Chapter 4, Graphs:

- Concepts and implementation (Friday, Feb 11)
- Traversal (Monday, Feb 14)
- Minimum spanning trees (**Wednesday and Friday**)
- Single-source shortest paths (next week Monday)
- (Start BSTs Wednesday Mar 2)

Today:

- A couple of test solutions
- Problem: *There and back again*
- Proof and analysis of Kruskal's algorithm
- Prim's algorithm, plus proof and analysis
- Performance comparison
- Solution to *There and back again*
- Introduce SSSP problem

```
void radixSort(int[] sequence) {
    int maxTenPow = 10;     // find a power of ten greater than all items
    for (int i = 0; i < squence.length; i++)
            while (squence[i] > maxTenPow) maxTenPow *= 10;

    for (int tenPow = 1; tenPow < maxTenPow; tenPow *= 10)
        sort(sequence, new Comparator<Integer>() {
                        int compareTo(Integer a, Integer b) {
                            return (a / tenPow) % 10 - (b / tenPow) % 10;
                        } });
}
```

Invariant for the first loop.

a. $\forall\, j \in [0, i), \texttt{sequence}[j] < \texttt{maxTenPow}$.

b. $\exists\, c \in \mathbb{W} \mid \texttt{maxTenPow} = 10^c$.

c. `i` is the number of iterations completed.

```
void radixSort(int[] sequence) {
    int maxTenPow = 10;     // find a power of ten greater than all items
    for (int i = 0; i < squence.length; i++)
            while (squence[i] > maxTenPow) maxTenPow *= 10;

    for (int tenPow = 1; tenPow < maxTenPow; tenPow *= 10)
        sort(sequence, new Comparator<Integer>() {
                        int compareTo(Integer a, Integer b) {
                            return (a / tenPow) % 10 - (b / tenPow) % 10;
                        } });
}
```

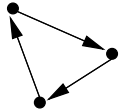
Invariant for the second loop.

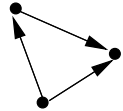
a. $\forall\, j \in [0, n-1), \texttt{sequence}[j]\%\texttt{tenPow} < \texttt{sequence}[j+1]\%\texttt{tenPow}$.

b. $\log_{10} \texttt{tenPow}$ is the number of iterations completed.

```
class X {
    public int priority;
}

// a.   O(1)
X x = xpq.max();
xpq.decreaseKey(x);


// b.   O(lg n)
X x = xpq.max();
x.priority = 0;
xpq.decreaseKey(x);


// c.  O(n)
y.priority = 0;
xpq.decreaseKey(y);
```

4.32 In the far-off land of Micomicona, all the cities have only one-way streets. Its tourism board claims that in every city it is possible to drive (legally) from any intersection in that city to any other intersection. Finish the method `VerifyThereAndBackAgain.verifyThereAndBackAgain()` which takes a `Graph` representing a city as a directed graph with intersections as vertices and street segments as edges and determines whether it is possible to go from any intersection to any other intersection. Tourists can use this method to test whether the tourism board's claim is true. For example, for the graph below left this is true, but false for the graph below right.





Hint: This can be done fairly straightforwardly using things talked about in this section in $\Theta(V^2 + VE)$ time, but with a little creativity one could find a $\Theta(V + E)$ algorithm that accomplishes this.

Minimum spanning tree problem

Given a weighted, undirected, connected graph, find minimum spanning tree:

Tree: A (sub)graph with no cycles. [We represent the tree a a set of edges.]

Spanning: All vertices in the original graph are included in the tree.

Minimum: For all spanning trees, this has least total weight.

General strategy for MST (both algorithms):

- Maintain a set of edges A that is a subset of a MST
- At each step, add one edge to A until it's a MST

Invariant (General MST main loop)

There exists $T \subseteq E$ such that T is a minimum spanning tree of G and $A \subseteq T$.

General algorithm outline:

$A = \emptyset$
While A isn't a MST
    add an edge to A that maintains the invariant

Insight 1: A implicitly partitions vertices int connected components. The lightest edge that connects two components is safe.

Lemma (Safe edges in Kruskal's algorithm.)

