Problem Set 4 Solutions

Problem 1. (a) Use a 2-universal hash function for each table on the second level, and let $f(s) = s^2$. If we insert $2s$ items into a table of size $f(2s)$, the expected number of collisions is

$$\begin{aligned} E[\#\text{ collisions}] &= \sum_{1 \le i < j \le 2s} \Pr[i \text{ collides with } j] \\ &= \sum_{1 \le i < j \le 2s} \frac{1}{4s^2} \\ &\le \frac{1}{2} \end{aligned}$$

which means, by Markov's inequality, that the probability of getting no collisions is at least $1/2$. Thus, when we have a table of size $f(2s)$, it will take a constant number of "attempts" in expectation to insert at least $2s$ items into the table, and each rebuild attempt takes $O(s)$ time. Thus, to insert 2^k items into the table, it takes

$$O(1 + 2 + \cdots + 2^k) = O(2^k)$$

time, and so in general, it takes $O(s)$ time to insert s items.

(b) Again, we use a 2-universal hash function for the top level. Suppose this table has size t. Note that the sum $\sum s_i^2$ is, just

$$\sum i, j[i \text{ collides with } j] - n + \sum_{i<j} [i \text{ collides with } j]$$

which, by pairwise independence, has expected value

$$\begin{aligned} E\left[\sum s_i^2\right] &= n + 2\sum_{i,j} \Pr[i \text{ collides with } j] \\ &= n + 2\sum_{i,j} \frac{1}{t} \\ &\le n + \frac{n^2}{t} \end{aligned}$$

Suppose we let $t \ge n$. Then the expected value of $\sum s_i^2$ is at most $2n$. Therefore, by Markov's inequality, the probability that $\sum s_i^2$ exceeds $4n$ is at most $1/2$ when only n items are inserted.

Suppose, whenever $\sum s_i^2$ exceeds $4n$, we rebuild the table to be of size $2n$. This means that, after our first rebuild after inserting n, we reach $2n$ items after a constant number of rebuilds in expectation. Each rebuild takes $O(n)$ time, so the total expected time is $O(1 + 2 + \cdots + 2^k) = 2^k$ where 2^k is the smallest power or 2 less than n. Thus the rebuilds take $O(n)$ time total.

(c) By (a) and (b), insertion takes $O(1)$ expected time without deletions. Suppose we now consider deletions. Every time we make a deletion, we mark a node. Consider a potential function Φ equal to the number of marked nodes. Then, each deletion has amortized $O(1)$ cost. Every time we rebuild the entire hash table, we take expected $O(n)$ time to do so. This also removes at least $n/2$ marked nodes, so this $O(n)$ time is payed for the by $O(n)$ decrease in potential. Thus, insertions still take only $O(1)$ time.

Problem 2. **(a)** False: consider vertices v and w having an edge from v to w and another from w to v. Given a flow f defined on these edges, we can increment both $f((v, w))$ and $f((w, v))$ by Δ to get another valid flow. (In the net flow model, note $f((v, w)) = -f((w, v))$, so falseness is obvious.)

(b) True: consider any pair (v, w) with both $f(v, w)$ and $f(w, v)$ positive. Assume without loss of generality that $f(v, w) \leq f(w, v)$. Decrease both quantities by $f(v, w)$. One is now zero, but flow conservation and capacity bounds have been maintained.

(c) False. Consider the graph with $V = \{s, 1, 2, 3, t\}$ and $E = \{(s, 1), (s, 2), (1, 3), (2, 3), (3, t)\}$. The capacities are $u(s, 1) = 2$, $u(s, 2) = 3$, $u(1, 3) = 4$, $u(2, 3) = 5$, and $u(3, t) = 1$. Essentially, the edge $(3, t)$ is a bottleneck, but you can choose whether to go through node 1 or node 2 to get the maximum flow. Thus there is no unique maximum flow even though all directed edges have distinct capacities.

