Optimal On-line Decremental Connectivity in Trees

Stephen Alstrup* Jens Peter Secher* Maz Spork*

Abstract

Let T be a tree with n nodes from which edges are deleted interspersed with m on-line connectivity queries. Even and Shiloach gave an $O(n \log n + m)$ algorithm to process edge deletion and m queries (An On-Line Edge-Deletion problem, J. ACM, Vol. 28, Nr. 1, 1981). In this paper we present an $O(n + m)$ algorithm for the same problem.

keywords: Algorithms, trees and connectivity.

1 Introduction

Dynamic connectivity under different constraints has been investigated in several papers, see e.g. the references in [7]. In this paper we consider on-line decremental connectivity in trees. Let T be a tree with n nodes from which edges are deleted interspersed with m on-line connectivity queries. Two nodes are connected if and only if none of the edges on the unique path between them have been deleted. Even and Shiloach [1] gave an $O(n \log n + m)$ algorithm to solve this problem. In this paper we give an $O(n + m)$ algorithm for the same problem.

In section 2 we combine standard tree techniques and the result due to Even and Shiloach, in order to reduce the problem to connectivity in trees of small size. The reduced problem is solved in section 3 and finally in section 4 we briefly discuss applications of the algorithm.

2 Connectivity in trees

The algorithm in this paper is based on a *MicroMacroUniverse* which is initialized before any operations. First, we describe the universe, how to use it, and finally how to implement it. The MicroMacroUniverse is built by partitioning the set of nodes from the original tree T into disjoint subsets, where each subset induces a subtree of T called a *micro tree*. Furthermore, the division is constructed such that at most two nodes in a micro tree are incident with nodes in other micro trees. The nodes in a micro tree incident with nodes in another micro tree are henceforth denoted *boundary* nodes. The boundary nodes are inserted in a *macro tree*. The macro tree contains an edge between two nodes if, and only if, T has a path between the two nodes which does not contain any other boundary nodes. To answer connectivity for the original tree T we will maintain the MicroMacroUniverse as follows.

Algorithm 1 *If an edge removed from T belongs to a micro tree, it is removed from this micro tree. Furthermore, if the removal causes two incident boundary nodes in the macro tree to be disconnected in T, the edge between the boundary nodes in the macro tree is removed.*

*E-mail: (stephen,jpsecher,halgrim)@diku.dk. Department of Computer Science, University of Copenhagen, Universitetsparken 1, 2100 Copenhagen, Denmark. Phone: (+45) 3532 1400. Fax: (+45) 3532 1401.

Let connectivity for two nodes v, w in T, micro trees and the macro tree be denoted as $con(v, w)$, $microcon(v, w)$ and $macrocon(v, w)$ respectively. Furthermore, let $boundary(v)$ be the set of boundary nodes in the micro tree v belongs to.

Lemma 2 *Let v and w be two nodes in T and assume we have maintained a MicroMacroUniverse for T as described above; then $con(v, w)$ can be determined using a bounded number of microcon and macrocon operations.*

Proof. Assume v and w belong to different micro trees (since the lemma is otherwise trivially true). If v and w are connected in T, the path connecting them must include one of the boundary nodes in $boundary(v)$ and one in $boundary(w)$. The boundary nodes that v and w can reach can be computed using at most 4 *microcon*, since a micro tree contains at most 2 boundary nodes. Finally, using at most 4 *macrocon* we can determine if the boundary nodes are connected.□

To maintain the universe we will need an algorithm to maintain the macro tree and an algorithm to maintain the micro trees. We will use Even and Shiloach's algorithm on the macro tree and in section 3 we will present an algorithm to maintain micro trees of size $\log n$, such that each operation in a micro tree takes constant time. In order to do this we will partition the orignal tree T using the following lemma. The lemma follows from [2, 3] [1].

Lemma 3 *Let T be a tree with n nodes, where the degree of a node is at most 3. A linear time algorithm exists which partitions a tree T into $O(n/\log n)$ micro trees, where each micro tree includes at most $\log n$ nodes and two boundary nodes.* □

To use this lemma we make the following simple linear time transformation of our original tree. Let v be a node incident with the nodes $w_1, \cdots, w_d$, where $d > 3$. Substitute v with a path $v_1, \cdots v_d$ and each edge (v, w_i), $1 \leq i \leq d$, with an edge (v_i, w_i). Clearly, the tree size has become at most twice as big. Furthermore, if we let the edge (v_i, w_i) in the new tree be deleted when the edge (v, w_i) is deleted, then connectivity for v in the orignal tree can be substituted with any v_i, $1 \leq i \leq d$, in the new tree. With this we are able to give the main theorem.

