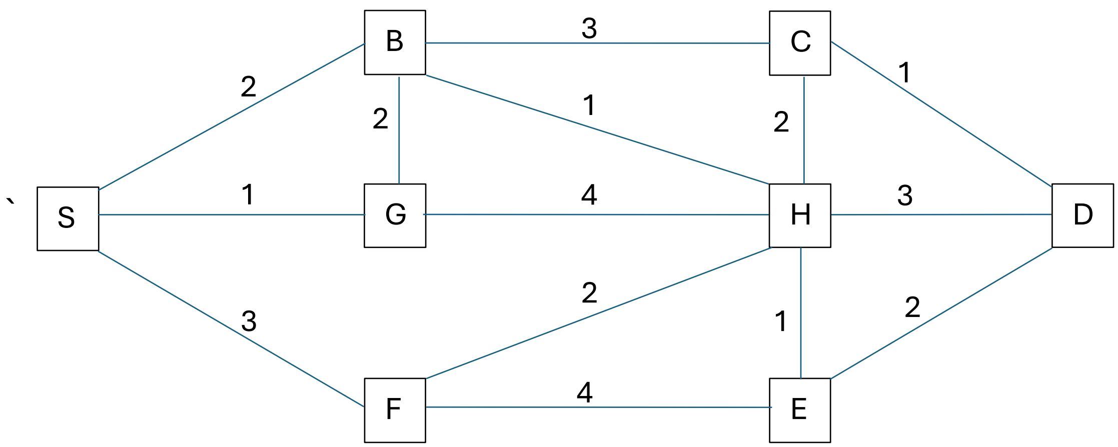
Dijkstra’s Shortest Path Greedy Algorithm O(m log n)

Prerequisites

1. Graph is undirected.
2. Weights are positive.



Summary

Repeat

1. Add one vertex at a time,
2. Compute distances to the vertices that can be reached.
3. Pick the vertex with minimum cost.

dis[S] =0. Path[S] = { }.

Compute the distance to all vertices that are adjacent to S.

dis[B] = dis[S] + wt(S, B) = 0 + 2 = 2.

dis[G] = dis[S] + wt(S, G) = 0 + 1 = 1.

dis[F] = dis[S] + wt(S, F) = 0 + 3 = 3.

Pick the vertex that can be reached with <u>minimum cost.</u> (Greedy approach).

Pick G.

Value of G will never change. Value of G is finalized.

dis[G] = dis[S] + wt(S, G) = 1.

Path[G] = path[S] $\cup$ {(S, G)} = {(S, G)}.

Compute the distance to all vertices that are adjacent to G.

dis[B] = dis[G] + wt(G, B) = 1 + 2 = 3. (This is larger dis[B]. Ignore.)

dis[H] = dis[G] + wt(G, H) = 1 + 4 = 4.

Pick the vertex that can be reached with <u>minimum cost</u> from S or B.

Pick B.

Value of B will never change. Value of B is finalized.

dis[B] = dis[S] + wt(S, B) = 2.

Path[B] = path[S] $\cup$ {(S, B)} = {(S, B)}.

Compute the distance to all vertices that are adjacent to B.

dis[C] = dis[B] + wt(B, C) = 2 + 3 = 5.

dis[H] = dis[B] + wt(B, H) = 2 + 1 = 3.

Pick the vertex that can be reached with minimum cost from S, G, or B.

Pick H.

Value of H will never change. Value of H is finalized.

dis[H] = dis[B] + wt(B, H) = 3.

Path[H] = path[B] ∪ {(B, H)} = {(S, B), (B, H)}.

Compute the distance to all vertices that are adjacent to H.

dis[C] = dis[H] + wt(H, C) = 3 + 2 = 5.

dis[F] = dis[H] + wt(H, F) = 3 + 2 = 5.

dis[E] = dis[H] + wt(H, E) = 3 + 1 = 4.

dis[D] = dis[H] + wt(H, D) = 3 + 3 = 5.

Pick the vertex that can be reached with minimum cost from S, G, B, or H.

Pick F.

Value of F will never change. Value of F is finalized.

dis[H] = dis[S] + wt(S, F) = 3.

Path[H] = path[S] ∪ {(S, F)} = {(S, F)}.

