Elementary Graph Algorithms

Graph traversal
- BFS (Queue)
- DFS (Stack)

BFS

$d(v) = +\infty \quad \forall v \in V \setminus \{s\}$

$d(s) = 0, Q = \{s\}$

While $Q \neq \phi$

so

$u \leftarrow$ Dequeue (Q)

{ for each $v \in Adj[u]$
  if $d[v] = +\infty$
  then
    $d(v) = d(u) + 1$ } Relaxation step
    Enqueue (Q, v)

BFS is Dijkstra's Algorithm for unweighted graph

$\{\pi(v), v / v \in V \setminus \{s\}\}$ ↕

Tree like structure. BFS tree

T.C = $O(V+E)$

Sub path of a shortest path is also shortest path

Diffrence is 1

$\delta(s,v) = d(v)$

Invariance : v comes after u in Q

$d(v) = d(u)$ or

$d(v) = d(u)+1$

$Q : v_1, v_2, \ldots v_m$

Queue contains vertices with diff in distance being equal to 1

Prove yourself

Q. Detection of cycle in graph, is it acyclic?

Q. Is G bipartite ? ⇔ G contains no odd cycles

$G = (V, E)$ $V = V_1 \cup V_2$, $V_1 \cap V_2 = \phi$

To prove (i) G contains odd cycles, so it is not bipartite

(ii) G is bipartite, so it does not contain any odd cycle.

Choose a vertex s

Then vertex which are at even distance from s put them in V_1

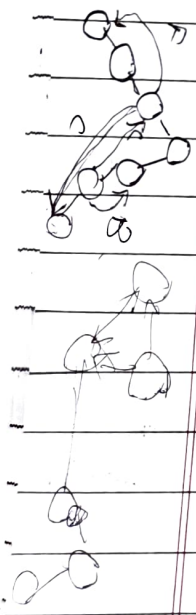
Those vertices which are at odd distance from s put them in V_2

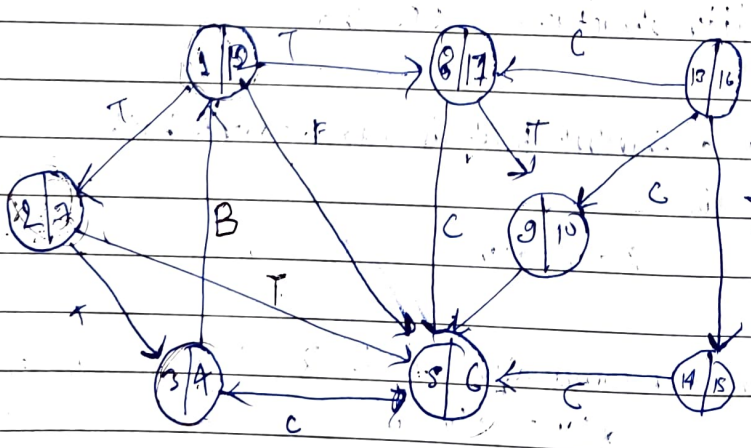
Trees → bipartite (contains no cycles, forget odd cycles)

DFS

W, g, b → colours to do traversal

visit visit of all descendants (DFS tree)





Tree : edg to visit

Back :  From descendant to anestor

ALGO

Forward edge :  From ancestor to descendant

In the given graph we will not get a tree but a forest.

Cross edges → Edges across trees, all other edges

For every vertex, there is discovery time and finishing time
$[d(v), f(v)]$

Parantheis Theorem:

↳ Consider d[v] & f[v] as paranthesis, to prove, prove there is no overlap

* for every two vertices u and v, one of the following is true

(i) the intervals $[d(u), f(u)]$ & $(d[v], f[v])$ are disjoint

(ii) one interval contains the other

Assume: $d(u) < d(v) < f(u)$

To prove: $f(v) < f(u)$

For v is a descendant of u

$I_v \subseteq I_u$

DFS Properties -

(i) Paranthesis theorem.

(ii) White Path Theorem : In a DFS forest of a graph G, v is a descendant of u ⟺ at time d[u], there is a path from u to v that contains only white vertices.

Proof: Since v is a descendant of u, implies

$I_v \subseteq I_u = [d(u), f(u)]$

At time d[u], u ⟶ v white path

$u = v_1 v_2 \cdots v_k = v$

CLASSMATE Date Page

Q3_iA

To prove: v is a descendant of u

Base case - v_0 is a descendant of u

Induction hypothesis - v_i is a descendant of u

To prove - v_{i+1} is a descendant of u

↳ (i) $d(u) < d(v_{i+1})$

(ii) $f(v_{i+1}) < f(u)$

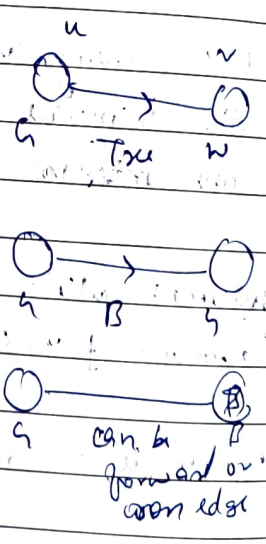
$d(u) < d(v_{i+1}) < f(u)$ is true

Claim: $d(v_{i+1}) < f(v_i)$

By induction hypothesis

$d(u) < d(v_i) < f(v_i) < f(u)$

Modify DFS to tell what kind of edge it is



Theorem: G is undirected. DFS will encounter only tree and back edge

Suppose (u,v) is an arbitrary edge

$d(u) < d(v)$: v is to G

u - grey

$u \to v$

G w

Topological Sort

DAG (Directed Acyclic Graph)

