Department of Computer Science & Engineering, Faculty of Engineering

CSC2100B/S: Data Structures

Spring 2011

Written Assignment 4

Due: *29 Apr, 2011*

Name: _____

SID: _____

(Rev. 1.0 @ 11/04)

Section I: Revision Questions (10 points)

Put down your answers to the following simple revision questions.

1. A graph with 10 vertices has at most _____ edges. [ ] 1 pt

2. If a graph has 1000 vertices and 2000 edges, it is better to use the _____ representation to store the graph. [ ] 1 pt

3. Deleting an edge in the adjacency *matrix* representation takes _____ time. [ ] 1 pt

4. Deleting an edge in the adjacency *list* representation takes _____ time. [ ] 1 pt

5. Backtracking in depth-first search is accomplished by _____ in common C implementations. [ ] 1 pt

6. A spanning tree of a graph with 1000 vertices contains _____ edges. [ ] 1 pt

7. Prim's algorithm is a _____ algorithm that keeps picking edges with minimum weights, as long as no cycle is formed. [ ] 1 pt

8. If adjacency matrix representation is used, the time complexity of Prim's algorithm is _____ . [ ] 1 pt

9. Bellman-Ford algorithm can be used to detect _____ in graphs. [ ] 1 pt

10. Efficient implementation of Dijkstra's algorithm requires the use of _____ . [ ] 1 pt

_____

[ ] 10 pts

Section II: Short Questions (30 points)
Unless otherwise stated, you should answer the following questions in plain English, diagrams or math expressions instead of code snippets.

1. You are given the following set of edges of an undirected graph G:

4 pts

```
0-2 0-5 2-5 2-3 1-4 5-1 3-5 4-3 4-5
```

Draw G.
Write down any **TWO** cycles of length 4 in G.

2. The following is the adjacency matrix of graph G:

4 pts

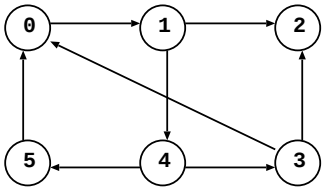
$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Draw G.
Put down the adjacency list representation of G.

8 pts

3. A directed graph G is shown below:

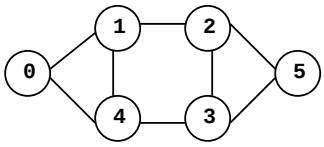
4 pts

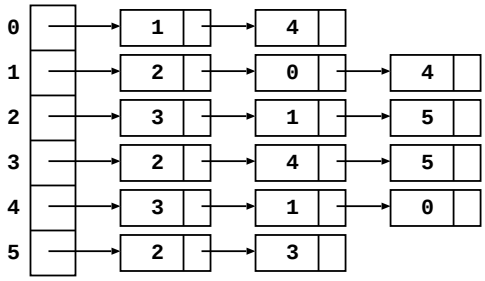


Assume that the graph is represented by *adjacency matrix*, apply depth-first search (DFS) to G: (i) write down the order in which the vertices are discovered; (ii) name all edges involved in the discovery; (iii) draw the DFS tree.

4. A undirected graph G with its adjacency list is shown below:

4 pts



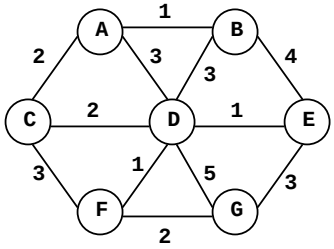


Begin with vertex 0, execute breadth-first search (BFS) on G: (i) write down the order in which the vertices are discovered; (ii) name all edges involved in the discovery; (iii) draw the BFS tree.

8 pts

4 pts

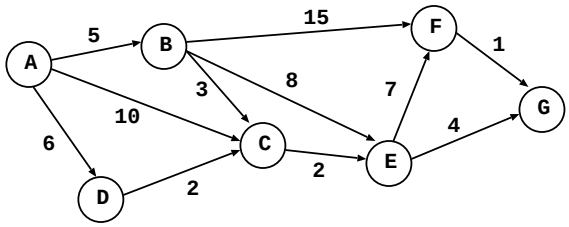
5. A weighted and undirected graph G with weights annotated along the edges is shown below:



Use Prim's algorithm to the find the minimum spanning tree for G. Begin your tree construction with vertex A.
Draw the minimum spanning tree found.

