Vlad Predovic
CS325
Due 5/8/2016

Homework 3

1. **Walkthrough of Prim's algorithm on graph shown (figure 1):**

Step 1: Visted = {a}	Cost = 0
Step 2: Visted = {a,j}	Cost = 1
Step 3: Visted = {a,j,b}	Cost = 3
Step 4: Visted = {a,j,b,d}	Cost = 6
Step 5: Visted = {a,j,b,d,h}	Cost = 10
Step 6: Visted = {a,j,b,d,h,f}	Cost = 12
Step 7: Visted = {a,j,b,d,h,f,c}	Cost = 16
Step 8: Visted = {a,j,b,d,h,f,c,e}	Cost = 20
Step 9: Visted = {a,j,b,d,h,f,c,e,i}	Cost = 25
Step 9: Visted = {a,j,b,d,h,f,c,e,I,g}	Cost = 32

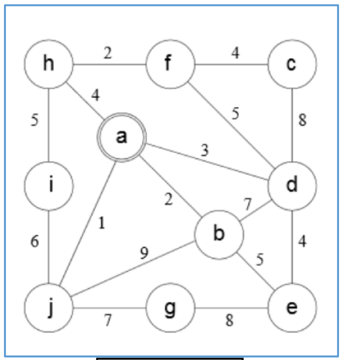


Figure 1

2. **Walkthrough of Dijkstra's algorithm on the graph shown (figure2):**
Note: Distances measured are from starting node (a).

Step 1: Unvisited = {a,b,c,d,e,f,g,h}
Distances = {a:0,b: ∞,c: ∞,d: ∞,e:∞,f:∞,g:∞,h:∞}
Step 2: Unvisited = {**~~a~~**,b,c,d,e,f,g,h}
Distances = {a:0,b:2,c:3,d:6,e:∞,f:∞,g:∞,h:∞}
Step 3: Unvisited = {~~a~~,**~~b~~**,c,d,e,f,g,h}
Distances = {a:0,b:2,c:3,d:6,e:11,f:∞,g:∞,h:∞}
Step 4: Unvisited = {~~a~~,~~b~~,**~~c~~**,d,e,f,g,h}
Distances = {a:0,b:2,c:3,d:6,e:11,f:7,g:∞,h:∞}
Step 5: Unvisited = {~~a~~,~~b~~,~~c~~,**~~d~~**,e,f,g,h}
Distances = { a:0,b:2,c:3,d:6,e:11,f:7,g:∞,h:∞}
Step 6: Unvisited = {~~a~~,~~b~~,~~c~~,~~d~~,e,**~~f~~**,g,h}
Distances = { a:0,b:2,c:3,d:6,e:8,f:7,g:∞,h:15}
Step 7: Unvisited = {~~a~~,~~b~~,~~c~~,~~d~~,**~~e~~** ,~~f~~,g,h}
Distances = { a:0,b:2,c:3,d:6,e:8,f:7,g:9,h:15}
Step 8: Unvisited = {~~a~~,~~b~~,~~c~~,~~d~~,~~e~~ ,~~f~~,**~~g~~**,h}
Distances = { a:0,b:2,c:3,d:6,e:8,f:7,g:9,h:10}
Step 9: Unvisited = {~~a~~,~~b~~,~~c~~,~~d~~,~~e~~ ,~~f~~,~~g~~,**~~h~~**}
Distances = { a:0,b:2,c:3,d:6,e:8,f:7,g:9,h:10}

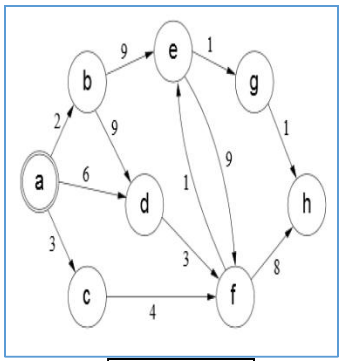


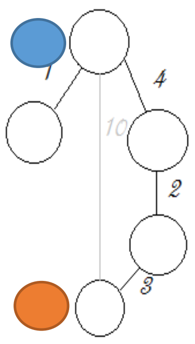
Figure 2

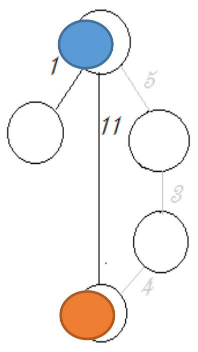
3. Consider an undirected graph G=(V,E) with nonnegative edge weights w(u,v)≥0. Suppose that you have computed a minimum spanning tree G, and that you have also computed shortest paths to all vertices from vertex s *E* V.
Now suppose each edge weight is increased by 1: the new weights w'(u,v) = w(u,v) + 1.

