SIAM J. COMPUT.
Vol. 19, No. 5, pp. 902-906, October 1990

009

ON THE COMPLEXITY OF A GAME RELATED TO THE DICTIONARY PROBLEM*

K. MEHLHORN†, ST. NÄHER†, AND M. RAUCH†

Abstract. A game on trees that is related to the dictionary problem is considered. There are two players, A and B, which take turns. Player A models the user of the dictionary and player B models the implementation of it. At his turn, player A modifies the tree by adding new leaves and player B modifies the tree by replacing subtrees. The cost of an insertion is the depth of the new leaf, and the cost of an update is the size of the subtree replaced. The goal of player A is to maximize cost and the goal of B is to minimize it. It is shown that there is a strategy for player A, which forces a cost of $\Omega(n \log\log n)$ for an n-game, i.e., a game in which each player takes n turns, and that there is a strategy for player B, which keeps the cost within $O(n \log\log n)$.

Key words. dictionary problem, lower bound, hashing, search tree

AMS(MOS) subject classification. 68

1. Introduction. We consider a two-person game on trees, which is related to the dictionary problem. The two players A and B take turns. Player A models the user of the dictionary and player B models the implementation of it. At his turn, player A modifies the tree by replacing a leaf by a tree consisting of a single node with two children, cf. Fig. 1. This is called an *insertion.* The *cost of an insertion* is the depth of the leaf replaced. At his turn, player B performs zero or more *updates.* An update replaces a subtree by another subtree with the same number of leaves. A precise definition is as follows.

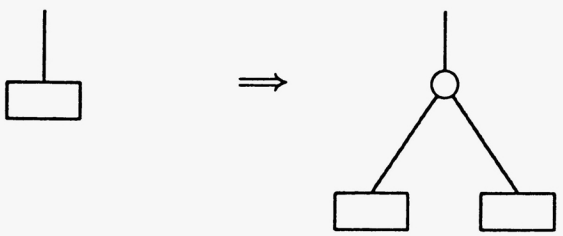

FIG. 1. *An insertion. The cost of an insertion is the depth of the new node v.*

Let T be a rooted tree and v a node of T. A *subtree* rooted at v is a connected subgraph of T with the following two properties:

(1) It contains v but no ancestor of v.

(2) If a node $w \neq v$ belongs to the subgraph, then all siblings of w also belong to the subgraph.

A subtree T' is *complete* if for every node w in T' all children of w also belong to the subtree. The complete subtree rooted at v is denoted by T_v.

* Received by the editors May 24, 1989; accepted for publication (in revised form) February 6, 1990. The research was partially supported by the Deutsche Forschungsgemeinschaft under grant SPP1.

† Fachbereich Informatik, Universität des Saarlandes, 6600 Saarbrücken, Federal Republic of Germany.

An *update* of a tree T is specified by a subtree T' of T rooted at some node v of T and a tree T'' having the same number of leaves as T'. The result of the update is a tree $\tilde{T}$, which can be obtained as follows:

(1) Delete all interior nodes of T' from T.

(2) Make the root of T'' a child of $parent(v)$.

(3) Identify the ith leaf of T' with the ith leaf of T'' for $1 \leqq i \leqq m$, where m is the number of leaves of T'.

The *cost of an update* is the number of leaves of tree T'. Figure 2 gives an example for an update of cost 6. The trees T' and T'' are shown bold. The leaves of T' and T'' are numbered from 1 to 6.

