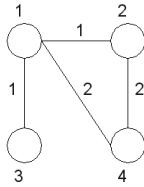


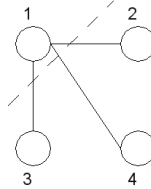
CS360 Assignment 6

21.1-2 Let $G(V, E)$ be a connected, undirected graph with real valued weights $w(u, v)$. Let A be a subset of E that is included in some MST for G , $(S, V - S)$ be any cut of G that respects A , and (u, v) be a safe edge for A that crosses $(S, V - S)$. Show that (u, v) is *not necessarily* a light edge. (Note: this is showing that the *converse* of Theorem 23.1 is *not* true.)

Consider the following graph:



Clearly an MST for this graph is:



If we let $S = \{1\}$ and $A = \emptyset$, then the cut $\{1\}, \{2, 3, 4\}$ trivially respects A . Furthermore, all three edges $(1, 2), (1, 3), (1, 4)$ are *safe* edges (since they are in the MST). However edge $(1, 4)$ is **not** a light edge and hence the converse of Theorem 23.1 is not true, i.e. light edge \Rightarrow safe edge, but safe edge $\not\Rightarrow$ light edge).

21.2-4 Suppose all edge weights are integers in the range $1 \rightarrow |V|$. How fast can Kruskal's algorithm be made to run? What if the weights are integers in the range $1 \rightarrow W$ for a *constant* W ?

The running time for a general implementation of Kruskal's algorithm consists of:

$$\begin{array}{rcl}
 O(V) & - & \text{Initialization} \\
 O(E \lg E) & - & \text{Edge sorting} \\
 + \quad O(E\alpha(V)) & - & \text{Edge processing} \\
 \hline
 & & O(E \lg E)
 \end{array}$$

Knowing the range of the weights allows for the use of *counting sort* to improve the time for the sorting step - which is the dominant term in the general implementation. If the weights are in the range $1 \rightarrow |V|$, sorting of the edges using counting sort becomes $O(n + k) = O(E + V)$. For connected graphs

$V = O(E)$ giving a sort time of $O(E)$. Hence the total running time becomes $O(V) + O(E) + O(E\alpha(V)) = O(E) + O(E) + O(E\alpha(V)) = O(E\alpha(V))$
 If the weights are in the range $1 \rightarrow W$ (where W is a *constant*), then sorting becomes $O(n + k) = O(E + W) = O(E)$ (same as above) which again gives a running time of $O(E\alpha(V))$.

21.2-5 Suppose all edge weights are integers in the range $1 \rightarrow |V|$. How fast can Prim's algorithm be made to run? What if the weights are integers in the range $1 \rightarrow W$ for a *constant* W ?

The running time for a general implementation of Prim's algorithm (assuming a Fibonacci heap implementation) consists of:

$$\begin{array}{r} VO(\lg V) \quad - \text{Extract-min for each vertex} \\ + EO(1) \quad - \text{Decrease key for each edge} \\ \hline O(V \lg V + E) \end{array}$$

Implementing the queue as an array of linked-lists with one element per weight value changes the running time of Extract-min() which now consists of *scanning* the array for the first non-empty list element. (Note: Decrease-key() remains $O(1)$ since it simply involves moving the element from one list to another.) If the weights range from $1 \rightarrow |V|$ then Extract-min() becomes $O(V)$ per scan giving a total time $O(V(V) + E) = O(V^2)$ which is *worse* than using a heap!

However if the weights range from $1 \rightarrow W$ (where W is a *constant*), then Extract-min() becomes $O(1)$ per scan giving a total time of $O(V(1) + E) = O(V + E) = O(E)$ (for a connected graph) which turns out to be the best we can do.

22.1-1 Run the Bellman-Ford algorithm on the directed graph of Fig 22.4 using the vertex z as the source, relaxing edges in the same order as in the figure, giving the d and π values after each pass. Change the weight of edge $(z, x) = 4$ and rerun the algorithm with s as the source.

