Saturday, 4 December 2021 10:00 PM

Minimum Spanning Tree

→ Prim's

→ Kruskal

Prim's Algorithm

visited: { 0, 1, 7, 6, 5, 2, 8, 3, 4 }

cost:

0	4	4	2	4	2	0	8
0	1	7	6	5	2	8	3

Problem:

visited: { 0, 1, 4, 6, 5, 7 }

cost:

3	0	2	4	6	
0	1	4	6	5	7

Cost → 0 + 2 = 2
→ 4 + 1
→ 5

TC → O(E log V)

Kruskal's Algorithm

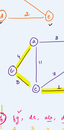
→ grow spanning tree by adding edges

→ greedy approach

① Sort the edges with respect to weights (w)
(ab), (ac), (bc), (bd) ...

② start adding edges to the MST with smallest to largest.

③ add only edges which do not form a cycle



cost → 1 + 2 + 3 + 4
= 10

(a) (f, d), (d, g), (g, f), (c, d), (c, e), (e, d), (d, b), (g, e, b)

(a, f, d, g, c, e, b)

(a) + (b) + (1 + 3) + (2 + 4) + (3) + (4) → (10) ✓✓

Prim → vertex

Kruskal → edges

graph is dense → Prim

is edges 2m → Prim

e = 6 = D

e = 10

prim / kruskal
with weak?

→ (a)

with weak edges → kruskal

with weak vertices → prim

Union - Find DS

5 people. A B C D E

group

A	in	A	friend	B
B	in	A	friend	D
C	in	A	friend	E
D				
E				

A, B, C are connected to each other

D, E are connected to each other.

* Union Find DS helps to think whether one friend is connected to another friend or a distant at. friend of friend.

{ A, B, C }

{ D, E }

Union (a, b): → count two elements as 1 gp

Find (a, b) → check if there is path from a to b

E: S = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 } , N = 10

→ uni → to manage the connectivity of elements

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

find of two S subsets can be implemented in a connected component.

	count
①	union (2,3)
②	union (9,3)
③	union (4,5)
④	union (7,8)
⑤	union (3,4)
⑥	find (3,8)
⑦	find (4,9)
⑧	find (9,7)
⑨	find (3,5)

→ false

→ true

⑩ union (0,1)

void initialize(arr[], int n) {
for (i=0; i<n; i++)
arr[i] = i;
}

}

void union (arr, a, b) {
int temp = arr[a];
for (i=0; i<n; i++) {
if (arr[i] = temp)
arr[i] = arr[b];
}
}

void find (arr, a, b) {
if (arr[a] = = arr[b])
return true;
else return false
}

}

→ for 1 union → O(n)

2nd approach

Idea: arr[a] is a parent of A

S = { 0, 1, 2, 3, 4, 5 } → (1, 2)

arr =	0	1	2	3	4	5
0	1	2	3	4	5

find (a, b) → a is to connected of every best adam

root (5) = 4
root (1) = 1

① union (1,2)
② find (5,3)
③ Union (3,4)
④ union (3,5)
⑤ find (1,4)

int root (arr, i) {
while (arr[i] != i)
i = arr[i];
}
return i;
}

union (arr, a, b) {
root_A = root (arr, a);
root_B = root (arr, b);
arr [root_A] = root_B;
}

find (a, b) {
if (root(a) = = root(b))
return true;
else return false.
}

→ Keep track of size of each subset.

→ connect the root of subset having smaller elements to the root of subset having larger elements.

① union (1,2)

②

③

③ is better

Weighted union operation

S = { 0, 1, 2, 3, 4, 5 }

size =

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

arr =

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

① union (0,1)
② union (4,3)
③ union (3,2)

→ Implement & weighted