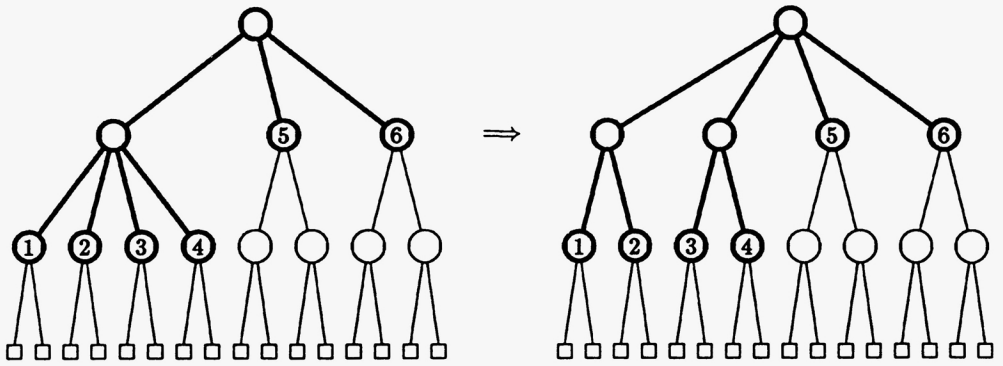

FIG. 2

An *n-game* starts with a trivial tree consisting of a single node, and ends after each player takes n turns. We will prove the following results:

THEOREM 1. *There is a strategy for player A such that the cost of any n-game is* $\Omega(n \log \log n)$.

THEOREM 2. *There is a strategy for player B such that the cost of any n-game is* $O(n \log \log n)$.

Theorems 1 and 2 are proven in §§ 2 and 3, respectively. Our game on trees models all solutions to the dictionary problem where a search is performed by repeated splitting of the dictionary until a dictionary of size one results. Special cases are multilevel hashing and search trees.

In (multilevel) hashing schemes a hash function is used for the splitting process, i.e., in each node of the tree a hash function h is stored, and the ith subtree of a node is a dictionary for all keys that are mapped to i by function h. Hashing with chaining as well as the perfect hashing schemes of Fredman, Komlós, and Szemeredi [FKS], Aho and Lee [AL], and Dietzfelbinger et al. [DKMMRT] are examples of multilevel hashing schemes. In these examples our definition of insertion cost fairly reflects the cost of an insertion, but our definition of update cost underestimates the true cost since it only measures the amount of structural change in the tree but ignores the cost of finding appropriate hash functions.

In search trees, e.g., AVL-trees or (2, 3)-trees, comparisons between keys are used for the splitting process. The updates correspond to small structural changes of the search tree; e.g., Fig. 2 shows how a node of degree 4 is split into 2 nodes of degree 2 in a (2, 3)-tree. In these examples our definition of update cost fairly reflects the true cost, but our definition of insertion cost underestimates the true cost because the

amount of work needed to identify the successor of a node on the search path depends on the degree of the node.

Together, these examples show that many solutions for the dictionary problem are within our model, and hence the lower bound given in Theorem 1 applies to them. The examples also indicate that the upper bound given in Theorem 2 does not imply a solution to the dictionary problem with the same performance, in fact, no such solution is known.

The work reported here extends work of Dietzfelbinger et al. [DKMMRT], who showed that player A can force a cost $\Omega(n \log n)$ in an n-game provided that player B always has to replace complete subtrees. This is a severe restriction and excludes balanced search trees. Note that Dietzfelbinger et al. [DKMMRT] have shown that in their model there is always a single turn (of either A or B), that has cost $\Omega(\sqrt{n})$ in any n-game. However, balanced search trees keep the cost of any turn in $O(\log n)$ and hence are excluded by their model. They are included in our model.

2. The lower bound. Player A follows a very simple strategy called the "insert-into-heaviest-child-strategy" (IHS). Let the weight $w(v)$ of a node v be the number of leaves of the complete subtree rooted at v. Then the path of insertion $v_0, v_1, \cdots, v_l$ is defined as follows: v_0 is the root of the current tree, v_{i+1} is a heaviest child of v_i, and v_l is a leaf.

THEOREM 1. *If player A plays according to the "insert-into-heaviest-child-strategy," then the cost of an n-game is* $\Omega(n \log \log n)$.

Proof. Consider any n-game where A plays IHS. We may assume without loss of generality that $n \geqq 4 \cdot 3^7$. Let $K = \max\{k \in \mathbb{N};\ k^{(2^k-1)} \leqq n/4\}$. Then $K = \Theta(\log \log n)$. We now distinguish cases. Assume first that there are at least $n/2$ insertions that have cost at least $K+1$ each. Then the total (insertion) cost is clearly $\Omega(n \log \log n)$.

