Name: [Mohamed Ghanem]

CS 3510: Design & Analysis of Algorithms 02/09/2023

Problem Set 4: Graph Algorithms

Prof. Abrahim Ladha Due: 02/20/2023 11:59pm

- **Please TYPE your solutions using Latex or any other software. Handwritten solutions *won't* be accepted.**
- **Your solutions must be in plain English and mathematical expressions.**
- **Do not use pseudocode. Your answer will receive zero credit even if the pseudocode is correct.**
- **You can use the following algorithms as black-box for this problem set i.e. you don't have to explain the algorithm while using it. For example, you can write "Using DFS, compute the post labels":**
  - **BFS**
  - **DFS**
  - **Explore(v)**
  - **Toposort**
  - **SCC**
  - **Dijkstra's**
  - **Kruskal's**
  - **Construct a reverse graph**
  - **Construct the metagraph**
- **When we say "design an efficient algorithm", we are asking for a detailed description of the algorithm in English, i.e., without any code or psuedocode. Fastest (and correct) solutions are worth more credit.**
- **Show your work wherever necessary to receive full credits.**

Problem 1

(40 points; 20 points each)

Suppose you want to find the shortest path from u to v in a graph G that *passes through* a specified vertex w. Design an algorithm that runs in $O((|V|+|E|)\log|V|)$ for each of the following settings. Justify its correctness and explain its runtime.

a.) G is an undirected weighted graph.

Solution:

Proposed algorithm

1. Run Dijkstra's algorithm on the vertex w in the graph G. Dijkstra's returns the shortest distance from w to every other vertex in the graph G and a parent array A. The parent array A contains, for every vertex s in the graph G, the vertex immediately before s in the shortest path from w to s.
2. From Dijkstra's output, reconstruct the shortest paths w to v and w to u by following the back-pointers in the parent array A.
3. Find shortest path from u to w by reversing the order of vertices in the path w to u.
4. The shortest path from u to v that contains w is equivalent to the shortest path from u to w appended with the shortest path from w to v.

Time Complexity

The proposed algorithm consists of a Dijkstra's algorithm run, two path reconstructions, and a path reversal. Assume we have implemented Dijkstra's algorithm using a binary heap. The runtime complexity of Dijkstra's on the graph G is $\mathcal{O}((|V|+|E|)\log|V|)$. The runtime complexity of reconstructing a shortest path from the parent array A is $\mathcal{O}(|V|)$. The runtime complexity of reversing a path is also $\mathcal{O}(|V|)$. Therefore, the overall runtime complexity of the proposed algorithm is $\mathcal{O}((|V|+|E|)\log|V|)$.

Proof of Correctness

The shortest path from u to v that passes through the vertex w can be split into two sub-problems, the shortest path from u to w, and the shortest path from w to v. The graph is undirected, therefore if there exist some shortest path between any two vertices v_1 to v_2 in the graph, then there exists a shortest path from v_2 to v_1 that is identical in the set of vertices, but the reverse in their order. Using this property for undirected graphs, the algorithm runs Dijkstra's only on w. Dijkstra's returns the shortest path from w to every other vertex in the graph. From Dijkstra's output the shortest paths from w to v and w to u can be directly determined. Using the aforementioned property of undirected graphs, the path w to u can then be reversed to

obtain the shortest path from u to w. The solutions of the two sub-problems are then recombined to return the shortest path from u to v that passes through the vertex w, therefore, it must be correct.

b.) G is a directed weighted graph, and instead of a single u and v, we have k *queries* (u_1, v_1), (u_2, v_2), ..., (u_k, v_k) where for each one we want to find *only the weight* of the shortest path that passes through w (which is fixed) where $w \in G$ and $0 < k \leq |V|$.

Solution:

The graph G is now directed. This means the approach used in the part a. is now invalid because a path from w to some u_k where u_k is a source when reversed will not be a valid path in the graph G. The trivial solution would be to to run Dijkstra's twice every time a pair of vertices (u_k, v_k) is queried. This is $\mathcal{O}(k(|V| + |E|) \log |V|)$ which does not satisfy the runtime complexity requirement.

