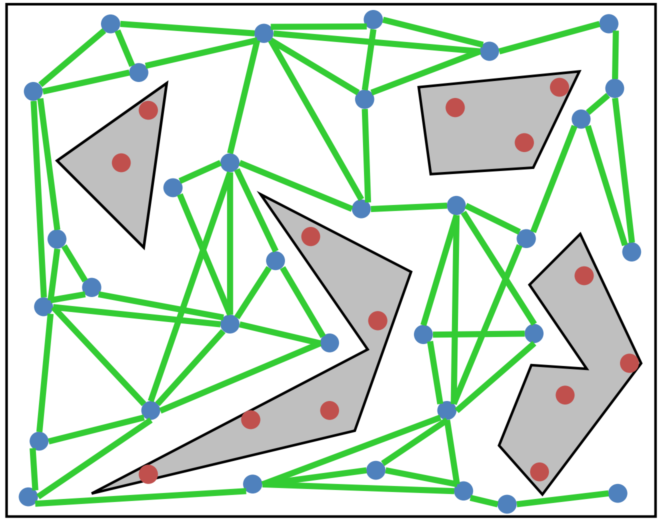
MEAM 520
Lecture 13: Graph Representations Practical

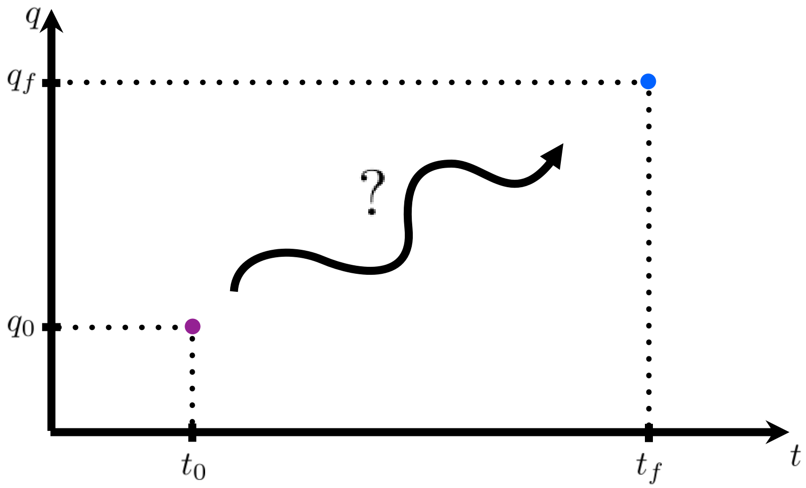
Cynthia Sung, Ph.D.
Mechanical Engineering & Applied Mechanics
University of Pennsylvania

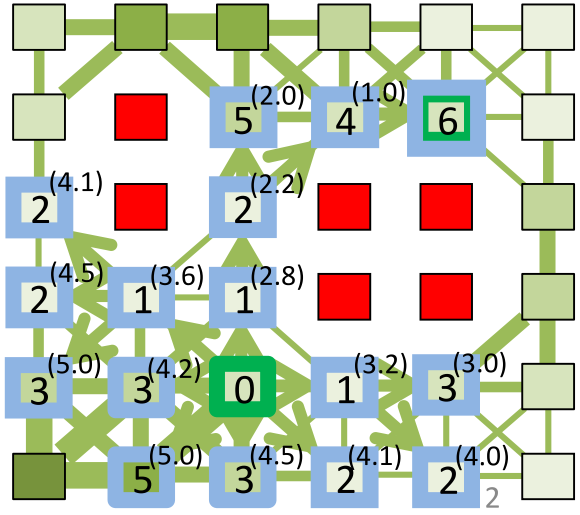
Last Time: Trajectory Planning

Planning strategy:

1. Convert your free C-space into a graph/roadmap
2. Find a path from q_{start} to a node q_a that is in the roadmap
3. Find a path from q_{goal} to a node q_b that is in the roadmap
4. Search the roadmap for a path from q_a to q_b



