CSE 373 / L-24 / 19.05.2024 /

Quiz - 3

L-25 / 21.05.2024 /

* u.d = discovery time

  u.f = finish time

* Three type of edge :

  i. Forward edge ⇒ Predecessor to successor

  ii. Back edge ⇒ return to predecessor

  iii. Cross edge ⇒ No forward connection

* DFS Running Time

  - for initialize each vertex ⇒ $O(V)$

  - DFS - for loop ⇒ $O(V)$

  - DFS - VISIT ⇒ $\sum_{v \in V} |Adj[v]| = \Theta(E)$

  Run time = $O(V + E)$

  called $O(V)$ time

Parenthesis Structure

- Represent the discovery vertex with left parenthesis.
- finishing vertex with right parenthesis.

(u, u)

Any one condition needs to be hold. for u & v

i. $[u.d, u.f] \cap [v.d, v.f] = \emptyset$

- neither u nor v is descendant/successor of the other in DFF

ii. $[u.d, u.f] \subset [v.d, v.f]$

- u is descendant of v

iii. $[v.d, v.f] \subset [u.d, u.f]$ Slide-25

- v is descendant of u

Classification of edges

⇒ A directed graph is acyclic if and only if a depth-first search yields no 'back' edges.

Tree edges :

- in DFF
  - Edge (u,v) is a 'tree' edge if v was the first discovered by exploring edge (u,v)

* Color Code:

WHITE ⇒ tree edge

GRAY ⇒ back edge

BLACK ⇒ forward / cross edge

Slide - 29

* Topological Sort

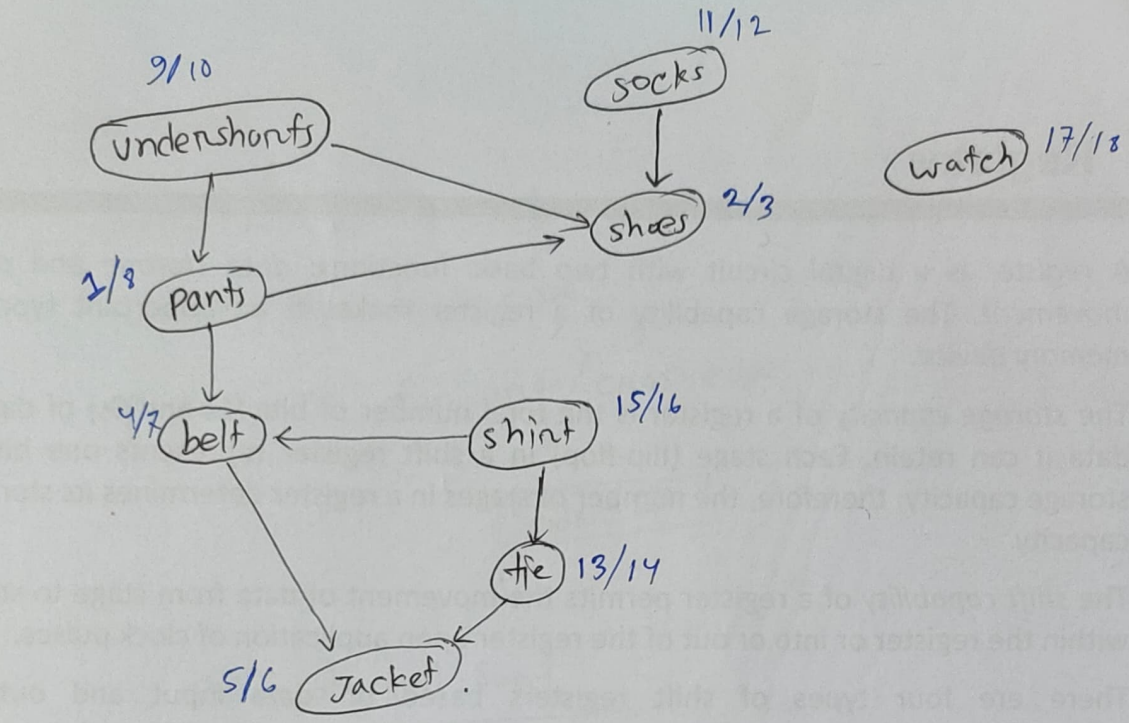
* Directed Acyclic Graph (DAG)
  - ↳ No cycle exist.
  - linear ordering
  - (u,v) is an edge, then u will appears before v in the order.
  - order along horizontal line so that all directed edge go from left to right.