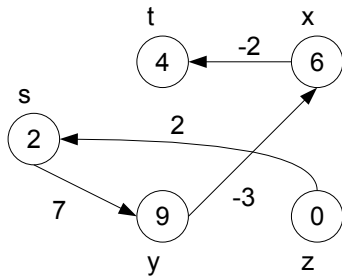
Considering edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$ gives the following table:

pass	d					π				
	s	t	x	y	z	s	t	x	y	z
0	∞	∞	∞	∞	0	/	/	/	/	/
1	2	8	7	9	0	z	s	z	s	/
2	2	5	6	9	0	z	x	y	s	/
3	2	4	6	9	0	z	x	y	s	/
4	2	4	6	9	0	z	x	y	s	/

Checking edges

$$\begin{aligned}
 d[x] = 6 &< d[t] = 4 + 5 \\
 d[y] = 9 &< d[t] = 4 + 8 \\
 d[z] = 0 &= d[t] = 4 - 4 * \\
 d[t] = 4 &= d[x] = 6 - 2 * \\
 d[x] = 6 &= d[y] = 9 - 3 * \\
 d[z] = 0 &< d[y] = 9 + 9 \\
 d[x] = 6 &< d[z] = 0 + 7 \\
 d[s] = 2 &= d[z] = 0 + 2 * \\
 d[t] = 4 &< d[s] = 2 + 6 \\
 d[y] = 9 &= d[s] = 2 + 7 *
 \end{aligned}$$

All edges are OK so routine returns *TRUE*. Note: All * edges (equalities) are ones that appear in the final graph shown below.



Setting edge $(z, x) = 4$ and rerunning the algorithm gives:

pass	d					π				
	s	t	x	y	z	s	t	x	y	z
0	0	∞	∞	∞	∞	/	/	/	/	/
1	0	6	∞	7	∞	/	s	/	s	/
2	0	6	(11)4	7	2	/	s	(t) y	s	t
3	0	2	4	7	2	/	x	y	s	t
4	0	2	2	7	-2	/	x	z	s	t

Checking edges

$$\begin{aligned}
 d[x] = 2 &< d[t] = 2 + 5 \\
 d[y] = 7 &< d[t] = 2 + 8 \\
 d[z] = -2 &= d[t] = 2 - 4 * \\
 d[t] = 2 &> d[x] = 2 - 2X
 \end{aligned}$$

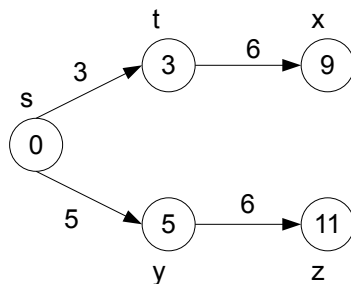
Check *fails* so routine returns *FALSE*. Note: $\langle x, t, z \rangle$ is a negative weight cycle.

22.3-1 Run Dijkstra's algorithm on the directed graph of Figure 22.2 first using vertex s then vertex z as source. Show the d and π values and the vertices in S after each iteration.

Using vertex s

u	d	π	u	d	π	u	d	π	u	d	π	u	d	π
s	0	/	t	3	s	y	5	s	x	9	t	z	11	y
Q			Q			Q			Q			Q		
key	u	π	key	u	π	key	u	π	key	u	π	key	u	π
0	s	/	3	t	s	5	y	s	9	x	t	11	z	y
∞	t	/	5	y	s	9	x	t	11	z	y			
∞	x	/	∞	x	/	∞	z	/						
∞	y	/	∞	z	/									
∞	z	/												

Giving the following graph:



Using vertex z

u	d	π	u	d	π	u	d	π	u	d	π	u	d	π
z	0	/	s	3	z	t	6	s	x	7	z	y	8	s
Q			Q			Q			Q			Q		
key	u	π	key	u	π	key	u	π	key	u	π	key	u	π
0	z	/	3	s	z	6	t	s	7	x	z	8	y	s
∞	s	/	7	x	z	7	x	z	8	y	s			
∞	t	/	∞	t	/	∞	y	/						
∞	x	/	∞	y	/									
∞	y	/												

Giving the following graph:

