

## Problem Set 3 Solutions

Wednesday, October 7, 2009

**Problem 1.** (a) Consider the  $(k + 1)^{st}$  item inserted. Since only  $k$  buckets (at worst) are occupied, the probability that *both* candidate locations are occupied is only  $(k/n^{1.5})^2$ . Thus, the expected number of times an item is actually inserted into an already-occupied bucket is at most

$$\sum_{k=0}^{n-1} (k/n^{1.5})^2 = \frac{(n-1)(n)(2n-1)}{6n^3} \leq 1/3$$

Now let's consider pairwise collisions. Item  $k$  collides with item  $j < k$  only if (i) one of the candidate locations of item  $k$  is the location of item  $j$  (this has probability at most  $2/n^{1.5}$ ) and (ii) the other candidate location for item  $k$  contains at least one element (probability  $k/n^{1.5}$ ). Thus, the probability  $k$  collides with  $j$  is at most  $k/n^3$ . Summing over the  $k$  possible values of  $j < k$ , we find the expected number of collisions for item  $k$  is at most  $k^2/n^3$ . Summing over all  $k$ , we get the same result as above:  $O(1)$  expected collisions.

(b) Start with a 2-universal family of hash functions mapping  $n$  items to  $2n^{1.5}$  locations. Consider any particular set of  $n$  items. Consider choosing a random function from the hash family. The probability that item  $k$  collides with item  $j$  is  $1/2n^{1.5}$  by pairwise independence, implying by the union bound that the probability  $k$  collides with *any* item is at most  $1/2\sqrt{n}$ .

Now suppose that we allocate *two* arrays of size  $2n^{1.5}$  and choose a random 2-universal hash function from the family independently for each array. If an item has no collision in *either* array, then it will be placed in an empty bucket by the hash function. We need merely analyze the probability that this happens for every item (this would make the hash function perfect).

The probability that item  $k$  has a collision in *both* arrays is at most  $(1/2\sqrt{n})^2 = 1/4n$ . It follows that the expected number of items colliding with some other item is at most  $1/4$ . This implies in turn that with probability  $3/4$ , every item is placed in an empty bucket by the (perfect) hash function. This in turn implies that *some* pair of 2-universal hash functions defines a perfect hash for our set of  $n$  items.

Since every set of items gets a perfect hash from this scheme, it follows that the family of pairs of 2-universal functions above is a perfect hash family. Since the 2-universal family has size polynomial in the universe, so does the family of pairs of 2-universal functions.

- (c) When we sample a hash function from the above 2-universal family, we get a  $3/4$  probability of having no collisions. It follows that if we make 2 or more attempts, we can expect to find a collision-free hash function.
- (d) If we map our  $n$  items to  $k$  candidate locations in an array of size  $n^{1+1/k}$ , our collision odds work out as above and we get a constant number of collisions. Similarly,  $k$  random 2-universal hash families, each mapping to a set of size  $n^{1+1/k}$ , has a constant probability of being perfect for any particular set of items, so the set of all such functions provides a perfect family (of polynomial size for any constant  $k$ ). This gives a tradeoff of  $k$  probes for perfect hashing in space  $O(n^{1+1/k})$ .

Note that while we can achieve perfect hashing to  $O(n)$  space, the resulting family does *not* have polynomial size (since a different, subsidiary hash function must be chosen for each sub-hash-table).

- (e) We give an algorithm to decide if all people can be moved out in  $t$  steps. Now, we can increment  $t$  to find the shortest time in which all the people can move out.

The algorithm is as follows: given  $G$ , construct  $G_t$  as follows. For each  $v \in V$ , make  $t$  copies of  $v$ :  $v_1 \dots v_t$ . Construct an edge from  $v_i$  to  $v_{i+1}$  at time  $t$  with infinite capacity (people can just stay in rooms at a time step). Construct an edge from  $v_i$  to  $w_{i+1}$  with capacity  $C$  if there exists an edge from  $v$  to  $w$  with capacity  $C$  in  $G$ .

