DATE 11-02-19
PAGE

T-3

GRAPH ALGORITHMS:-

$G = \langle V, E \rangle$

$E = \{(u,v) \mid u \in V \& v \in V \& (u,v) \in R\}$

↓

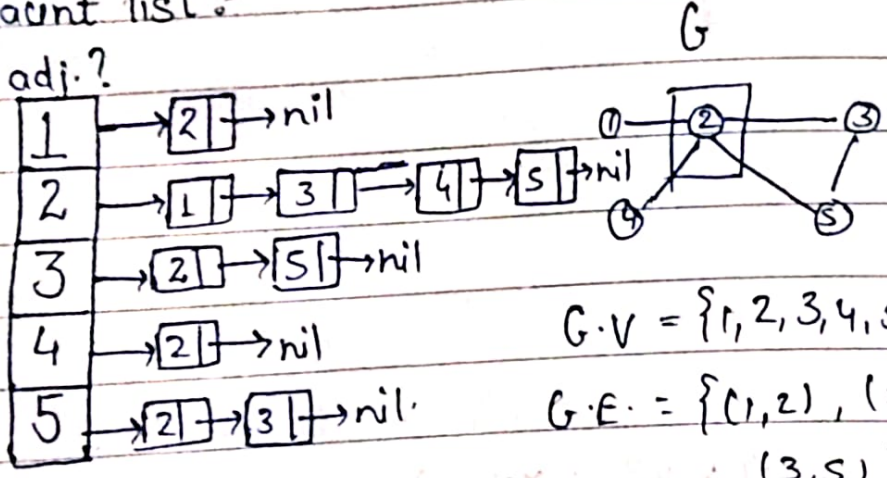
Relation.

if $\langle u,v \rangle$: Directed graph. →

(u,v), undirected graph. ——

Representation of graphs:-

1. Adjacent list:



$G.V = \{1,2,3,4,5\}$

$G.E = \{(1,2), (2,3), (2,4), (2,5), (3,5)\}$.

Space complexity $= |V| + 2|E|$

$= O(|V| + |E|)$

($|V| \rightarrow n$, $|E| \rightarrow m$)

Worst case $= O(n^2)$

Best case $= O(n)$.

if the graph is weighted.

then list would have a additional field.

key	weight	pointer

2. Adjacency matrix:

Adj [1...n] [1...n].

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

Adj [i] [j]
= 1 $(i,j) \in E$
= 0 $(i,j) \notin E$.

space : O(n) : (Base case, Worst case).

look up: O(1)

For weighted graph:-

Adj [i] [j] = w(i,j)

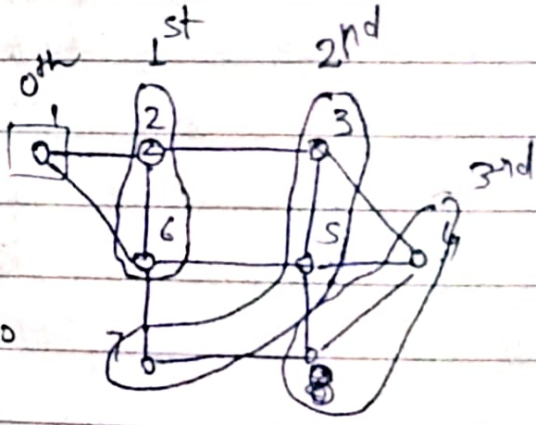
* Breadth first search:-

→ adjacency list.

BFS(G,s)
↳ start node

(π) produces tree

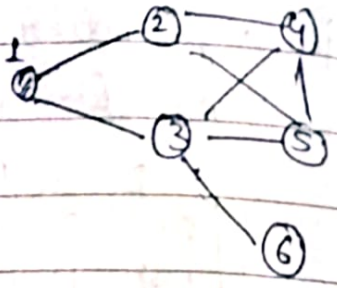
for every node u, set u.π to predecessor v such that path from s to u is predecessor tree has v as u's previous node.



Breadth first tree construction.

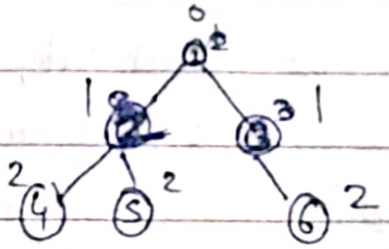
u.d = shortest distance from s to u in the graph.

