Solutions to Midterm 1 for CS 170

Problem 1. [Divide and conquer] (30 points)

Suppose there are three alternatives for dividing a problem of size n into subproblems of smaller size: if you solve 3 subproblems of size $\frac{n}{2}$, then the cost for combining the solutions of the subproblems to obtain a solution for the original problem is $\Theta(n^2\sqrt{n})$; if you solve 4 subproblems of size $\frac{n}{2}$, then the cost for combining the solutions is $\Theta(n^2)$; if you solve 5 subproblems of size $\frac{n}{2}$, then the cost for combining the solutions is $\Theta(n \log n)$. Which alternative do you prefer and why?

Answer: The first recurrence is $T(n) = 3T(\frac{n}{2}) + \Theta(n^{2.5})$; since $\log_2 3 < 2.5$, the master theorem tells us that $T(n) = \Theta(n^{2.5})$. The second recurrence is $T(n) = 4T(\frac{n}{2}) + \Theta(n^2)$; since $\log_2 4 = 2$, the master theorem says $T(n) = \Theta(n^2 \log n)$. The third recurrence is $T(n) = 5T(\frac{n}{2}) + \Theta(n \log n)$; since $\log_2 5 > 2$, the master theorem says $T(n) = \Theta(n^{\log_2 5})$. The second alternative is the best.

Problem 2. [Lower bounds] (30 points)

Consider the following problem: given an array $A[1..n]$ of distinct integers, and a number $1 \leq k \leq n$, find any one of the k largest elements in A. For example, if $k = 2$, it is ok to return the largest or second largest integer in A, without knowing if the return value is the largest or if it is the second largest array element.

(a) Give an algorithm that solves this problem using no more that $n - k$ comparisons of array elements.

(b) Argue that every algorithm that solves this problem must, in the worst case, perform at least $n - k$ comparisons.

Answer:

(a) $x := A[1]$; for $i = 2$ to $n - k + 1$ do if $A[i] > x$ then $x := A[i]$ end; return x.

(b) Suppose an algorithm performs only $n - k - 1$ comparisons. Then at the end, there are at least $k + 1$ elements that have not lost a comparison. The algorithm must return one of them, say $A[i]$. The adversary can choose the other k elements that have not lost a comparison to be larger than $A[i]$, thus proving the algorithm wrong.

Problem 3. [High school] (30 points)

You are a guidance counselor in charge of putting high school students into one of two study halls. It doesn't matter how many students are in each study hall; what does matter is that certain pairs of students do not get along well and would cause a major disruption if they were placed in the same study hall. There are n students and you have a list of b

pairs of students who shouldn't be placed together. Give an algorithm that determines in time $O(n+b)$ whether it is possible to allocate the students to the two study halls without violating the b constraints. If it is possible to perform such a designation, your algorithm should produce it. (Note that some students may occur multiple times in the list of "bad" pairs, but no student would be paired with him/herself.)

Answer: There is an assignment of students to study halls if and only if the undirected graph (V, E), where V is the set of n students and E is the set of b bad pairs, is bipartite. A modification of DFS will do the job in time $O(n+b)$. The following algorithm computes for each node v a boolean value $hall(v)$: if $hall(v) = $ true, then student v is assigned to the first study hall, otherwise to the second.

```
for each v ∈ V do visited(v) := false;
for each v ∈ V do if not visited(v) then Explore(v, true).

procedure Explore(v, h):
visited(v) := true; hall(v) := h;
for each (v, w) ∈ E do
    if visited(w) and hall(w) = h then stop and report "no assignment possible";
    if not visited(w) then Explore(v, not h).
```

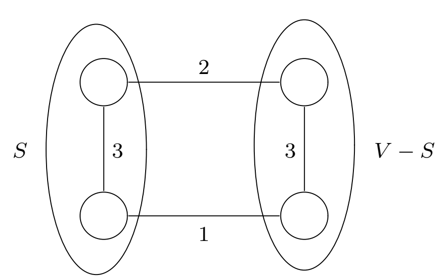
Problem 4. [Minimum spanning trees] (30 points)

Somebody proposes the following recursive algorithm to find a minimum spanning tree (MST) of a connected undirected graph $G = (V, E)$ with edge weights:

> First, partition the nodes V into two non-empty sets, S and $V - S$, so that each of the resulting parts of the graph, call them G_S and G_{V-S}, is connected. Second, recursively find a MST T_S for the subgraph G_S, and a MST T_{V-S} for the subgraph G_{V-S}. Third, construct from T_S and T_{V-S} a spanning tree for G by choosing from all edges $\{v, w\} \in E$ with $v \in S$ and $w \in (V - S)$ one of minimum weight.