10 pts

6. A weighted and directed graph G is shown below:



We are interested to find the shortest path from vertex A to vertex G, using the algorithms we discussed in the lecture. To illustrate your working in the algorithm(s), use tables to show both $d[v]$ and $\pi[v]$. Also write down clearly the shortest path and the cost.

(a) (5 pts) Apply Bellman-Ford algorithm to find the shortest path from A to G.

14 pts

(b) (5 pts) Apply Dijkstra's algorithm to find the shortest path from A to G. (Use vertex name for tie-breaking in the minimum selection, if $d[v]$'s are the same)

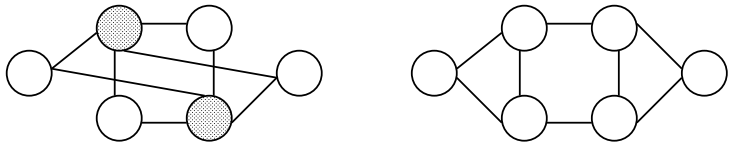
---

0 pts

Section III: Long Question / Code Study (30 points)
Unless otherwise stated, you should answer the following questions in plain English instead of code snippets.

1. A graph is two-colourable if there is a way to assign one of two colours (say black and white) to each vertex such that no edge connects two vertices of the same colour. For example, the graph on the left of the diagram below is two-colourable but that on the right is not.

15 pts



In this question, we assume a graph is represented by adjacency matrix defined in C as follows:

```
typedef struct {
    int V; /* no. of vertices */
    int E; /* no. of edges */
    int **adj; /* the adj. matrix */
} graph;
typedef struct {
    int v, w;
} edge_t;
```

(a) (5 pts) Based on DFS, design an algorithm that can check whether a graph is two-colourable or not. Write down the time complexity of your algorithm.

(b) (6 pts) Implement your algorithm stated in (a) using C language. Suppose that your routine is written in the C function `int is_two_colourable(graph G)` that returns 1 when it is two-colourable; 0 otherwise. You may assume the graph is undirected and the matrix is well-initialized.

15 pts

(c) (4 pts) Can BFS be used to solve the same problem? If yes, describe briefly your algorithm; if no, explain why.

2. The following code snippet shows a method to compute shortest path distance (stored in `d`) for a weighted and directed graph represented by adjacency matrix `adj`. The weights of the edges are stored in a separate array `w`. You may assume all arrays are well allocated and initialized.

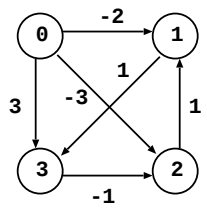
15 pts

```
1  #define INF 100
2  typedef struct {
3      int V; /* no. of vertices */
4      int E; /* no. of edges */
5      int **adj; /* the adj. matrix */
6  } graph;
7  void shortest_path(graph G, int **w, int **d){
8      int i, j, k;
9      for (i = 0; i < G->V; i++)
10         for (j = 0; j < G->V; j++)
11             d[i][j] = G->adj[i][j] == 1 ? w[i][j] : INF;
12
13     for (k = 0; k < G->V; k++)
14         for (i = 0; i < G->V; i++)
15             for (j = 0; j < G->V; j++)
16                 if (d[i][j] > d[i][k] + d[k][j])
17                     d[i][j] = d[i][k] + d[k][j];
18 }
```

(a) (2 pts) What is usage of the value `INF`?

(b) (3 pts) Analyze the time complexity of the routine `shortest_path()` in terms of V, the number of vertices in the graph, using the O-notation.

(c) (4 pts) Illustrate the algorithm on the following graph:



15 pts

Show your matrix `d` after every iteration of `k` loop on line 13.

(d) (4 pts) Explain how to modify the algorithm so that it can check whether a directed and weighted graph contains a negative weighted cycle or not.

(e) (2 pts) Suggest two advantages of the above algorithm over Dijkstra's algorithm in finding all pair shortest paths.

0 pts