Department of Informatics
Database Technology Group
University of Zurich Prof. Dr. Michael Böhlen

Informatics II
Exercise 12

May 21, 2022

Graphs

Task 1

1. For an undirected graph G with n vertices and m edges, which statement is correct?

   A. if m<n, G is unconnected

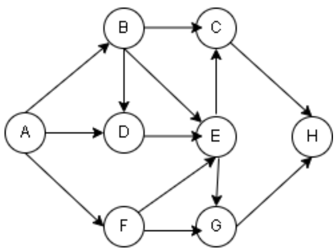
   B. if m>=n, there is a loop in G

   C. if m>n, G is connected

   D. if m<n, there is no loop in G

   Solution: B

2. A possible sequence of Depth-First search on the graph is



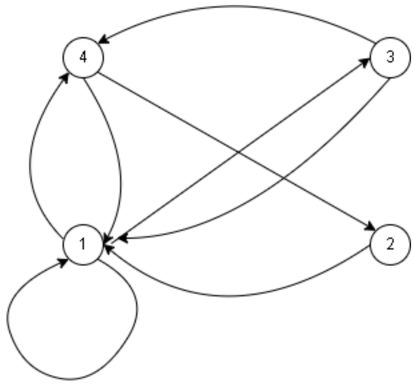
   A. ABDFCEGH

   B. ABCHDEGF

   C. ADECHBFG

   D. AFBDCEGH

   Solution: B



3. Show adjacency matrix of the graph below

   Solution: $\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$

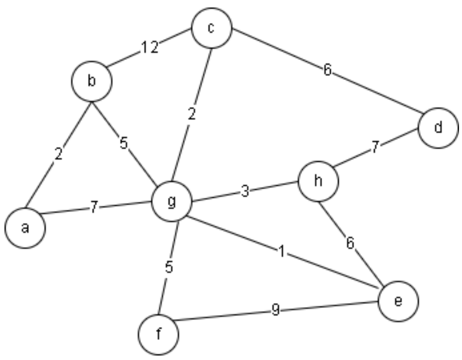
4. For an undirected weighted graph, its minimum spanning tree may not exist, but if it exists, it might not be unique.

   A. True

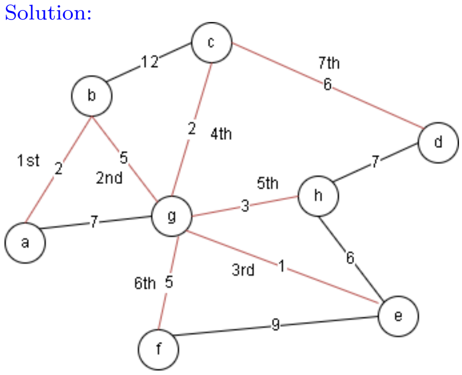
   B. False

   Solution: A

5. Use the Prim-Jarnik algorithm to compute to compute a Minimum Spanning Tree for the graph below (start from node 'a').

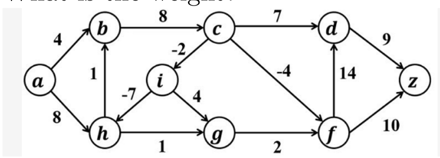


Solution:



Task 2

1. Which algorithm can help us get the minimum weight from 'a' to 'z'? What is the weight?



   A. Bellman-Ford 21

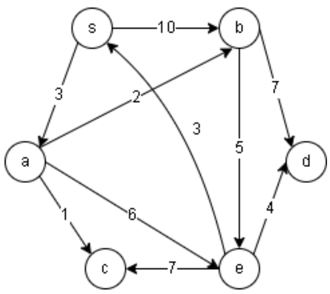
   B. Bellman-Ford 16

   C. Dijkstra 21

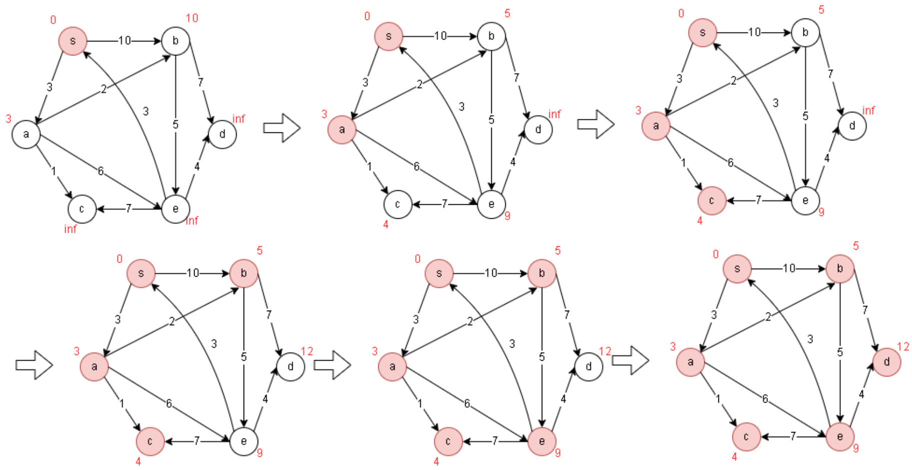
   D. Dijkstra 16

   Solution: B

2. Use Dijkstra algorithm to compute shortest path from start node 's' for the graph.



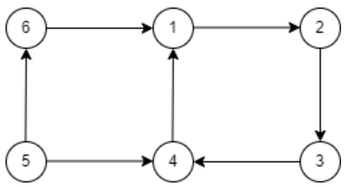
   Solution:



Task 3

A root vertex of a directed graph is a vertex u with a directed path from u to v for every pair of vertices (u, v) in the graph. In other words, all other vertices

in the graph can be reached from the root vertex. Given a graph, write C code that finds the root vertex using Breadth First Search as well as Depth First Search approach. A graph can have multiple root vertices. In such cases, the solution should find all the root vertices.
For example, the root vertex is 5 since it has a path to every other vertex in the following graph:



A framework code with the Queue implementation is provided. Use an adjacency matrix to represent the graph in the code.

Solution:

Clarification: The solution below uses an array named visited to keep track of whether the vertex v has been visited – v is not visited: visited[v] == 0 and v is visited : visited[v] = 1. In the slides, we use v.visited = True or False.

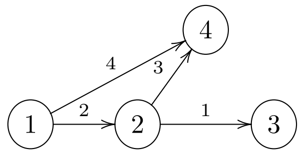
```
void dfs_tree(int graph[7][7], int v,int n, int visited[]) {
    visited[v]=1;
    int i;
    for(i=1;i<=n;i++){
        if(graph[v][i] ==1 && visited[i]==0) {
            dfs_tree(graph,i,n,visited);
        }
    }
}

int dfs(int graph[7][7], int v,int n){
    int i;
    int visited[n+1];
    for(i=0;i<=n;i++){visited[i]=0;}

    dfs_tree(graph, v,n, visited);
    for(i=1;i<=n;i++){
        if(visited[i] == 0){
            return 0;
        }
    }
    return 1;
}

int bfs(int graph[7][7], int v, int n) {
    int visited[n+1];
    int i=0;
    for(i=0;i<=n;i++){visited[i]=0;}
    node_t *head = NULL;
    enqueue(&head, v);
    int node;
    while (!isEmpty(&head)) {
        node = dequeue(&head);
```

```
            visited[node]=1;
            for(i = 1; i <=n; i++){
                if(graph[node][i] > 0 && visited[i] == 0) {
                    visited[i] = 1;
                    enqueue(&head, i);
                }
            }
        }
        for(i=1;i<=n;i++){
            if(visited[i] == 0){
                return 0;
            }
        }
        return 1;
}
int main() {
        int adj[7][7] = {
            {0,0,0,0,0,0,0},
            {0,1,1,0,0,0,0},
            {0,0,1,1,0,0,0},
            {0,0,0,1,1,0,0},
            {0,1,0,0,1,0,0},
            {0,0,0,0,1,1,1},
            {0,1,0,0,0,0,1}
            };
        int v=1;
        int n=6;
        printf("Algo: BFS \n");
        for (v=1;v<=n;v++){
            printf("Starting Vertex: %d \n", v);
            if(bfs(adj,v,n)){
                printf("Root Vertex is: %d \n",v);
            }
        }
        printf("Algo: DFS \n");
        for(v=1;v<=n;v++){
            printf("Starting Vertex: %d \n", v);
            if(dfs(adj,v,n)){
                printf("Root Vertex is: %d \n",v);
            }
        }
}
```

