Review

YAO ZHAO

MST

Prim

Kruskal

Implementation: Prim's Algorithm

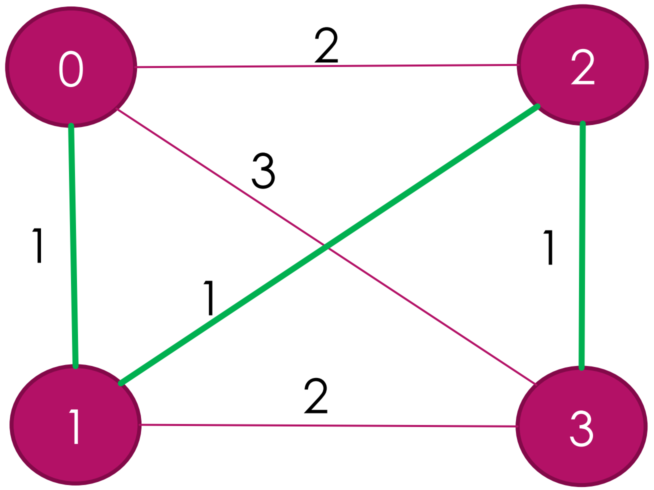
Implementation. Use a priority queue ala Dijkstra.

- Maintain set of explored nodes S.
- For each unexplored node v, maintain attachment cost a[v] = cost of cheapest edge v to a node in S.
- $O(n^2)$ with an array; $O(m \log n)$ with a binary heap.

For a completely connected graph

Use heap   or array?

- Why array faster than heap?(Prim's algorithm)
- Observe the following graph,



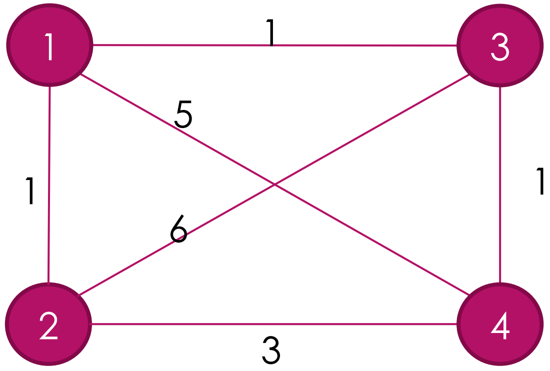
Adjacency matrix

	0	1	2	3
0	-	1	2	3
1	1	-	1	2
2	2	1	-	1
3	3	2	1	-

In a completely connected graph: $m = n(n-1)/2$

$O(m\log n) \rightarrow O(n^2 \log n) > O(n^2)$

Prim(array) vs Dijkstra(array)

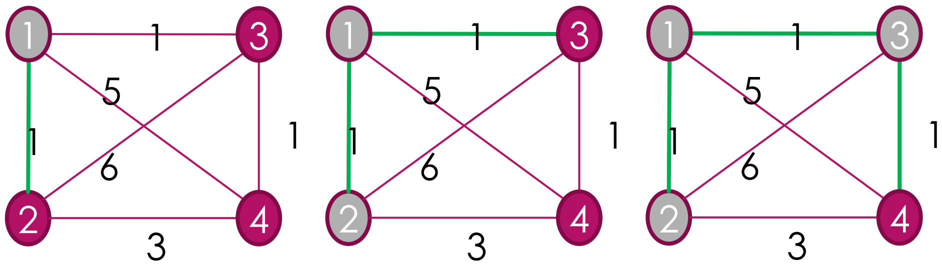


	1	2	3	4
1	-	1	1	5
2	1	-	6	3
3	1	6	-	1
4	5	3	1	-

Prim

index	1	2	3	4
loop1	**0**	**1**	**1**	**5**
loop2	**0**	**1**	**1**	**5->3**
loop3	**1**	**1**	**1**	**3->1**

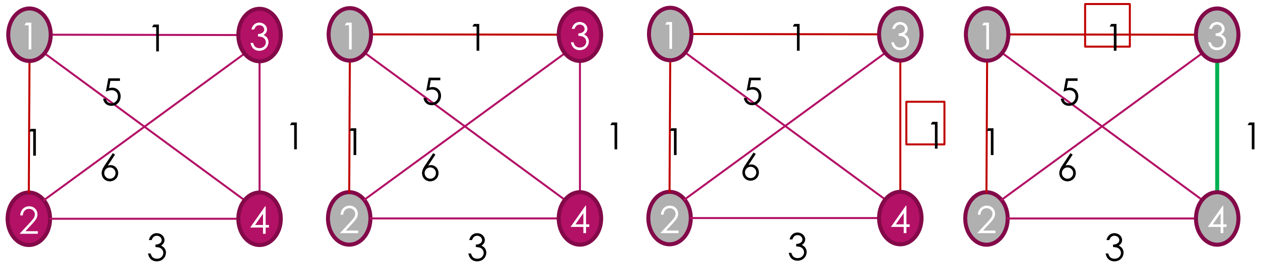
explored 1 → select(1,2) → next node 2
explored 2 → select(1,3) → next node 3
explored 3 → select(3,4) → end



