MARYLAND ROBOTICS CENTER
THE INSTITUTE FOR SYSTEMS RESEARCH

ENPM661: Planning for Autonomous Robots

Splines Revisited, Graphs, Queues, Graph Search

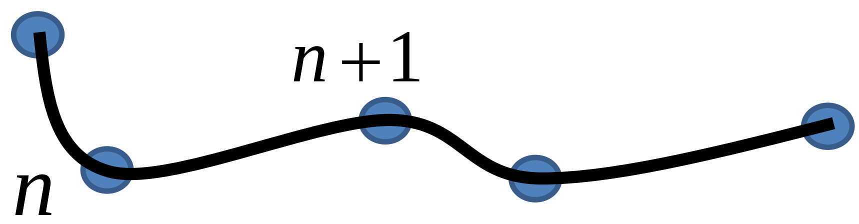
Dr. Michael Otte

Spring 2017

University of Maryland

Cubic Hermite splines, revisited

- Multi-segment curve $g(x)$
- Each segment is a 3rd degree polynomial
  - At endpoints of the n-th segment we choose:
    - Locations: p_n and p_{n+1}
    - Tangent vectors: d_n and d_{n+1}



Cubic Hermite splines, revisited

- Focus on a single segment
- In general we can use a parametric form

$$(x, y, z, \ldots) = f(s)$$

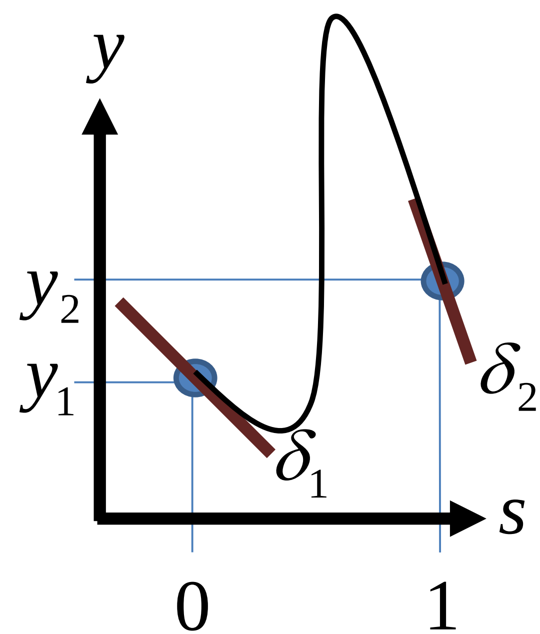
- Here we will just focus on the 1-D case

$$y = f(s)$$

- We assume s goes from 0 to 1
- We get to pick $y_1, y_2, \delta_1, \delta_2$

$$\delta_1 = \frac{dy_1}{ds} \qquad \delta_2 = \frac{dy_2}{ds}$$

$$y = f(s)$$



Cubic Hermite splines, revisited

- Want to find $y = f(s) = as^3 + bs^2 + cs + d$ that goes through y_1 and y_2 at δ_1 and δ_2 .
- Do this by solving:

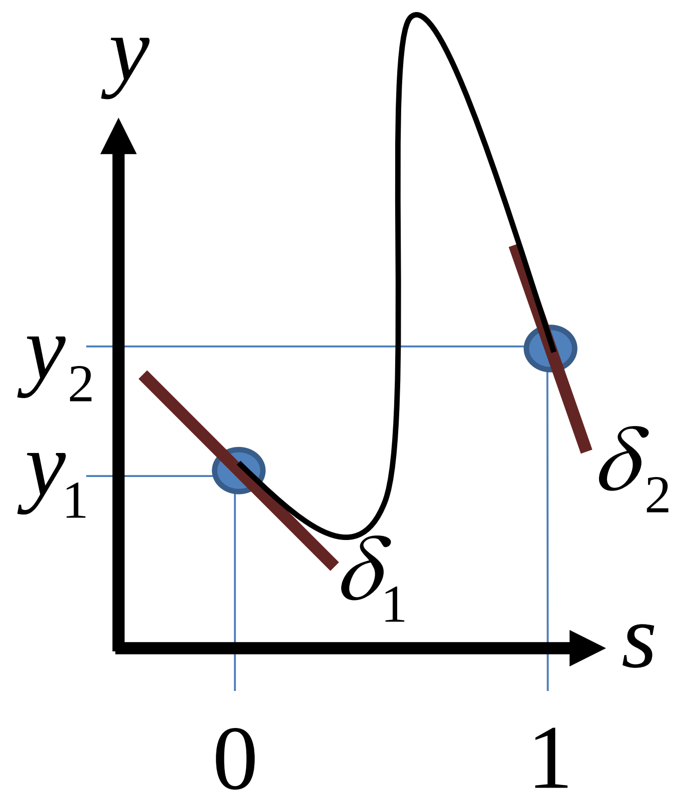
$$f(0) = y_1 = d$$

$$f(1) = y_2 = a + b + c + d$$

$$f'(0) = \delta_1 = c$$

$$f'(1) = \delta_2 = 3a + 2b + c$$

$$y = f(s) = as^3 + bs^2 + cs + d$$

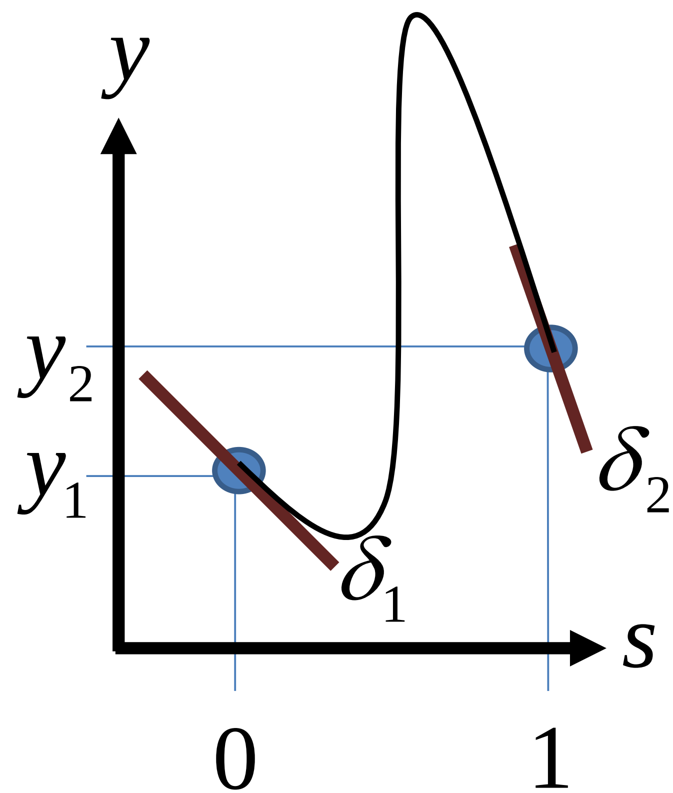


Cubic Hermite splines, revisited

- Want to find $y = f(s) = as^3 + bs^2 + cs + d$ that goes through y_1 and y_2 at δ_1 and δ_2 .
- Do this by solving:

$$f(0) = y_1 = d$$
$$f(1) = y_2 = a + b + c + d$$
$$f'(0) = \delta_1 = c$$
$$f'(1) = \delta_2 = 3a + 2b + c$$

$y = f(s) = as^3 + bs^2 + cs + d$



$$\mathrm{H} = \begin{bmatrix} y_1 \\ y_2 \\ \delta_1 \\ \delta_2 \end{bmatrix} \quad \mathrm{M} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \quad \mathrm{C} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

- Solve: H=MC for C

Cubic Hermite splines, revisited

- Want to find $y = f(s) = as^3 + bs^2 + cs + d$

$$\mathrm{M}^{-1}\mathrm{M} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Solve: H=MC for C