To test if all the people can get from the source to the sink in  $t$  timesteps, we check if the max flow in  $G_t$  is equal to the number of people initially at the source. If so, we can move all the people across this graph in  $t$  timesteps.

Note that the size of the graph is polynomially large, so the algorithm runs in polynomial time.

- (f) We can use the same overall idea: construct a graph  $G_t$ , and compute its max flow. If its max flow is equal to the total number of people we are trying to move, then  $t$  time units suffice to move all the people across the graph.

The construction of  $G_t$  is the same, except for the following. We create a sink  $s$  and source  $t$ . Let  $S$  be the start vertices, and let  $T$  be the sink vertices. We create a link from  $s$  to each  $x_1$ , for each  $x \in S$  with capacity equal to the number of people starting at  $x$ . Similarly, we create a link from each  $x_t$  (for each  $x \in T$ ) to  $t$  with infinite capacities.

- (g) Again, the overall idea is the same. But when we construct  $G_t$  now, we create edges between the layers in a different way: construct the edge linking  $v_i$  to  $w_{i+\delta}$  with capacity  $C$  if there is an edge between  $v$  and  $w$  with transit time  $\delta$ .

**Problem 2.** (a) Consider the admissible graph. Let  $L_i$  denote the vertices located in the layer  $i$  (with distance  $i$  from the source). We note that  $L_0 = s$  and  $L_d = t$ .

Then we take the summation of number of elements in all pairs of layers

$$\sum_{i=1}^d |L_i| + |L_{i+1}| \leq 2n .$$

We conclude the inequality because we count each layer at most twice. By pigeonholing, when we add  $d$  elements to get  $2n$ , there must be at least one element of value  $\leq 2n/d$ . Thus, we conclude that  $|L_i| + |L_{i+1}| \leq 2n/d$  for some  $i$ , or  $(|L_i| + |L_{i+1}|)/2 \leq n/d$ .

The number of edges between layers  $i$  and  $i + 1$  is at most<sup>1</sup>

$$|L_i| |L_{i+1}| \leq \left( \frac{|L_i| + |L_{i+1}|}{2} \right)^2 \leq \left( \frac{n}{d} \right)^2 .$$

Since the full graph has no edges jumping layers that are in the admissible graph (otherwise, the layers are incorrect in the admissible graph), we conclude that the cut between layers  $L_i$  and  $L_{i+1}$  has a capacity  $\leq (n/d)^2$ , as each edge has unit capacity. Thus, the max flow is at most  $(n/d)^2$ .

After  $k$  blocking flows,  $s$  and  $t$  are at a distance of  $k$  from each other. The max flow of the residual graph is  $(n/k)^2$  from above, thus we need to do at most  $(n/k)^2$  more blocking flows. Setting  $k = n^{2/3}$ , we get the total number of blocking flows is  $O(k + (n/k)^2) = O(n^{2/3} + (n/n^{2/3})^2) = O(n^{2/3} + (n^{1/3})^2) = O(n^{2/3})$ .

**Problem 3.** (a) Define the desired-meetings graph  $G$  to be a bipartite graph with left vertices  $L$  for students, right vertices  $R$  for faculty members, and an edge  $(x, y)$  whenever student  $x$  wishes to meet faculty  $y$ . There are  $d$  edges for each student, and  $d$  edges for each faculty member, so that  $d|L| = d|R|$ , and by cancellation  $|L| = |R|$ .

We wish to find a maximum matching in  $G$ , so we augment it with a source  $s$  connected to all the students, and a sink  $t$  that all the faculty are connected to, obtaining a graph  $G'$  on which we seek a max-flow. We will find the value of the max-flow by lower-bounding the value of any  $(s, t)$ -cut.

It is clear that the minimum  $(s, t)$ -cut has value at most  $n$ , because the cut  $\{s\}$  achieves this. Consider an arbitrary  $(s, t)$ -cut  $S, \bar{S}$ .  $S$  consists of the source  $s$ , some students  $L' \subseteq L$  and some faculty members  $R' \subseteq R$ . Then the cut consists of

- $n - |L'|$  edges from  $s$  to  $L \setminus L'$ .
- Edges from  $L'$  to  $R \setminus R'$ , and from  $L \setminus L'$  to  $R'$ . There are at least  $d||L'| - |R' ||$ .
- $|R'|$  edges from  $L'$  to  $t$ .

---

<sup>1</sup>First step of inequality follows from  $\sqrt{ab} \leq (a + b)/2$ , or geometric mean is at most arithmetic mean.

This yields a total of  $n + (d \pm 1)|L'| - |R'| \geq n$ , so we are guaranteed that all students and faculty can be matched up for a single slot.

- (b) Each time we apply the argument from (a), we reduce the degree of each student and faculty node by 1, but keep the regularity property. So we can do this repeatedly and schedule  $d$  time slots in which everyone can meet everyone they want to meet.
- (c) Let  $d$  be the maximum number of students who want to meet any one faculty member, or faculty members any one student wants to meet. Then we can reduce the general problem to the previous case by adding dummy students or faculty to make the numbers equal, and then adding dummy edges between students and faculty whose degree is less than  $d$ . In essence, we set up extra meetings that won't actually happen. We can always add these edges, because as long as some student has fewer than  $d$  faculty members to meet, some faculty member must have fewer than  $d$  students to meet. Note that this might create multiple edges for a student-professor pair. However, this does not affect the minimum cut, so the argument from (a) still holds.

**Problem 4.** Let  $i = 0$  represent the Red Sox. Then in the best case, the Red Sox win all their remaining  $\sum_j q_{0j}$  games for a total of  $W = w_0 + \sum_j q_{0j}$  season wins. We wish to determine if it possible to decide the remaining games so that any other team wins less than  $W$  games in all. We can formulate this as a max-flow problem as follows:

Let  $T$  be the set of teams other than the Sox. Define the network  $G = (V, E, u)$  where  $V = \left\{ \binom{T}{2} \cup T \cup \{s, t\} \right\}$  and

$$E = \{s\} \times \binom{T}{2} \cup \{(\{i, j\}, i) \mid i, j \in T\} \cup T \times \{t\}$$

with capacities

$$u(x, y) = \begin{cases} q_{ij} & x = s \text{ and } y \in \binom{T}{2} \\ q_{ij} & x \in \binom{T}{2} \text{ and } y \in T \\ W - w_i - 1 & x \in T \text{ and } y = t. \end{cases}$$

(Here,  $\binom{T}{2} = \{\{i, j\} \subset T \mid i \neq j\}$ ). Consider an (integer) max-flow of this network that saturates all the edges leaving  $s$ . We can interpret the flow on an edge of the form  $(\{i, j\}, i)$  as the number of wins team  $i$  scores against team  $j$  in the rest of the season. The capacity constraints on edges entering  $t$  ensure that every team wins fewer games than the Sox at the end of the season. Conversely, if the Sox can win at the end of the season, the record of wins and losses will give us an integer max-flow of this graph.

Therefore, we can check if the Sox can still win by computing the max-flow of this graph and seeing if it saturates all the edges leaving  $s$ .

**Problem 5.** The *assignment problem* specifies a bipartite graph with costs on the edges; the goal is to find a maximum matching of minimum total cost.

- (a) Give an efficient algorithm for the assignment problem.
- (b) You are given a complete graph  $G$  with edge costs in which some of the vertices are red and some are blue. You must find a set of edges of minimum cost that induce (i) odd degree on all red vertices and (ii) even degree on all blue vertices. Give a reduction to the assignment problem.
- (c) The lunch trucks have decided to start a delivery service. Model MIT as a collection of hallways (edges) of various lengths connecting intersections (vertices). If all intersections have even degree, then there exists an *Eulerian tour* that will traverse every hallway exactly once, allowing delivery throughout campus. If some intersections have odd degree, this is not possible. Nonetheless, you can find a route of *minimum total length* to traverse all hallways (in either, not necessarily both) directions by repeating certain hallways. Give a reduction to the previous part. **Hint:** what edges must be added to make the graph Eulerian?