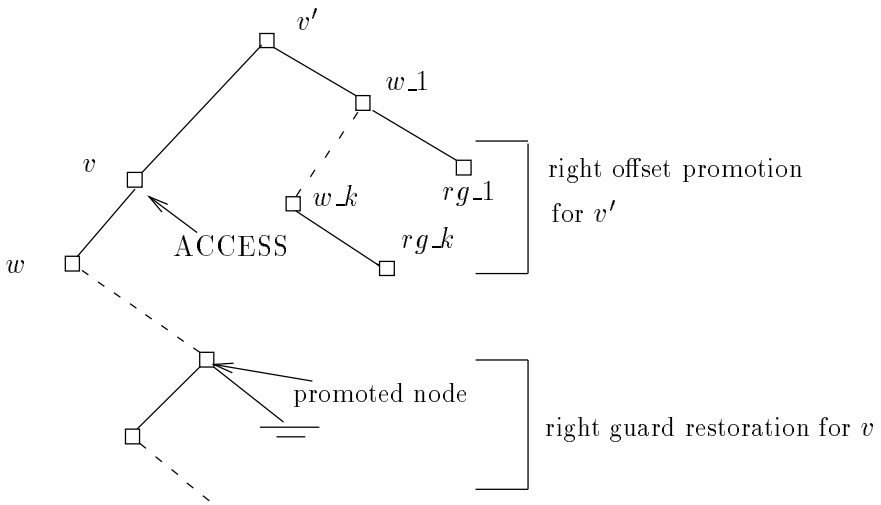


Figure 21: Promotions in Case 2

for v'. In addition, if v' is not the root of the lazy tree, v' is promoted. v' becomes the root of an ordinary block. If v' is not the root of the lazy tree, rg_1 becomes the left guard for the remainder of the old lazy tree, while if v' was the root, then rg_1 did not need promoting as it was already the right guard of the lazy tree. In the imaginary tree, v''s lazy weight is at least its normal weight; so its promotion comes for free. As in Case 1, $gp \cdot jump(v')$ suffices to pay for the right offset promotions.

The promotion of v' and the associated right offset promotions are also performed if just the left guard, lg, is separated from the remainder of the lazy tree (through being accessed from its right child), and then v' is defined to be the leftmost node in the lazy block tree (note v' will be traversed in the access). This applies even if lg is the root of the large lazy tree.

We need to provide the new lazy tree rooted at v with a right guard. So let x be the rightmost heavy node in v's lazy tree which is a right descendant of v in the imaginary tree (possibly $x = v$). There are two possibilities to consider.

(ii) Alternatively, suppose that v is not on the left path of L and is on the right split path. Let v' be a heavy node which is a right descendant of v, not necessarily proper. Then $lazy_rank(v') + jump(v) \geq g_rank(v)$.

Proof. We prove (i); the proof of (ii) is very similar. Let w be v's right child in L (if w is not present, define $g_rank(w) = 0$). Also, let w' be v's right child in the imaginary tree (again, if w' is not present, define $g_rank(w') = 0$). Then, by Corollary 1(i), $lazy_rank(v) \geq g_rank(w)$. Now w' must be a descendant of w, so $g_rank(w) \geq g_rank(w')$. Finally, $jump(v) = g_rank(v) - g_rank(w')$. So $lazy_rank(v) + jump(v) \geq g_rank(w') + jump(v) = g_rank(v)$. •

We now discuss which nodes are promoted in a split and how this is paid for. There are a number of cases.

Case 1. v is a heavy node in the lazy block tree, which is not on the left extreme path. In addition, v is accessed from its left child, u.

Then v is promoted. v's promotion costs at most $gp \cdot jump(v)$ (apply Corollary 2(ii)).

Let w_1 be v's right child, if any, in the lazy block tree, and let $w_2, w_3, \cdots, w_k$ be the maximal left path descending from w_1 in the lazy block tree (see Figure 20). $w_1, w_2, \cdots, w_k$ are also

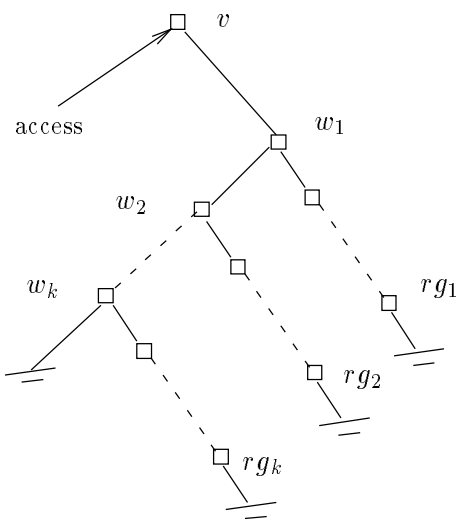


Figure 20: Right offset promotions for v

promoted. Between them, these promotions cost at most $gp \cdot jump(v)$. For, by Corollary 1(ii), $lazy_rank(w_i) \geq g_rank(w_{i+1})$, for $1 \leq i < k$. In addition, $lazy_rank(w_k) + jump(v) \geq g_rank(v)$ (by Corollary 2(ii)), and $g_rank(v) \geq g_rank(w_1)$. The nodes w_i become the roots of new lazy trees. Let rg_i be the rightmost node in the subtree of the old lazy block tree rooted at w_i, for $1 \leq i < k$. Temporarily, rg_i is made the right guard for w_i. rg_i is promoted; this is paid for by its reserve potential (for note that rg_i has an empty right subtree in the large lazy block tree and apply Invariant 12(ii)). Then the right guard restoration process is applied to rg_i, for $1 \leq i \leq k$. Also, rg_{i+1} becomes the left guard for the new lazy tree tree rooted at w_i, for $1 \leq i < k$; the tree rooted at w_k uses v as its left guard. If $rg_i = w_i$, for some i, the new lazy tree rooted at w_i comprises only one block; so it is treated as an ordinary block

We view a split as occurring in three phases. In Phase 1 no rotations are performed, but certain heavy nodes are marked as *promoted* (a node is promoted by increasing its lazy potential to its global potential). In fact the nodes are not promoted yet, but the splay will proceed as if they had been promoted. The effect of the promotions is to partition the original lazy tree into several new lazy trees. Phase 3 pays for these promotions. This ensures that in Phase 2, the actual splay, only extreme paths of lazy trees are traversed. However, there will be one difference in paying for Phase 2 as compared to the previous traversals. Any (apparently) promoted node, whose global rank drops during the splay does not use gp units of its global potential to pay for the associated segment, for it does not yet have its global potential; instead paying for the segment becomes a charge to be paid for by the promotion (a charge of at most gp units). It is called the *segment charge*; the segment charge is paid for directly by the promoted vertex. Phase 3 pays for the remaining costs of the promotions, at most $4gp \log n$ units.

Phase 3 uses the following *imaginary* tree. Consider performing all the zig-zag operations of the insertion but replacing each of the zig-zig operations by two single rotations performed in bottom to top order. This creates two paths, called *split paths*, descending from the inserted item, the root of the imaginary tree; one path, the *left split path*, to the left of the root, descends to the right, the other path, the *right split path*, to the right of the root, descends to the left (see Figure 19). The items on the split paths are exactly the items that will be

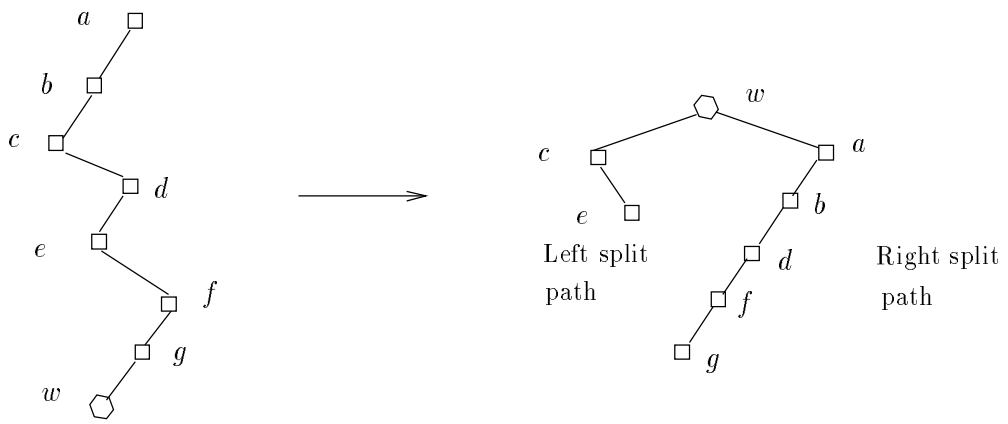


Figure 19: The split paths

traversed in the splay operation. We provide each global node on the split paths with an *imaginary* global rank, namely the global rank it has in the imaginary tree. The global ranks of the other global nodes are the same in the imaginary tree and the actual tree. Each of the split paths is traversed from bottom to top; for each global node at which there is a jump in imaginary global rank, denoted $jump(v)$, the following potential is provided: $2gp \cdot jump(v)$. The following corollary is helpful.

Corollary 2 *Let v be a heavy node in lazy tree L. Suppose that L is split.*

(i) Also, suppose that v is on the left path of L and is also on the left split path. Then $lazy_rank(v) + jump(v) \geq g_rank(v)$.

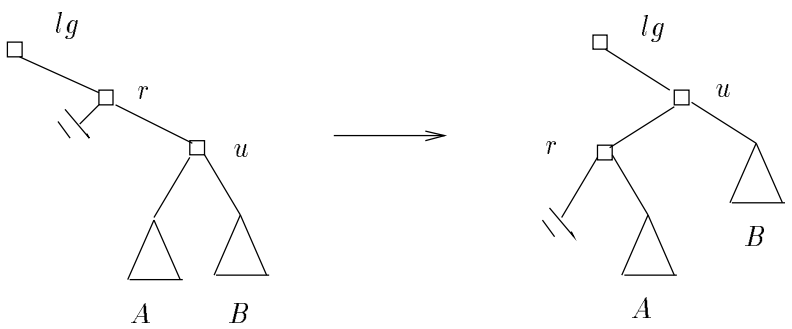


Figure 17: Guard restoration for a root and right child couple

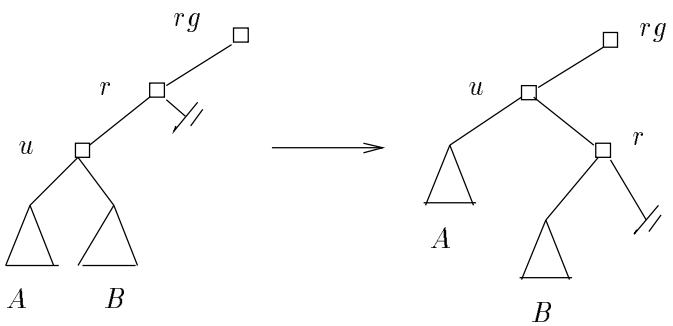


Figure 18: Guard restoration for a root and left child couple

rooted at r. This is called the *right guard restoration process* applied to rg.

