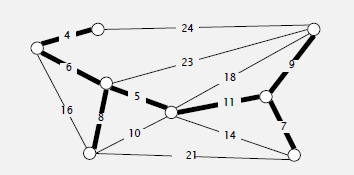
**Minimum Spanning Trees**

**Spanning Tree** Ështe një Nëngraf T i grafit G i cili është aciklik , i lidhur dhe i përfshinë të gjitha kulmet. **Minimum Spanning Tree** është spanning tree ku segementet kan pesha minimale. Një graf mund të ket shumë Spanning tree por ka vetëm një Minimum Spanning tree.

Në Undirected graphs kur peshat janë të pa caktuara si njësi e matjes përdoret **hopat**. **Hop count :** numri i nyjeve(kulmeve) qe kalonjë pët të arritur deri në destinacioin.por kjo nuk e cakton gjatesinë e rrugës , në fakt kjo gjatësi caktohet nga pesha e segementeve.



Më lart është paraqitur një minimun spanning tree , nëse i marin të gjitha peshat dhe i mbledhim së bashku do fitojmë qmimin(cost) të kti MST. Në këtë rast , çmimi = 4+6+5+11+9+7 = 50. Sa me i ult çmimi aq më i mirë MST.

Për të gjetur MST me çmimin më ultë mund të përdorim algoritmin **brute force** , algoritëm që gjen gjithomnë rezultatin duke i provuar të gjitha mundësitë por është shumë i ngadaltë (N2).

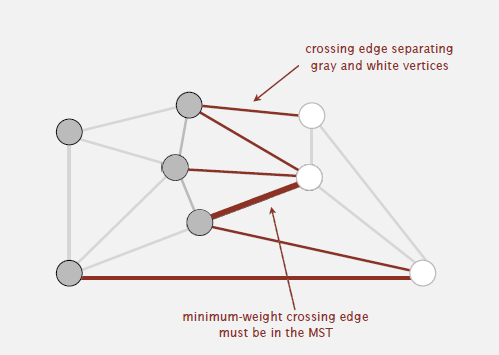
Si aplikim të MST mund të përmendim real-time face verification.

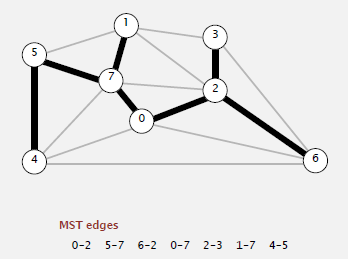
**Greedy Alorithm :** Algoritëm për gjetjen e Minimun Spanning tree. Është rast i përgjithshem i algoritmeve tjera greedy.

**Cut** : është ndarja e grafit në dy bashkësi jo boshe.

**Crossing edge** : është segmenti që lidh një kulm në njëren bashkësi me kulmin e bashkësis tjetër, të fituara nga cut.

**Minimum-weight crogssing edges :** janë crossing edges me peshen më të vogël kto të fundit vendosen në MST.

**Rregullat :** fillojmë me një prerje(cut) të grafit , prerje e cila duhet të plotësoj një kusht : mos ti prej kulmet të cilat janë të futura në MST (segmentet bold) , do të bëjm prerje deri të arrijm të fitojmë V-1 segemente bolded. Me rastin e prerjes fitojmë dy bashkësi në graf , segmentet të cilat i lidhin kulmet e këtyre dy bashkësive i quajm **crossing edges**. 

**Greedy Algorithm Demo :** Fillo kur të gjitha segmented janë të përhimta, Gjej cut pa crossing edges të zeza; ngjyrose min-weightedge në të zezë, Përserite derisa V-1 segmente janë të ngjyrosura zi. 

Weighted edge API

Public class Edge implements Comparable<Edge>

Edge(int v, int w, double weight) -krijon një segment me peshë mes kulmit v dhe w

Int either() -Kulmin nga duam të marim pesh

Int other(int v) -Kthen kulmet që janë të lidhura me v

Int compareTo(Edge that) -Krahason dy segmente

Double weight() -Pesha ktu përdoret double për të ulur shancen të kemi dy kulme të njëjta

String toString -reprezentim në String

Metoda compareTo(Edge that){

If(this.weight < that.wight) return -1;

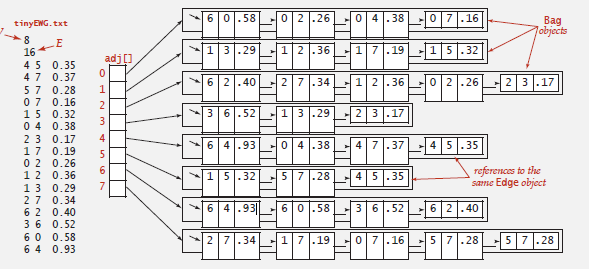
else if (this.weight > that.weight) return +1;

else return 0;

}

EdgeWeightedGraph lejojnë self-loops dhe paralel edges.

