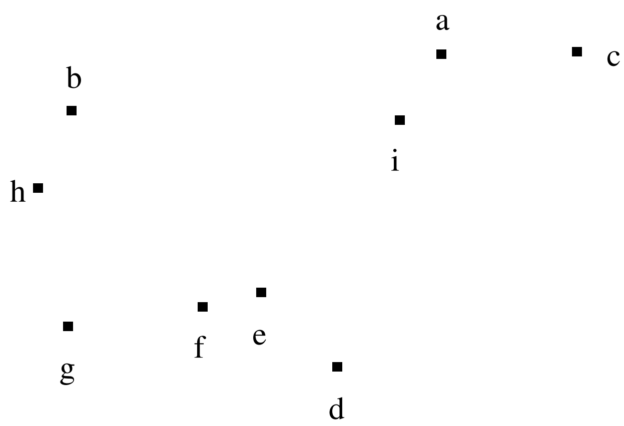
Data Structures (CS 1520) Lecture 27 Name:_________________

1. Suppose you had a map of settlements on the planet X




We want to build roads that allow us to travel between any pair of cities. Because resources are scarce, we want the total length of all roads build to be minimal. Since all cities will be connected anyway, it does not matter where we start.

a) Assuming we start at city “a” which city would you connect first? Why these cities?

b) What cities would you connect next?

c) What would be some characteristics of the resulting "graph" after all the cities are connected?

d) Does your algorithm come up with the overall best (globally optimal) result?

2. Prim’s algorithm for determining the minimum-spanning tree of a graph is an example of a *greedy algorithm*. Unlike divide-and-conquer and dynamic programming algorithms, greedy algorithms DO NOT divide a problem into smaller subproblems. Instead a greedy algorithm builds a solution by making a sequence of choices that look best ("locally" optimal) at the moment without regard for past or future choices (no backtracking to fix bad choices).

a) What greedy criteria does Prim’s algorithm use to select the next vertex and edge to the partial minimum spanning tree?

b) What data structure could be used to efficiently determine that selection?

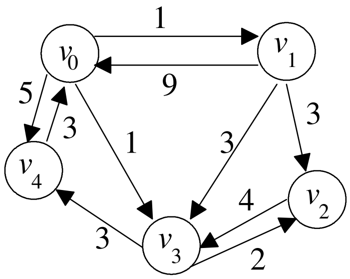
Prim's Algorithm (Graph *g*):

```
mark all edges as unvisited
mark all vertices as unvisited
mark any vertex v as visited
for each edge leading from v do
    add the edge to the heap
count = 1
while count < number of vertices in g do
    remove an edge from the heap
    if one end of the edge, say vertex w, is not visited then
        mark the edge and w as visited
        for each edge leading from w do
            add the edge to the heap
        count = count + 1
```

c) What would the run-time be for Prim's algorithm assuming an adjacency matrix graph implementation? Let n be the number of vertices and m be the number of edges.

d) What would the run-time be for Prim's algorithm assuming an adjacency matrix graph implementation?

3. Consider the following directed graph (diagraph) $G = (V, E)$ with adjacency matrix W:




W:

| From \ To | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | ∞ | 1 | 5 |
| 1 | 9 | 0 | 3 | 3 | ∞ |
| 2 | ∞ | ∞ | 0 | 4 | ∞ |
| 3 | ∞ | ∞ | 2 | 0 | 3 |
| 4 | 3 | ∞ | ∞ | ∞ | 0 |

| | vertex |
|---|---|
| 0 | $v_0$ |
| 1 | $v_1$ |
| 2 | $v_2$ |
| 3 | $v_3$ |
| 4 | $v_4$ |

Dijkstra's Algorithm is another greedy algorithm that finds the shortest path from some vertex, say $v_0$, to all other vertices. Four parallel arrays [0..(n-1)] are used:

included[i] = marks whether a known shortest path has be determined to $v_i$

distance[i] = length of current shortest path from $v_0$ to $v_i$ using only vertices in Y as intermediates

parent[i] = index of last vertex in Y on current shortest path from $v_0$ to $v_i$

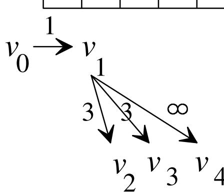
vertex[i] = mapping between vertex label and index position

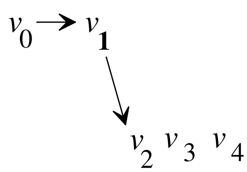
a) Initially, the length and touch arrays are shown below starting from $v_0$. Complete the trace of the algorithm.

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| included: | T | F | F | F | F |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| distance: | 0 | **1** | ∞ | 1 | 5 |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| parent: | | $v_0$ | $v_0$ | $v_0$ | $v_0$ |




| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| included: | T | T | F | F | F |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| distance: | 0 | 1 | 4 | 1 | 5 |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| parent: | | $v_0$ | $v_1$ | $v_0$ | $v_0$ |

Algorithm

1) Find closet vertex to $v_0$ that's not in Y (smallest distance[i] with included[i] = F)

2) Update distances now that this vertex is in Y.

3) Update parent accordingly if you find a shorter path

4) Mark this vertex as included in Y