Second, consider a rotation between lg and heavy node u on the left path. Let v be the parent of u in the large lazy block tree. Following the rotation, $lazy_rank(u) \geq g_rank(u)$ (for by Invariant 12(i), $lazy_rank(u) \geq \lfloor \log(wt(SL(v))) \rfloor$ — remember the lemma applies to the normal form of the lazy tree). In addition, following the rotation, u has an empty left subtree in the large lazy block tree (since lg had an empty right subtree before the rotation). So u is made the root of a new lazy tree, with left guard lg; temporarily, its right guard, rg_new, is the rightmost node in its right subtree in the large lazy block tree; rg_new acquires its global potential by adding its reserve potential to its lazy potential (by Invariant 12(ii), this suffices). See Figure 16. rg_new also becomes the left guard for the remainder of the lazy tree rooted

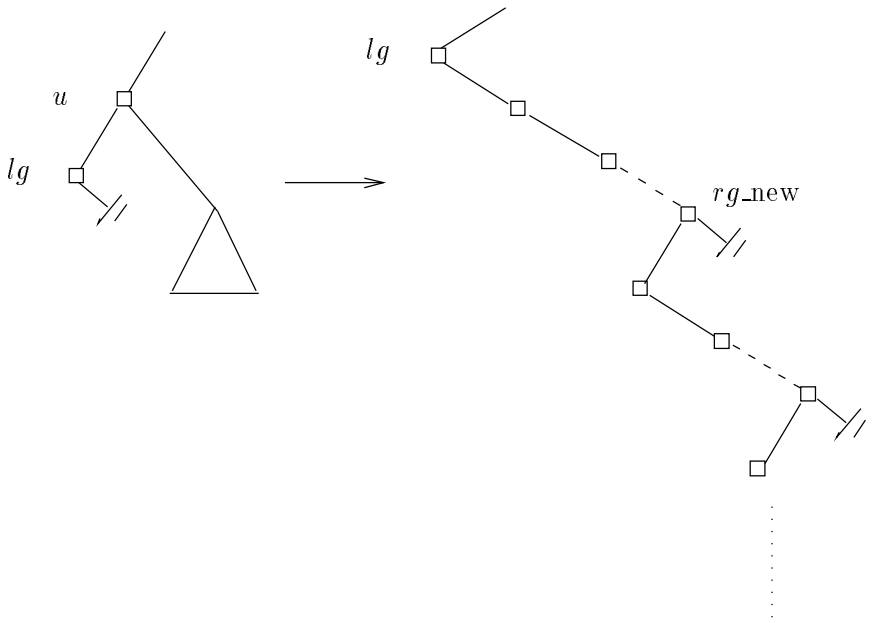


Figure 16: The left guard restoration process

at r. Then the right guard restoration process is applied to rg_{new}, to obtain the actual new right guard for the new lazy tree rooted at u. There is one special case: u has an empty right subtree; then u simply becomes the root of a normal block and r acquires u as its new left guard. This is called the *left guard restoration process* applied to u.

Third, consider case (ii) of Invariant 13; suppose there is a rotation between r and its right child, u, a heavy node (see Figure 17). Then, following the rotation, the left guard restoration process is applied to r, which is now the left child of u, the new root of the lazy tree. Fourth, consider case (iii) of Invariant 13; suppose there is a rotation between r and its left child, u, a heavy node (see Figure 18). Then, following the rotation, the right guard restoration process is applied to rg.

It remains to discuss how to maintain Invariants 1–10. We explain how to do this later in the section, when we show how to maintain these invariants following a split operation.

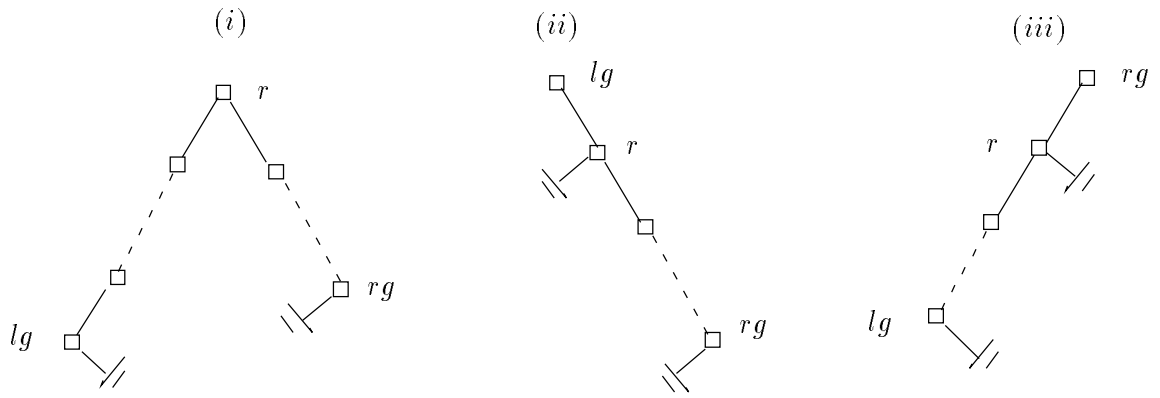


Figure 14: The large lazy block tree

(iii) *rg is the root of the large lazy block tree. Then r is the left child of rg, r has an empty right subtree, as does lg.*

Now, we show how to restore Invariant 13 following a traversal of an extreme path. Invariant 13 can cease to hold only if there is a couple containing a guard or the root of the lazy tree and a heavy node. So first consider a couple comprising a heavy node u on the right path and rg (see Figure 15). Following the rotation, let $rg_0 = rg$, and let rg_{i+1} be the rightmost

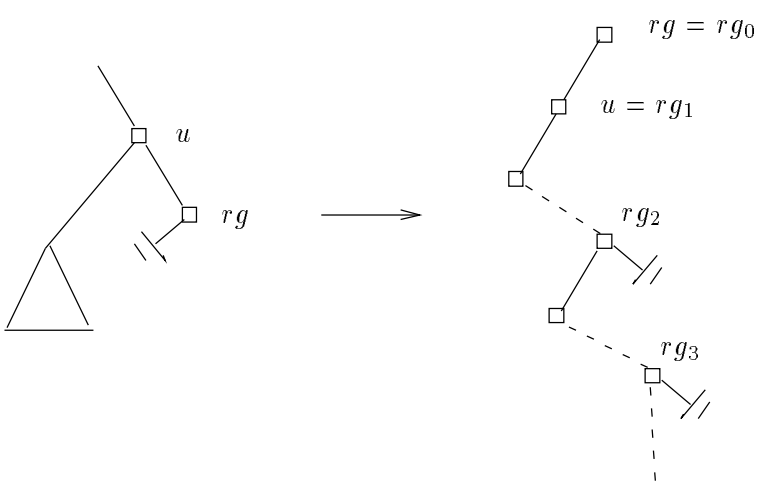


Figure 15: The right guard restoration process

node in the left subtree of rg_i in the large lazy block tree, if this subtree is not empty. This defines a non-empty sequence of nodes $rg_1, \cdots, rg_k$; all these nodes are promoted (i.e., they have their global potential restored) by adding their reserve potential to their lazy potential (this suffices by Invariant 12(ii)). rg_k becomes the new right guard for the remaining lazy tree

Invariant 12 *Let v be a heavy node in the normal form of the lazy block tree for lazy tree L. Then:*

(i) $lazy_rank(v) \geq \lfloor \log(wt(SL(v)) \rfloor$.

(ii) If, in the normal form of the large lazy block tree, v has no heavy node or guard in its right subtree, $lazy_rank(v) + 1/gp \cdot reserve(v) \geq g_rank(v)$. *While if v does have a heavy node or guard in its right subtree, then* $lazy_rank(v) + 1/gp \cdot reserve(v) \geq lazy_rank(r_n(v))$.

Invariant 12 can be seen to hold when the lazy tree is created by considering node v at the point at which it becomes lazy. Also, is is readily seen that this invariant continues to hold following traversals of the extreme paths (for they leave the lazy rank of nodes in the normal form of the lazy tree unchanged).

Corollary 1 *See Figure 13. Let v be a heavy node other than the root in lazy tree L. Let w*

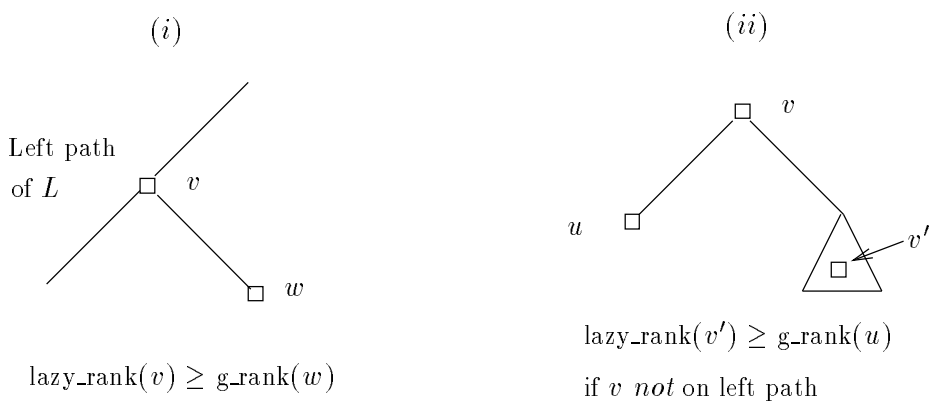


Figure 13: Heavy nodes

be v's right child and u be v's left child.

(i) Suppose that v is on the left path of L; then $lazy_rank(v) \geq g_rank(w)$.

(ii) Suppose that v is not on the left path of L. Let v' be a right descendant of v, not necessarily proper. If v' is a heavy node, $lazy_rank(v') \geq g_rank(u)$.

It is convenient to define v's *lazy weight* to be $wt(SL(v))$.

The analysis is simplified by, on occasion, modifying the guards of a lazy tree following a traversal of one of its extreme paths. Specifically, we maintain the following invariant.

Invariant 13 *See Figure 14. Consider a lazy tree L. Let r denote the root of the truncated lazy block tree, and let lg and rg denote, respectively, the left and right guards of L. The large lazy block tree has one of the following three forms.*

(i) r is the root, the left subtree of rg is empty, as is the right subtree of lg.

(ii) lg is the root of the large lazy block tree. Then r is the right child of lg, r has an empty left subtree, as does rg.

(i) v is not on the left path of the lazy tree.

(ii) If v is on the right path of the lazy tree then so are its parent and right child in the splay tree.

