Informatics II, Spring 2023, Solution 12

Publication of exercise: May 21, 2023
Publication of solution: May 28, 2023
Exercise classes: May 30 - June 2, 2023

Task 1

1. A. Note C is no cycle since cycle must not visit an vertex twice
2. B
3. C, D
4. A, D

Graphs

5. FALSE. G_2 contains 2 cycles and is thus not acyclic. The first cycle is formed by the vertices 3 - 5 - 3 (Backedge). The second cycle is 3 - 4 - 5 - 3.

6. TRUE. This statement can easily be verified, since the adjacency matrix is not symmetric which must be the case for undirected Graphs. $G_2(1, 2)$ shows there is an edge from 1 to 2 but $G_2(2, 1)$ shows there is no edge back from 2 to 1.

7. TRUE. The Dijkstra algorithm can only be used for Graphs with non-negative edge weights. As soon as one edge is negative only one iteration of the Dijkstra Algorithm may return a suboptimal result. Therefore the Bellman-Ford must be used which basically runs the Dijkstra algorithm multiple times to adjust for the negative edge weights.

MST and SSSP

8. C
9. A
10. B, D

Task 2: Finding cycle in undirected Graph

```
algo: DFS-Acyclic(G, s) isAcyclic = TRUE;

s.col = G;
foreach u ∈ s.adj do
    if u.col==W and isAcyclic == TRUE then
        u.pred = s;
        isAcyclic = isAcyclic and DFS-Acyclic(G,u);
    end
    else if u.col == G and u != s.pred then
        return FALSE;
    end
end
s.col = B;
return isAcyclic;
```

Algorithm 1: DFS-Acyclic

Task 3: Count all possible Paths between two Vertices

```
algo: Count-Paths(M, visited, s, t) PathCount = 0;

visited[s] = true;
for i = 0 to M.length do
    if M[s][i] == 1 then
        if i == t then
            PathCount = PathCount + 1;
        end
        else if visited[i] == false then
            PathCount = PathCount + CountPaths(M, visited, i, t);
        end
    end
end
visited[s] = false;
return PathCount;
```

Algorithm 2: Count-Paths

Task 4: k-hop [20 FS Final Exam]

1.

Adjacency Matrix:

	1	2	3	4	5	6
1	0	1	0	1	1	0
2	0	0	1	0	1	0
3	1	0	0	0	0	0
4	0	0	0	0	0	1
5	0	0	0	0	0	0
6	0	0	0	0	0	0

2.

- The 2-hop neighbors of node 1 are nodes 3 and 6.
- Run a BFS search until first node in the queue has a distance of k. Dequeue and report nodes as long as they have a distance equal to k.

3. **Pseudocode:**

Algorithm: `khop`(G, v, k)

```
for (i = 1; i <= n; i++) do dist[i] = -1 ;
;
dist[v] = 0;
InitQueue(Q);
while (v ≠ −1 && dist[v] < k) do
    for (i = 1; i ≤ n; i++) do
        if (a[v, i] == 1 && dist[i] == −1) then
            dist[i] = dist[v] +1;
            Enqueue(Q,i);
    v = Dequeue(Q);
while v ≠ −1 && dist[v] == k do
    print(v);
    v = Dequeue(Q);
```

C code:

```c
void neighbors(int edges[7][2], int m, int n, int v, int k){

  int graph[n + 1][n + 1];

  for(int i = 0; i < m; i++) {
    graph[edges[i][0]][edges[i][1]] += 1;
  }

  int distance[n + 1]; // visited[i] means minimum distance from v to i .
  node_t *head = NULL;

  enqueue(&head, v);
  distance[v] = -1;

  int steps = 1;
  // k times BFS
  while(steps <= k) {
    node_t *new_head = NULL;
    int node;
    while ((node=dequeue(&head)) > 0) {
      for(int i = 1; i <= n; i++){
        if(graph[node][i] > 0 && distance[i] == 0) {
           distance[i] = steps; // update minimum distance.
           enqueue(&new_head, i);
        }
      }
    }
    head = new_head;
    steps += 1;
  }
  printf("%d-hop␣neighbors␣of␣node␣%d␣is:␣\n", k, v);
  for(int i = 1; i <=n ; i++) {
    if(distance[i] == k) {
      printf("node␣%d␣\n", i);
    }
  }
}
```