Mini Lecture: Running Time of Merging

COSC 311 *Algorithms*, Fall 2022

Last Time

Kruskal's Algorithm for MSTs:

- iterate over all edges in ascending order of weight
- if an edge connects two previously un-connected components, add it to MST

Kruskal's Algorithm

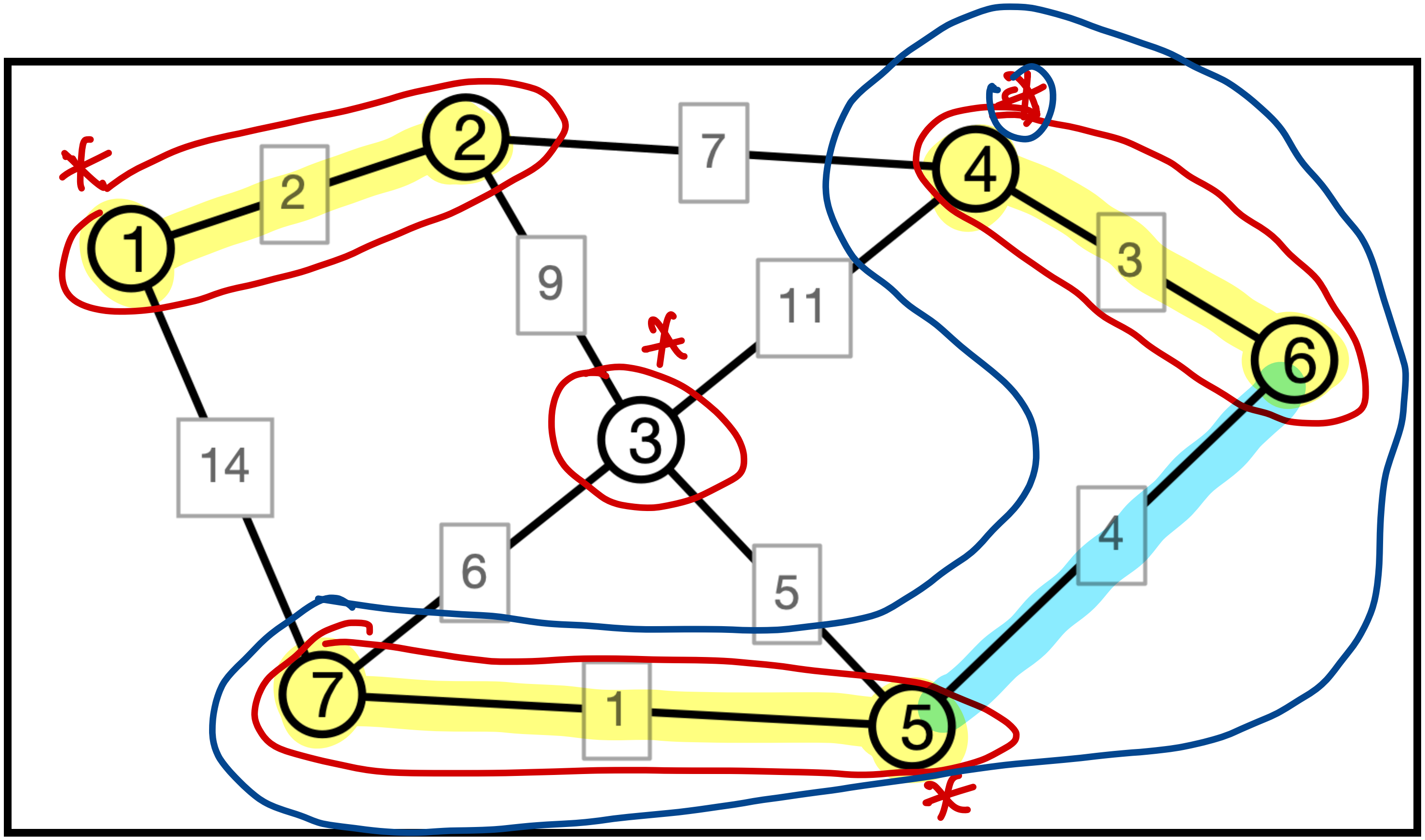
```
Kruskal(V, E, w):
  C <- collection of components
    initially, each vertex is own component
  F <- empty collection
  # iterate in order of increasing weight
  for each edge e = (u, v) in E
    if u and v are in different components then
      add (u, v) to F
      merge components containing u and v
    endif
  endfor
  return F
```

