Problem Set 5

All parts are due Thursday, November 20 at 11:59PM. Please download the .zip archive for this problem set, and refer to the `README.txt` file for instructions on preparing your solutions. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Name: Rishad Rahman

Collaborators: None

Part A

Problem 5-1. We can use a greedy algorithm which is similar to Dijsktra. Instead of keeping track of shortest path length to a vertex, we keep track of maximum height of a vehicle that can reach reach a vertex. The initialization makes all the node values ∞ but the relaxation is different and we do max-extracts instead. So $\text{RELAX}(u, v, w)$: if $d[v] < \min(d[u], w(u, v))$ then $d[v] \leftarrow \max(d[v], \min(d[u], w(u, v)))$. This relaxation basically updates $d[v]$ to a new maximum height of a larger vehicle can get to it from u. Then we modify Dijsktra so that we $EXTRACT-MAX(Q)$ (instead of the minimum). We do this until Q i.e. all vertices have been extracted and then we return $d[t]$. This algorithm is parallel to Dijkstra so the runtime is $O(V \lg V + E)$. To argue correctness we need to make sure that when we extract v, $d[v]$ is the maximum height of a vehicle that can reach v. Suppose it is not the maximum height then $\exists\epsilon$ such that a vehicle of height $d[v] + \epsilon$ can reach it. Well it couldn't have done that using (u, v) where u was extracted since $d[v] \leftarrow \max(d[v], \min(d[u], w(u, v)))$. Suppose some other path exists such that a $d[v] + \epsilon$ can reach v from vertex u' which has not been extracted. This means that $d[u'] \geq d[v]+\epsilon$ since relaxing is safe (won't go below the optimal value) so u' would be extracted before v, a contradiction hence we are always extracting when we have found the maximum height of a vehicle that can reach a vertex so our algorithm is correct.

Problem 5-2. Notice that if we ran Dijkstra for this graph, at each step the current distances to the non-extracted vertices would all be in the range $[d[v], d[v] + W]$ (or infinity) where v is the most recently extracted vertex. We can prove this by induction. The base case is easy since the first set of vertices all have distances in $[0, W]$ and $d[s] = 0$. Now suppose this is true for some iteration of Dijkstra so that all the non-extracted distances are between $[d[u], d[u] + W]$ with u the most recently extracted vertex. Now we extract the minimum-key vertex, v, and relax the edges coming out of it. After this for all non-extracted vertices, v', either $d[v']$ is unchanged in which case $d[v'] \leq d[u] + W \leq d[v] + W$ (by induction hypothesis) and $d[v'] \geq d[v]$ since $d[v]$ was the minimum, otherwise it does $d[v'] \leftarrow d[v] + w(v, v')$ but $d[v] + w(v, v') \leq d[v] + W$. This means $d[v] \leq d[v'] \leq d[v] + W$ for all non-extracted v' after extracting v which completes the induction. We can use this fact to develop an efficient way to extract the minimum valued vertex. First initialize a list A of length $W(V - 1) + 2$ whose entries will be linked lists and a variable to keep track of $\delta(u)$ where u is the most recently extracted vertex. Every time $d_1[v]$ changes (due to edge relaxation) into $d_2[v]$ we decrease the key of v in the list by deleting $d_1[v]$ from the linked list located at $A[d_1[v]]$ and then add it to the linked list located at $A[d_2[v]]$. We can let the linked list containing infinite distance estimates be located at $A[W(V-1)+1]$ since the total path weight will never exceed won't be more than $W(V - 1)$. Every time we need to extract min we iterate from $A[\delta[u]]$ to $A[\delta[u] + W]$ and extract any vertex from the first non-empty linked list then we update $\delta[u]]$. The decrease key operation takes $O(1)$ amortized time and extracting min takes $O(W)$ time so our total runtime is $O(WV + E)$.

Problem 5-3. We can use a naive BFS on every node and keep track of how many times we reach a node two edges away from a node. So for each node do a 2-stage BFS and mark the nodes you are able to reach on the 2nd step. If you ever try to visit a node already marked on the 2nd stage then you have a 4-cycle. Since a BFS is done on every point the runtime is $O(n^3)$ since there are n points and BFS is $O(n^2)$. To prove correctness suppose we have a node marked that was detected by BFS on the 2nd step. This means there are two separate paths of length 2 from a common node that reach it hence we have a 4-cycle. Now suppose we have a 4-cycle. Doing a BFS starting on any one vertex will mark the vertex directly across it twice so we detect that cycle.

Problem 5-4.

(a) **True**; Suppose the shortest path is made of weights w_i then $\sum w_i \leq \sum w'_j$ where w'_j is the weights that make some other path. Multiplying both sides by a positive constant does not change the inequality hence we must have that the shortest path remains the same.

(b) **False**; Consider the directed graph with three vertices s, u, v and edge weights $w(s, u) = 1$, $w(u, v) = 0$ and $w(v, u) = -1$. Obviously there is no shortest path from s to$u$ because of the negative weight cycle but making all the weights nonnegative would return a shortest path after running Dijkstra.

(c) **False**; Consider the directed graph s, u, t, $w(s, u) = -1$, $w(u, t) = 1$, $w(s, t) = 1$. The shortest path currently goes through u since it has total weight 0 but after squaring it gets a total weight of 2 so the shortest path goes to t directly instead.

(d) **True**; Negate each edge, turning it into a shortest path problem, and run Bellman-Ford which works since now equivalently there's no negative weight cycle and is $O(nm)$.

Part B

Submit your implemented python script on alg.csail.mit.edu.

Part C

Indicate whether or not you complete the Piazza poll. Completed.