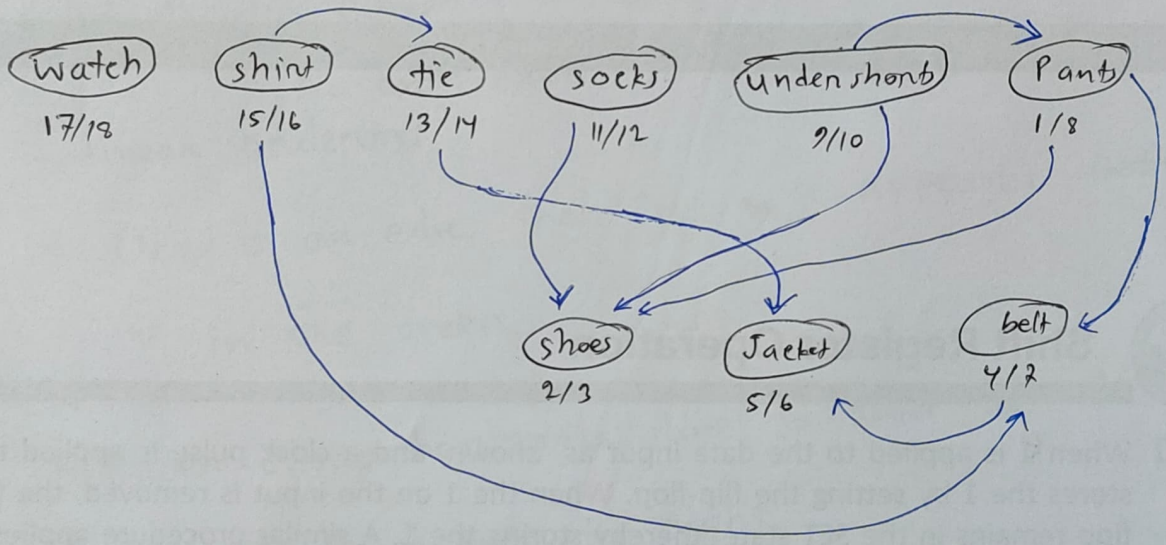
* Topological sort - algorithm.

TOPOLOGICAL-SORT(G) → $\Theta(V+E)$

$\Theta(V+E)$ ← call DFS(G) // compute finish time

$\Theta(1)$ { as each vertex is finished, insert it onto the front of a linked list.
return the linked list of vertices.





* Maybe this is the worst case.

* Strongly Connected Components

⇒ Set of vertices $C \subseteq V$ such that for every pair of vertices u and v in C, we have both $u \rightsquigarrow v$ and $v \rightsquigarrow u$.

⇒ Vertices u and v are reachable from each other.

Algorithm:

$\rightarrow \Theta(V+E)$

STRONGLY-CONNECTED-COMPONENTS(G)

call DFS(G)

compute G^T

call DFS(G^T) // in DFS consider the vertices in order of decreasing u.f

output the vertices of each tree in depth-first forest formed in line 3 as a separate strongly connected component.

Slide - 39,40

Question Pattern:

- apply number of u.d & u.f
- Draw edge
- write the sequence
- inverse the graph direction
- again apply number and start as decreasing order of u.f previously computed.
- Draw edge
- Write sequence
- show the strongly connected components.

Chapter-23

Minimum Spanning Tree (MST)

* All the vertices will be there
  - if two vertices are connected, they will be connected in the tree.
  - Some edge may not be there.
  - Sum of the weight is minimum.
  - Spanning tree can be different, but sum of the weight will be same.

L-26/26.05.2024

* There can be multiple spanning tree.
* Properties of MST
  - MST is not unique
  - MST has no cycle
  - No. of edges in MST
    $\Rightarrow |V| - 1$
* Safe ~~edges~~ ~~An~~ edge

Safe edge:

- An edge (u,v) is safe for A if and only if $A \cup \{(u,v)\}$ is also a subset of some MST.