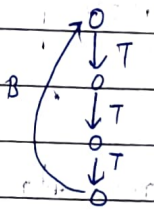
linear ordering of vertices ∋ whenever an edge $(u,v) \in E \iff$ u appears before v in my ordering.

Topo sort

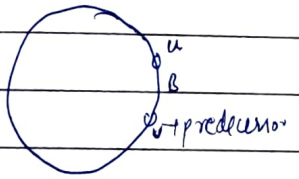
Sort in decreasing order of finishing time. //

Lemma: A directed graph G is acyclic iff a DFS of G yields no back edges.

Proof: Back edge ⇒ cycle



To show: If there is a cycle, you encounter a back edge.



cycle ⇒ back edge

Correctness of Topo Sort

T.P: If $(u,v) \in E$, then $f(v) < f(u)$

Exploring (u,v) → What is colour of u & v?

→ u is gray

→ Is v gray too? ← No, back edge

→ Can v be ~~black~~ white? v is a descendant of u.

so, $f(v) < f(u)$ Paranthesis Thm

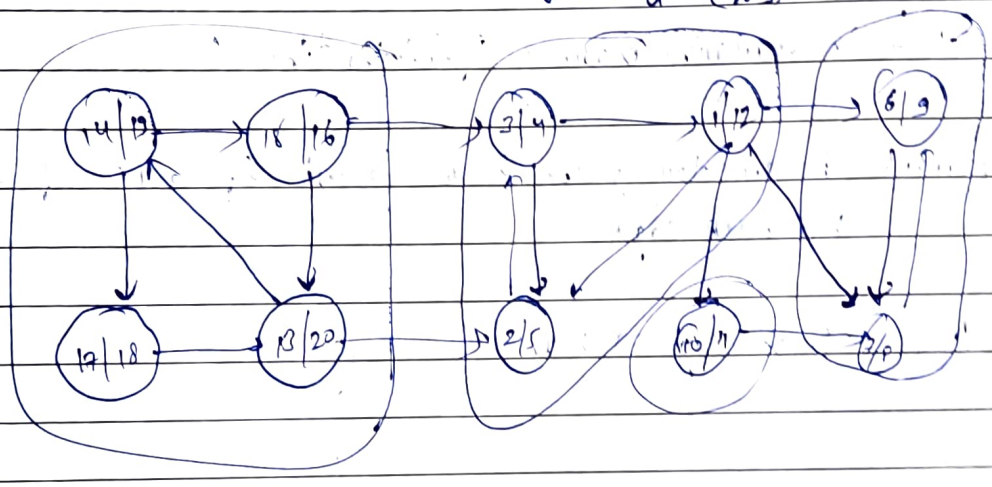
→ Is v black

Strongly Connected Components

Directed graph : $G = (V, E)$

SCC of G is a maximal set of vertices

$C \subseteq V$ $\Rightarrow$ $\forall u, v \in C$, both $u \rightsquigarrow v$ & $v \rightsquigarrow u$ exist.

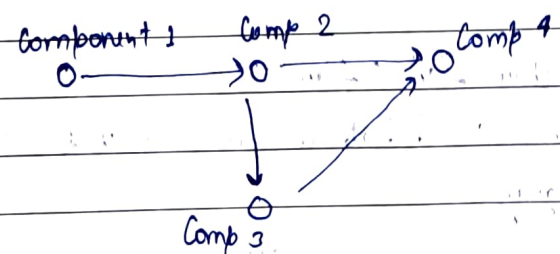


Kosaraju Algorithm

G_{SCC} → directed acyclic graph

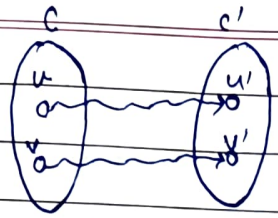
Component graph :

$G^{SCC} = (V^{SCC}, E^{SCC})$



Claim : ~~SC~~ G^{SCC} is a DAG

To prove acyclic → If there is a path from u to v', then cannot be a path from v' to u



