CS 4120: Homework 3

Adam Camilli (aocamilli@wpi.edu)

April 27, 2018

Problem 1

Suppose we perform a sequence of n operations on a data structure in which the i-th operation costs i if i is an exact power of 2, and 1 otherwise. Use the accounting method to determine the amortized cost per operation and use this to get an upper bound on the actual cost of the sequence of n operations.

Let the actual cost c_i be represented as

$$c_i = \begin{cases} i, & \text{if } i \text{ is an exact power of 2} \\ 1, & \text{Otherwise} \end{cases}$$

Using the accounting method, let amortized cost $\widehat{c_i} = 3$ for all $0 \leq i \leq n$ such that

$$\sum_{i=1}^{n} \widehat{c_i} = 3n$$

The collective cost of a series of n operations can thus be said to have lower bound

$$\sum_{i=1}^{n} c_i \geq 1 + \sum_{j=0}^{\lfloor \log_2 n \rfloor} 2^j$$

and upper bound

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log_2 n \rfloor} 2^j \leq n + (2n - 1) \leq \sum_{i=1}^{n} \widehat{c_i} = 3n$$

-

Problem 2

Consider the same set-up as in the previous problem but now use the potential method to determine the amortized cost per operation and use this to get an upper bound on the actual cost of the sequence of n operations.

Let potential function $\Phi(D_i) = 2(i - 2^{\lfloor \log_2 i \rfloor})$.

Amortized cost $\widehat{c_i}$ can be therefore be expressed as twice the distance from the largest power of 2 that is less than or equal to i:

$$\widehat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

This gives us

$$\Phi(D_i) = 2(i - 2^{\lfloor \log_2 i \rfloor}) \geq 0 = \Phi(D_0) \Rightarrow \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \widehat{c_i}$$

which we expand by estimating $\widehat{c_i}$ for cases $i \neq 2^k$ and $i = 2^k$ for some positive integer k:

Case $i \neq 2^k$:

$$\begin{aligned}
\widehat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}) &= 1 + 2(i - 2^{\lfloor \log_2 i \rfloor}) - 2(i - 2^{\lfloor \log_2 i - 1 \rfloor}) \\
&= 1 + 2 + (2i - 2i) - 2(2^{\lfloor \log_2 i \rfloor} - 2^{\lfloor \log_2 i - 1 \rfloor}) \\
&= 1 + 2 \\
&= 3
\end{aligned}$$

Note that the value $(2^{\lfloor \log_2 i \rfloor} - 2^{\lfloor \log_2 i - 1 \rfloor})$ is always 0 if $i \neq 2^k$.

Case $i = 2^k$:

$$\begin{aligned}
\widehat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}) &= 2^k + 0 - 2(2^k - 1 - 2^{k-1}) \\
&= (2^k + 2^k) - 2(2^k) + 2 \\
&= 2
\end{aligned}$$

Thus,

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \widehat{c_i} \leq 3n$$

Problem 3

Professor Pinocchio claims that the height of an n-node Fibonacci heap is $O(\log_2 n)$. Show that the professor is mistaken by exhibiting, for any positive integer n, a sequence of Fibonacci-heap operations that creates a Fibonacci heap consisting of just one tree that is a linear chain of n nodes.

Let F_h be an empty Fibonacci heap. In order to prove the professor wrong, we need to create a recursive algorithm for populating the heap with a linear chain of $n-1$ nodes, and then add one more node to the chain:

Assuming $n > 2$ (otherwise trivial):

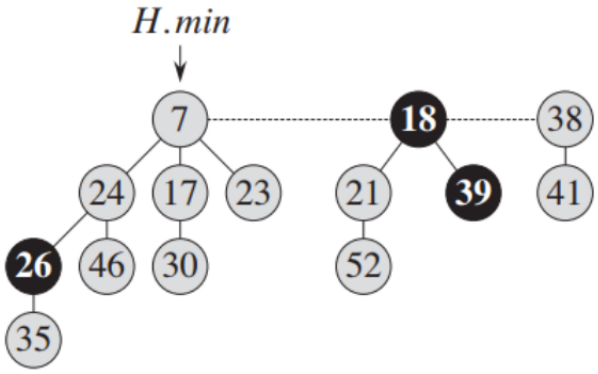
```
linearHeap(F_h, r, n)
linearHeap(F_h, r + 1, n - 1)
insert(F_h, min(F_h) + 1)
insert(F_h, min(F_h) - 1)
insert(F_h, min(F_h) - 2)
deleteMin(F_h)
b = min(F_h).secondchild
decreaseKey(b, min(F_h) - 2)
deleteMin(F_h)
```

