Elementary Graph Algorithms with Applications

Dr. Sachin Tripathi

Department of CSE,
Indian School of Mines, Dhanbad.

Outline...

1 Representations of Graphs

2 Depth First Search

3 Topological Sort

4 Strongly Connected Components

5 Single Source DAG Shortest Path Problem and Application

Adjacency-list and Adjacency-matrix Representation

Adjacency list

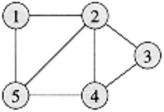
The adjacency-list representation of a graph G = (V, E) consists of an array Adj of $|V|$ lists, one for each vertex in V . For each u $\in$ V, the adjacency list Adj[u] contains all the vertices v such that there is an edge (u, v) $\in$ E.

Adjacency-matrix

The adjacency-matrix representation of a graph G = (V,E), we assume that the vertices are numbered 1,2, $\ldots, |V|$ in some arbitrary manner. Then the adjacency-matrix representation of a graph G consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

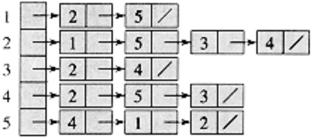
$$(a_{ij}) = \begin{cases} 1 & \text{if (i,j)} \in \text{E} \\ 0 & \text{otherwise.} \end{cases}$$

Representation of Undirected Graph



(a)

Graph



(b)

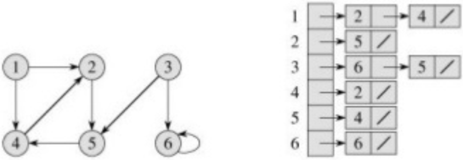
Adjacency list

$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

(c)

Adjacency matrix

Representation of Directed Graph



(a) (b)