Compute the distance to all vertices that are adjacent to F.

dis[E] = dis[F] + wt(F, E) = 3 + 4 = 7. (This is larger dis[E]. Ignore.)

Pick the vertex that can be reached with minimum cost from S, G, B, H or F.

Pick E.

Value of E will never change. Value of E is finalized.

dis[E] = dis[H] + wt(H, E) = 4.

Path[E] = path[H] $\cup$ {(H, E)} = {(S, B), (B, H), (H, E)}.

Compute the distance to all vertices that are adjacent to E.

dis[D] = dis[E] + wt(E, D) = 4 + 2 = 6.

Pick the vertex that can be reached with minimum cost from S, G, B, H, F or E.

Pick C.

Value of C will never change. Value of C is finalized.

dis[C] = dis[B] + wt(B, C) = 5.

Path[C] = path[B] $\cup$ {(B, C)} = {(S, B), (B, C)}.

Compute the distance to all vertices that are adjacent to C.

dis[D] = dis[C] + wt(C, D) = 5 + 1 = 6.

Pick the vertex that can be reached with minimum cost from S, G, B, H, F, E or C.

Pick D.

Value of D will never change. Value of D is finalized.

dis[D] = dis[C] + wt(C, D) = 6.

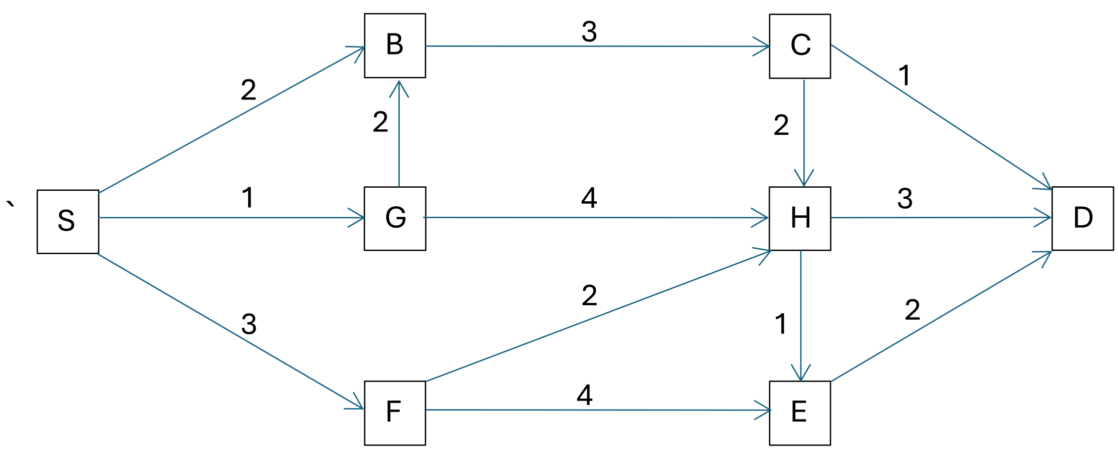
Path[D] = path[C] $\cup$ {(C, D)} = {(S, B), (B, C), (C, D)}.

THE END

Dijkstra’s Dynamic Programing Algorithm   O(n + m)

Prerequisites

1. Graph is directed.
2. Graph is acyclic.
3. Negative weights allowed.



Summary

Step 1.

Perform topological ordering of all vertices.

Repeat

1. Pick the vertex **N** at based on the topological ordering.
2. Compute distances to N for each in coming edde.
3. Compute the minimum.

Topological Ordering: SFGBCHED

dis[S] =0.   path[S] = { }.

---

dis[F] = dis[S] + wt(S, F) = 3.

Path[F] =path[S] $\cup$ {(S, F)} = {(S, F)}.

