CPSC 629: Analysis of Algorithms, Fall 2003

Solutions to Homework 4

Solution to 1 (24.3-4)
Let s and t be the two given vertices. The reliability of a path P between s and t, denoted $r(P)$, is defined to be the probability that the channel will not fail. Since the probabilities of different edges failing are independent, $r(P) = \prod_{e \in P} r(e)$. Our goal is to find a path P such that $r(P)$ is maximized. Note that $\lg r(P) = \sum_{e \in P} \lg r(e)$ and that each $\lg r(e)$ is negative since $r(e) < 1$. Our problem becomes finding a path P such that

$$-\lg r(P) = \sum_{e \in P} -\lg r(e)$$

is minimized. Thus we can solve the problem by assigning weight $-\lg r(u, v)$ to every edge (u, v), and using Dijkstra's algorithm to solve it.

Comments on common errors: The probability of a path not failing is the intersection of probabilities of all its edges not failing. Because the logarithm of a number between 0 and 1 is negative, you need to take its inverse. If you don't want to use logarithm, you need to modify the relax procedure of the Dijkstra's algorithm and use a max heap data structure in place of a min heap. □

Solution to 2 (24.3-6)
Create an array A of size $(V-1)W+2$. $A[0]$ is a pointer to the given source s. For $i = 1, \ldots, (V-1)W$, the cell $A[i]$ contains a pointer to a linked list of vertices with estimated distance i from s. The last cell $A[(V-1)W+1]$ contains a pointer to a linked list of vertices with estimated distance ∞ from s. The total space used in our data structure is $O(VW)$.

Initially all vertices are in the linked list $A[(V-1)W+1]$. DECREASE-KEY simply moves a vertex from one linked list to the head of another. To find the minimum key in this data structure, we scan the array upwards starting from the position of the latest EXTRACT-MIN (the first-round scan starts from $A[(V-1)W+1]$). We are guaranteed to find the minimum key in the array by scanning at most W cells since the maximum weight of the edges is W. The EXTRACT-MIN simply removes the head of the linked list at the minimum key.

Therefore, our data structure allows us to perform DECREASE-KEY in time $O(1)$ and EXTRACT-MIN in time $O(W)$. Using our data structure as the min-priority queue in Dijkstra's algorithm, we have desired running time.

Comments on common errors: Since the length of a shortest path can be up to $(V-1)W$, you need to have an array that is large enough to contain it. In EXTRACT-MIN it is important to start from the position of last EXTRACT-MIN. This will guarantee that only W cells in the array are scanned before the minimum key is found. □

Solution to 3 (24-1)

- a) Edges in G_f go from lower numbered vertex to higher number vertex. So a cycle in G_f would contradict the fact that the integers are totally ordered. Thus by definition $\langle v_1, v_2, \ldots, v_{|V|} \rangle$ is a topological sort of G_f. A similar argument applies to G_b.
- b) Define a FB-block as a path that has the pattern $f_1 \ldots f_i b_1 \ldots b_j$, where $f_1, \ldots, f_i \in E_f$ and $b_1, \ldots, b_j \in E_b$. Part of the FB-block can be empty. That is, a FB-block is allowed to take the form $f_1 \ldots f_i$, or $b_1 \ldots b_j$. Fix a shortest path P from s to x. Denote the minimum number of disjoint FB-blocks in P by $B(P)$. For any shortest path P, we have $|P| < |V|$, and hence $B(P) \leq \lceil |V|/2 \rceil$.

  In every pass, the improved Bellman-Form algorithm can relax edges in a block. In at most $\lceil |V|/2 \rceil$ passes, the algorithm will relax all the edges in the short path from s to x for any x. The algorithm finds shortest path for every vertex in the graph.

- c) The improved Bellman-Form algorithm had the same running time in every pass as the original one, but cuts the number of passes by half. They have the same asymptotical running time.

Comments on common errors: If you want to use induction in your proof, be sure to distinguish several cases depending on types of the edges.
□

Solution to 4 (25.2-6)
If any element on the main diagonal of the matrix becomes negative, we have a negative-weighted cycle.
□

Solution to 5 (25-2)

- a) INSERT and DECREASE-KEY have running time equal to the height of the heap, which is

$$\lg_d n = \lg n / \lg d = \lg n / \alpha \lg n = 1/\alpha.$$

  EXTRACT-MIN has running time $d \lg_d n = n^\alpha / \alpha$, because the procedure needs to scan the d children at every node.

