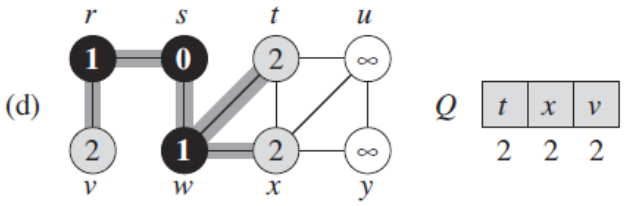
Assignment 5 Solutions

COT 6405 - Introduction to Theory of Algorithms

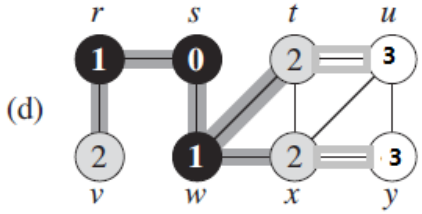
1. **Argue that in a breadth-first search, the value $u.d$ assigned to a vertex u is independent of the order in which the vertices appear in each adjacency list. Using Figure 22.3 as an example, show that the breadth-first tree computed by BFS can depend on the ordering within adjacency lists.**

As shown in the correctness proof of BFS, the algorithm ensures each u.d is the shortest distance from the source to u, so this value is independent of the specific route taken, if multiple routes would result in the same shortest distance for a particular u. However, the order of the adjacency list can affect the calculated tree in the case of a u with multiple shortest paths.

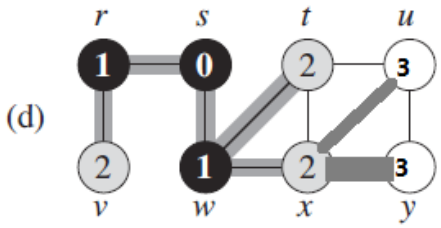


Take 22.3 d as an example.

When the queue Q has t, x and v, the tree is shown as the gray lines. If we process BFS according to the queue as text book, we could get same result as text book. As the picture bellows shows, u will be t's child. Y will be x's child.



If we change the queue order and use BFS to process x first, the while loop will choose both u and y as x's children. This is because BFS will choose all adjacent vertexes. Both of them have the same u.d value 3. And t will leave as a leaf without child since u is x's child now. The tree will be like:



Thus, the value u.d will not change if the ordering of vertex in adjacency lists. But the breadth-first tree computed by BFS could be different and depend on the ordering.

2. Show that using a single bit to store each vertex color suffices by arguing that the DFS procedure would produce the same result if line 8 of DFS-VISIT was removed.

1 bit (0 or 1) will be enough to store to the vertex color, like 0 for white,1 for grey.
Suppose we delete the line8, u.clolor = BLACK, so we have only Grey and White in the graph.
White means the vertex is initiated and not visited.
Grey means it is visited.

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

Line 6 in DFS and line 5 in DFS-VISIT check whether color is white. If it is grey, the vertex won't be processed. So even we remove line 8, **the black color is not used for making any decision**. Grey could do the same job as black. Thus, removing line 8 won't affect DFS algorithm behavior.

3. Show that edge (u, v) is
a. a tree edge or forward edge if and only if $u.d < v.d < v.f < u.f$,
b. a back edge if and only if $v.d \leq u.d < u.f \leq v.f$, and
c. a cross-edge if and only if $v.d < v.f < u.d < u.f$.

a. A tree edge, part of a depth-first search path , always follows a pattern of: A probe of its depth, and then a backing out. If edge (u,v) is on this tree edge path, u will be visited, then v, then potentially beyond; upon backing out, v will turn black, then u, then onward back to the source. Thus:

u.d < v.d < v.d < u.f

b. For a backwards edge, we must discover v first, then u, then finish u, then finish v. Thus, we arrive at v.d < u.d < u.d < v.f. However, self-loops are also backward edges, and in these cases, the node's discovery and finish times will of course be the same, since u and v are the same node. Thus:

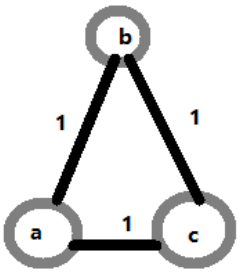
v.d ≤ u.d < u.f ≤ v.f

c. For an edge to be a cross edge, the destination, v, must be part of an already-explored search path. Thus, its discovery and finish times will always come before u's. Therefore:

v.d < v.f < u.d < u.f

4. Give a simple example of a connected graph such that the set of edges { $,v$: *there exists a cut* $S,V-S$ *such that* u,v *is a light edge crossing* $S,V-S$ } does not form a minimum spanning tree.