This invariant meets condition (i) of Remark 1; so long a condition (ii) is also met, the analysis of the traversal of extreme paths of the lazy trees is unaffected. But the presence of new debits, of value ed say, merely raises the cost of a couple by at most $2ed$. It is convenient to let

$$md = \max\{ed, hd\} \tag{15}$$

The new debits can be handled in the present analysis by modifying Equations 9, 10, 13 and 14 so as to pay for the removal of up to two debits of value ed from each couple being analyzed, yielding:

$$c' \geq s + 1 + 2md \tag{16}$$

$$hd \geq s + 1 + \max\{ld, 2sd, 2ed\} \tag{17}$$

$$\frac{d \cdot s'}{2} \geq \max\{(s + 1 + 2ed + 2hd), \frac{a}{d}\} \tag{18}$$

$$\frac{a}{2d} \geq s + 1 + 2ed + 2hd \tag{19}$$

4.3 Splitting the Lazy Tree

A split of the lazy tree occurs when the inserted item lies in value between the left guard and the right guard in the lazy tree. Since a split causes a zig-zag rotation within the lazy tree, and hence in the splay tree other that at the splay tree root, a split can occur only during a global insertion. For this section, we assume that each node belongs to at most one lazy tree, apart possibly, for the sharing of guards. In Section 4.5, we analyze the general case.

The split is analyzed in three parts. First, we show that if the lazy tree satisfies invariant 12, below, then by promoting appropriate nodes (i.e., restoring their global potentials) the lazy tree is split into several new lazy trees which all also satisfy Invariant 12. In addition, the promotions cost at most $4gp \cdot logn$ units of potential. Second, we show that the cost of removing debits so as to restore all the invariants concerning debits can be charged to the promoted nodes at no further cost. Third, we show how to create the lazy tree potential for each new lazy tree from the corresponding potential for the lazy tree prior to the split.

In order to permit more general accesses in the Part II paper, we show how to analyze a split when a local item in a block is accessed as well as when a global item is accessed. The goal, when a local item e is accessed, is to remove e's block from the lazy tree so that it becomes a normal block.

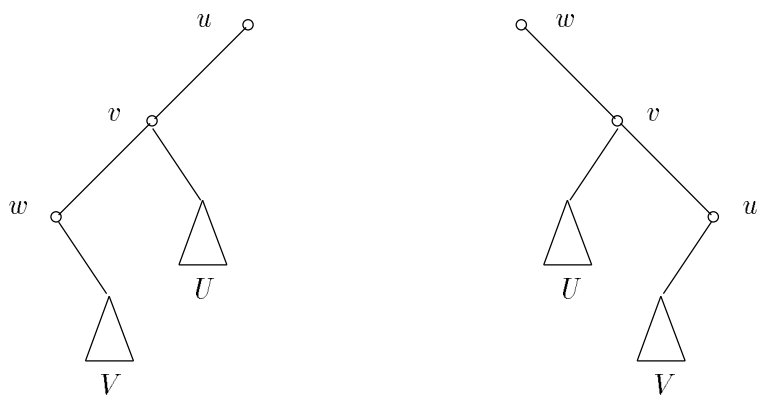
4.3.1 The Promotions

At this point it is helpful to mention a few properties of the lazy and reserve potentials. Consider the heavy nodes in a new lazy tree. Let SL be the set of global nodes contained strictly between the guards of a lazy tree. For each heavy node, u, in the lazy tree, define its right neighbor $r_n(u)$ to be the heavy node immediately to its right in the large lazy block tree, and define $SL(u)$ to comprise the subset of SL strictly to the left of u. Finally, define the lazy rank of heavy node v to be $\frac{1}{gp}$ times its lazy potential.

$$\frac{a}{2d} \geq s + 1 + ld + 2hd \tag{14}$$

We turn to the analysis of left path traversals.

Consider a couple, w, v, on the left path, with v the parent of w; let u be the parent of v (see Figure 12). If nodes u and v were on the right path, subtree U (resp. V) would be the left



Actual situation v, u rotated onto right path

Figure 12: Left path traversals

subtree of u (resp. v). We treat the rotation of couple w, v on the left path as if it were the rotation of couple u, v on the right path. We pay for the rotation between v and w using w's present $\frac{a}{2d}$ units of l-black potential, where w is l-active (note that the present potentials of v and w correspond respectively to those v and u would have if on the right path). Then w and v swap potentials; v and all the nodes in its right subtree reduce their black spans accordingly. Invariants 8–10 continue to hold, as can be seen by arguments similar to those used for the right path traversal.

Remark 3 A left path traversal itself does not cause the spending of any spares, for all the rotations on the left path involving two heavy nodes of the lazy tree (which are the only rotations to spend spares) are paid for by black potentials.

We can conclude (as asserted in the second paragraph of Section 3):

Lemma 7 *For each of the first $e - 1$ local insertions in a block's insertion, for each couple, s spares can be provided to the couple.*

In a later paper we will again use lazy trees generalized in various ways. However, the nodes of the lazy trees in the later paper will still have small, large and lazy debits which obey the present invariants, namely Invariants 1–7. Other additional debits may be present. They will satisfy the following invariant.

Invariant 11 *Let light node v carry an additional debit. Then*

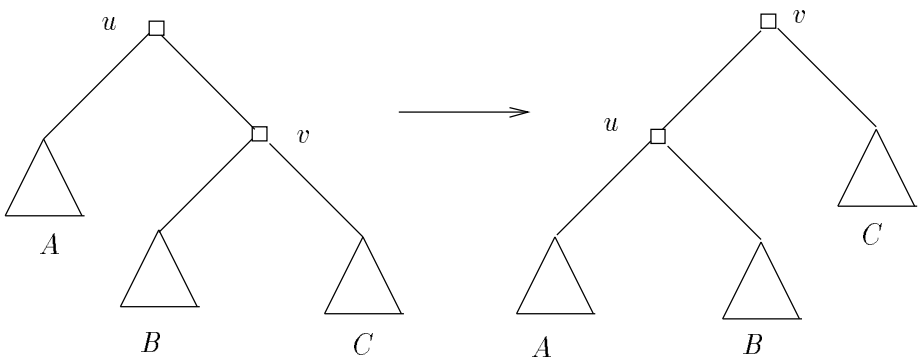
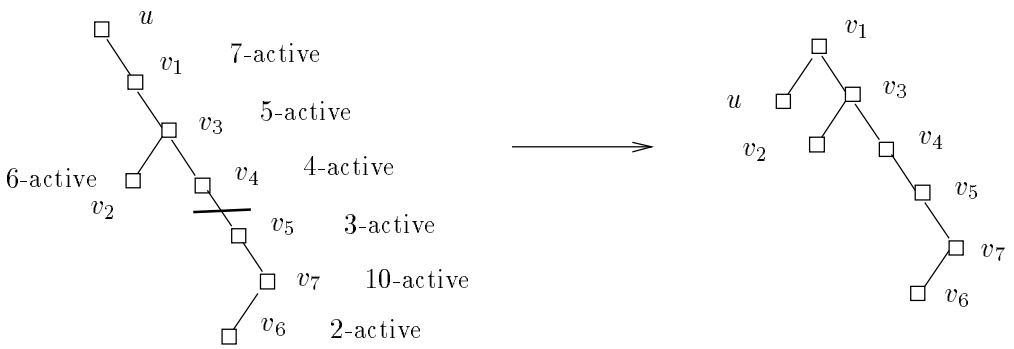


Figure 10: Rotations with black nodes

w' transfers a second portion of $\frac{a}{2d}$ units of potential. Note that by Invariant 8(iv) each active layer j' of w' satisfies $j' > j$ and so $d_{j'}(w') < j'$; thus the j'-potential of w' is reduced by $\frac{a}{d}$, owing to the creation of the black node, and w' can afford to transfer two sets of $\frac{a}{2d}$ units of j'-potential to the new black node (one set for itself and one set for node w). (See Figure 11 for an example.)



v_3 pays for the 5-black potential and the 2-black potential

Figure 11: Creating a black node

We summarize the above discussion. The rotation of a couple costs $coup = s + 1 + ld + 2hd$ units. Suppose the bottom node of the couple is l-active; then these *coup* units are paid for by either $\frac{a}{d}$ units of l-potential, or by $\frac{a}{2d}$ units of l-black potential, or by the $\frac{d \cdot s'}{2}$ l-spares. While if the l-active node was not involved in a rotation with another global node of the lazy tree, the $\frac{d \cdot s'}{2}$ l-spares may have to provide it with $\frac{a}{d}$ units of potential. So it suffices to have:

$$\frac{d \cdot s'}{2} \geq \max\{(s + 1 + ld + 2hd), \frac{a}{d}\} \tag{13}$$

(ii) If a node u is l-black all u's left ancestors, apart from the root, in the corresponding normal tree are l-black.

We define the following distances for each node v, active at layer l. Its *right path l-distance*, $d_l(v)$, is the number of proper left ancestors of v on the right path, excluding all l-black nodes. (Recall that a left ancestor of v is an ancestor of v to the left of v in symmetric order.) Its *layer l interior right path distance*, $id_l(v)$, is the number of proper left ancestors of v below the first l-black node and below the right path.

For each l-active node we maintain an l-potential, which satisfies the following invariant.

Invariant 10 *The l-potential at node v is at least*

$$a \cdot min\{\frac{d_l(v)}{d}, l\} + \frac{a}{d} \cdot id_l(v) \tag{12}$$

We note that initially a node has $a \cdot l$ units of potential for each layer l at which it could become active, so when a node first becomes l-active Invariant 10 holds.

In addition, for each l-black node we maintain an l-black potential of $\frac{a}{2d}$ units.

In turn, we analyze traversals of the right and left extreme paths.

Immediately prior to a traversal of the right path, each node on that path is given $s'/2$ spare rotations (whether or not the node is part of a couple comprising two heavy nodes of the same lazy tree). Each node's spares are subsequently provided by the rotation which involves that node. Suppose node v remains on the right path following the rotation. Let layer l be the largest layer at which v is presently active. Suppose node u, the other node in v's couple, is not l-black. If $d_l(v) > d \cdot l$, then the spares, called the *l-spares*, for the nodes at depths $(d(l-1), d \cdot l]$ on the right path, are used to pay for v's rotation. Otherwise, a/d units of v's l-potential are used to pay for this rotation. Note that Invariants 8–10 continue to hold following this rotation.

If node v is l-active, but is not on the right path, then its l-potential can increase, but only if $d_l(v) > d \cdot l$. In this case, the unit increase in potential is paid for by the l-spares as in the previous paragraph.

A rotation between an l-black node u and its child v, where v is l-active, is paid for by the $\frac{a}{2d}$ units of l-black potential at u; u reduces its black span accordingly, as do all the nodes in u's new right subtree (the subtree following the rotation). Again, it is clear Invariants 8, 9 hold following the rotation. To verify Invariant 10 we argue as follows. (See Figure 10.) First, for j-active node w, $j < l$, in v's old right subtree, $d_j(w)$ is unchanged or reduced, since u ceases to be an ancestor of w; in addition, $id_j(w)$ is unchanged. Second, for j-active node w, $j > l$, in v's old left subtree, as in the previous paragraph, if v's j-potential increases, it is paid for by the j-spares. Third, for j-active node w, $j > l$, in u's left subtree, $d_j(w)$ and $id_j(w)$ are unchanged since the locations of w's left ancestors are unchanged, as is their j-black status. Reference to Invariant 8 (iv) shows that these are the only possible cases.

