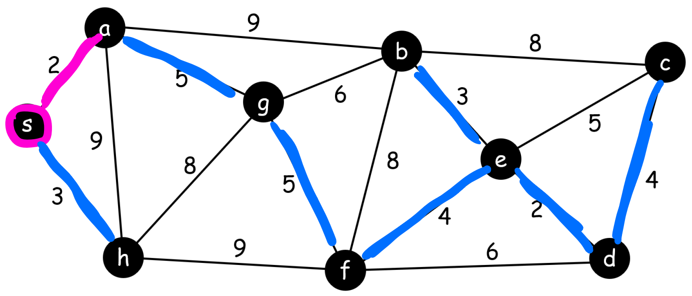
CISC2006: Algorithm Design and Analysis

Homework 4 **Due Date:** 23:59, 21 April 2023

Rules: Discussion of the problems is permitted, but writing the assignment together is not (i.e. you are not allowed to see the actual pages of another student). You can get at most 100 points if attempting all problems. Please make your answers precise and concise.

1. (10 pts) Consider the problem of computing the Minimum Spanning Tree (MST) of the following edge-weighted graph.



- List the edges that appear in the MST in the order they are included using Kruskal's algorithm. For example, if edge (a, b) is included as the first edge in the MST and (c, d) is the second, . . . , then the list is $(a, b), (c, d), \ldots$.

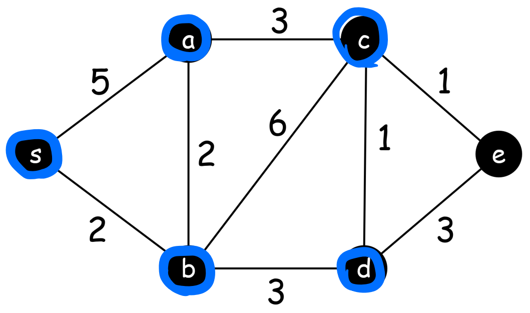
9 nodes, 8 edges is enough.

$\underbrace{(a,s),(e,d)}_{2}, \underbrace{(s,h),(b,e)}_{3}, \underbrace{(c,d),(e,f)}_{4}, \underbrace{(a,g),(g,f)}_{5}$

- List the edges that appear in the MST in the order they are included using Prim's algorithm. Suppose we begin with node s.

(s,a), (s,h), (a,g), (g,f), (f,e), (e,d), (e,b), (c,d)

2. (15 pts) Consider running Dijkstra's Algorithm on the following graph to compute the distance $d(s,u)$ between s and every other node $u \in V$.



Let $S \subseteq V$ contain the set of nodes u whose distance $d(s,u)$ is known.

Initially we have $S = \{s\}$ and $d(s,s) = 0$. In each step each node maintains an estimate $d'(s,u)$ (which is also an upper bound) of $d(s,u)$.

Fill in the values for each step of the execution.

Initialization: set $d'(s,a) =$ 5, $d'(s,b) = 2$, $d'(s,c) = d'(s,d) = d'(s,e) = \infty$.

Pick the node b with the smallest estimate, and include it into S: $S \leftarrow S \cup \{b\}$; set $d(s,b) = 2$.

Update the estimates of nodes in $V \setminus S$ as follows:

$d'(s,a) = \min\{5, d(s,b) +$ W_{ab} $\} =$ 4, $d'(s,c) =$ 8

$d'(s,d) =$ 5, $d'(s,e)$ remains the same value as before.

Pick the node a with the smallest estimate, and include it into S: $S \leftarrow S \cup \{a\}$; set $d(s,a) =$ 4.

Update the estimates of nodes in $V \setminus S$ as follows:

$d'(s,c) = \min\{8, d(s,a) +$ W_{ac} $\} =$ 7.

$d'(s,d)$ and $d'(s,e)$ remain the same values as before.

Pick the node d with the smallest estimate, and include it into S: $S \leftarrow S \cup \{d\}$

set $d(s,$ d $) =$ 5.

Update the estimates of nodes in $V \setminus S$ as follows: $d'(s,c) = 6$,

$d'(s,e) = 8$.

Pick the node c with the smallest estimate, and include it into S: $S \leftarrow S \cup \{c\}$;

set $d(s,$ c $) =$ 6.

Update the estimates of nodes in $V \setminus S$ as follows: $d'(s,e) =$ 7.

Pick the node e with the smallest estimate, and include it into S: $S \leftarrow S \cup \{e\}$; set $d(s,e) = 7$. Since $S = V$, the algorithm terminates.

$d(u) = |N(u)|$.

3. (15 pts) Recall that the degree $d(u)$ of a node u in a graph is the number of neighbors of u. Prove the following statements.

- (5pts) In any graph, the number of nodes with odd degree must be even.

Known : the sum of degree of all nodes is 2 |E|, which is an even number

We prove by contradiction

Sum of degree of all nodes is even, substract those nodes with even degree, the remaining result D is still even.
Now, what we left are nodes with odd degree, and remaining result of D. (Even number)

If number of nodes with odd degree is odd, then the remaining result of even X cannot be subtract until 0. Because even number subtract with odd number, will result in odd number. Hence, when we subtract all nodes' degree, the result won't be 0.

Hence, the number of nodes of odd degree must be even

- (5pts) Every tree contains at least two nodes with degree 1.

