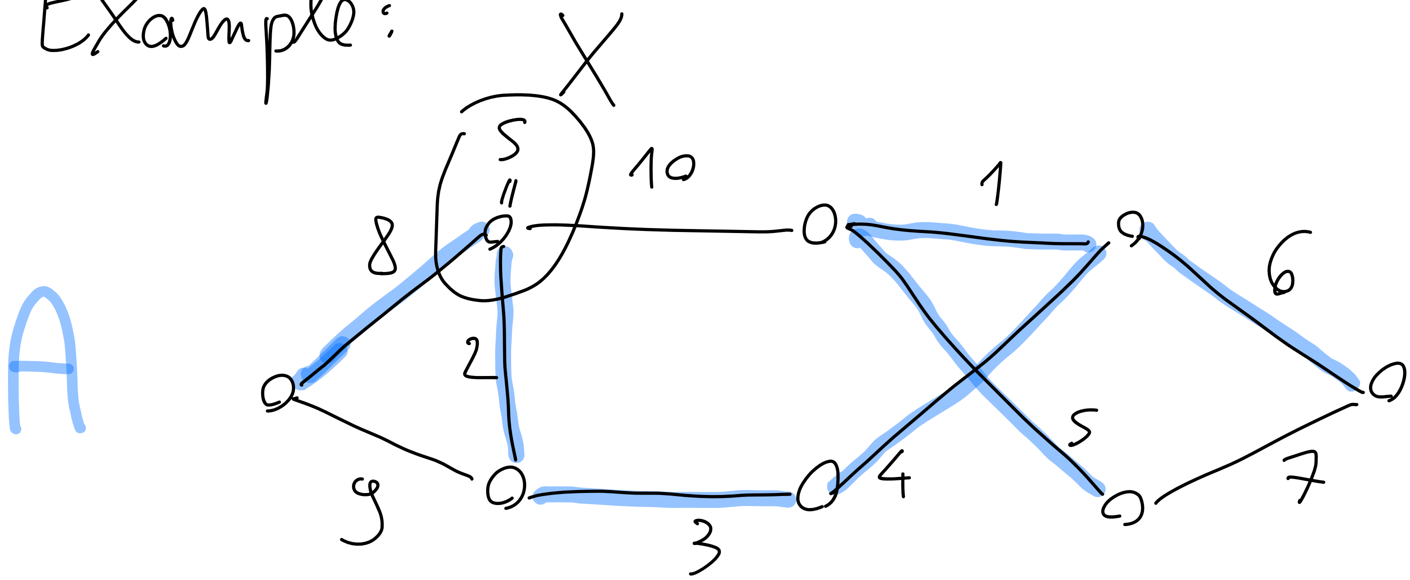
Example:



Correctness: follows from the Theorem

Complexity: $O(m \cdot n)$ (assume G represented with an adjacency list)

polynomial time $\Rightarrow$ efficient

Is $O(m \cdot n)$ really efficient in practice?

Think of Facebook graph: $n \simeq 2B$

$m \simeq 2B \cdot$ hundreds

So, not so efficient in very large graphs

Key observation: in the basic implementation the computation of the light edge is done <u>repeatedly</u> $\Rightarrow$ should speed it up

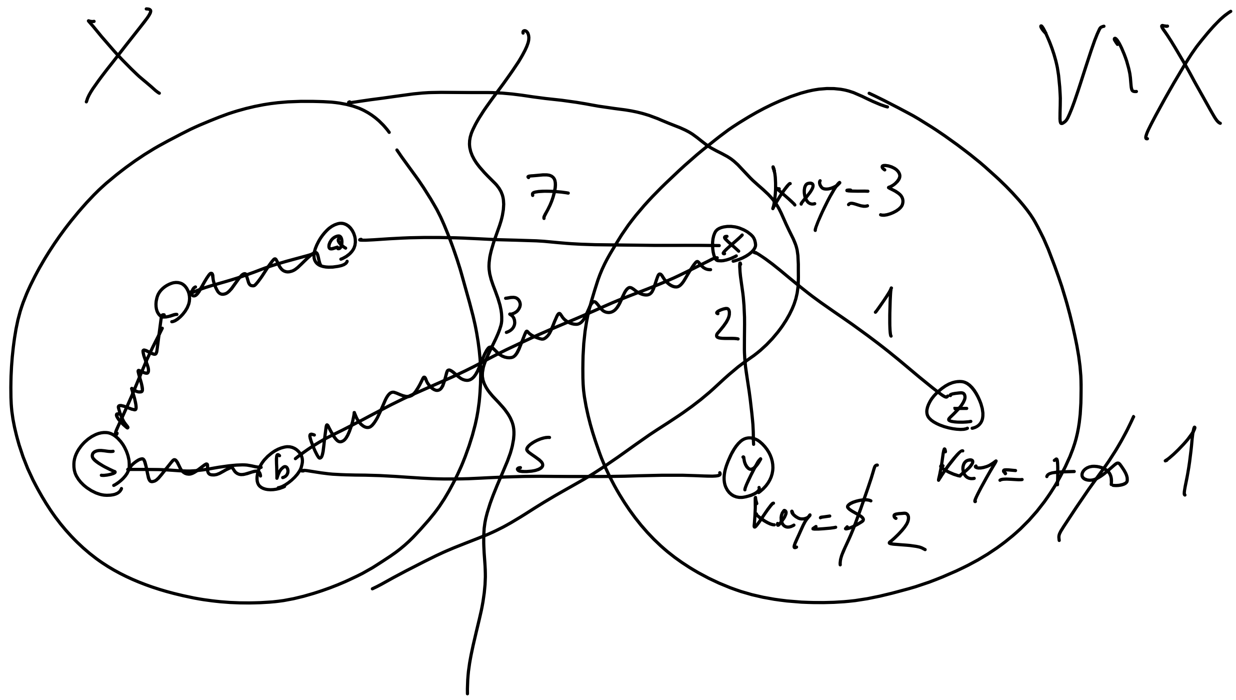
<u>golden rule</u> in algorithms/coding: when an alg. repeats frequently the same operation, look for the "right" data structure to speed that operation up