A black node is created in a rotation with the root. v becomes the new root. u, the old root, acquires all of v's potentials; v acquires the root's global potential. If u now has minimum active layer l it acquires black span $[1, l-1]$. The new portion of its black span, $[1, k-1]$, for some $k \leq l$, is paid for as follows. Let w be the j-active node, for some $j < k$. If $d_j(w) \leq j$ before u's rotation, as $d_j(w)$ is reduced by at least one by u's rotation, $a/2d$ units of w's j-potential provide u's j-black potential. If $d_j(w) > j$, then let w' be the node at depth j.

touched node is provided with $\frac{s'}{2}$ spare units of potential, $s' \leq s$ (this is part of the node's portion of the s spares provided to its couple in the splay tree)[1]. The spare potential of the touched nodes, together with any changes to the potentials of the touched couples will pay for the rotations of the touched couples.

At any time, certain nodes, called *active* nodes, are the nodes that pay for a traversal of the extreme path. Some nodes may pay more than other nodes. This is captured by the notion of *active layers*. A node of creation_height h has an associated *span* of layers $[1, h]$; each active node v of creation_height h has an *active span* of *active layers*, $(i, h]$, $0 \leq i \leq h$; for each l, $i < l \leq h$, we say v is *l-active*. If the active span is non-empty we say the node is *active*.

Initially, only the nodes on the right path are active. An inactive node becomes active when it first reaches the right path. Once a node becomes active it remains active, whether or not it remains on the right path; also, the active span of a node can only grow.

Recall that a rotation of the splay operation involving a heavy node, v, and the root, r, of the lazy tree causes the two nodes to swap all their potentials. Also, r acquires v's creation_height and active span.

The following invariant states several properties of the active nodes. We prove the invariant later. In order to avoid special cases for the left path we state the invariant with respect to the normal form of the tree, defined as follows. It is obtained by performing the following series of single rotations: one by one, the left path nodes are moved to the right path; each such rotation, between node v and node r, the root of the lazy tree, makes v the root and places r on the right path. r acquires v's active span. The resulting tree is called the *normal tree*.

Invariant 8 *Let H be the maximum creation_height for the nodes, other than the root, present in the tree initially. Then, in the corresponding normal tree:*

(i) There is exactly one l-active node, $1 \leq l \leq H$.

(ii) Every node on the right path is active (this does not include the root).

(iii) Apart from the root, the ancestors of an active node are all active.

(iv) Let v be an l-active node. Let w be a j-active node. If $j < l$ then w is to the right of v in symmetric order, while if $j > l$ then w is to the left of v in symmetric order.

(v) Let inactive node v have creation_height l. Then its parent has creation_height greater than l.

When a lazy tree is created the active spans for the nodes in the corresponding lazy block tree are initialized as follows. Let u be a node on the right path, of creation_height h; suppose it has a right child v of creation_height i (if there is no such node v let $i = 0$). Then u is given active span $[i + 1, h]$. Clearly, the new tree obeys Invariant 8.

The layers of a node are further categorized as *black* or *white*; an active node, with active span $[j, h]$, can be black with respect to each of the layers $[1, j - 1]$. In general, an active node v, with active span $[j, h]$, is black with respect to all the layers in some range $[i, j - 1]$, $i \geq 1$, called its *black span*; we say v is *l-black*, for $i \leq l \leq j - 1$. If the black span is non-empty we say the node is *black*. Nodes are initially white at all layers. A node becomes black as a consequence of a rotation with the root. The following invariant applies to black nodes.

Invariant 9 *(i) All the nodes on the left path of a tree are fully black, i.e., a node with active span $[j, h]$ has black span $[1, j - 1]$.*

[1] In this paper we can take $s' = s$; the more general form will prove useful in a later paper.

We need to mention one detail about couples containing the root, u, of a lazy tree and another node, v in the same lazy tree. Following the rotation, v becomes the root of the lazy tree; v and u interchange roles and potentials (so if v had been light, resp. heavy, u becomes light, resp. heavy).

In Section 4.3 we will explain how a lazy tree is split.

4.1 Analysis of Couples Including One Light Node

Consider a couple comprising nodes u and v, where u is the parent of v, both u and v are on an extreme path of their lazy tree and one, at least, of u and v is a light node.

Case 1. u and v are both light. u ceases to be on the skeleton. u's c' potential pays for the rotation, s spares, and the removal of all debits on u and v. So it suffices to have:

$$c' \geq s + 1 + 2hd \tag{9}$$

Case 2. u is a light node and v is a heavy node. By Invariants 6 and 7, u and v do not have lazy debits. If u leaves the skeleton, the operation is paid for by u's c' potential, as in Case 1; Equation 9 suffices. Otherwise, u is given a lazy debit; this then pays for the operation. The cost of the operation comprises the rotation, s spares, and the removal of small or large debits, if any, from u and v. So it suffices to have:

$$hd \geq s + 1 + \max\{ld, 2sd\} = s + 1 + ld \tag{10}$$

Case 3. v is a light node, u is the root of v's block (but is not the lazy block tree root). By Invariants 6 and 7, u and v do not have lazy debits. v becomes the block root. As in Case 2, if u leaves the skeleton, the operation is paid for by u's c' potential. Otherwise, u is given a lazy debit, which pays for the operation. The cost of the operation comprises the rotation, s spares, and the removal of small or large debits, if any, from u and v. Here too Equations 9 and 10 suffice.

4.2 Analysis of Couples Comprising Two Heavy Nodes

The potential associated with the lazy block tree is called the *lazy complete tree* potential.

Consider a couple comprising nodes u and v, where u is the parent of v and both u and v are heavy nodes on an extreme path of the lazy tree. We need to pay for the rotation, for s spares, and for the removal of small or large debits from u and v, if any. In addition, we may need to reestablish Invariant 7 for node u; this may require the removal of up to two lazy debits, which will also be paid for by the rotation. So the cost of this rotation is at most

$$s + 1 + ld + 2hd \tag{11}$$

For the remainder of Section 4.2 we focus on the lazy block tree. Hence when we refer to a node we mean a node in the lazy block tree; likewise a reference to a tree refers to the lazy block tree. The *depth* of a node in the tree is its distance from the root. The height of a node at the time of the creation of the lazy tree is called its *creation_height*; the creation_height does not change subsequently.

Consider how an extreme path traversal appears to the lazy block tree. A top contiguous portion of the nodes on this path are all *touched* (these are the nodes contained in couples of the splay tree). Some (arbitrary) subset of disjoint pairs of touched nodes form couples. Each

units, $hd \geq ld$, a constant to be defined later. We now state two invariants concerning the lazy debits.

Invariant 6 *Let L be a lazy tree. Suppose node v in L has a lazy debit. Then v is a light node of L. Also, v is not on the left path of L. Finally, v does not carry a small or large debit.*

Invariant 7 *Let L be a lazy tree. Let u be a local node of L. Suppose u has a lazy debit. Let v be the root of u's block.*

(i) Suppose that v is not on the left path of L. Then if u is on the right path descending from v in the splay tree, both the parent and child of u on this path are local nodes in u's block.

(ii) If u is on the right path of L then its parent and child in the splay tree are local nodes in u's block; this holds regardless of whether u is on the right path descending from v in the splay tree.

Next, we show how to incorporate lazy trees into the analysis of global insertions. A global insertion can traverse a lazy tree in one of three ways:

(a) Traverse the right extreme path of the lazy tree (or rather a topmost portion of it).

(b) Traverse the left extreme path of the lazy tree (or rather a topmost portion of it).

(c) Traverse the interior of the lazy tree and thereby split the lazy tree.

Actually, it is convenient to classify a traversal of type (a) which is to the left of the right guard to be a split (a type (c) traversal); likewise a traversal of type (b) to the right of the left guard is defined to be a split. As we will see later, local insertions only involve traversals of type (a) or (b).

A traversal of Type (c) will be paid for in two phases. First, in a preprocessing phase, the current lazy tree is partitioned into several lazy trees and/or ordinary blocks, so as to ensure that the actual splay (the second phase) comprises only traversals of Types (a) and (b). (In fact, as we will see, we need a third phase in order to pay for some of the partitioning performed in the first phase.)

We start by considering the interactions between the lazy tree and the remainder of the splay tree (which may include other lazy trees). We treat the lazy tree, as delimited by its extreme paths, as a block. We note that Invariants 6 and 7 ensure that a node on an extreme path, carrying a lazy debit, satisfies condition (i) of Remark 1. In addition, we note that on creation, the nodes of the lazy tree have no debits; so Invariants 1–4 all hold at this point.

Next, we need to show that condition (ii) of Remark 1 is satisfied. That is we have to show that couples involving two nodes on an extreme path of the lazy tree, which will be marked, can pay for the removal of their debits and $s + 1$ rotations. There are two classes of rotations: those involving at least one light node of the lazy tree and those involving two heavy nodes; they are treated in Sections 4.1 and 4.2, respectively.

Segments are defined and paid for essentially as before. The one difference occurs if either node of the leading couple of the segment carries a lazy debit; then the couple pays for its own rotation and the removal of its debits as part of the lazy tree, as we see later; but this can only reduce the total remaining cost of the segment and so the bounds of the previous analysis are still valid.

block. The root of the lazy tree corresponds to the block root, and the other nodes of the lazy tree, called *local* nodes of the lazy tree, correspond to the local nodes of the block. The lazy tree local nodes may carry small and large debits according to the invariants specified for blocks; a further *lazy* debit may be carried as specified later.

The right guard of each new lazy tree has its global potential restored (this is paid for by adding its reserve potential to its lazy potential, which suffices, as is stated in Invariant 12, in Section 4.3). In addition, Invariants 3 and 4 need to be restored for the right guard; this requires the removal of at most two small debits. These are paid for by the potential $a+b$ (see the following paragraph) associated with the right guard prior to the restoration of its global potential. So it suffices to have

$$2sd \leq a + b \tag{6}$$

Thus both the left and right guards of each lazy tree have their global potentials.

Next, we provide additional potentials to the nodes in the lazy tree (in addition to the potentials these nodes already carry). All light nodes on the lazy tree carry a potential of $2c'$ units, except for the light nodes from the leftmost block in the lazy tree, which carry a potential of only c' units; c' is a constant to be specified later. In addition, we give the following potential to the heavy nodes. Let v be a heavy node; suppose it had height h in the lazy block tree. Then v receives potential $a\sum_{i=1}^{h} i + b$, where a and b are constants to be specified later. This potential is provided by redistributing the potential q given to each lazy node in the initial lazy trees. Note that each initial lazy tree has the following form: Consider the right path descending from its root; the left subtree of each node on this path, excluding the root, is a complete binary tree; in top to bottom order, these trees have strictly decreasing height. The q potentials are redistributed as follows. First, a potential of $a\sum_{i=1}^{k} i + b$ is given to each height k node in the initial lazy tree, other than the lazy tree root. The following argument shows that it suffices to provide each node with a potential of $8a + b$. Each node of height k in a complete binary tree receives a potential of $\frac{3}{2}ak(k+1) + b$; it provides this by passing a charge of $\frac{3}{2}a[(k-1)k + 4(k-1)]$ to each of its children; if it has a parent, in turn, it receives a charge of $\frac{3}{2}a[k(k+1) + 4k]$ from the parent; adding its own initial potential of $12a + b$ provides exactly the required final potential (note that a leaf will pass a charge of 0 to its non-existent children). The nodes on a right path acquire their potential from their left child. A node of height $k+1$ receives potential $k(k+1)a$ from its right child, which together with its own initial potential of $12a + b$ suffices to provide the required $[\frac{1}{2}(k+1)(k+2)]a + b$ potential; note that the right child still has potential $[\frac{1}{2}k(k+1)]a + b$, as required. So it suffices to have

$$12a + b \leq q \tag{7}$$

But the potentials in the initial lazy tree upper bound the potentials desired in the (truncated) lazy block tree. Finally, to provide the light nodes with their potential, it suffices to have

$$2c' \leq a + b \tag{8}$$

The left (resp. right) extreme path of (truncated) lazy tree, L, is the path from the root of L to the leftmost (resp. rightmost) node in L; it is convenient to exclude the lazy tree root from the extreme paths. We will call these the left and right paths of L, for short.

