⇒ GREEDY APPROACH:- (only for optimization problems)

↳ Involves constructing a solution through (suggests) a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.

Properties ⇒ At each step the choice is,

↳ Feasible
↳ locally optimal
↳ Irrevocable.

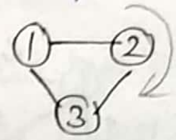
Greedy Approach ⇒
* Minimum spanning tree (Kruskal's, Prim's algorithm)
* Single source shortest path (Dijkstra's)
* Huffman trees.

— MINIMUM SPANNING TREES

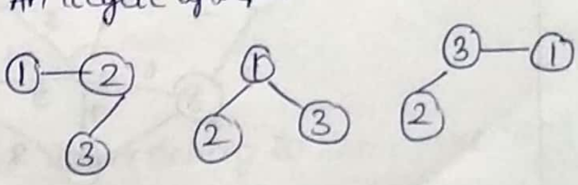
↳ Given a connected graph (can be non-weighted / weighted), a spanning tree is a connected acyclic sub-graph that contains all the vertices of the graph.

↳ In a weighted graph, a minimal spanning tree (*) (always weighted) is a spanning tree of the graph with the smallest sum of the weights of the edges in the spanning tree.

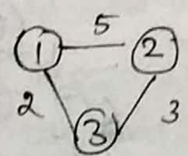
Eg:- Given a graph



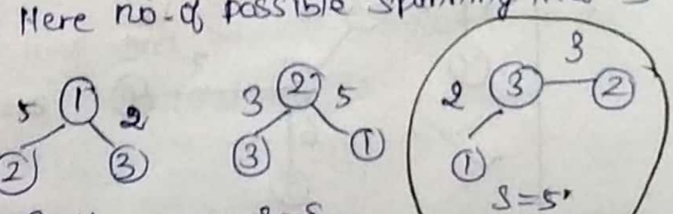
Spanning tree (can have more than one too) ⇒ An acyclic graph



Consider a weighted graph



↳ Here no. of possible spanning trees = 3



Soln:- out of these 3, only one can be a MST (i.e) the one with min sum of the weights of all edges.

S = 4 S = 8 S = 5 (min)

This is the Minimum spanning Tree

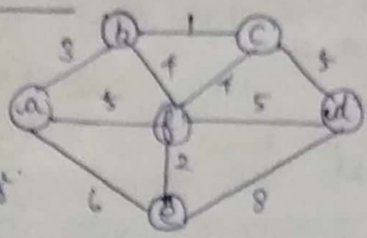
Next page → There are 2 algorithm for finding minimum spanning Tree

Kruskal's algorithm:- (To Find MST)

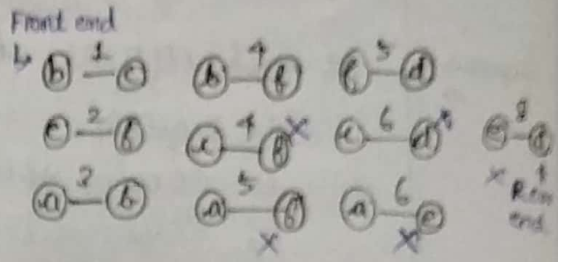
1. ↳ Sort the edges according to the weights in AO
   ↳ Construct the MST.

[Note] Remove loops and parallel edges (more than 1 edges) between 2 vertices, choose the min among the parallel edge.

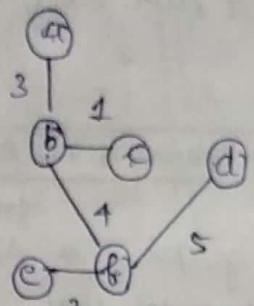
Given



(It's like a priority queue)



Soln:-



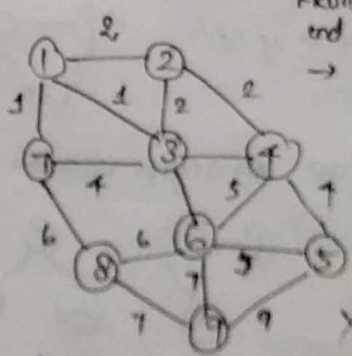
[Note]:-
(*) Before deleting an edge and adding it to the MST, check whether a cycle is forming. If so don't add else add it to MST.

