COMP 3711 – Design and Analysis of Algorithms
2024 Fall Semester – Written Assignment 4
Distributed: 9:00 on November 16, 2024
Due: 23:59 on November 29, 2024

Your solution should contain

(i) your name, (ii) your student ID #, and (iii) your email address

at the top of its first page.

Some Notes:

- Please write clearly and briefly. In particular, your solutions should be written or printed on *clean* white paper with no watermarks, i.e., student society paper is not allowed.
- Please also follow the guidelines on doing your own work and avoiding plagiarism as described on the class home page. ***You must acknowledge individuals who assisted you, or sources where you found solutions.*** Failure to do so will be considered plagiarism.
- The term *Documented Pseudocode* means that your pseudocode must contain documentation, i.e., comments, inside the pseudocode, briefly explaining what each part does.
- Many questions ask you to explain things, e.g., what an algorithm is doing, why it is correct, etc. To receive full points, the explanation must also be *understandable* as well as correct.
- Submit a SOFTCOPY of your assignment to Canvas by the deadline. If your submission is a scan of a handwritten solution, make sure that it is of high enough resolution to be easily read. At least 300dpi and possibly denser.

1. (20 points)

   (a) (5 points) Prove that if G is an undirected graph with n vertices and n edges with no vertices of degree 0 or 1, then the degree of every vertex is 2.

   (b) (5 points) Let G be an undirected graph with at least two vertices. Prove that it is impossible for every vertex of G to have a different degree.

   (c) (10 points) In a group of 10 people, each one has 7 friends among the other nine people. Prove that there exist 4 people who are friends of each other.

   Solution:

   (a) **Solution 1:**

   The sum of vertex degrees is twice the number of edges. We have n edges so the sum of vertex degrees is $2n$. By assumption, every vertex has degree 2 or above. Therefore, if some vertex has degree three or above, the sum of vertex degrees is at least 2n + 1, which is a contradiction.

   Solution 2:

   When we have n edges, it means the sum of vertex degrees is equal to $2n$. Now if every vertex has a degree of 2 (which is the claim itself) we are fine. We Also know there are no vertex with degree 0 or 1. Now consider there exists at least one vertex with degree more than 2. This means there must exists a vertex with degree less than 2 (As the average degree for each vertex is 2) which is a contradiction. Notice that 2 can be seen as the "Average" degree of vertex so it is obvious if there exists a vertex with higher degree than 2, there must exists a vertex with lower degree to maintain the same average.

   (b) If there are n vertices, the possible values for degrees are $\{0, 1, 2, \cdots, n-1\}$. There are n values here. However, degrees 0 and $n-1$ can not happen at the same time. Because 0 means there exists a vertex that is not connected to any other vertex and $n-1$ means there exists a vertex that is connected to every other vertex. So there are $n-1$ possible values for the degrees and there are n vertices. Based on Pigeonhole Principle, at least there are two vertices with the same degree. This proves that it is not possible for every vertex to have a different degree.

(c) Consider an edge between two people means friendship between them. Every node is connected to 7 other nodes. Consider two nodes A, B that are already friend of each other. Out of the remaining 8 people, A, B are each connected to 6 of them. So based on pigeonhole principle, They have at least 4 mutual friends. Consider this 4 mutual friends to be x_1, x_2, x_3, x_4. If there exists any edge between any pair of them we are done because we will have 4 people who are pairwise friend of each other. So there must be no edge among any pair of them. Now consider any of those nodes. It has to be connected to 7 nodes. However, Now maximum degree of that node is 6 as it can not be connected to 3 nodes (and itself), which is a contradiction. So it means there exists 4 people that are pairwise friend of each other.

2. (20 points) Let $G = (V, E)$ be an undirected connected graph. Let n be the number of vertices in G. Let m be the number of edges in G. Design an algorithm to output a set of cycles $C_1, C_2, C_3, \ldots$ in G such that for every edge e of G, if e is contained in some cycle in G, then e is contained in some output cycle C_i. Explain the correctness of your algorithms. Analyze its running time which should be polynomial in n and m. Note that you are not required to output all cycles in G, and an edge of G may appear in multiple output cycles.