$dis[G] = dis[S] + wt(S, G) = 1.$

$path[G] = path[S] \cup \{(S, G)\} = \{(S, G)\}.$

$dis[B] = \min\{ dis[S] + wt(S, B) = 0 + 2 = 2$ ⟸

$\qquad dis[G] + wt(G, B) = 1 + 2 = 3 \}$

$path[B] = path[S] \cup \{(S, B)\} = \{(S, B)\}.$

$dis[C] = dis[B] + wt(B, C) = 2 + 3 = 5.$

$path[C] = path[B] \cup \{(B, C)\} = \{(S, B), (B, C)\}.$

$dis[H] = \min\{ dis[C] + wt(C, H) = 5 + 2 = 7$

$\qquad dis[G] + wt(G, H) = 1 + 4 = 5$ ⟸

$\qquad dis[F] + wt(F, H) = 3 + 2 = 5 \}$

$path[H] = path[G] \cup \{(G, H)\} = \{(S, G), (G, H)\}.$

$dis[E] = \min\{ dis[H] + wt(H, E) = 5 + 1 = 6$ ⟸

$\qquad dis[F] + wt(F, E) = 3 + 4 = 7 \}$

$path[E] = path[H] \cup \{(H, E)\} = \{(S, G), (G, H), (H, E)\}.$

$dis[D] = \min\{ dis[C] + wt(C, D) = 5 + 1 = 6$ ⟸

$\qquad dis[H] + wt(H, D) = 5 + 3 = 8$

$\qquad dis[E] + wt(E, D) = 6 + 2 = 8 \}$

$path[D] = path[C] \cup \{(C, D)\} = \{(S, B), (B, C), (C, D)\}.$

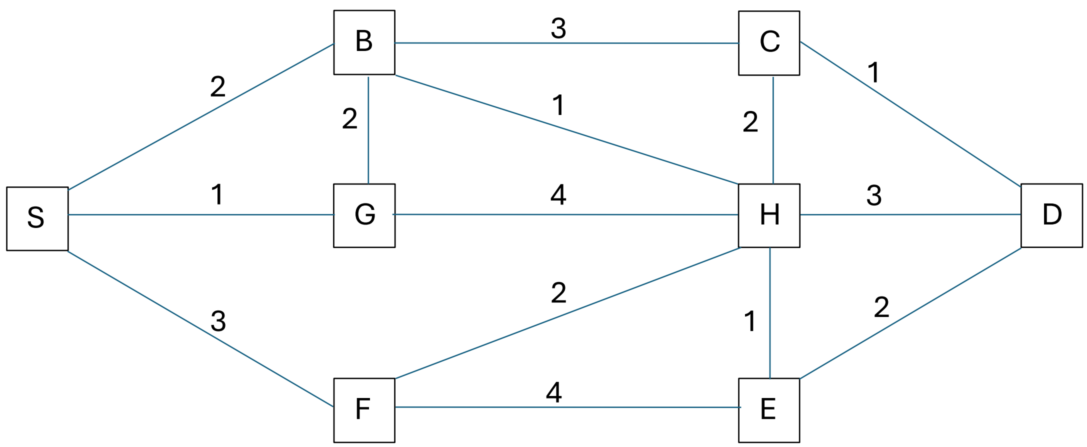
THE END

Kruskal’s Minimum Spanning Tree Algorithm   O(m log n)

Prerequisite

1. Graph is undirected.
2. Graph has positive weights.

`



Summary

Step 1. Sort the edges by weight and keep it in a list **L.**

Step 2. Initialize a Union-Find data structure with vertices such that

each vertex is a singleton.

Step 3.

Repeat

Pick next edge (x, y) from the list **L.**

if (Find(x) == Find(y)) Union (x, y)

else delete (x, y) from the list L.

Output: L contains all the edges of the minimum spanning tree.

,   The sum of all weights of edges in L gives the weight of MST.

Sorted List of edges

(A, G)

(C, D)

(H, E)

(B, H)

(A, B)

(B, G)

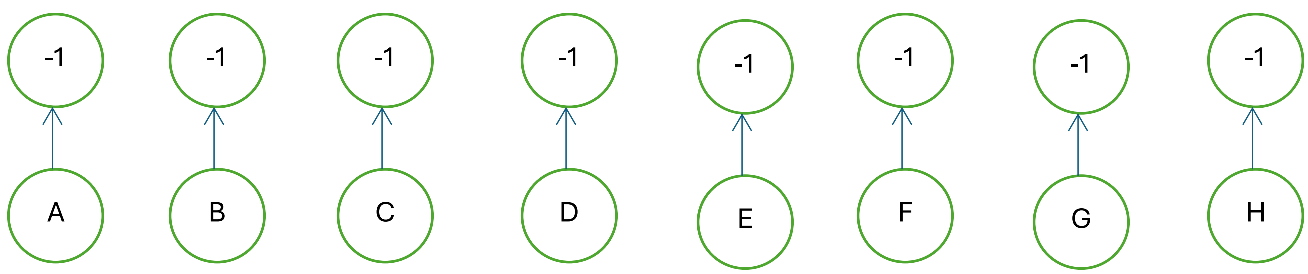
(F, H)