Dijkstra

index	1	2	3	4
loop1	**0**	**1**	**1**	**5->4**
loop2	**0**	**1**	**1**	**4->2**
loop3	**0**	**1**	**1**	**2**
loop4	**0**	**1**	**1**	**2**

explored 1 → select(1,2) update a[4] to 4→ next node 3
explored 2 → select(1,3) update a[4] to 2→ next node 3
explored 3 → select(3,4) → next node 4
explored 4 → reach target→ end



MST

Prim

Kruskal

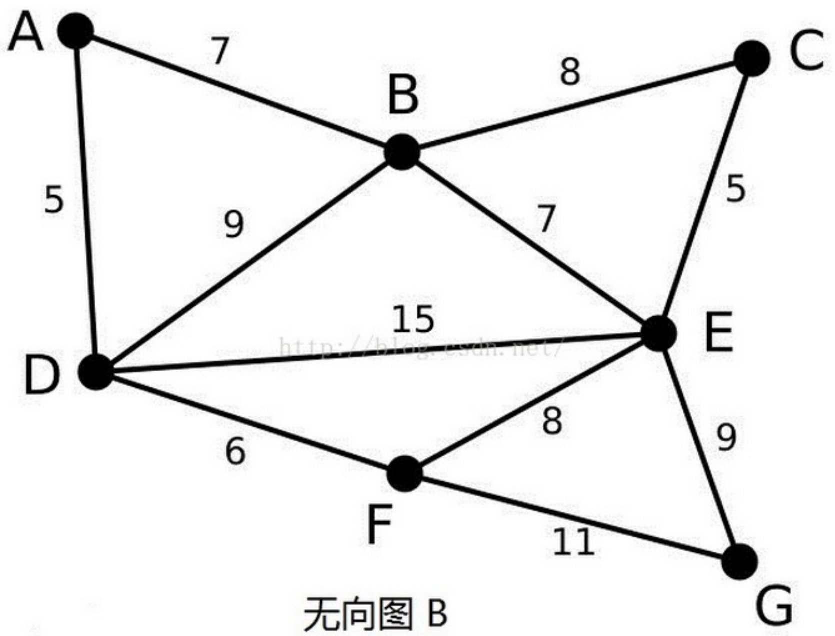
Implementation: Kruskal's Algorithm

Implementation. Use the union-find data structure.

- Build set T of edges in the MST.
- Maintain set for each connected component.
- $O(m \log n)$ for sorting and $O(m\, \underbrace{\alpha\, (m, n)}_{\text{essentially a constant}})$ for union-find.

$m \le n^2 \Rightarrow \log m$ is $O(\log n)$

Kruskal



无向图 B

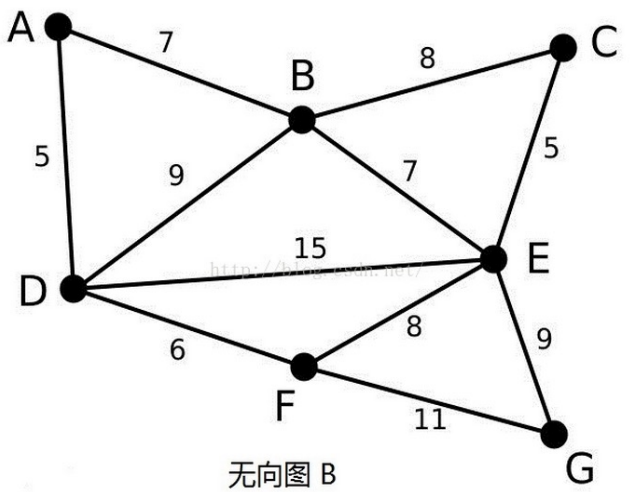
Kruskal:

1. Sorting all the edges
2. Finding the edge(n, m) with the smallest weight
3. Whether node n and node m are in a same tree?
   If yes, skip
   If no, merge two trees
4. If the number of node is N, we should merge N-1 times.
5. When merge two trees, add the w value

