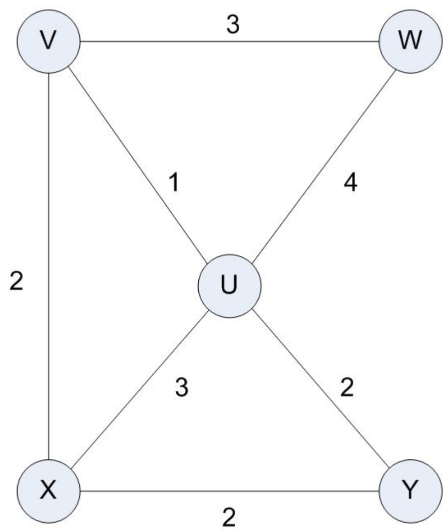
Lab 13 Solutions

1. Carry out the steps of Dijkstra's algorithm to compute the length of the shortest path between vertex V and vertex Y in the graph below. Your final answer should consist of three elements:

    a) The length of the shortest path from V to Y
    b) The list A[] which shows shortest distances between V and every other vertex
    c) The list B[] which shows shortest paths between V and every other vertex



Problem 2 without PQ

step 0:

$A[v] = 0 \quad B[v] = \emptyset$

step 1:

pool $\leftarrow \{(v,u), (v,w), (v,x)\}$

$A[v] + wt(v,u) = 0 + 1 = 1$ ← minimal

$A[v] + wt(v,w) = 0 + 3 = 3$

$A[v] + wt(v,x) = 0 + 2 = 2$

$A[u] = 1 \quad B[u] = B[v] \cup \{(v,u)\} = \{(v,u)\}$

add u to X

Step 2: Pool $\leftarrow \{(v,w), (v,x), (u,w), (u,x), (u,Y)\}$

$$\begin{array}{lccccc} & (v,w) & (v,x) & (u,w) & (u,x) & (u,Y) \\ \text{greedy length:} & 3 & 2 & 5 & 4 & 3 \\ & & \uparrow \text{minimal} & & & \end{array}$$

$A[x] = 2$

$B[x] = B[v] \cup \{(v,x)\} = \{(v,x)\}$

add x to X

Step 3: Pool $\leftarrow \{(v,w), (u,w), (u,Y), (x,Y)\}$

$$\begin{array}{lcccc} & (v,w) & (u,w) & (u,Y) & (x,Y) \\ \text{greedy length:} & 3 & 5 & 3 & 4 \\ & \uparrow \text{minimal} & & & \end{array}$$

$A[w] = 3$

$B[w] = B[v] \cup \{(v,w)\} = \{(v,w)\}$

add w to X

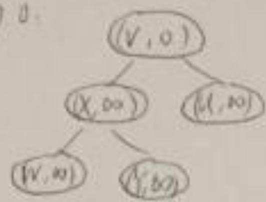
Step 4: Pool $\leftarrow \{(u,Y), (x,Y)\}$

$$\begin{array}{lcc} & (u,Y) & (x,Y) \\ \text{greedy length:} & 3 \leftarrow \text{minimal} & 4 \end{array}$$

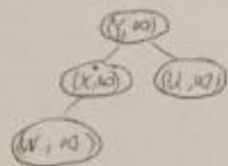
$A[Y] = 3 \quad B[Y] = B[u] \cup \{(u,Y)\} = \{(v,u), (u,Y)\}$

Problem 2 with PQ

step 0:



step 1: (V,0) ← remove Min



adj to V and in PQ W, X, U

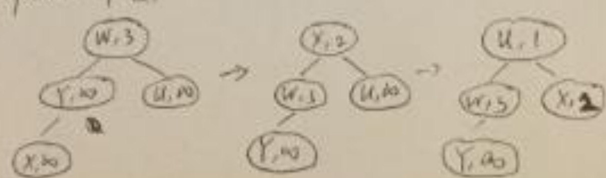
compute greedy lengths for (V,W), (V,X) (V,U)

$A[V]+wt(V,W)=0+3=3 < A[W]$

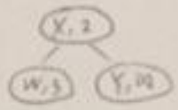
$A[V]+wt(V,X)=0+2=2 < A[X]$

$A[V]+wt(V,U)=0+1=1 < A[U]$

update PQ.