So Let's suppose we have a graph G={a,b,c} E = {ab=1,bc=1,ac=1} as shown here



So each edge could be a light edge crossing the cut ({a},{ b ,c}),({b},{ a c}),({c},{ b a}). It does not form a minimum spanning tree since no cycle is allowed.

5. Suppose that all edge weights in a graph are integers in the range from 1 to |V|. How fast can you make Kruskal's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?

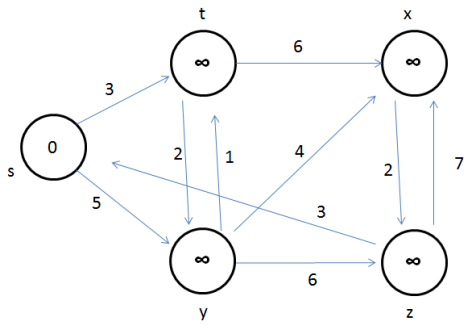
Kruskal's algorithm only depends on edge weights during sorting; therefore, we can derive a speedup when the values of the edge weights are in [1, W] for some constant W. Armed with this condition, we can use radix sort to sort the the edges in O(E) time; while our final efficiency is still O(ElgV) due to Make-Set(), we can be certain there will be some speedup.
In addition, when edge weights are in [1, |V|], we can use counting sort to sort the the edges in O(E + V) time; our final efficiency is again O(ElgV), but we can be certain there will be some speedup.

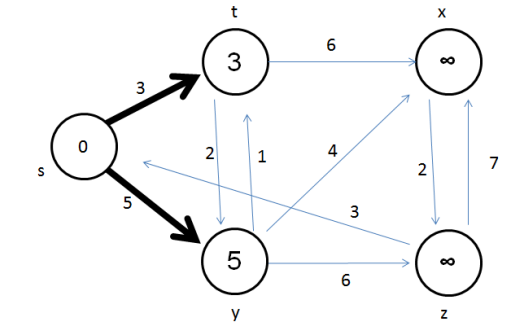
6. Suppose that all edge weights in a graph are integers in the range from 1 to |V|. How fast can you make Prim's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?

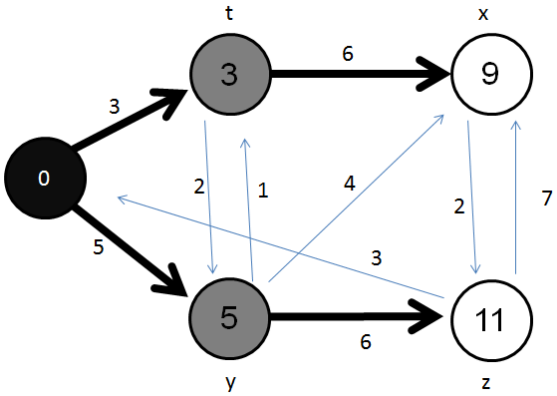
Please read the page636.
Build min heap O(V)
Extract-min O (VlgV)
Decrese-key O(ElgV)
1)Total=O(ElgV)
2)W is a constant, O(V(1)+E)=O(E)

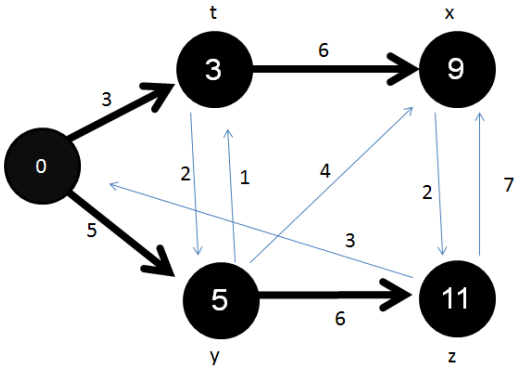
7. Run Dijkstra's algorithm on the directed graph of Figure 24.2, first using vertex s as the source and then using vertex z as the source. In the style of Figure 24.6, show the d and π values and the vertices in set S after each iteration of the while loop.