Proposed algorithm

1. Run Dijkstra's algorithm on the vertex w in the graph G. Dijkstra's returns the shortest distance from w to every other vertex in the graph G in an array A.
2. Reverse the graph G to obtain G_r.
3. Run Dijkstra's algorithm on the vertex w in the graph G_r. Dijkstra's returns the shortest distance from w to every other vertex in the graph G_r in the array A_r.
4. Create a map M that will map each vertex pair (u_k, v_k) to the weight of the shortest path from u_k to v_k that passes through w.
5. For every vertex pair in $(u_1, v_1), (u_2, v_2), ..., (u_k, v_k)$, set the map entry at the queried vertex pair to the sum of the weight of the shortest path from w to v_k in A and the weight of the shortest path from w to u_k in A_r.

$$M[(u_k, v_k)] = A[v_k] + A_r[u_k]$$

6. Return M which contains the result of the k queries.

Time Complexity

The proposed algorithm consists of two Dijkstra's runs, a graph reversal, and $2k$ array queries. Assume we have implemented Dijkstra's algorithm using a binary heap. The runtime complexity of Dijkstra's on the graphs G and G_r is $\mathcal{O}((|V| + |E|) \log |V|)$. The runtime complexity of reversing a Graph is $\mathcal{O}(|V| + |E|)$. Querying an array is a constant time operation. Therefore, the overall runtime complexity of the proposed algorithm is:

$$\mathcal{O}(2(|V| + |E|) \log |V| + (|V| + |E|) + 2k) = \mathcal{O}((|V| + |E|) \log |V|)$$

Proof of Correctness

The weight of the shortest path from some u_k to v_k that passes through the vertex w can be split into two sub-problems, the shortest distance from u_k to w, and the

shortest distance from w to v_k. After running Dijkstra's on G and G_r, we have the arrays A and A_r, that contain the shortest distance from w to every vertex in G and from every vertex in G to w. The weight of the shortest paths from w to $v_1, v_2, ...v_k$ can be determined from A. The weight of the shortest paths from $u_1, u_2, ...u_k$ to w in G is equal to the weight of the shortest path from w to $u_1, u_2, ...u_k$ in G_r. The solutions of the two sub-problems are then recombined to return the weight of the shortest path for all k queries $(u_1, v_1), (u_2, v_2), ..., (u_k, v_k)$ that pass through the vertex w, therefore, it must be correct.

Solution: (Finding the path)

The graph G is now directed. This means the approach used in the part $a.$ is now invalid because a path from w to some u_k where u_k is a source when reversed will not be a valid path in the graph G. The trivial solution would be to to run Dijkstra's twice every time a pair of vertices (u_k, v_k) is queried. This is $\mathcal{O}(k(|V| + |E|) \log |V|)$ which does not satisfy the runtime complexity requirement. Proposed algorithm

1. Run Dijkstra's algorithm on the vertex w in the graph G. Dijkstra's returns the shortest distance from w to every other vertex in the graph G that is reachable and a parent array A.
2. Reverse the graph G to obtain G_r.
3. Run Dijkstra's algorithm on the vertex w in the graph G_r. Dijkstra's returns the shortest distance from w to every other vertex in the graph G_r that is reachable and a parent array A_r.
4. Reconstruct the shortest paths from w to $v_1, v_2, ..., v_k$ by following the back-pointers in the parent array A.
5. Reconstruct the shortest paths from w to $u_1, u_2, ..., u_k$ by following the back-pointers in the parent array A_r.
6. Find shortest paths from $u_1, u_2, ...u_k$ to w by reversing the order of vertices in the paths from w to $u_1, u_2, ..., u_k$.
7. The shortest paths $(u_1, v_1), (u_2, v_2), ..., (u_k, v_k)$ that contain w are equivalent to the shortest paths from $u_1, u_2, ..., u_k$ to w appended with the shortest paths from w to $v_1, v_2, ..., v_k$.

Time Complexity

The proposed algorithm consists of two Dijkstra's runs, a graph reversal, $2k$ path reconstructions, and k path reversals. Assume we have implemented Dijkstra's algorithm using a binary heap. The runtime complexity of Dijkstra's on the graphs G and G_r is $\mathcal{O}((|V|+|E|) \log |V|)$. The runtime complexity of reconstructing a shortest path from the parent array A is $\mathcal{O}(|V|)$. The runtime complexity of reversing a path is also $\mathcal{O}(|V|)$. The runtime complexity of reversing a Graph is $\mathcal{O}(|V| + |E|)$. Therefore, the overall runtime complexity of the proposed algorithm is:

$$\mathcal{O}(2(|V| + |E|) \log |V| + (|V| + |E|) + 2k|V| + k|V|) = \mathcal{O}((|V| + |E|) \log |V|)$$

. Proof of Correctness

The shortest path from u to v that passes through the vertex w can be split into two sub-problems, the shortest path from u to w, and the shortest path from w to v. After running Dijkstra's on G and G_r, the parent arrays A and A_r contain the path information needed to construct that shortest paths $u_1, u_2, ...u_k$ to w and $v_1, v_2, ...v_k$ to w. The solutions

of the two sub-problems are then recombined to return the shortest path from u to v that passes through the vertex w, therefore, it must be correct.