Suppose $v' \rightsquigarrow v$ in G

$u \rightsquigarrow u' \rightsquigarrow v'$

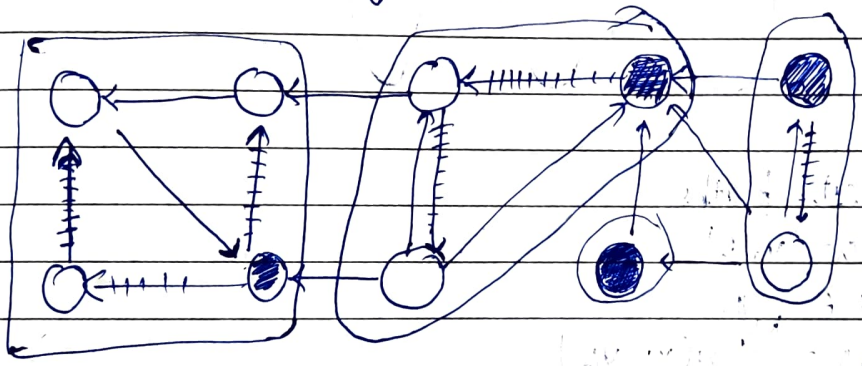
$v' \rightsquigarrow v \rightsquigarrow u$

u, v are in same component $\Rightarrow \Leftarrow$

→ Run DFS(G) → $O(V+E)$

→ Compute G^T

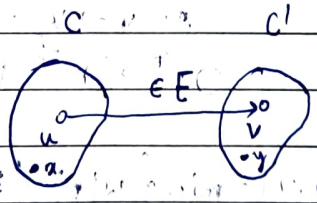
→ Run DFS(G^T) : Consider vertex ordering in decreasing order of finishing time (u)



$d(U) = \min_{u \in U} d(u)$

$f(U) = \max_{u \in U} f(u)$

Lemma:



$f(C) > f(C')$

Suppose : $d(C) < d(C')$

x is the first vertex to be discovered in C

$\Rightarrow d(x) = d(C)$

At time $d(x)$, all vertices in both C & C' are white

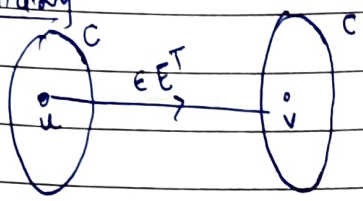
By White Path Theorem

Suppose : $d(C) > d(C')$

y is the first vertex to be discovered in C'

Since, there is no path from C' to C, the vertices of C' are discovered first before any vertex of C

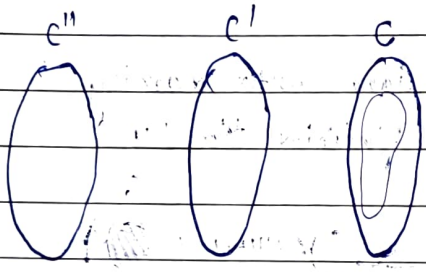
Corollary



$f(C) < f(C')$

Corollary II : Contrapositivity of corollary I

$C, C' \rightarrow$ SCC's of G & $f(C) > f(C')$, Then $\nexists$ an edge from C to C' in G^T.



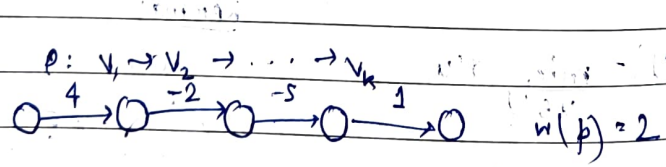
Shortest paths:

$G = (V, E)$

$w : E \rightarrow \mathbb{R}$

$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$

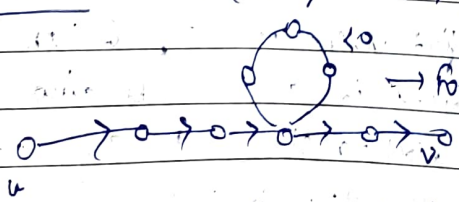
weight of the path



$\delta(u,v) = \min \{ w(p) / p : u \rightsquigarrow v \}$

shortest path wt.

Theorem: $\delta(u,v) \leq \delta(u,x) + \delta(x,v) \quad \forall u, v, x \in V$

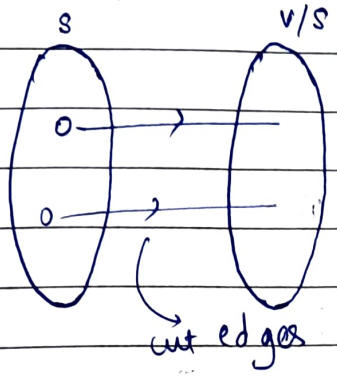


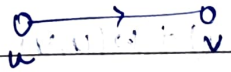
→ for negative weights, we will keep moving in the loop forever inorder to minimize the path weight.

$\delta(s,v) \quad \forall v \in V$

$d(v) = +\infty \quad \forall v \in V \setminus \{s\}$

$d(s) = 0$



If 

$d(v) \geq d(u) + w(v,u)$

$d(v) \leftarrow$

Shortest Path Algorithms:

SSSP : Dijkstra's Algorithm

single source shortest path

$G = (V, E) \quad W: E \to R$

$s \in V$

Find $\delta(s,v) \quad \forall v \in V \setminus \{s\}$

Optimal substructure : Sub path of shortest path is shortest path.