Theorem 4 *For a tree T with n nodes, edge deletions interspersed with m connectivity queries can be performed using $O(n + m)$ time and $O(n)$ space, given the availability of $O(n)$ time and space for preprocessing.*

Proof. According to lemma 3, we can in linear time partition the tree into $O(n/\log n)$ micro trees, where each micro tree includes at most $\log n$ nodes. The number of nodes in the macro tree equals the number of micro tree boundary nodes. Thus, for the macro tree we can use Even and Shiloach's algorithm, such that all edge deletions in the the macro tree are performed in $O(n/\log n \log(n/\log n)) = O(n)$ time and each *macrocon* is performed in constant time. In the next section we show how the micro trees can be preprocessed in linear time such that edge deletion in micro trees and *microcon* can be performed in constant time per operation. Clearly, the micro trees can also be preprocessed in linear time, such that given a node in a micro tree, we can determine the boundary nodes in the tree in constant time. Now algorithm 1 can be realized as follows. If an edge deleted from the tree T belongs to a micro tree, we can in constant time determine if this implies that the boundary nodes in the micro tree are no longer connected using *microcon* on the boundary nodes. If this is the case, we delete the edge in the macro tree which is incident with the

[1] In [3] a micro tree is called a cluster. The external degree of a cluster is defined as the number of tree edges incident to exactly one node in the cluster. A cluster has external degree at most 3 and if the external degree of a cluster is 3 the cluster consists of only 1 node, implying that the number of boundary nodes in a cluster is at most 2.

boundary nodes. If an edge deleted from T does not belong to a micro tree, it must be an edge incident with boundary nodes from two distinct micro trees, and is therefore deleted in the macro tree. Maintaining the universe, we can, according to lemma 2, answer connectivity queries in T using a bounded number of connectivity queries in the macro tree and micro trees. □

3 Connectivity in small trees

In this section we show how to preprocess a tree of size $\log n$ in $O(\log n)$ time such that each edge deletion and connectivity query can be computed in constant time. To do this we will take advantage of the fact that any subset of the edges E can be represented in a single machine word. Let the edges be enumerated $1..|E|$. A subset of the edges can be represented in a single machine word by letting the i'th bit be 1 if the edge numerated i belongs to the set and 0 otherwise.

Let the tree be rooted in an arbitrary node r. To each node v in the tree we associate the set of edges on the path from v to r, denoted $rootpath(v)$. Since any subset of the edges can be represented in a single machine word it is possible only to use linear space and time to assign the set $rootpath$ to each node in the tree. First, the word (representing the set) for the root r, $rootpath(r)$, is initialized such that every bit is 0. Then the tree is traversed top-down and for a node v we have that $rootpath(v)$ is identical to $rootpath(parent(v))$ except that the bit numbered as the edge $(v, parent(v))$ is set to 1. Finally, let $CurrentEdges$ be the set of edges not deleted from the tree. For two nodes v and w we have that the edges included in both $rootpath(v)$ and $rootpath(w)$ are the edges on the path from the nearest common ancestor of v and w to the root r. Thus, the edges on the path between the nodes v and w are the remaining edges from $rootpath(v)$ and $rootpath(w)$.

If $A \otimes B$ denotes symmetric difference $((A - B) \cup (B - A))$, connectivity can be expressed as follows.

Lemma 5 *Let the tree be initialized as described above; then*

$$microcon(v, w) = \begin{cases} true, & \text{if } rootpath(v) \otimes rootpath(w) \subseteq CurrentEdges \\ false, & \text{otherwise} \end{cases}$$

□

Lemma 6 *In a tree of size* $\log n$ *edge deletions and connectivity queries, can be perform in constant time per operation, given the availability of* $O(\log n)$ *time and space to preprocessing.*

Proof. Since edge sets are represented in a single machine word we can in constant time perform symmetric difference, intersection, deletion and insertion on the sets using bitwise xor, and, $\neg$ (inverse the bits) operations, and by clearing/setting a bit in a word. Thus, the sets $rootpath(\cdot)$ and $CurrentEdges$ can be initialized in linear time. A deletion of an edge in the tree is performed by clearing a bit in $CurrentEdges$. To determine connectivity we use lemma 5, which is realized as $(rootpath(v)\ xor\ rootpath(w))\ and\ \neg CurrentEdges = 0$.□

4 Applications

In this section we briefly mention applications of the algorithm presented in this paper. As direct applications we have that decremental connectivity in trees is a subproblem for connectivity in graphs using randomized algorithms (see e.g. [6, 7]). With slight modification, the algorithm can also be used for the special case of UNION-FIND discussed by Gabow and Tarjan [5], which

has several applications. Since the algorithm by Gabow and Tarjan is linear, our algorithm does not represent any theoretical improvement but we do belive that our very simple approach gives improvement in practice. The technique presented in the last section could also be used to simplify the algorithm for finding nearest common ancestors in small trees which grow under addition of leaves [4].

References

[1] S. Even and Y. Shiloach. An on-line edge-deletion problem. *Journal of the Association for Computing Machinery*, 28:1–4, 1981.

[2] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Computing*, 14(4):781–798, 1985.

[3] G.N. Frederickson. Ambivalent data structures for dynamic 2–edge–connectivity and k smallest spanning tree. *SIAM J. Computing*, 26(2):484–538, 1997. See also FOCS'91.

[4] H.N. Gabow. Data structure for weighted matching and nearest common ancestors with linking. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, volume 1, pages 434–443, 1990.

[5] H.N. Gabow and R.E. Tarjan. A linear–time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985.

[6] M.R. Henzinger and V. King. Randomized dynamic graph algorithms with polylogarithmic time per operation. In *Proocedings of the 28th ACM Symposium on the Theory of Computing (STOC)*, pages 519–527, 1995.

[7] M. Thorup. Decremental dynamic connectivity. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 305–313, 1997.