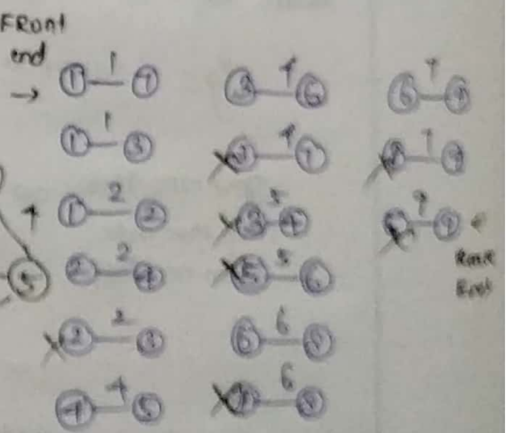
Sum of wt. on edges = 15 /.

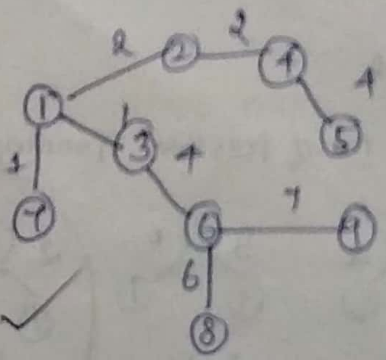
If no. of vertices in MST = No. of vertices in original tree [cond. satisfied]

once, its satisfied too we can stop constructing MST.

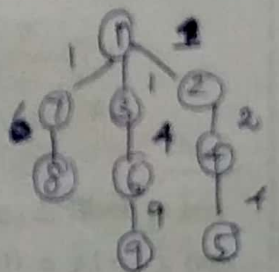
Eg:-







(or)



Sum of wt. of edges in the MST] = 27

Algorithm ⇒ Kruskal (<V, E, W>)

Sort E so that $W(e_1) <= W(e_2) <= \dots <= W(e_{|E|})$
// Sorting weights on edges in Ascending order.

$E_t \leftarrow \{\}$

// E_t Represents MST and for time being its empty

count ← 0 // no. of vertices
k ← 0 // no. of edges.
While count < |V|-1
  k ← k+1 // Everytime an edge is drawn
  If E_t union $\{e_k\}$ isacyclic // After e_k is added to MST
    $E_t \leftarrow E_t$ union$\{e_k\}$
    count ← count+1
  return E_t

Three operations ⇒

(i) construction of tree
(ii) Finding whether the vertices belong to the same tree
(iii) If they dont belong, connect them.

OPERATIONS:-

↳ (i) Creating a single vertex:
  ↳ use an abstract data type makeset.

Eg:- Makeset(x): creates the subset {x}
  ⇒ Makeset(1): {1}
  ⇒ Makeset(2): {2}

↳ (ii) Find whether the Vertex belong to the same tree
  ↳ Find(x): Returns the current subset that contains x (or a unique identifier of it)

(ie) To choose a vertex and find whether its present in the MST, use Find(x) → used to find, where the element belongs to.

Whenever a set is created, a representative (an element) will be there.

(Say) for Makeset (1) = {1} → 1 is Rep
for Makeset (2) = {2} → 2 is Rep.

If we give Find (x) → usually Returns the Representative.

↳ To include the vertex to the tree:-

Union(x,y); causes the subset containing x and the subset containing y to be joined together so that x and y belong to the same subset. This new subset replaces the subsets that used to contain x and y respectively.

(ie) It join together the set having x and y.

↳ Eg:- Union (1,2) = {1,2}
Union (4,5) = {4,5}
Union (2,4) = {1,2,4,5}.

As we are considering disjoint sets (an element can be present in only one set), once union is created, individual sets are deleted.

Every time when union is performed, Rep changes for the set.

(Say) Union(1,2) = {1,2}
Union (1,3) = {1,2,3}

Here 1's, 2's and 3's rep is 1. (Just we considered 1st element of the set to be Rep, but any other element can be too)

So, Find (3) → 1 is Returned.

Implementation ⇒ < Linked List Representation
Tree Representation

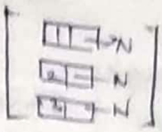
Linked list Representation :-

↳ Here w.r.to this, makeset(1) → creates a list with one element here = 1

↳ While creating a list let us have a header having no. of elements in each list and also has pointer towards first and last element in the list.

↳ for makeset(x) → Total count is 1

Here find is easy



↳ But suppose if total count is 'n', and if we find, all elements in the list has to be traversed

How to Reduce? → Use Indexing Technique (Quick Find)

Quick Find :-

↳ Every element is considered as a index and for every index we have a rep.

Eg:- Rep of {1, 4, 5, 2} is 1

Rep of {3, 6} is 3 Likewise.

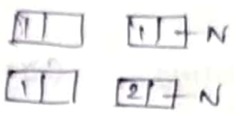
Here what happens in case of union?

