Tuesday, 14 June 2022 9:32 PM

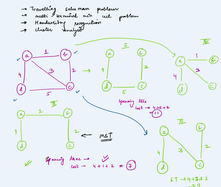
Minimum Spanning Tree

Given an undirected graph, $G = (V, E)$, a spanning tree of the graph G is a tree that spans G and is a subgraph of G.

MST:

MST is a spanning tree where the sum is minimum among all the spanning trees

→ Transitory cost-max problem
→ made associated with net problem
→ Redundancy independent
→ clusters involved



spanning tree
cost → 7+2+4
= 13

MST

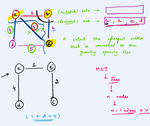
Spanning trees → cost → 4+6+4 = 14

Two algorithms

1. Prim's
2. Kruskal's

Prim's Algorithm — MST

→ greedy approach
→ we grow the spanning tree from a starting position
→ we grow by adding vertex.



(subset) tree → [ ]
(bigger) tree → [a, b, c, d]

✓ select the cheapest vertex that is connected to our growing spanning tree

MST
↓ tree
↓
← vertex
↓
a + b + c + d + e

(1 + 4 + 6 + 4)

4 + 2 + 1 + 2 + 2 + 4 + 3 = 19

Kruskal's Algorithm

→ grow spanning tree by adding edges
→ greedy approach

1. find the edge with lowest to weighted
   (a,b) (b,c) (d,e) (a,c) (c,d) (b,d) (c,e)
2. start adding edges to our MST with smallest weighted
3. only add edges which do not form cycle.

1 edge

1 + 2 + 3 + 5 = 11

→ ✓(a,b), ✓(d,e), ✓(b,c), ✗(a,c), ✓(c,d), ✗(b,d), ✓(c,e), ✗(a,e)

(c,a) → b, ✗✗

Prim's → easier → for a dense graph
Kruskal → edges → edges are in, order in future does edges are already sorted.

so easier
↓ for edges

↳ Prim & Kruskal's work with any weighted subgraph

S = 8

S = 10

= 2

Union - find DS

S pairs

	A	B	C	D	E
A	1	0	0	0	0
B	0	1	0	0	0
C	0	0	1	0	0
D	0	0	0	1	0
E	0	0	0	0	1

1. A, B, C are connected to each other
2. D, E are connected to each other.

(1) to check whether two faculty are connected to two different departments or different department.
{A, B, C} , {D, E}

1. Union (A, D) → connect two elements A and E.
2. find (A, D) → find if they is any path from A to E.

Eg.

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

we'll merge connecting 9 elements.

union → 0 → 1

S → 4

arr[5] = arr[6] 5 | 6

find(4, 5)
union (4, 5) → connect 4 to 5

arr

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

union (2, 3)
union (7, 4)
union (1, 4)
union (4, 9)
union (8, 6)
find (8, 9) false
find (2, 3) true
union (2, 8)

{2, 3, 8, 9, 5}
{7, 4, 5}
{1, 4}
{6, 8}
{0}

arr
arr → [ ] → arr.length
each value.

initialize (arr, n) {
  for (i = 0; i < n; i++)
    arr[i] = i;
  arr[0] = 1;
}

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

n elements
↓ n² times

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

O(1)

Directory approach

arr[0] → find {}
↓
S

S = {0, 1, 2, 3, 4, 5} root(1) = 0

① Union (1, 0)
② Union (0, 2)

③ Union (3, 4)
④ Union (4, 5)

find (4, 5)

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

root

root(5) → find (4)

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

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

root (4) Union(4,5)

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

int union (arr, a, b) {
  int rootA = root (arr, a);
  int rootB = root (arr, b);
  arr[rootA] = rootB;
}

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

union (1, 9)