CS787 Homework 3

Mikola Lysenko

March 5, 2009

1. We start by giving the following procedure for incrementing an arbitrary sized binary counter, C, in place:

```
INCREMENT-COUNTER($C$)
1  $i \leftarrow 1$
2  while $C_i = 1$
3      do $C_i \leftarrow 0$
4         $i \leftarrow i + 1$
5  $C_i \leftarrow 0$
```

Observe that adding one to the counter increases the total number of 1's by at most one and that each carry removes exactly one '1' from the counter, so the total number of carries performed must be bounded by the total number of increments. Next, we fix the constant cost a for executing a carry and the cost b for executing the rest of the procedure. Therefore, a charge of $a + b$ per call to INCREMENT-COUNTER is sufficient to amortize cost over $O(n)$ iterations.

2. From class we know that for any subtree of a Fibonacci heap rooted at v, $size(v)$ is exponential in $rank(v)$, where $rank(v)$ is the degree at v and $size(v)$ is the size of the tree.[1] More formally, $size(v) \in \Theta(2^{rank(v)})$ which implies that $rank(v) \in \Theta(\log size(v))$. By definition $rank(v) \in \Theta(height(v))$, so $height(v) \in O(\log size(v))$ for all subtrees, and thus height of the entire Fibonacci heap can never be greater than the log of the total number of elements.

3. Here is a version of Dijkstra's SSP algorithm using a Fibonacci heap:

```
DIJKSTRA-PATH($G, w, s$)
 1  $H \leftarrow$ CREATE-FIBONACCI-HEAP($\langle 0, s \rangle$)
 2  $Pred[...] \leftarrow$ UNREACHABLE
 3  $Cost[...] \leftarrow \infty$
 4  while $H$ is not empty
 5      do $\langle c, v \rangle \leftarrow$ EXTRACT-MIN($H$)
 6         for each $\langle n, e \rangle \in Adj(v)$ and $c + w(e) < Cost[n]$
 7             do $Cost[n] \leftarrow c + w(e)$
 8                if $Pred[n] =$ UNREACHABLE
 9                   then INSERT($H, \langle Cost[n], n \rangle$)
10                   else DECREASE-KEY($H, \langle Cost[n], n \rangle$)
11                $Pred[n] \leftarrow e$
12  return $Pred[...]$
```

[1]This result may also be found in the lecture notes from 2-19-2009

Where $G = (V, E)$ is some graph, $w : E \rightarrow \Re^+$ is some weight function and $s \in V$ is the source. We return a set of indexed directed edges pointing toward each vertex (which is sufficient to reconstruct any path given a destination.) Now we make the following observations:

- i Extract-Min is called no more than $O(|V|)$ times.
- ii Insert and Decrease-Key are called no more than $O(|E|)$ times.
- iii The size of the heap is at most $O(|V|)$.
- iv The cost of the remainder of the procedure is bounded by the cost of calling the above 3 functions.

From class we have proven that the cost of both Insert and Decrease-Key are $O(1)$ for a Fibonacci heap, so the cost of the second item is bounded by $O(|E|)$. We also showed that the cost of Extract-Min is bounded by $O(\log n)$, with n being the size of the heap which is in $O(|V|)$ in this case. Therefore, the amortized cost of this algorithm is within $O(|E| + |V| \log |V|)$.

4. Pick a graph $G = (V, E)$ and weight function $w : E \rightarrow \Re^+$. Here is a version of Prim's algorithm using the Fibonacci heap:

```
Prim-MST($G, w$)
 1  $Pred[...] \leftarrow$ null
 2  $Cost[...] \leftarrow \infty$
 3  $Visited \leftarrow \emptyset$
 4  $T \leftarrow \emptyset$
 5  while $|Visited| < |V|$
 6          do $r \leftarrow$ random vertex in $V - Visited$
 7             $Cost[v] \leftarrow 0$
 8             $H \leftarrow$ Create-Fibonacci-Heap($\langle Cost[v], v \rangle$)
 9             while $H$ is not empty
10                  do $\langle c, v \rangle \leftarrow$ Extract-Min($H$)
11                     if $v \in Visited$ continue
12                     $Visited \leftarrow Visited \cup \{v\}$
13                     if $Pred[n] \neq$ null $T \leftarrow T \cup \{\langle Pred[v], v \rangle\}$
14                     for each $\langle n, e \rangle \in Adj(v)$ and $w(e) < Cost[n]$ and $n \notin Visited$
15                          do if $Pred[n] =$ null
16                                then Insert($H, \langle w(e), n \rangle$)
17                                else  Decrease-Key($H, \langle w(e), n \rangle$)
18                             $Pred[n] \leftarrow v$
19                             $Cost[n] \leftarrow w(e)$
20  return $T$
```

As we have laboriously shown in class, the set of partial Minimum Spanning Trees forms a matroid. Notice that the edge added to T in line 13 does not violate the MST property for T and is minimum cost. Because the procedure is executed for each vertex, the final tree is also maximal. Therefore, the above algorithm returns a minimum cost spanning tree.

We start by analyzing the inner loop which computes the MST for some connected component of $C = (V', E') \subseteq G$. By an argument analogous to the one used for Dijkstra's algorithm, we know that the cost of executing this loop is at most $O(|E'| + |V'| \log |V'|)$. To compute the total cost, we now consider the sum over all connected components, $C_1, C_2, ...C_n \subseteq G$:

$$T(G) \leq a(|E_1| + ... + |E_n|) + b(|V_1| \log |V_1| + ... + |V_n| \log |V_n|)$$

With a, b arbitrary constants. Because the sets $C_1, ...C_n$ partition v and are non-empty, we have the following relations:

$$\begin{aligned}
|E_1| + ... + |E_n| &\leq |E| \\
|V_1| + ... + |V_n| &\leq |V| \\
\log |V_i| &\leq \log |V|
\end{aligned}$$

Substituting these into the previous equation gives a total time complexity in $O(|E| + |V| \log |V|)$.

5. Let A, B be boolean matrices and let M be a square matrix such that:

$$M = \begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix}$$

With boolean algebra it is easy to verify that the following identities hold:

$$\begin{aligned}
M^2 &= \begin{pmatrix} I & A & AB \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix} \\
M * M^2 &= M^2 \\
M^2 + M + I &= M^2 \\
M^2 + M^2 &= M^2
\end{aligned}$$

Note that M defines a reflexive binary relation. Let M^* denote the transitive closure of M. By definition:

$$M^* = I + M + M^2 + ... + M^n$$

However, according by the previous observations this sum reduces to M^2, so we conclude that $M^* = M^2$. Notice that M^2 contains the product AB, and its construction costs $O(|A| + |B|)$ which is also the size of the input. Therefore, we have reduced the boolean matrix product of A, B to the cost of computing M^*.