Say union (3,4) → prefer appending small to larger (based on size perform)

Note :- No. of times the rep changes is dependent on how many times the union operation take place and viceversa

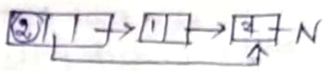
makeset(1)

makeset(2)



Suppose if union(1,2),

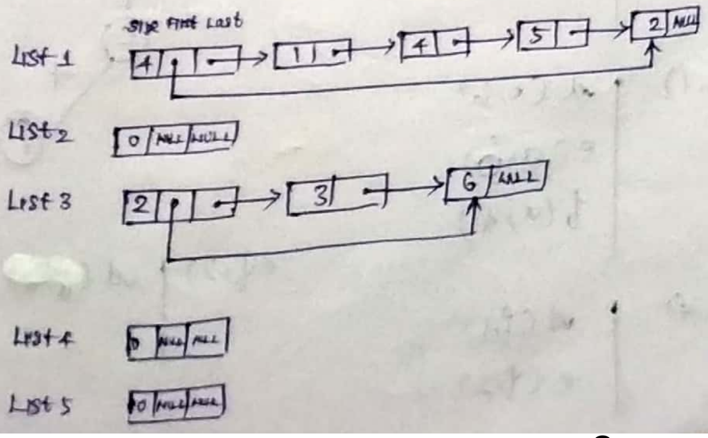
Rep changes =>



↳ Quick find:-

Say we have totally 6 indices

Subset Representative	
element index	Representative
1	1
2	1
3	3
4	1
5	1
6	3



$3 \to 4 \to 1 \to 2$

Note

w.r.t to every single element, Rep changes

No. of times the rep changes $= i$

Then size of list $= 2^i$

where $2^i \le N \Rightarrow \boxed{i = \log_2 N}$

No. of times Rep changes	Size of obtained list (merged)
1	$2 = 2^1$
2	$4 = 2^2$
3	$8 = 2^3$

~~For N element, (n-1) union~~ operation

Complexity $\Rightarrow$

↳ Makeset(x) : $\Theta(1)$

↳ Find (x) : $\Theta(1)$

↳ Union (x, y) : $\Theta(n \log n)$

↳ For (n-1) unions and m find operations : $\Theta(m + n\log n)$.

Scanned with CamScanner

ALGORITHM Prim(G)

// Prim's algorithm for constructing a minimum spanning tree

// Input: A weighted connected graph $G = (V, E)$

// Output: E_T, the set of edges composing a minimum spanning

// tree of G, $V_T \leftarrow [V_0]$ // the set of tree vertices can

// be initialized with any vertex

$V_T \leftarrow [V_0]$

$E_T \leftarrow \emptyset$

for $i \leftarrow 1$ to $|V|-1$ do

find a minimum weight edge $e^* = (v^*, u^*)$ among all the

already present

not present

edges (v, u) such that v is in V_T and u is in $V - V_T$

$V_T \leftarrow V_T \cup [u^*]$

$E_T \leftarrow E_T \cup [e^*]$

return E_T

Proof

T_0 – one vertex.

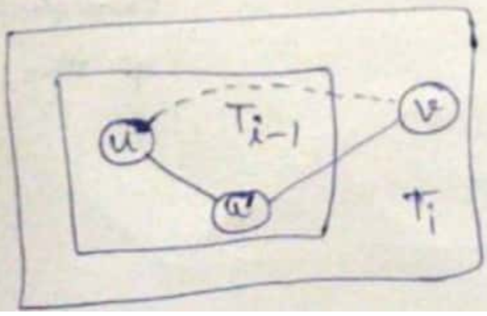
T_1 – 2 vertices.

T_2 – 3 vertices.

$T_i = T_{i-1}$ **

To prove: T_i is a MST (Proof by contradiction)

Assume $\begin{cases} T_{i-1} \text{ is a MST} \\ T_i \text{ is not a MST} \end{cases}$



$v(u, -)$ – // Not possible → Contradiction

$v(u', -)$ – Minimum as minimum vu' is already added.

Assumption false

H/P

Complexity

- Graph is represented as adjacency list and priority queue is implemented as a min heap

- No. of deletions : $|V|-1$

- No. of verifications : $|E|$.

- Changing the priority in a min heap takes place once $|V|$ times. Complexity : $\log |V|$.

- Complexity : $(|V|-1+|E|)\log V = |E|\log V$

as $|V|-1 < |E|$ in a connected graph.

