COS 226 Lecture 18: Minimal Spanning Trees

Classic algorithms for solving "network" problems

MINIMAL SPANNING TREE

- Kruskal’s algorithm
- Prim’s algorithm

SHORTEST PATHS

- Dijkstra’s algorithm
- Floyd’s algorithm

Related to generalized graph search
    using a priority queue

Ex: telephone wiring

Ex: supplies to troops

Generalized search (Priority-first)

Use PQ, rather than stack or queue

TREE vertices
    visited
FRINGE vertices
    adjacent to visited
UNSEEN vertices
    others

Put all fringe vertices on PQ

Search algorithm
- take next vertex off PQ
- change status to TREE
- for each adjacent vertex
    - UNSEEN: move to FRINGE
    - FRINGE: update prioirity

Encompasses several basic graph algorithms
- depth-first search
    - (use decreasing counter for priority)
- breadth-first search
    - (use increasing counter for priority)
- minimal spanning tree
- shortest path tree

Minimal spanning tree

Weighted graph (edges have values)

Ex: Vertices: points in space
Edges: lines connecting them
Weights: lengths of lines

MST: minimal weight set of edges
that connects all the vertices

Theorem:

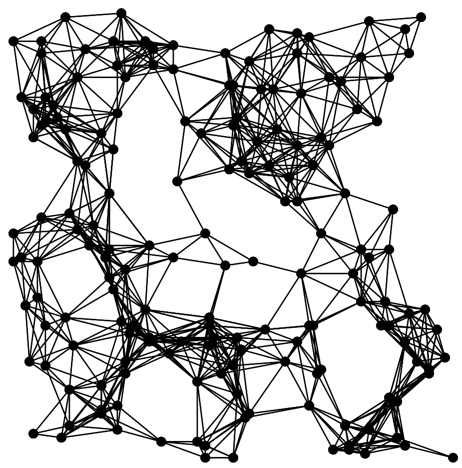
Divide the vertices into two sets A and B.
Shortest edge X connecting a vertex in A
with a vertex in B must be in the MST

Proof: by contradiction.
If X not in MST, add it to MST

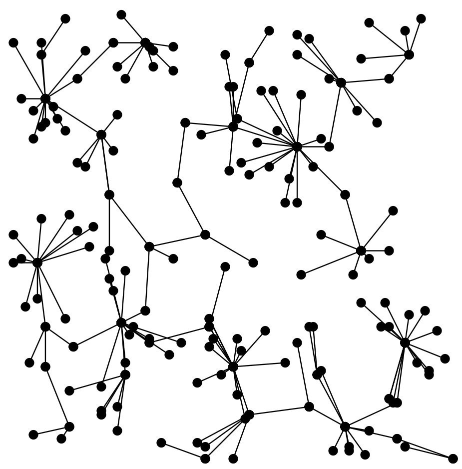
* creates a cycle
* some other edge Y from A to B
must be on cyccle

Delete Y and add X to get better tree.