$\delta(u,v)$ is not defined

Greedy method : $s, v \notin s$

vertices for which $\delta(s,v)$ is known

```
d(s) ← 0
d(v) ← ∞   ∀ v ∈ V \ {s}
S ← ϕ
Q ← V
while ( Q ≠ ϕ )
    do
        u ← EXTRACT_MIN (Q)
        S ← S ∪ {u}
        for each v ∈ Adj[u]
            do
                if d(v) > d(u) + w(u,v)
                then
                    d(v) ← d(u) + w(u,v)   ← relaxation step
```

$O(E)$ = $\deg(u)$

$\rightarrow O(E \log V)$

if we are using binary heap implement priority queue

$d(u)$ → $O(\log V)$ decrease key

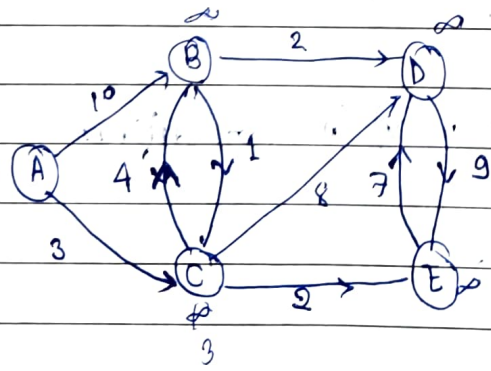
T.C = $O(V + E \log V)$

Proof

If we use fibonacci heap

↳ T.C = $O(E + V \log V)$

decrease key → $O(1)$ amortized



Q	A	B	C	D	E
	0	∞	∞	∞	∞
S = {A}		10	3	∞	∞
{A, C}		7		11	5
{A, C, E}					9
{A, C, E, B}					

0 → shortest path distance

Lemma: $d(s) \leftarrow 0$ and $d(v) \leftarrow \infty \quad \forall v \in V - \{s\}$

$\underbrace{d(v) \geq \delta(s,v)}_{\text{invariant}} \quad \forall v \in V$

This invariant is maintained over any sequence of relaxation steps.

Proof Let us use contradiction to prove

Suppose for some $v \in V$ is the first vertex s.t.

$d(v) < \delta(s,v)$

u - vertex that caused this to happen

$d(v) = d(u) + w(u,v)$

$d(v) < \delta(s,v)$

$\leq \delta(s,u) + \delta(u,v)$ ⟵ Triangle inequality

$d(u) \quad w(u,v)$

since v is the first vertex with violation

$\Rightarrow \Leftarrow$

$d(v)$ cannot be less than $d(u) + w(u,v)$ since we have relaxed v using u.

Lemma: $s \rightsquigarrow u \rightarrow v$ shortest path

if $d(u) = \delta(s,u)$ and edge (u,v) is relaxed, then $d(v) = \delta(s,v)$

Proof Suppose $d(v) > \delta(s,v)$ before relaxation

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

$d(v) > d(u) + w(u,v)$ ⟵ Test must have succeeded

$d(v) > \delta(s,v) = \delta(s,u) + w(u,v)$

$= d(u) + w(u,v)$

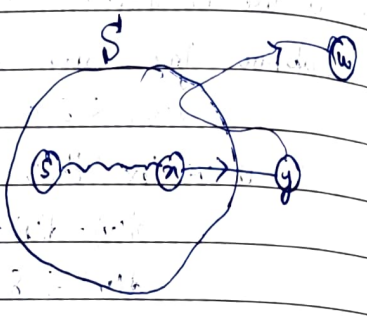
$= \delta(s,v)$

Theorem: Dijkstra's terminates with $d(v) = \delta(s,v)$ $\forall v \in V$

Enough to prove:

$d(v) = \delta(s,v)$ for every $v \in V$ when v is added to S.

Suppose u is the first vertex added to S for which $d(u) < \delta(u,v)$



y is the first vertex in $V \setminus S$ along the shortest path from s to u and x is the predecessor of y.

Just before adding u to S

$d(y) = \delta(s,y) \leq \delta(s,u) < d(u)$

$d(y) \leq d(u)$ $\Rightarrow\Leftarrow$

Bellman-Ford Algorithm

G contains $(-)$ve weight cycles

Find all SP lengths from $s \in V$ to all $v \in V$ (or) determining that a $(-)$ve weight cycle exists.

```
d(s) ← 0
d(v) ← ∞        ∀ v ∈ V \ {s}
for (i ← 1 to |V|-1)
  do
     for each edge (u,v) ∈ E
     do
        if d(v) > d(u) + w(u,v)
        then
           d(v) ← d(u) + w(u,v)
```

for each edge $(u,v) \in E$
do

if $d(v) > d(u) + w(u,v)$
then

report -ve weight cycle exist.