If $G = (V, E)$ is a graph, A is a subset of a minimum spanning tree for G, and (u, v) is the lightest edge connecting any distinct connected components of A, then (u, v) is a safe edge for A, that is, $A \cup \{(u, v)\}$ is a subset of a minimum spanning tree.

Invariant (General MST main loop)

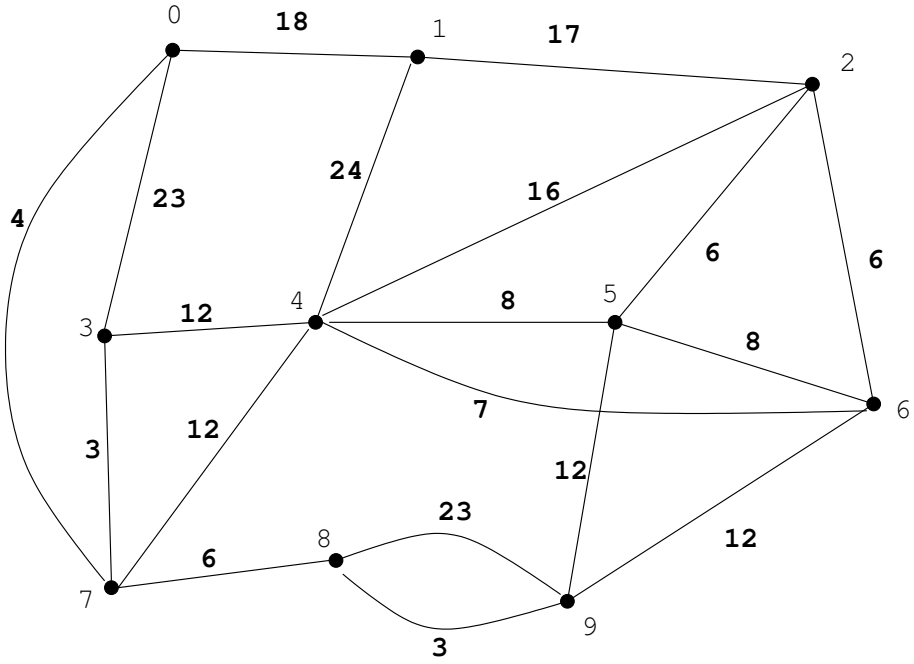
There exists $T \subseteq E$ such that T is a minimum spanning tree of G and $A \subseteq T$.

Insight 1: A implicitly partitions vertices int connected components. The lightest edge that connects two components is safe.

Invariant (Prim's algorithm main loop)

A is a (single) tree.

Insight 2: The lightest edge that connects a new vertex to A is safe.


0
1
2
3
4
5
6
7
8
9
18
17
4
23
24
16
6
6
12
8
8
7
3
12
12
23
6
12
3

Minimum Spanning Tree Problem

Given a weighted, undirected graph, find the tree with least-total weight that connects all the vertices, if one exists.

Single-Source Shortest Paths Problem

Given a weighted directed graph and a source vertex, find the tree comprising the shortest paths from that source to all other reachable vertices.

- Both are defined for weighted graphs
- Both produce trees as a result
- Both minmize by weight
- For each we have two algorithms

Input is only a graph
Problem usually is described on an undirected graph
Goal is to minimize total weight
There is no clear winner between the algorithms

Input is a graph and a starting point
Problem usually is described on a directed graph
Goal is to minimize weight on each path
One algorithm is clearly more efficient

Coming up:

Due **Fri, Feb 25** *[class time]*
Read Section 4.4
Do Exercises 4.(39, 41, 42)
Take MST quiz

Do **MST** *project (suggested by Monday, Feb 28)*

Due **Tues, Mar 1** *[end of day]*
Read Section 4.5
Do Exercises 4.(47, 48, 56) (note correction on 4.52)
Take SSSP quiz

Do **SSSP** *project (suggested by Thursday, Mar 3)*

Due **Fri, Mar 4** *[class time]*
Read Sections 5.(1 & 2)
Do Exercises 5.(2 & 6)
Take BST quiz