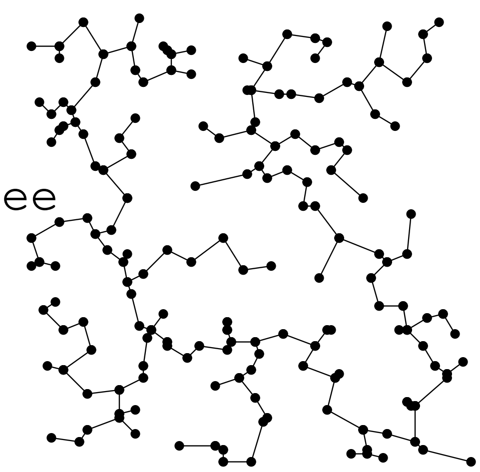
sample graph



a spanning tree



minimal spanning tree



Priority-first search MST algorithm

Set A: TREE

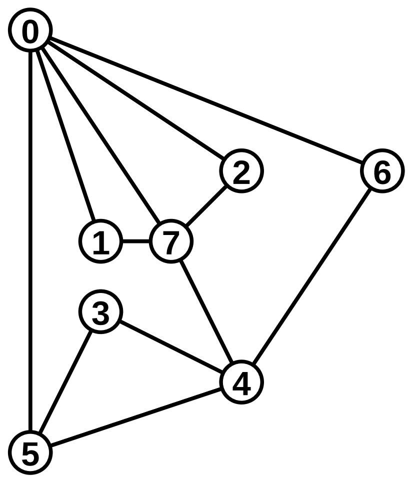
Set B: UNSEEN+FRINGE

Smallest edge in FRINGE must be in MST

```
#define P t->w
visit(link* adj, int k)
  { link t;
    dad[k] = k; PQreduce(k, 0);
    while (PQminkey() < maxNum)
      {
        k = PQdelmin();
        printf("%d-%d ", k, dad[k]);
        for (t = adj[k]; t != z; t = t->next)
          if (onPQ(t->v) && (P < val[t->v]))
            { PQreduce(t->v, P); dad[t->v] = k; }
      }
  }
listpfs(link* adj, int V, int E)
  { int k;
    val = malloc((V+1)*sizeof(weightType));
    dad = malloc(V*sizeof(int));
    PQinit(val, V);
    for (k = 0; k < V; k++)
      if (onPQ(k)) visit(adj, k);
  }
```