- b) Let $d = n^\epsilon$. The running time of Dijkstra's algorithm is

$$O(V \cdot V^\alpha/\alpha + E/\alpha) = O(V^{1+\epsilon} + E) = O(E).$$

- c) Run the algorithm in (b) V times. Each time select a different vertex in G as the source.

- d) Our modification to Dijkstra's algorithm only affects how the min-priority heap is implemented. We can still use Johnson's algorithm to deal with negative-weight edges.

Comments on common errors: EXTRACT-MIN has different running time than INSERT and DECREASE-KEY in a d-ary min-heap.
□

Solution to 6 (26.1-9)
Model the map as a flow graph, where streets are modeled as edges and corners as vertices. Each edge has capacity 1. Look for a maximum flow larger than or equal to 2.
□

Solution to 7 (26.2-9)
Pick any vertex in the graph. Let it be v. Add a source s, and an edge $s \to v$ with capacity n^2. Convert every undirected edge (x, y) into two directed edges (x, y) and (y, x), each with capacity 1.

Then do the following for every vertex $u \neq v$ in G: add a sink t and an edge $u \to t$ with capacity n^2, and run the max flow algorithm on it.

The above loop runs at most $n-1$ times. Each time the max flow algorithm finds a flow that corresponds to a cut in the undirected graph. Eventually the minimum among all the cuts is returned as the edge connectivity.

Comments on common errors: You need to construct a flow network before the max flow algorithm is applied. A flow network is a directed graph. When converting a undirected graph into a directed graph, be sure to specify how the edges are directed. □

Solution to 8 (26-3)

- a) If there exists $I_k \in R_j$ such that $I_k \in S$, then the edge between I_k and E_j must be cut, which results in an infinite capacity cut.

- b) Let (S, T) be a min cut of G. Let I_T (respectively, I_S) be the subset of $\{I_1, \ldots, I_n\}$ that belongs to T (resp. S). Let E_T (resp. E_S) be the subset of $\{E_1, \ldots, E_m\}$ that belongs to T (resp. S). By (a), for $E_i \in E_T$, every $I_k \in R_i$ belongs to I_T. Therefore, we can choose to perform the experiments in E_T and carry instruments in I_T and meet the requirements. The net revenue is $\sum_{E_i \in E_T} p_i - \sum_{I_i \in I_T} c_i$. The net revenue and min cut capacity are related as follows:

$$\begin{aligned} MIN\ CUT\ CAPACITY &= \sum_{I_i \in I_T} c_i + \sum_{E_i \in E_S} p_i \\ &= \sum_{I_i \in I_T} c_i + \sum_{all\ E_i} p_i - \sum_{E_i \in E_T} p_i \\ &= \sum_{all\ E_i} p_i - (\sum_{E_i \in E_T} p_i - \sum_{I_i \in I_T} c_i) \\ &= \sum_{all\ E_i} p_i - NET\ REVENUE \end{aligned}$$

- c) Run the Edmonds-Karp algorithm on the flow graph. Once the max flow f has been found, define $S = \{v \in V : \text{there exists a path from } s \text{ to } v \text{ in } G_f\}$ and $T = V - S$. Then the partition (S, T) is a cut, and the edges across the cut from S to T belong to the min cut. The running time is dominated by the max flow algorithm, which is $O(VE^2) = O((m+n)(m+n+r)^2)$, where $r = \sum_{j=1}^{m} |R_j|$.

Comments on common errors: Notice how min cut capacity and net revenue are related.
□

Solution to 9 (26-4.a)
Update the residual network for the original max flow after increasing the capacity of a single edge by 1. Try to find an augmenting path in time $O(V + E)$ using breadth-first search. If there is an augmenting path in the updated residual network, then we can augment the original flow by 1 on that path.
□

Note: The solutions given here are terse, and in some cases, incomplete. Your answers should be complete and have more details. But you will lose points if they are too long or too complex.