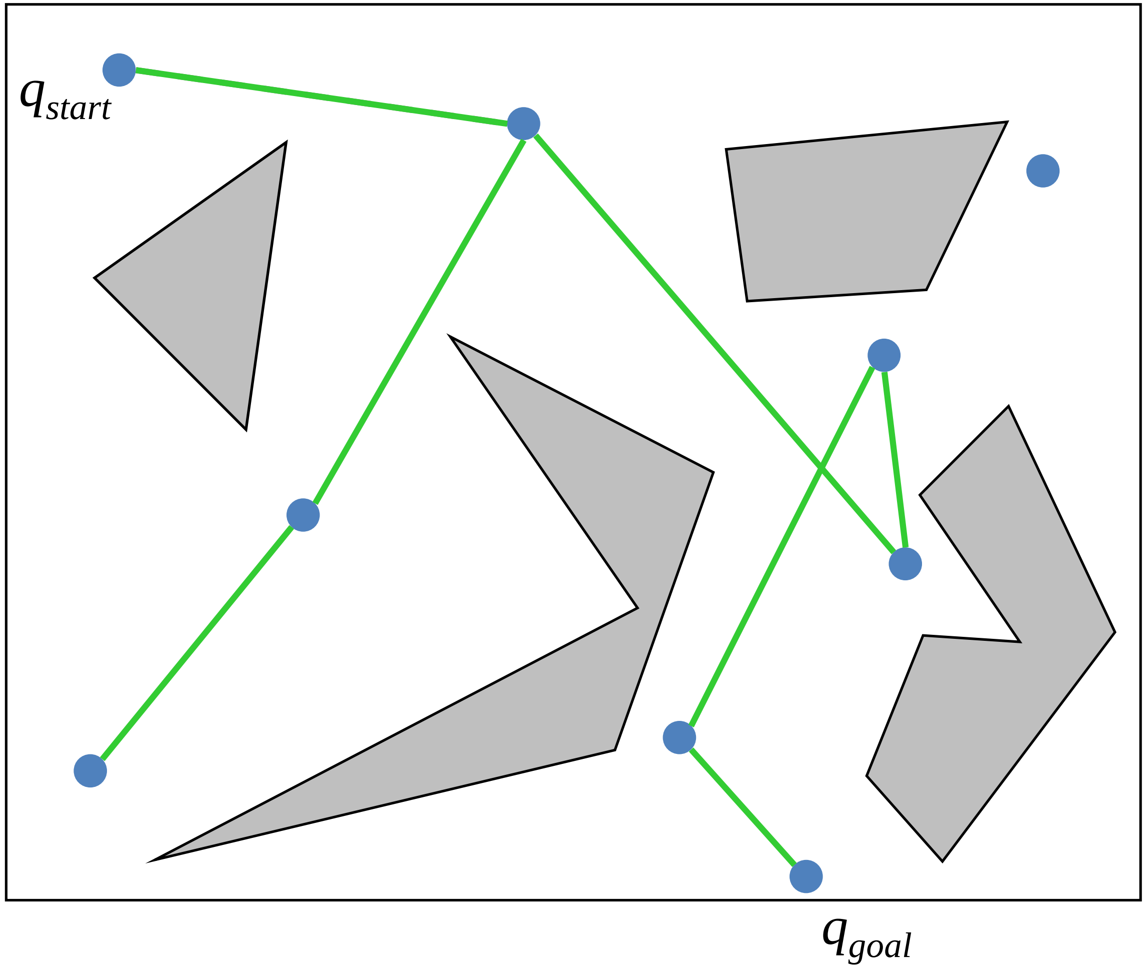




Last Time: Rapidly-exploring Random Trees (RRTs)

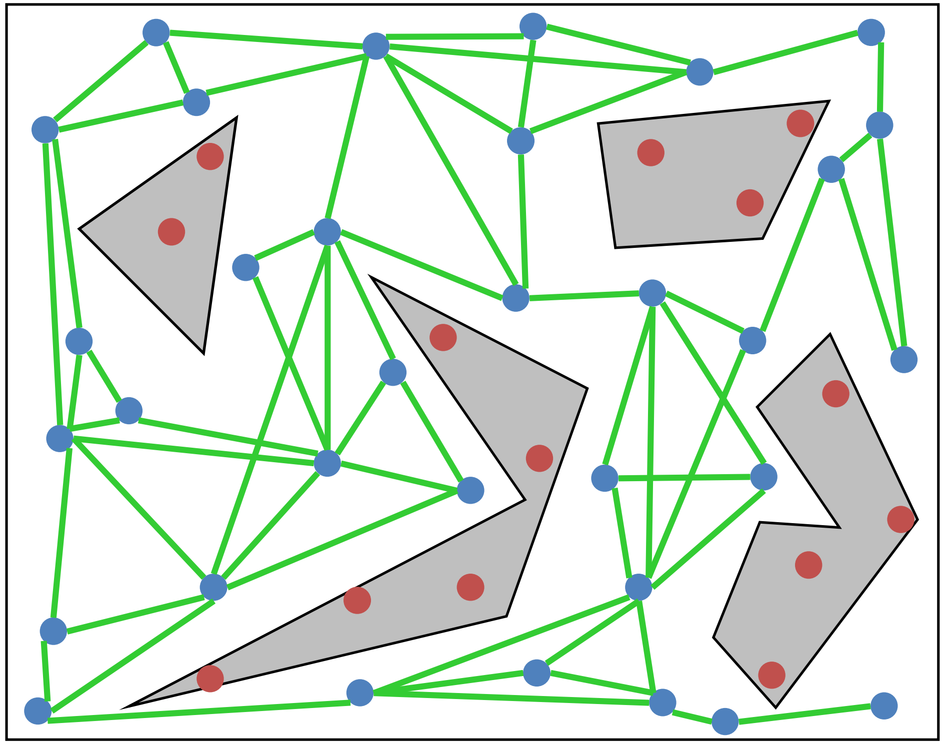
Combine graph construction and path search