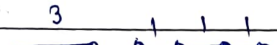
Shortest path:

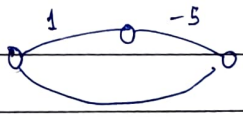
SSSP : Dijkstra Algo - Greedy $O(E \log V)$

(G, w) — PQ: Binary heap

↓ unit weight

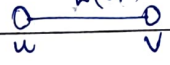
BFS





→ Dijkstra will fail here because it will choose path of -2 over path of 1 → but shortest path is path of 1.

Bellman-Ford Algorithm:



if $d(v) > d(u) + w(u,v)$
then
$d(v) = d(u) + w(u,v)$
relaxation

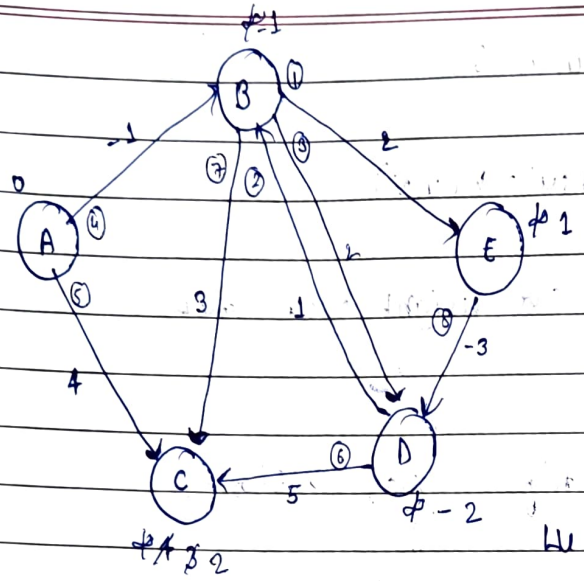
for all edges

↓

Do this for $|V|-1$ times

For all the edges, check if $d(v) > d(u) + w(u,v)$

Report a (-)ve cycle.



	A	B	C	D	E
Pass 1	0	-1	2	∞	∞
Pass 2	0	-1	2	-2	1

We will do passes till |V|-1, if after that even if the ~~weight~~ distances converge (because less than the previous one) ⇒ (-)ve weight cycle exists

Prove the theorem:

Theorem : If G contains no (-)ve weight cycles, then at the end, $d(v) = \delta(s,v)$ $\forall v \in V$

Proof : $v \in V$

P : $s = v_0 \to v_1 \to v_2 \to \cdots \to v_{k-1} \to v = v_k$

P is the shortest path

$$\delta(s,v_i) = \delta(s,v_{i-1}) + w(v_{i-1},v_i)$$

$d(v_i) = \delta(s,v_i)$ → After pass i

$d(v_i) = \delta(s,v_i)$

If d value fails to converge report (-)ve weight cycle.

Solving a system of difference constraints:

$x_1 - x_2 \leq 3$

$x_2 - x_3 \leq -2$

$x_1 - x_3 \leq 2$

Solution:

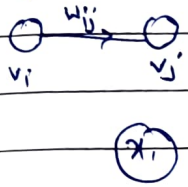
$x_1 = 3$

$x_2 = 0$

$x_3 = 2$

$x_j - x_i \leq w_{ij}$

Constraint graph



Theorem: If the constraint graph contains a (-)ve cycle, then the system of differences is unsatisfiable.

$v_1 \longrightarrow v_2 \longrightarrow v_3 \longrightarrow \cdots \longrightarrow v_k \longrightarrow v_1$ is a negative weighted cycle in G.

$x_2 - x_1 \leq w_{12}$

$x_3 - x_2 \leq w_{23}$

$\vdots$

$x_1 - x_k \leq w_{k1}$

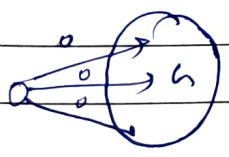
$+$

$0 \leq \sum w_{ij}$

$\downarrow$

(-) ve, so a contradiction.

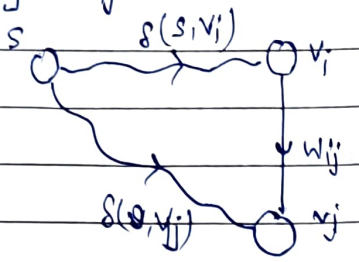
Theorem: Suppose no (-)ve weight cycle exist in G. Then the constraints are satisfiable.



$x_i = \delta(s, v_i)$

Claim.

$x_i - x_j \leq w_{ij}$



$$\delta(s, v_j) \leq \delta(s, v_i) + w_{ij}$$

$$x_j \qquad x_i$$

$$x_j - x_i \leq w_{ij}$$

Proved.