"black" node
- expanded.
"gray" nodes
- discovered
not yet expanded
"white" undiscovered.




```
for each node u in G.V
        u.colour = white
        u.d = INF
        u.π = nil.
S.colour = gray
S.d = 0.

Q = ∅ {S}
while (Q ≠ ∅)
    u = dequeue(Q)
    for each node v ∈ Adj(u)
        if (v.colour = white)
            v.d = u.d + 1
            v.π = u
            v.colour = gray
            enque(Q, v)
    u.colour = black.
```



12-03-19 Graph Algorithms:

- Complexity analysis - BFS.
- DFS - Depth first Search.

Queue

$|V| = n$ n times { for each v ($n-1$) adjacent to u.

$\rightarrow O(n + 2|E|)$

$|E| = m$

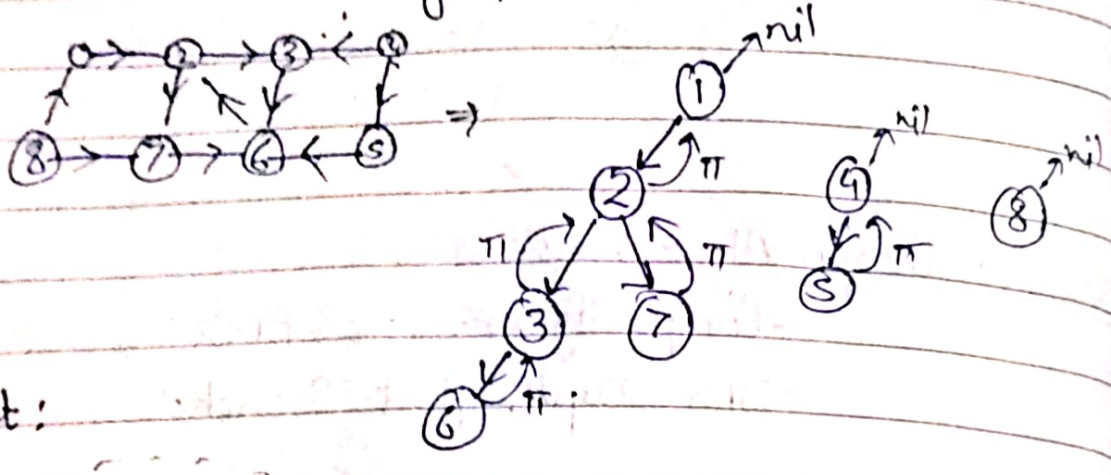
$= O(n+m)$

$\delta(u, v)$: Shortest distance from u to v.

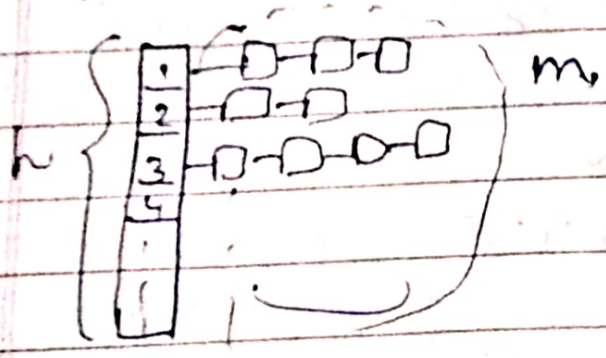
Claim: At the end of the BFS(G, s) For every node v, $v.d = (s, d)$.

Success
DATE
PAGE

Depth First Search: We shall take a Directed graph as the case.



Adj. list:



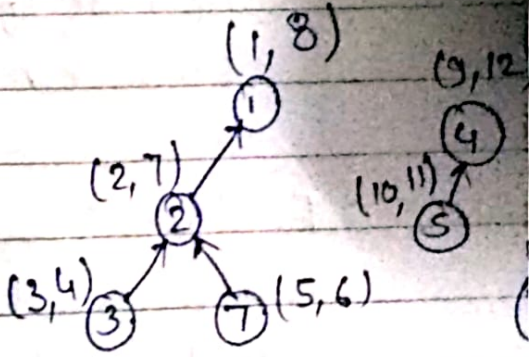
attribute π predecessor.
start & finish time.
colour: white, grey or black.
v.d : discovery time / start time.
v.f : finish time.

