CS314 – Minimum Spanning Trees

Note Title 10/2/2013

Announcements

- HW due next Tuesday (oral grading)

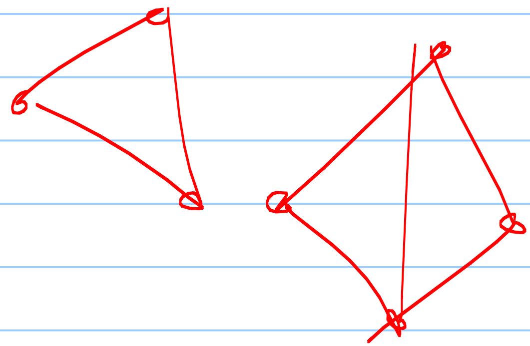
Dfns:

A tree is a maximal acyclic graph, always with n−1 edges.

any edge is cut edge

↪ BFS / DFS tree

A component of a graph is a maximal connected subset of G.



Setting: a weighted graph

- A graph $G = (V, E)$ together with a function $w: E \to \mathbb{R}$ that gives a weight $w(e)$ to each edge $e \in E$

Goal: Find minimum set of edges which connects everything.

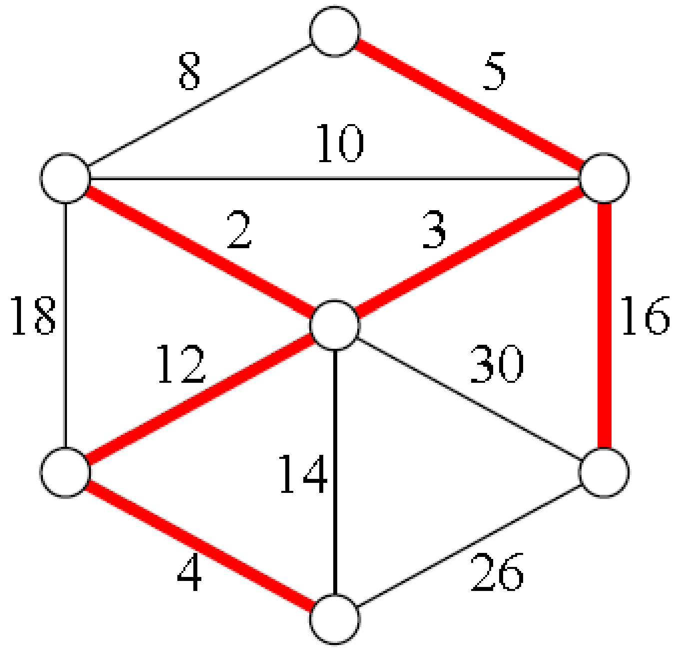
tree

(Obvious applications!)

MST

Note:

We'll assume edge weights are unique, so $w(e) \neq w(e')$ for any $e, e' \in E$.

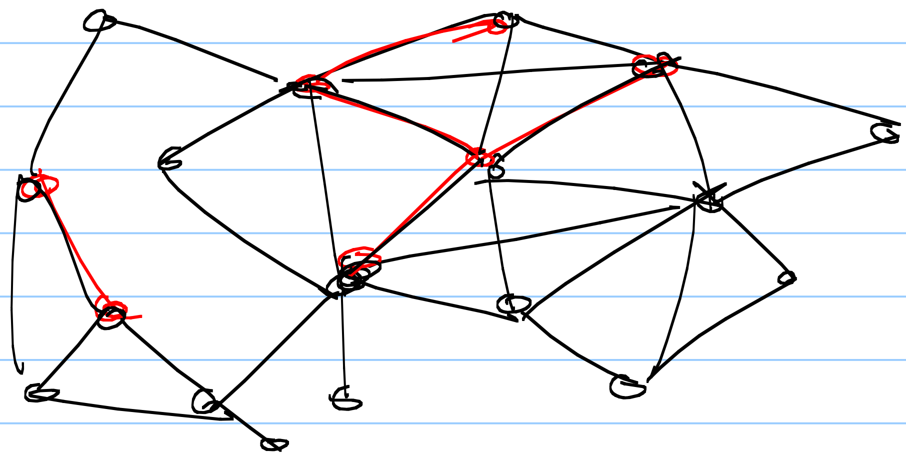


Greedy algorithms

Strategy

We'll try to iteratively build the MST.