(H, C)

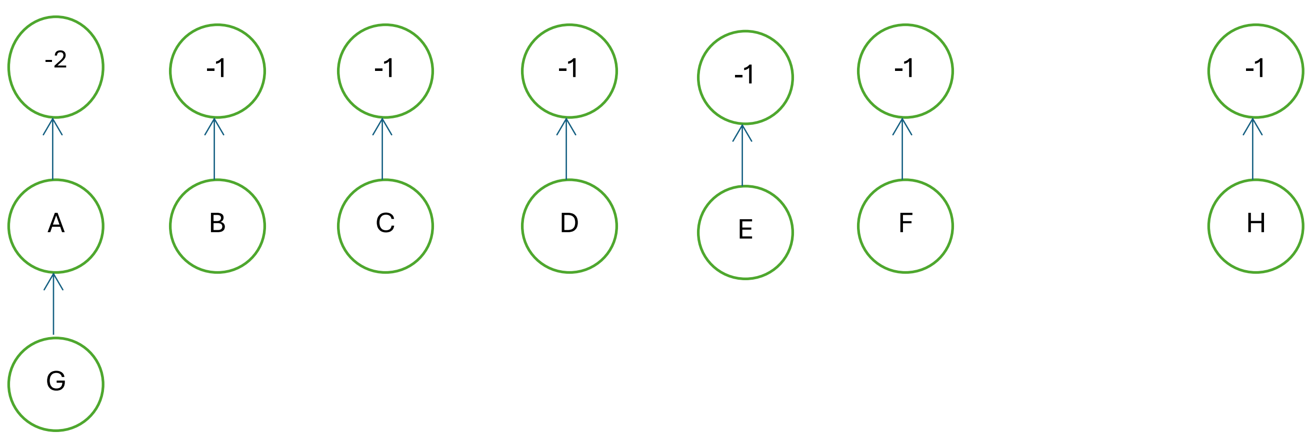
(D, E)

...