Step 2: $(U,1) \leftarrow$ remove Min



adj to U still in PQ: W, X, Y

compute greedy lengths for (U,W) (U,X) (U,Y)

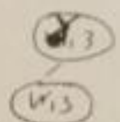
$A[U] + wt(U,W) = 1+4=5 \not< A[W]$

$A[U] + wt(U,X) = 1+3=4 \not< A[X]$

$A[U] + wt(U,Y) = 1+2=3 < A[Y]$

update PQ:

Step 3: $(X,2) \leftarrow$ removeMin



adj to X still in PQ: Y

compute greedy length for (X,Y)

$A[X] + wt(X,Y) = 2+2=4 \not< A[Y]$

Step 4: $(Y,3) \leftarrow$ remove Min

PQ: (W,3)

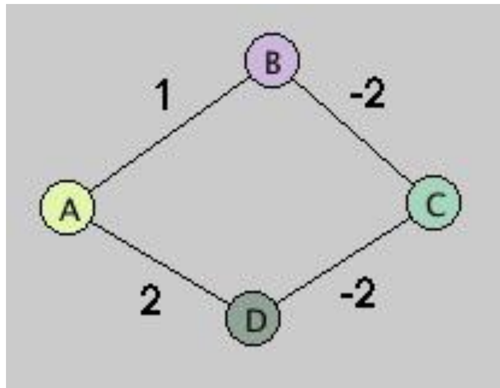
adj to Y still in PQ: nothing

Step 5: $(W,3) \leftarrow$ removeMin

adj to Y still in PQ: nothing

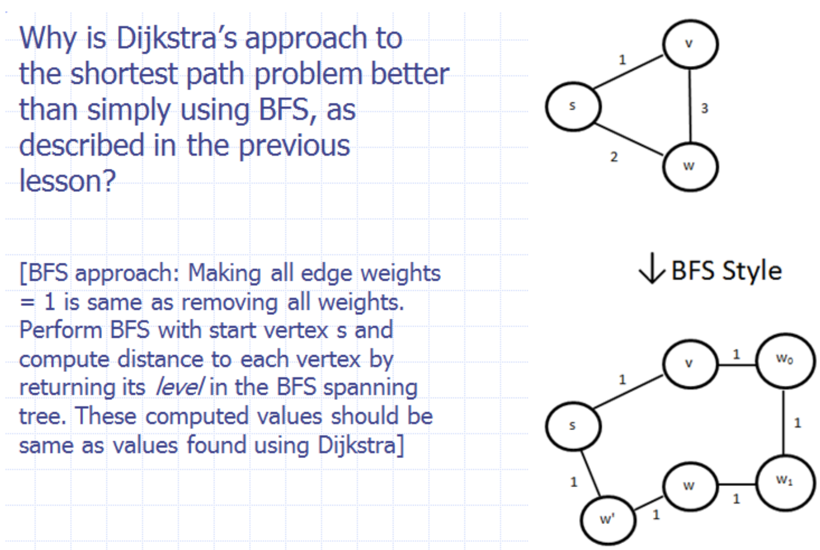
2. Points about Dijkstra's Algorithm

a. What is the shortest path from A to C in the graph below (using any algorithm you like)?



Solution: There is no shortest path because the edge CD can be traversed back and forth as many times as desired to create ever shorter (more negative length) paths.

b.



Solution: When edge weights on the original graph are large, the number of new vertices that need to be added causes performance of BFS to slow down. If the sum of the edge weights is as big as n^4, then the running time of the BFS version slows to $\Omega(n^4)$, which is worse than the Dijkstra running time $O(m\log n)$. However, if all edge weights are small, the BFS approach is faster.

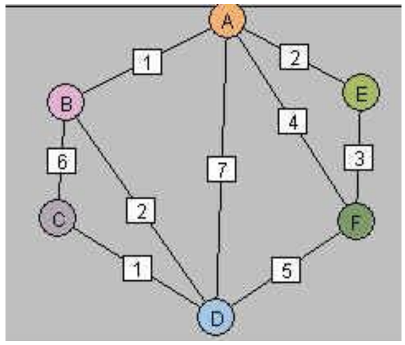
3. Describe an algorithm for deleting a key from a heap-based priority queue that runs in O(log n) time, where n is the number of nodes. (Hint: You may use auxiliary storage as the priority queue is built and maintained. Assume there are no two nodes have the same key.) This technique is needed for the optimized Dijkstra algorithm discussed in the slides.

Solution: Maintain a HashMap where the key of the HashMap is the key used in the priority queue, and the value is the node that contains this key. (This is where we use the assumption that no two nodes have the same key.) This HashMap must be kept synchronized with the priority queue after insert and removeMin operations, but this is easy to do. For instance, when a removeMin operation is done, pulling a node n from the top of the queue, we must also remove n and the key it contains from the HashMap (by looking up this entry in the HashMap using the key).

When we need to delete the key, use the key to look up the node in the HashMap. Then Replace the key with the key of the last node and do upheap or downheap if necessary. Delete last node. The running time of this process is O(log n).

Kruskal's Algorithm and Disjoint Sets

4. Carry out the steps of Kruskal's algorithm for the following weighted graph, using the tree-based DisjointSets data structure to represent clusters. Keep track of edges as they are added to T and show the state of representing trees through each iteration of the main while loop.



Solution: