

COS 226 Lecture 19: Minimal Spanning Trees (MSTs)

Classic algorithms for solving a natural problem

MINIMAL SPANNING TREE

- Kruskal's algorithm
- Prim's algorithm
- Boruvka's algorithm

Long history

Ex: power-station wiring (1920s)

Low-level algs and data structures play key role

- union-find
- priority queues
- sorting

Optimal algorithm? Still open

19.1

Generalized graph search (PFS)

TREE vertices: visited

FRINGE vertices: adjacent to visited

UNSEEN vertices: others

GRAPH SEARCH

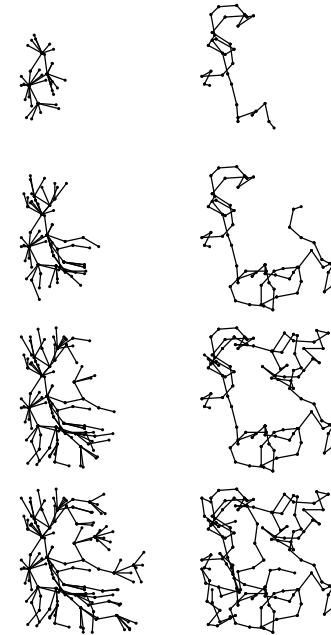
- initialize PQ with any vertex
- while PQ nonempty
 - take next vertex v off PQ
 - change v 's status to TREE
 - for each vertex adjacent to v
 - UNSEEN: move to FRINGE
 - FRINGE: update priority

Encompasses several basic graph algorithms

- DFS (use decreasing counter for priority)
- BFS (use increasing counter for priority)
- minimum spanning tree
- shortest path tree

19.2

PFS BFS and DFS examples



19.3

MST definitions

Weighted graph (value associated with each edges)

Ex:

- power distribution network
(least amount of wire to connect everything)

Spanning tree:

- set of edges that connects all the vertices

MST: no other spanning tree has smaller weight

If equal weights are present, may have more than one MST

MINIMUM (tree weight) is the smallest value

MINIMAL (tree) has minimum weight

19.4

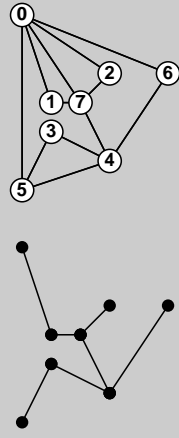
MST example

Edges

0-6 5.3
0-1 3.1
0-2 3.6
4-3 2.2
5-3 2.2
7-4 2.2
5-4 3.1
0-5 6.0
6-4 3.6
7-0 3.6
7-2 1.4
7-1 1.0

Adjacency lists

0: 7 5 2 1 6
1: 7 0
2: 7 0
3: 5 4
4: 6 5 7 3
5: 0 4 3
6: 4 0
7: 1 2 0 4



MST

1-0 7-1 2-7 4-7 3-4 5-3 6-4

19.5

Cut property

CUT: Divide vertices into two sets A and B

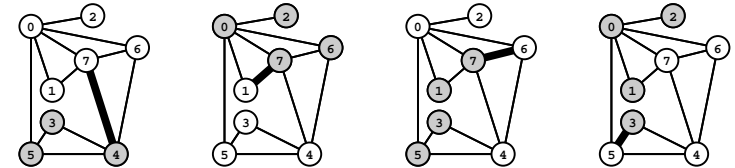
CROSSING EDGE: connects a vertex in A with a vertex in B

THM: For ANY cut,

every minimal crossing edge must be in some MST

Proof (assuming no duplicate weights):

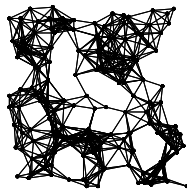
- if X is not in an MST, add it to a given MST
it creates a cycle
some other edge Y from A to B must be on the cycle
- Y must also be minimal
- delete Y and add X to get another MST



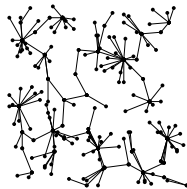
19.7

MST (larger example)

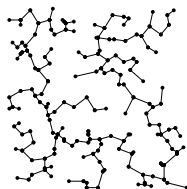
sample graph



a spanning tree



a minimal spanning tree



19.6

Priority-first search MST algorithm

PQ rule: choose SHORTEST tree-fringe edge

THM: TREE edges are MST of TREE vertices

Proof:

- Apply cut property
A: TREE vertices
B: UNSEEN and FRINGE vertices

Graph-search spanning tree must be an MST

19.8

PFS MST code

st: TREE vertices (parent links)
fr: FRINGE vertices (links to tree)
val: weights of these links

```
static int fr[maxV];
#define P t->wt
void pfs(Graph G, int v, int st[], double val[])
{ link t; int w;
  fr[v] = v; PQinsert(v);
  while (!PQempty())
  {
    v = PQdelmin(); st[v] = fr[v];
    for (t = G->adj[v]; t != NULL; t = t->next)
      if (fr[w = t->v] == -1)
        { val[w] = P; PQinsert(w); fr[w] = v; }
      else if ((st[w] == -1) && (P < val[w]))
        { val[w] = P; PQdec(w); fr[w] = v; }
  }
}
```

19.9

Priority queue issues

Need to implement

- DELETE MINIMUM
- DECREASE KEY

Priority queue holds non-tree vertices

- priority is shortest edge to tree vertex
- for each vertex added to the tree each of its edges may give a shorter connection to a non-tree vertex

REQUIRES HANDLES!

- given vertex, change its value (use val array)

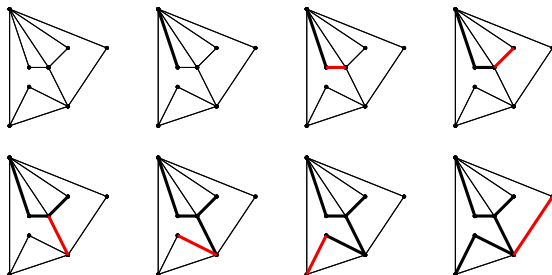
Ex: heap implementation

- priorities move around in heap
- **INDIRECT** heap: keep track of indices
- (see section 9.6 for details)

19.11

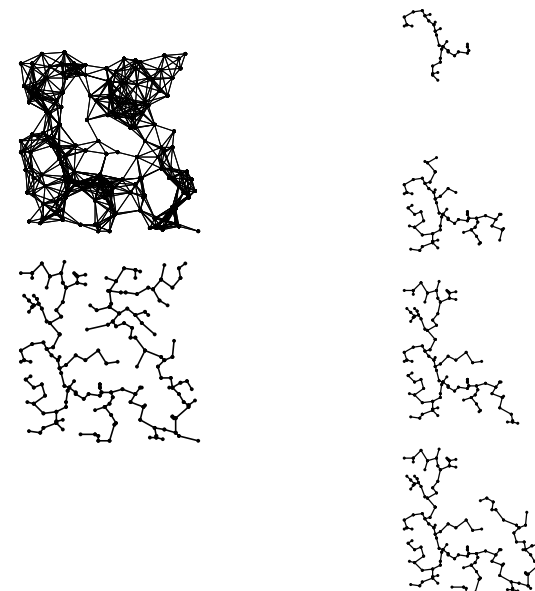
PFS MST example

.	0	1	2	3	4	5	6	7	min	out	updates
.	0	*	*	*	*	*	*	*	0	0-0	0-1 < 0-*, etc.
.	0	0	*	*	0	0	0	1	1-0	1-7 < 7-0	
.	0	*	*	0	0	1	7	7-1	7-2 < 2-0, 7-4 < 4-0		
.	7	*	7	0	0	2	2-7	none			
.	*	7	0	0	4	4-7	4-6 < 6-0, 4-5 < 5-0, 4-3 < 3-*				
.	4	4	4	3	3-4	3-5 < 5-4					
.		3	4	5	5-3	none					
.		4	6	6-4	none						



19.10

Larger PFS MST example



19.12

Prim's algorithm

Invented in 1957 (or earlier!)

No explicit priority queue

Use ADJACENCY MATRIX

V steps to check edges adjacent to current vertex

V steps to find closest fringe vertex

Do both in same loop!

Algorithm predates

- formalization of graph search
- invention of PQ

Different implementation of generic algorithm

19.13

Prim's algorithm implementation

```
#define P G->adj[v][w]
void GRAPHmstV(Graph G, int st[], double val[])
{ int v, w, min;
  for (v = 0; v < G->V; v++)
    { st[v] = -1; fr[v] = v; val[v] = maxWT; }
  min = 0; st[0] = 0; val[G->V] = maxWT;
  for (v = 0; min != G->V; st[v = min] = fr[v])
    for (w = 0, min = G->V; w < G->V; w++)
      if (st[w] == -1)
        {
          if (P < val[w])
            { val[w] = P; fr[w] = v; }
          if (val[w] < val[min]) min = w;
        }
}
```

19.14

Prim's algorithm vs. PFS implementation

Depends on whether graph is DENSE or SPARSE

- Prim's V^2
- PFS: $(E + V) \log V$
- Clever PQ: $E + V \log V$

Ex: 2 thousand vertices, 1 million edges

- PFS 2-3 times SLOWER

Ex: 100 thousand vertices, 1 million edges

- PFS 500 times FASTER

Ex: 1 million vertices, 2 million edges

- PFS tens of thousands of times FASTER

Bottom line:

- Prim's optimal for dense graphs
- PFS far better for sparse graphs

19.15

Kruskal's algorithm

Invented in 1956 (or earlier!)