Start from s:

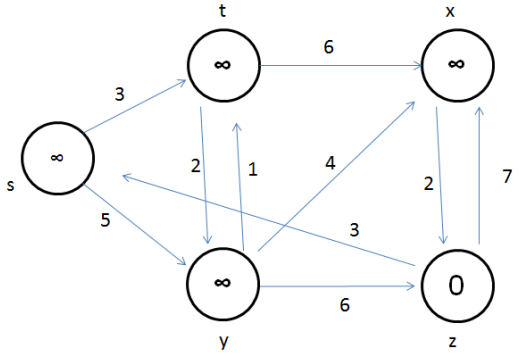


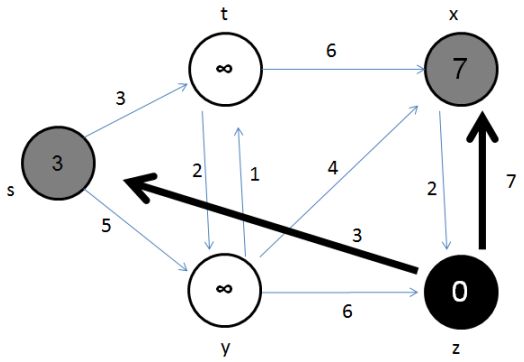

t
x
3
6
∞
3
0
2
1
4
2
7
s
5
3
5
6
∞
y
z

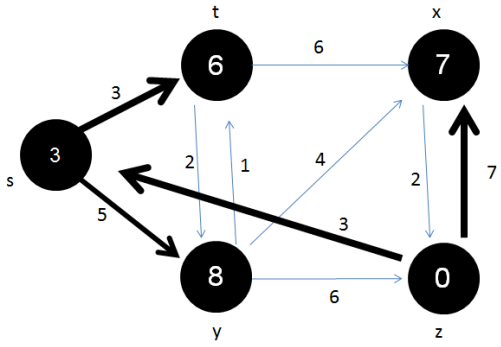

t
x
3
6
9
3
0
2
1
4
2
7
5
3
5
6
11
y
z


t
x
3
6
9
3
0
2
1
4
2
7
5
3
5
6
11
y
z

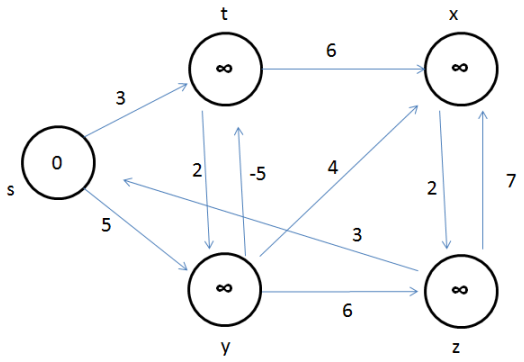
Start from z


t
x
∞
6
∞
3
∞
2
1
4
2
7
s
5
3
∞
6
0
y
z





8. Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers. Why doesn't the proof of Theorem 24.6 go through when negative-weight edges are allowed?



In this example, if y to t is -5, t will get 0 by update of y and y will be 5, but t to y is 2, so y could also be 2. The distance of y and t cannot be determined by the loop 2 and -5 of t to y and y to t.

We know that if there is a negative weight, the existing minimal distance could be update by pervious expanded vertex. This will cause trouble like the example we show. So the upper bound property only works when all the weights are non-negative.

10. Which is a more efficient way to determine the optimal number of multiplications in a matrix-chain multiplication problem: enumerating all the ways of parenthesizing the product and computing the number of multiplications for each, or running RECURSIVE-MATRIX-CHAIN? Justify your answer.

Running RECURSIVE-MATRIX-CHAIN is more efficient.

In textbook page 372, formula 15.6

Recurrence P(n) has a $\Omega(2^n)$ complexity to get all the multiplications. Every time, we computer the number requires O(n). Thus it would cost $O(n2^n)$ for the total time.

However, in page 385 and 386, the solution of RECURSIVE-MATRIX-CHAIN by subsituion method is proved as $\Omega(2^n)$

11. Determine an LCS of <1, 0, 0, 1, 0, 1, 0, 1> and <0, 1, 0, 1, 1, 0, 1, 1, 0>.

	y	0	1	0	1	1	0	1	1	0
X	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1	1
0	0	1	1	2	2	2	2	2	2	2
0	0	1	0	2	2	2	3	3	3	3
1	0	1	2	2	3	3	3	4	4	4
0	0	1	2	3	3	3	4	4	4	5
1	0	1	2	3	4	4	4	5	5	5
0	0	1	2	3	4	4	5	5	5	6
1	0	1	2	3	4	5	5	6	6	6

The algorithm runs like this.
So LCS is <1,0,0,1,1,0>, length 6