(d) False. Consider graph with $V = \{s, t\}$ and $E = \{(t, s)\}$, the edge having a capacity of 1. Then, the initial graph has a max flow of 0, while the modified graph has a max flow of 1.

(e) False. Consider graph with

$$V = \{s, 1, 2, 3, t\},$$

and

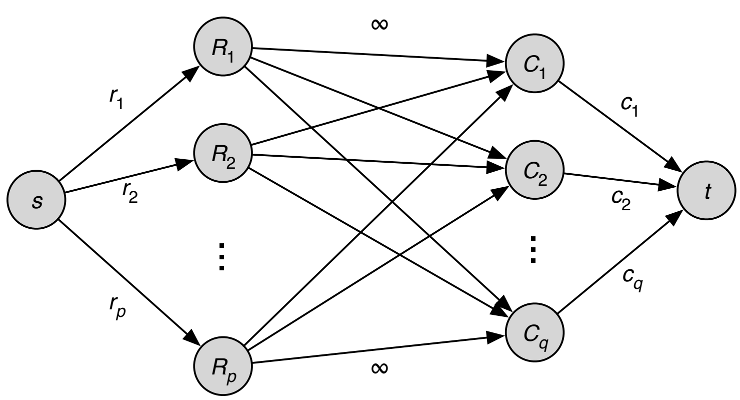
$$E = \{(s, 1), (1, 2), (1, 3), (2, t), (3, t)\}.$$

The capacities are $u(s, 1) = 3$, $u(1, 2) = 1$, $u(1, 3) = 1$, $u(2, t) = 1$, $u(3, t) = 1$. For this graph a min cut is $S = \{s, 1\}$. However, if we add a value of $\lambda = 100$ to the capacity of each edge, then the min cut becomes $S = \{s\}$

(f) True. Suppose there was no flow of value v from s to u. Then there exists an $s - u$ cut $s \in S, u \in \bar{S}$ such that $u(S, \bar{S}) < v$. Then either $t \in S$ or $t \in \bar{S}$. If $t \in S$, then there is a cut between t and u that is less than v and there is no flow between t and u of value v. If $t \in \bar{S}$, then there is no flow between s and t of value v. In either of these cases, we reach a contradiction, so flow must be transitive.

Problem 3. First, we go through the elements in Y. For all d_{ij} given in Y, we reduce r_i by d_{ij} and c_j by d_{ij}. In other words, we set $r'_i = r_i - \sum_j d_{ij}$ and $c'_j = c_j - \sum_i d_{ij}$, where d_{ij} is given in Y. Next, we construct a graph. We create vertices for each of the rows R_i and columns C_j along with a source s and sink t. We draw edges from the source to each of the rows R_i with capacity equal to the adjusted row sum. That is to say, we draw the edge (s, R_i), with $u(s, R_i) = r'_i$. Then we draw edges (C_j, t) from each of the column vertices C_j to the sink t with capacities $u(C_j, t) = c'_j$. Finally, we draw edges from all rows R_i to columns C_j with $u(R_i, C_j) = \infty$ provided that d_{ij} wasn't given in Y. If d_{ij} was given, we do not draw the edge (R_i, C_j).

For example, suppose Y is empty. Then we can draw the graph as follows:



Claim 1 *Consider a flow f on the graph. This flow corresponds to a solution to the matrix and all the constraints if and only if $f = \sum_i r'_i$ and is feasible.*

Proof. ($\Rightarrow$) Suppose there is a solution $\{d_{ij}\}$ to the matrix. Then we claim that $f(s, R_i) = r'_i$, $f(R_i, C_j) = d_{ij}$, and $f(C_j, t) = c'_j$ provides a feasible flow.

It is fairly obvious that $f = \sum_i r'_i$, as we saturated all the edges leaving s. Next, we just need to show that the flow is feasible. Consider the vertex representing the row R_i. We know that $\sum_j d_{ij} = r'_i$ as we satisfy our matrix constraints. Thus, if we send r'_i flow to R_i, we can send (exactly) the d_{ij} flow necessary along the edge (R_i, C_j) (these edges have infinite capacity). Thus, we are in accord with the capacity and conservation conditions. Similarly for C_j.

($\Leftarrow$) This direction is also trivial. Just reverse argument from above. If we're given a flow, we let $d_{ij} = f(R_i, C_j)$, then we argue that that is a matrix solution. If we have a flow with $f = \sum_i r'_i$, then we must be saturating all edges (s, R_i) and (C_j, t). Therefore, we must have $\sum_j d_{ij} = \sum_j (R_i, C_j) = r'_i$ for all i. Similarly, we have $\sum_i d_{ij} = c'_j$ for all j, and the solution is valid. ■

Thus, we can just find a solution to the matrix by finding a max flow on the graph. If the max flow f is not equal to $\sum_i r'_i$, then there is no solution. If it is, then we proceed to verify each d_{ij} as being protected or unprotected.

To verify whether d_{ij} is protected, we just need to ask the question, "can any other value work for d_{ij}?" We can break this into two smaller questions: "Does there exists a valid solution with $d'_{ij} < d_{ij}$?" and "Does there exist a valid solution with $d'_{ij} > d_{ij}$?" If the answer to either of these questions is "yes," then there is not a unique answer for d_{ij}. Thus, d_{ij} is protected. If the answer to both of these questions is "no," then there is no other answer for d_{ij}, and we conclude that d_{ij} is unprotected.

Okay, so now we just need to ask these questions about all edges. As we showed in the above claim, a valid $f(R_i, C_j)$ corresponds to a valid solution for d_{ij}. Let us look at each question individually.

''Does there exist a valid solution with $f'(R_i, C_j) < f(R_i, C_j)$?" To answer this question, we find *a*ny s-t path[1] going through (R_i, C_j), decrease the flow on this path by 1, and then decrease the capacity $u'(R_i, C_j) = f(R_i, C_j) - 1$.[2] In other words, we decrease the flow by 1 along a path going through (R_i, C_j) and then decrease the capacity of this edge such that it can only support flow $< f(R_i, C_j)$. Now, we just need to search for an augmenting path. If one exists, then there is a valid solution with $f'(R_i, C_j) < f(R_i, C_j)$.

Answering this question takes $O(m)$ time to find an augmenting path, and we need to ask this question for each of the $O(m)$ edges, for a total of $O(m^2)$ time.

''Does there exist a valid solution with $f'(R_i, C_j) > f(R_i, C_j)$?" Now, instead of forcing the flow to be less than a certain amount on this edge, we want to force it to be greater than a certain amount. Thus, we kinda want to force $f(R_i, C_j) + 1$ flow across (R_i, C_j). In other words, we assume that $d_{ij} = x + f(R_i, C_j) + 1$ for some value of $x \geq 0$. We reduce the problem to one in which all of r'_i, c'_j, and d_{ij} are decreased by $f(R_i, C_j) + 1$. These problems are isomorphic. To accomplish this on our graph, we first reduce the flow $f'(R_i, C_j) = 0$. We then reduce $u(s, R_i)$ and $u(C_j, t)$ by $f(R_i, C_j) + 1$ and set $f'(s, R_i) = u'(s, R_i) = f(s, R_i) - (f(R_i, C_j) + 1)$ and $f'(C_j, t) = u'(C_j, t) = f(C_j, t) - (f(R_i, C_j) + 1)$. This reduction corresponds to subtracting $f(R_i, C_j) + 1$ off of r'_i, c'_j, and d_{ij}. At this point, we have a flow $f' = f - (f(R_i, C_j) + 1)$. However, this flow is not feasible as we violate conservation at R_i and C_j. We just need to search for a path $C_j \rightsquigarrow R_i$ (in the residual graph) and send 1 flow along this path (augmenting path from C_j to R_i in $O(m)$ time). If we can find such a path, then we satisfy conservation, and we have a feasible max flow. Hence, there exists $f'(R_i, C_j) > f(R_i, C_j)$, and d_{ij} does not have a unique solution. If we cannot find such

[1] Try $s \to R_i \to C_j \to t$.

