Graduate Algorithms

First Problem Set

Virginia Vassilevska, Elisabeth Crawford

01/18/04

Problem 1

Let $q = \lfloor log_2\ p \rfloor$.
The paper "Time Bounds for Selection" describes an algorithm A which computes the i-th smallest number in a given set of size n in time $O(n)$. Recursively use this algorithm as follows:

```
Subdivide($S$, level)

1. if level=$q$, return $S$
2. else find median $m$ using algorithm $A$
3. go through $S$ once and let $S_< = \{s \in S|\ s < m\}$, $S_= = \{s \in S|\ s = m\}$,
   $S_> = S \backslash \{S_< \cup S_=\}$ (multisets)
4. Move $\lfloor |S|/2 \rfloor - |S_<|$ of the elements in $S_=$ to $S_<$ and the rest to $S_>$.
5. return (Subdivide($S_<$, $q+1$), Subdivide($S_>$, $q+1$))
```

Subdivide(S, 0) returns a partitioning of S into $p' = 2^q$ sets T_i each of size $\lfloor n/p' \rfloor$ or $\lceil n/p' \rceil$, and such that if $i < j$, then $\forall s \in T_i$ and $t \in T_j, s \leq t$. The number of steps performed at level $L < q$ is $O(2^L(1 + n/2^L) = O(2^L + n)$ since there are 2^L sets of size at most $n/2^L + 1$ at level L and steps 2, 3 and 4 take time linear in the size of the set. There are q levels $L < q$ and Subdivide(S, q) takes $O(1)$ time. Thus the entire subdivision takes $O(qn + \sum_{L=0}^{q-1} 2^L) = O(qn) = O(n\ log(p))$.

Suppose $n = \eta$ $(mod\ p)$. Now we can sequentially find the n/p-th element in each of the T_i as follows:

```
1.  Let $k = 1$, $N = \lceil n/p \rceil$
2.  for $i$ from 1 to $p'$
3.      using algorithm $A$ find the $N$-th element $m$ in $T_i$
4.      go through $T_i$ and split into $T_{i1} = \{t \in T_i | t < m\}$, $T_= = \{t \in T_i |\ t = m\}$
        and $T_{i2} = T_i \backslash \{T_{i1} \cup T_=\}$
5.      Move $N - |T_{i1}|$ of the elements in $T_=$ to $T_{i1}$ and the rest to $T_{i2}$
6.      let $P_k = T_{i1}$, $k = k + 1$
7.      if $i = \eta$, $N = \lfloor n/p \rfloor$
8.      if $|T_{i2}| < N$ let $T_{i+1} = T_{i+1} \cup T_{i2}$
9.      else using $A$ find the $N$-th element $m'$ in $T_i$
10.         split $T_{i2}$ into $T_{i21} = \{t \in T_{i2} |\ t < m'\}$, $T_{i23} = \{t \in T_{i2} |\ t = m'\}$
            and $T_{i22} = T_{i2} \backslash \{T_{i21} \cup T_{i23}\}$
11.         Move $N - |T_{i21}|$ of the elements in $T_{i23}$ into $T_{i21}$ and the rest into $T_{i22}$.
12.         let $P_k = T_{i21}$, $k = k + 1$
13.         let $T_{i+1} = T_{i+1} \cup T_{i22}$
```

Since each T_i had size at most $n/p' + 1 < 2n/p + 1$, and since at most n/p elements from T_{i-1} are added to T_i before going through it, at most 3 searches for an N-th element are done in each T_i. Hence the split of each (modified) T_i is done in at most 3 $(3n/p + 1)$ steps. Since there are at most p T_i, the entire final split is done in $O(n + p) = O(n)$ steps. Overall, partitioning the original set takes $O(n\ log\ p)$ time. Since the last partitioning is done using ceilings in the first η searches, and floors in the rest, the partitioning will work exactly as asked: $n = p\ \lfloor n/p \rfloor + \eta = \eta \lceil n/p \rceil + (p - \eta) \lfloor n/p \rfloor$. The resulting partitions P_k for $k = 1, \ldots, p$, will contain η partitions of size $\lceil n/p \rceil$ and $p - \eta$ partitions of size $\lfloor n/p \rfloor$.