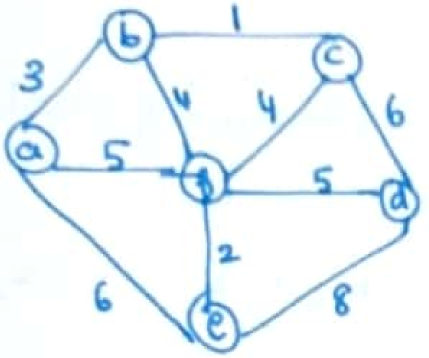
Delete the vertex, edges remain

$$0 \leq E \leq \frac{V(V-1)}{2}$$

$|V|(V-1) \geq 2E$ $\quad$ $|V|-1 < |E|$

Prim's Algorithm

* Choose any vertex initially.
* Every vertex ⓐ (,) — ↓ vertex; ↑ No. of adjacent vertices; ↓ weight.



Tree vertex V_T . | MST . | Remaining vertex

MST .

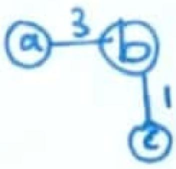
b (a,3) c (,∞) d (,∞)
e (a,6) f (a,5)

a(–,–)

b (a,3)

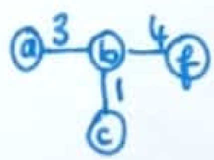
c (b,1) d (,∞)
e (a,6) f (b,4)

c (b,1)



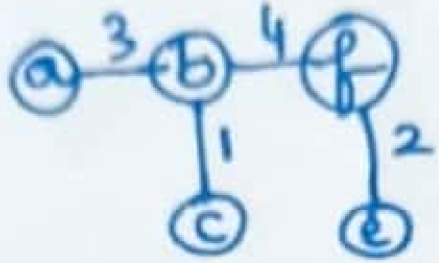
d (c,6) e (a,6) f (b,4)

f (b,4)



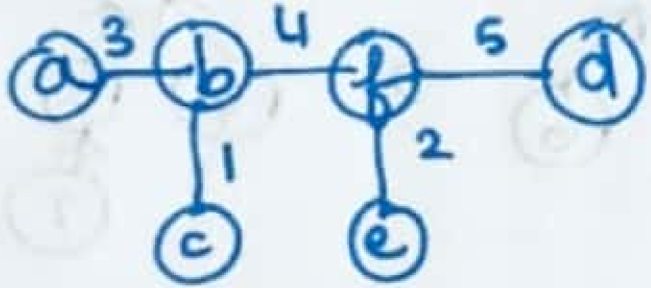
d(f,5) , e(f,2)

e (f, 2)



d(f, 5)

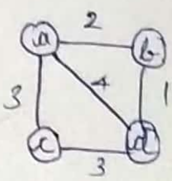
d)(f, 5)



$3+4+5+1+2=15$

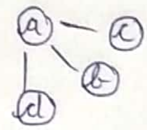
(ii) Single source shortest path (Dijkstra's Algorithm):-

Let's say we have a weighted graph. Here we will select a vertex (source) and we will be finding the shortest path from this source to other vertices.



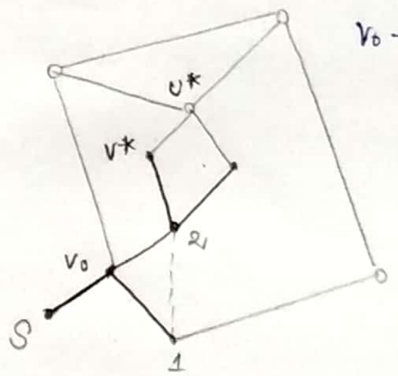
source = 'a' (say).

So, To find:- shortest paths from a to b, c and d respectively.

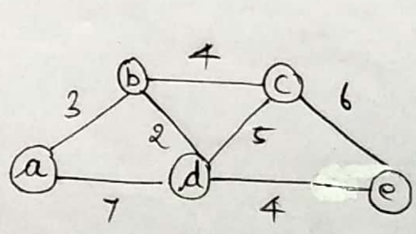


↳ Applicable for both directed and undirected graph with ⊛ NON-NEGATIVE weights only.

Here
$v_0 \rightarrow$ adj to many vertices



Here once we find the shortest distance from S to v_0,
↳ find the vertices adjacent to v_0.
↳ v_0 is adjacent to vertices 1 and 2.
↳ To reach 2, find the shortest distance through v_0 to 2 from S. i.e. We are finding shortest distance from S to 1 and 2 through v_0.



(i) 'a' is chosen as source.
(ii) Its added to to the final graph.