   Solution:

   We solve this question using the following approach:

   (a) First, we flag each edge as **"Unvisited"** and **"Unsatisfied"**.

   (b) For each **"unvisited"** edge, we aim to find a cycle that contains this edge. Let us assume this edge e connects the vertices v and u.

   (c) To achieve this, we temporarily delete the edge e from the graph G and check if there exists a path between u and v. If such a path exists, we find the path between u and v. Note that this can be done efficiently using either BFS or DFS.

   (d) If there is a path between u and v, it implies that a cycle exists that includes the edge e (think about why this is true). After identifying this cycle, we:
   - Print the cycle.
   - Mark each of the edges in the cycle as **"Satisfied"** and **"Visited"**.
   - Add the edge e back into the graph G.

   (e) If no path exists between u and v, then the edge e cannot belong to any cycle. In this case, we flag e as **"unsatisfiable"** and **"visited"**.

   (f) We repeat this process for all **"unvisited"** edges until we visit all of them.

   Our goal is to identify cycles such that they cover all edges that can be part of any cycle.

   ### Complexity Analysis

   - There are $O(m)$ edges in the graph.
   - For each edge, we perform either a BFS or DFS to find a cycle that includes the edge. This takes $O(n + m)$ time.
   - Therefore, the total complexity of the algorithm is $O(m \cdot (n + m))$, which is polynomial in n and m.

3. (20 points) Let $G = (V, E)$ be an undirected connected graph with n vertices and m edges. Each edge in G is also given an non-negative integer weight. Given a path P in G from a vertex u to a vertex v, the *bottleneck weight* of P, denoted by $wt(P)$, is the minimum edge weight in P. A *maximum bottleneck path* between u and v is the path Q between u and v such that $wt(Q) \geq wt(P)$ for all paths between u and v. Our problem is to report the maximum bottleneck paths between all pairs of vertices in G. Show that this problem can be solved by finding the minimum spanning tree of some graph. Explain the running time of your algorithm.

   Solution: Let H be G but with the edge weights negated. We find the minimum spanning tree T of H. We claim that the same tree T in G contains the maximum bottleneck paths between all pairs of vertices. Assume to the contrary that this is false. So there exists a pair of vertices u and v and a maximum bottleneck path Q between u and v such that $wt(Q) > wt(P)$, where P is the path between u and v in T. Let e_P be the edge of minimum weight in P. So deleting e_P splits T into two subtrees T_u and T_v such that T_u contains u and T_v contains v. Since Q is a path between u and v, some edge e of Q must connect T_u and T_v. Let e_Q be edge of minimum weight in Q. Therefore, $weight(e) \geq weight(e_Q) = wt(Q) > wt(P) = weight(e_P)$. It follows that $-weight(e) < -weight(e_P)$. Therefore, if we replace e_P by e, we would get a spanning tree of H that has a smaller weight than T, which is an impossibility.

   We first form H in $O(n + m)$ time. Then, we run Kruskal's algorithm on H in $O(m \log n)$ to obtain a minimum spanning tree T. Then, for each vertex v, we can run a BFS of T from v in $O(n)$ time. This sets up the parent points that we can then traverse to report the maximum bottleneck paths from v to all other vertices in $O(n^2)$ time. Thus, the total running time is $O(n^3)$.

4. (20 points) Let $G = (V, E)$ be a directed graph with positive edge weights.

   (a) (10 points) The cost of a cycle is the sum of the weights of edges on that cycle. A cycle is called shortest if its cost is the minimum possible, Design an algorithm to return the cost of the shortest cycle in G. If G is acyclic, your algorithm should say so. Your algorithm should run in $O(n^3)$ time, where n is the number of vertices in G. Explain the correctness of your algorithm. Derive its running time.

   (b) (10 points) Suppose that the edge weights in G are integers from the given range $[0, W]$. Describe an implementation of Dijkstra's algorithm that runs in $O((n + m) \log W)$ time.

   Solution:

   (a) Consider you want to solve this question with a small modification to the original question. Imagine our goal is to find the shortest cycle that includes a specific node s.