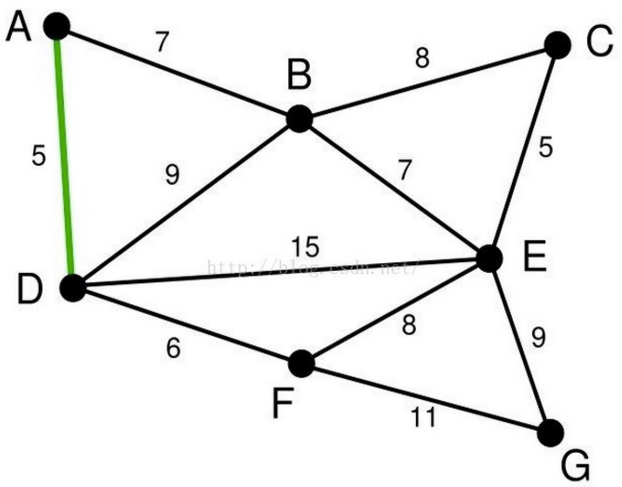
How to merge two trees (n, m)? Union-find

1. Find root of n and m respectively
2. If root of n equals to root of m, n and m is in a same tree. Skip
3. Get the height of root n and root m
   if(rootN.height > rootM.height) rootM.parent =rootN
   else if(rootN.height<rootM.height) rootN.parent=rootM
   else rootM.parent=rootN rootN.height++;

Sample Kruskal-1

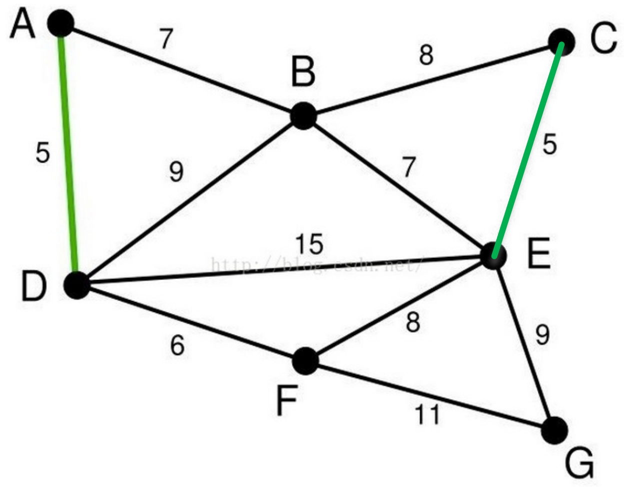


index	1	2	3	4	5	6	7
node	A	B	C	D	E	F	G
parent	0	0	0	0	0	0	0
height	0	0	0	0	0	0	0



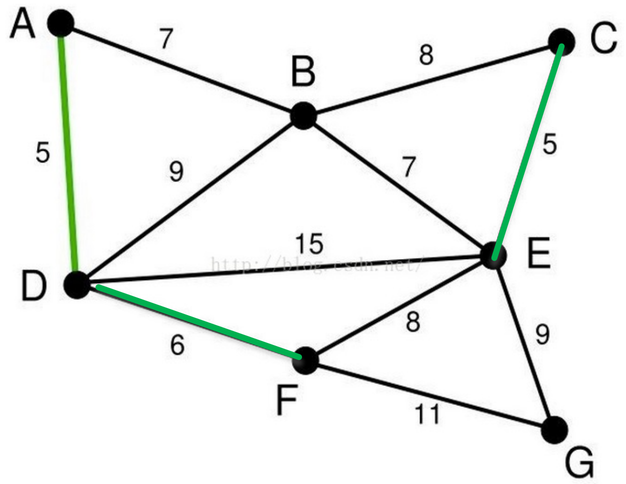
index	1	2	3	4	5	6	7
node	A	B	C	D	E	F	G
parent	0	0	0	1	0	0	0
height	1	0	0	0	0	0	0

Sample Kruskal-2



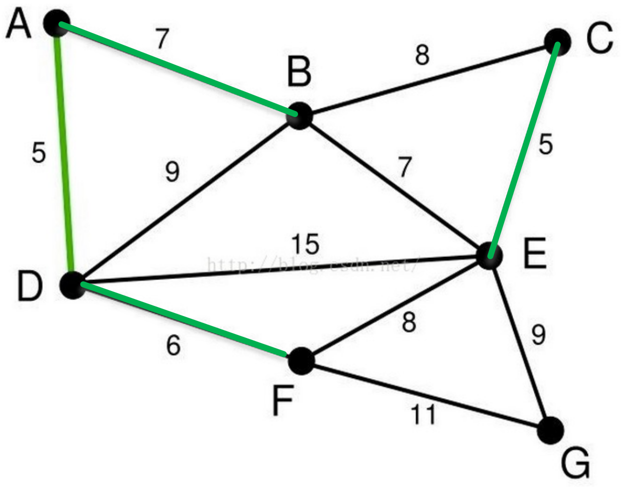
index	1	2	3	4	5	6	7
node	A	B	C	D	E	F	G
parent	0	0	0	1	3	0	0
height	1	0	1	0	0	0	0

