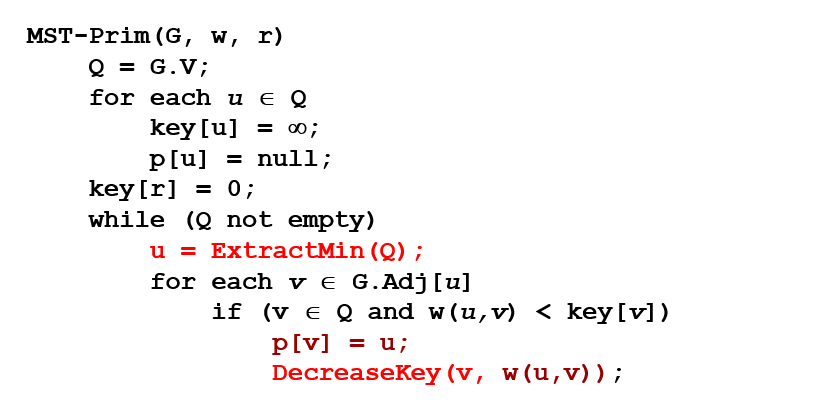


Prim using Prio Queue

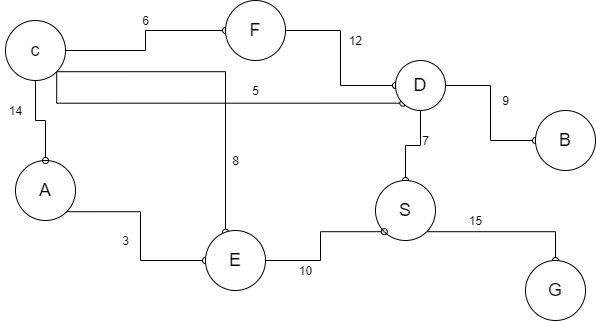


Initialization

* Maintain a priority queue. Init all values with infinity v.key = inf
* For an arbitrary vertex set v.key = 0
* Initial empty set S

Do this until the queue is empty:

* Extract min from the queue and add it to S
* Update adjacent values



Prio Queue:

A – inf

B – inf

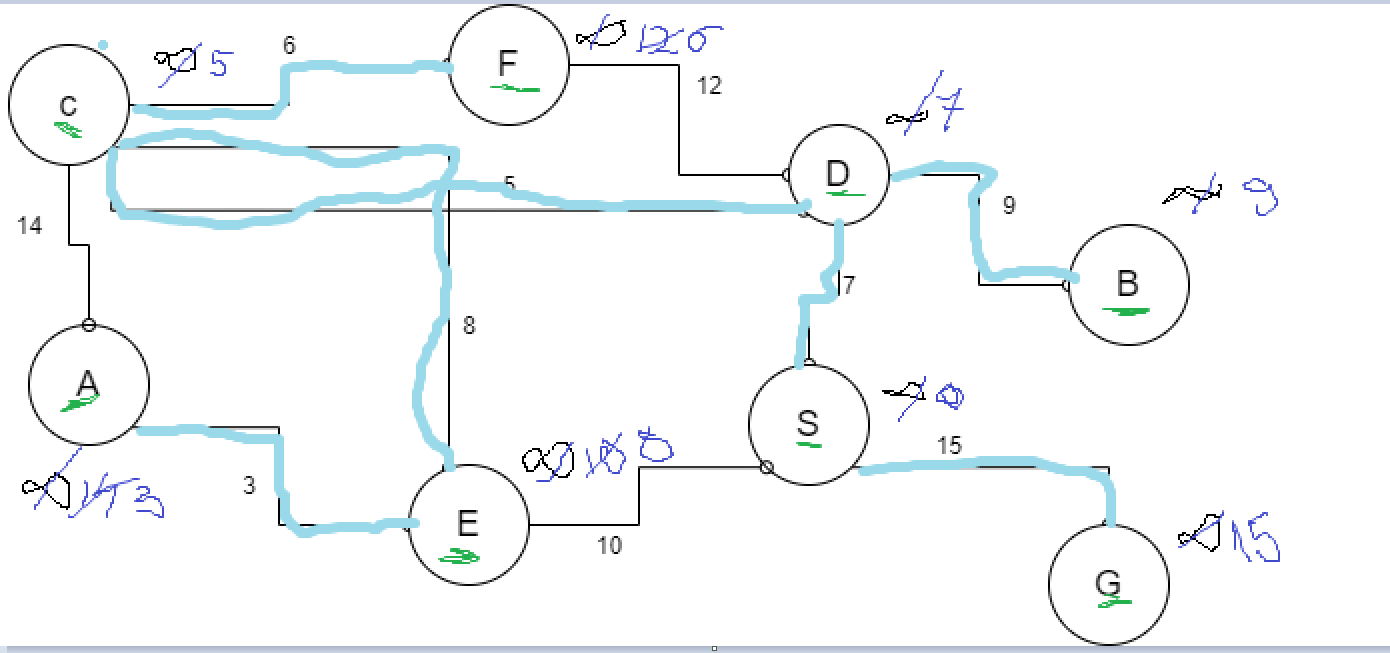
C – inf

…

S – 0

* Start with S -> extract S form prio queue -> add S to the set -> update adjacent vertices in prio queue
* Next is D -> extract D form prio queue -> add D to the set -> update values
* Next is C -> extract C form prio queue -> add C to the set -> update values
* Next is F -> extract F form prio queue -> add F to the set -> update values
* Next is E -> extract E form prio queue -> add E to the set -> update values
* Next is A, B and G

Underline all edges with the same weight as vertex key



Kruskal

Initialization

* Maintain connected components in disjoint set (union find)

(make set, find set, union)

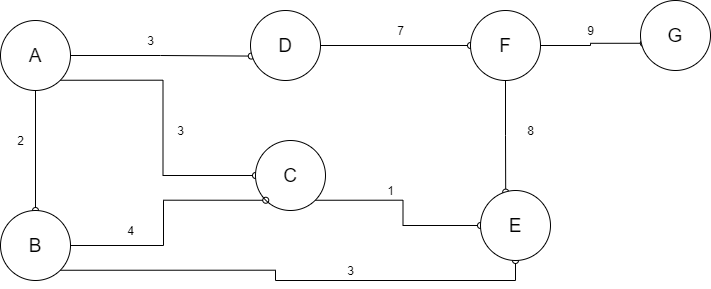
* Init an empty tree
* Add each vertex in a separate set (make set)

Do until one set remains

* Sort edges by weight (increasing)

For each edge e = {u,v}

if find-set(u) != find-set(v) then add to tree & union (u,v)



Before:

Set1: A

Set2: B

Set3: C

Set4: D

Set5: E

Set6: F

Set7: G

After:

Set1: A, B, D, C, E, F, G

* C – E in different sets => add C – E weight 1 in tree + union sets
* A – B in different sets => add A – B weight 2 in tree + union sets
* A – D in different sets => add A – D weight 3 in tree + union sets
* A – C in different sets => add A – C weight 3 in tree + union sets
* B – E in same set => skip
* B – C in same set => skip
* D – F in different sets => add D – F weight 7 in tree + union sets
* F – E in same set => skip
* F – G in different sets => add F-G weight 9 + union sets