Task 4

Consider a directed weighted graph $G = (V, E)$. Let $n = |V|$ and $m = |E|$. Each vertex of V has a unique integer label between 1 and n. Each edge of E has a distinct weight between 1 and m.

The edges of E are stored in an array A. Each array element is a record with three fields: f (from), t (to), and w (weight). The array elements are *sorted* by the edge weight in ascending order. Note that position in A starts at 1.



$G = (V, E)$

	1	2	3	4
f	2	1	2	1
t	3	2	4	4
w	1	2	3	4

Array A representing G

Example. A[1] is for the directed edge $(2, 3)$, so $A[1].f = 2$, $A[1].t = 3$, and $A[1].w = 1$.

A **path** $p =< v_0, v_1, ..., v_k >$ is a sequence of vertices where adjacent vertices v_i and v_{i+1} are connected by an edge. A path can visit the same vertex *multiple* times. The **length** of a path p is the number of vertices of p.

Task 4.1:

A path $p =< v_0, v_1, ..., v_k >$ is a **weight-incremented path** if and only if the weights of two neighboring edges in p are strictly increased, *i.e.*, $w(v_{i-1}, v_i) < w(v_i, v_{i+1})$ where $1 \leq i < k$. A path with only one edge is also consider as a weight-incremented path.

Determine the maximum length of weight-incremented paths in G_2.

Answer:2; 1-2-4.

Task 4.2:

The path $p =< v_0, v_1, ..., v_k >$ terminates at the vertex v_k. Consider an array dp of size n. Let $dp[v]$ store the **maximum length** of weight-incremented paths that are terminated at vertex v in G with m edges. Note that position in dp starts at 1.

When edges with weights $\leq i$ are considered, we use dp_i to denote the states of dp.

Consider G and the array dp with size $4 (= |V_2|)$ for G.

Fill in $dp_1 =$	0	0	1	0	when edges with weights ≤ 1 are considered.
Fill in $dp_2 =$	0	1	1	0	when edges with weights ≤ 2 are considered.
Fill in $dp_3 =$	0	1	1	2	when edges with weights ≤ 3 are considered.
Fill in $dp_4 =$	0	1	1	2	when edges with weights ≤ 4 are considered.

Task 4.3:Determine the recursive problem formulation for $dp_i[v]$. Assume that the directed edge (a, b) has the weight i.

$$\begin{cases} dp_i[v] = dp_{i-1}[v] & \texttt{v} \neq \texttt{b} \\ dp_i[v] = max(dp_{i-1}[a] + 1, dp_{i-1}[v]) & \texttt{v = b} \end{cases}$$

Task 4.4: Write the pseudocode algorithm `maximum_len(A, m)` that returns the maximum length of weight-incremented paths in $G = (V, E)$ with m $(m = |E|)$ edges.

Requirement: The asymptotic complexity of your solution should be $O(m)$.

Algorithm: MAXIMUM_LEN(A, m)

```
initialize an array dp[n];
for i = 1 to n do
    dp[i] = 0;
max_len = 0;
for i = 1; i ≤ m; i = i + 1 do
    u = A[i].f
    v = A[i].t
    dp[v] = max(dp[v], dp[u] + 1)
    max_len = max(max_len, dp[v])
return max_len;
```