I want to maximize : $\max x_1 + x_2 + \dots + x_n$

APSP

All path shortest path algorithm

Run $n \times$ Dijkstra's — ~~$O(n^2 \log n)$~~ $O(nm \log n)$

$n \times$ Bellman ford — $O(n^2 m)$

sparse → $O(n^2 \log n)$

dense — $O(n^3 \log n)$

n = no. of edges

m = no. of vertices

sparse $O(n^3)$

dense $O(n^4)$

If the graph is sparse we are not going to get anything better than this

APSP

All pair shortest path : $G = (V, E)$ → directed
$C_e, e \in E$
O/P : $\delta(u, v)$ $\forall u, v \in V$ (or) report (-)ve weight cycle

SSSP : Dijkstras → not (-)ve weight cycle
Bellman-Ford → (-)ve weight

Dijkstra → $O(m \log n)$ - PQ implementation
Bellman-Ford → $O(mn)$

Invoke -
$n \times$ Dijkstra = $O(mn \log n)$
$n \times$ Bellman = $O(mn^2)$

No. of edges = $O(n)$
sparse → $O(n^2 \log n)$
dense → $O(n^2 \log n)$
No. of edges = $O(n^2)$

Best : If the graph is sparse and doesn't have (-)ve edge wts, Dijkstra is the best

sparse → $O(n^3)$
dense → $O(n^4)$
Tough to beat

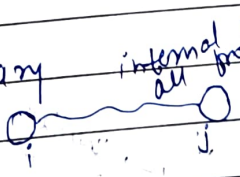
Q. Find shortest s-v path with cycles. (NP-hard)
Q. Hamiltonian path - visit every edge of the graph exactly once. (NP-hard)

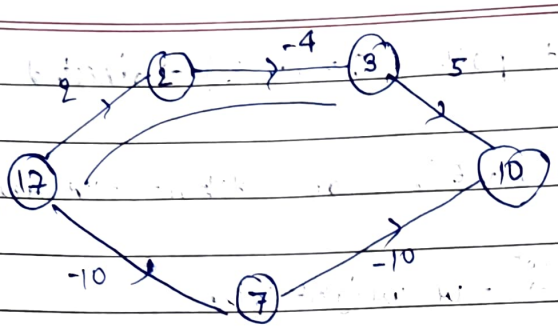
Floyd-Warshall Algorithm

$O(n^3)$ - in case of dense graph, it beats both the above algos

- (-)ve weights allowed
- Dynamic programming

$V = \{1, 2, \ldots, n\}$ arbitrary
$V^{(k)} = \{1, 2, \ldots, k\}$

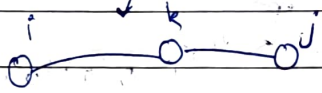




$i = 17,\ j = 10,\ k = 5$

$\rightarrow \{1, 2 \ldots 5\}$

$$A[i,j,k] = \min \begin{cases} A[i,j,k-1] & \text{, if } k \text{ is not part of the} \\ A[i,k,k-1] + A[k,j,k-1] & \text{, if } k \text{ is included} \end{cases}$$



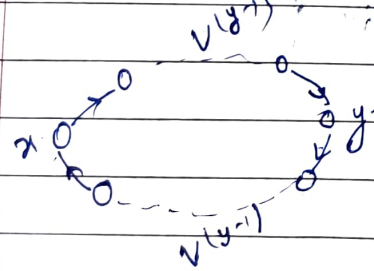
Base case

$$A[i,j,0] = \begin{cases} 0 & , i = j \\ c_{ij} & , (i,j) \in E \\ \infty & , i \neq j \ \& \ (i,j) \notin E \end{cases}$$

Running time = $O(n^3)$ $\rightarrow$ n^3 entries I need to fill & each take $O(1)$ time.

Assumption :- graph doesn't have a (-)ve weight cycle.

If $A[i,i,n] < 0$ for atleast one "i", report (-)ve weight cycle.



$y \rightarrow$ vertex with max value

$A[x,y,y-1] + A[y,x,y-1]$

After this y, the term gets on decreasing

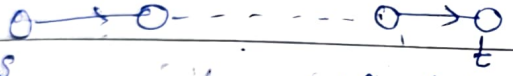
B[i,j] = k if 2nd case recurrence is used to compute A[i,j,k]

Johnson's Algorithm

$$c'_{uv} = c_{uv} + p_u - p_v$$

⇓

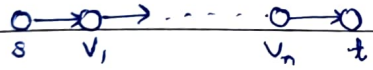
This will give the correct length of the shortest path.



