Minimum Spanning Trees

Minimum Spanning Tree Problem

- Input : G(V, E)

  $$c : E \to \mathbb{N}$$

- Output : $T \subseteq E$ such that G(V, T) is connected and $\sum_{e \in T} c(e)$ is minimised.

Minimum Spanning Tree Problem

- Input : G(V, E)

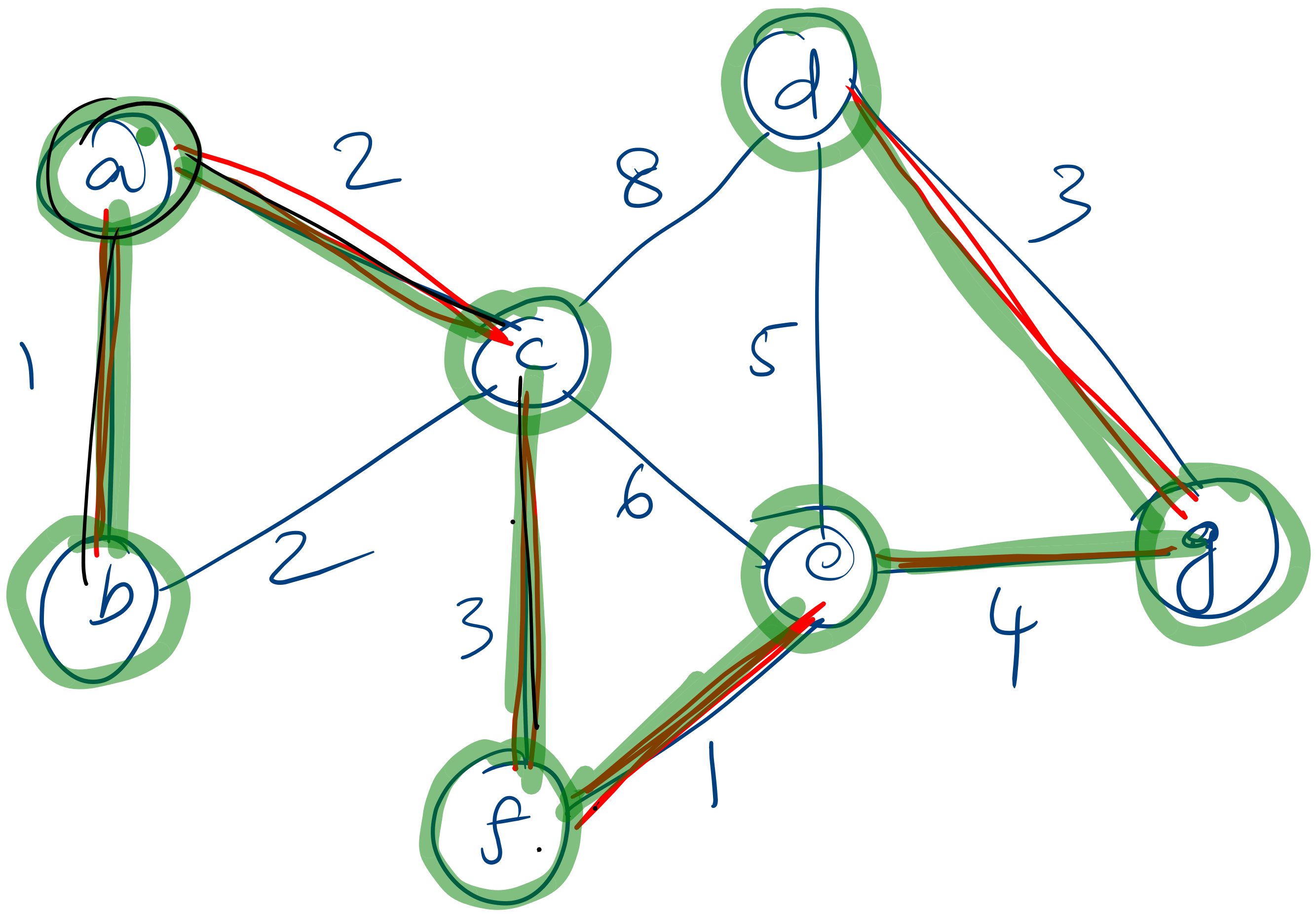
$$c : E \to \mathbb{N}$$

- Output : $T \subseteq E$ such that G(V, T) is connected and $\sum_{e \in T} c(e)$ is minimised.

Can be solved using greedy strategy

Kruskal's Algorithm

- Insert edges in order of increasing cost so that no cycles are formed.



	key	π
a	0	NIL
b	1	a
c	2	a
d	~~∞~~ 8	c
e	~~∞~~ ~~6~~ 1	~~c~~ f
f	~~∞~~ 3	c
g	∞	

Prim's Algorithm

- Start with a root node s and try to greedily grow a tree from s outward.
- At each step, add the node that can be attached as cheaply as possible to the partial tree we already have.

for all v

visited[v] = F. $\Big]\ O(n)$

E : edges sorted by c. — $m \log m$

visited[a] = T ; $F = \phi$

while $\exists v \in V$, s.t visited[v] = F.

find the first $e = (u,v)$ in E s.t

visited[u] = T & visited[v] = F

add e to F

delete e from E

visited[v] = T. $\Big]\ n \cdot m$

Kruskal's Algorithm

```
A = ∅

Add each vertex v to a separate component of A

Sort the edges of E by weight

For each edge (u,v) in order

  if u and v are not in the same component

    A = A ∪ {u,v}
```

Kruskal's Algorithm

Implementation and Running Time :

- sort the edges

- checking whether two elements are in the same component and merge (O(log n) time using Union Find data structure)

$O(n + m \log m + m \log n)$

Prim's Algorithm

```
For each u ∈ V
    key[u] = ∞      - min cost needed to add u to the solⁿ.
    π[u] = NIL      - The nbr of u that realised key.
key[r] = 0
Q = V  —— ordered by key
while Q ≠ ∅
    u = ExtractMin(Q), Add(u, π(u))
    For each v ∈ Adj[u]
        if v ∈ Q and w(u,v) < key[v]
            π[v] = u
            key[v] = w(u,v)
```

Kruskal’s Algorithm

Implementation and Running Time :

- Find the minimum element from Q (n times)

- Update the value of key (m times)

Kruskal’s Algorithm

Implementation and Running Time :Using binary heap

- Find the minimum element from Q (n times) - n log n

- Update the value of key (m times) - m log n

  - O((m+n) log n)

Kruskal's Algorithm

Implementation and Running Time :Using Fibonacci heap

- Find the minimum element from Q (n times) - n log n

- Update the value of key (m times) - m

  - O(m+n log n)

Proof of correctness

- some edges are always safe to be added to a MST

Proof of correctness

Cut Property

- Assume all edge costs are distinct
- Let $S \subset V, S \neq \emptyset$
- Let e be the minimum cost edge with one end in S and the other edge in V-S
- Every MST contains the edge e