UF algorithms from Lecture 1 find a spanning tree

- (keep edges that do not create a cycle)

THM:

If the edges come in order of their length,
the spanning tree from UF algs is a MST!

Algorithm:

- sort the edges
- use weighted quick union to find MST

19.16

Kruskal's algorithm implementation

```

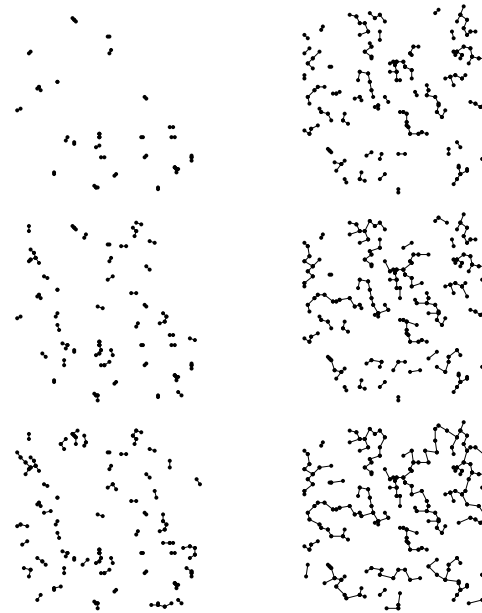
void GRAPHmstE(Graph G, Edge mst[])
{
  int i, k; Edge a[maxE];
  int E = GRAPHedges(a, G);
  sort(a, 0, E-1);
  UFinit(G->V);
  for (i= 0, k = 0; i < E && k < G->V-1; i++)
    if (!UFfind(a[i].v, a[i].w))
      {
        UFunion(a[i].v, a[i].w);
        mst[k++] = a[i];
      }
}

```

Detail: returns array of edges [see text]

19.17

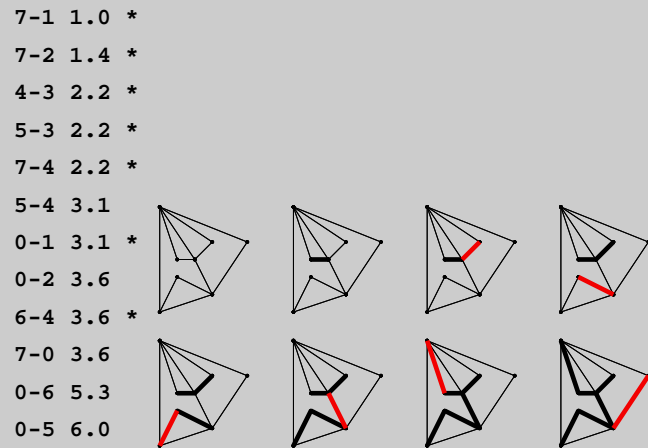
Larger Kruskal example



19.19

Kruskal MST example

Take edges in order, skip those that make cycles



19.18

Kruskal's algorithm vs PFS/Prim's

Depends on structure of graph

To within a constant factor

- PFS: $E \log V + V \log V$
- fancy PFS: $E + V \log V$
- Kruskal: $E + M \log E$

where M is the number edges in the graph shorter than the longest edge in the MST

Performance differences depend upon:

- dense vs. sparse
- structure of graph
- priority queue algorithm
- priority queue implementation

19.20

Better PQ implementations

THM: PFS implementation finds MST in

- $\leq E$ DECREASE PRIORITY ops
- V DELETE SMALLEST ops

HEAP PQ implementation

- $O(\lg V)$ for DECREASE PRIORITY
- $O(\lg V)$ for DELETE SMALLEST
- total: $O((E+V) \lg V)$

d-way HEAP PQ implementation

- total: $O((E+V) \lg_d V)$

FIBONACCI HEAP PQ implementation

- $O(1)$ for DECREASE PRIORITY
- $O(\lg V)$ for DELETE SMALLEST
- total: $O(E + V \lg V)$

Better PQ implementations improve ALL graph-search algs

19.21

Other MST algorithms

VYSSOTSKY (1960s)

- add edges one at a time
- delete longest on cycle formed

BORUVKA (1920s)

- connect each vertex to nearest neighbor
- (makes a forest)
- connect each TREE to nearest neighbor
- iterate

YAO (1995)

- THM: can find MST in $O(E \log \log V)$ steps

TARJAN et al. (1995)

- THM: randomized alg finds MST in linear time

CHAZELLE (2000)

- THM: MST same complexity as union-find

19.22