DFS(G): G is directed graph whose adjancy list is given.

```
for each vertex u in G.V
    v.colour = white
    v.π = nil
    v.d = ∞
    v.f = ∞
time = 0 // Global variable.
for each vertex u in G.V
    if (u.colour == white)
        dfs-visit(G, u).

dfs visit (G, u)
    time = time + 1
    u.d = time
    u.colour = gray
    for each v ∈ adjacency list(u)
        if (v.colour == white)
            v.π = u
            dfs visit (G, v).
    u.colour = black.
    time = time + 1
    u.f = time.
```

time complexity:
- list: O(n+m)
- matrix O(n+n²)



DFS forest:

predecessor subgraph of G.

$E_p = (v.\pi, v)$ if $v.\pi \neq nil$.

AND

$(v.\pi, v) \in E$.

$V_p = V$.

Edge Classification:

Tree Edges: i.e. those in the DFS forest.

$\langle v.\pi, v \rangle$

DFS - some properties.

BFS - Correctness.

Dijkstra's shortest path.

Predecessor graph:

V_π = set of vertices in G.

$E_\pi = \{v.\pi, v \mid v.\pi \neq NIL\}$.

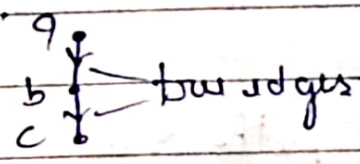
If (u, v):

u.d & u.f

v.d & v.f

) $u.d < v.d < v.f < u.f$ {( )}
) $u.d < u.f < v.d < v.f$ ( ) { }
) $v.d < v.f < u.d < u.f$ { } ( )

} Parenthesization theorem.



Tree edges: The edges of G.E that become edges in the DFS forest.
Back edges: $\langle u, v \rangle$ where u belongs to surface of v in the DFS forest.
) Forward Edge: $\langle u, v \rangle$ in G.E. is such that v belongs to subtree rooted at u in the DFS forest.
Cross Edge: Any edge not in the first 3 sets.

Can all types of edges arise in an undirected graph?
No cross edges
No forward edges.

"v.colour = white" : Tree edge.
"v.colour == gray" : back edge
v.colour == black AND
$u.d < v.d$
→ forwarded.

v.colour = = black AND
v.f < u.d
→ Crossid edge.

18-03-19 BFS - correctness & shortest path
Dijkstra's shortest Path algorithm.

$\delta(s,u)$ = Shortest distance from s to u.

1) if (u,v) is an edge
$\delta(s,v) \leq \delta(s,u) + 1$
shortest path to v may thru u, or a shorter path.

2) $v.d \geq \delta(s,v)$
(i.e. an upper bound on $\delta(s,v)$ is given by $v.d$).
Let, u be $v.\pi$
$v.d = u.d + 1$.
Base case: d=0, only s has h=0.
~~sd~~
$s.d = 0 \geq \delta(s,s)$
Assume it is true upto some d value, $u.d$.
Then,
$v.d = u.d + 1$
$\geq \delta(s, u) + 1$
$\geq \delta(s,v)$ by ① & ②.

Contents of the queue.

① Only grey nodes.

for any node v in the queue (except s)

$v.\pi$ is non null

$v.d$ is not ∞.

a) $\forall i \; v_i.d \leq v_{i+1}.d$

b) if $v_i.d < v_{i+1}.d$ then

$v_i.d+1 = v_{i+1}.d$

How many different d values can be there for nodes in Q, at a time at the most 2.

CLAIM

$(v_1 \cdots v_i)_{v_1.d} \; (v_{i+1}.d \cdots v_r)_{v_1.d+1}$.

Base : $Q = \{s\}$

$v_1.d = 0 \rightarrow$ satisfies

Induction step:

Assume true upto some value of $d = d$.

$0, 1, 2, \cdots, d$.

$\{ v_1 \cdots v_i \; v_{i+1} \cdots v_r \}$

$d-1$ d

$\Downarrow$

$\{ v_2 \cdots v_i \; v_{i+1} \cdots v_r \, v_{r+1} \cdots v_n \}$.

When any node u turns black, it would have
$v.d = \delta(s, v)$
$v.\pi$ = (predecessor) u, along a shortest path from s to v.

Nodes turn black only in queue order:
increasing d value order
i.e. all nodes with $v.d = d$ turn black before those with d+1.

Assume this to be true upto some d value, say u.d. i.e. all nodes turning black with d=u.d or less would satisfy ① and ②.