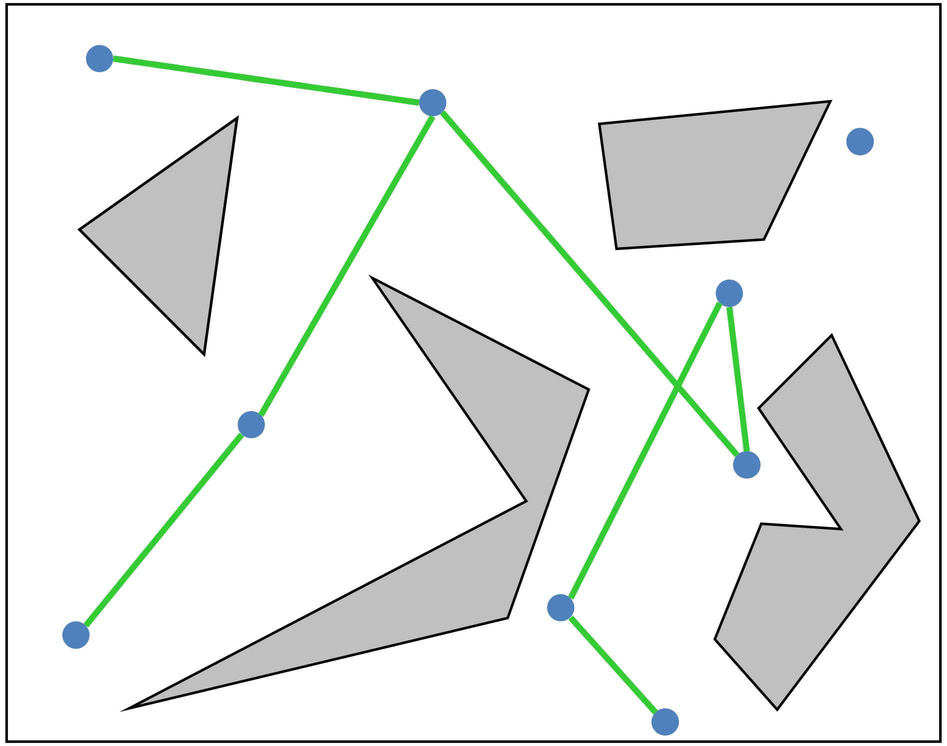
Bias the search by changing your sampling distribution or connection strategy



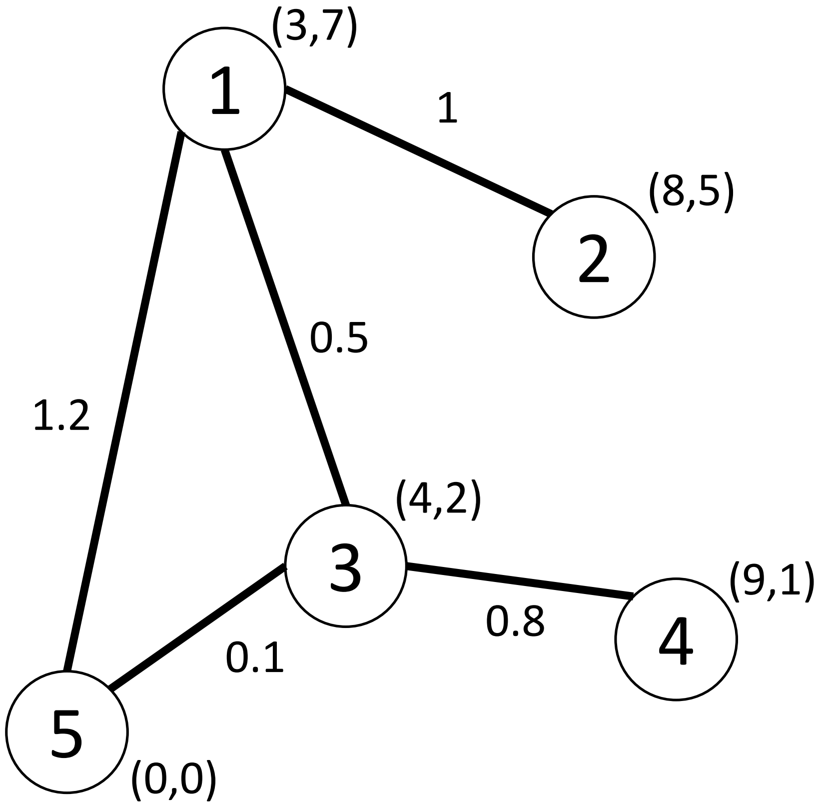
These algorithms are fine to describe pictorially.