Problem 2

Consider the following structure S_{V_i} storing a set V_i: S_{V_i} has a head storing the index i, and the size of V_i, having a child pointer to a node representing some element of V_i. Think of having arranged the elements of V_i on a straight line thus giving them some order. The head points to the first element in that order, and each element points to the next element in that order, and to the head of the list. The "find" operation corresponds to checking the index stored in the head which takes one step since there is a direct pointer from each element to the head. The "merge" operation corresponds to:

- comparing the sizes of V_i and V_j by checking their heads,
- adding the size of the smaller one V_i to the size of V_j stored in the head of V_j,
- making each element of V_i point to the head of V_j (by following the child pointers starting from the head of V_i and changing the head pointer of each element),
- making the last element of V_i (which points to NULL) now point to the first element in V_j,
- making the head of V_j point to the first element of V_i and
- deleting the original head of V_i.

The above clearly takes $O(min\ \{|V_i|, |V_j|\})$ time.

Problem 3

a

i) Any cycle in B would also be a cycle in A. Therefore B must be acyclic and so $B \in I(G)$.

ii) Let $A \in I(G)$ and $B \in I(G)$, s.t. $|A| < |B|$. Say that a vertex v is in A if there is an edge in A s.t. one of its end points is v. Now suppose that $\forall x \in BA \cup \{x\} \notin I(G)$. Then each $x \in B$ is either in A or creates a cycle in A (i.e. there is a path in A between its end points). In particular its end points are vertices in A. And so the number of vertices in A is at least the number of vertices in B. Then if two edges x and y are in the same connected component in B, then their end points are in the same connected component in A. Therefore the number of connected components in B is at least the number of connected components in A. Hence the number of edges in B = number of vertices in B - number of components in B, which is at most number of vertices in A - number of components in A = number of edges in A. Contradiction since $|B| > |A|$. Therefore there exists an $x \in B$ such that $A \cup \{x\} \in I(G)$.

b

i) Clearly, $\emptyset \in I(M)$. So let $A = \{a_i\} \in I(M)$ and $B \subseteq A$, $B \neq \emptyset$, and F be the field. Suppose $\sum_{i:\ b_i \in B} \beta_i\ b_i = 0$ for some choice of $\beta_i \in F$ at least one of which is nonzero. Then for each $a_i \in A$ let $\alpha_i = \beta_j$ for $a_i = b_j \in B$ and $\alpha_i = 0$ for $a_i \notin B$. Then clearly $\sum_{i:\ a_i \in A} \alpha_i a_i = 0$ and at least one $\alpha_i \neq 0$. So $A \notin I(M)$ and we have a contradiction. So any subset of A is in $I(M)$.

ii) Let $A, B \in I(M)$, such that $|A| < |B|$. Then need to show that $\exists x \in B$ such that $A \cup \{x\} \in I(M)$. Suppose this is not so. Then if $L(A)$ is the linear span of

A, then $\forall x \in B\ x \in L(A)$. Suppose $|A| = k$. Take any $k+1$ subset of B, call it B'. We'll show by an induction on k that B' is dependent. If $k = 1$, then $L(A)$ consists of scalar multiples of the unique $a \in A$. So take any two $b_1, b_2 \in B$, $b_1 = \beta_1 a$, $b_2 = \beta_2 a$, $\beta_1, \beta_2 \neq 0$. These are clearly linearly dependent and so by i) $B \notin I(M)$.
Now suppose the assertion holds for all A with $|A| = k-1$ and B with $|B| > |A|$. Consider $A, B \in I(M)$ such that $k = |A| < |B|$ and $B' \subset B$ with $|B'| = k+1$. For each $b_j \in B'$ consider the unique representation of b_j as a linear combination of vectors in A: $b_j = \sum_i \alpha_{ij}\ a_i$. Examine all scalars α_{1j}. We have two cases:

1. $\alpha_{1j} = 0\ \forall j$. Then each $b_j \in B'$ is in the linear span of $\{a_2, \dots, a_k\}$. By the induction hypothesis it follows that B is dependent and so $B \notin I(M)$.
2. Without loss of generality $\alpha_{11} \neq 0$. Let $q_j = \alpha_{1j}\alpha_{11}^{-1}$. Then $q_j b_1 = \alpha_{1j} a_1 + \sum_{i=2}^{k} q_j \alpha_{i1} a_i$. Then $q_j b_1 - b_j = \sum_{i=2}^{k} (q_j \alpha_{i1} - \alpha_{ij}) a_i$ for all k elements $b_j \in B'$, $j > 1$ and so by the induction hypothesis applied to $A' = \{a_2, \dots, a_k\}$ and $B'' = \{q_j b_1 - b_j \mid j = 2, \dots, k+1\}$ it follows that B'' is dependent. This means there are some $\gamma_j \in F$ such that $\sum_{j=2}^{k+1} \gamma_j (q_j b_1 - b_j) = 0$, and so $(\sum_{j=2}^{k+1} \gamma_j q_j) b_1 - \sum_{j=2}^{k+1} \gamma_j b_j = 0$. Therefore B' and thus B is dependent.