Note ↓ This is very similar to Prim's Algorithm But the only diff. is There we always take the shortest distance from the respective chosen vertex to its adjacent vertices. But Here we always find the shortest distance to a particular vertex from the source always. [Difference occurs from the 2nd chosen vertex] a(–, 0) | Remaining vertices

b(a, 3)

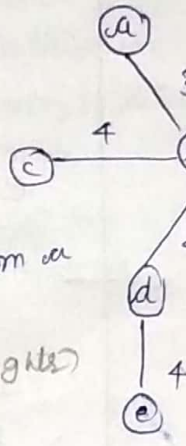
c(–, ∞)

d(a, 7)

e(–, ∞)

Next, vertex with associated a minimum weight is chosen.

Final Tree



b(a, 3) | c(b, 7) → distance of c from a

d(b, 5)

e(–, ∞)

Sum(weights) = 13

d(b, 5) | c(b, 7) For a to e, we have 2 paths

e(d, 9)

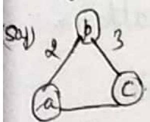
(i) a–e through d – 11

(ii) a–e through b and d – 9 ✓

c(b, 7) | e(d, 9)

Note !

If 2/more paths have same 'sum' of weights, say)



a → c [both directly and from a to c through b]

Anything (path) can be chosen.

→ Find shortest distance between a and c.

[we can find either with a, b, c d or the combo of these as intermediate vertices]

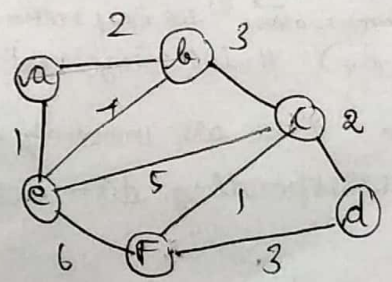
1) a – d – e – 11 ✓
2) a – b – d – e – 9
3) a – b – d – c – e – 16
4) a – b – c – e – 13
5) a – b – c – d – e – 16
6) a – d – b – c – e – 19

Example:-

a-b-c-d → 7 ✓

a-e-c-d → 8

a-e-b-c-d → 10

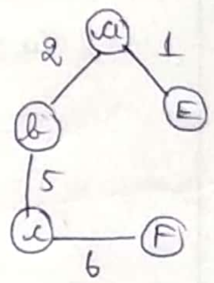


Given a graph, find the final graph using Dijikstra's algorithm:-

(source)

a(−,0)	b(a,2)
	c(−,∞)
	d(−,∞)
	e(a,1)
	f(−,∞)
e(a,1)	b(a,2)
	c(e,6)
	f(−,∞)
	d(−,∞)
b(a,2)	c(b,5)
	d(−,∞)
	f(−,∞)
c(b,5)	d(c,7)
	f(c,6)
f(c,6)	d(f,10)

Final graph



Algorithm →

Dijikstra (G,s)

// Dijikstra's algorithm for single source shortest path

// Input:- A weighted connected graph G = {V, E} with non-negative weights and its vertex s.

// Output:- The length d_v of a shortest path from s to v and its penultimate vertex p_v for every vertex v in V

Note:- Use PQ, as everywhere we are selecting min distant vertex. Here priority is the distance.

Initialize (Q) // Initialize priority queue to empty

for every vertex v in V

$d_v \leftarrow \infty$; $p_v \leftarrow$ null

(i) At first no vertex is added into the min-weighted graph. So, now for every vertex in the graph, its distance from s to v is ∞.

(ii) Since [no vertex is added]

(ii) For every vertex in the graph, we have to assign the distance

Insert (Q, v, d_v) // initialize vertex priority in the priority queue. → We are inserting every vertex (v) along with its corresponding distance from s into PQ.

(dist from source to itself)

$d_s \leftarrow 0$; Decrease (Q, s, d_s) // update priority queue

// of s with d_s — update the distance of s into the PQ.

First, vertex is chosen and added to PQ which is the source

$V_t \leftarrow \emptyset$ → shortest path graph [Any vertex with minimum d_v is deleted and added to the PQ]

for $i \leftarrow 0$ to $|V|-1$ do — Total no. of vertices (V) we have.

$u^* \leftarrow$ DeleteMin(Q) // delete the minimum

// priority element. → Element with min. priority is deleted

$V_t \leftarrow V_t \cup \{u^*\}$ and added to ~~the~~ SPG.

for every vertex u in $V - V_t$ that is adj. to u^* do — Remaining vertices.

if $d_{u^*} + w(u^*, u) < d_u$

$d_u \leftarrow d_{u^*} + w(u^*, u)$; $p_u \leftarrow u^*$

Decrease (Q, u, d_u);

Complexity ⇒ (same as Prim's algo)

→ Adjacency matrix and PQ as an unordered array: $\Theta(|V|^2)$

→ Adjacency lists and the priority queue implemented as a min-heap: $O(|E| \log |V|)$