From that point, let v be a node with v.d set to (u.d + 1)
It must have been assigned by adjacency list some predecessor node u's exploration
$[u.d = \delta(s, u)]$ by assumption in induction step.
We claim that u is predecessor of v along a shortest path & $u.d + 1 = v.d$ is $\delta(s, v)$.

Suppose shortest path $\delta(s,v) < v.d$ $(u.d+1)$ assume that w is its predecessor in that path.

$$\delta(s,v) = 1 + \delta(s,w) < 1 + \delta(s,u).$$

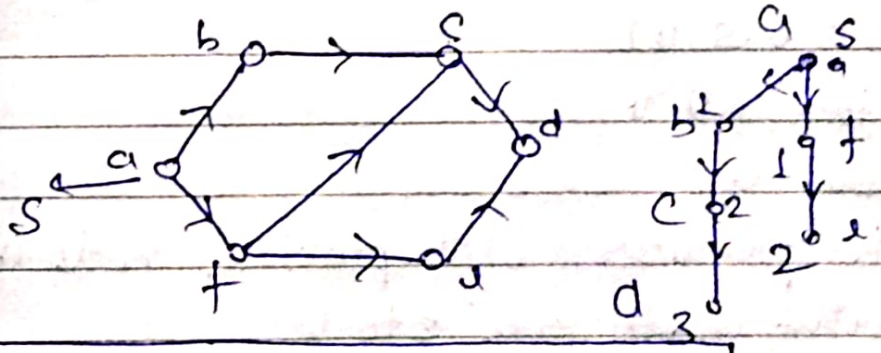
$$\delta(s,w) < \delta(s,u) \Rightarrow w.d < u.d$$

$\Rightarrow w$ inserted & left Q before u.

$\Rightarrow v$ was white when w left Q.

i.e. v must have turned grey before u was expanded $\because v \in Adj.(w)$

But, v turned grey only from u's adj. list
↳ contradiction.



Weighted Graph:

$\downarrow$ $w: E \to R^+$

Directed graph $G = \langle V, E \rangle$.

Dijkstra's shortest path algorithm.

DATE: 19-03-19 Success
PAGE:

- Dijkstra's shortest path algorithm
- Spanning tree
  - Prim's minimum spanning Tree.

Minimum spanning Tree algorithms
Dijkstra's shortest

$v.d = \delta(s, v)$
- proved yesterday.

s ⟶ source vertex.

Dijkstras: sp (G, w) ← adjacency list format

$G = \langle V, E \rangle$

$w: E \rightarrow R^{+}$.

Output

$v.d = \delta(s, v)$
for all v

$v.\pi = u$

u is predecessor, i.e. previous node in a shortest path from s to v.

for each u in G.V.

$u.d = \infty$

$u.\pi = NIL$

u.colour = white

$s.d = 0$

s.colour = grey

```
insert (PQ, s)
while (PQ ≠ ∅)
u = extractmin (PQ) → u.colour = black.
for each v ∈ Adj(u)
    if (v.colour == white)
        v.π = u
        v.d = u.d + w(u,v)
        v.colour = grey
        insert (PQ, v).
    else if (v.colour == grey)
        if (V.d > u.d + w(u,v))
            v.d = u.d + w(u,v)
            v.π = u
```

If key = d. field

decrease key operation

→ Relax(u,v,w)

When a node is extracted $v.d = \delta(s,v)$.

Complexity = $O(|V| + |E| \cdot \log n)$

($|V|$ → n, $|E|$ → m)

$\log n$ → Decrease key operation.

List

$O(n + n\log n + m) = O(n\log n + m)$.

- n → initialization
- $n\log n$ → extract min
- m → for each $v \in Adj$

matrix:

$O(n + n\log n + n^2)$

$= O(n^2)$.

① Initialize single source (G, w, s):

Just to use again & again:

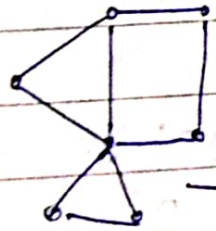
for each u in G.V
    u.d = ∞
    u.π = NIL
    u.color = white
    s.d = 0
    s.color = grey

② relax (u, v, w) # relax by edge (u, v)

if v.d > u.d + w(u, v)
    u.π = u
    v.d = u.d + w(u, v)

Undirected graph G:

free tree → connected, acyclic, undirected.



→ subgraph that are trees on G.V.

$w : E \to R$

$$w(B) = \sum_{e \in B} w(e)$$

↳ set of edges.

$$w(T) = \sum_{e \in T} w(e)$$

Minimum weighted spanning tree of a given connected graph:

↳ covers all the vertices.