But how to do we actually represent this graph structure?





Graph Components



$$V = \{v_1, v_2, v_3, v_4, v_5\}$$

The edge $\{v_i, v_j\}$ connects vertices v_i and v_j

$$E = \{ \{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_5\}, \{v_3, v_4\}, \{v_3, v_5\} \}$$

In code, we can write:

```
V = [3, 7;
     8, 5;
     4, 2;
     9, 1;
     0, 0]
```

Each row is a coordinate location

```
E = [1, 2, 1.0;
     1, 3, 0.5;
     1, 5, 1.2;
     3, 4, 0.8;
     3, 5, 0.1]
```

vertices cost

Example Use: Dijkstra

```
V = [3, 7;
     8, 5;
     4, 2;
     9, 1;
     0, 0]

E = [1, 2, 1.0;
     1, 3, 0.5;
     1, 5, 1.2;
     3, 4, 0.8;
     3, 5, 0.1]
```

Say we are currently expanding vertex `i`

This step can be slow

1. Search first 2 columns of E for `i`
2. Add the other vertex to the neighbor list and the cost from the 3rd column

Note: if you have a cost metric, you may have to go back into the V matrix to calculate the cost.

Adjacency Matrix

```
E = [1, 2, 1.0;
     1, 3, 0.5;
     1, 5, 1.2;
     3, 4, 0.8;
     3, 5, 0.1]
```

```
E = [0, 1, 1, 0, 1;
     1, 0, 0, 0, 0;
     1, 0, 0, 1, 1;
     0, 0, 1, 0, 0;
     1, 0, 1, 0, 0]
```

Entry `(i,j)` is 1 if there is an edge
0 if there is no edge

Looking up neighbors is really fast!

It is possible to also store costs

Adjacency Matrix

```
E = [1, 2, 1.0;
     1, 3, 0.5;
     1, 5, 1.2;
     3, 4, 0.8;
     3, 5, 0.1]
```

```
E = [Inf, 1.0, 0.5, Inf, 1.2;
     1.0, Inf, Inf, Inf, Inf;
     0.5, Inf, Inf, 0.8, 0.1;
     Inf, Inf, 0.8, Inf, Inf;
     1.2, Inf, 0.1, Inf, Inf]
```

Entry `(i,j)` is `cost` if there is an edge
`Inf` if there is no edge

Looking up neighbors is really fast!

It is possible to also store costs

Adjacency Matrix

```
E = [1, 2, 1.0;
     1, 3, 0.5;
     1, 5, 1.2;
     3, 4, 0.8;
     3, 5, 0.1]
```

```
E = [Inf, 1.0, 0.5, Inf, 1.2;
     1.0, Inf, Inf, Inf, Inf;
     0.5, Inf, Inf, 0.8, 0.1;
     Inf, Inf, 0.8, Inf, Inf;
     1.2, Inf, 0.1, Inf, Inf]
```

Entry `(i,j)` is `cost` if there is an edge
`Inf` if there is no edge

Looking up neighbors is really fast!

It is possible to also store costs

BUT, these extra 0/Inf entries take a lot of space

Adjacency List: Store edges by vertex

```
E = [1, 2, 1.0;
     1, 3, 0.5;
     1, 5, 1.2;
     3, 4, 0.8;
     3, 5, 0.1]
```

```
E = {[2,1.0; 3,0.5; 5,1.2],
     [1,1.0],
     [1,0.5; 4,0.8; 5,0.1],
     [3,0.8],
     [1,1.2; 3,0.1]}
```

Entry `i` contains all of the edges incident on `i`

Only take as much storage space as needed

It is possible to also store costs

Looking up neighbors is fast (though not as fast as Adjacency Matrices)

Adjacency Matrices vs Adjacency Lists

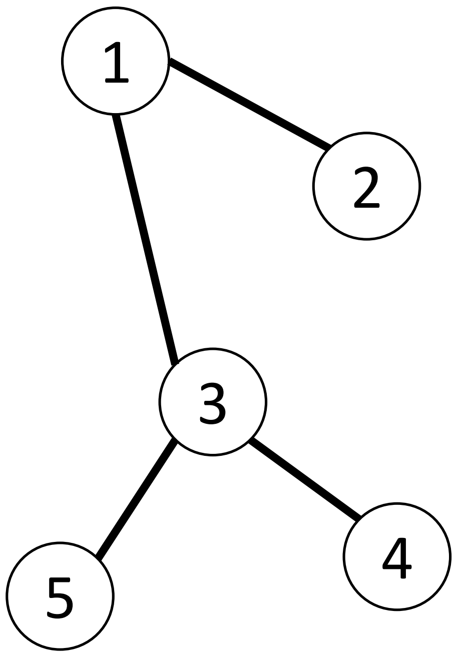
```
E = [Inf, 1.0, 0.5, Inf, 1.2;
     1.0, Inf, Inf, Inf, Inf;
     0.5, Inf, Inf, 0.8, 0.1;
     Inf, Inf, 0.8, Inf, Inf;
     1.2, Inf, 0.1, Inf, Inf]
```

