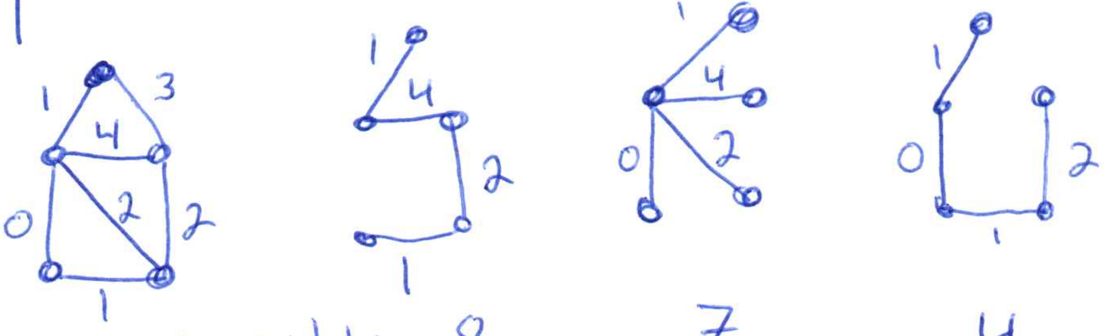
6-14

MST

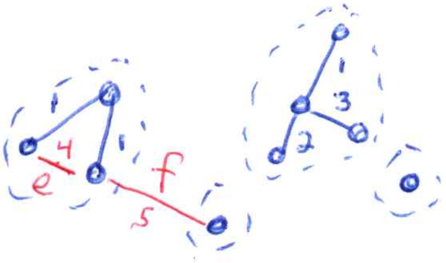
Eg



weight: 8 7 4

Kruskal: • order edges by weight
• if e does not make a cycle, pick it

General Situation:



• e makes a cycle
=> don't use it

• f does not
=> add f to T

Correctness - Exchange proof

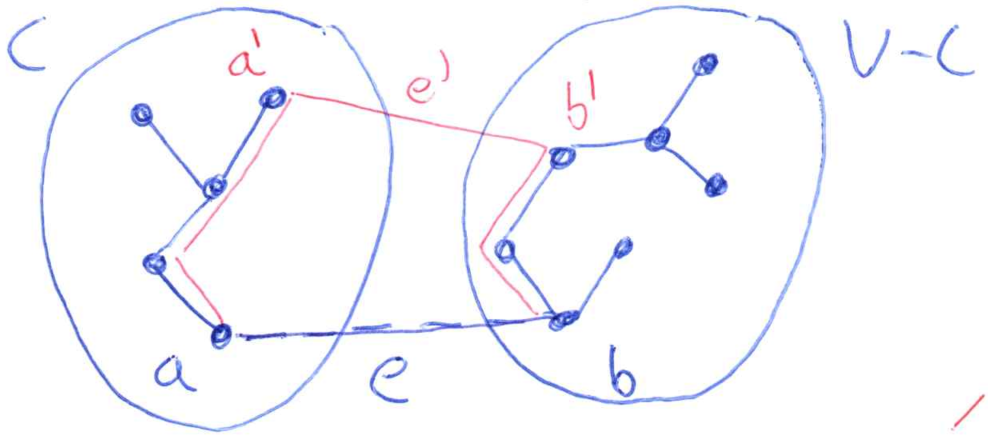
Base case: $i = 0$ - Trivially true

Induction Hypothesis: Assume there is a MST M matching T on the first $i-1$ edges.

$$\begin{array}{llllll} \text{Kruskal's } T & t_1 & \ldots & t_{i-1} & t_i & \ldots t_{n-1} \\ & \shortparallel & & \shortparallel & \not\shortparallel & \\ \text{MST } M & m_1 & \ldots & m_{i-1} & m_i & \ldots m_{n-1} \end{array}$$