20-03-19

Minimum Spanning Tree Algorithms:

- Prim's
- Kruskal's.

Input: undirected, connected, weighted graph G G, w

$w: E \to R$

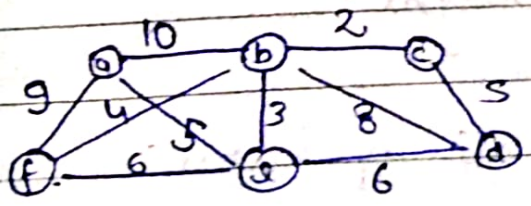
Spanning Tree: a tree that is a subgraph of G including all the n vertices of G.

(set of n-1 edges).

Minimum SpTree:

↳ A spanning tree with minimum weight of all spanning trees.

$$w(T) = \sum_{e \in T} w(e).$$



Cut: Partition of the vertices into 2 disjoint sets.

V:

Cut $(S, V-S)$

set of vertices

Edges crossing a cut:

(u, v) such that

$u \in S$ and $v \in V-S$

or

$u \in V-S$ and $v \in S$.

Eg: {a, b, c} {d, e, f}

eg: ad, bf crosses the cut.

Set of edges E'; cut respecting E': no edges in E' crosses the cut.

$E' = \{(b, c), (a, b), (e, f)\}$.

$\cup \{(G, s)\}$

⇒ does not respect the set of edges.

* Many edges may cross the cut, the edge with smallest wt.: "light weight".

* X: set of edges, a subset of a tree.

SAFE EDGES: A edge of G, not in X, such that it does not create a cycle added to X.

Any edge that creates a cycle with X, is NOT SAFE.

PRIM'S ALGORITHMS:-

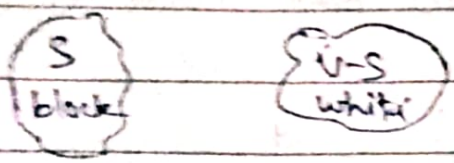
```
prims-mst(G, s)
    T = ∅ # built out MST into T.

prims_mst (G, s)
    for all u ∈ G.V
        u.π = NIL
        u.d = INF
        u.colour = white → insert(PQ, u).
    s.colour = grey
    s.d = 0.
    insert (PQ, s)
    while (PQ ≠ ∅)
        u = extract_min (PQ)
        u.colour = black.
        for each v in adjm. list of u.
            if (v.colour ≠ black)
                if (w(u,v) < v.d)
                    decrkey (v, w(u,v))
                    v.π = u.
```

Success
DATE .....
PAGE .....

Proof of "that the Spanning tree is least weighted"

The edges in S form a part minimum spanning tree.



Assume, at any point of time, the black set consists of edges that are part of MST, (& white set no edges).

```
* Kruskals (G, w)
      T = ∅  # set of edges
      sort the edges by weight
      For every vertex u in G
            makeset (u)
Repeat,
      each edge in the sorted list
            (u,v)
      if find(u) ≠ find(v)
            T = T ∪ {u, v}
      return(T)
      until |T| = n-1
      return T.
```

Prims : $O(m + n \log n)$

Kruskals : $O(m \log m + n + m)$

$= O(m \log m + n)$.

26-03-19 Graph Algorithms:

- Kruskals min. spanning Tree

Insertion in RB Trees.

Kruskals_min_sp_tree (G, w) → $\langle V, E \rangle$. → $w : E \to R$

sp.tree edges

$\leftarrow A = \phi$

sort all edges in ascending weight

cuts for every vertex. i.e., in V

makeset (u)

for each edge (x, y) in sorted edge list

if (findset (x) ≠ findset (y))

$A = A \cup (x, y)$

union (x, y).

~~Init~~ Until $(n-1)$ edges are added.

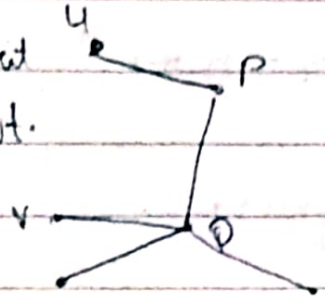
Cuts

light edge, crossing

- safe - cycle

Basi: Suppose MST, M does not have (u, v)

(u, v) that ∴ has min wt.