- Use this when you have a dense graph (lots of edges)
- Use this when neighbor lookup has to be VERY fast

```
E = {[2,1.0; 3,0.5; 5,1.2],
     [1,1.0],
     [1,0.5; 4,0.8; 5,0.1],
     [3,0.8],
     [1,1.2; 3,0.1]}
```

- Use this when you have a sparse graph (not many edges)
- Use this when adding new vertices has to be very fast

Example Use: RRT



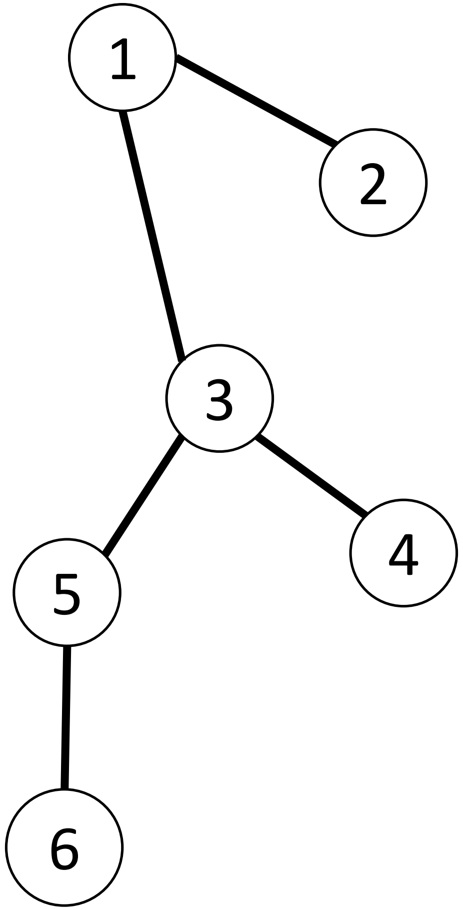
```
V = [3, 7;
     8, 5;
     4, 2;
     9, 1;
     0, 0]
```

```
E = [1, 2;     E = [0,1,1,0,0;     E = {[2,3];
     1, 3;          1,0,0,0,0;          [1];
     3, 4;          1,0,0,1,1;          [1,4,5];
     3, 5]          0,0,1,0,0;          [3];
                    0,0,1,0,0];         [3]}
```

Example Use: RRT

Needed operations:

1. Add a new vertex




```
V = [3, 7;
     8, 5;
     4, 2;
     9, 1;
     0, 0;
     0,-2]
```

```
E = [1, 2;
     1, 3;
     3, 4;
     3, 5]
```

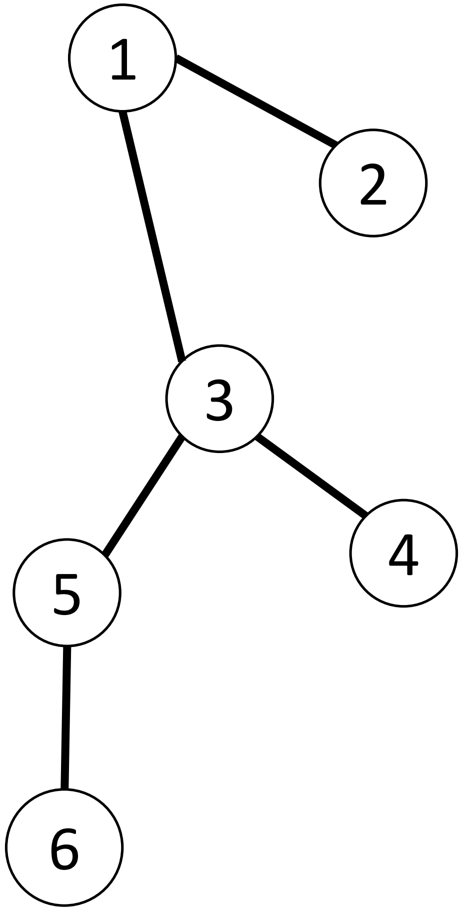
```
E = [0,1,1,0,0;
     1,0,0,0,0;
     1,0,0,1,1;
     0,0,1,0,0;
     0,0,1,0,0];
```

```
E = {[2,3];
     [1];
     [1,4,5];
     [3];
     [3]}
```

Example Use: RRT

Needed operations:

1. Add a new vertex
2. Add a new edge




```
V = [3, 7;
     8, 5;
     4, 2;
     9, 1;
     0, 0;
     0,-2]
```

```
E = [1, 2;
     1, 3;
     3, 4;
     3, 5;
     5, 6]
```