Let $t_i = e = (a, b)$ and let C be the connected component of T containing a

Look at a path in M from a to b



possibly multiple times

It must cross from C to V-C

- say at $e' = (a', b')$

* Then $w(e) \leq w(e')$ since Kruskal's always picks the edge of least weight

Exchange: Let $M' = (M - \{e'\}) \cup \{e\}$

Claim: M' is a MST

- also now matches T on i edges

- if e' is later in ordering swap to m_i

① M' connects all vertices:

e' is replaced by $a' \rightarrow a, e, b \rightarrow b'$

② M' has minimal weight

same # of edges & connected

$$w(M') = w(M) - w(e') + w(e)$$
$$\leq w(M) \text{ by } *$$

so M' is a MST

- note: any connected graph: n vertices, n-1 edges => is a tree

Implementation: Union-Find

Find(v) - return set/component label of v

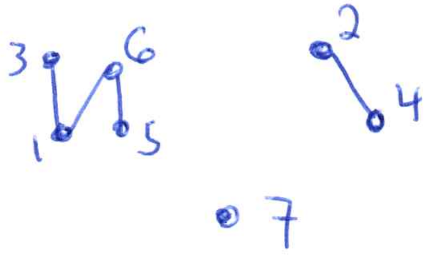
Union(X, Y) - merge sets/components X and Y

- both should have same label

Simple Implementation:

- Array S[1...n], S[i] = component label of item (vertex) i
- Linked List of items in each set

vertices:	1	2	3	4	5	6	7
eg S:	1	2	1	2	1	1	3



LL: C_1: 1, 3, 5, 6

C_2: 2, 4

C_3: 7

Find: O(1) ~ array lookup

Union: - merge 2 LL (order doesn't matter)

- must renumber one of the 2 sets in S
  - use LL to find each index in S

Always renumber the smaller set!!!

eg Union(C_1, C_2): - renumber C_2: $C_1 \leftarrow C_1 \cup C_2$

- update S[2] = 1, S[4] = 1

* New set is always at least double the size of the smaller set

6-17

Max size of a set to renumber?

$O(n) \approx$ # vertices

How many times can you double the size of a set before it is size n?

1 2 4 8 ... n $\Rightarrow \leq O(\log n)$ times

Total Union work: $O(n \log n)$

Kruskal's runtime:

$$\underbrace{O(m \log n)}_{\text{sort}} + \underbrace{O(m)}_{\text{finds}} + \underbrace{O(n \log n)}_{\text{unions}}$$

$\Rightarrow O(m \log n)$ assuming G is connected

i.e., $m \geq n-1$

Aside: sort $O(m \log m)$ assume G is connected

$m \geq n-1$

- upper bound on $m \in O(n^2)$ $m \leq n^2$
- replace in log: $O(m \log n^2) \in O(m \log n)$

Why not do the same for first m?

Aside: fancier implementations for Union-Find

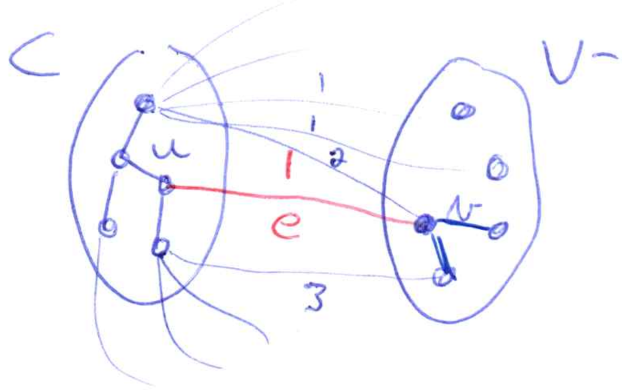
CS 466 · analysis not fun

log* & Ackerman function

Prim's Alg

- grow one connected component C in a greedy way
- add a vertex $v \in V-C$

such that $e=(u,v)$ is minimum weight, $u \in C$



- e connects C and V-C

```
$C \leftarrow \{s\}$
$T \leftarrow \emptyset$
while $C \neq V$ do
    find vertex $v \in V-C$ such that
        there exists a $u \in C$ with $e=(u,v)$
        and $e$ is minimum weight
    $C \leftarrow C \cup \{v\}$
    $T \leftarrow T \cup \{e\}$
```

Implementation

Need to find a vertex in $V-C$ connected to a minimum weight edge leaving C.

