Solutions

October 8, 2015

22.1-1

Given an adjacency-list representation of a directed graph, how long does it take to compute the out-degree of every vertex? How long does it take to compute the in-degrees?

Solution

The out-degree of a vertex 'V' is equal to the length of its adjacency list i.e. Adj[V]. Also, the length of all the adjacency list in the Adj[] is equal to the number of edges i.e. $|E|$.Therefore, the time required to calculate all the out degree of every vertex is $\theta(|E| + |V|)$
The in-degree of a vertex 'V' is the number of times it appears in the adjacency list of other vertices. Hence to search the occurrence of a vertex in adjacency list, we need to search all the lists, i.e. all the edges $|E|$. So for all the vertices, time taken to find in-degree will be $\theta(|V||E|)$

22.1-3

The transpose of a directed graph G=(V,E) is the graph G^T=(V,E^T), where E^T={(v,u) $\in$ V x V:{u,v}$\in$E} Thus, G^T is G with all its edges reversed. Describe efficient algorithms for computing G^T from G, for both the adjacency-list and adjacency-matrix representations of G. Analyze the running times of your algorithms.

Solution

We consider two graphs G and G^T where, Adj[ ] represents adjacency list of Graph G whereas Adj"[ ] represents adjacency list of Graph G^T

Algorithm for changing adjacency lists of Graph:

1 procedure Transpose_list(G,G^T)

```
2     begin
3     for each vertex u = G^T.V
4          Adj"[u] = Nil
5     for each vertex u = G^T.V
6          for each v = G.Adj[u]
7          Adj"[v] = Adj"[v] union u
8     end
```

The time required will be $\theta(|V| + |E|)$

Now we consider two Adjacency matrices for G and G^T which are M and M' respectively.

Algorithm for changing adjacency matrix of Graph:

```
1     procedure Transpose_matrix(G,G^T)
2     begin
3     for each vertex u = G^T.V
4          for each vertex v = G^T.V
5               M'[u][v] = M[v][u]
6     end
```

The time required will be $\theta(|V^2|)$

22.3-2

Show how depth-first search works on the graph of Figure 22.6. Assume that the for loop of lines 5 - 7 of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finishing times for each vertex, and show the classification of each edge.

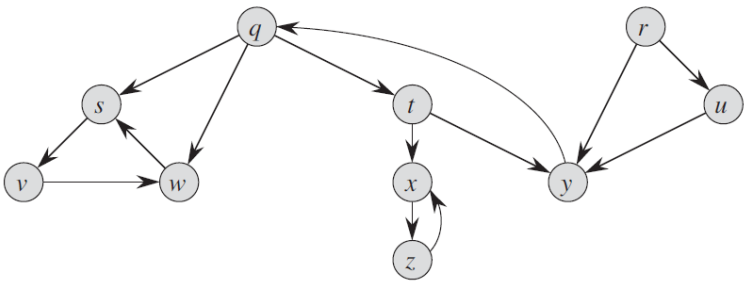

Figure 1: A directed graph for use.

Solution

G.V = { q, r, s, t, u, v, w, x, y, z}

Vertex q is fetched :
q.d = 1
q.color = Grey
Adj[q] = { s, t, w}

Vertex s is fetched :
s.π = q
s.d = 2
s.color = Grey
Adj[s] = { v }

Vertex v is fetched :
v.π = s
v.d = 3
v.color = Grey
Adj[v] = { w }

Vertex w is fetched :
w.π = v
w.d = 4
w.color = Grey
Adj[w] = { s }
Since all vertices in Adj[w] are visited, hence
w.color = Black
w.f = 5

Since all vertices in Adj[v] are visited,
v.color = Black
v.f = 6

Since all vertices in Adj[s] are visited,
s.color = Black
s.f = 7

Vertex t is fetched :
t.π = q
t.d = 8
t.color = Grey
Adj[t] = { x, y }

Vertex x is fetched :
x.π = t

x.d = 9
x.color = Grey
Adj[x] = { z }

Vertex z is fetched :
z.π = x
z.d = 10
z.color = Grey
Adj[z] = { x }
Since all vertices in Adj[z] are visited,
z.color = Black
z.f = 11