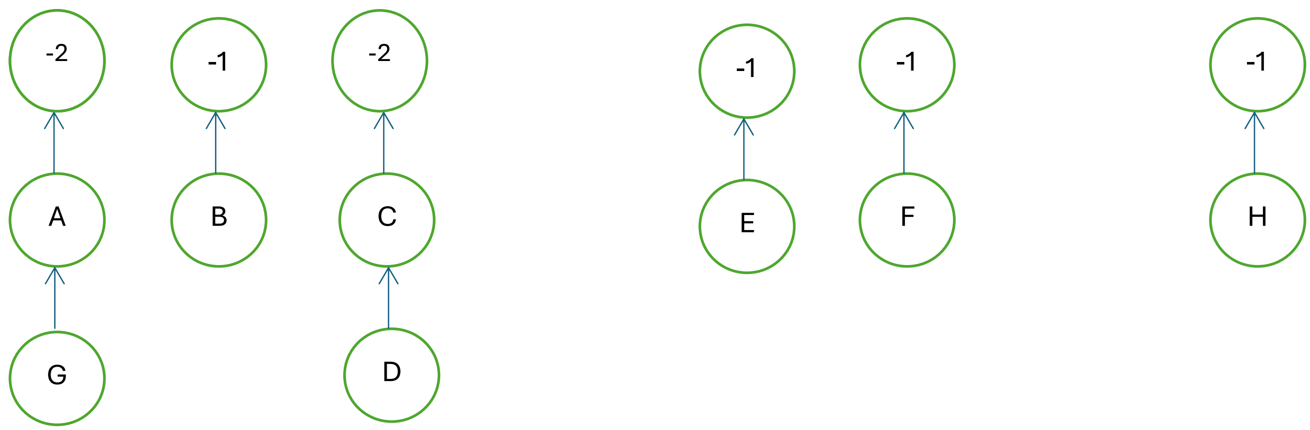
Initialize Union-Find



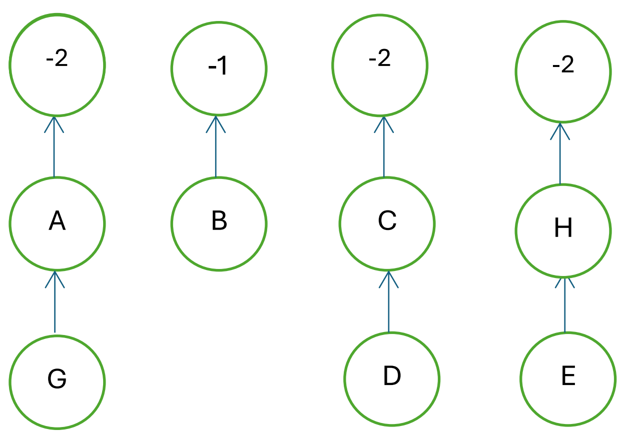
Find(A) $\neq$ Find(G). Union(A, G).



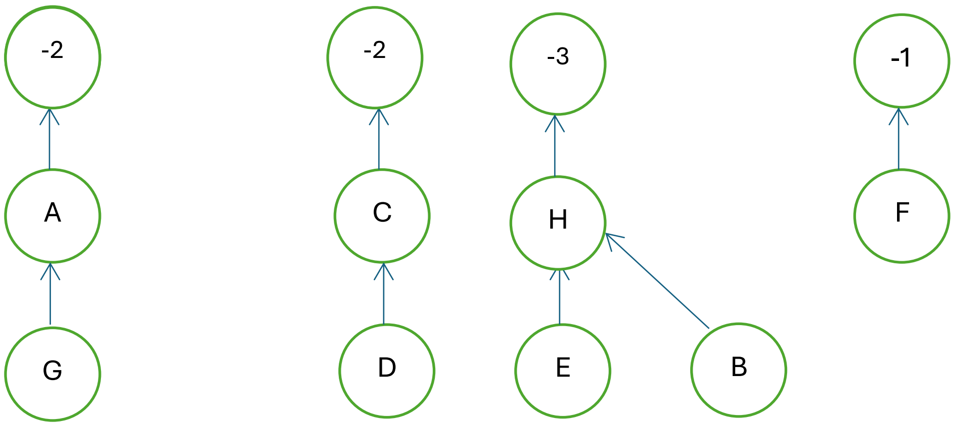
Find(C) ≠ Find(D). Union(C, D)



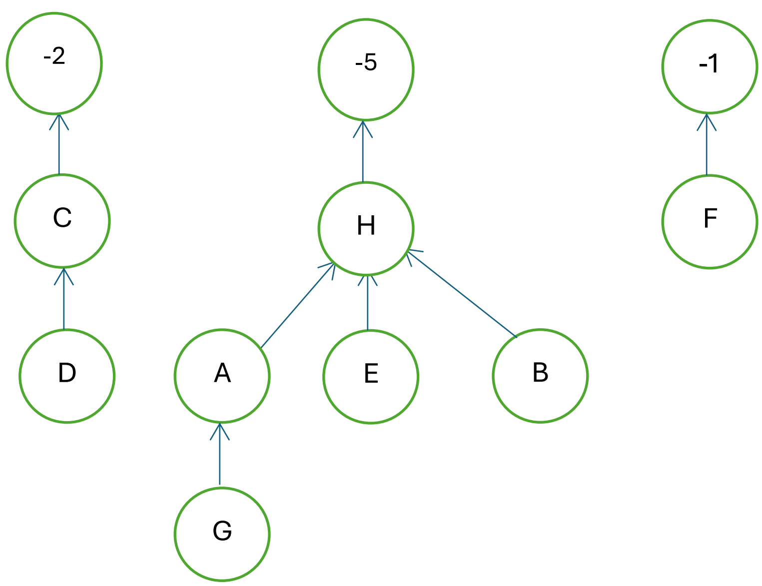
Find(H) ≠ Find(E). Union(H, E)