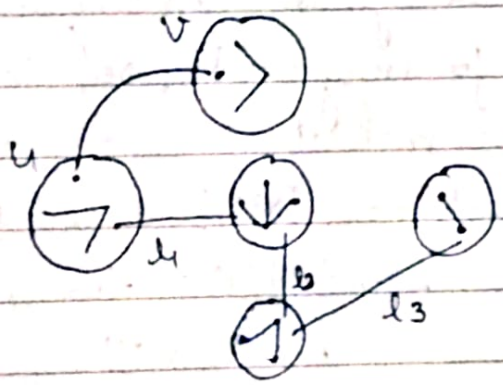
Construct:

$M' = M - \{(p, q)\} \cup \{u, v\}$

$w(M') - w(M) \leq 0$

$\Rightarrow M'$ is minimum spanning tree.

Let A be a forest which is a subset of MST, M. If we add (u, v) the lightest edge 'Safe edge' into A, it still remains the subset of a Minimum spanning - Tree.



Suppose $(u, v) \notin M$. In M there must be a path from u, v. $(l_1, l_2, l_3, l_4, \cdots, l_k)$ are not part of A & their weights are $\geq w(u, v)$

remove one of them & put in (u,b).
So, $M' = M - \{[a_i, b_i]\} \cup \{(u,b)\}$

$w(M') \leq w(M)$.

RB Tree (AGAIN):-

rbinsert (T, x) — red node

#input: root r of a subtree, which is a perfect rb tree except possible violation of prop 2 (i.e. root or root and x is red)

#Output: x is root of a subtree of T ∪ {x}, it is a perfect red black tree except, possible violation of prop. 2, and rule 4 and its child nodes of y are perfect rb trees with possible rotation of rule 2, i.e. they may red. Also black height of original parent of original x is conserved.

```
rbinsert(T, x)
    if (x == NIL) return(x).
    else
        if (x.key < T.key)
            T.left = rbinsert(T.left, x).
```

adjust x.left to preserve rb properties (except 2).

27-03-19

Red-Black Insertion:
Shortest Path- Bellman Ford
Algorithms:

```
#if x is black, simply return.
if (x.left.colour == red)
    if (x.left.right.colour=Red || x.left.left.colour
        if (x.right.colour == red)
          x.right.colour = black
          x.left.colour = black
          x.colour = red.
          #x is ready for return.
        else
          if (x.left.right.colour == red)
              x.left = left rotate (x.left)
          y = x
          x = right rotate(y)
          x.left.left = black;
```

DATE 28-03-17

Shortest path Algorithms

- Bellman Ford
- Floyd Warshal

given: Graph $G = \langle V, E \rangle$ & $w : E \to R$

vertices i, j.

Find: $\delta(i,j)$

$p = (v_0, v_1, v_2, \ldots, v_k)$ where $v_0 = i$, $v_k = j$

$(v_x \; v_{x+1}) \in E$

is a path from i to j

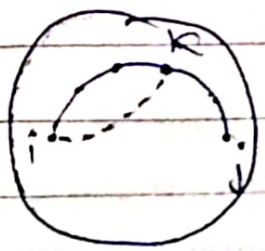
$$w(p) = \sum_{(v_x, v_{x+1}) \in p} w(v_x \; v_{x+1})$$

weight of shortest path between i & j : $\delta(i,j)$

$\delta(i,j) = \min \{ w(p) : p$ is a path from i to $j \}$.

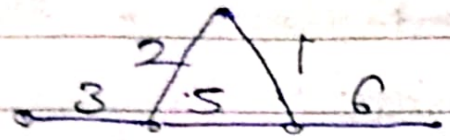
* Optional substructure property: (In this problem)

If a shortest path between i & j contains the vertex k, the path also contains the shortest path from i to k and k to j.



Can we have a cycle in a shortest path?

Positive weights



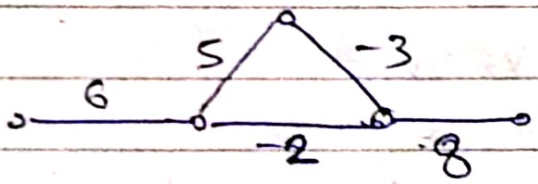
→ No cycles will be taken.

Negative weights:

Negative ~~cycles~~ Cycles:

→ Undefined.

Negative weights but no negative cycles



No cycles in shortest path.

Initialize-single-source (G, w, s)

for all u in V

u.d = ∞

u.π = NIL

s.d = 0.