For $v \in V-C$, define

$$(*) \quad \text{weight}(v) = \begin{cases} \infty & \text{if no edge } (u,v) \text{ with } u \in C \\ \min\{w(e) \mid e = (u,v) \in E \text{ and } u \in C\} & \text{otherwise} \end{cases}$$

Priority Queue (heap)

- Maintain a set $V-C$ as an array in heap order, according to weight (*)
- ExtractMin(): remove & return vertex with minimal weight
- Insert(v, weight(v)): insert vertex v with weight(v)
- Delete(v): delete vertex v

Implementation: $O(\log k)$ per operation

- $k = |V-C|$

Implementation is tricky!

Delete(v) in $O(\log k)$?

- If we need to search heap for $v \to O(n)$

Create array $\bar{C}[1..n]$

$$\bar{C}[v] = \begin{cases} -1 & \text{if } v \notin V-C \\ \text{"location" of } v \text{ in heap} & \text{otherwise} \end{cases}$$

* likely a pointer - we don't want to keep updating locations/index

Analysis

- 1 ExtractMin to add each v to C
- Scan v's adj list to find $e=(u,v)$ with $w(e) = \text{weight}(v)$ to add to MST
- Need to update/reduce weight of vertices v' such that (v, v') with $v' \in V-C$
  - because v is now in C

Size of heap: $O(n)$

- $n-1$ ExtractMin ops
- $O(m)$ reduce weight ops
  - Delete & Insert ops ← Exercise

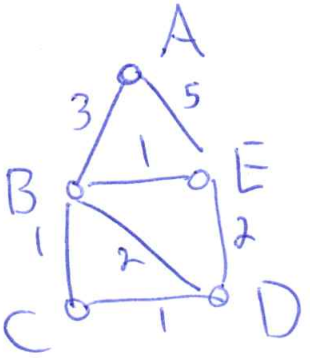
Total cost: $O(m \log n)$

6-21

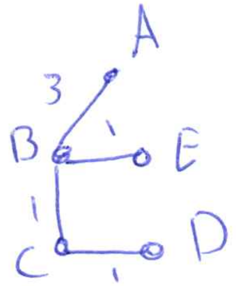
Shortest Paths in Edge Weighted Graphs

- BFS ~ shortest paths from a v in unweighted undirected graphs

General Input: weights on edges



Does MST always contain shortest path? No



eg A->D: ABD weight 5
or ABCD 5

A->E ABE 4

E->D? use edge (E,D) = 2
MST: EBCD = 3

Many Shortest Path Algs.

Dijkstra's Alg (1959)

Input: graph or digraph $G=(V,E)$

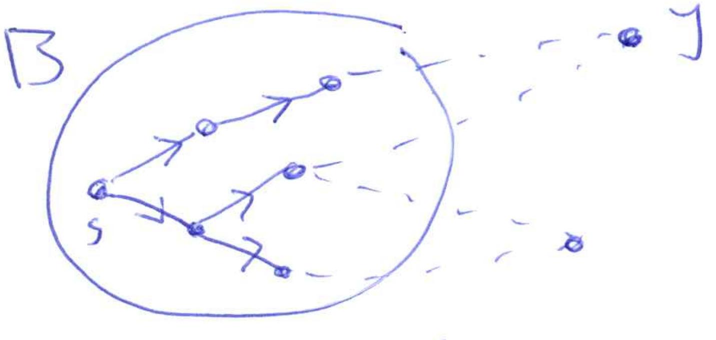
$w: E \to \mathbb{R}^{\geq 0}$, $s \in V$

Output: shortest path from s to every other vertex v.