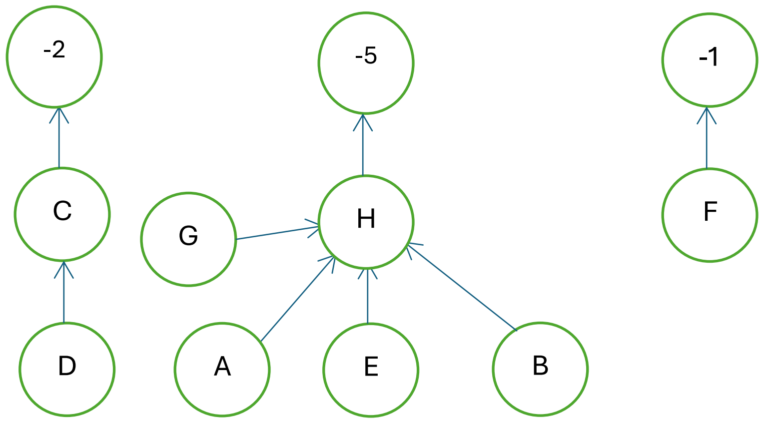
Find(B) ≠ Find(H). Union(B, H)



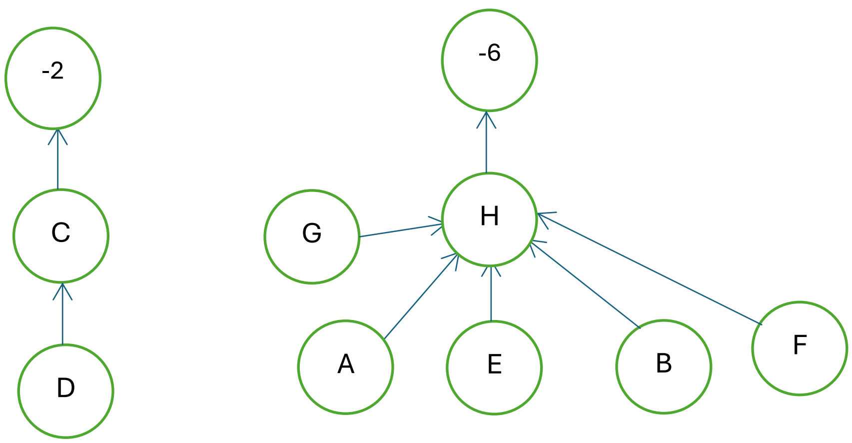
Find(A) ≠ Find(B). Union(A, B)



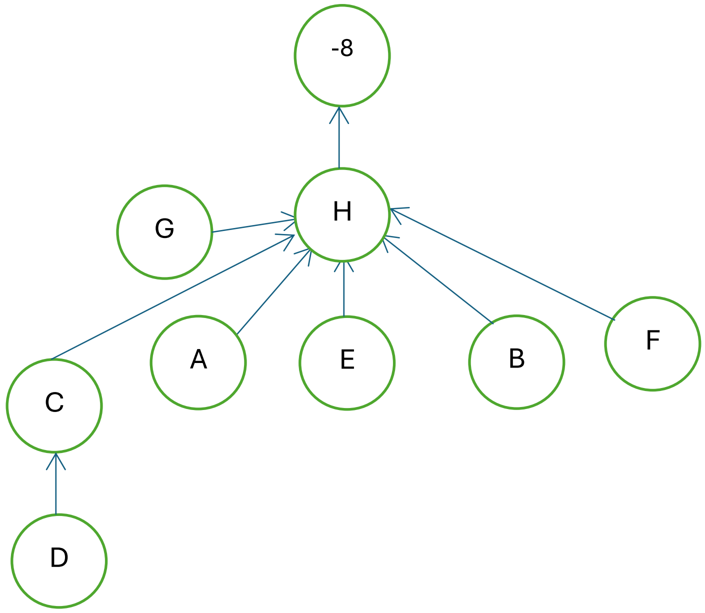
Find(B) == Find(G). Delete(A, B). Find(G) will compress H-A-G path to H-G.



Find(F) ≠ Find(H). Union(F, H)



Find(F) ≠ Find(C). Union(F, B)



Find(F) == Find(D) will delete edge (D, E) from the list L.

The same will happen to remaining edges. Thus the edges of the spanning tree are

(A, G) 1

(C, D) 1

(H, E) 1

(B, H) 1

(A, B) 2

(F, H) 2

(H, C) 2

Total weight of the spanning Tree is 1+1+1+1+2+2+2=10.

THE END