Assume next that there are at least $n/2$ insertions with cost K or less. Let $D = (n/4)^{1/(2^K-1)}$. Then $D \geqq K \geqq 3$ by the definition of K. For a node v of tree T we use depth (v, T) to denote the depth of node v in T and deg (v, T) to denote the degree of node v in T; the depth of the root is 0. Nodes are created and destroyed by updates and insertions. Consider an update where subtree T' is replaced by subtree T''. Then T' and T'' have the same number of leaves and the update identifies the ith leaf of T' with the ith leaf of T''. We therefore say that the update *destroys* the interior nodes of T' and *creates* the interior nodes of T''. The leaves of T' are identified with the leaves of T'' and hence are said to *exist* before and after the update. In the example of Fig. 2 the update destroys two nodes and creates three nodes. An insertion creates a leaf and a node of degree 2. With this definition, the degree of a node never changes during its existence. We may therefore write deg (v) instead of deg (v, T). The depth of a node v, however, can change over time. For the ith update we use V_i to denote the set of nodes of degree 2 or more created by the update. Also, $V = \bigcup_{i=1}^{n} V_i$ is the set of nodes that are created by updates and have degree two or more.

LEMMA 1. *Consider an update of cost C. Let V' be the set of nodes of degree at least 2 that are created by the update. Then* $2C \geqq \sum_{v \in V'} \deg(v)$.

Proof. This follows from the simple observation that $\sum_{v \in V'} \deg(v)$ is at most twice the number of leaves of the new subtree T''. □

We call a node v of T *big* (*with respect to* T) if depth $(v, T) < K$ and deg $(v) \geqq D^{2^{K-\text{depth}(v,T)-1}}$. Note that an update may change the status of a node from big to nonbig and vice versa by changing the depth of the node. Since $D \geqq 3$, only nodes created by updates can ever be big.

LEMMA 2. *Every insertion of cost at most K, except the first $n/4$, goes through a big node, i.e., the path of insertion contains a node which is big with respect to the current tree.*

Proof. Let $v_0, v_1, \cdots, v_l$ with $l \leqq K$ be the path of insertion. Let d_j be the degree of node v_j and let w_j be the weight of node v_j, $0 \leqq j \leqq l$. Then $w_l = 1$, since v_l is a leaf, $w_{l-1} \leqq d_{l-1} \cdot w_l$ since A plays IHS, and $w_0 \geqq n/4$. Thus $n/4 \leqq \prod_{j=0}^{l-1} d_j$ and hence $d_j \geqq D^{2^{K-j-1}}$ for some j. □

Since there are $n/2$ insertions of cost K or less, we conclude that there are $n/4$ insertions that go through a big node. For every such insertion I let $v(I)$ be the big node of largest depth on the path of insertion; we say that I is *assigned* to v. For every node v let $n(v)$ be the number of insertions assigned to v. Note that $n(v) > 0$ implies $v \in V$ and $\sum_{v \in V} n(v) \geqq n/4$.

LEMMA 3. $n(v) \leqq \deg(v)/D$ *for every vertex* $v \in V$.

Proof. Consider any vertex v. Let i be the minimal integer such that $\deg(v) \geqq D^{2^{K-i-1}}$. Consider any insertion I that is assigned to v. As in the proof of Lemma 2, let $v_0, v_1, \cdots, v_l$ with $l \leqq K$ be the path of insertion, let d_j be the degree of node v_j, and let w_j be the weight of node v_j, $1 \leqq j \leqq l$. Since $v = v(I)$ we have $v = v_{i+p}$ for some $p \geqq 0$. Also, the nodes $v_{i+p+1}, v_{i+p+2}, \cdots, v_{l-1}$ are not big at depths $i+p+1, \cdots, l-1$, respectively, and hence $d_{i+p+1} \leqq D^{2^{K-(i+p+1)-1}}, \cdots, d_{l-1} \leqq D^{2^{K-(l-1)-1}}$. Thus $w_{i+p+1} \leqq D^{2^{K-i-2}} \cdot D^{2^{K-i-3}} \cdots D^{2^0} \leqq D^{2^{K-i-1}-1} \leqq \deg(v)/D$. □