MST example

use line lengths
as edge weights

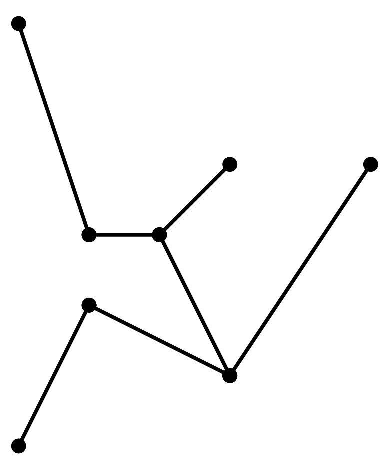


Edges

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

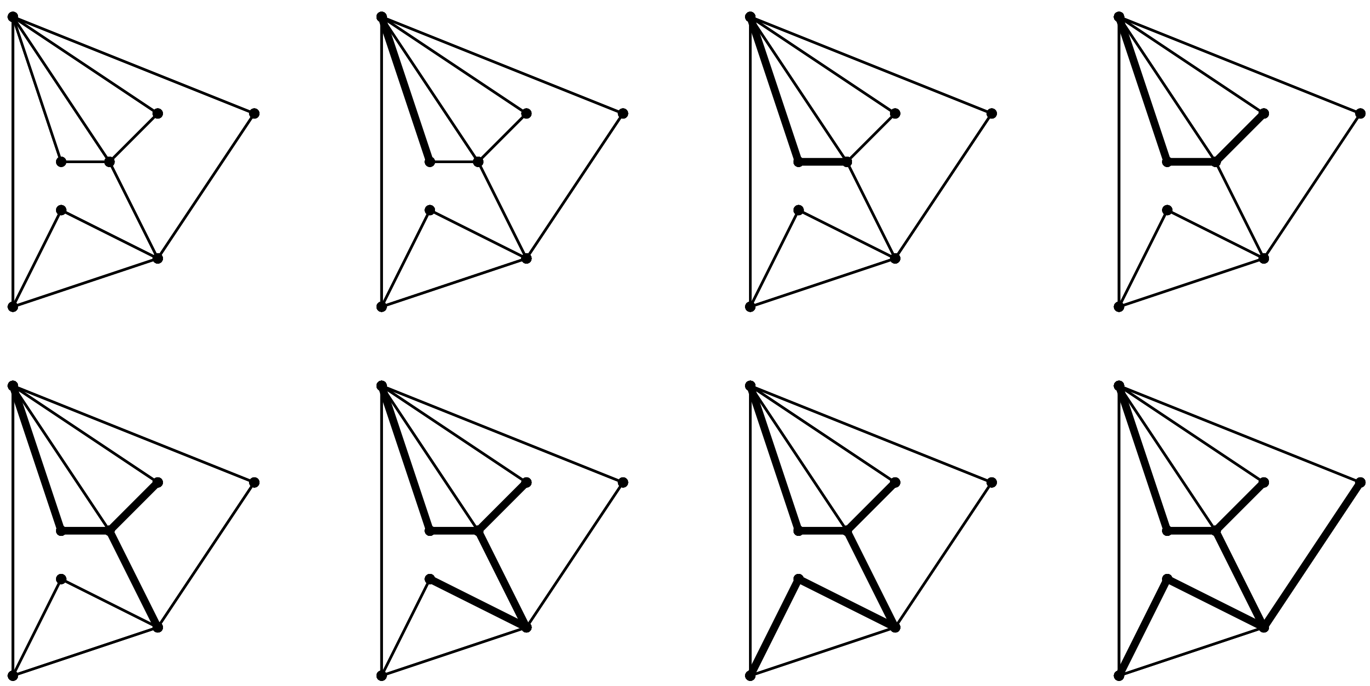


MST

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

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
```



Priority queue issues

Need to implement

    DELETE MINIMUM
    UPDATE

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, have to change its value

Example: heap implementation

    priorities move around in heap
    INDIRECT heap: keep track of indices
    (see book for details)

```
| 9.9             | 9.9
|       1.4       |       3.1
| 3.1             | 5.5
|             1.1 |             1.4
| 6.7             | 6.7
|       4.2       |       4.2
| 5.5             |
```

Another approach

    INDEXED TOURNAMENT
    Separate array on top of data
        keep track of winners
        do not move data itself

Indexed tournament implementation

```
int *pq;
int pqV; numType* pqval;
void PQinit(numType val[], int V)
   { int i;
      pqV = V; pqval = val; pqval[V] = maxNum;
      pq = malloc((V+V)*sizeof(link));
      for (i = 0; i < V; i++) pq[V+i] = V;
      for (i = V-1; i > 0; i--) pq[i] = pq[i+i];
   }
int onPQ(int v)
   { return pq[pqV+v] == v; }
numType PQminkey()
   { return val[pq[1]]; }
void promote(int k)
   { int j, l, r;
      pq[pqV+k] = k;
      for (j = (pqV+k)/2; j > 0; j = j/2)
         if (val[pq[j+j]] < val[pq[j+j+1]])
               pq[j] = pq[j+j];
         else pq[j] = pq[j+j+1];
   }
int PQdelmin()
   { int k = pq[1];
      pq[pqV+k] = pqV; promote(k); return k; }
PQreduce(int v, numType x)
   { val[v] = x; promote(v); }
```

Indexed tournament example

```
0.0 0
          0
*** 8
               0
*** 8
          8
*** 8
                   0
*** 8
          8
*** 8
               8
*** 8
          8
*** 8
```