⟶ here, a priority queue is what we need

⟶ implemented with a <u>Heap</u>:

Recap:
- INSERT: add an object to the heap
- EXTRACT-MIN: remove an object with the smallest key
- DELETE: given a pointer to an object, remove it

⟶ in a heap with n objects: $O(\log n)$



it's simpler to store vertices in the heap (instead of edges)

Prim's implementation with a heap:

Prim(G, s)

for each $v \in V$ do

$key(v) = +\infty$ \\ min. weight of any edge connecting v to the tree

$\pi(v) = NULL$ \\ parent of v in the tree

$key(s) = 0$

$H = V$ // contains all vertices not in tree

while $H \neq \emptyset$ do

  $v^* = \text{extractMin}(H)$

  for each v adjacent to v^* do

    // update key and π of each vertex adjacent to v^* but not in tree

    if $v \in H$ and $w(v^*, v) < \text{key}(v)$

      $\pi(v) = v^*$

      // $\text{key}(v) = w(v^*, v)$

      delete v from H

      $\text{key}(v) = w(v^*, v)$

      insert v into H

$$A = \{ (v, \pi(v)) : v \in V \setminus \{s\} \setminus \{H\} \}$$

Complexity: init $\rightarrow O(n)$

while $\longrightarrow$ n iterations

extractMin $\longrightarrow$ $O(\log n)$

---

total cost of extractMin $O(n \log n)$

for loop : executed $O(m)$ times in total

- $v \in H \to O(1)$
- $key(v) \to$ delete + insert : $O(\log n)$

total cost of for loop $O(m \log n)$

Total : $O(n \log n + m \log n) =$

$= O(m \log n)$

recall : G is connected

it's near-linear time

Exercise : (Uniqueness of MSTs) Show that if the weights of the edges are all distinct then there exists exactly one MST.
(Hint : cut & paste argument)

Kruskal's algorithm (1956)

- it's very simple, very famous
- as fast as Prim, both in theory and in practice
- it gives us the opportunity to study a new data structure

GENERIC-MST(G) $\longrightarrow$ A: is a forest

$\searrow$ safe edge: a light edge connecting 2 distinct components

KRUSKAL(G) // no source vertex needed

$A = \emptyset$

sort edges of G by weight

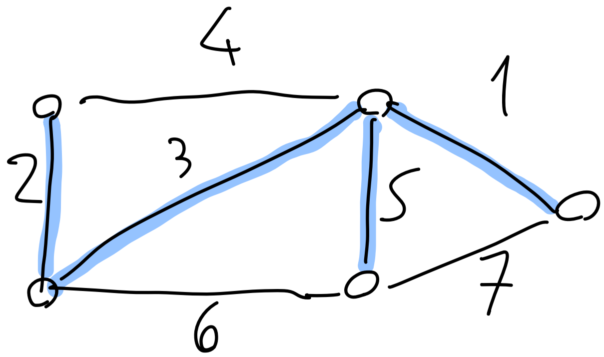
for each edge e, in nondecreasing order of weight do

if $A \cup \{e\}$ is acyclic then

$A = A \cup \{e\}$

return A

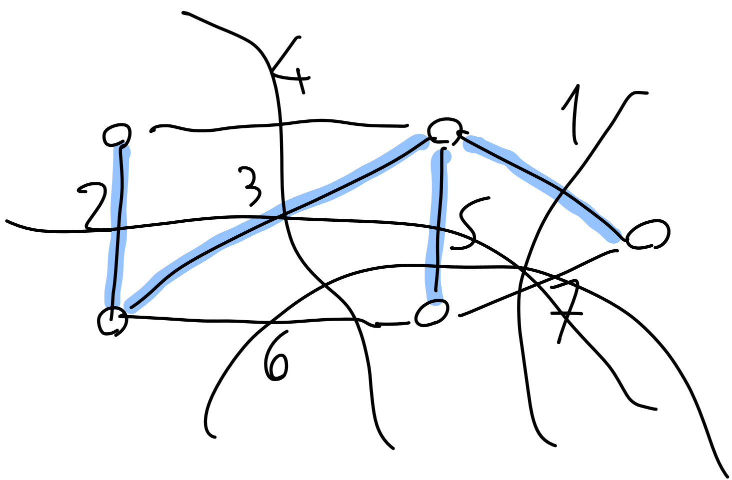
Example :



A

(simple optimization : stop the for loop when A has $n-1$ edges)

Correctness: follows from correctness of GENERIC-MST



Complexity: sorting: $O(m \log n)$

for loop: check whether $e=(u,v)$ closes a cycle is equivalent

to check whether A contains an u–v path $\longrightarrow$ DFS on $G=(V,A)$

$\rightarrow$ complexity $O(n)$

Total: $O(m \cdot n)$

Can we implement it faster?