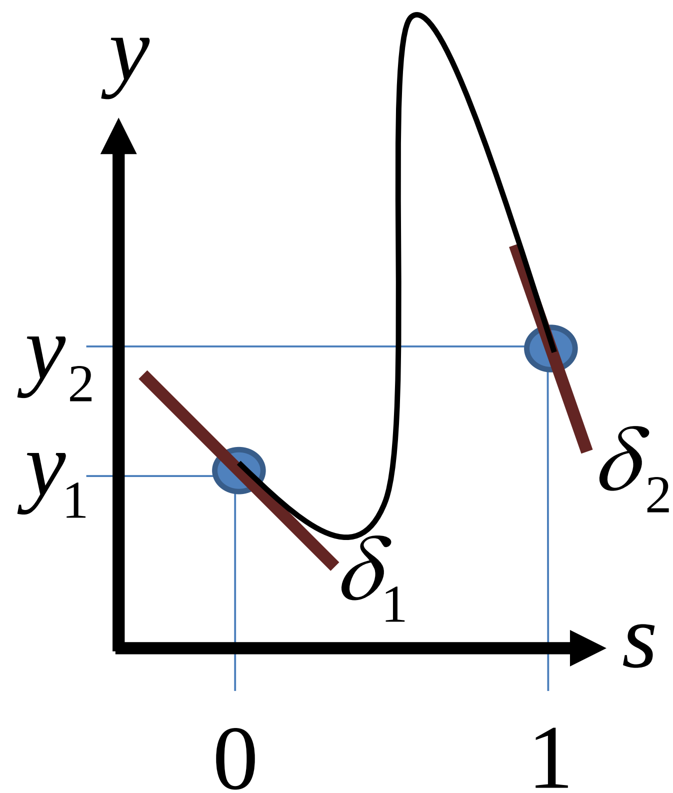
$$\mathrm{H} = \mathrm{MC}$$

$$\mathrm{M}^{-1}\mathrm{H} = \mathrm{M}^{-1}\mathrm{MC}$$

$$\mathrm{M}^{-1}\mathrm{H} = \mathrm{C}$$

$$\mathrm{C} = \mathrm{M}^{-1}\mathrm{H}$$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} y_1 \\ y_2 \\ \delta_1 \\ \delta_2 \end{bmatrix}$$



Switch to code

Cubic Hermite splines, revisited

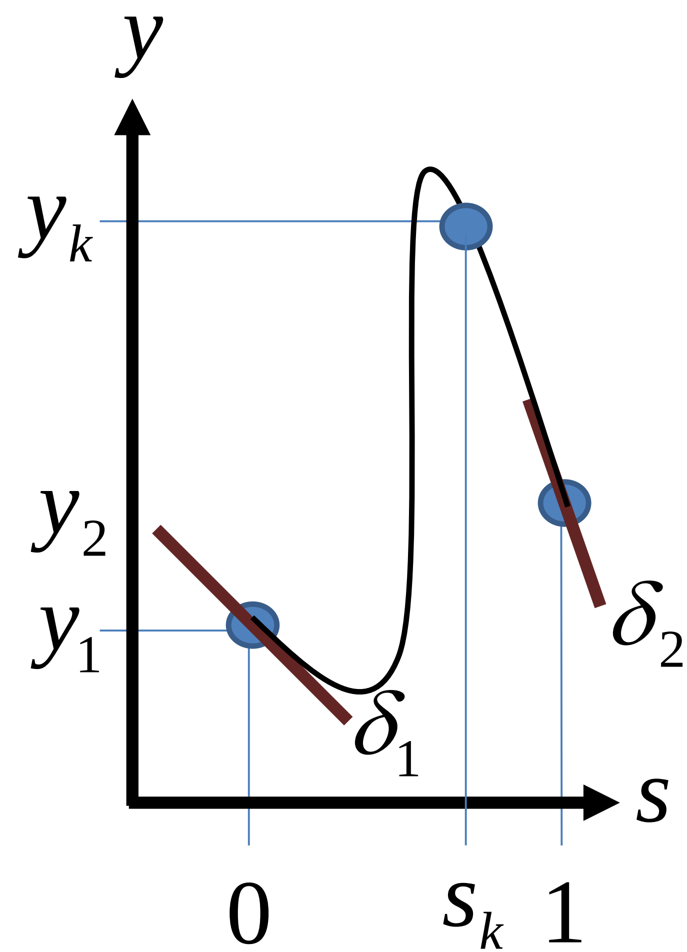
- If all we care about is a single value $y = f(s)$

