CS3000
6/5 - thurs !!

Admin

- HW4 due 9pm
- HW5 out, due 6/12 9pm
- Fun algo today
- exam #2 6/12
- second chance HW due 6/18

Agenda

1. Minimum Spanning Trees
2. Kruskal
3. Prim

1. Minimum Spanning Trees

Last time... DFS
- find all reachable vertices
- (may not be shortest paths)
- original Graph ⇝ DFS Tree (only one path from u ⇝ v)

↪ algo DFS
v.π

Good for:
- identifying cycles (back edge?)
- topologicize sort (dependency graph)

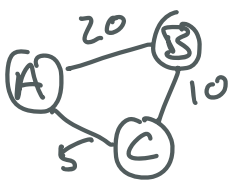
v.color
v.f (finish time)

today focus: graph → tree

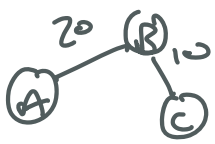
↳ weighted, undirected

goal: valid soln is a spanning tree
- collection of edges
- which form a tree
- includes all vertices from G

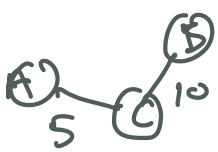
(ex) original graph G



⇝ spanning tree ⇝



tree? ✓
all vertices? ✓



tree? ✓
all verts? ✓

Both are valid solns

weight (top) = 20 + 10 = 30

weight (bottom) = 5 + 10 = 15 : better valid solution!

MST optimization problem

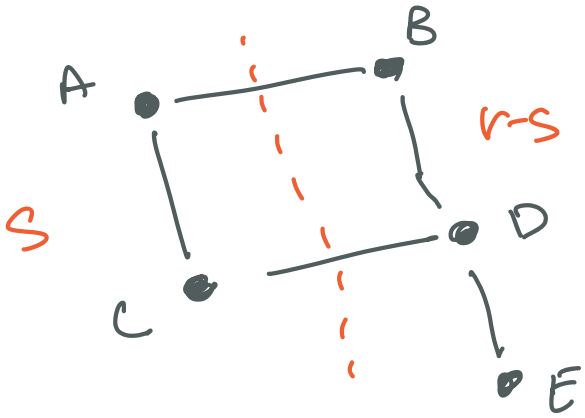
Algo structure today:

params - G, maybe s, weight function w

w(u,v) - weight of edge u,v

Definitions:

- MST is a set of edges
- set A ⊆ MST (while we're building MST)
- safe edge: can be added to A, and A ⊆ MST
- a cut [S, V-S] : partition of vertices V



V = {A, B, C, D, E}

S = {A, C}

V-S = {B, D, E}

cut!

- an edge (u,v) with u ∈ S, v ∈ V-S
  "crosses the cut" ⇝ 2 bars (C,D) and (A,B)
  (only need one in MST!)

- a [light edge] is edge with min weight crossing the cut → that's the one to keep!!

MSTs algos are greedy - choose light edge

2. Kruskal + Prim → Both greedy, create MST

Kruskal

- uses sets
- looks at each edge, one at a time
- sort edges by weight
- add light edge to A, done w/ edge

Prim

- uses heap
- looks at each vertex, explores its neighbors
- tracks v.π, v.key (edge, weight of edge)
- updates as we go

G, w

MAKE-SET(v) {v}

FIND-SET(v) what set is v in?

A ∪ {v} puts v in A

UNION(u,v) merges subtrees

G, w, r

INSERT(H, v) insert, reheapify

EXTRACT-MIN(H) remove root, reheapify

DECREASE-KEY(H, v, w) update v.key to be w

Qs to answer:

- run-time?
- value or actual soln?
- Kruskal: in A? what are sets?
- Prim: what are v.π, v.key?

KRUSKAL(G, w)

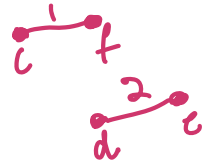
1 $A = \{\}$ — A ⊆ MST
2 **for** each vertex $v \in G.V$
3 MAKE-SET(v)
4 create a single list of the edges in $G.E$
5 sort the list of edges into monotonically increasing order by weight w
6 **for** each edge (u, v) taken from the list in sorted order
7 **if** FIND-SET$(u) \neq$ FIND-SET(v)
8 $A = A \cup \{(u, v)\}$
9 UNION(u, v)
10 **return** A

A: { }

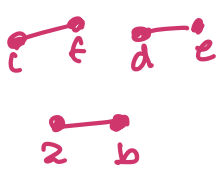
sets: a, b, c, d, e, f

add (c,f) — cf, a, b, d, e

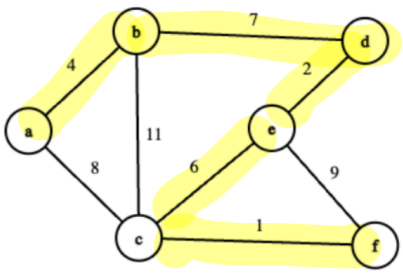
add (d,e) — cf, de, a, b



add (a,b) — cf, de, ab



add (c,e) — cfde, ab



PRIM(G, w, r)

1 **for** each vertex $u \in G.V$
2 $u.key = \infty$
3 $u.\pi =$ NIL
4 $r.key = 0$
5 $H = \{\}$
6 **for** each vertex $u \in G.V$
7 INSERT(H, u)
8 **while** $H \neq \{\}$
9 $u =$ EXTRACT-MIN(H)
10 **for** each vertex $v \in G.adj[u]$
11 **if** $v \in H$ and $w(u, v) < v.key$
12 $v.\pi = u$
13 $v.key = w(u, v)$
14 DECREASE-KEY$(H, v, w(u, v))$

	a	b	c	d	e	f
v.key	0	~~∞~~ 4	~~∞~~ 8 6	~~∞~~ 7	~~∞~~ 2	~~∞~~ 9 1
v.π	nil	a	~~a~~ e	b	~~b~~ d	~~d~~ nil e c

H = ~~a, b, c, d, e, f~~ ~~4 8 7 2 9 1~~

u = a

u = b

u = d

u = e

u = c

u = f

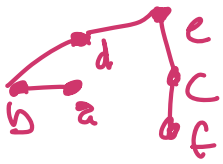
now... v.key

a = 0
b = 4
c = 6
d = 7
e = 2
f = 1

20

2nd (b,d)

(f,e,a,b)



actual optimal soln: A

value of optimal? 20

actual optimal soln: predecessors ↑↑

value of optimal: key values 20

KRUSKAL(G, w)

```
1  A = {}
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  create a single list of the edges in G.E
5  sort the list of edges into monotonically increasing order by weight w
6  for each edge (u, v) taken from the list in sorted order
7       if FIND-SET(u) ≠ FIND-SET(v)
8            A = A ∪ {(u, v)}
9            UNION(u, v)
10 return A
```

run-time:

bound by sort of E

$\Theta(E \lg E)$

in real life:

run times are basically the same!

People use Kruskal more

PRIM(G, w, r)

```
1  for each vertex u ∈ G.V
2       u.key = ∞
3       u.π = NIL
4  r.key = 0
5  H = {}
6  for each vertex u ∈ G.V
7       INSERT(H, u)
8  while H ≠ {}
9       u = EXTRACT-MIN(H)
10      for each vertex v ∈ G.adj[u]
11           if v ∈ H and w(u, v) < v.key
12                v.π = u
13                v.key = w(u, v)
14                DECREASE-KEY(H, v, w(u, v))
```

run-time:

while loop

extract, decrease $\lg V$

altogether: $\Theta((V+E)\lg V)$