Success
DATE
PAGE

Relax (u, v, w)
    if (v.d > u.d + w(u,v))
        v.d = u.d + w(u,v);
        v.π = u.

Bellman Ford (G, w, s)
    # if G has -ive cycle returns false
    else it find shortest path from s to all vertices (in their v.d, & sets v.π to predecessor)

    Initialize single source (G, w, s)
    for i=1 to |V|-1
        for each (u, v)
            relax (u, v, w)
    for each edge (u, v)
        if (v.d > u.d + w(u,v))
            print "false";

Time complexity: O(n.m) ⟶ O(n³).

DATE: 29-03-19 Success
PAGE:

Shortest Path:

- All pairs → Floyd Warshal's
  - - Transitive closure of a directed graph

[-ive wts $w: E \to R$ (no -ive cycles) ⟶ Bellman Ford $O(n^2m)$ ⟶ All pairs mis $O(n^2) : O(n^4)$

$E \to R^+$, Dijkstra $O(n(m+n\log n)) = O(nm+n^2\log n)$

If m is $O(n^2)$ $O(n^3)$

$$D^0_{ij} = G_w = g_{ij} = \begin{cases} \text{if } \langle i,j\rangle \in E \text{ then } w(i,j) \\ i=j \text{ then } 0 \\ \langle i,j\rangle \notin E \text{ then } \infty \end{cases}$$

D^k; d^k_{ij} = wt. of min. path from i to j using only vertices out of the set $\{1 \dots k\}$.

→ $O(n^3)$: time

$O(n^3)$: space

i.e. D^0 : wt. of min. path from i to j using no additio vertices i.e from set ϕ.

Π^k_{ij} = In the best path from i to j, using vertices from the set $\{1, \dots, k\}$, the predecessor of j

if no path from i to j using vertices from {1...k} only, then NIL (i=j) then i

What is Π_{ij}^{k} → containing previous node is shortest path in group from i to j

```
printpath (Π^n, i, j)
    if (i==j) print (i)
        else if Π_ij^n == NIL print "NIL, no path"
            else printpath (Π^n, i, Π_ij^n)
                print j
```

Floyd Warshal (G, w)

1. $$D^0 = \begin{cases} w(i,j) & \text{if } \langle i,j \rangle \in E \\ 0 & \text{if } i=j \\ \infty & \text{if } \langle i,j \rangle \notin E \end{cases} \Big\} O(n^2)$$

2. $$\Pi^0 = \begin{cases} NIL, & \text{if } \langle i,j \rangle \notin E \\ i, & \text{if } i=j \text{ or } \langle i,j \rangle \in E \end{cases} \Big\} O(n^2)$$

3. for k = 1 to n
    # Compute D^k and π^k.

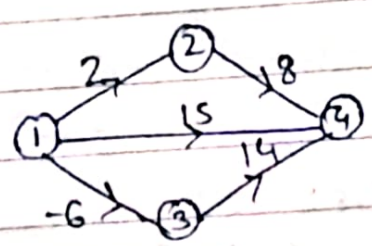
4. for i=1 to n
   for j=1 to n

$$d_{ij}^{k} = \min\left(d_{ij}^{k-1},\ d_{ik}^{k-1} + d_{kj}^{k-1}\right)$$

if $(d_{ij}^{k} < d_{ij}^{k-1})$ $\pi_{ij}^{k} = \pi_{kj}^{k-1}$

else $\pi_{ij}^{k} = \pi_{ij}^{k-1}$.

5. D^n & π^n contain the shortest path



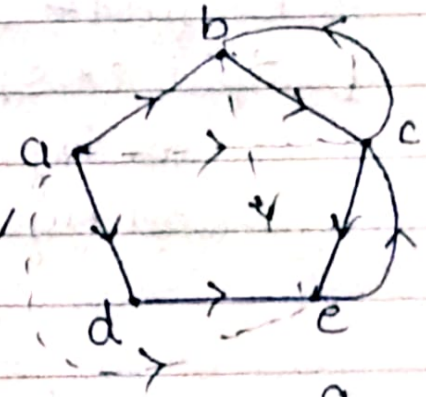
$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 2 & -6 & 15 \\ \infty & 0 & \infty & 8 \\ \infty & \infty & 0 & 14 \\ \infty & \infty & \infty & 0 \end{bmatrix} \end{array} \qquad \pi^0 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \text{nil} & 2 & \text{nil} & 2 \\ \text{nil} & \text{nil} & 3 & 3 \\ \text{nil} & \text{nil} & \text{nil} & 4 \end{bmatrix}$$

$$D^1 \begin{bmatrix} 0 & 2 & -6 & 15 \\ \infty & 0 & \infty & 8 \\ \infty & \infty & 0 & 14 \\ \infty & \infty & \infty & 0 \end{bmatrix} \qquad \pi^1 \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