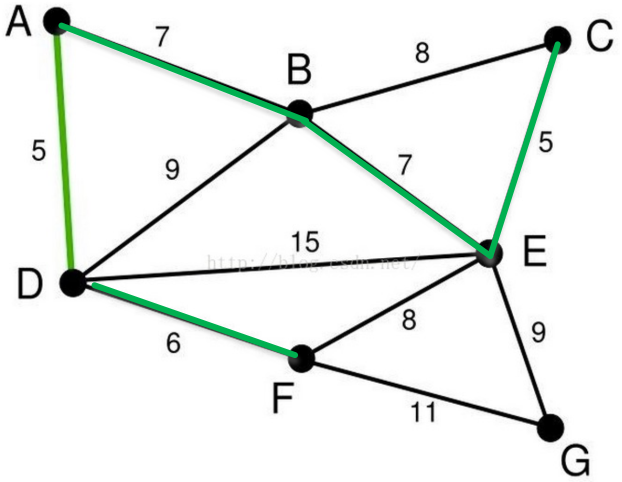
f(root). height <d(root). height



index	1	2	3	4	5	6	7
node	A	B	C	D	E	F	G
parent	0	0	0	1	3	1	0
height	1	0	1	0	0	0	0

Sample Kruskal-3





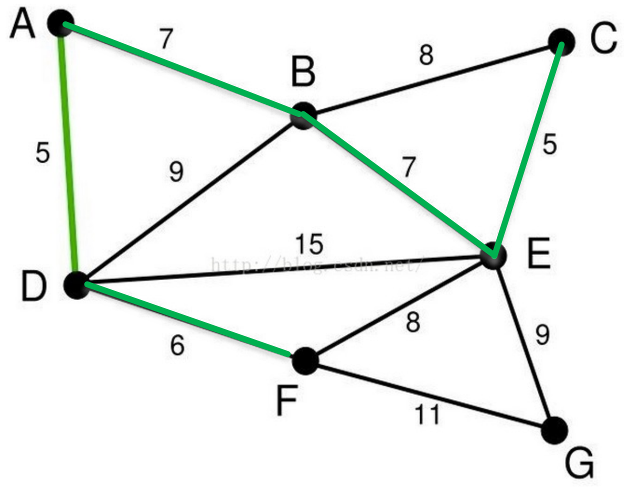
b(root). height <a(root). height
b.parent =a index

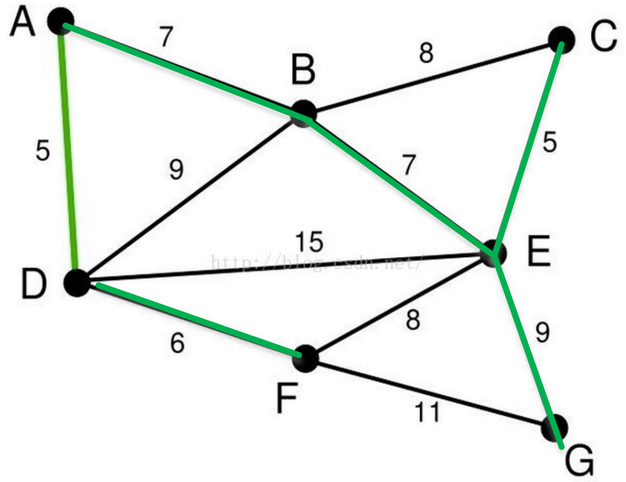
index	1	2	3	4	5	6	7
node	A	B	C	D	E	F	G
parent	0	1	0	1	3	1	0
height	1	0	1	0	0	0	0

e(root). height ==b(root). height
c.parent = a index
a. height ++

index	1	2	3	4	5	6	7
node	A	B	C	D	E	F	G
parent	0	1	1	1	3	1	0
height	2	0	1	0	0	0	0

Sample Kruskal-4





B (root)==C (root) skip
F (root) ==E (root) skip

e(root). height >g (root). height
g.parent=a

index	1	2	3	4	5	6	7
node	A	B	C	D	E	F	G
parent	0	1	1	1	1	1	1
height	2	0	1	0	0	0	0