Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
S. Daum, H. Ghodselahi, O. Saukh

November 26, 2013

Algorithm Theory, Winter Term 2013/14
Problem Set 4

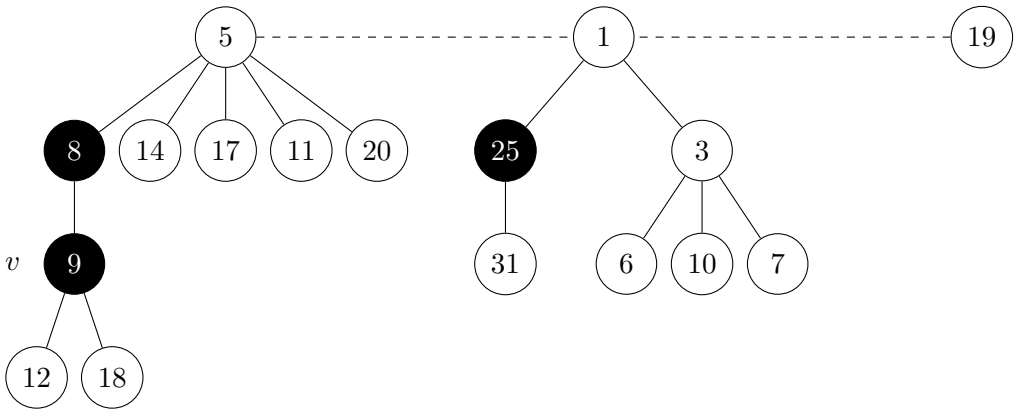
hand in by Thursday, December 5, 2013

Exercise 1: Binomial Queue

We have seen in the lecture that when using a Binomial heap to implement Dijkstra's algorithm for a graph G with n nodes and m edges, we can upper bound the total running time by $O(m \log n)$. The goal of this exercise is to show that this bound is tight for dense graphs.
Give a graph $G = (V, E)$ with positive edge weights and $m = \binom{n}{2} = \Theta(n^2)$ edges on which Dijkstra's algorithm implemented with a Binomial heap requires $\Theta(n^2 \log n)$ time.
When solving the exercise, you can assume that initially the nodes are inserted into the heap in the worst possible order. You still need to show the inefficiency of using that heap in your counter example.

Exercise 2: Fibonacci Heap

(a) Consider the following Fibonacci heap (black nodes are marked, white nodes are unmarked). How does the given Fibonacci heap look after a *decrease-key*$(v, 2)$ operation and how does it look after a subsequent *delete-min* operation?



(b) Fibonacci heaps are only efficient in an amortized sense. The time to execute a single, individual operation can be large. Show that in the worst case, both the *delete-min* and the *decrease-key* operations can require time $\Omega(n)$ (for any heap size n).

Hint: *Describe an execution in which there is a delete-min operation that requires linear time and describe an execution in which there is a decrease-key operation that requires linear time.*

Exercise 3: Union-Find

(a) Show that when implementing a union-find data structure by using disjoint-set forests with the union-by-size heuristic, the height of each tree is at most $O(\log n)$.

Hint: *Show that any subtree with k nodes has height at most $\lfloor \log_2 k \rfloor$.*

(b) Demonstrate that the above analysis is tight by giving an example execution (of merging n elements in that data structure) that creates a tree of height $\Theta(\log n)$. Can you even get a tree of height $\lfloor \log_2 n \rfloor$?

Exercise 4: Amortized Analysis

You plan to implement a hash table. Because you don't know how many keys will be inserted into the hash table and because the number of keys stored in the hash table might change over time, you want to be able to adapt the size of the hash table to the number of keys stored in the table. Your implementation should allow two operations
Assume that you already have an efficient implementation for fixed table size s. Your implementation allows new keys to be inserted efficiently as long as the load of the hash table is less than 3/4. That is, as long as the number of elements in the table is less than $\frac{3}{4}s$, your implementation guarantees that an *insert* operation (which inserts one new key) costs $O(1)$. A *delete* operation (which deletes one key) can always be done in $O(1)$ time. To adjust the table size based on the number of keys stored in the table, you use the available implementation as follows.
Initially, the hash table is empty. When the first element x is inserted into the table, you build an initial table of fixed size s_0 and insert x. Assume that the cost for doing this is $O(1)$. For simplicity, assume that $s_0 = 8$. Throughout, we will always work with one instance of the available hash table implementation for a table of sufficiently large size s (which will always be divisible by 8). Operations *insert* and *delete* are implemented as follows. Assume that s is the current table size and n is the current number of keys stored in the table.

***insert*(x):**

- If $n < \frac{3}{4}s$, x is inserted into the current table. Recall that this can be done in $O(1)$ time.
- If $n = \frac{3}{4}s$, we set up a new hash table of size $2s$ and move all items from the old table to the new larger table and insert x into the latter. We assume that the time to do this is $O(s)$.

***delete*(x):**

- If $n > \frac{1}{8}s$, x is deleted from the current table. Recall that this can be done in $O(1)$ time.
- If $n = \frac{1}{8}s$, we first delete x. If $s > 8$, we then set up a new hash table of size $s/2$ and move all items from the old table to the new smaller table. We assume that the time to do this is $O(s)$.

For simplicity, we normalize time units such that all the above operations that can be done in $O(1)$ time need time at most 1 and the operations that take $O(s)$ time need time at most s.
Show that the amortized running times of *insert* and *delete* are $O(1)$,

a) once by using the accounting method (the "bank account method") and

b) once via the potential function method.