Nodes on the extreme paths of the lazy tree may have a small or large debit; these debits satisfy Invariants 1–4, where lazy tree L replaces block B in the statement of the invariants. Also, each light node in the lazy tree may have a *lazy* debit, which is *huge* and has value hd

4 The Analysis of Lazy Trees

Most of the analysis focuses on a subtree of the initial lazy tree, called the *truncated lazy tree* or the *lazy tree*, for short. It is defined as follows. Consider a new initial lazy tree, L, created by a sequence of local insertions. Consider the set of heavy nodes in L plus the root of L; the tree they induce is called the *initial lazy block tree*. We remove the rightmost node from the initial block tree; this defines the (truncated) lazy block tree for the (truncated) lazy tree. This rightmost node is called the *right guard* for the lazy tree (see Figure 9). The *left guard*

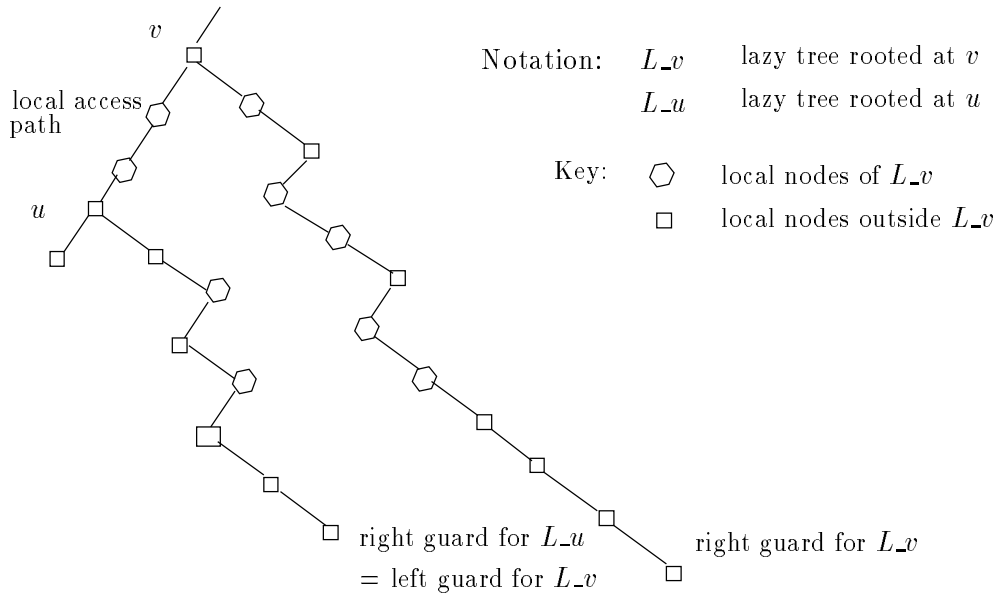


Figure 9: Left and right guards

is a global node, defined later, to the left of the nodes of the lazy block tree. The root of the (truncated) lazy block tree is called the root of the (truncated) lazy tree. We define the tree induced by the nodes of the (truncated) lazy block tree plus the left and right guards to form the *large lazy block tree*. Now we define the (truncated) lazy tree as follows.

We define the *left guard*, w, of the lazy tree as follows. Let L be a new lazy tree with root u. Let v be the the first proper global descendant of u on the local access path, if any. Suppose v exists; if v is the root of another new lazy tree, let w be the right guard in this lazy tree, while if v is not the root of a new lazy tree, then let $v = w$. Otherwise, let w be the root of the splay tree. In general, a node may be a right guard in one lazy tree and a left guard in a second lazy tree.

The truncated lazy tree comprises the nodes of the (truncated) block tree together with the following light nodes. For each node v in the (truncated) block tree we add the following nodes from v's block. Let w be any descendant of v in the large lazy block tree. Those nodes in v's block on the path from v to w in the splay tree are added to the (truncated) lazy tree. The light nodes added to the lazy tree form its *skeleton*.

Intuitively, a truncated lazy tree is a megablock comprising several of the blocks at hand. In its interactions with the remainder of the splay tree it will behave in the same way as a

u removed from the path may have too small a potential, which is called its *lazy potential*. Again, the cost of this rotation is $s + 1$ units; however, Invariants 3 and 4 may no longer be true for the block B whose root now has a lazy potential. All these rotations, as noted in the previous paragraph, are paid for by the potential at hand.

Lemma 6 *A true local insertion has amortized cost $c + s + 1$ units.*

Proof. As discussed in the paragraph before the lemma, all couples comprising two nodes on the local access path are paid for by the potential associated with the nodes of the couple. So the only rotation to be paid for is the final rotation of the insertion; it costs $s + 1$ units. In addition, the item displaced from root of the splay tree is given c units of potential for it has become an additional node on an extreme path of its block. •

Remark 2 We will need to generalize this analysis to take account of the presence of lazy trees on the local access path. The previous analysis will continue to apply if the following conditions hold.

(i) Each node on the local access path has potential $(2^{e-1} - 1)s$.

(ii) Each node on the local access path does not carry any debit.

(iii) The rotation of each couple on the local access path costs at most $s + 1$ units.

(i) and (ii) will be achieved as here. (iii) will be discussed later.

Every node removed from the access path in a true local insertion and remaining on an extreme path of its block is called a *lazy node*, whether or not it has a lazy potential. Those lazy nodes with a lazy potential are called *heavy* nodes; the other lazy nodes are called *light* nodes. When the insertion of the current block is completed we form *lazy trees*. Each global node v remaining on the local access path becomes the root of a new lazy tree. Its left subtree is empty; its right subtree comprises those lazy nodes created during the insertion of the current block that are in v's right subtree in the splay tree. It is straightforward to see that each new lazy node is contained in a new lazy tree. We call the lazy trees as defined above *initial lazy trees*. (We will be adding and removing a few nodes from the initial lazy tree in order to obtain other lazy trees, which are the lazy trees that are actually analyzed.)

The intuition behind the lazy tree is that if all the lazy nodes were restored to a left path then the lazy potentials would be the actual global potentials (perhaps with some swapping). Actually, difficulties are caused by the fact that the left subtree of the path may increase in size through rotations between the lazy tree and the remainder of the splay tree. So strictly speaking the intuition may be incorrect. More precisely, the constant potentials q provided to the lazy nodes will pay for rotations among lazy tree nodes until these nodes have their global potentials restored (in general, this will not happen by recreating the left path from which these nodes originated).

Let us return to the cost of the insertions. A global insertion, as noted above, costs $e(3 \log n + 1)gp + gp \log n$ units. We see below that the presence of lazy trees adds a further $4gp \log n$ units to the cost of the global insertions, for a total of

$$e(3 \log n + 1)gp + 5gp \log n \text{ units} \tag{5}$$

In Section 4 we show how to modify the analysis of global and local insertions to account for the presence of lazy trees.

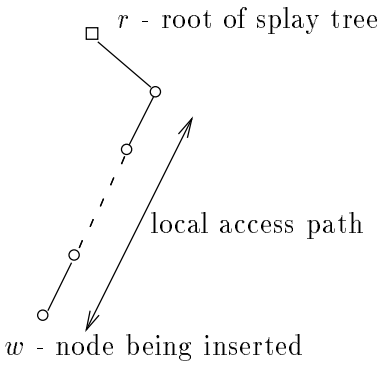


Figure 8: The local access path

$e-1$ local insertions, each node on the local access path, except the topmost, will have at least $(2^{e-1}-1)s$ spare rotations. The rotations in subsequent local insertions will be self-paying, apart from the rotation involving the splay tree root. These subsequent local insertions are called *true local insertions*.

In fact, we provide another $gp \log n$ units to the global insertion. Following the first $e-1$ local insertions we provide a *reserve* potential to each global node on the local access path. The reserve for global node u is defined as follows: let v be the first proper global descendant of u on the local access path, if any; let $g_rank(v)$ be v's global rank, if v is present; otherwise, let $g_rank(v) = 0$. Then $reserve(u) = gp(g_rank(u) - g_rank(v))$. The role of the reserve potential will become clear later.

We need one further constant, q, to be specified later. We choose e so that

$$(2^{e-1}-1)s \geq q + (s+1) \tag{4}$$

We note $e \geq 2$. In a true local insertion, each node v removed from the local access path is given potential q; $s+1$ will bound the cost of the rotation involving v. This is paid for by the $(2^{e-1}-1)s$ potential associated with the node removed from the local access path. At the end of the sequence of local insertions, the nodes remaining on the local access path, apart from the topmost node, are also given potential q (these nodes all have $(2^{e-1}-1)s$ potential at hand, which more than suffices).

Consider a true local insertion. The inserted item, e, is not provided a global potential; rather, following the final rotation of the insertion, e becomes the root of the splay tree and acquires the global potential associated with the old root of the splay tree. There are three types of rotation on the local access path. They all involve a couple u, v, with u the parent of v. Either both u and v are local, or one of u or v is local and the other is global, or both are global. Only in the last case is a global node removed from the local access path. The rotations, apart from those involving two global nodes, cost at most $s+1$ units, for there is no increase in global rank in such rotations and there are no debits to remove (for following the global insertion there are only large debits on the nodes of the local access path, which are removed by the first local insertion; as $e \geq 2$, the current insertion is not the first local insertion). In the case of a rotation involving two global nodes we swap their global potentials. The node remaining on the local access path still has the correct global potential, but the node

Remark 1 *Later, the form of the blocks will be generalized to allow more complex situations involving local nodes. Specifically, we will allow local nodes to carry other debits. For the above analysis to continue to apply, it will suffice that:*

(i) If a local node on its block's right (resp. left) path carries a new debit then its parent and right child (resp. left child) are both local nodes of the block.

(ii) A zig-zig rotation in which the couple comprises two local nodes pays for the removal of all debits on the couple's nodes, plus $s+1$ rotations, if the rank of the accessed item does not increase.

