Algorithms

Graphs:
Reductions
MST intro

Recap

- HW5 up, due (on paper) Wed. after break
- Midterms back on Friday
- Reading due before class Fri, + after break

Graph Searching: post-reading recap

All variants of this:

```
WhateverFirstSearch(s):
    put s into the 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
```

≤ once per vertex $= V \cdot T$

$T \cdot d(v) + 1$

Runtime: "bag": a data structure

need to add & remove: $O(T)$

Stacks & queues: $O(1)$

total:

$$\sum_v \left(T d(v) + 1 \right) = V + E \cdot T$$

$+ VT$ (for outer loop)

```
WHATEVERFIRSTSEARCH(s):
    put s into the 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
```

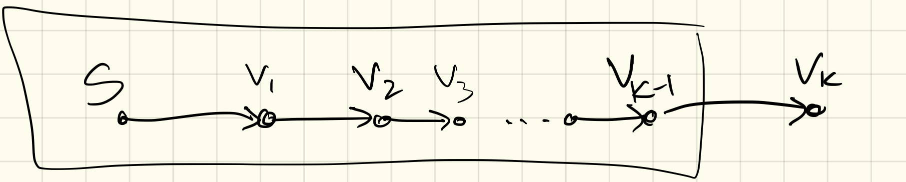
Correctness:

Need to show it marks all vertices reachable from s, + no others.

Proof: induction!

BC • s is marked

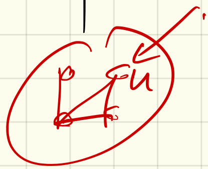
IS • Assume vertices at distance (# edges) k-1 are marked + show all at distance k will also be marked

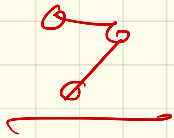


To get connected components, need one more thing:

Make sure you actually get every vertex!

(He shows a couple of ways.)





Other notes:

"Best-first" search:

Wait for next chapters – these are a bit more subtle, so we'll spend more time later.

Dfn: Reduction

A reduction is a method of solving a problem by transforming it to another problem.

We'll see a ton of these!

(Especially common in graphs...)

Key: - What graph to build

- What algorithm to use

First example:

Given a pixel map, the flood-fill operation lets you select a pixel & change the color of it & all the pixels in its region.

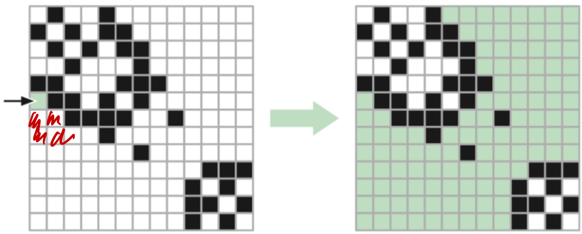
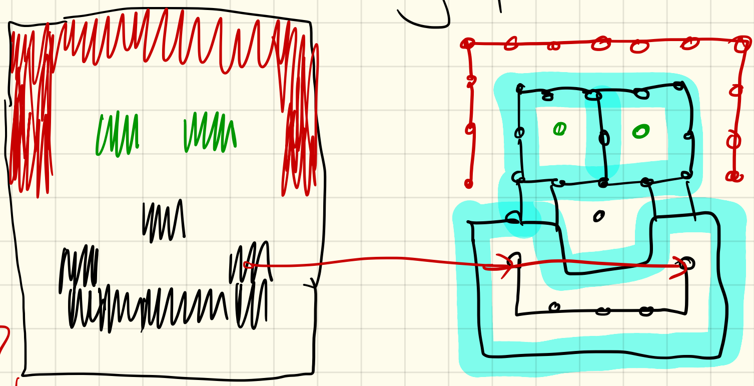


Figure 5.13. An example of flood fill

How?

So: Build a graph from pixels:



input: $n \times n$ pixel grid

Graph with $V = n^2$, $E = \Theta(n^2)$

Set up graph w/ adjacencies based on input colors.

Then, our algorithm:

if pixel p is selected
WFS(p)
↳ augment to change color when marked

Correctness: WFS reaches every connected node, & I built the graph so same color regions only are connected

runtime: $O(V+E) = O(n^2)$

Next: Minimum Spanning Trees!

Goal: Given an edge weighted graph G, w, find a spanning tree of G that minimizes:

$$w(T) = \sum_{e \in T} w(e)$$

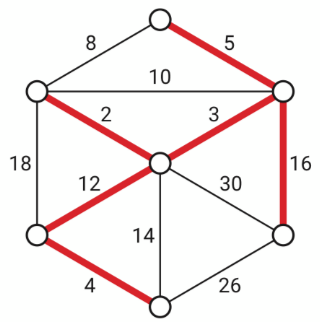


Figure 7.1. A weighted graph and its minimum spanning tree.

Motivation: Everywhere

First:

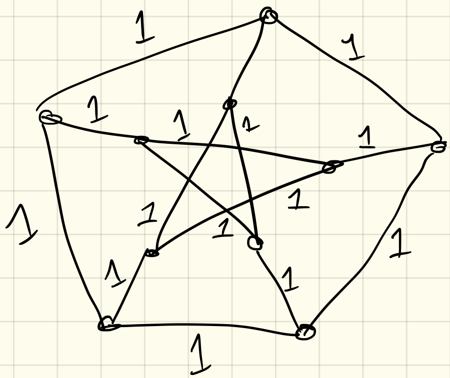
Does it have to be a tree?

min way to connect every vertex:
if cycle, remove an edge (assuming positive weights)

Second:

These are "obviously" not unique!

Ex:



tree? any spanning tree has weight $n-1$

Things will be cleaner if we have unique trees. So:

<u>Lemma</u>: Assuming all edge weights are distinct, then MST is unique.

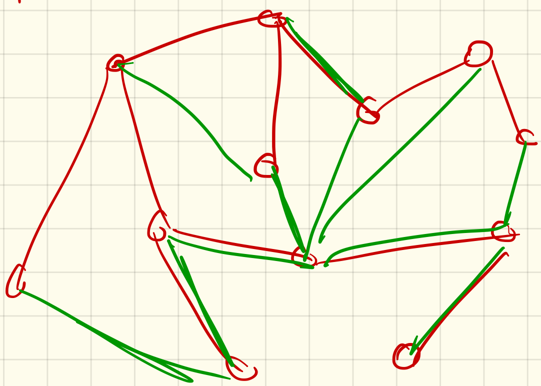
<u>Pf</u>: By contradiction:

Suppose T & T' are both MSTs, with $T \neq T'$.

We'll show two edges have the same weight:

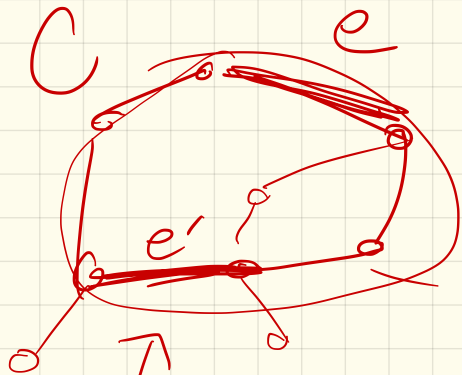
$T+T'$ has a cycle

T T'



Pick one call it C.

$T + T'$ → pick a cycle in it



C has ≥ 1 edge from T (& not T') & one from T', but not T

e

e'

Consider $T - e + e'$: this is a spanning tree, so $w(e') > w(e)$ (since T chose e)

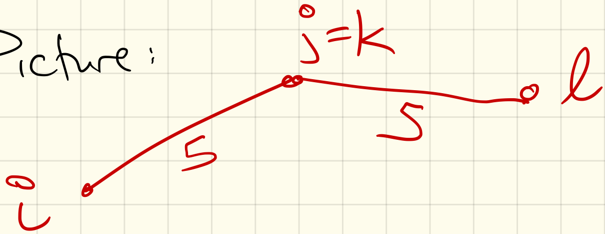
Repeat for $T' - e' + e$:
$\Rightarrow w(e) > w(e')$ ↯.

Now, what if weights aren't unique?

Just need a way to consistently break ties.

```
ShorterEdge(i, j, k, l)
  if w(i, j) < w(k, l)        then return (i, j)
  if w(i, j) > w(k, l)        then return (k, l)
  if min(i, j) < min(k, l)    then return (i, j)
  if min(i, j) > min(k, l)    then return (k, l)
  if max(i, j) < max(k, l)    then return (i, j)
  ⟨⟨if max(i,j) > max(k,l)⟩⟩  return (k, l)
```

Picture:



So, takeaway:

Can assume unique MST.

Next: an algorithm.

The magic truth of MSTs:

You can be SUPER greedy.

Almost any natural idea will work!

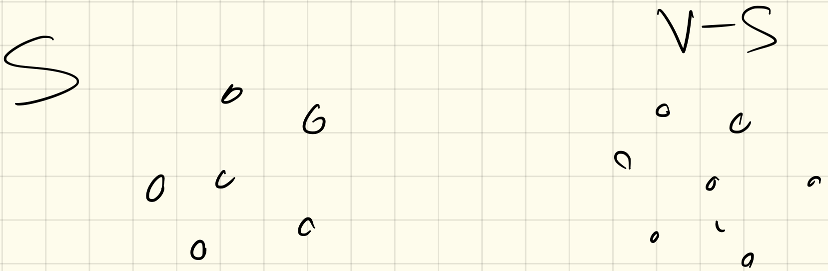
This is highly unusual, & there's a reason for it:

these are a (rare) example of something called a matroid.

(Way beyond this class...)

Key property:

Consider breaking G into two sets: S and V/S



The MST will always contain the lowest edge connecting the two sides.

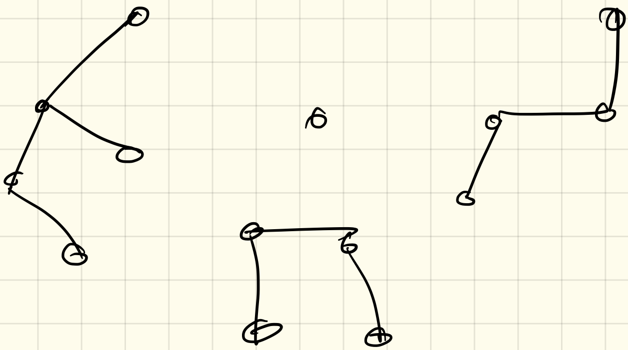
Generic Algorithm:

Build a <u>forest</u>: an acyclic subgraph

<u>Dfn</u>: An edge is <u>useless</u> if it connects 2 endpts in same component of F

An edge is <u>safe</u> if it is minimum edge from some component of F to another.

F:



So idea:

Add safe edges until you get a tree

If everything isn't connected, must have some safe edge.

Why?

Add it & recurse.

We'll see 3 ways:

1. Find <u>all</u> safe edges. Add them + recurse.

2. Keep a single connected component.
   At each iteration, add 1 safe edge.

3. Sort edges + loop through them.
   If edge is safe, add it.