Problem 2
(20 points)

A *feedback edge set* of an undirected graph $G = (V, E)$ is a subset $F \subseteq E$ such that *every* cycle in G contains at least one edge in F. Design an algorithm that takes as input a **connected** undirected weighted graph $G = (V, E)$ with all weights being positive and distinct, and returns a *minimum-weight* feedback edge set.

Justify correctness and running time, including explaining why your output is a feedback edge set (i.e. why every cycle contains edges on it), and why it is of minimum weight.

Hint: Think Kruskal's algorithm.

Solution: Proposed Solution

1. Initialize a set F, the minimum-weight feedback set.
2. Run Kruskal's algorithm on the graph G with the following modification:
   i.) Iterate through the edges in order of descending edge weight.
   ii.) If, when iterating through the edges, the algorithm determines that adding an edge will form a cycle in the spanning tree, add the edge to the feedback edge set F.
3. Return F as the minimum-weight feedback edge set.

Time Complexity

The proposed algorithm consists only of a Kruskal's algorithm run, the modifications to Kruskal's algorithm are all constant time changes and do not affect its runtime complexity. This means the proposed algorithm's runtime complexity is identical to Kruskal's algorithm which is $\mathcal{O}(|E|log|E|)$ (or $\mathcal{O}(|E|log|V|)$ if $E = V^2$).

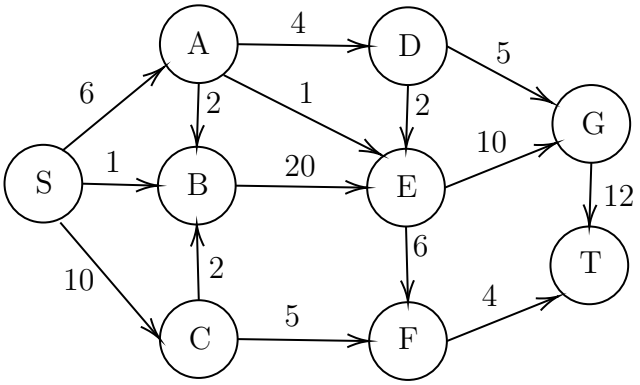
Proof of Correctness

Kruskal's algorithm avoids edges that form a cycle when constructing the minimum spanning tree. By definition an edge is not in any minimum spanning tree if and only if for that edge there exists a cycle for which that edge has the maximum-weight over every other edge in that cycle. This means that the edges in the minimum spanning tree and maximum-weight feedback edge set are complements of each other. This property allows us to use Kruskal's algorithm to determine a feedback edge set F such that every edge in F is part of a cycle in the graph G. Using normal Kruskal's algorithm would generate a minimum spanning tree, and a maximum-weight feedback edge set. The proposed algorithm iterates over the edges in decreasing edge weight. Using this modified version of Kruskal's algorithm would generate the maximum spanning tree, and its compliment set would be the minimum-weight

feedback edge set. The proposed algorithm uses modified Kruskal's algorithm to determine a minimum-weight feedback set F, therefore, it must be correct.

Problem 3
(20 points)

Consider the network shown below.



a.) What is the maximum flow from S to T? List the flow associated with each edge in the following format: (u, v, f_e) where $u \in G, v \in G$ and f_e is the flow along the edge $u \to v$.

Solution: The maximum flow calculated using the Ford-Fulkerson algorithm is equal to 13.
The flow associated with each edge for this maximum flow:

$(S, A, 6), (S, B, 1), (S, C, 6), (A, B, 1), (A, D, 4), (A, E, 1), (B, E, 4), (C, B, 2), (C, F, 4)$
$(D, G, 4), (D, E, 0), (E, G, 5), (E, F, 0), (F, T, 4), (G, T, 9)$

Maximum Flow $= 13$

b.) Find a matching minimum cut. Clearly list the edges in the minimum cut.

Solution:

The edges will be listed in the following format: (u, v, f_c) where $u \in G, v \in G$ and f_c is the capacity of the edge $u \to v$.

Call $explore(S)$ on the residual graph after from the Ford-Fulkeron's algorithm to determine $L = (S, C, F)$ and $R = G - L = (A, B, D, E, G, T)$. The minimum cut is equal to $cut(L, R)$.

Edges: $(S, A, 6), (S, B, 1), (C, B, 2), (F, T, 4)$

Minimum Cut $= 6 + 1 + 2 + 4 = 13$

Minimum Cut = Maximum Flow