Algorithm:

```
GENERIC-MST(G, w)
    A = ∅
    while A does not form a spanning tree
        find an edge (u,v) that is safe for A
        A = A ∪ {(u,v)}
    return A.
```

Kruskal's Algorithm

→ $O(E \lg E)$

```
MST-KRUSKAL(G, w)
    A = ∅
O(V) { for each vertex v ∈ G.V
         MAKE-SET(v)
O(E lg E) ← sort the edge of G.E into nondecreasing order by weight w
O(E) ← for each edge (u,v) ∈ G.E, taken in nondecreasing order by
                                                        weight
O(lg V) { if FIND-SET(u) ≠ FIND-SET(v)
            A = A ∪ {(u,v)}
            UNION(u,v)
    return A
```

Slide - 12-20

Kruskal's algorithm example:

- ✓ (g,h) = 1
- ✓ (c,i) = 2
- ✓ (f,g) = 2
- ✓ (a,b) = 4
- ✓ (c,f) = 4
- × (g,i) = 6
- ✓ (c,d) = 7
- × (h,i) = 7
- ✓ (a,h) = 8
- × (b,c) = 8
- ✓ (d,e) = 9
- × (e,f) = 10
- × (b,h) = 11
- × (d,f) = 14

For exam, do this part only

$A = \{(g,h), (c,i), (f,g), (a,b), (c,f), (c,d), (a,h), (d,e)\}$

a	b ×	c ×	d ×	e ✓	f ×	g ×	h ×	i ×
b		d			g	h		
c		f			h			
d		g						
e		h						
f		i						
g								
h								
i								

Question Pattern:

Do the step and draw the edge in the graph.

Prim's Algorithm

- The edges in set A always form a single tree.
  - starts from an arbitary root vertex r
  - At each step:
    - Find a light edge that connects A to an isolated vertex
    - Add this edge to A
    - Repeat until the tree spans all vertices.

Algorithm:

$O(V) + O(V \lg V + E \lg V) = O(E \lg V)$

```
MST-PRIM(G, w, r)
O(V)      for each u ∈ G.V
              u.key = ∞
              u.π = NIL
          r.key = 0
          Q = G.V
O(V)      while Q ≠ ∅
O(lgV)        u = EXTRACT-MIN(Q)
O(E)          for each v ∈ G.Adj[u]
                  if v ∈ Q and w(u,v) < v.key
                      v.π = u
O(lgV)                v.key = w(u,v)
```

Slide- 23 - 32

Chapter-24

Shortest Path Problem

* vertices = cities

  edges = road segment between cities

  edge weight = road distance

* Shortest-path weight from u to v:

$$\delta(u,v) = \min \begin{cases} w(p) : u \overset{p}{\leadsto} v & ; \text{if there exist a path from } u \text{ to } v \\ \infty & ; \text{otherwise} \end{cases}$$

* Variants of shortest path

  ⇒ Single Source shortest path
  - Source : Only one vertex
  - Destination: All other vertex

  ⇒ Single Destination shortest path
  - Source : All other vertex
  - Destination: Only one vertex

  ⇒ Single Pair shortest path
  - Source : Only one vertex
  - Destination: Another one vertex

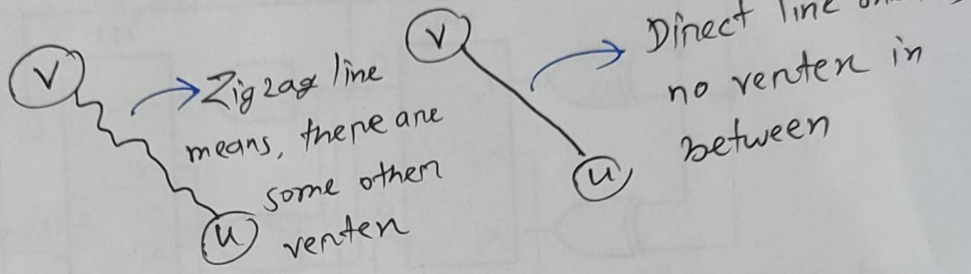
  ⇒ All pair shortest path
  - Source : Any one vertex
  - Destination: Any one vertex

  } For all possible pair.