graph Adjacency list

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

Adjacency matrix

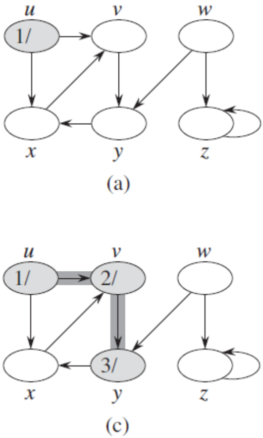
Depth First Search

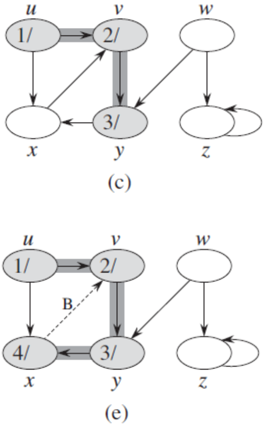
```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

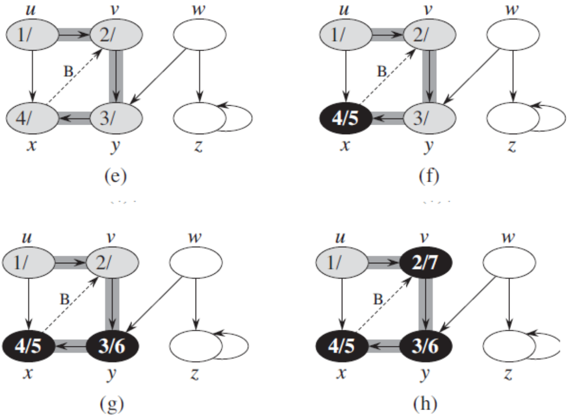
```
DFS-VISIT(G, u)
time = time + 1              // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]        // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK              // blacken u; it is finished
time = time + 1
u.f = time
```

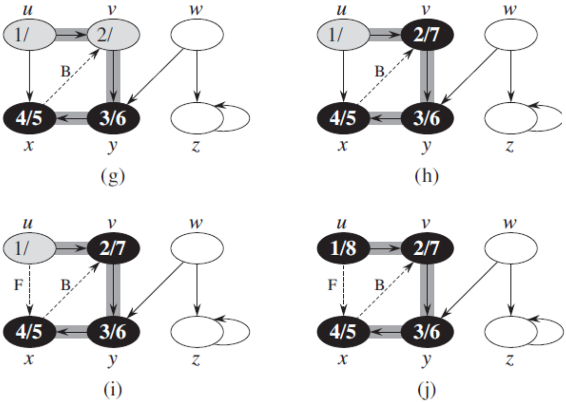
Run time complexity is $\theta(V + E)$

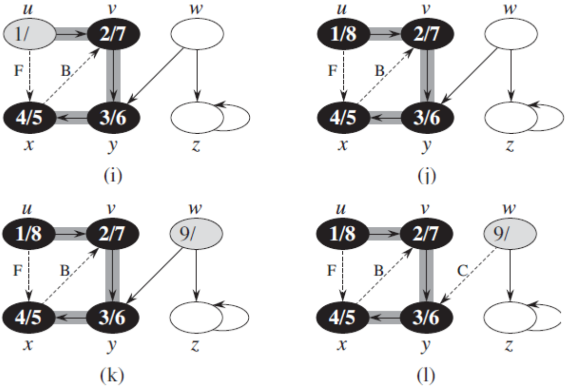
DFS Example

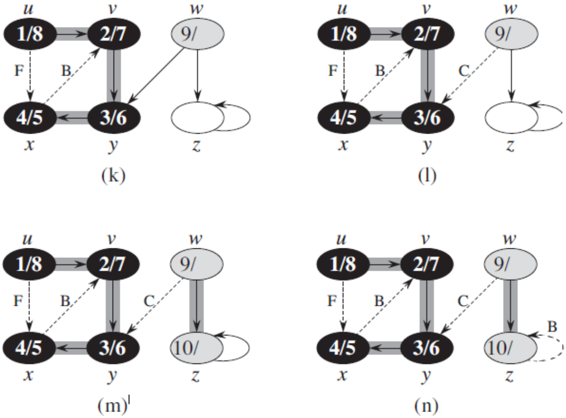


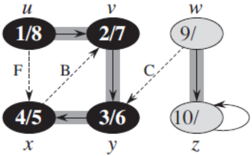

u
v
w
1/
2/
3/
x
y
z
(c)
u
v
w
1/
2/
4/
3/
x
y
z
(d)
u
v
w
1/
2/
B
4/
3/
x
y
z
(e)
u
v
w
1/
2/
B
4/5
3/
x
y
z
(f)


u
v
w
1/
2/
B
4/
3/
x
y
z
(e)
u
v
w
1/
2/
B
4/5
3/
x
y
z
(f)
u
v
w
1/
2/
B
4/5
3/6
x
y
z
(g)
u
v
w
1/
2/7
B
4/5
3/6
x
y
z
(h)

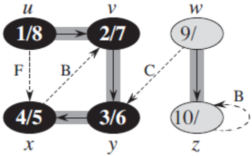

u v w
1/ 2/
B
4/5 3/6
x y z
(g)
u v w
1/ 2/7
B
4/5 3/6
x y z
(h)
u v w
1/ 2/7
F B
4/5 3/6
x y z
(i)
u v w
1/8 2/7
F B
4/5 3/6
x y z
(j)


u
v
w
1/
2/7
F
B
4/5
3/6
x
y
z
(i)
u
v
w
1/8
2/7
F
B
4/5
3/6
x
y
z
(j)
u
v
w
1/8
2/7
9/
F
B
4/5
3/6
x
y
z
(k)
u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
x
y
z
(l)

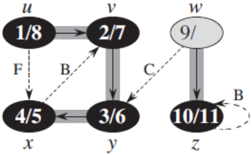

u
v
w
1/8
2/7
9/
F
B
4/5
3/6
x
y
z
(k)
C
(l)
10/
(m)
(n)


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
10/
x
y
z

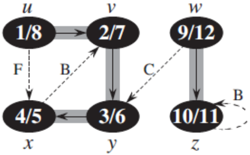
(m)


u
v
w
1/8
2/7
9/
F
B
C
B
4/5
3/6
10/
x
y
z

(n)


u
v
w
1/8
2/7
9/
F
B
C
B
4/5
3/6
10/11
x
y
z

(o)


u
v
w
1/8
2/7
9/12
F
B
C
B
4/5
3/6
10/11
x
y
z

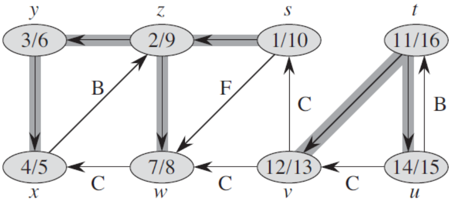
(p)

Properties of Depth First Search

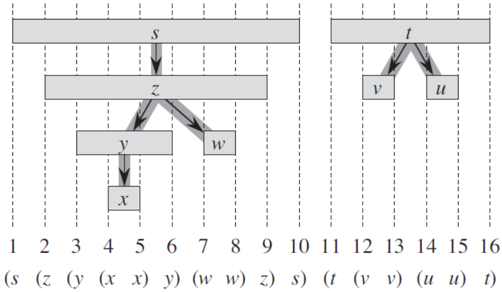
Depth-first search yields valuable information about the structure of a graph.

- DFS can be used to classify the edges of input graph $G = (V, E)$, This edge classification can be used to glean important information about a graph. For Example:- Directed graph is acyclic iff DFS search yields no back edges.
  Types of edges are Tree, Back, Forward and Cross
- If we represent the discovery of vertex u with a left parenthesis "$(u$" and represent its finishing by a right parenthesis "$u)$", then the history of discoveries and finishings makes a well-formed expression in the sense that the parentheses are properly nested. i.e.,

(a)

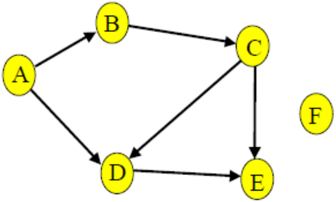

y
z
s
t
3/6
2/9
1/10
11/16
B
F
C
B
4/5
7/8
12/13
14/15
C
C
C
x
w
v
u

(b)


s
t
z
v
u
y
w
x
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)

Topological Sort

Topological sorting problem: given digraph $G = (V, E)$,
find a linear ordering of vertices such that:
for all edges (v, w) in E, v precedes w in the ordering

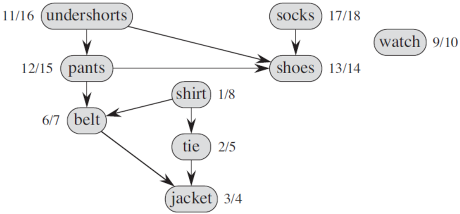


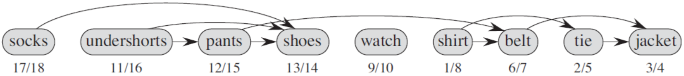
Topological Sort Algorithm