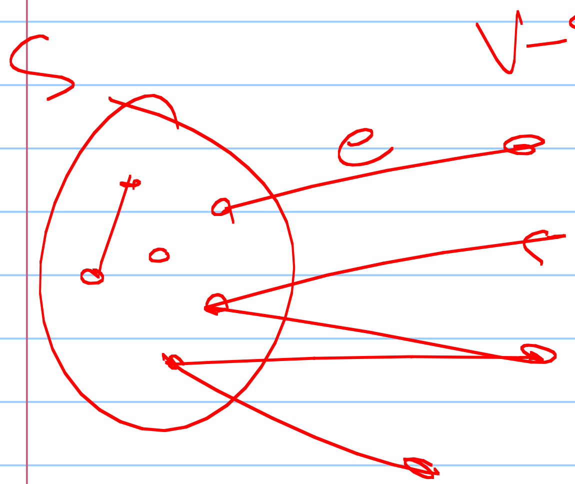
At each stage, some subgraph of the MST will exist.
(called a spanning forest)



Key question:

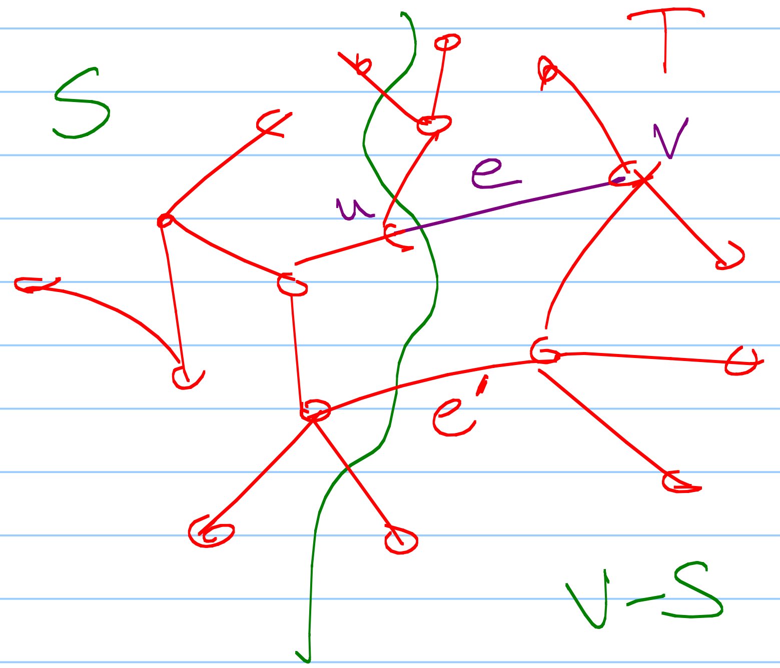
which edge should I look at adding to F next?

Key Lemma: Let S be any subset of V (besides $\emptyset$ or V itself). Let e be the edge of minimum weight with one endpoint in S and one in V−S. Then e is in any MST of G.



proof: Consider a tree T which doesn't contain e.

We need to show T is not minimum.



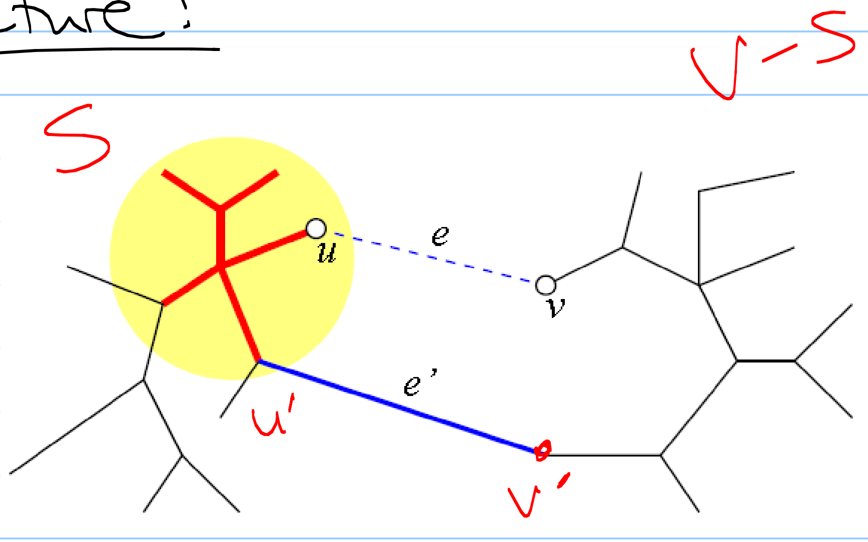
Let $e = \{u, v\}$

T is connected, so it contains a u to v path.