This can be proved correct by induction, with the hypothesis that our algorithm can create a linear chain of nodes:

1. Algorithm is clearly correct for $n = 1$.
2. Assume correct for $n = k$:
3. For $n = k+1$ the algorithm creates a linear chain of k nodes (true from hypothesis) and then adds two elements: one that is greater than the minimum y, and one that is smaller x. It later adds b, which is guaranteed to be smaller than the minimum.
4. When b is deleted with `deleteMin()`, a chain of k nodes from x and y. Since x and y each have degree 0, they are consolidated.
5. A chain of two elements with root x is obtained with degree 1.
6. Thus two chains, with height 2 and height k, are consolidated.
7. By deleting x, we obtain a linear chain of $k+1$ nodes.

Problem 4

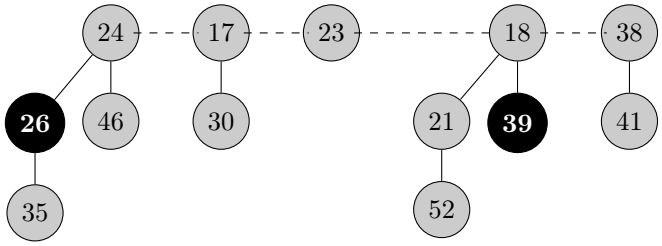
Show the resulting Fibonacci Heap (step by step) that is obtained after an EXTRACT-MIN() operation on the following Fibonacci Heap:



First, noting that root 18 does not need to be marked, we see the potential of the tree is

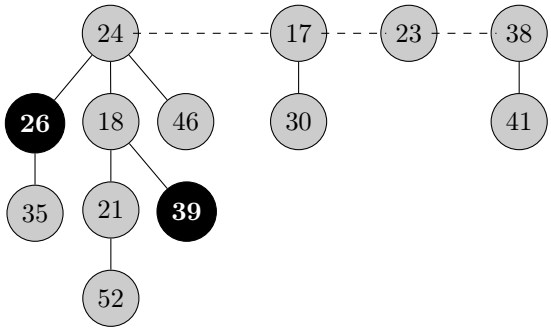
$$\Phi(H) = 3 + 2(2) = 7$$

We first remove $H.min$:

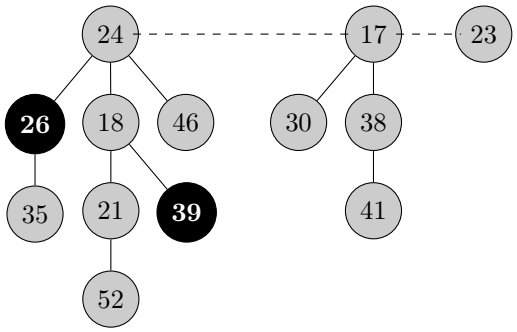


Here we there are now several roots with the same degree. To consolidate them, we meld roots with the same degree going from highest to lowest.

First meld 24 with 18:



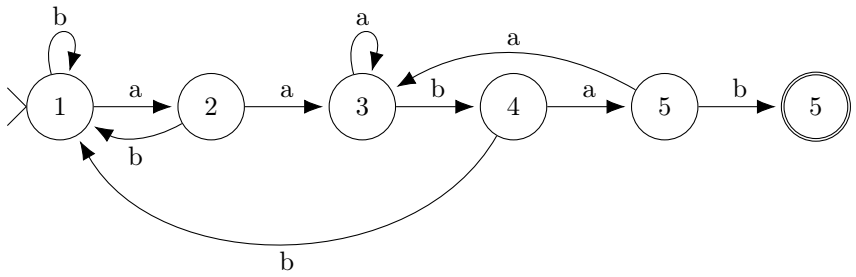
Now meld 38 with 17:



All roots have different degrees: We are done.