   If a cycle exists it will look like this:

   $s \to v_1 \to v_2 \to \cdots \to v_k \to s$

   The idea is to find the shortest path from s to all the nodes $U \in V$ that have an outgoing edge to s. This will form a cycle as when we calculate the shortest path to those nodes, we are going there and there is an edge back to s which forms a cycle.

   Now we need to find the shortest cycle that includes s. We just need to simply calculate:

   $$min_{u \in U}(shortest\ path(s, u) + d(u, s))$$

   So far this can be done in $O(n^2) + O(n) = O(n^2)$ if we just use the naive Dijsktra (using array to implement) and find the minimum value among those cases.

   Now remember that the original question is to find the shortest cycle in the whole graph. So we need to find the shortest cycle including each node. And find the minimum among them.

   So the total complexity is $O(n \cdot n^2) = O(n^3)$

   (b) Note that in the standard Dijkstra's algorithm, we use priority queue to manage vertices. Each vertex v has a priority based on its estimated distance d_v. The queue hold up to n vertices, and using a heap, each operation takes $O(log\ n)$ time. This results in an overall running time of $O((n + m)log\ n)$.

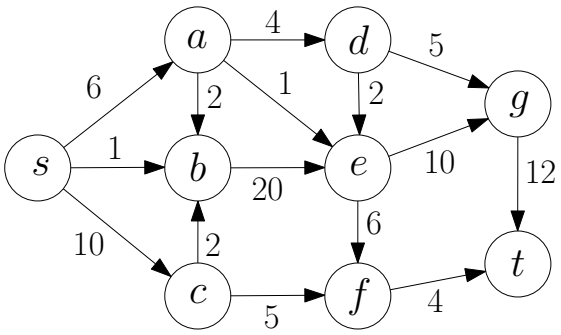
   the key is to observe that it suffices to store $W + 2$ items in the priority queue at any time. Once this observation is proved, the priority queue operations take $O(logW)$ each, giving rise to a running time of $O((n+m)logW)$. Now we discuss why this observation is true. There are three reasons.

   First, the distances of vertices extracted from the priority queue are monodically non-decreasing.

Second, if we extract a vertex u from the priority queue and relax an edge (u, v), it means that $d[v] <= d[u]+w(u, v) <= d[u]+W$. On the other hand, $d[v] >= d[u]$ because v is not even in the queue yet. Since the edge weights are integers, $d[v]$ must be an an integer, implying that $d[v]$ is an integer in the range $[0, W]$. Therefore, the distances of the vertices in the priority queue are in the range $\{d[u], d[u] + 1, d[u] + 2, \cdots, d[u] + W, \infty\}$. There are at most W+2 possibilities, irrespective of what d[u] is.

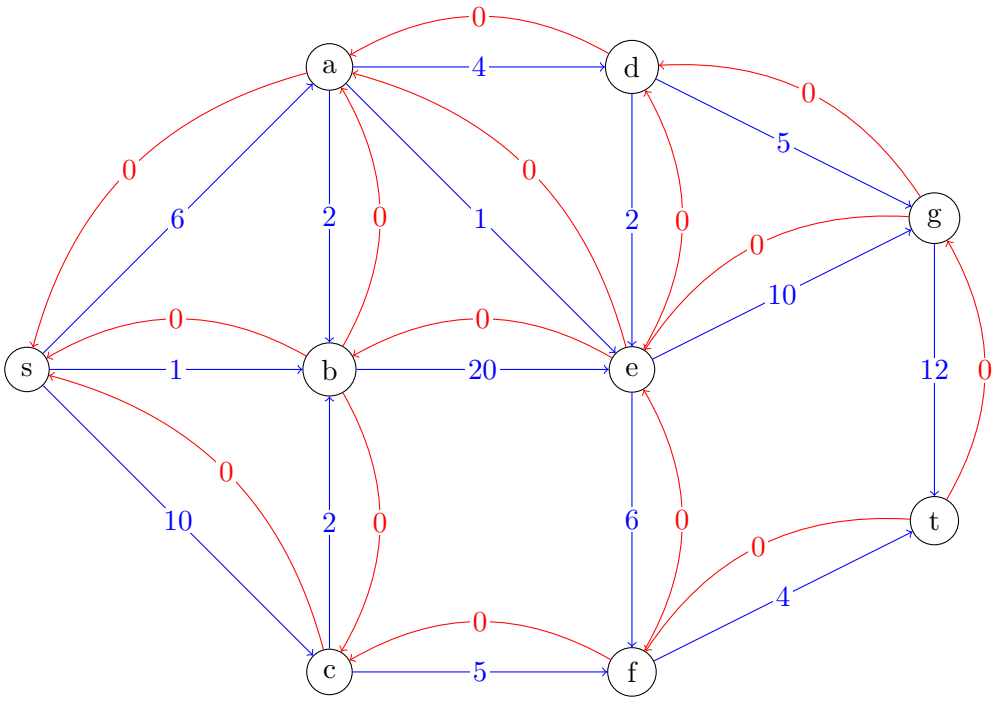
Third, give the insight above, instead of storing vertices in the priority queue, each item j in the priority queue is a non-empty doubly linked list L_j of vertices that have distance estimates equal to j.