Take "first" edge going from S to $V-S$ along this path $\Rightarrow e'$.

Take $T - e' + e$

Picture:



Any path in T that used edge e' will now use: path u' → u + e + v → v'

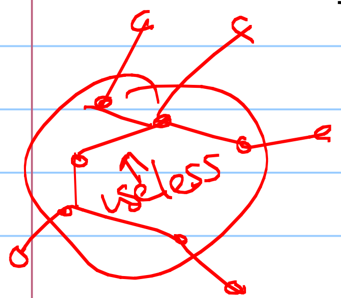
So T−e'+e is still connected, has n−1 edges, so is a tree. & T−e'+e has weight < T

⇒ T was not MST. □

A bit further: Suppose we have a spanning forest, F.

A safe edge for a component is the minimum weight edge with only one endpoint in that component.

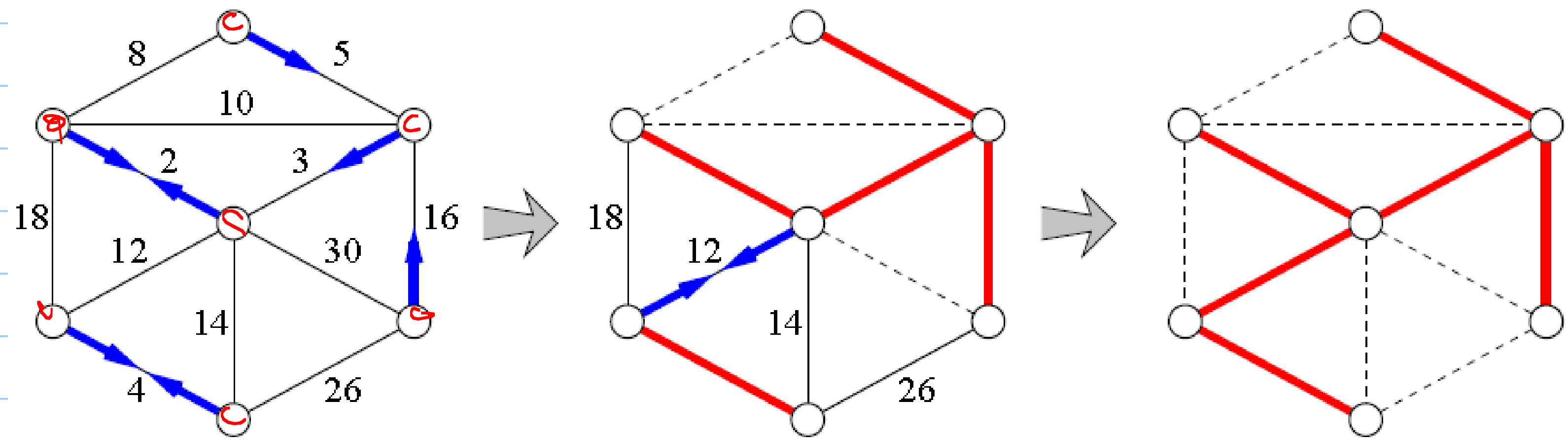
A useless edge is an edge not in F but with both endpoints in the same component.



Note: Our lemma says safe edges should be in the MST!

So an algorithm:

Add all the safe edges.
Recurse.



This is Borůvka's algorithm, from 1926.

(Also others - often called Sollin's algorithm.)

Pseudocode - first, find components:

(see last lecture)

```
TRAVERSEALL(s):
    for all vertices v
        if v is unmarked
            TRAVERSE(v)
```

```
TRAVERSE(s):
    put s in bag
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            for each edge vw
                put w into the bag
```

Now:

```
BORŮVKA(V, E):
  F = (V, ∅)
  TRAVERSEALL(F)   ⟨⟨count and label components⟩⟩
  while F has more than one component
      ADDALLSAFEEDGES(V, E)
      TRAVERSEALL(F)
  return F
```

```
ADDALLSAFEEDGES(V, E):
  for i ← 1 to V
      S[i] ← NULL   ⟨⟨sentinel: w(NULL) := ∞⟩⟩
  for each edge uv ∈ E
      if label(u) ≠ label(v)
          if w(uv) < w(S[label(u)])
              S[label(u)] ← uv
          if w(uv) < w(S[label(v)])
              S[label(v)] ← uv
  for i ← 1 to V
      if S[i] ≠ NULL
          add S[i] to F
```

$O(n+m)$

$= O(m)$

Runtime:

At each stage, # of components goes down by at least $\frac{1}{2}$.