Problem 5

Construct the string-matching automaton for the pattern P = aabab and illustrate its operation on the text string T = aaababaabaababaab



String	Transition	Matches
a	$1 \to 2$	0
aa	$2 \to 3$	0
aaa	$3 \to 3$	0
aaab	$3 \to 4$	0
aaaba	$4 \to 5$	0
aaabab	$5 \to 6$	$0 \to 1$
a**aabab**a	$1 \to 2$	1
a**aabab**aa	$2 \to 3$	1
a**aabab**aab	$3 \to 4$	1
a**aabab**aaba	$4 \to 5$	1
a**aabab**aabaa	$5 \to 3$	1
a**aabab**aabaab	$3 \to 4$	1
a**aabab**aabaaba	$4 \to 5$	1
a**aabab**aabaabab	$5 \to 6$	$1 \to 2$
a**aabab**aab**aabab**a	$1 \to 2$	2
a**aabab**aab**aabab**aa	$2 \to 3$	2
a**aabab**aab**aabab**aab	$3 \to 4$	2

Problem 6

The longest-cycle problem is the problem of determining a cycle of maximum length in a graph G. Formulate a related decision problem, and show that the decision problem is NP-complete. You may use any of the NP-complete problems learned in class for your reduction.

Generally speaking, NP-Completeness applies to decision problems, or problems that can be formed as a yes or no question of a set of input values. NP-Completeness of a given decision is most often shown via reduction of a known, more generalized version of the problem.

For example, let a SIMPLE-CYCLE to be a cycle in a graph G in which no vertices are repeated. It can be shown via reduction of the well-known Hamiltonian cycle problem that LONGEST-SIMPLE-CYCLES is also NP-Complete:

1. Define the decision problem LONGESTSIMPLECYCLE as:

   LONGESTSIMPLECYCLE(G, k): Given an undirected graph G and integer k, does G have a simple cycle of length $\leq k$?

2. Define the decision problem HAMCYCLE as:

   HAMCYCLE(H): Given an undirected graph H, does H contain a Hamiltonian cycle?

3. LONGESTSIMPLECYCLE$(G, k) \in$ NP:
   (a) Let (G, k) be an instance of LONGESTSIMPLECYCLE.
   (b) Given a certificate c, we can traverse G in polynomial time to verify that:
      i. c is a cycle.
      ii. No vertex in c appears more than once.
      iii. c is of length $\leq k$.

4. HAMCYCLE$(H) \leq_p$ LONGESTSIMPLECYCLE(G, k):
   (a) From an instance of HAMCYCLE(H), we can construct an instance of LONGESTSIMPLECYCLE $(H, |V|)$ in polynomial time.
   (b) We claim graph $H(V, E)$ is a Hamiltonian cycle iff the length of its longest simple cycle is equal to $|V|$. This is correct:
      i. If $H(V, E)$ has a Hamiltonian cycle, that cycle is a simple cycle of length $|V|$.
      ii. Otherwise, the length of the longest simple cycle must be strictly less than $|V|$.

5. From the above proofs, we can conclude LONGESTSIMPLECYCLE is NP-complete.

Problem 7

An independent set of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that each edge in E is incident on at most one vertex in V'. The independent-set problem is to find a maximum-size independent set in G.

(a) Formulate a related decision problem for the independent-set problem, and prove that it is NP-complete. (Hint: Reduce from the clique problem.)

A decision problem for the independent-set problem can be formulated as

$$\text{INDEPENDENT-SET} = \{< G, K >: G \text{ has independent set of size} \leq K\}$$

Let the certificate be a set of nodes with size $\leq K$. It is trivial to verify the certificate. In order to show the independent set is NP-complete, the CLIQUE problem can be reduced:

1. Given an instance $< G, K >$ of the CLIQUE problem, take G', the complement of G.
2. G has a CLIQUE of size K iff G' has an independent set of size K.
3. Therefore,
$$\text{CLIQUE} \leq p \cdot \text{INDEPENDENT-SET}$$

(b) Suppose that you are given a "black-box" subroutine to solve the decision problem you defined in part (a). Give an algorithm to find an independent set of maximum size. The running time of your algorithm should be polynomial in $|V|$ and $|E|$, counting queries to the black box as a single step.

1. Run subroutine for independent-set problem on $< G, K >$ for $K = 1 \ldots n$.
2. Find largest $K = K_0$ such that subroutine returns true.
3. Let V be a vertex in G. Let $G(V')$ be the sub-graph of G constructed by removing V and its induced edges.
4. Run subroutine on $(G(V'), K_0)$.
5. If subroutine returns true, recursively find an independent set of size $K_0 - 1$ in $G(V')$.

With queries to black box taking $O(1)$, this algorithm is clearly in polynomial time.