Let N nodes, M edges
Known sum of d(u) = 2 M
Known every tree has M= N-1 edges
By definition of tree, we know that all nodes have d(u) >= 1

We proved by contradiction.
Assume there is at most one node with degree 1. Then we have at least n-1 nodes with degree >= 2, which means
Sum of d(u) >= (n-1)*2 + 1= 2N-1 > 2M = 2(N-1) = 2N-2, which contradicts with our known

Hence, every tree contains at least 2 nodes with degree 1

- (5pts) If every node of a graph G has degree at least $n/3$, where n is the total number of nodes in the graph, then G has at most two connected components.

prove by contradiction.

assume 3 connected components, $C1, C2, C3$.

$\hookrightarrow$ sum of degree in $C_i = 2|E_i| \geq n/3\ |C_i|$.

we have $n \geq$ sum of degree in $C_i \geq n/3\ |C_i|$

multiply both side by 3, $3n \geq n|C_i|$.

$3n \geq n$

4. (20 pts) Prove the following statements.

- (10 pts) Any complete graph K_n with $n \geq 3$ is not bipartite.

Complete Graph with n >= 3 contains N nodes that are pairwise connected. For example, N=3, has 3 nodes that are pairwise connected.
By definition of bipartite graph, it has no odd cycle. Obviously complete graph with n>= 3 contains a cycle of length =3. So, it's not bipartite

- (10 pts) Any graph $G(V, E)$ with $\underset{\text{number of edge}}{|E|} \geq \underset{\text{number of nodes}}{|V|}$ contains at least one cycle.

We prove by contradiction

assume $|E| \geq |V|$ contain no cycle.

1) If it's connected, then it's a tree by definition. A tree has $|E| = |V| - 1$, which contradicts with $|E| \geq |V|$, hence it contains at least one cycle.

2) If it's not connected, then it has several connected components. (assume k)
Within each connected components, it's again a tree. Hence, we have $|E_i| = |V_i| - 1$ for all connected components, which mean $|E| = \sum |E_i| = \sum |V_i| - k = |V| - k < |V|$ contradicts with $|E| \geq |V|$ again.

5. (20 pts) **Implementation of the Graph Traversal Algorithms**.

   In this problem you need to implement the graph traversal algorithms using DFS and BFS. In the main function, your algorithm needs to (1) read an integer n that denotes the number of nodes; (2) read a sequence of edges, where each edge is specified by (i, j), indicating that there is an edge between node i and node j.

   You are guaranteed that both $i, j \in \{1, 2, \ldots, n\}$.

   Based on the input, you need to construct an undirected unweighted graph using adjacency lists. That is, every node u stores its set of incident edges as a list.

   Then a function is called to traverse the graph from node 1, and output the sequence of nodes in the order they are visited by the algorithm.

   You need to implement each of the following 2 algorithms as a function.

   - Depth First Search (DFS);
   - Breadth First Search (BFS).

   Several test cases of input will be provided. For each test case, the main function calls the two traversal algorithms, each of them outputs a traversal sequence of nodes.

6. (20 pts) **Implementing the Dijkstra's Algorithm**.

   In this problem you need to implement the Dijkstra's algorithms for computing the single source shortest path (SSSP) problem. In the main function, your algorithm needs to (1) read an integer n that denotes the number of nodes; (2) read a sequence of edges, where each edge is specified by (i, j, w), indicating that there is an edge between node i and node j with integer edge weight w.

   You are guaranteed that both $i, j \in \{1, 2, \ldots, n\}$ and $w \geq 1$.

   Based on the input, you need to construct an undirected weighted graph using adjacency lists. That is, every node u stores its set of incident edges as a list. Then a function is called to compute the distance $d(1, i)$ between node 1 and node i, for every $i \in \{2, 3, \ldots, n\}$, and output the distances $d(1, 2), d(1, 3), \ldots, d(1, n)$.

   Recall that to implement Dijkstra's algorithm we need to update and maintain an estimate $d'(1, i)$ for the distance between 1 and i, for every $i \in \{2, 3, \ldots, n\}$. You need to store the estimates in a Binary Search Tree (BST) that supports the query of minimum value. You can import libraries for the BST data structure.

   Several test cases of input will be provided. For each test case, the main function calls the Dijkstra's algorithm to compute and output the distances.

7. (20 pts) **Implementation of Algorithms for the MST Problem**.

   In this problem you need to implement the Kruskal's algorithm and Prim's algorithm for the Minimum Spanning Tree (MST) problem. In the main function, your algorithm needs to (1) read an integer n that denotes the number of nodes; (2) read a sequence of edges, where each edge is specified by (i, j, w), indicating that there is an edge between node i and node j with integer edge weight w.

   You are guaranteed that $i, j \in \{1, 2, \ldots, n\}$ and $w \geq 1$.

   Based on the input, you need to construct an undirected weighted graph using adjacency lists. That is, every node u stores its set of incident edges as a list. Then a function is called to compute the MST of the input graph, output the edges in the MST and also output the total edge weight in the MST.

   You need to implement each of the following 2 algorithms as a function.

   - Kruskal's algorithm that repeatedly includes an edge with minimum weight as long as it does not create a cycle. Recall that to test whether a new edge creates a cycle or not, you need to maintain a Union-Find data structure for the nodes. You can import libraries for the Union-Find data structure.
   - Prim's algorithm that repeatedly includes an edge with minimum weight to grow the current partial tree. You should run Prim's algorithm with $s = 1$, i.e., start growing the tree from node 1 outwards. Recall that to find the minimum weight edge, you need to maintain a Binary Search Tree (BST) that supports the query of minimum value. You can import libraries for the BST data structure.

   Several test cases of input will be provided. For each test case, the main function calls the two algorithms, each of them outputs a MST and its total edge weight.