Since all vertices in Adj[x] are visited,
x.color = Black
x.f = 12

Vertex y is fetched :
y.π = t
y.d = 13
y.color = Grey
Adj[y] = { q }
Since all vertices in Adj[y] are visited,
y.color = Black
y.f = 14

Since all vertices in Adj[t] are visited,
t.color = Black
t.f = 15

Since all vertices in Adj[q] are visited,
q.color = Black
q.f = 16

Vertex r is fetched :
r.π = Nil
r.d = 17
r.color = Grey
Adj[r] = { u, y }

Vertex u is fetched :
u.π = r
u.d = 18
u.color = Grey
Adj[u] = { y }
Since all vertices in Adj[u] are visited,

u.color = Black
u.f = 19

Since all vertices in Adj[r] are visited,
r.color = Black
r.f = 20

Tree Edges = { (q, s), (s, v), (v, w), (q, t), (t, x), (x, z), (t, y), (r, u) }
Forward Edges = { (q, w) }
Back Edges = { (w, s), (y, q) (z, x) }
Cross Edges = { (r, y), (u, y) }

24.1-1

Run the Bellman-Ford algorithm on the directed graph of Figure 24.4, using vertex z as the source. In each pass, relax edges in the same order as in the figure, and show the d and π values after each pass. Now, change the weight of edge {z,x} to 4 and run the algorithm again, using s as the source.

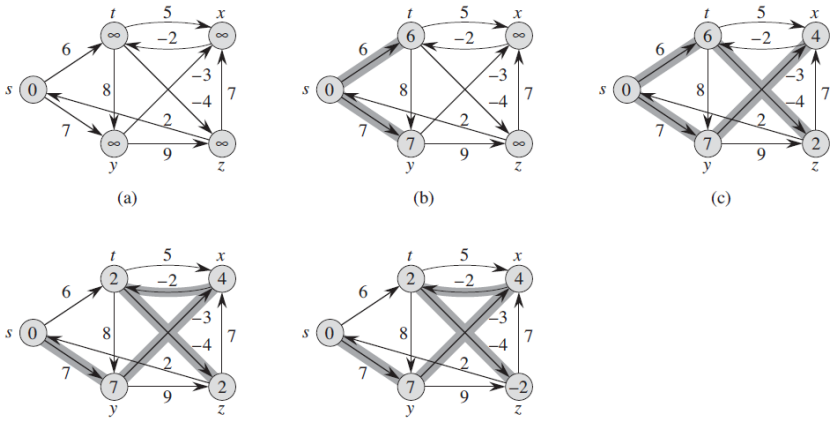

Figure 2: A directed graph for use.

Solution

Part 1: source = z

z.d = 0; s.d = x.d = y.d = t.d = ∞
s.π = x.π = y.π = t.π = z.π = Nil

G.E = {(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,s),(z,x)}
G.V = { z, x, y, t, s }

1st Loop:

Edge(z,s):
s.d = 2 ; s.π = z
Edge(z,x):
x.d = 7 ; x.π = z

2nd Loop:

Edge(s,t):
t.d = 8 ; t.π = s
Edge(s,y):
y.d = 9 ; x.π = s
Edge(x,t):
t.d = 5 ; t.π = x

3rd Loop:

Edge(y,x):
x.d = 6 ; x.π = y

4th Loop:

Edge(x,t):
t.d = 4 ; t.π = x

Returns TRUE
Hence Final Result:
z.d = 0 ; z.π = Nil
s.d = 2 ; s.π = z
y.d = 9 ; x.π = s
x.d = 6 ; x.π = y
t.d = 4 ; t.π = x