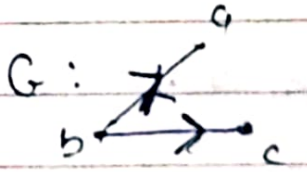
Success
DATE
PAGE

1-4-19 Transitive closure of a directed graph: $G = \langle V, E \rangle$

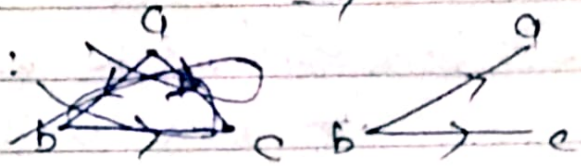
$T^*(G) = \begin{cases} V_{T(G)}: \text{set of vertices} \\ E_{T(G)}: \langle u, v \rangle : u \in V \text{ and } v \in V \text{ and } u \overset{P}{\sim} v \text{ in } G \end{cases}$



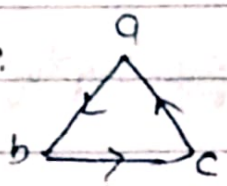
G:



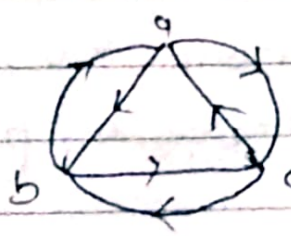
T(G):



G:



T(G):



$D^n : d_{ij}^{k} = d_{ij}^{k-1} \vee (d_{ik}^{k-1} \wedge d_{kj}^{k-1})$.

Space complexity: $O(2n^2) = O(n^2)$.

Time complexity = $O(n^3)$.

SPLAY TREES:

BST

Amortized complexity $O(\log_2 n)$

0) Splay tree, S is a set of keys.
1) Keys are totally ordered; distinct.
2) Insertion, deletion & search --- $O(\log n)$ amortized time.
3) key field & refer with key field.
4) Split (x, S) results in S' & S'' such that
   all key $\leq x$ are in S'
   all keys $\geq x$ are in S''
   S & S'' are also ~~Splay tree~~ Splay tree.

2) Join (S, S') takes ~~two~~ two sets S, S' such that
   $\forall\ x \in S$ and $y \in S'$
   $x < y$.
   and return $S \cup S'$ as a proper Splay tree
   (Inorder is preserved.

3) Member (x, S): yes if $x \in S$.
no if $x \notin S$.

4) Insert (x, S) $S = S \cup \{x\}$

5) ~~Bulk~~ Delete (x, S) $S = S - \{x\}$

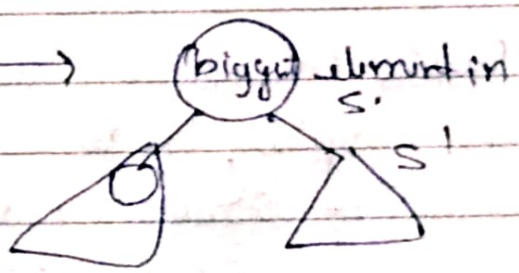
SPLAY (x, S) used to implement other operations.

: x becomes the root of S.

if $x \notin S$ then the predecessor of succ of x becomes the root

1) Join (S, S')

splay (∞, S) ⟶ (biggest element in S.)
s.right = S'



2) Split (x, S)

splay (x, S)
if (S.root == x)
S'' = x.right
x.right = nil
S' = x

```
else
    if (S.root < x)
        s'' = S.root.right
        S.root.right = nil
        s' = S
    else
        s' = S.root.left
        S.root.left = nil
        s'' = S
```

3) Member (x, s)

```
Splay (x, s)
if s.root == x  return yes
    else return no
```

4) Delete (x, S)

```
splay (x, S)
if S.root == x
    S = x.left
    s' = x.right
    x.left = nil
    x.right = nil
    JOIN (S, S')
```

Insert (x, s).

splay (x, s)

x.left = s.root.left

s.right = s.root

s.root.left = nil

s = x.

rotate (x) (New defination).