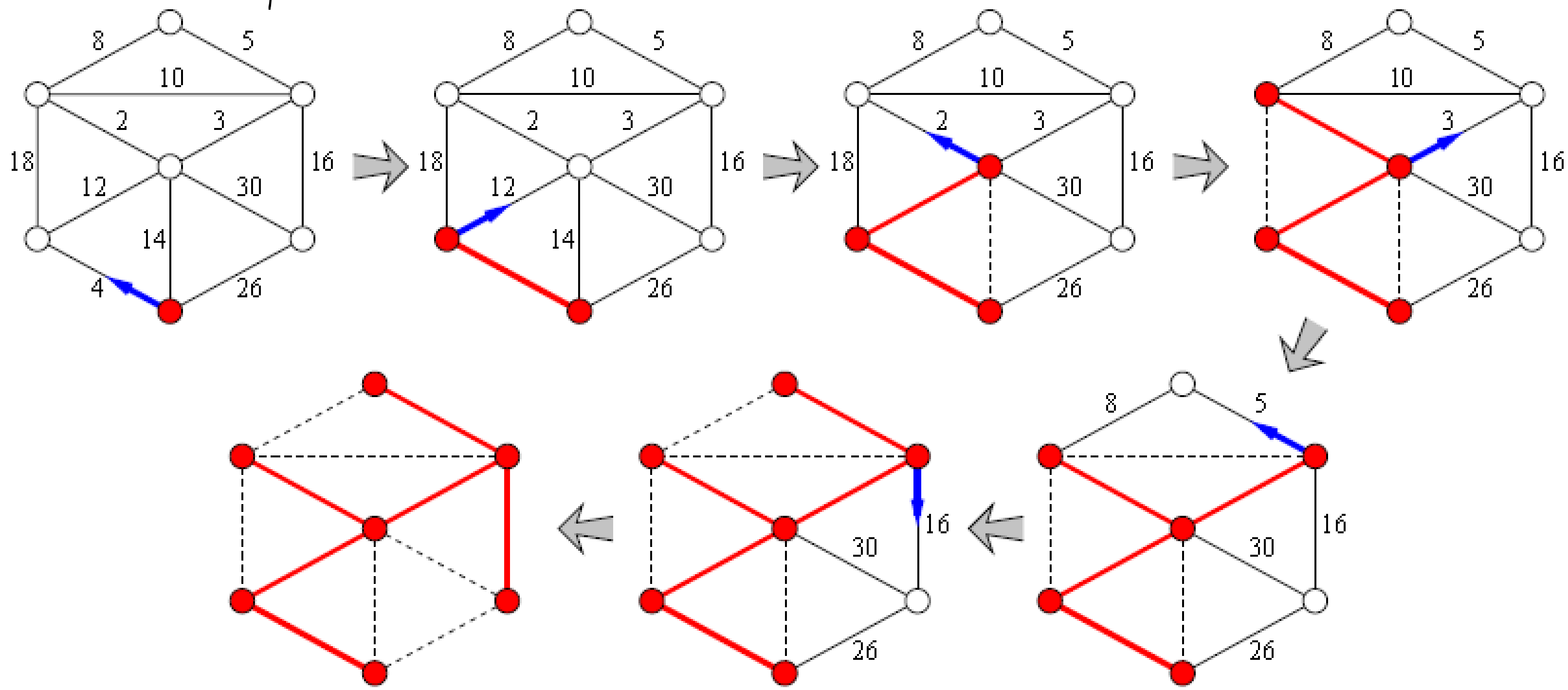
$$T(n) = T(\tfrac{n}{2}) + O(m)$$

$$\Rightarrow O(m \lg n)$$

$O(E \lg V)$ (in notes)

Prim's algorithm: add a safe edge, one at a time

(really Jarník's from 1929)



Code: Actually, similar to DFS, but
keep edges in a heap.

Take min edge, & check if endpts
are both marked.
If not, add to T.