Maintaining Components

Associate a **leader** with each component

- leader is a vertex in the component
- maintain array of leaders
  - `leader[i] = v` means that `v` is leader of `i`'s component
- for each leader `v`, maintain a (linked) list of elements in `v`'s component
  - list also stores size of the component

Illustration



Leader :

1	2	3	4	5	6	7
1	1	3	4	~~5~~	4	~~5~~
				4		4

1 : 1 2

3 : 3

4 : 4 6 5 7

5 : ~~5~~ ~~7~~

Merging Components

To merge components with leaders u and v

1. Choose larger component's leader to be new leader (u)
2. Iterate over each vertex x in v's list and
    - add x to u's list
    - update `leader[x] <- u`

Running time: O(size of smaller component)

- time *per element* is $O(1)$

iterate over list, for each elt,

- single leader array update
- append elt to u's list

G has n vertices, m edges

Simplistic Analysis

```
Kruskal(V, E, w):
  C <- collection of components
    initially, each vertex is own component
  F <- empty collection
  # iterate in order of increasing weight
  for each edge e = (u, v) in E
    if u and v are in different components then
      add (u, v) to F
      merge components containing u and v
    endif
  endfor
  return F
```

m iterations

O(size of smaller comp.) $= O(n)$

$\Rightarrow O(m \cdot n)$ (Prim: $O(m \log n)$)

Fewer Merges

```
Kruskal(V, E, w):
  C <- collection of components
    initially, each vertex is own component
  F <- empty collection
  # iterate in order of increasing weight
  for each edge e = (u, v) in E
    if u and v are in different components then
      add (u, v) to F
      merge components containing u and v
    endif
  endfor
  return F
```

run n-1 times

Obs: after merge # components decreases by 1

- at start, have n components
- after get to 1 comp., no further merge

$\Longrightarrow$ only do n-1 merges! $\Rightarrow \Theta(n^2)$ r.t.

Amortized Cost of Merges

Consider the number of times each element's leader is updated

Claim. If x is relabeled k times, then x's component has size at least 2^k.

Why?

If x is relabeled, x's old comp is no larger than the component it gets merged into

$\Rightarrow$ each merge in which x is relabeled $\geq$ doubles size of x's comp.

$\Rightarrow$ k relabelings has size $\underbrace{2 \cdot 2 \cdots 2}_{k} = 2^k$

Amortized Cost of Merges

Consider the number of times each element's leader is updated

Claim. If x is relabeled k times, then x's component has size at least 2^k.

Consequence 1. If x's component has size ℓ, then x was relabeled at most $\log \ell$ times.

$$\ell \geq 2^k \iff \log \ell \geq k$$

Amortized Cost of Merges

Consider the number of times each element's leader is updated

Claim. If x is relabeled k times, then x's component has size at least 2^k.

Consequence 1. If x's component has size ℓ, then x was relabeled at most $\log \ell$ times.

Consequence 2. Running time of all merge operations in Kruskal is $O(n \log n)$

When Kruskal terminates,
all vertices in comp. of size n
$\Rightarrow$ each vtx relabeled $\leq \log n$ times
$\Rightarrow$ $O(n \log n)$ r.t. merges b/c relabel is $O(1)$ per vtx.

Conclusion

Theorem. Kruskal's algorithm can be implemented to run in time $O(m \log n)$ in graphs with n vertices and m edges.

- running time dominated by getting edges in ascending weight order

(merge ops only take n log n)

Conclusion

Theorem. Kruskal's algorithm can be implemented to run in time $O(m \log n)$ in graphs with n vertices and m edges.

- running time dominated by getting edges in ascending weight order

Remark. More efficient data structures for merging sets exist

- "Union-find" ADT, "disjoint-set forest" data structure
- time to perform merges is $O(n\alpha(n))$
  - $\alpha(n)$ is "inverse Ackerman function"
  - $\alpha(n)$ grows so slowly, it is practically constant