Part 2: source = s , Edge(z,x) = 4

s.d = 0; z.d = x.d = y.d = t.d = ∞
s.π = x.π = y.π = t.π = z.π = Nil

G.E = {(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,s),(z,x)}
G.V = { z, x, y, t, s }

1st Loop:

Edge(s,t):
t.d = 6 ; t.π = s
Edge(s,y):
y.d = 7 ; y.π = s

2nd Loop:

Edge(t,x):
x.d = 11 ; x.π = t
Edge(t,z):
z.d = 2 ; z.π = t
Edge(y,x):
x.d = 4 ; x.π = y

3rd Loop:

Edge(x,t):
t.d = 2 ; t.π = x

4th Loop:

Edge(t,z):
z.d = -2 ; z.π = t
Edge(z,x):
x.d = 2 ; x.π = z

The cyclic check condition returns False,
Since for Edge (x,t)
$t.d > x.d + w(x, z)$
$i.e. 2 > 2 - 2$
$2 > 0$
Hence False

24.3-1

Run Dijkstras 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.

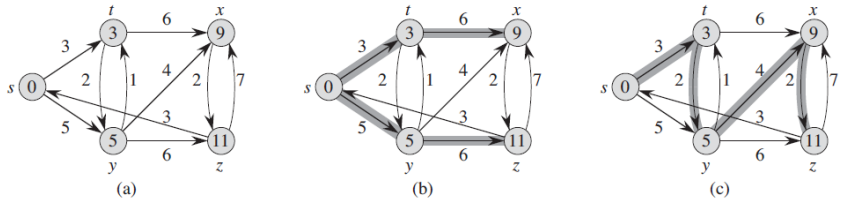

Figure 3: A weighted, directed graph

Solution

Part 1: source = s

S = Nil
Q = {s, y, t, x, z}
s.d = 0; y.d = z.d = t.d = x.d = ∞
s.π = x.π = y.π = t.π = z.π = Nil

1st Loop:
Vertex s is taken out from priority queue.
u = s , Q = { y, t, x, z }
S = {s}
t.d = 3 , t.π = s
y.d = 5 , y.π = s

2nd Loop:
Vertex t is taken out from priority queue.
u = t , Q = { y, x, z}
S = {s, t}
x.d = 9 , x.π = t

3rd Loop:
Vertex y is taken out from priority queue.
u = y , Q = { x, z }
S = { s, t, y }
z.d = 11 , z.π = y

4th Loop:
Vertex x is taken out from priority queue.
u = x , Q = { z }
S = { s, t, y, x }

5th Loop:
Vertex z is taken out from priority queue.
u = x , Q = Nil
S = { s, t, y, x, z }

Hence final result:
s.d = 0 , s.π = Nil
t.d = 3 , t.π = s
y.d = 5 , y.π = s
x.d = 9 , x.π = t
z.d = 11 , z.π = y

Part 2: source = z

S = Nil
Q = {s, y, t, x, z}
z.d = 0; y.d = s.d = t.d = x.d = ∞
s.π = x.π = y.π = t.π = z.π = Nil

1st Loop:
Vertex z is taken out from priority queue.
u = z , Q = { y, t, x, s }
S = { z }
x.d = 7 , x.π = z
s.d = 5 , s.π = z

2nd Loop:
Vertex s is taken out from priority queue.
u = t , Q = { y, x, t}
S = { z, s}
t.d = 6 , t.π = s
y.d = 8 , y.π = s

3rd Loop:
Vertex t is taken out from priority queue.
u = t , Q = { x, y }
S = { z, s, t }

4th Loop:
Vertex x is taken out from priority queue.
u = x , Q = { y }
S = { z, s, t, x }

5th Loop:
Vertex y is taken out from priority queue.
u = x , Q = Nil

S = { z, s, t, x, y }

Hence final result:
z.d = 0 , z.π = Nil
s.d = 3 , s.π = z
t.d = 6 , t.π = s
x.d = 7 , x.π = z
y.d = 8 , y.π = s