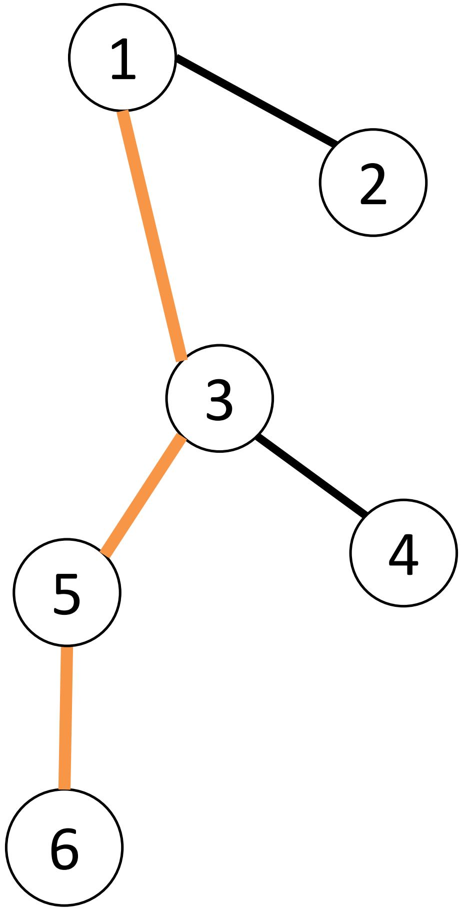
```
E = [0,1,1,0,0,0;
     1,0,0,0,0,0;
     1,0,0,1,1,0;
     0,0,1,0,0,0;
     0,0,1,0,0,1;
     0,0,0,0,1,0];
```

```
E = {[2,3];
     [1];
     [1,4,5];
     [3];
     [3,6];
     [5]}
```

Example Use: RRT

Needed operations:

1. Add a new vertex
2. Add a new edge
3. Backtrack the path



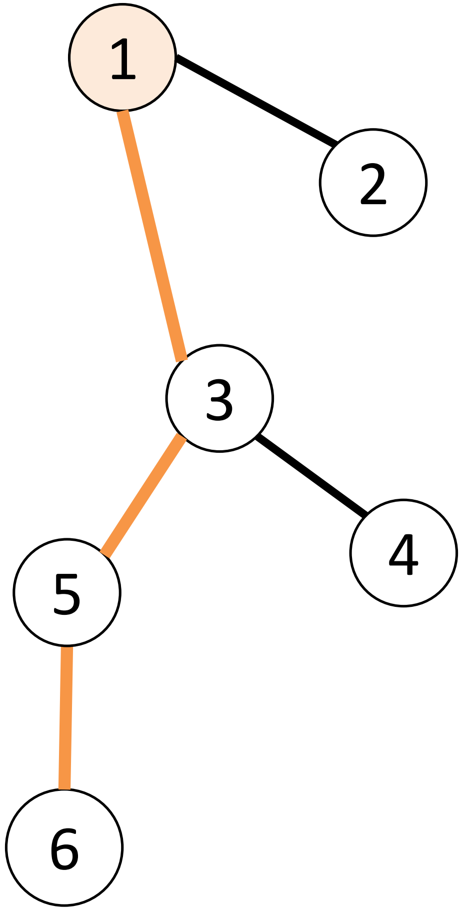
$$V = \begin{bmatrix} 3, & 7; \\ 8, & 5; \\ 4, & 2; \\ 9, & 1; \\ 0, & 0; \\ 0, & -2 \end{bmatrix}$$

$$E = \begin{bmatrix} 1, & 2; \\ 1, & 3; \\ 3, & 4; \\ 3, & 5; \\ 5, & 6 \end{bmatrix}$$

$$E = \begin{bmatrix} 0,1,1,0,0,0; \\ 1,0,0,0,0,0; \\ 1,0,0,1,\cancel{1},0; \\ 0,0,1,0,0,0; \\ 0,0,1,0,0,\cancel{1}; \\ 0,0,0,0,1,0 \end{bmatrix};$$

$$E = \{[2,3]; \; [1]; \; [1,4,\cancel{5}]; \; [3]; \; [3,\cancel{6}]; \; [5]\}$$

Example Use: RRT



V = [3, 7;
8, 5;
4, 2;
9, 1;
0, 0;
0,-2]

E = [1, 2;
1, 3;
3, 4;
3, 5;
5, 6]

Needed operations:

1. Add a new vertex
2. Add a new edge
3. Backtrack the path

Trees are special because there is a ROOT node

We are trying to find a path back to the root

Add some directionality to the representation!