Idea: grow tree of shortest paths starting from s

General Step: We have a tree of shortest paths to all vertices in set B.

Initially $B = \{s\}$



Choose edge (x,y), $x \in B$, $y \notin B$

to minimize $d(s,x) + w(x,y)$

where $d(s,x)$ is the (known) minimum distance from s to x.

Call this minimum d.

6-23

$d(s,y) \leftarrow d$

add (x,y) to tree // parent$(y) \leftarrow x$

Greedy: always add vertex with next minimum distance from s
(in a sense)

Claim: d is the minimum distance from s to y
- when added to B
- also justifies output is a tree

Proof: Any path π from s to y has
- $\pi_1 \sim$ initial path in B // $s \to u$
- $e = (u,v) \sim$ first edge leaving B
- $\pi_2 \sim$ rest of path

Note: path to y is either

we choose shortest of this form

or

weight from v to $y \geq 0$

$w(\pi) \geq w(\pi_1) + w(u,v)$ ~ remove path v to y

$\geq d(s,u) + w(u,v)$ • we choose min one of this form

$\geq d$
- $w(\pi_2) \geq 0$
- breaks if negative weight cycles

$\Rightarrow$ By induction on $|B|$ the alg correctly finds $d(s,v)$ for all v

we always pick shortest path not in B
- its always in form

6-24

Implementation

- use PQ similar to Prim
- array to store "tentative" distance to v

d' for all $v \notin B$ - if in B its the shortest path

Updating path lengths when y is chosen

- go through y's adj list
  - for each z in y's adj list
    - check current path length $d(z)$ with new $d(y) + w(y,z)$

If shorter, update - delete from PQ / re-insert to PQ

to maintain heap order

* extra array to find z in heap in $O(1)$ - *like Prim

Runtime: assuming G is connected

$$O(n \log n) + O(m \log n) \in O(m \log n)$$

find min
- pick each vertex once

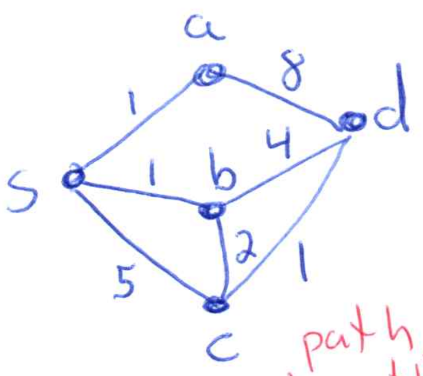
adjust heap
- all adj lists traversed

Fancier heap: Fibonacci heap

$$O(n \log n + m)$$

Dijkstra's Example :

Choose shortest path
$d(u)$ - path s to $u \in B$
$+ w(u,v)$ - weight edge
$u \in B, v \notin B$



Adj List:
S: a b c
⋮

	s	a	b	c	d
path length - d	0	∞	∞	∞	∞
parent	∅				

start with s and go through adj list
- update any shorter paths found

d	0	1	1	5	∞
parent	∅	s	s	s	

PQ: (a,1) (b,1) (c,5)

Pick: (a,1) go through Adj list, update paths

d	0	1	1	5	9
parent	∅	s	s	s	a

PQ: (b,1) (c,5) (d,9)

Pick (b,1)

d	0	1	1	3	5
parent	∅	s	s	b	b

PQ: (c,3) (d,5)

(c,3):

d	0	1	1	3	4
parent	∅	s	s	b	c

PQ: (d,5)
done

---

update path length • PQ.delete - extra arra
to find in
PQ.insert heapinO(1

* implementation similar to Prim's

Single Source 6-25

Dijkstra's Alg O(m logn)
- no negative weights

No cycles O(n+m)
- uses topological sort

Bellman-Ford O(nm)
- general weights, no negative cycle

<u>All pairs</u>: Floyd-Warshall

Topological Sort - use DFS on directed graphs

Shortest path alg: compute shortest paths starting with v_1 to see where it gets to.

Then check next vertices
- update if new shorter path found

O(n+m)