(a) Does the minimum spanning tree change? Give an example it changes or prove it cannot change.

No the minimum spanning tree does not change. It cannot change because th relationship of weight difference between all vertices V in G remains the same. To prove this one can run Prim's algorithm on the undirected graph G to find the minimum spanning tree. Since all the edge weights are distinct, and they are all increased by 1, Prim's algorithm will still run through all the edges in the same order as before. they were increased.
Since Prim's algorithm has been proven to be correct for the case of distinct undirected graphs, therefore this proof is complete and the minimum spanning tree cannot change.

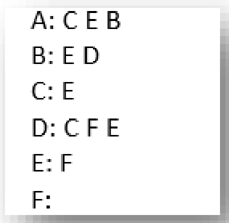
(b) Do the shortest paths change? Give an example where they change or prove they cannot change.





In this example the black lines represent the minimum spanning tree. The greyed out lines represent edges that were not the shortest path to each node. In the first example the shortest path from the blue to the orange node goes through two additional vertices for a total cost of 9. In the second example the direct path has a cost of 11 while the alternate path taken previously has a cost 12.

4. For the following problem, use the directed unweighted graph given by the following adjacency list. Be sure to consider edges in the given order.



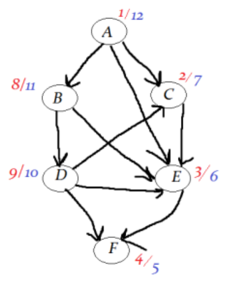
A: C E B
B: E D
C: E
D: C F E
E: F
F:

(a) What is the order in which the vertices are visited by DFS (depth first search)? For each vertex give the discovery and finishing time.

Stack Output→ A:1,C:2,E:3,F:4,F)5,E)6,C:7,B:8,D:9,D)10,B)11,A)12

Order:	A →	C →	E →	F →	B →	D
Discovery Time:	1	2	3	4	8	9
Finishing Time:	12	7	6	5	11	10

(b) Suppose that this graph represents courses in a CS department with course A being a prerequisite for courses C, E and B; course B a prerequisite for E and D, etc. Using your work above either give a valid order to take the courses or prove there is no valid order.



Valid order: A → B → D → C → E → F (Reverse finishing times)

5. A Hamiltonian path in a graph G=(V,E) is a simple path that includes every vertex in V. Design an algorithm to determine if a directed acyclic graph (DAG) G has a Hamiltonian path. Your algorithm should run in O(V+E). Provide a written description of your algorithm including why it works, pseudocode and an explanation of the running time.

Hamiltonian Path: Path through graph that touches each vertex once
DAG: Edges go in a direction, the edges never form a cycle.

A hamiltonian path in a directed graph touches each vertex once in a given order. In order for this to occur there would have to be an order within the entire graph where one vertex always come before another. A topological sort puts all vertexes in a list in order such that a vertex *u* must always come after the vertex that precedes it *v* in the list. Therefore in the case where a hamiltonian path exists, a topological sort would be forced to output a list that would represent the hamiltonian path as long as the *graph is acyclic*. If the graph was not acyclic, the loopback through would defeat the algorithm as there is no topological sort possible.

Therefore for this algorithm a topological sort will be used and the resulting graph will be checked for a hamiltonian path The one I used is essentially Khan's algorithm [1]. A list is created from all nodes with no incoming vertices. Then a while loop iterates through this list removing a node N and adding it to the sorted list. By default the nodes will be in topoloical order. Then it takes an arbitrary node with an edge incoming that had as source N and removes the edges, then placing it into No_Edge_list. After this is all complete, the algorithm checks that each vertex in Sorted_list had an edge going from the previous element in the list to the current one. If true, then a Hamiltonian path has been found.

Pseudocode:

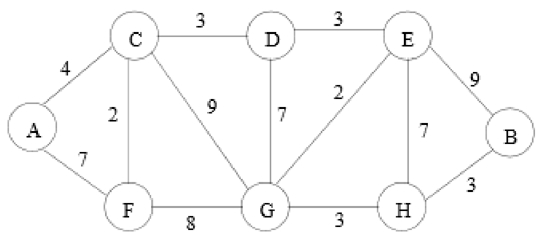
```
Sorted_list = list where final sort order goes, size = # of vertices, initialize to zero
For each vertex check if an edge has it as the destination. If none found add it to No_edge_list     O(v + e)
No_edge_list = list of nodes with no incoming edges

    While No_edge_list is not empty                                        O(v + e)
        Remove a vertex v from  No_edge_list and insert into Sorted_list
        For each vertex y with an edge a from v to y
            Remove edge a from the graph
            If vertex y has no other incoming edges
                Insert y into No_edge_list

    For each vertex v in Sorted_list starting with the second one.          O(v)
        check that there is an incoming edge to v from the previous vertex y
```