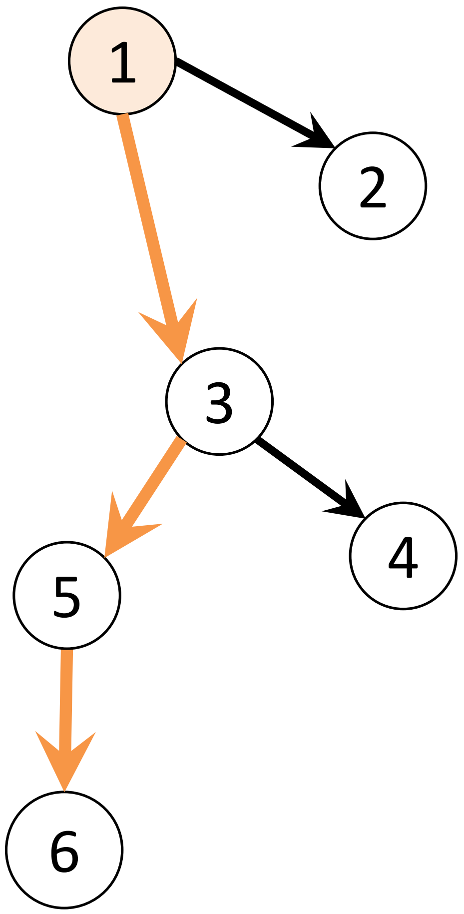
E = [0,1,1,0,0,0;
1,0,0,0,0,0;
1,0,0,1,~~1~~,0;
0,0,1,0,0,0;
0,0,1,0,0,~~1~~;
0,0,0,0,1,0];

E = {[2,3];
[1];
[1,4,~~5~~];
[3];
[3,~~6~~];
[5]}

Example Use: RRT

Needed operations:

1. Add a new vertex
2. Add a new edge
3. Backtrack the path




```
V = [3, 7;
     8, 5;
     4, 2;
     9, 1;
     0, 0;
     0,-2]
```

Trees are special because there is a ROOT node

We are trying to find a path back to the root

Add some directionality to the representation!

```
E = [1, 2;
     1, 3;
     3, 4;
     3, 5;
     5, 6]
```

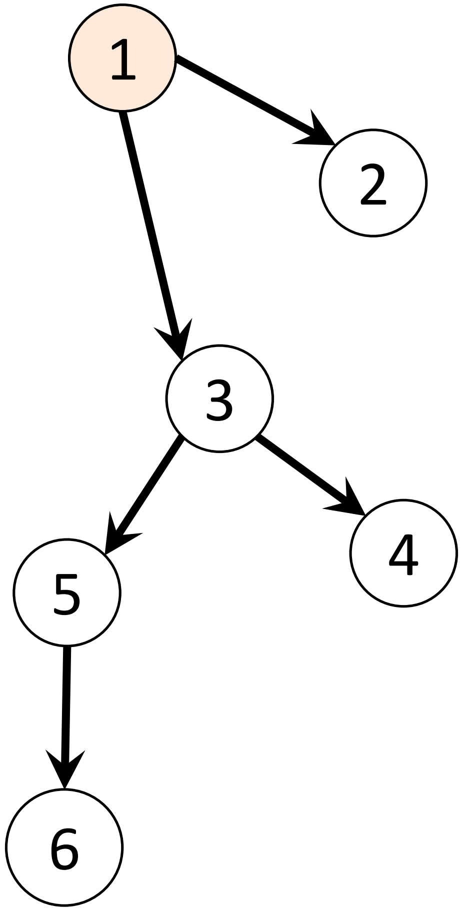
directionality inherent in column order

```
E = [0,1,1,0,0,0;
     -1,0,0,0,0,0;
     -1,0,0,1,1,0;
     0,0,-1,0,0,0;
     0,0,-1,0,0,1;
     0,0,0,0,-1,0];
```

Use -1s for “from” node and 1s for “to” nodes

```
E = {[2,3]; ??
     [1];
     [1,4,5];
     [3];
     [3,6];
     [5]}
```

Tree Representations




```
Node {
    coord: [x,y]
    parent: i
    children: [j,k,...]
}
```

Implement this as a struct in MATLAB

```
v(1) = {
    coord: [3,7]
    parent: NaN
    children: [2,3]
}
```

```
v(3) = {
    coord: [4,2]
    parent: 1
    children: [4,5]
}
```

To find a path back to the root, follow the parent pointer

Tree struct vs Adjacency Info

```
Node {
   coord: [x,y]
   parent: i
   children: [j,k,...]
   cost-to-goal: c
}
```

```
V = [3, 7;
     8, 5;
     4, 2;
     9, 1;
     0, 0;
     0,-2]
```