[2] Recall $u(R_i, C_j)$ was ∞ before.

a path, then we conclude that there is no feasible flow f' with $f' = f - (f(R_i, C_j) + 1)$. Thus, the answer to the question is "no," there is no solution with $f'(R_i, C_j) > f(R_i, C_j)$.

Answering this question takes $O(m)$ time to find an augmenting path, and we need to ask this question for each of the $O(m)$ edges, for a total of $O(m^2)$ time.

We have that asking all questions takes us only $O(m^2)$. Thus, the total running time of our algorithm is $O(\text{max-flow}) + O(m^2)$. Just to ground these numbers, a $p \times q$ matrix results in a graph with $n = \Theta(p + q)$ vertices and $m = \Theta(pq)$ edges.

Problem 4. **(a)** 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: $v_1 \ldots v_t$. 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.

(b) 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.

(c) 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 δ.

Problem 5. Let G be the graph under consideration.

(a) We assume that the array creation takes constant time. There are n insert, m decrease-key and n delete-min operations. The insert and decrease-key operations take $O(1)$ time. Delete-min takes $O(1 + d)$ time, where d is the number of empty buckets skipped during the delete-min operation. The bucket number of the last element deleted is D. Thus the total cost of delete-min is $O(m + D)$.

(b) Given the shortest path P from s to v of length d_v, the path $P + vw$ is known to have length $d_v + l_{vw}$. Therefore

$$d_w \leq d_v + l_{vw} \tag{1}$$

and the reduced edge length l_{vw} is non-negative.

(c) For any path $P = (sv_2, v_2v_3, \ldots, v_{k-1}v_k)$, the total reduced edge length is

$$\begin{aligned} l^d_{sv_2} + \ldots + l^d_{v_{k-1}v_k} &= (l_{sv_2} + d_s - d_{v_2}) + \ldots + (l_{v_{k-1}v_k} + d_{v_{k-1}} - d_{v_k}) \\ &= d_s + (l_{sv_2} + \ldots + l_{v_{k-1}v_k}) - d_{v_k} \end{aligned}$$

Therefore all paths to a vertex v have reduced length as the length minus (constant) d_{v_k}. So the shortest path to v is the same and has length $d_v - d_v = 0$.

(d) The scaling algorithm works as follows. We initially start with edge lengths 0, and distance function $d^1(v) = 0$ for all v. In step k, we shift a bit of the length in each edge, and compute distances d_0^{k+1} with reduced edge lengths based on d^k. We use distance function $d^{k+1} = d^k + d_0^{k+1}$ for the reduced costs in the next step. After $\lceil \log C \rceil$ steps the exact distance will be computed. We will now prove the correctness of this algorithm and analyze its running time.

Consider graph $G' = (V, E)$ constructed from the original graph $G = (V, E)$ with edge length $l'_{vw} = \lfloor l_{vw}/2 \rfloor$. In a scaling step, the distances in G' are used to compute reduced edge length in G. Notice that part (b) and (c) of this problem work for any distance function satisfying (**??**). So if we define the distance function as distance in G', we still have the same shortest paths in G and G'. This proves the correctness of the algorithm.

The length of shortest paths is 0 in the reduced graph G'. This means that in the original graph G the length of the shortest path is at most n. So Dial's algorithm takes $O(m + n) = O(m)$ time. The total time complexity for $\lceil \log C \rceil$ steps is therefore $O(m \log C)$.

(e) If a base b representation is used, there are $\lceil \log_b C \rceil$ scaling steps. The maximum distance D in each shortest path computation is bounded tightly by $n(b - 1)$. Thus the time complexity of our scaling algorithm is $O((m + n(b - 1)) \cdot \log_b C)$. If we set $b = 2 + m/n$ we achieve $O(m \log_{2+m/n} C)$ running time.

Problem 6. There is no available solution for this problem at this time.