BU CLA CS 530: Analysis of Algorithms
Solutions for End-of-Term Test

Problem 1 (5 points).

Increasing all weights by the same positive constant c does not change a MST, because only the relative order of the weights is important in determining a MST — think, for example, of how Kruskal's and Prim's algorithms work. Whenever weights are used in these algorithms, they are only compared to each other for their relative magnitudes; they are not used in any other way. Hence, because adding the same constant to all the weights does not change their order, it does not change the final result.

Problem 2 (10 points).

(a) (5 pts) We assign one processor to each item in the list, for a total of n processors. The behavior of each processor is determined by whether or not its item is X. If a processor's item is X, then this processor checks whether it has the highest index among the items in the list which are equal to X, by examining the next item in the list; the processor with the highest index whose item is X (there can be at most one such processor) outputs its index. If a processor's item is not X, then this processor attempts to detect whether X does not appear in the list at all, by checking if its own item is smaller than X and the next one is larger than X; if it determines that this is indeed the case (there can be at most one such processor), it outputs *no*.

(b) (5 pts) This is a parallel constant-time algorithm that uses n processors. The running time is therefore $\mathcal{O}(1)$. The *work* performed by this algorithm is $\mathcal{O}(n)$, which is the product of its running time and the number of processors it uses. This algorithm is therefore *not* work-efficient, because there are sequential algorithms which solve the problem in time $\mathcal{O}(\log n)$.

(There are other parallel algorithms for this problem, which are more efficient, i.e. perform less *work*, but are also more complicated to design. If you are interested, an appropriate reference is Chapter 5 of the book [A].)

Problem 3 (10 points).

(a) (2 pts) The certificate we choose is a simple path π of negative weight from s to t. We can verify that π has indeed negative weight by traversing each of the edges of π, adding every new weight to the sum of all previous weights. The total time is proportional to the length of π, which is polynomial time.

(b) (8 pts) We reduce the directed hamiltonian cycle problem to the present problem. An instance of the directed hamiltonian cycle problem is a directed graph $G = (V, E)$. Let $|V| = n$. Pick a vertex $u \in V$ and split it into two new vertices v and t with no edge between. Each of v and t is connected to the other vertices of G in the same way u was. Let s be a new vertex and add a single edge of weight $n-1$ from s to v. Assign weight -1 to every other edge. The resulting weighted graph is $G' = (V', E')$, where $V' = (V - \{u\}) \cup \{s, v, t\}$ has exactly $n+2$ vertices, and

$$\begin{aligned} E' \;\; &= \;\; (\; E - (\{u\} \times V) - (V \times \{u\}) \;) \\ &\quad \cup \{\; (v,x),\; (t,x) \mid (u,x) \in E \;\} \;\cup\; \{\; (x,v),\; (x,t) \mid (x,u) \in E \;\} \;\cup\; \{\; (s,v) \;\} \end{aligned}$$

Any path from s to t must begin with the edge from s to v of weight $n-1$ and, therefore, must traverse at least n other edges in order to be of negative weight. In order to be simple, this path can traverse at most n other edges — to connect $n+2$ vertices with a simple path you need $n+1$ edges. Hence, a simple negative-weight path from s to t is of length exactly $n+1$ and contains a simple path from v to t of length n. This gives rise to a hamiltonian cycle in the original graph G, i.e. G has a hamiltonian cycle iff there is simple negative-weight path from s to t in G'.

Problem 4 (10 points).

(a) (5 pts) If the answer returned is *yes*, then the randomly chosen x is a majority element, and hence there is a majority element in A. On the other hand, if the answer returned is *no*, it is still possible that A contains a majority element $a \neq x$. If A does indeed contain a majority element a, then the probability that the randomly chosen x is a minority element, i.e. the probability that $x \neq a$, is less than $1/2$ — because majority elements occupy more than half the locations in A. In conclusion,

If A has a majority element, MAJOR returns *yes* with probability $> 1/2$.

If A has no majority element, MAJOR returns *no* with probability 1 (i.e. with certainty).

Hence, in all cases, the error probability $p < 1/2$.