The outer while loop gets calculated V(vertices) number of times because it runs once for each vertex, placing it in the sorted list and removing it from the initial list. The inner for loop runs E(edges) number of times because as can be seen from the algorithm, it only executes while there is an incoming edge to a given vertex. This gives a time complexity of ***O(V+E).***

6. A region contains a number of towns connected by roads. Each road is labeled by the average number of minutes required for a fire engine to travel to it. Each intersection is labeled with a circle. Suppose that you work for a city that has decided to place a fire station at location G. (While this problem is small, you want to devise a method to solve much larger problems).



(a) What algorithm would you recommend be used to find the fastest route from the fire station to each of the intersections? Demonstrate how it would work on the example above if the fire station is placed at G. Show the resulting routes.

Since this problem involves placing a station, something long-term at G, it is beneficial to use an algorithm that will find the distance from this spot to all other vertices. Therefore Djikstra's algorithm is a great option for this problem. The time complexity can be as low as O(|E|+|V|log|V|) if implemented with a Fibonnaci heap [2].

1. **Walkthrough of Dijkstra's algorithm for problem 6a:**
   Note: Distances measured are from starting node (G).

   Step 1: Unvisited = {A,C,F,D,G,E,H,B}
   Distances = {A:∞,C:∞,F:∞,D:∞,G:0,E:∞,H:∞,B:∞}
   Step 2: Unvisited = {A,C,F,D,**G**,E,H,B}
   Distances = {A:∞,C:9,F:8,D:7,G:0,E:2,H:3,B:∞}
   Step 3: Unvisited = {A,C,F,D,~~G~~,**E**,H,B}
   Distances = {A:∞,C:9,F:8,D:5,G:0,E:2,H:3,B:11}
   Step 4: Unvisited = {A,C,F,D,~~G,E~~,**H**,B}
   Distances = {A:∞,C:9,F:8,D:5,G:0,E:2,H:3,B:6}
   Step 5: Unvisited = {A,C,F,**D**,~~G,E,H~~,B}
   Distances = {A:∞,C:8,F:8,D:5,G:0,E:2,H:3,B:6}
   Step 6: Unvisited = {A,C,F,~~D,G,E,H~~,**B**}
   Distances = {A:∞,C:8,F:8,D:5,G:0,E:2,H:3,B:6}
   Step 7: Unvisited = {A,**C**,F,~~D,G,E,H,B~~}
   Distances = {A:12,C:8,F:8,D:5,G:0,E:2,H:3,B:6}
   Step 8: Unvisited = {A,~~C~~,**F**,~~D,G,E,H,B~~}
   Distances = {A:12,C:8,F:8,D:5,G:0,E:2,H:3,B:6}
   Step 9: Unvisited = {**A**,~~C,F,D,G,E,H,B~~}
   Distances = {A:12,C:8,F:8,D:5,G:0,E:2,H:3,B:6}

(b) Suppose one "optimal" location (maybe instead of G) must be selected for the fire station such that it minimizes the distance to the farthest intersection. Devise an algorithm to solve this problem given an arbitrary road map. Analyze the time complexity of your algorithm when there are f possible locations for the fire station (which must be at one of the intersections) and r possible roads.

A somewhat brutal way of accomplishing this would be to run Djikstra's algorithm on every vertex and then compare the maximum value in each vertice's distance list. The vertex with the smallest greatest distance would be chosen as the optimal location. This algorithm would have to run Djikstra's V times so the time complexity would be O(V*E+ V^2log|V|)

(c) In the above graph what is the "optimal" location to place the fire station? Why?

The optimal location is E. The largest edge has as its destination vertex A with a cost of 10. No other vertex has a lower largest distance cost.

EXTRA CREDIT: Now suppose you can build two fire stations. Where would you place them to minimize the farthest distance from an intersection to one of the fire stations? Devise an algorithm to solve this problem given an arbitrary road map. Analyze the time complexity of your algorithm when there are f possible locations for the fire station (which must be at one of the intersections) and r possible roads

References

[1] http://connalle.blogspot.com/2013/10/topological-sortingkahn-algorithm.html

[2] https://www.cs.princeton.edu/courses/archive/fall03/cs528/handouts/fibonacci%20heaps.pdf