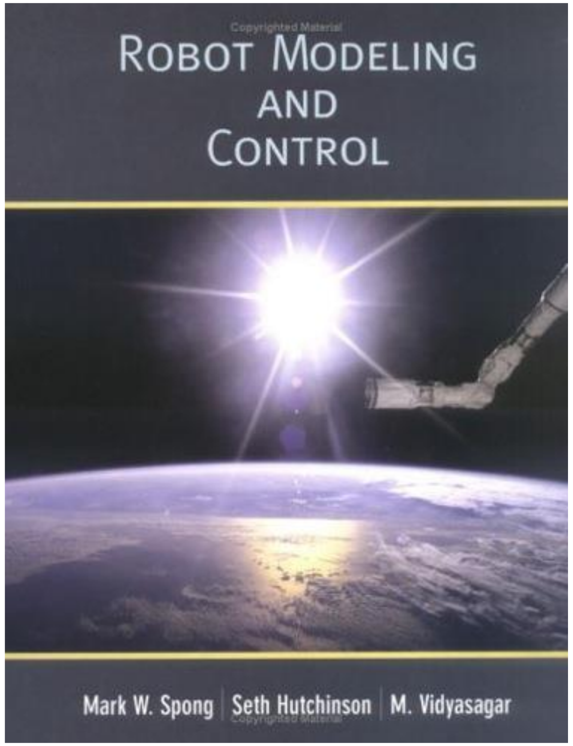
```
E = [1, 2;
     1, 3;
     3, 4;
     3, 5;
     5, 6]
```

- Can store extra information in the struct
- Can store structs in data structure useful for search (lookup k-d trees)

- Matrix manipulation usually faster than struct manupulation

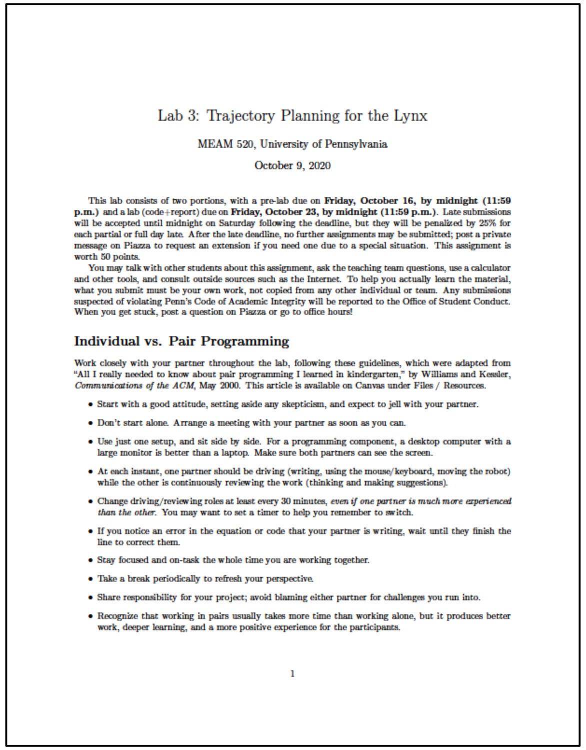
That brings us to the end of position/orientation analysis

Next time: Velocity



Chapter 4: Velocity Kinematics

- Read 4.intro – 4.4



Lab 3: Trajectory Planning for the Lynx

MEAM 520, University of Pennsylvania

October 9, 2020

This lab consists of two portions, with a pre-lab due on **Friday, October 16, by midnight (11:59 p.m.)** and a lab (code+report) due on **Friday, October 23, by midnight (11:59 p.m.)**. Late submissions will be accepted until midnight on Saturday following the deadline, but they will be penalized by 25% for each partial or full day late. After the late deadline, no further assignments may be submitted; post a private message on Piazza to request an extension if you need one due to a special situation. This assignment is worth 50 points.

You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. To help you actually learn the material, what you submit must be your own work, not copied from any other individual or team. Any submissions suspected of violating Penn's Code of Academic Integrity will be reported to the Office of Student Conduct. When you get stuck, post a question on Piazza or go to office hours!

Individual vs. Pair Programming

Work closely with your partner throughout the lab, following these guidelines, which were adapted from "All I really needed to know about pair programming I learned in kindergarten," by Williams and Kessler, *Communications of the ACM*, May 2000. This article is available on Canvas under Files / Resources.

- Start with a good attitude, setting aside any skepticism, and expect to jell with your partner.
- Don't start alone. Arrange a meeting with your partner as soon as you can.
- Use just one setup, and sit side by side. For a programming component, a desktop computer with a large monitor is better than a laptop. Make sure both partners can see the screen.
- At each instant, one partner should be driving (writing, using the mouse/keyboard, moving the robot) while the other is continuously reviewing the work (thinking and making suggestions).
- Change driving/reviewing roles at least every 30 minutes, *even if one partner is much more experienced than the other*. You may want to set a timer to help you remember to switch.
- If you notice an error in the equation or code that your partner is writing, wait until they finish the line to correct them.
- Stay focused and on-task the whole time you are working together.
- Take a break periodically to refresh your perspective.
- Share responsibility for your project; avoid blaming either partner for challenges you run into.
- Recognize that working in pairs usually takes more time than working alone, but it produces better work, deeper learning, and a more positive experience for the participants.

1

Lab 3: Planning due 10/23