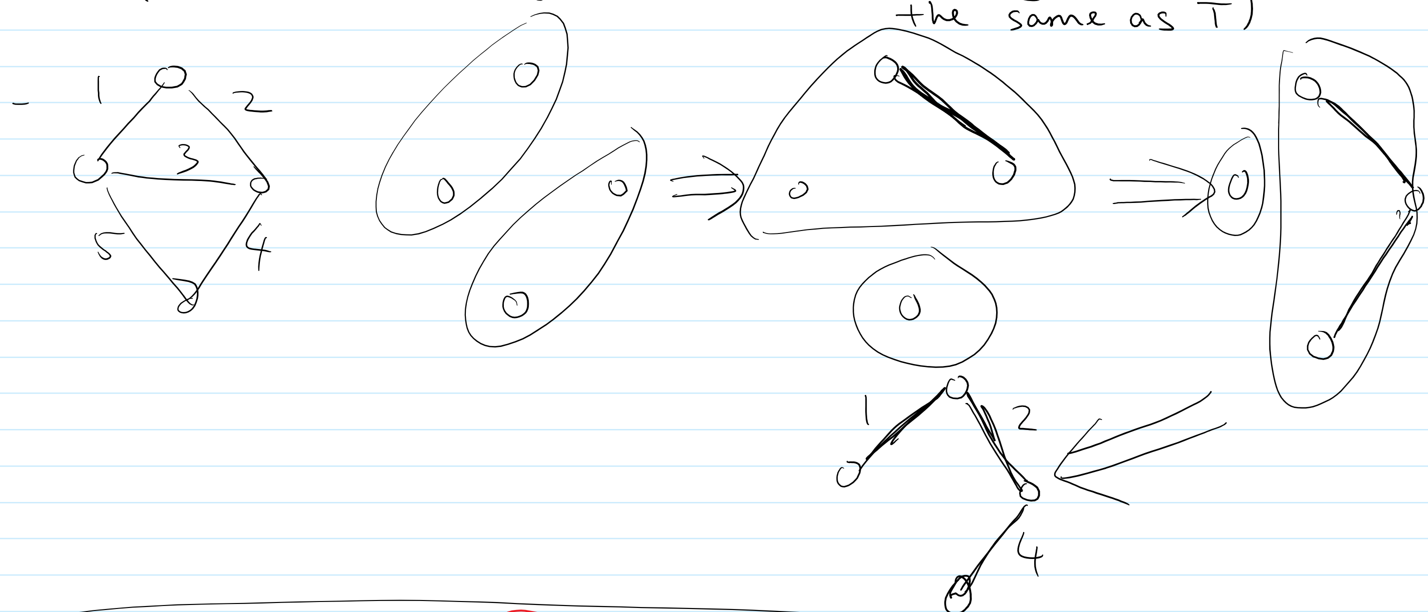
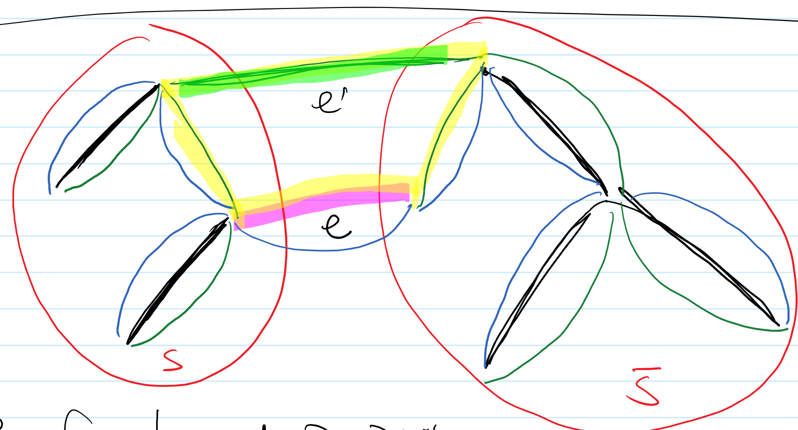
Lecture 14 Minimum Spanning Trees 2

Thursday, March 5, 2020 9:47 AM

- cycle property: for any cycle in graph G, removing any one of its longest edges does not change the length of MST for G.

- cut property: Given a subset of edges F, suppose F is a subset of an MST T. Pick any cut $(S, \bar{S})$ that does not intersect with F, let e be (any) edge with minimum cost in $(S, \bar{S})$, then $F \cup \{e\}$ is a subset of an MST T' (T' may or may not be the same as T)

- 



black: edges in F

green: MST T containing F

purple: min cost edge e

yellow: cycle formed by adding edge e.

green: e', an edge in C, $(S, \bar{S})$, T.

blue: new MST, T'

Proof of cut property:

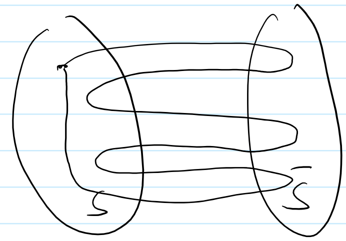
let e be min cost edge of cut $(S, \bar{S})$

case 1: e is an edge in T, this is trivial because $F \cup \{e\} \subseteq T$, can choose $T' = T$

case 2: e is not an edge in T.

adding e to T form a cycle, call it C.

every cycle that intersects with $(S, \bar{S})$ must intersect an even number of times.



there must be another edge $e' \in C$

$e' \in T$, e' also crosses the cut $(S, \bar{S})$

we will swap e and e'

define $T' = (T \setminus \{e'\}) \cup \{e\}$

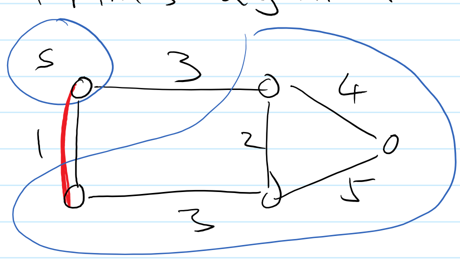
$$\mathrm{cost}(T') = \mathrm{cost}(T) \underbrace{- w(e') + w(e)}$$

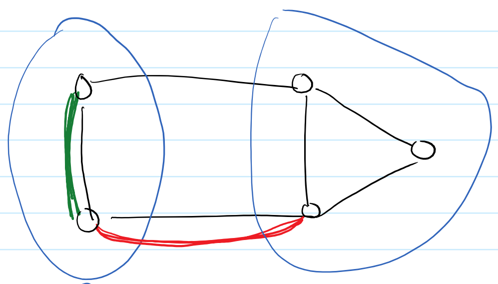
$$\leq \mathrm{cost}(T) \quad \text{by assumption } w(e) \leq w(e')$$

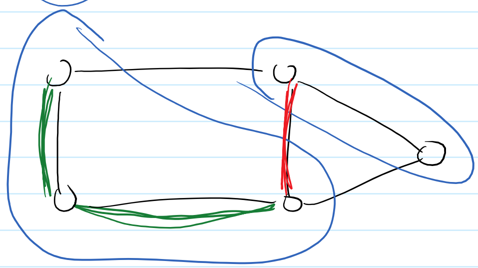
if T is an MST, then T' is also an MST.

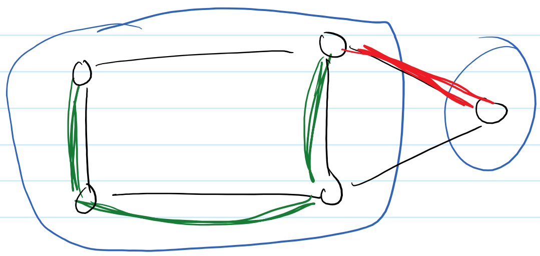
since $F \cup \{e\} \subseteq T'$, this concludes the proof. $\square$

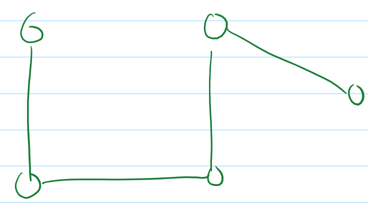
- Prim's algorithm

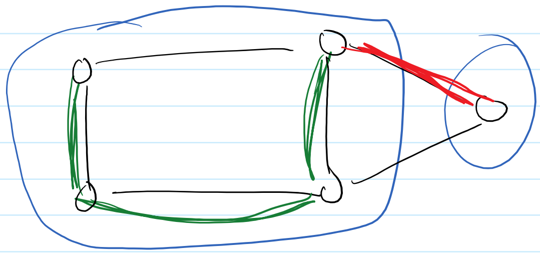


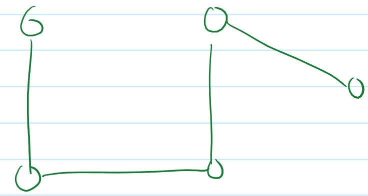












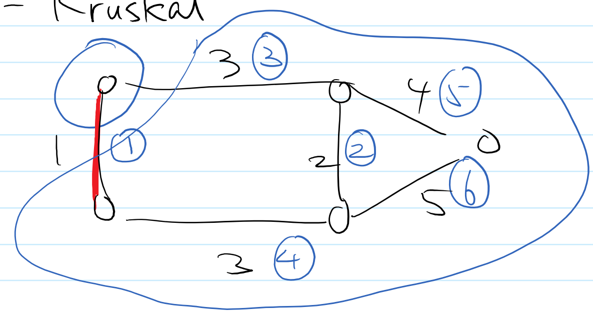
- in implementation: want to find min cost edge in the cut efficiently.

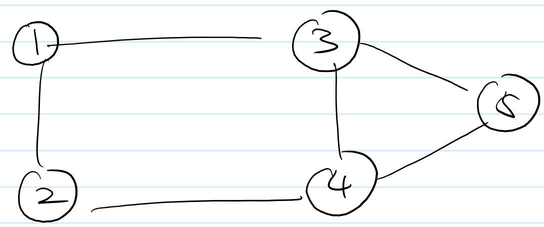
  maintain an array dis[u]

  dis[u]: minimum cost of an edge (u,v) where v is already connected to S.

  running time: $O(m + n\log n)$ (use Fibonacci heap)

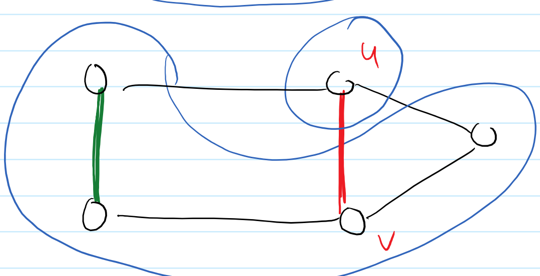
- Kruskal





union (1,2)

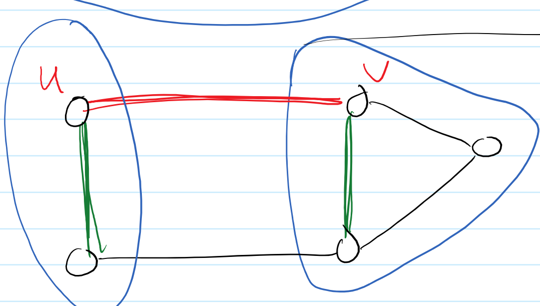
{1,2} {3} {4} {5}



find(3) ≠ find(4)

union (3,4)

{1,2} {3,4} {5}

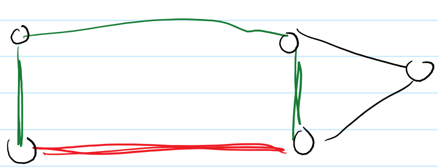


find(1) ≠ find(3)

union (1,3)

{1,2,3,4} {5}

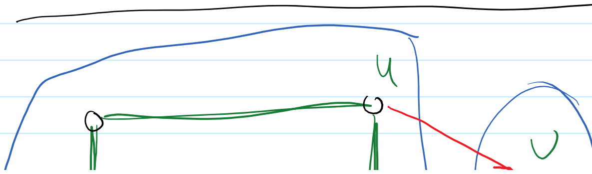
find(2) = find(4)

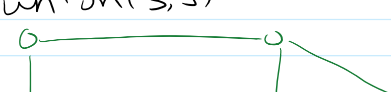


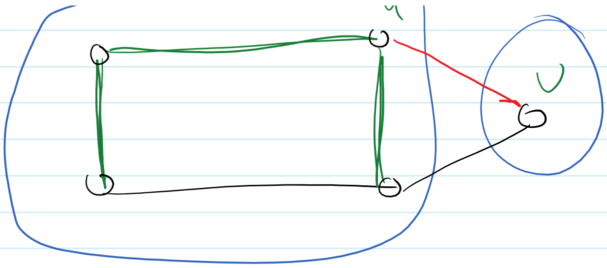
adding this edge creates a cycle, Kruskal will not add this edge.

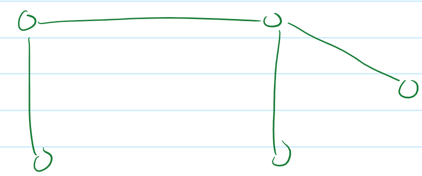
find(3) ≠ find(5)

union(3,5)









$\{1, 2, 3, 4, 5\}$

- Implementing Kruskal
  - union-find data structure.

    maintain disjoint sets of $\{1, 2, \cdots, n\}$

    initially, every element is in a separate set

    $\{1\}, \{2\}, \{3\}, \ldots, \{n\}$

    (corresponds to the case that none of the vertices are connected)

  - two operations
    1. union: merges two sets
    2. find: for every element u, find(u) identifies the set that u belongs to.

