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

Think of FB graph: $n \simeq 2.5$ B
$m \simeq 2.5$ B $\cdot$ hundreds

not so efficient in very large graphs
↳ FB-scale

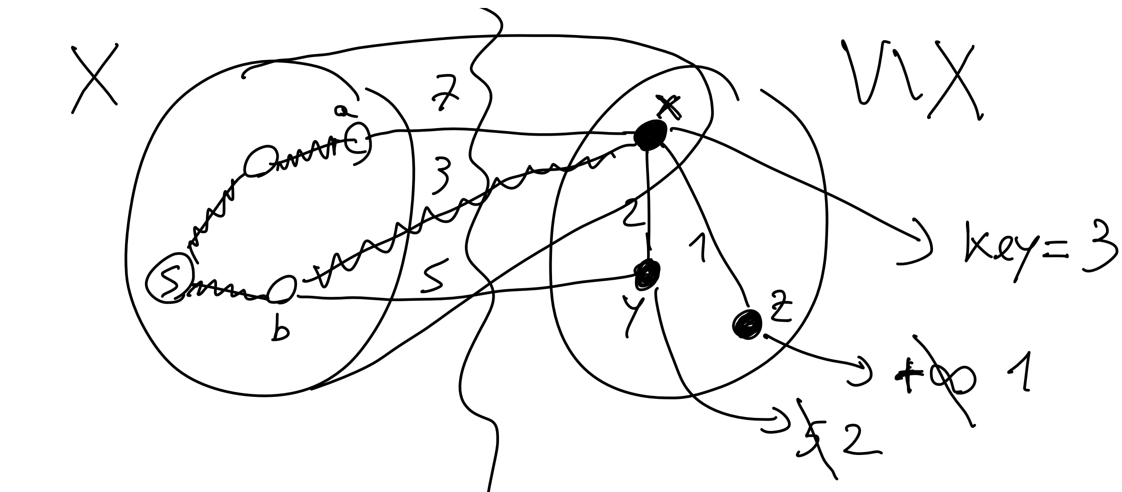
Key observation: in the basic implementation the calculation of a min is done <u>repeatedly</u>
$\Rightarrow$ should speed it up

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

A HEAP is exactly what we need now

Recap: Heap:
- insert: add an object to the heap
- extractMin: remove an object with 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(G, s)
  for each v ∈ V do
      key(v) = +∞          \\ min. weight of any edge connecting v to the tree
      π(v) = NULL          \\ parent of v in the tree
  key(s) = 0
  H = V                    \\ contains all vertices not in the tree
  while H ≠ ∅ do
      v* = 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 ∈ H and w(v*, v) < key(v)
              π(v) = v*
              \\ key(v) = w(v*, v)
              delete v from H
              key(v) = w(v*, v)
              insert v into H
```

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

Complexity: init: $O(n)$

while $\rightarrow$ n iterations

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

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

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

– $v \in H \rightarrow O(1)$

– key$(v) \rightarrow$ delete + insert: $O(\log n)$

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

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

"near-linear" time complexity $O(m \cdot \log^{O(1)} n)$

recall: G is connected

Exercise: (uniqueness of MSTs) Show that if the weights of the edges are all distinct then there exists exactly one MST.

<u>Kruskal's algorithm</u> (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

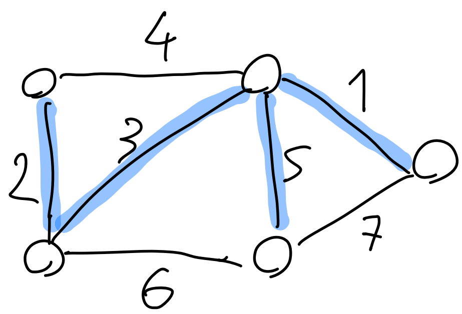
GENERIC-MST(G) → A : is a forest

→ safe edge: a light edge connecting 2 distinct components

KRUSKAL(G) // no source vertex needed

```
A = ∅
sort edges of G by weight          // e.g. using MergeSort
for each edge e, in nondecreasing order of weight do
    if A ∪ {e} is acyclic then
        A = A ∪ {e}
return A
```

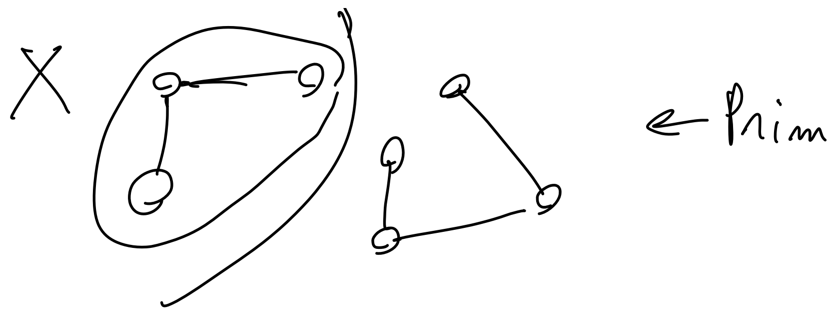
Example:

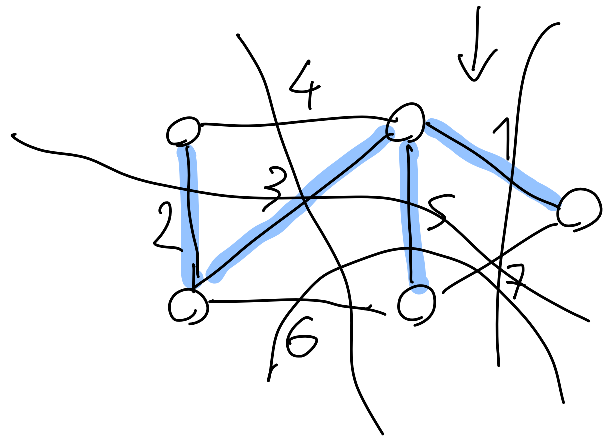


A

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

$\longrightarrow$ complexity: $O(n)$

Total: $O(m \cdot n)$

Can we implement Kruskal's algorithm faster?