It is now easy to complete the proof of Theorem 1. We have by Lemmas 1-3

$$\begin{aligned} \textit{total update cost} &\geqq \sum_{i=1}^{n} \sum_{v \in V_i} \deg(v)/2 \\ &\geqq \sum_{v \in V} n(v) \cdot D/2 \\ &\geqq n \cdot D/8 \\ &= \Omega(n \log\log n). \end{aligned}$$

□

3. The upper bound. Player B maintains a (balanced) tree where

(1) all leaves have the same depth, and

(2) the nodes of height i (leaves have height 0) have degree at most $2 \cdot 2^{2^{(i-1)}}$ and at least degree $2^{2^{(i-1)}}$, except for the root which has degree at least 2.

Clearly, if such a tree has depth K then it has at least $2^{2^{(K-2)}}$ leaves and hence any insertion in an n-game has cost $O(\log\log n)$.

Player B updates the tree as follows. He first restores property (1) by incorporating the new subtree into its parent (see Fig. 3).

This has cost at most 5 and may create a node of degree 5. Player B then walks back to the root. When it encounters a node of degree $2 \cdot 2^{2^{(i-1)}} + 1$ at height i, it then splits the node into two nodes of degree $2^{2^{i-1}}$ and $2^{2^{i-1}} + 1$, respectively, and increases the degree of the parent node. This has cost $O(2^{2^i})$. If the root is split a new root of degree 2 is created. In this way property (2) is maintained.

We next compute the total update cost. Let s_i be the number of times a node of height i is split. Then $s_1 \leqq n$, clearly, and $s_i \leqq s_{i-1}/2^{2^{(i-1)}}$, since a split at height $i-1$ increases the degree of a node at height i by one, nodes at height i start with degree at most $2^{2^{i-1}} + 1$, and split when their degree reaches $2 \cdot 2^{2^{i-1}} + 1$. Thus $s_i \leqq n/2^{2^i - 2}$. The total update cost is

$$O\left(n + \sum_{i \leqq O(\log\log n)} s_i \cdot 2^{2^i}\right) = O(n \log\log n).$$

This proves Theorem 2.

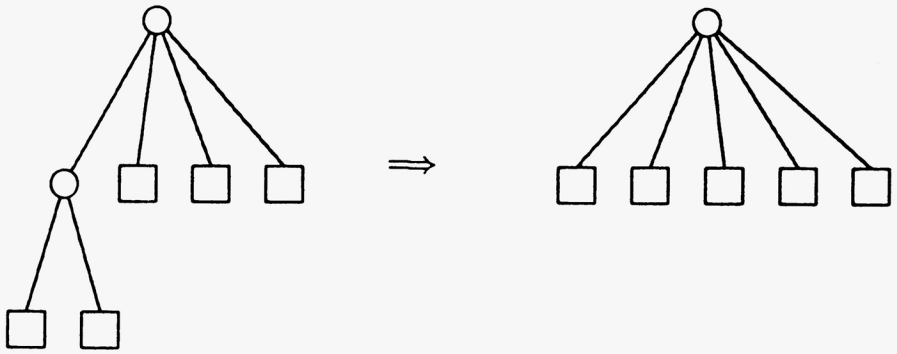

FIG. 3. *B's first move after an insertion.*

4. Conclusion. We studied the complexity of a game on trees. The game encompasses many solutions to the dictionary problem, in particular all balanced tree and multilevel hashing schemes. We have shown that in our model the amortized cost of maintaining a dictionary of size n is $\Theta(\log \log n)$.

REFERENCES

[AL] H. V. AHO AND D. LEE, *Storing a dynamic sparse table*, 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 55-60.

[DKMMRT] M. DIETZFELBINGER, A. KARLIN, K. MEHLHORN, F. MEYER AUF DER HEIDE, H. ROHNERT, AND R. TARJAN, *Upper and lower bounds for the dictionary problem*, in Proc. 29th IEEE Symposium on Foundations of Computer Science, 1988.

[FKS] M. L. FREDMAN, J. KOMLÓS, AND E. SZEMERÉDI, *Storing a sparse table with $O(1)$ worst case access time*, J. Assoc. Comput. Mach., 31 (1984), pp. 538-544.