Indeed, in a subsequent paper, we will introduce a hierarchy of blocks. A block will comprise a contiguous set of items (with respect to the items' sorted order). A superblock will comprise a contiguous set of blocks. (In this section, the superblock is simply the whole splay tree.) As here, the root of each block is a global item and the other items are local items. Suppose each global item has a positive integer weight and each local item has non-negative integer weight. Suppose further that there are small and large debits, which obey Invariants 1–4; there may be additional types of debits also, but they must obey the following invariant:

Invariant 5 *Let v be a local node on an extreme path of block B. Suppose that in the splay tree v is the left (resp. right) child of its parent u. v can have an additional debit only if:*

(i) u is a local node of block B,

(ii) v has a left (resp. right) child w which is a local node of block B.

Then the following theorem has been shown:

Lemma 5 *Let v be a global node in a superblock B. Suppose that v is the accessed item. Let I be the increase in global rank undergone by v during some (arbitrary) portion P of the access. The amortized cost of the rotations in portion P of the access is bounded by $(3I+1)gp$ units, where $gp, ld = 17(s+1)$, $c = 18(s+1)$ and $sd = s+1$, provided the conditions of Remark 1 hold. The amortized cost of the traversal comprises a payment of $s+1$ rotations for each couple, the removal of debits on the path traversed and the removal of other debits needed to restore Invariants 3 and 4, minus the debits that are created.*

3 Local Insertions

We show how to analyze a sequence of local insertions. This leads to the introduction of *lazy trees* which forces a reanalysis of global and local insertions. Following this reanalysis, the overall analysis is readily concluded.

Recall that the insertion of each block comprises one global insertion followed by a sequence of $\log n - 1$ local insertions. Each local insertion traverses a left path up to the right child of the splay tree root (see Figure 8); we call this path the *local access path.* By providing $e(3\log n + 1)gp$ units to the global insertion we can treat the first $e-1$ local insertions as if they were global insertions. Throughout these $e-1$ local insertions, spare rotations will be accumulating (though for reasons that will become clear later it may be that no spares accumulate during the global insertion itself); more precisely, each couple receives s spares in each of these local insertions (see Lemma 7). These spares, plus the spares already accumulated by the nodes of the couple are given to the node remaining on the local access path. So after

(2) By the creation of an abutting portion, in the case that this abutting portion is not traversed. This can only occur for the top node v of a couple and then only if both nodes in the couple are global and the rotation is a zig-zig rotation. Here, the contribution to $Cost3$ is at most $2sd$.

Next, we determine, in turn, the contribution to $Cost2 + Cost3$ of the truncated segment and of the first couple.

First, we consider the truncated segment. By Lemma 1, the contribution to $Cost2$ is at most $7sd$. For a contribution to $Cost3$, the discussion of the previous paragraph shows we need only consider possibility (2) since global nodes in the truncated segment do not have a change in global rank. Likewise, possibility (2) cannot arise for the couples of the truncated segment each have at most one global node. We conclude that the truncated segment contributes at most $7sd$ to $Cost2 + Cost3$.

We turn to the leading couple. The contribution to $Cost2$ is at most sd for each local node it contains. By the above discussion regarding contributions to $Cost3$, the leading couple may contribute up to $4sd$ to $Cost3$, but only if both nodes of the couple are global; if there is only one global node in the couple then the contribution to $Cost3$ is at most $2sd$. In any event, the total contribution of the first couple to $Cost2 + Cost3$ is at most $4sd$. •

In case A, the cost of the rotation is paid for either by the drop in global potential, which provides at least gp units, or, if there is an increase in global rank, by gp units of the at least $3gp$ units provided for this rotation. In case B, the middle node in the sequence of three contiguous nodes is given a large debit, which pays for the rotation. So it suffices to have:

$$gp, ld \geq 6(s+1) + 11sd \tag{3}$$

Now, we show how to pay for the incomplete segment, if present. Recall that we provided $(3\log n + 1)gp$ units to pay for the present insertion. The $+gp$ term is used to pay for the incomplete segment; in addition, this term is used to pay for the incomplete rotation, if any; however, the $+gp$ term does not need to account for any increase in rank, on the part of the inserted item during the incomplete rotation, for this has already been accounted for. Clearly, the result of Lemma 2 applies here too (in fact, a tighter bound can be shown). Here too, Equation 3 suffices.

On taking equalities in Equations 1–3, we conclude:

Lemma 3 *A global insertion has amortized cost at most* $(3\log n + 1)gp$ *units, where* $gp, ld = 17(s+1)$, $c = 18(s+1)$ *and* $sd = s+1$.

Actually, we have shown:

Lemma 4 *The path traversed in an access of an item can be partitioned into segments, such that for any segment* σ, *the amortized cost of the rotations for the couples of* σ *is at most* $gp + 2gp \cdot I_\sigma$, *where* I_σ *is the increase in rank undergone by the accessed item in the last rotation of segment* σ. *Furthermore,*

(i) $gp \cdot I_\sigma$ *units of the cost are used to raise the potential of the accessed item.*

(ii) $gp \cdot I_\sigma$ *units of the cost are used to raise the potential of the lower global item, if any, in the final couple of segment* σ.

(iii) gp *units are used to pay for all the rotations of* σ, *and all the associated spare rotations and debit removals.*

Type 2 See Figure 7. By Invariants 2 and 4, node u does not have a debit (for $g_rank(x) = g_rank(u)$ as x and u are both in the truncated segment). The rotation is paid for by giving u a small debit following the rotation. Again, it is straightforward to check that Invariants 1–4 are unaffected. Here too, Equation 1 suffices.

Type 3 This is the Case 1 analyzed earlier. We need to pay for the rotation plus the removal of two small debits or one large debit. Again, it is straightforward to check that Invariants 1–4 are unaffected. So it suffices that:

$$c \geq s + 1 + \max\{2sd, ld\} \tag{2}$$

The remaining rotations are paid for by the first couple in the segment, which was removed in truncating the segment and which causes a violation of at least one of the conditions (i)–(iv), except in the case of an incomplete topmost segment, which is handled subsequently. Either the rotation involving the first couple causes a change in global potential (case A) or it creates a sequence of three contiguous local nodes from the same block (Case B), or possibly both. Both cases involve three costs.

- $Cost1$. The rotation and spares for each couple: $\leq 6(s+1)$.
- $Cost2$. Removal of small debits from nodes on the segment; $Cost2$ is analyzed below. (A node with a large debit will always be a marked node of Type 3.)
- $Cost3$. Removal of small debits for local nodes that now violate Invariant 3 or 4; $Cost3$ is analyzed below.

Lemma 2 *For each segment* $Cost2 + Cost3 \leq 11sd$.

Proof. A definition is helpful here. Let B be a block and let v be the root of B. Consider an extreme path of block B and consider the topmost portion that is contiguous in the splay tree. If this topmost portion is incident on v in the splay tree then it is called an *abutting extreme path portion* for v, or an *abutting portion* for short. (*Comment.* Node v of Invariant 3 is the top node of an abutting portion for u, while node w of Invariant 4 is the bottom node of an abutting portion for u.)

Next, we make some observations about $Cost3$. Note that none of the nodes from marked couples on the traversed path retain small debits; so these nodes do not contribute to $Cost3$. Contributions to $Cost3$ can arise in one of two ways:

(1) Through a traversed global node v having a reduction in rank. Then, on an already abutting portion (for v) which continues to be abutting (for v), if the portion was not traversed, we may need to remove up to two single debits in order to maintain Invariants 3 and 4. v can have at most one such abutting portion. This contributes up to $2sd$ to $Cost3$. This contribution can be a consequence of either a zig-zag rotation or a zig-zig rotation. We examine each in turn.

   (a) In a zig-zag rotation, there may be one such abutting portion for each global node in the couple. In this case the contribution to $Cost3$ can be as large as $4sd$.

   (b) In a zig-zig rotation only the top node in the couple can retain such an abutting portion; so here the contribution to $Cost3$ is at most $2sd$.

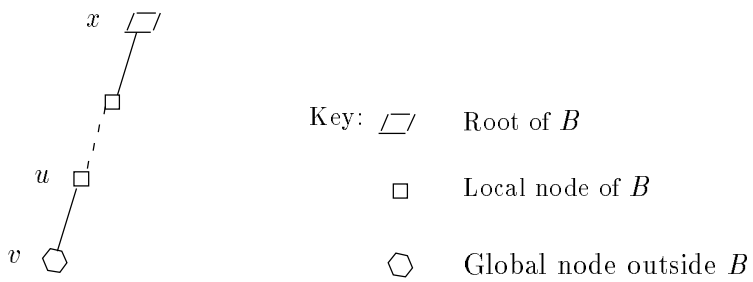


Figure 7: Type 2 couple

Type 2 See Figure 7. Suppose u is a local node and let x be the root of u's block. Further suppose u is on the left (resp. right) path descending from x. Let v be the left (resp. right) child of u; if x is in the truncated segment and if v is global then both u and v are marked.

Type 3 Suppose u and v are both local nodes of the same block; then both u and v are marked.

It is not hard to see that the marked nodes are all involved in zig-zag rotations.

We can now prove a bound on the length of a truncated segment.

Lemma 1 *A truncated segment comprises at most 10 unmarked nodes, of which at most 7 are local.*

Proof. *Observation 1.* Define the *left (resp. right) side* of the segment to comprise those nodes that are to the left (resp. right) of the item being inserted. From (ii) we deduce that one side, at least, contains no global nodes; without loss of generality suppose that this is the right side. As the right side contains no global nodes, its nodes must all come from the same block; so, by (iv), the right side comprises at most two sets of contiguous nodes, each one of at most two nodes. Because of Type 3 markings the right side contains at most two unmarked nodes.

The left side of the segment is partitioned in the obvious way into subsegments by the nodes on the right side. By Observation 1, the left side comprises at most three subsegments.

Observation 2. There are at most two couples that include both a node on the left side and a node on the right side. This follows from (iv) applied to the right side of the segment.

Observation 3. There is at most one unmarked couple within each subsegment; if present, this couple comprises the topmost global node and its parent, a local node. For the marking strategy marks all other couples within the subsegment. This is a total of at most three unmarked couples.

Thus there are a total of at most 5 unmarked couples, and at most 2 of these do not include a global node; so there are at most 7 unmarked local nodes. •

Next, we show how to pay for the rotations (and associated spares) along a segment. Each marked couple pays for its rotation (and spares), as follows.