(b) (5 pts) If A does not have a majority element, then each of the two calls to MAJOR is certain to return *no*, so that MAJOR$'$ also returns *no*. If A does have a majority element, the probability that the first call to MAJOR returns *yes* is $q > 1/2$, and in this case MAJOR$'$ also returns *yes*. On the other hand, if the first call to MAJOR returns *no*, which happens with probability $1-q$, the second call to MAJOR may still return *yes* with probability q, in which case MAJOR$'$ also returns *yes*. Hence, the probability that MAJOR$'$ returns *yes* when A contains a majority element is:

$$q + (1-q) \cdot q \;\; = \;\; 1 - (1-q)^2 \;\; > \;\; 3/4$$

Hence, in all cases, the error probability $p' < 1/4$.

Problem 5 (15 points).

(a) (5 pts) First, sort the input set S in order of non-decreasing sizes, say $s_1 \leq s_2 \leq \cdots \leq s_n$, in time $\mathcal{O}(n \log n)$. Second, by packing the objects into the single unit-size bin in order of non-decreasing sizes, we obtain an optimal solution T, i.e. a subset T of S containing the largest number of objects that can be packed into a single bin. The second phase takes time $\mathcal{O}(n)$, and the two phase together $\mathcal{O}(n \log n)$.

(b) (5 pts) This is a variation on the solution for part (a) of Problem 4 in Set 13. We write an algorithm for **partition** which uses as a subroutine an algorithm k-BIN-PACKING for k-**bin-packing**. Our solution is good for every $k \geq 2$.

PARTITION(S)
1. $s \leftarrow \sum S$
2. **if** $|S| = 1$ **or** s is odd **or** $\exists a \in S$ such that $2a > s$
3.   **then return** *no*
4.   **else if** $\exists a \in S$ such that $2a = s$
5.   **then return** *yes*
6.   **else** $S' \leftarrow \{\ 2a/s \mid a \in S\}$
7. $b \leftarrow$ a number such that $0 < b < 1$ and $b + x > 1$ for every $x \in S'$
8. $S'' \leftarrow S' \cup \{\ (k-2)$ copies of $b\ \}$
9.   **if** $S'' = k$-BIN-PACKING(S'')
10.   **then return** *yes*
11.   **else return** *no*

The difference with the algorithm in Problem 4 of Set 13 is in lines 7 to 11. S, S', and S'' are multiset of numbers. In line 7, we set b to a value large enough so that each copy of b must be placed in a unit-size bin by itself. When $k = 2$, $S'' = S'$ in line 8. The correctness of the algorithm is shown as in Problem 4 in Set 13.

(c) (5 pts) Consider the case for $k = 2$, the case for $k \geq 3$ being an obvious generalization. As in part (a), we start by sorting the input set S in order of non-decreasing sizes: $s_1 \leq s_2 \leq \cdots \leq s_n$, in time $\mathcal{O}(n \log n)$. Second, we start filling the first bin B_1 so long as enough space remains in it. Third, start filling the second bin B_2 with the remaining objects, still sorted in order of non-decreasing sizes. Call the resulting algorithm 2-BIN-PACK, which clearly runs in time $\mathcal{O}(n \log n)$.

We claim that 2-BIN-PACK is an approximation algorithm with the desired property. To see why, let $T^* \subseteq S$ be an optimal solution and $T =$ 2-BIN-PACK(S). Consider our strategy of placing objects in order of non-decreasing sizes, but now when we have a single bin $\widetilde{B}$ of size $= 2$, in which case we get an optimal number p of objects stored in $\widetilde{B}$. Clearly, $p \geq |T^*|$ and $s_1 + \cdots + s_p \leq 2$. Let j be the largest index such that $s_1 + \cdots + s_j \leq 1$. Then $j \leq p$ and 2-BIN-PACK places the first j objects in B_1. Moreover,

$$s_{j+1} + \cdots + s_{p-1} \quad \leq \quad s_{j+2} + \cdots + s_p \quad \leq \quad 1$$

Hence, 2-BIN-PACK places at least $s_{j+1}, s_{j+2}, \ldots, s_{p-1}$ in B_2, and the only object that may be excluded by 2-BIN-PACK is s_p (compared with the optimal strategy packing items into a single bin $\widetilde{B}$ of size 2). So that, finally, $|T| \geq p - 1$ and $|T^*| - |T| \leq 1$.

Assaf Kfoury
Created: 95.05.08