5. (20 points) Find the maximum flow from s to t in the following flow network. Determine the corresponding minimum cut as well. Follow the notation in the lecture slides to show your intermediate steps.

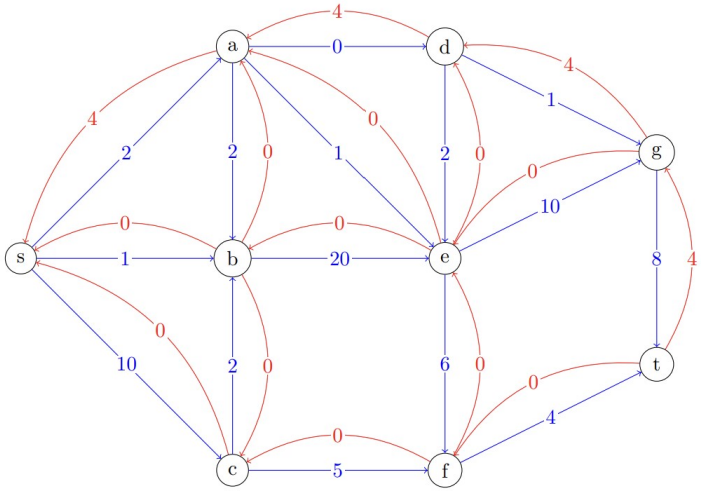


Solution: We solve the problem using the **Ford-Fulkerson algorithm** to find the maximum flow from s to t. Note that the blue edges are the forward edge and red edges are the backward edges.

Here is the initial residual graph of our input:

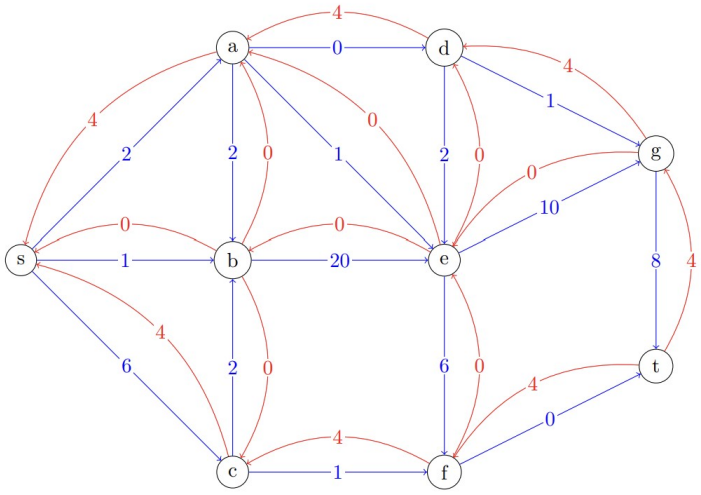


Step 1: One possible path in this graph can be $s \to a \to d \to g \to t$. The bottleneck edge is $a - d$ with flow 4. We update the residual graph as follows:



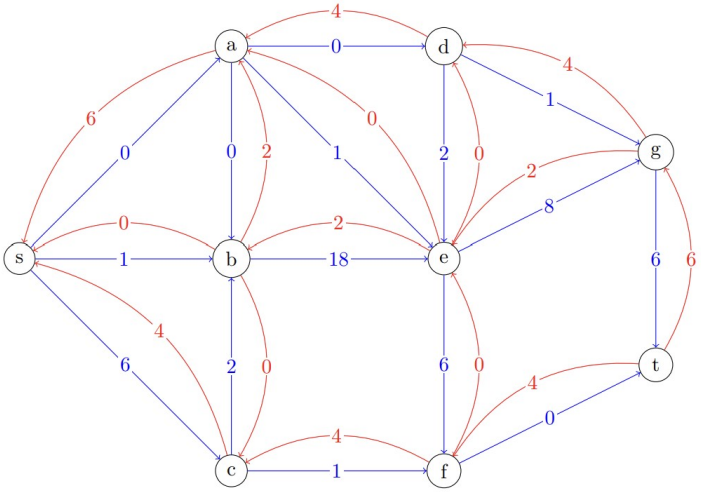
Total flow: 4

Step 2: In this residual graph one possible path is: $s \to c \to f \to t$. The bottleneck edge is $f - t$ with flow 4. We update the residual graph as follows:



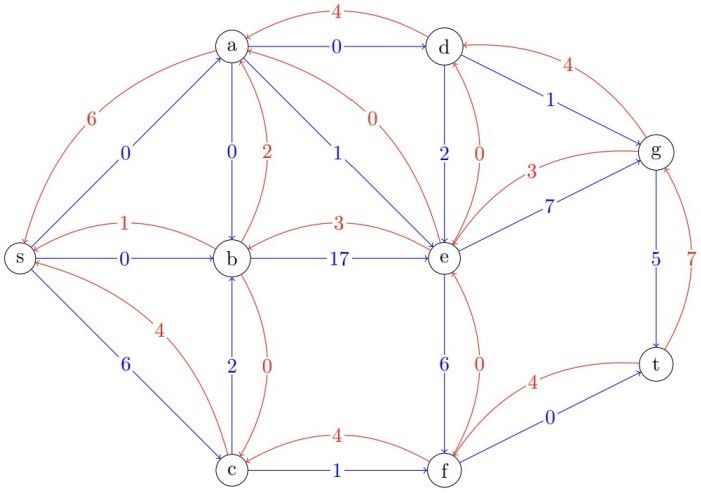
Total flow: 8

Step 3: In this residual graph one possible path is: $s \to a \to b \to e \to g \to t$. The bottleneck edge is $s - a$ with flow 2. We update the residual graph as follows:



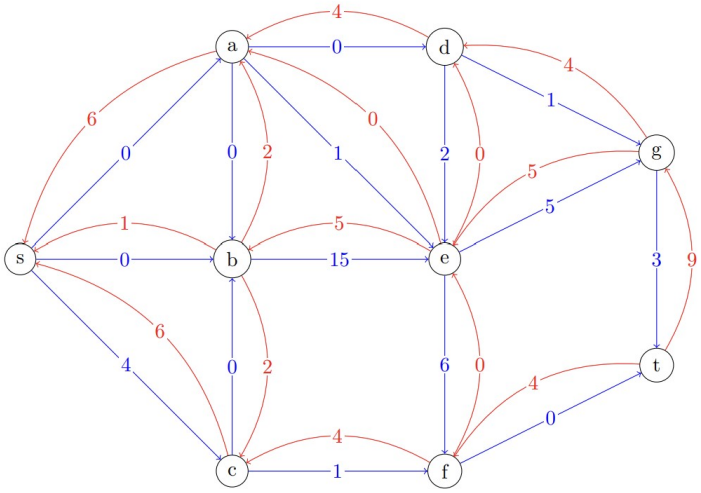
Total flow: 10

Step 4: In this residual graph one possible path is: $s \to b \to e \to g \to t$. The bottleneck edge is $s - b$ with flow 1. We update the residual graph as follows:



Total flow: 11

Step 5: In this residual graph one possible path is: $s \to c \to b \to e \to g \to t$. The bottleneck edge is $c - b$ with flow 2. We update the residual graph as follows:



Total flow: 13

There are no more path from s to t. So this is the final answer.

The min cut is when $A = \{s, c, f\}$ and $B = \{a, b, d, e, g, t\}$. The cut is 6+1+2+4 =13