Type 1 See Figure 6. By Invariants 2 and 3, node v does not have a debit. The rotation is paid for by giving node u a small debit following the rotation. It is straightforward to check that Invariants 1–4 are unaffected. So it suffices that:

$$sd \geq s + 1 \tag{1}$$

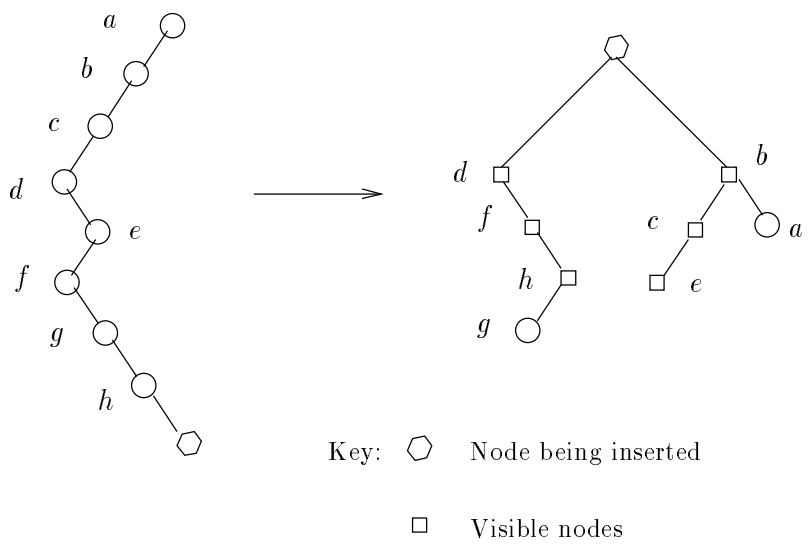


Figure 5: Visible nodes

(iv) See Figure 5. Define a node to be *visible* if it is involved in a zig-zag rotation or it is the lower node in a couple. (Intuitively, the visible nodes are those that remain on one of the traversed paths following the splay. Note that the splay, in general, creates two traversed paths.) For each block there are at most two visible nodes in the truncated segment that are local following the rotation.

The topmost segment is said to be *incomplete* if it satisfies conditions (i)–(iv) prior to truncation. We consider an incomplete segment to comprise a (trivially) truncated segment.

Next, we mark the following types of coupled nodes in each truncated segment. The rotations involving marked couples are self-paying, as is demonstrated later. For each type below, suppose the segment includes a couple, u, v, with u the parent of v.

Type 1 See Figure 6. Suppose u is the root of v's block; then both u and v are marked.

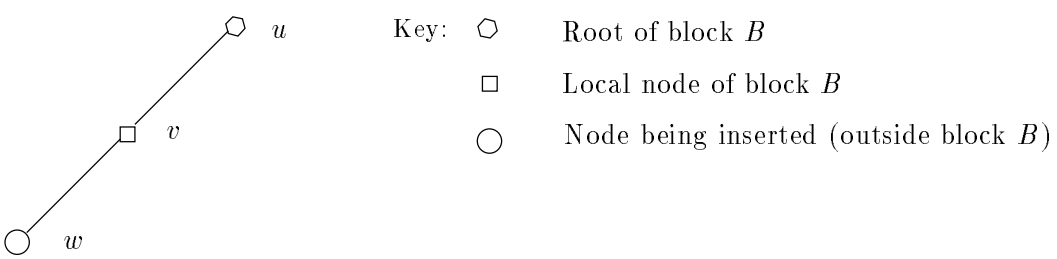


Figure 6: Type 1 couple

(i) u is a local node of block B,

(ii) v has a left (resp. right) child w which is a local node of block B, and

(iii) neither u nor w carry any debit.

Invariant 3 *Let u be the root of block B. Let v be a child of u. If v is in B, then v can have a small debit only if $g_rank(v) < g_rank(u)$.*

Invariant 4 *See Figure 4. Let u be the root of block B. Let P be an extreme path of the*

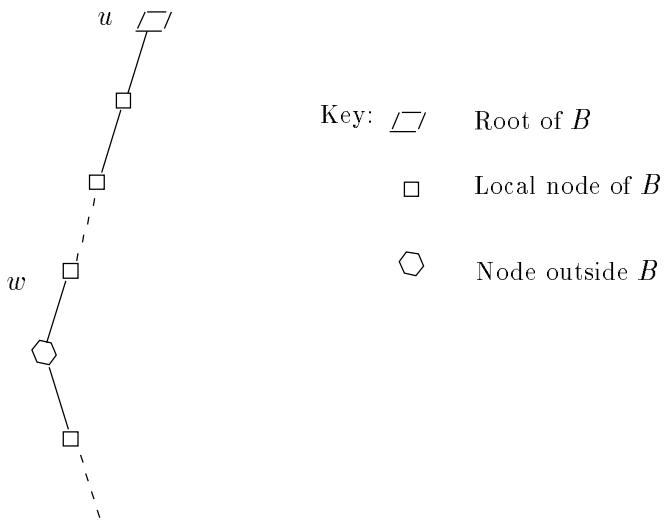


Figure 4: A small debit

subtree (of the splay tree) rooted at u. Let w be the bottommost node on the path P which is in B. w can have a small debit only if $g_rank(w) < g_rank(u)$.

For the purposes of the analysis the access path is partitioned into segments. Each segment comprises an even number of nodes, every two nodes on the path forming the coupled nodes of a rotation of the present splay operation. The segments are created by a traversal of the access path from bottom to top; each segment is chosen to have the maximum length such that following the removal of its front (top) two nodes, the (truncated) segment satisfies the following conditions:

(i) The node being inserted has the same global rank throughout the rotations involving the truncated segment.

(ii) Each global node on the truncated segment has the same rank following its rotation.

(iii) (This is implied by (i) and (ii).) Each couple in the truncated segment includes at least one local node (i.e., it does not comprise two global nodes).

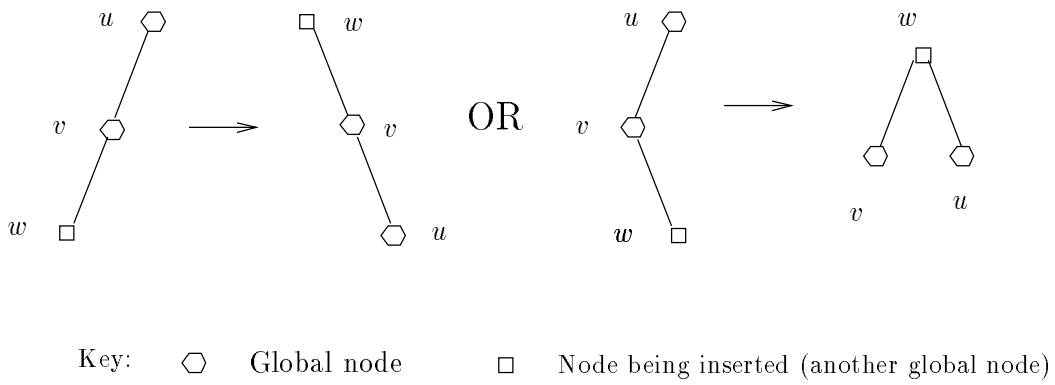


Figure 2: A couple comprising two global nodes

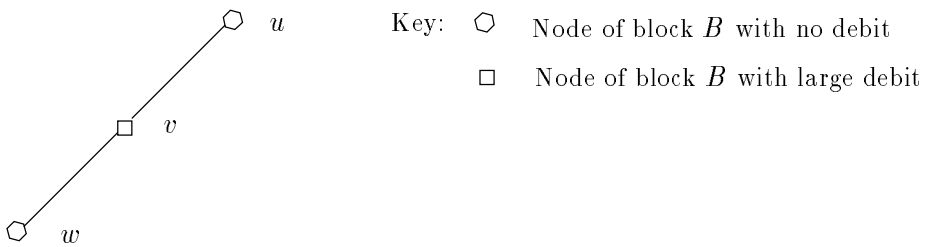


Figure 3: Large debits

In the analysis of global insertions, whenever a rotation is performed (and paid for) another s spare rotations are also paid for, s a constant to be specified later. The spare rotations are needed subsequently, to handle the effects of local insertions. We provide $(3\log n + 1)gp$ units to pay for a global insertion. If a rotation increases the global rank of the item being inserted by Δ, then the rotation is paid for with $3\Delta gp$ of these units. Note that the rank of an inserted item increases by at most $\log n$ (from ≥ 0 to $\log n$). If the last rotation (the one involving the root of the splay tree) involves just two nodes, we call it an *incomplete rotation*; the remaining gp units are used as additional payment for the incomplete rotation, if any.

To avoid special cases it is convenient to redefine the access path for an insertion to exclude the splay tree root r in the event that r is involved in an incomplete rotation. Now consider a rotation performed during the splay along the access path. Of the three nodes involved in the rotation, the top two are called the *coupled* nodes of the rotation, or a *couple* for short. The analysis focuses on the coupled nodes in a rotation. In order to provide the reader some intuition we describe two simple cases.

Case 1 The coupled nodes are both local nodes in the same block (see Figure 1). Nodes u

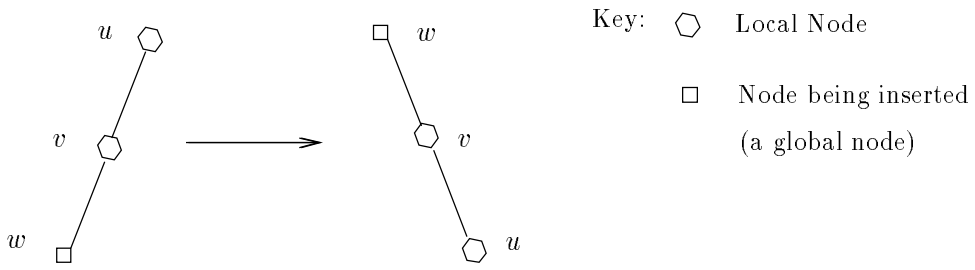


Figure 1: A couple comprising two local nodes

and v are on an extreme path of their block, without loss of generality the left path; v is the left child of u, and w, the node being inserted, is the left child of v. Since node u leaves the extreme path of its block it loses potential c. This pays for the removal of debits (small or large) from nodes u and v and in addition pays for $s+1$ rotations. Later, we see that we want $c \geq s + 1 + \max\{2sd, ld\}$.

Case 2 The coupled nodes are both global nodes (see Figure 2). Again, let w be the inserted node, let v be w's parent and u be v's parent. This case is analyzed using the centroid potential analysis of Sleator and Tarjan. The cost is at most 3 times the jump in global potential from w to u; to pay for $s+1$ rotations it suffices to have $gp \geq s + 1$; later, we see that in fact we want $gp \geq 6(s+1) + 11sd$.

The full analysis is more involved. We begin by stating several invariants about the debits.

Invariant 1 *Only local nodes on an extreme path of their block can have debits.*

Invariant 2 *See Figure 3. Let v be a local node on an extreme path of block B. Suppose that in the splay tree v is the left (resp. right) child of its parent u. v can have a large debit only if*

- The potential depends on the access sequence; that is, identical trees, if created by distinct access sequences, may have different potentials. (Actually, this is a difference 'in principle'; we have not constructed any examples to demonstrate it.) Nonetheless, the bounds we obtain do not depend on the input sequence.

These techniques are used later in the proof of the Dynamic Finger Conjecture [C93]. Some of the techniques are slightly generalized so that they can be used as modules in this later paper.

2 Global Insertions

We start with some definitions.

The *left path* of a tree comprises the nodes traversed in following left child pointers from the root, but excluding the root itself; the *right path* is defined analogously. Collectively, they are called *extreme paths*. Extreme paths of a subtree rooted at v are defined analogously; they are also called v's extreme paths. Node u is a *right ancestor* of node v if u is an ancestor of v that is to the right of v in symmetric order; note that v must be in u's left subtree. A *left ancestor* is defined analogously. A *right edge* is an edge to a right child; a *left edge* is defined analogously.