```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times v.f for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

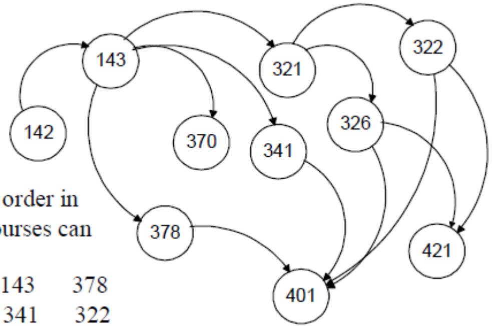
Time complexity is $\theta(V + E)$

Example 1:


11/16
undershorts
socks
17/18
watch
9/10
12/15
pants
shoes
13/14
shirt
1/8
6/7
belt
tie
2/5
jacket
3/4


socks
undershorts
pants
shoes
watch
shirt
belt
tie
jacket
17/18
11/16
12/15
13/14
9/10
1/8
6/7
2/5
3/4

Example 2:



Problem: Find an order in which all these courses can be taken.

Example: 142 143 378 370 321 341 322 326 421 401

How we Define ?

Strongly Connected Component

strongly connected component of a directed graph $G = (V, E)$ is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices u and v in C, we have both $u \to v$ and $v \to u$; that is, vertices u and v are reachable from each other.

Observations.

1: Graph and Transpose of a graph have exactly the same strongly connected components. i.e., u and v are reachable from each other in G, iff they are reachable from each other in G^T.

2: The finishing time of each source in DFS forest is always greater than their descendant.

Strongly Connected Components

BFS(G,s)

1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute G^T
3 call DFS(G^T) but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

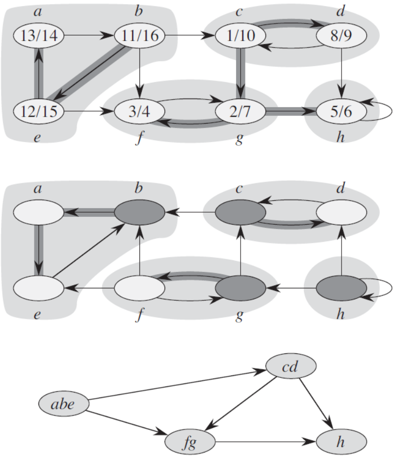
Time complexity is $\theta(V + E)$

Graph for Strongly Connected Components (Example 1)

(a)

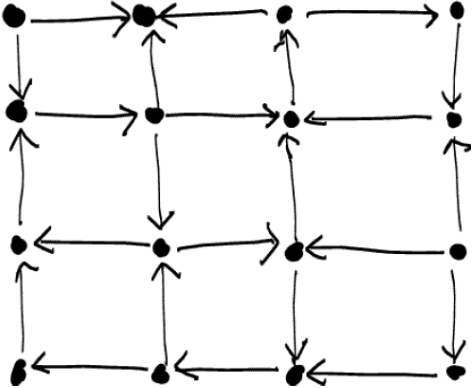
(b)

(c)



Example 2 (Tutorial)

Circle all of the strongly connected components of following directed graph.