Runtime: $O(m \lg n)$

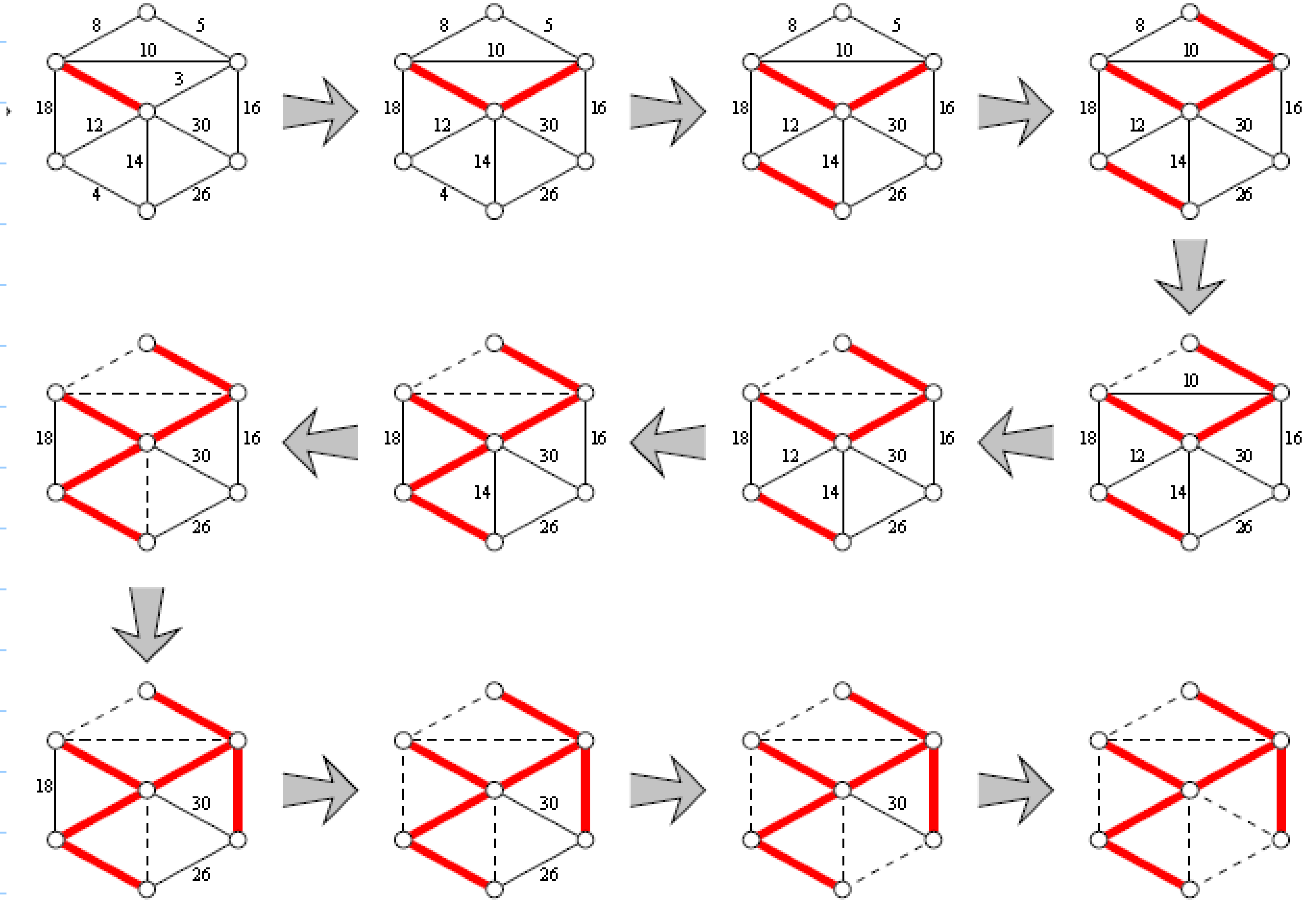
(Can improve with fancy data structures...)

Kruskal's algorithm (1953)

Idea: Scan edges in increasing order.
If edge is safe, add it to F.

Since we'll go in sorted order, at each stage the smallest safe edge gets added, so results in MST.

Picture:



8 5 10 3 18 12 30 16 14 4 26
8 5 10 18 12 30 16 14 4 26
8 5 10 18 12 30 16 14 26
8 10 18 12 30 16 14 26
10 18 12 30 16 14 26
18 12 30 16 14 26
18 30 16 14 26
18 30 16 26
18 30 26
30 26
30

How to implement?

Need a data structure to maintain a forest.

Allow:

- lookups: is this edge useless?
- unions (to join 2 components)

Next time — the details!

Union-Find data structure