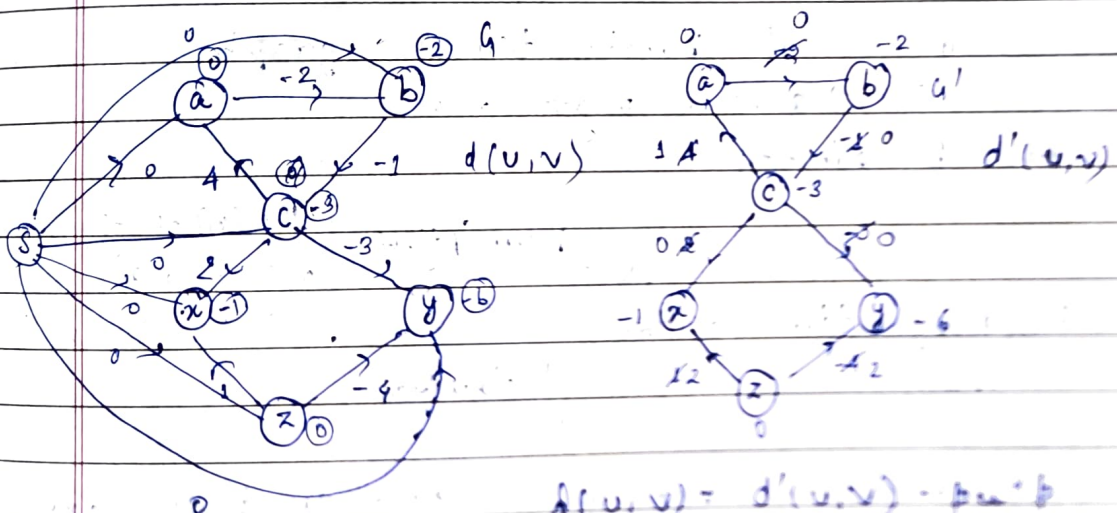
$L + p_s - p_t$

APSP ≥ 1 run of Bellmann-Ford
n runs of Dijkstra

$$c'_i = c_i + p_u - p_v$$



$L + p_s + p_t$

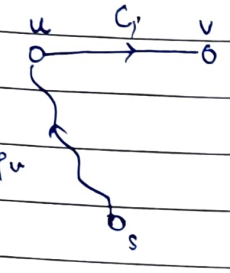


$d(u,v) = d'(u,v) - p_u + p_v$

Steps: i) Run Bellmann ford
ii) Do $c' = c - p_u + p_v$ for every edge
iii) Run dijkstra

To prove:

$c_i' \geq 0$



$p_v \leq c_i + p_u$

$c_i + p_u - p_v \geq 0$

$\Rightarrow c_i' \geq 0.$

Application : Transitive closure of a relation

Minimum Spanning Tree (MST) - undirected, weighted grh

Tree - connected acyclic graph

$T \subseteq E$ - Spanning tree of $G = (V, E)$

T - no cycle (V, T) is connected

$$W(T) = \sum_{(u,v) \in T} w(u,v)$$

Claim: There are n^{n-2} (larger node) spanning trees of a graph

$T \leftrightarrow$ a seq of length $(n-2)$

Proof by Kirchoff

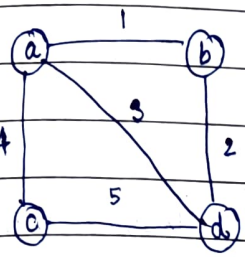
Prim (Dijkstra)	$f : [n] \to [n]$
	$n^2 T \leftrightarrow n^n$
	$T \leftrightarrow \frac{n^n}{n^2} = n^{n-2}$

MST : $O(m \log n)$ time algorithm (greedy) — PQ, Union-Find, Kruskal

MST → A tree in which we have N nodes & N-1 edges and all nodes are reachable from each other.

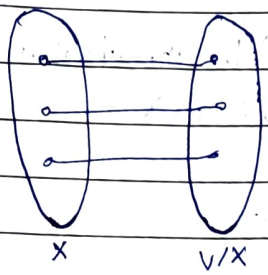
MST → Any ST with the least sum of edge wts is the MST.

i/p : G - undirected graph
edge
c_i → cost, can be negative
Edge costs are distinct



$X = \{s\}$

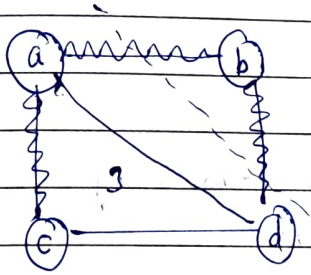
cut $(X, V \setminus X)$



Choose cheapest edge crossing this cut

$T = T \cup \{u, v\}$

$X = X \cup \{v\}$



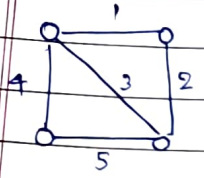
$X = \{b\}$ $T = \phi$

$X = \{b, a\}$ $T = \{(a,b)\}$

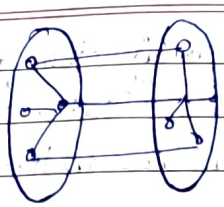
$X = \{a, b, d\}$ $T = \{(a,b), (b,c)\}$

MST : G-undirected
weights are distinct
$G = (V, E)$
$T \subseteq E$, (V, T) is connected and acyclic
Adjacency list given

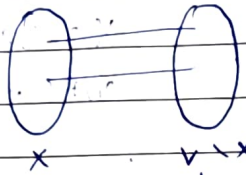
Greedy → PQ → Prim's
Greedy → UF → Kruskal



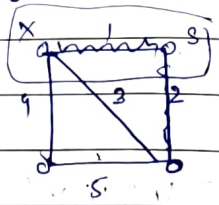
cut (A, B)
Partition of V



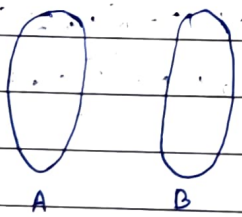
$X = \{s\}$
$T = \phi$
While $X \neq V$



Let $e = (u, v)$ be the cheapest edge of G with $u \in X$ & $v \in V \setminus X$
→ ~~Add~~ e to T
→ Add v to X