Single Source DAG Shortest Path

By relaxing the edges of a weighted dag (directed acyclic graph) $G = (V, E)$ according to a topological sort of its vertices, we can compute shortest paths from a single source in $\theta(V + E)$ time. The algorithm for this is:

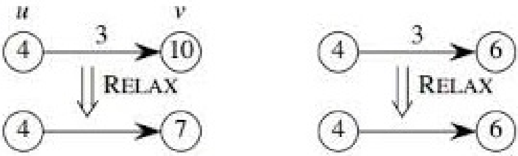
```
DAG-SHORTEST-PATHS(G, w, s)
1  topologically sort the vertices of G
2  INITIALIZE-SINGLE-SOURCE(G, s)
3  for each vertex u, taken in topologically sorted order
4      for each vertex v ∈ G.Adj[u]
5          RELAX(u, v, w)
```

INITIALIZE-SINGLE-SOURCE(G, s)

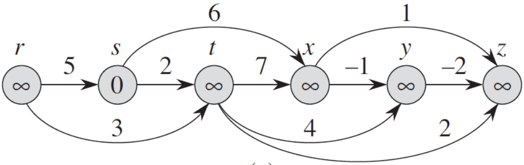
```
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

RELAX(u, v, w)

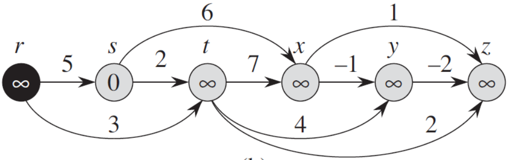
$$
\begin{array}{l}
\textbf{if } v.d > u.d + w(u, v) \\
\qquad v.d = u.d + w(u, v) \\
\qquad v.\pi = u
\end{array}
$$



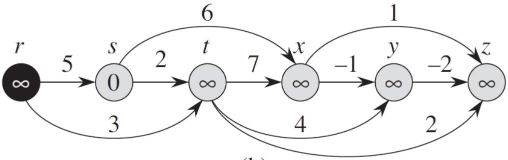
Consider The Graph where $'s'$ is the source



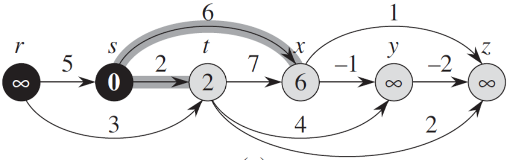
(a)



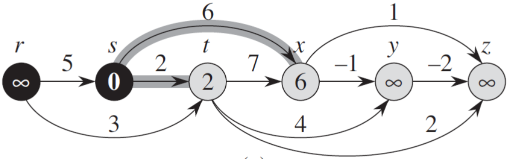
(b)



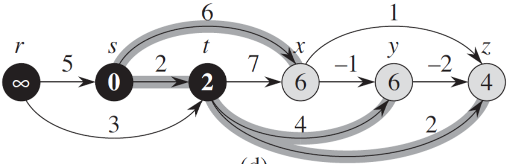
(b)



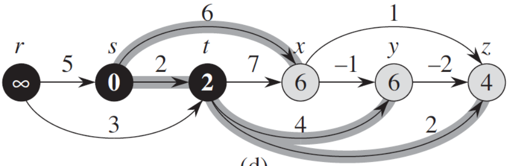
(c)



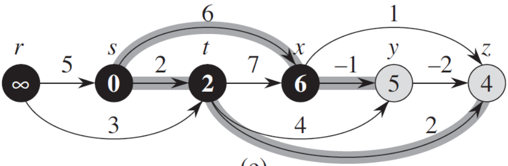
(c)



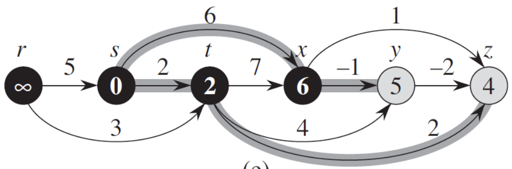
(d)



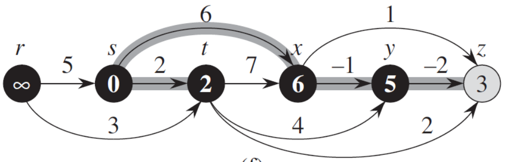
(d)



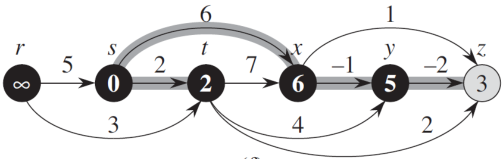
(e)



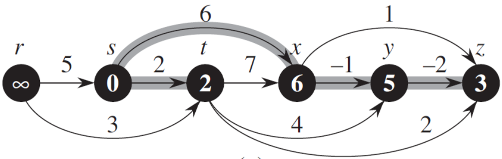
(e)



(f)



(f)



(g)

Application (Critical path in PERT chart analysis)

Critical Path: It is the longest path through the DAG corresponding to the longest time perform an ordered sequence of jobs.
It can be found either by:

- negating the edge weight and running DAG shortest path algorithm.
- Running DAG shortest path with the modification that replace ∞ to $-\infty$ in line two of the initialize single source algorithm and $>$ by $<$ in relax procedure.

Thank you