Njësoj si grafet pa pesh edhe grafet me pesh mund ti paraqesim në tri mënyra : **set-of-edges,adjacency-matrix** dhe **adjacency-lists.**

Paraqitje me adjancency-list : 

Minimum Spanning Tree API

Public class MST

Iterable<Edge>

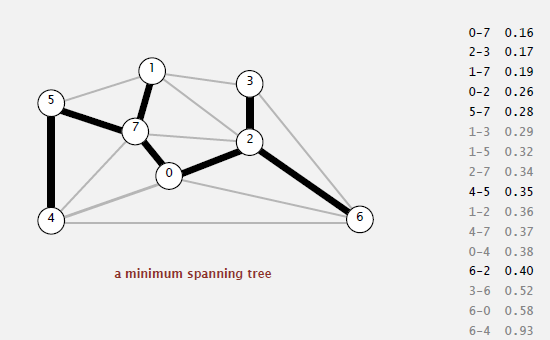
MST(EdgeWeightedGrap G) -konstruktori

H edges() -edges(segmentet) ne MST

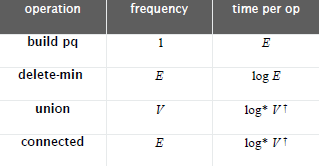
Double weight() -pesha e MST

Ekziostojnë edhe dy algoritme tjera që përdoren për gjetjen e MST , ato janë : **Kruskal** dhe  **Prim** alorithms. Kemi tri variacione të Prim algotimit : **general,lazy** dhe **eager.**

**Kruskal algorithms :** Është një greedy algoritëm ,qëllimi është të gjejm MST dhe të sigurohemi se aj MST do jet i njëti pa vartësisht nga lloji i algoritmit që përdoret.

**Rregullat :** I listojmë të gjitha segmentet dhe peshat e tyre të sortuara(me priority queue) nga pesha më e vogël deri tek më e madhja, pastaj duke filluar nga segmenti i par e deri në fund i vendosim në MST por jo edhe ato segmente të cilat krijojnë cikël.Dy segmentet e para ska nevoj të testohen a krijojn cikël shkak se është e pa mundur të ndodh një gjë tillë. 

Kurksual running time :

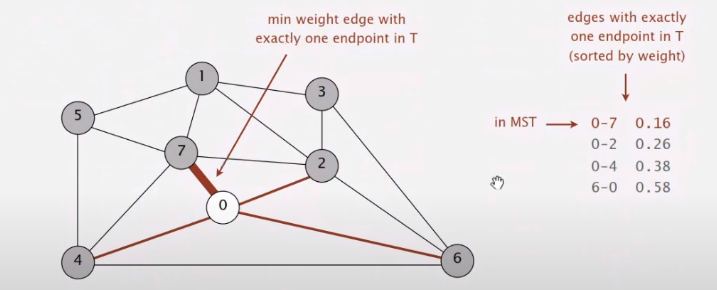


Nëse segmentet veq se janë të sortuara order of growth është ElogV.