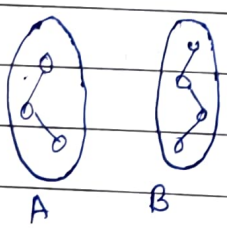
Invariant:
T - set of edges that spans X
$|V| = n$



2 choices for using vertex $2^n - 1$

Empty cut lemma.
G = not connected $\Leftrightarrow$ $\exists$ a cut (A, B) with no edges crossing the cut

($\Leftarrow$)

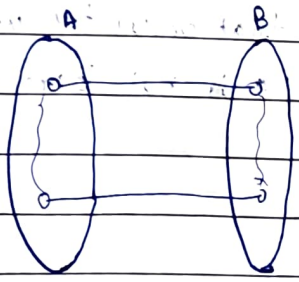


$\exists u, v \in V \ni$ there is no u-v path
A = of all vertices reachable from u
B = remaining vertices // Cut

Characterize the ~~cut~~ is connectivity in form of cuts

Double crossing lemma

C - cycle $\subseteq E$ has an edge crossing the cut (A, B), then some other edges of C also crosses the cut.



Lonely cut corollary

If ~~either only edge cross~~ e is the only edge crossing some cut (A, B), then it is not in any cycle.

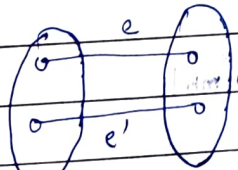
Prim's edge outputs a ST

Invariant : T spans X

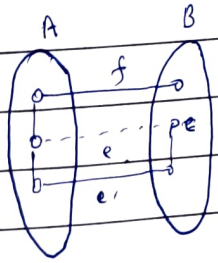
- T spans V
- e is the first edge crossing the cut $(X, V \setminus X)$, & is a lonely edge // Proved that T is a spanning tree

Now prove :

T is the minimum spanning tree



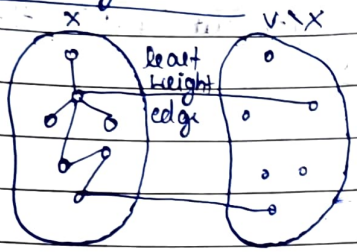
T ~~is~~ very likely we need to choose e' property, or either we'll end up with a disconnected graph or a cycle.



Exchange $e \mapsto f$ ✗

Exchange $e \mapsto e'$

Prim's Algorithm

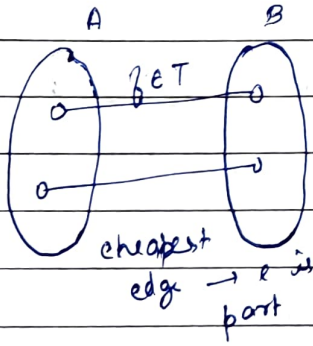


→ e is the first edge of T that crosses the cut (X, V\X)
→ e is not part of a cycle in T

→ T spans X
→ T spans V at the end
→ T is acyclic

Cut property

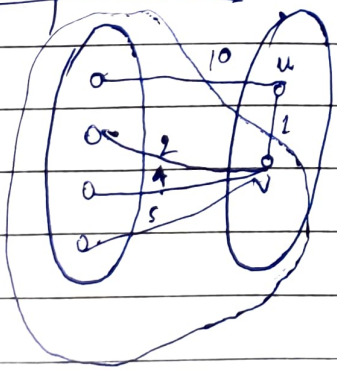
e ∉ G : Suppose ∃ a cut (A, B) such that e is the cheapest edge across (A, B). Then edge e belongs to the MST.



T \ {f} ∪ {e}
T ∪ {e} forms a cycle C
e ∈ C ⇒ ∃ e' ∈ C that crosses (A, B)

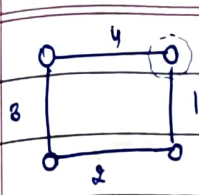
Naive implementation of Prim's : O(mn)
- (n-1) iterations of while loop
- each iteration takes O(m) time.

Priority Queue :



u = 10, 7
v = 2

- delete u
- add (u) with updated (d) value



Kruskal. $O(m \log n)$ greedy

Randomized algorithm

$O(m) \rightarrow$ Randomized algo

$m\alpha(n)$ time

inverse Ackerman function

$n = 2^{2^{2^{2^{2}}}}$ } t

$\log^*(n) \cdot t$

→ union : find data structure

→ Sort edges wrt weights $O(m \log n)$

→ Add edges

(Rename the edges as 1, 2, ... m ∋ $c_1 < c_2 \cdots < c_m$

$T = \phi$

for i = 1 to n

if $T \cup \{i\}$ has no cycles

Add i to T

first add 1

added 2

add 3

cannot add 4 → since it will form a cycle

add 5 // done.

We do not worry about connected or not.