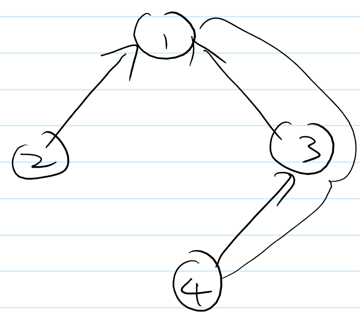
       if u, v are in the same set find(u) = find(v)

       u, v are in different sets find(u) $\neq$ find(v)

- one implementation of union-find.
  - idea: use a tree structure
    - every tree $\Longleftrightarrow$ set
    - every vertex maintains a pointer to its parent

    1 2 3 4 5

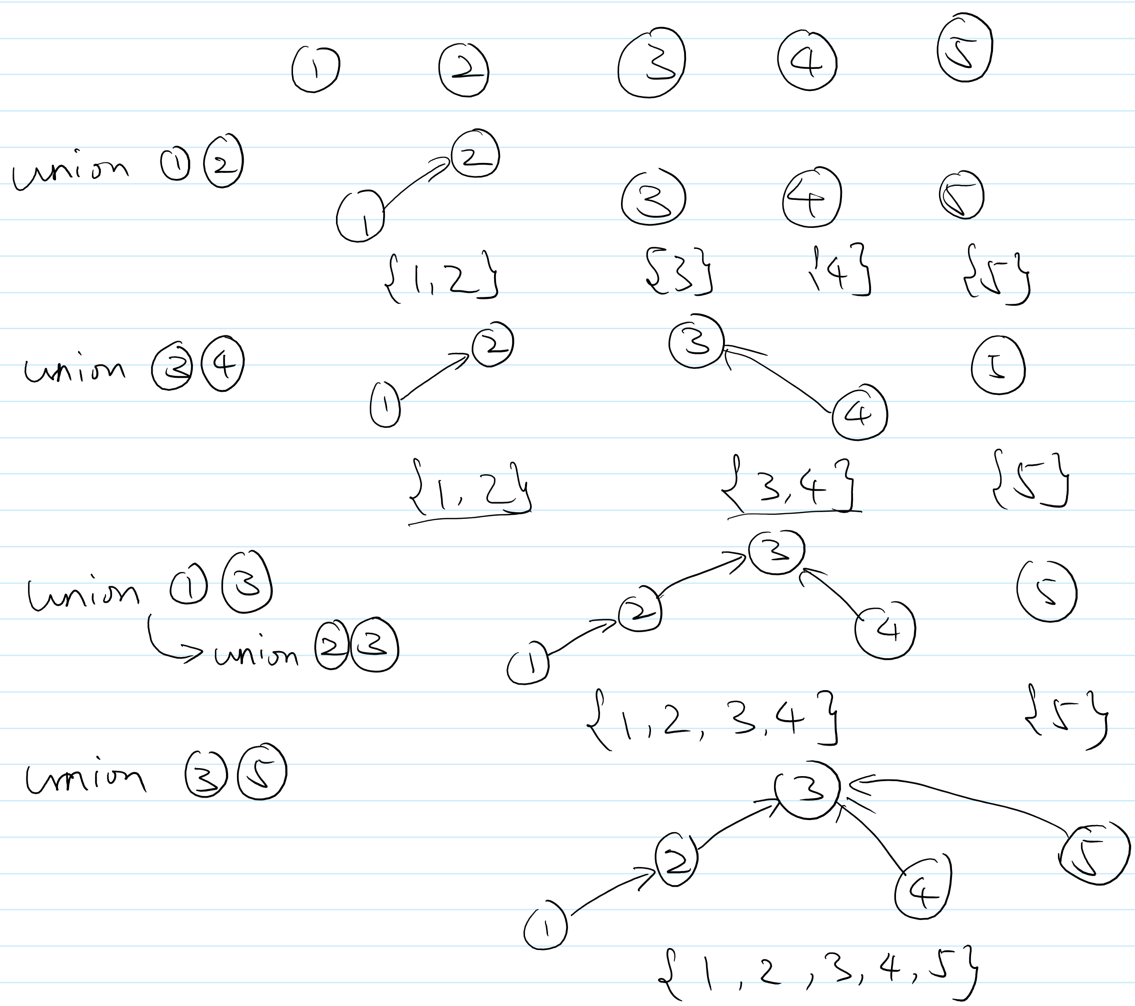
  - find: finds the root of tree



find(4) = 1 find(2) = 1

find(4) = 1   find(2) = 1

- union: first find the two roots, point one of them to the other



Claim: always link shallower tree to deeper tree, depth of the tree is at most $O(\log n)$.

runtime: find : proportional to depth $O(\log n)$

union: two find operations + linking $O(1)$

$O(\log n)$

this implementation: can check whether adding (u,v) creates a cycle in $O(\log n)$ time

$\Rightarrow$ Kruskal runs in $O(m \log n)$.