Hence we get a contradiction for any k and so if $A, B \in I(M)$ such that $|A| < |B|$, then there is an $x \in B$ so that $A \cup \{x\} \in I(M)$.

c

YES. Consider the incidence matrix M of the graph G, which is $n \times m$ ($n = |V(G)|$, $m = |E(G)|$) and $m_{ij} = 1$ if vertex i is an end point of edge j, and $m_{ij} = 0$ otherwise. The field is $GF(2)$. Then there is a clear bijection between edges j and columns j. Suppose now there is a subset of columns of M, say A, and that $\sum_{a \in A} a = 0$ (where the sum is in $GF(2)$ and the 0 is the zero column; also note that any dependent set of columns of M contains such a subset A). In order for this to hold, given any column $m_j \in A$ with $m_{i_1 j} = m_{i_2 j} = 1$ and $m_{ij} = 0\ \forall i \neq i_1, i_2$, one can find another two columns $m_{l_1}, m_{l_2} \in A$ such that $m_{i_1 l_1} = 1$ and $m_{i_2 l_2} = 1$. Since A is finite, there is a circular sequence of columns $m_{j_1}, m_{j_2}, \dots, m_{j_{p-1}}, m_{j_p} = m_{j_1}$ where each two consecutive columns agree in a 1-entry. This immediately gives a cycle in G defined by those edges, and thus the edge set of G corresponding to A is dependent. Therefore, if $S \in I(G)$, then necessarily the corresponding set of columns $b(S) \in I(M)$.

Problem 4

1. The data structures and the algorithm

First sort the edges of G with respect to weight and store edge structures representing them in a queue Q with the smallest edge in the front. Each edge structure will contain two pointers - one to each of the end points of the edge. The vertices will be stored in component structures, so that two vertices are in the same structure if and only if they are in the same connected component. The component structures are the structures described in problem 2. We will also keep a bit array A in which we will store the state of each component during a particular stage. The edges of the minimum spanning tree will be stored in a queue T.

Using these data structures, the algorithm can be done as follows: First initialize the queue Q with the sorted edges, and set T to the empty queue.
At each stage, go through the current edge queue Q:

- If the queue is empty, then return T.
- Set all bits in A to be 0.
- For the current edge e from the queue Q
  - If both end points of the current edge e are in the same component, then delete e from the queue and consider the next edge.
  - If the end points of e are in different components, and the bits in A corresponding to those components are both 1, then consider the next edge (leaving this one for the next stage)
  - If the end points of e are in different components and at least one of the bits is 0, then merge the two components, change the bit corresponding to the new index to a 1, delete e from Q and add it to T and consider the next edge.
- Once you've gone once through the queue, go to the next stage starting from the beginning of the modified queue.

2. The algorithm is correct

Notice that since the edges are sorted by weight, if an edge is added between two components, then it is the first one to be seen in the edge queue going out of (at least) one of these components (ensured by the bit array A). Therefore it is of the smallest weight for that component, and the algorithm does what is described in the problem. The algorithm terminates since at each step Q gets smaller: it ends when Q is empty. Since at each step two components are connected by adding a single edge, no cycles can ever be created. Since the algorithm terminates when all possible edges are added without creating a cycle (the queue is empty in the end so all edges are considered, and only duplicate component links are deleted), and since the graph was originally connected, in the end everything must be in the same connected component and the result is a spanning tree. Since the edges between each two connected components were chosen w.r.t. minimality of weight, the spanning tree must be minimum.

3. Number of stages

At each point the number of components is at least halved since each component is merged with at least one of the other ones. Initially there were n components and in the end there is 1 component. Therefore there are at most $log_2 n$ stages.

4. Complexity

Consider one stage of the algorithm. First all bits of A are set to 0. This takes $O(n)$ as there are n bits. Each merge takes O(size of the smaller component). We know that in each merge at least one of the components has not been touched yet. Thus we can charge the merge to the untouched component (where we are possibly overcounting because this may be the larger one). Thus all merges together take $O(n)$ time since the sum of the sizes of the components is n. The rest of the algorithm consists of doing a constant number of $O(1)$ operations for each edge in the queue which together takes at most $O(m)$ time. Hence each stage takes $O(m)$ time (as $n = O(m)$ since the graph is connected). Since there are at most $log_2\ n$ stages, the algorithm running time is $O(m\ log\ n)$.