```
    8
          1
3.1 1
               1
3.6 2
          2
*** 8
                   1
*** 8
          5
6.0 5
               7
5.3 6
          7
3.6 7
```

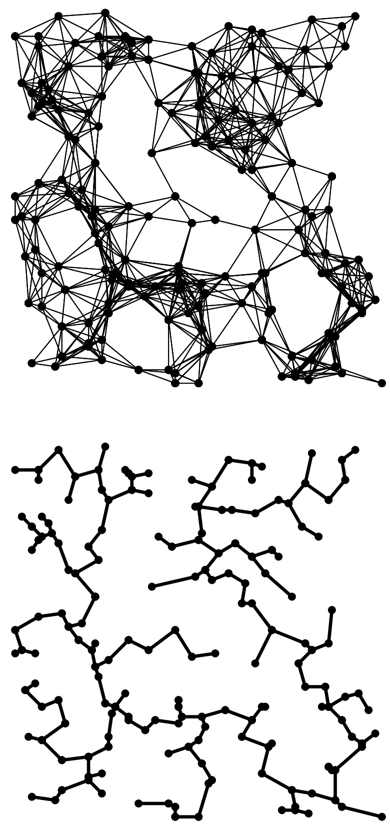
```
    8
          8
    8
               2
3.6 2
          2
*** 8
                   7
*** 8
          5
6.0 5
               7
5.3 6
          7
1.0 7
```

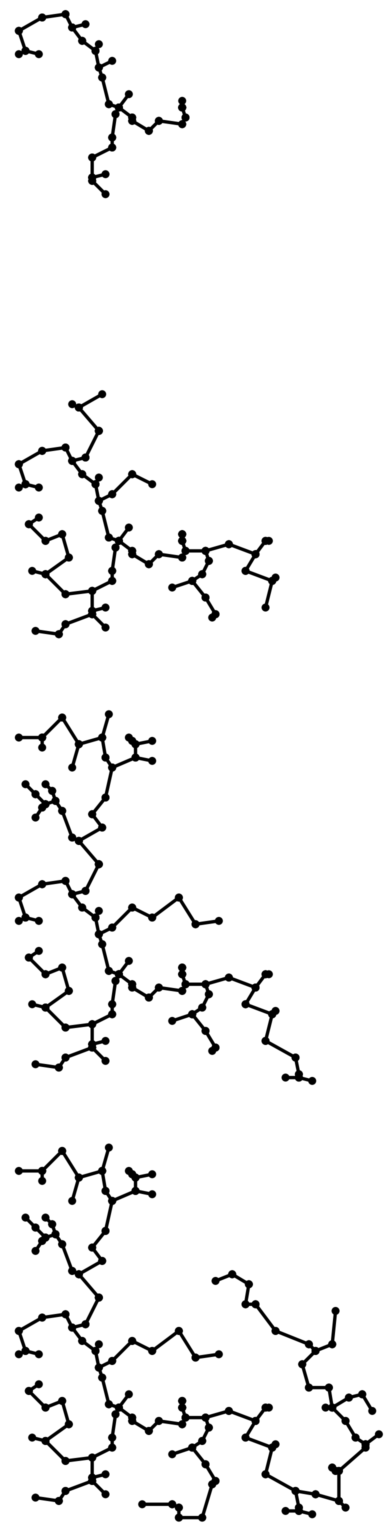
```
    8
          0
    8
               2
1.4 2
          2
*** 8
                   2
2.2 4
          4
6.0 5
               4
5.3 6
          6
    8
```

```
    8
          8
    8
               8
    8
          8
*** 8
                   4
2.2 4
          4
6.0 5
               4
5.3 6
          6
*** 8
```

```
    8
          8
    8
               3
    8
          3
2.2 3
                   3
    8
          5
3.1 5
               5
3.6 6
          6
*** 8
```

Larger PFS MST example





Prim’s algorithm

Use adjacency matrix

While going through the matrix

to find the vertices connected to the current one,

find the minimum at the same time

val[k]

neg: FRINGE; pos: TREE; –unseen: UNSEEN