Argue that this algorithm always finds a MST of G (for example, by demonstrating that it is an instance of the generic MST algorithm from class), or give a counterexample.

Answer: The algorithm is incorrect. On the following graph, it may return a spanning tree of cost 7, while the MST has cost 6.



Problem 5. [Hashing] (30 points)

Suppose we have a hash function h that, given a uniform distribution of input keys from a set U, maps each key with equal probability to one of m buckets. Suppose further that we are given a sequence $y_1, y_2, \ldots, y_n$ of keys to be hashed, each chosen uniformly at random from U. The i-th hash causes a collision if $h(y_i) = h(y_j)$ for some $j < i$. Hence there are between 0 and $n-1$ collisions. We want to compute the expected number of collisions.

(a) Assume that $n = 3$ and $m \geq 3$. What is wrong with the following argument? When we hash y_1, then there cannot be a collision. When we hash y_2, then the probability of a collision with y_1 is $\frac{1}{m}$. When we hash y_3, then the probability of a collision with y_1 is $\frac{1}{m}$, and the probability of a collision with y_2 is $\frac{1}{m}$. Hence the expected number of collisions is $\frac{3}{m}$.

(b) Still assuming $n = 3$ and $m \geq 3$, what is the correct value for the expected number of collisions and why?

Answer:

(a) When we hash y_3, the event that there is a collision with y_1 is not disjoint from the event that there is a collision with y_2, so we need to subtract the probability that $h(y_3) = h(y_1) = h(y_2)$, which is $\frac{1}{m^2}$. Hence the correct answer is $\frac{3}{m} - \frac{1}{m^2} = \frac{3m-1}{m^2}$.

(b) Another way of arriving at the same result is the following. The probability that no 2 of the 3 hash values are the same is $\frac{m-1}{m} \cdot \frac{m-2}{m} = \frac{(m-1)(m-2)}{m^2}$, and this event causes 0 collisions. The probability that exactly 2 of the 3 hash values are the same is $\binom{3}{2} \cdot \frac{1}{m} \cdot \frac{m-1}{m} = \frac{3m-3}{m^2}$, and this event causes 1 collision. The probability that all 3 hash values are the same is $\frac{1}{m} \cdot \frac{1}{m} = \frac{1}{m^2}$, and this event causes 2 collisions. The weighted sum is $0 \cdot \frac{(m-1)(m-2)}{m^2} + 1 \cdot \frac{3m-3}{m^2} + 2 \cdot \frac{1}{m^2} = \frac{3m-1}{m^2}$.

Problem 6. [Min cut] (30 points)

To *determinize* a randomized algorithm means to remove the random choices that the algorithm makes and replace them by deterministic (reproducible) decisions. Somebody determinizes the randomized min-cut algorithm from class so that in each contraction step, the algorithm always picks one of the edges with maximum weight (ties are handled in some unspecified manner).

(a) The input to a min-cut algorithm is a connected undirected graph with edge weights. Argue that if the input graph is a tree, then the determinized algorithm always finds a minimum cut.

(b) Give an input graph for which the determinized algorithm does not find a minimum cut (no matter how ties are handled).

Answer:

(a) In a tree, every edge by itself is a cut, because it disconnects the graph. Hence the minimum edges are the minimum cuts. The algorithm contracts all edges except for some minimum edge, and thus finds a minimum cut.

(b) For the following graph, the minimum cut has cost 2, but the algorithm finds a cut of cost 3. The first decision of the algorithm, to contract the edge with weight 2, is a wrong choice.