Lecture 31

Graphs III

- Graph Models
  - Bloxorz
- structure Package
- Minimum Spanning Trees
  - Greedy Algorithms
  - Prim’s Algorithm
Graph Models
Bloxorz
How can we model and solve Bloxorz using graph reachability or as a shortest path problem? What is the underlying graph?

- **There is one vertex per state.** We can define a state as follows: The top-left cell occupied by the bloxorz shape (which is the only cell when it is standing upright), and its orientation (i.e., laying down vertically or laying down horizontally or standing upright). For example, the above states are (2,2,horizontal) and (2,4,upright) when labeling the top-left cell (0,0).

- **There is an edge between two states if they differ by a single \( \leftarrow, \uparrow, \rightarrow, \downarrow \) button press.** So most vertices have degree 4. A vertex has smaller degree when a move falls off the board. For example, the two states (or vertices) given above are joined by an edge.
structure Package
Exercise: What is Graph in structure?

What do you think a graph should be in the structure package?

- Will it be an interface, a class, or an abstract class?
- Will it implement any other interfaces?
- Will it extend from any other classes?
- Will it have any type variables?

Think to yourself for 30 seconds.
Debate with a neighbor for 1 minute.

There is no “correct” answer.
However, you should be able to justify your answer.
Graph.java provides a Graph interface. There is a separate Edge class in Edge.java.
GraphList.java provides an abstract class GraphList that implements Graph.
GraphListUndirected.java provides a non-abstract (concrete) class GraphListUndirected that extends GraphList (and hence implements Graph).
GraphMatrixUndirected.java is another non-abstract (concrete) class. Note that this implementation never resizes the underlying matrix.

- When deleting a vertex it doesn’t remove rows or columns; it just “hides” the vertex.
Minimum Spanning Trees
Spanning Trees

Given a graph $G = (V, E)$ a *spanning tree* is a subset of edges $T \subseteq E$ with the following properties:

1. $T$ is *spanning*. This means that every $v \in V$ is incident with at least one edge in $T$.
2. $T$ is a *tree*. This means that the following two points hold.
   - The graph induced by $T$ is *connected*. In other words, there is at least one path between any two vertices in $T$.
   - The graph induced by $T$ is *acyclic*. In other words, there is at most one path between any two vertices in $T$.
Minimum-Weight Spanning Trees

In many applications there is a weight (or cost) associated with each edge, and these values together with the graph are called a weighted graph or an edge-labeled graph.

The weight of a spanning tree is the sum of the weights on its edges.

A minimum-cost or minimum-weight spanning tree is a spanning tree with the smallest weight.

A weighted graph \((G, w)\) with a spanning tree of weight \(5 + 10 + 10 + 10 + 10 + 20 + 25 + 40 = 130\). Is this a minimum spanning tree?
Sample Application of MST
Connect homes to electrical grid while minimizing cost (e.g., total wire, install time, road crossings).
Greedy Algorithms
Algorithmic Paradigms

In CSCI 256 you will learn about a handful of *algorithmic paradigms*. Essentially, these are general strategies that can be used when developing an algorithm to solve a particular problem.

- **Brute force.** Generate all possible solutions and test them (e.g., Two Towers lab).
- **Divide & Conquer.** Split a problem into smaller subproblems, then combine solutions (e.g., §6.4 merge sort).
- **Dynamic Programming.** Solve subproblems from smallest to largest (e.g., Floyd-Warshall §16.4.4).
- **Greedy.** Make the “best” next choice. Repeat. Hope for the best! (e.g., Prim’s algorithm §16.4.5).
- **Iterative Improvement.** Continually improve the current solution until it is optimal (e.g. §6.1 bubble sort).

16.4.5 Greedy Algorithms

We now consider two examples of greedy algorithms—algorithms that compute optimal solutions to problems by acting in the optimal or “most greedy” manner at each stage in the algorithm. Because both algorithms seek to find the best choice for the next step in the solution process, both make use of a priority queue.

The textbook’s brief description of greedy algorithms.

Note: There are other paradigms as well.
Greedy Algorithms #YOLO

A greedy algorithm always makes the next “best” local choice, and hopes these choices lead to a global solution.

What is the “best” next choice?

- Problems often have more than one way to define what the next “best” choice will be.
- Each definition of “best” leads to a different greedy algorithm.
- Some definitions of “best” work (i.e., the local choices lead to a global solution) and others do not.

In other words, a greedy algorithm always uses the same “best” choice, and there can be more than one natural greedy algorithm for a given problem, and some of these algorithms work and some do not.

What makes a good “best” choice?

- It should seem related to the overall goal.
- It should be easy to explain and easy to calculate.

The first point means that the choice is reasonable. But it won’t necessarily be obvious that it will work or not.

Some consider Dijkstra’s algorithm (§16.4.5) to be a greedy algorithm for finding shortest paths. For others, it does too much planning and computing, so it violates the last point.

How do we know that a greedy algorithm works?

- It requires a proof. You’ll see examples of this in CSCI 256.
Discussion: Greedy Algorithms for Minimum Spanning Tree

Suppose that we want to try to create a greedy algorithm for finding minimum spanning trees. What are some different ideas that we could try for the next “best” choice?

Think about this for 30 seconds.
Then discuss it with your neighbor for 3 minutes.

Notes:

- Your idea for “best” should be specific enough to be implemented by an algorithm. For example, the idea “add the smallest weight edge” is incomplete.
  - Are there any other restrictions on which edge you are adding?
  - You can’t add every edge, so when do you not add the smallest weight edge?
- There are several greedy ideas that work (see Wikipedia).
  - Consider ideas that don’t involve adding a single edge at a time.
Prim’s Algorithm*

* This is the algorithm behind `mst` in §16.4.5 of the textbook. It was discovered by Robert Prim, Vojtěch Jarník. See Wikipedia articles [Prim's Algorithm](https://en.wikipedia.org/wiki/Prim%27s_algorithm) and [Stigler's Law of Eponomy](https://en.wikipedia.org/wiki/Stigler%27s_Law_of_Eponomy).
Prim's Algorithm

Prim's algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.
Prim's Algorithm

Prim’s algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.

Building a minimum spanning tree using Prim's algorithm starting from vertex B.
Prim's Algorithm

Prim’s algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.

Building a minimum spanning tree using Prim's algorithm starting from vertex B.
Prim's Algorithm

Prim’s algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.

Building a minimum spanning tree using Prim's algorithm starting from vertex B.
Prim's Algorithm

Prim's algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.

Building a minimum spanning tree using Prim's algorithm starting from vertex B.
Prim's Algorithm

Prim's algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.

Building a minimum spanning tree using Prim's algorithm starting from vertex B.
Prim's Algorithm

Prim’s algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.

Building a minimum spanning tree using Prim's algorithm starting from vertex B.
Prim's Algorithm

Prim's algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.

Building a minimum spanning tree using Prim's algorithm starting from vertex B.
Prim's Algorithm

Prim's algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.

Building a minimum spanning tree using Prim's algorithm starting from vertex B.
Prim's Algorithm

Prim's algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.
Prim's Algorithm

Prim's algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The “best” choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.

Building a minimum spanning tree using Prim's algorithm starting from vertex B.
Prim's Algorithm

Prim's algorithm grows a tree one vertex at a time. It starts from a single vertex.

- The "best" choice is adding an edge of minimum weight that grows the tree. Break ties arbitrarily. That is, one vertex of the edge is inside of the tree and the other vertex is outside of the tree. We can refer to these as the available or fringe edges.
- Stop when all vertices are in the tree.

Building a minimum spanning tree using Prim's algorithm starting from vertex B.
An implementation of Prim’s algorithm from the textbook. The available edges are updated in q. Our file Prim.java also has a main method that runs the algorithm on our sample graph.