```
matrixpfs()
{ int k, t, min = 0;
   for (k = 1; k <= V; k++)
      { val[k] = -unseen; dad[k] = 0; }
   val[0] = -(unseen+1); // sentinel
   for (k = 1; k != 0; k = min, min = 0)
      {
         val[k] = -val[k];
         if (val[k] == unseen) val[k] = 0;
         for (t = 1; t <= V; t++)
            if (val[t] < 0)
               {
                 if (a[k][t] && (val[t] < -priority))
                   { val[t] = -priority; dad[t] = k; }
                 if (val[t] > val[min]) min = t;
               }
         }
}
```


Invented in 1957

Prim's algorithm vs. PFS

Depends on whether graph is DENSE or SPARSE

To within a constant factor

PFS: (E + V) log V

Prim's V*V

Ex: 2 thousand vertices, 1 million edges

PFS: 11 million steps

Prims's: 4 million steps

Ex: 1 million vertices, 2 million edges

PFS: 60 million steps

Prims's: 1 trillion steps

Ex: 100 thousand vertices, 1 million edges

PFS: 17 million steps

Prim's: 100 billion steps

Clever algorithms can reduce PFS to

E + V log V

Bottom line:

* Prim's optimal for dense graphs
* don't use Prim's for sparse graphs
  (space might be prohibitive, anyway)

Kruskal's algorithm

UF algorithms from lecture 1 find a spanning tree

THEOREM:

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

In Postscript:

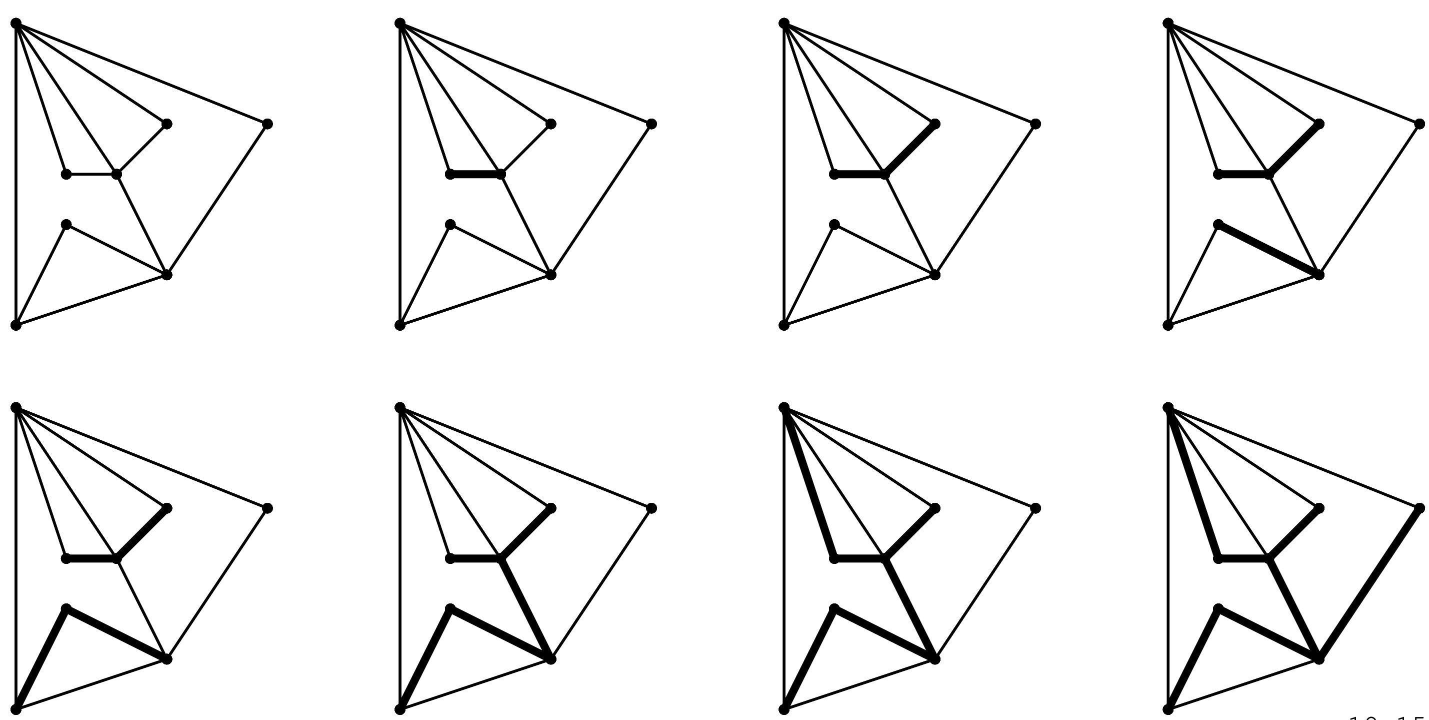
```
{
  ufwinit
  0 2 E E add 1 sub
    { /i exch def
      edges i get /v1 exch def
      edges i 1 add get /v2 exch def
      v1 v2 2 copy ufw
        { pop pop }{ drawedge } ifelse
    } for
} def
```

Invented in 1956.

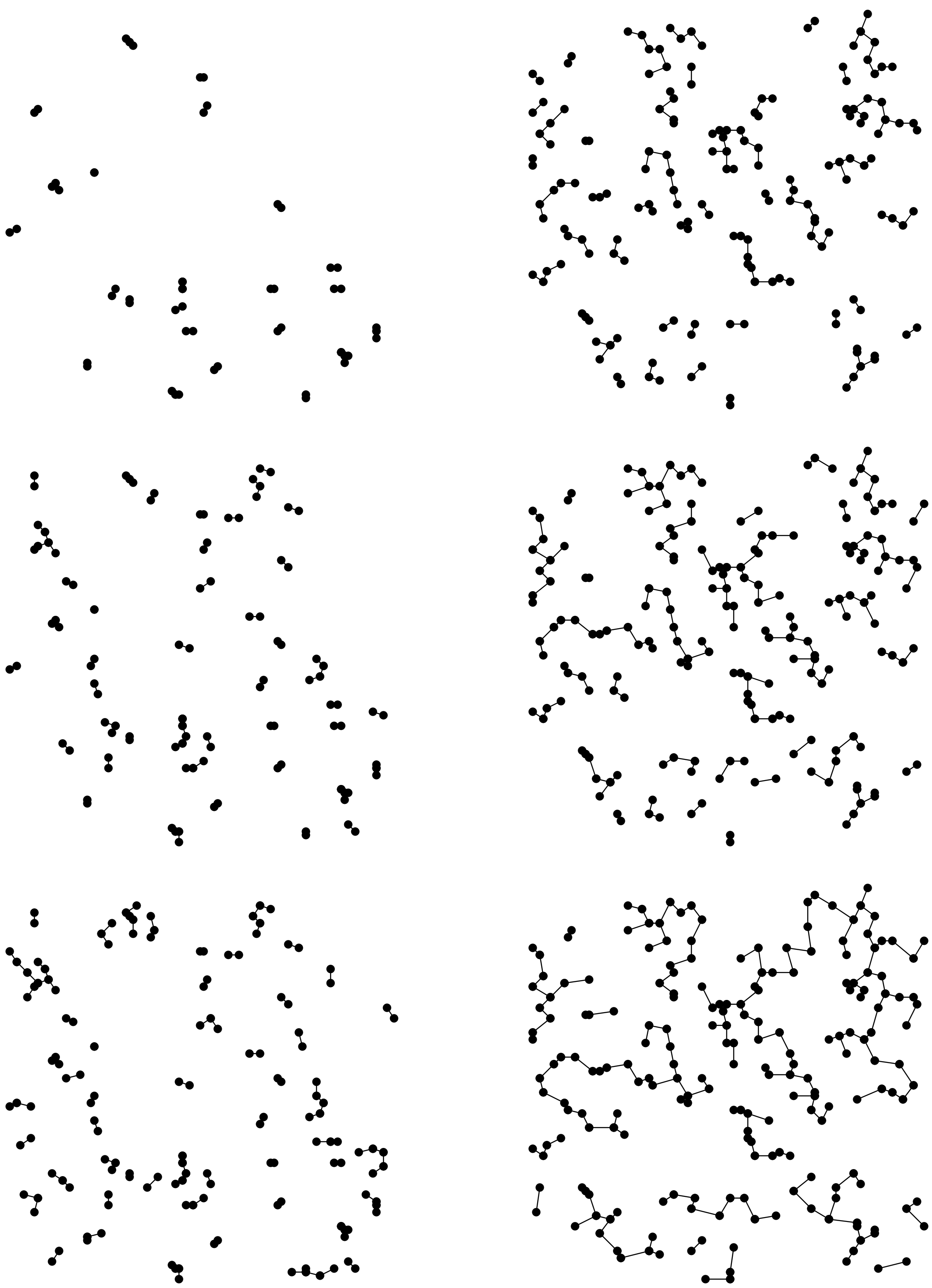
Kruskal MST example

Take edges in order, skip those that make cycles

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



Larger Kruskal example



Kruskal's algorithm vs PFS

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

Do Kruskal with heap built from the bottom up
otherwise E log E for the sort

Performance differences depend upon:

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

All require space for whole graph
(UF finds a spanning tree in O(V) space)
MST for huge graphs??