Some Representation:

$v.d$ = distance from source

$v.\pi$ = predecessor



Initialization Algorithm:

INITIALIZE-SINGLE-SOURCE (G, s) ← source vertex

⇒ Must need for all the shortest path algorithm.

for each vertex $v \in G.V$

$v.d = \infty$

$v.\pi = NIL$

$s.d = 0$

$\Theta(V)$

Relaxation!

- Test whatever we can improve the shortest path to v found so far by going through u.

RELAX (u, v, w) ← weight function

u → media

v → destination

⇒ Source vertex not used here.

Relaxation Algorithm:

RELAX(u, v, w) → must need for all the single source shortest path algorithm

$\Theta(1)$

```
if v.d > u.d + w(u,v)   // That means improve possible
    v.d = u.d + w(u,v)  // improvement
    v.π = u             // update the predecessor
```

Dijkstra's Algorithm

⇒ Single Source - Shortest Path

⇒ No negative-weight edges

Repeatedly select a vertex $u \in V-S$ with the minimum shortest-path estimate $v.d$

- V → set of all vertex
- S → vertex already determined
- $V-S$ → set of vertex that need to check. maintain min priority queue

Algorithm:

DIJKSTRA(G, w, s) → $O(V \lg V + E \lg V) = O(E \lg V)$

INITIALIZE-SINGLE-SOURCE(G, s) → $\Theta(V)$

```
S = ∅
Q = G.V                  → O(V) ⇒ Build min-heap
while Q ≠ ∅              → O(V) times, executed
    u = EXTRACT-MIN(Q)   → O(lg V)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]  → executed O(E) times,
        RELAX(u, v, w)   → O(lg V)
```

Slide - 12

L-27 / 28.05.2024 /

✳ Negative-Weight Edge

⇒ There is a possibility to have infinitely many path.
Where distance is $-\infty$

✳ Bellman-Ford Algorithm

⇒ Single-source shortest path

⇒ Allows negative edge weights

⇒ RETURNs:
- TRUE ⇒ if no negative cycle are reachable from the source
- FALSE ⇒ if exist, means no shortest path exist. if $-\infty$.

✳ Algorithm:

BELLMAN-FORD(G,w,S) → O(VE)
  INITIALZZE-SINGLE-SOURCE(G,S) → Θ(V)
  for i = 1 to |G.V| - 1 → O(V)
    for each edge (u,v) ∈ G.E → O(E)
      RELAX(u,v,w)
  for each edge (u,v) ∈ G.E → O(E)
    if v.d > u.d + w(u,v)
      return FALSE
  return TRUE

O(VE)

Detecting Negative cycles

Slide - 20

✳ As per the definition of DAG, there is no cycle exist in the graph.

⇒ As a result, ~~No~~ Negative cycle also not exist in the DAG.

✳ DAG shortest path algorithm:

$\Theta(V+E)$

```
DAG-SHORTEST-PATHS (G, w, s)
    topologically sort the vertices of G        → Θ(V+E)
    INITIALIZE-SINGLE-SOURCE (G, s)             → Θ(V)
    for each vertex u, taken in topologically sorted order
        for each vertex v ∈ G.Adj[u]
            RELAX (u, v, w)
```

$\Theta(V+E)$

Slide-24

Chapter-25

All-Pairs Shortest Paths

⇒ Dijkstras algorithm:

- only applicable, when no negative edge.
- Need to run V times

⇒ $O(VE \log V)$

⇒ Bellman-Fond Algorithm

- Allows negative weight.
- need to run V times

$O(V^2E) = O(V^4)$

✳ Floyed - Warshall Algorithm

⇒ Dynamic Programing Solution

⇒ $O(V^3)$

✳ Adjacency matrix representation:

$$W_{ij} = \begin{cases} 0 & ; i = j \\ w(i,j) & ; i \neq j \text{ and } (i,j) \in E \\ INF & ; i \neq j \text{ and } (i,j) \notin E \end{cases}$$

Output matrix representation:

$D = d_{ij} = \delta(i,j)$

↳ hold the weight of shortest path.

Slide - 7 - 10

✳ In dynamic solutions:

$d^{(0)}_{45} = \infty$ → No ~~med~~ intermediate vertices

4 to 5

$d^{(1)}_{45}$ → Only one allowed

$d^{(3)}_{45}$ → Max 3 intermediate vertices allowed.

Slide - 13 - 15