**Prim’s algorithm :**

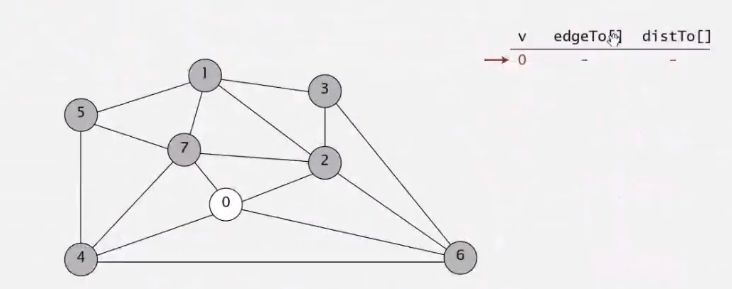
**General Impelemtion**

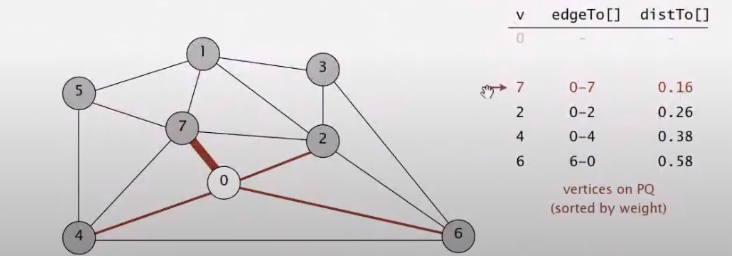
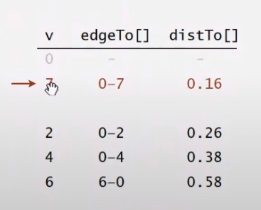
**Rregullat**: Në fillim i listojmë segmentet siç i listuam tek kruskal algoritmi ,fillojmë nga kulmi 0 vetëm nëse ipet një source tjetër dhe nga aty e rrisim MST, i marin të gjitha segmentet që hyjn në 0 i sortojmë sipas peshës dhe marim at edge që ka peshën më të vogel pra të parin edge nga lista e segmenteve të kulmit 0 , këtë edge e vendosim në MST. Pastaj vazhdojm tek kulmi ku ky segment na dërgon edhe nga ai nxjerrim të njejten list siç bëm me 0 , por i marin edhe ato segmentet të mbetura nga lista e mëparshme. Këtë proces e vazhdojmë deri të arrijm V-1 edges në MST.



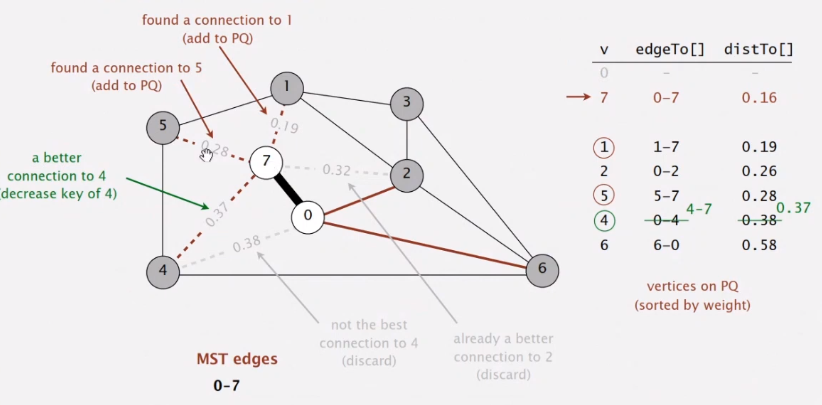
**Lazy Implemention :** Njësoj si tek impelmentimi gjeneral vetëm se ato segmente të cilat krijojnë cikël shënohen si të pa lejuara për tu futur në MST. Për ktë e ka mar emrin lazy , sepse segmentet që nuk na duhen nuk i fshin nga lista e krijuar me priority queue por vetëm i shenon si të pa përdorura , të cilat përsëri nxajn vend dhe përsëri testohen nga programi se a vlejnë apo jo.

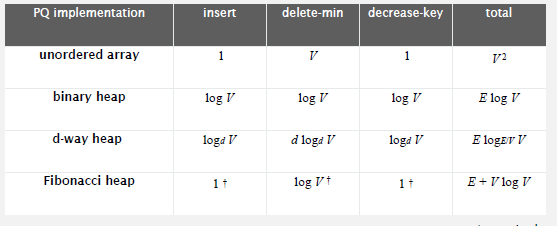
**Eager Implemention :** Si tek dy impelementimet e më parshme listojmë të gjitha segmentet e sortuara sipas peshës. Pastaj marim një tabel me tri kolona v, edgeTo[] , distTo[].



Pastaj i lisojmë të gjitha edges që dalin nga 0 , segmentin me peshën më të vogël e vendosim në tabelën e formuar më herët.

Pastaj vazhdojmë me 7 dhe marin edges që dalin prej kti kulmi, nëse fitojmë segmente të cilat na qojn në kulme të pa vizituara at segment e shtojm në list(PQ) , nëse ka edges që na qojn në kulme të vizituara e shkijom nëse ka pesh më të vogël nëse po e zëvendsojmë në list(PQ) me atë të vjetrën , nëse jo e fshijmë at segmente e injorojmë.Këto hapa i përsëritim deri të arrijm që në MST të kemi vendosur V-1 edges.



Prim’s algorithm run time sipas implementimit te PQ :

Më së mirë është të përdoret binary heap për të implementuar PQ.