A *block* B is an interval $[i, j] \subseteq [1, n]$ of items; here a block always comprises $\log n$ items with $i = c \log n + 1$ and $j = (c + 1) \log n$, for some integer c. Any block $[i, j]$ induces a binary tree T_B, called the *block tree* of B, which comprises exactly those nodes of the splay tree, S, containing items i to j; they are called the *nodes* of B. Loosely speaking, the block tree is constructed by shrinking paths in S between nodes of B to single edges. More formally, the root, r, of T_B is the lowest common ancestor of nodes i and j in S. The left (resp. right) subtree of r is the tree induced by the set of items in S to the left (resp. right) of r, if non-empty; otherwise the subtree is empty.

The *root* of block B is the root of the corresponding block tree. The *global nodes* are exactly the block roots. The remaining nodes are called *local nodes*.

Every node on an extreme path carries a potential of c units, c a constant to be specified later. Also, each node on an extreme path may carry a debit, either *small* or *large*, comprising sd and ld units, respectively; sd and ld are constants that are specified later. A node may not have both a small and a large debit. We note that for any block B the only nodes that may be visited henceforth are those presently on the extreme paths of its block tree, plus its root.

Each global node is given a centroid potential, called its *global potential*; it is defined on the splay tree using the following weights: each global node has weight 1 and each local node has weight 0. It is convenient to define a *global rank* for all nodes: this is the centroid rank in the splay tree under this weighting; $g_rank(v)$ denotes the global rank of v. Global nodes have a global potential equal to gp times their global rank, gp a constant to be specified later; the local nodes do not have a global potential.

To avoid difficulties created by the insertion of weighted nodes, for the purposes of the analysis we preinsert all the nodes into an ordinary binary search tree (without splays) in the order of their insertion into the splay tree. Then, in this same order, we access each node, in the binary search tree, by a splay operation. It is easy to see that the rotations performed on the binary search tree are identical to those performed on the splay tree. However, now each insertion can be treated as an access.

general results related to the above conjectures follow. [T85] proved the Scanning Theorem, a special case of the Traversal Conjecture (also a special case of the Dynamic Finger Conjecture): accessing the items of an arbitrary splay tree, one by one, in symmetric order, takes time $O(n)$. Sundar [Su89] considered various classes of rotations on binary trees. Among other things, this led to results concerning the Deque Conjecture; formulated by Tarjan [T85], it states that if a splay tree is used to implement a deque, in the natural way, then a sequence of m operations on a deque, initially of n items, take time $O(m+n)$; Sundar proved a bound of $O((m+n)\alpha(m+n))$.

In this paper we investigate a special instance of the Splay Sorting problem, which is related to the Dynamic Finger Conjecture. Splay Sorting is defined as follows. Consider sorting a sequence of n items by inserting them, one by one, into an initially empty splay tree; following the insertions, an inorder traversal of the splay tree yields the sorted order. We call this *Splay Sort*. A corollary of the Dynamic Finger Conjecture is the:

Splay Sort Conjecture. Let S be sequence of n items. Suppose the ith item in S is distance I_i in sorted order from the $(i-1)$th item in S, for $i > 1$. Then Splay Sort takes time $O(n + \sum_{i=2}^{n} \log(I_i + 1))$.

Incidentally, an interesting corollary of the Splay Sort Conjecture is the:

Splay Sort Inversion Conjecture. Let S be sequence of n items. Suppose the ith item in S has I_i inversions in S (counting inversions to both the left and right). Then Splay Sort takes time $O(n + \sum_{i=1}^{n} \log(I_i + 1))$.

In the remainder of this paper we prove the Splay Sort Conjecture for the following type of sequence. Suppose the sorted set of n items is partitioned into subsets of $\log n$ contiguous items, called *blocks*. Consider an arbitrary sequence in which the items in each block are contiguous and in sorted order. We call such a sequence a $\log n$-*block sequence*. We show an $O(n)$ bound for Splay Sorting a $\log n$-block sequence. It is convenient to assume that the set being sorted comprises the integers $1 \ldots n$.

In brief, our analysis has the following form. The first insertion in each block is provided $\Theta(\log n)$ potential; it is called a *global* insertion. Every other insertion in the block is provided $O(1)$ potential; these insertions are called *local* insertions. In Section 2, we analyze the global insertions; then, in Section 3, we modify the analysis to take account of local insertions.

Our work has a number of interesting features and introduces several new techniques for proving amortized results.

- We introduce the notion of *lazy potential*; this notion can be viewed as a tool for designing potential functions. The lazy potential is a refinement of an initial potential function that avoids waste when potentials decrease. (For an example of waste, consider the following splay tree analyzed using the centroid potential. The tree is a path of n nodes, each of unit weight. The last node on the path is accessed. The resulting splay will have a real cost of $\Theta(n)$, but will reduce the potential by $\Theta(n \log n)$. The desired amortized cost of this operation is $O(\log n)$, so essentially all the reduction in potential is wasted.) The idea of the lazy potential is to keep an old potential, ϕ_{old}, in situations in which the creation of a new (larger) potential, ϕ_{new}, may be followed by a return to the ϕ_{old} potential, with essentially $\phi_{new} - \phi_{old}$ potential being wasted. Not surprisingly, some care is needed in choosing which operations to treat as 'lazy'.

Henceforth, we refer to the rotation, single or double, performed by the splay step as a *rotation* of the access or splay operation. A rotation is the basic step for our analysis; the cost of one rotation is termed a *unit*; clearly, this is a constant.

Sleator and Tarjan use the following *centroid potential* to analyze the amortized performance of a splay operation. Node x is given weight, $wt(x)$, equal to the number of nodes in its subtree; they define the *centroid rank* of x, or simply the *rank* of x to be $rank(x) = \lfloor \log wt(x) \rfloor$ (our terminology). Each node is given a centroid potential equal, in units, to its centroid rank. Let δ denote the increase in centroid rank from x to g, if g is present; otherwise it denotes the increase in centroid rank from x to p. Sleator and Tarjan showed that the amortized cost of the splay step if g is present is at most 3δ units, while if g is not present the cost is at most $\delta + 1$ units. Since the total increase in rank for the complete access is bounded by $\log n$, the amortized cost of an access is at most $3 \log n + 1$ units. More generally, this analysis can be applied to weighted trees, in exactly the same way. We call this the *centroid potential analysis*.

The operation $insert(x)$ is performed as follows: first, item x is inserted as in a binary search tree and then the operation $splay(x)$ is carried out. Clearly, the cost of an insertion is dominated by the cost of the corresponding access. So, subsequently, when analyzing the cost of insertions we only count the cost of the splays themselves.

Now, we list the conjectures formulated by Sleator and Tarjan.

- **Dynamic Optimality Conjecture**. Consider any sequence of successful accesses on an n-node binary search tree. Let A be any algorithm that carries out each access by traversing the path from the root to the node containing the accessed item, at a cost of one plus the depth of the node containing the item, and that between accesses performs an arbitrary number of rotations anywhere in the tree, at a cost of one per rotation. Then the total time to perform all the accesses by splaying is no more than $O(n)$ plus a constant times the time required by algorithm A.

- **Dynamic Finger Conjecture**. The total time to perform m accesses on an arbitrary n-node splay tree is $O(m + n + \sum_{j=1}^{m} \log(d_j + 1))$, where, for $1 \leq i \leq m$, the jth and $(j-1)$th accesses are performed on items whose ranks differ by d_j (ranks among the items stored in the splay tree). For $j = 0$, the jth item is interpreted to be the item originally at the root of the splay tree.

- **Traversal Conjecture**. Let T_1 and T_2 be any two n-node binary search trees containing exactly the same items. Suppose the items in T_1 are accessed one after another using splaying, accessing them in the order they appear in T_2 in preorder (the item in the root of T_2 first, followed by the items in the left subtree of T_2 in preorder, followed by items in the right subtree of T_2 in preorder). Then the total access time is $O(n)$.

Sleator and Tarjan state that the Dynamic Optimality Conjecture implies the other two conjectures (the proof is non-trivial).

There have been several works on, or related to, the optimality of splay trees [STT86], [W86,T85,Su89,Luc88a,Luc88b]. [STT86] shows that the rotation distance between any two binary search trees is at most $2n - 6$ and that this bound is tight; they also relate this to distinct triangulations of polygons; although connected to the splay tree conjectures, this result has no immediate application to them. [W86] provides two methods for obtaining lower bounds on the time for sequences of accesses to a binary search tree; while some specific tight bounds are obtained (such as accessing the bit reversal permutation takes time $\Theta(n \log n)$) no

On the Dynamic Finger Conjecture for Splay Trees. Part I: Splay Sorting $\log n$-Block Sequences*

Richard Cole; Bud Mishra, Jeanette Schmidt, Alan Siegel

Courant Institute, New York University; Polytechnic University;
Ecole Normale Supérieure

Abstract

A special case of the Dynamic Finger Conjecture is proved; this special case introduces a number of useful techniques.

1 Introduction

The splay tree is a self-adjusting binary search tree devised by Sleator and Tarjan [ST85]. It supports the operations search, insert and delete, collectively called *accesses*. The splay tree is simply a binary search tree; each access will cause some rotations to be performed on the tree. Sleator and Tarjan showed that a sequence of m accesses performed on a splay tree takes time $O(m \log n)$, where n is the maximum size attained by the tree ($n \leq m$). They also showed that in an amortized sense, up to a constant factor, on sufficiently long sequences of searches, the splay tree has as good a running time as the optimal weighted binary search tree. In addition, they conjectured that its performance is, in fact, essentially as good as that of any search tree. Before discussing these conjectures it will be helpful to review the operation of the splay tree and the analysis of its performance. The basic operation performed by the splay tree is the operation $splay(x)$ applied to an item x in the splay tree. $splay(x)$ repeats the following step until x becomes the root of the tree.

Splay step.
Let p and g be respectively the parent and grandparent (if any) of x.
Case 1. p is the root: Make x the new root by rotating edge (x,p).
Case 2 — the zig-zag case. p is the left child of g and x is the right child of p, or vice-versa: Rotate edge (x,p), making g the new parent of x; rotate edge (x,g).
Case 3 — the zig-zig case. Both x and p are left children, or both are right children: Rotate edge (p,g); rotate edge (x,p).

*This work was begun while all four authors were at New York University; Richard Cole continued the work while on leave at Ecole Normale Supérieure. Presently Jeanette Schmidt is at Polytechnic University. The work was supported in part by NSF grants CCR-8702271, CCR-8902221, CCR-8906949, CCR9202900, DMS-8703458, ONR grants N00014-85-K-0046 and N00014-89-J3042, and by a John Simon Guggenheim Memorial Foundation Fellowship. The work of Richard Cole was made possible, in part, by the hospitality of the Laboratoire d'Informatique, Ecole Normale Supérieure; it is associated with and supported by CNRS as URA 1327.