$$S = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}$$

$$f(y) = SC$$

$$f(y) = SM^{-1}H$$

$$f(y) = [s^3 \quad s^2 \quad s \quad 1] * \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} y_1 \\ y_2 \\ \delta_1 \\ \delta_2 \end{bmatrix}$$



- Inefficient for more than 1 point
- Yes, S is a row vector
- (notation change vs. last class)

MARYLAND ROBOTICS CENTER
THE INSTITUTE FOR SYSTEMS RESEARCH

Switch to code

Cubic Hermite splines, revisited

- Basis function interpretation

$$f(s) = \mathrm{SC}$$

$$f(s) = \mathrm{C}^{\mathrm{T}}\mathrm{S}^{\mathrm{T}}$$

$$f(s) = \mathrm{C}^{\mathrm{T}}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\mathrm{S}^{\mathrm{T}}$$

$$f(s) = \mathrm{C}^{\mathrm{T}}\mathrm{M}^{\mathrm{T}}(\mathrm{M}^{-1})^{\mathrm{T}}\mathrm{S}^{\mathrm{T}}$$

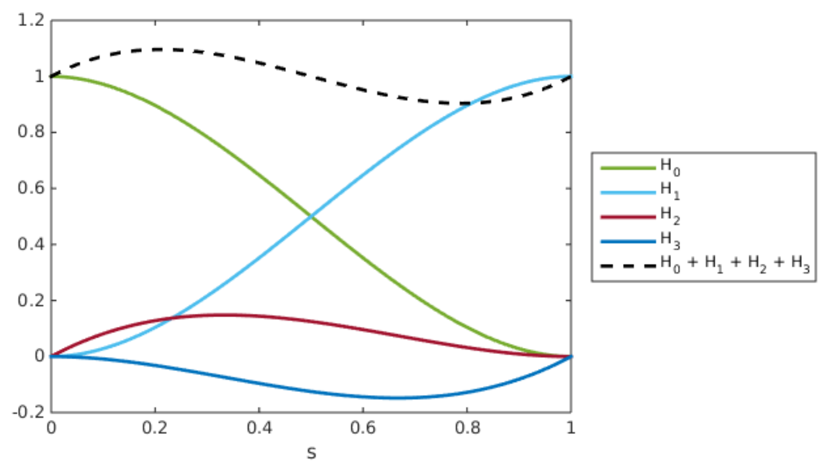
$$f(s) = (\mathrm{C}^{\mathrm{T}}\mathrm{M}^{\mathrm{T}})((\mathrm{M}^{-1})^{\mathrm{T}}\mathrm{S}^{\mathrm{T}})$$

$$f(s) = \mathrm{h}\,\mathbf{H}_{(s)}$$

$$\mathrm{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = \mathrm{C}^{\mathrm{T}}\mathrm{M}^{T} = \begin{bmatrix} a & b & c & d \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{H}_{(s)} = \begin{bmatrix} \mathrm{H}_1(s) \\ \mathrm{H}_2(s) \\ \mathrm{H}_3(s) \\ \mathrm{H}_4(s) \end{bmatrix} = (\mathrm{M}^{-1})^{\mathrm{T}}\mathrm{S}^{\mathrm{T}} = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} s^3 \\ s^2 \\ s \\ 1 \end{bmatrix}$$

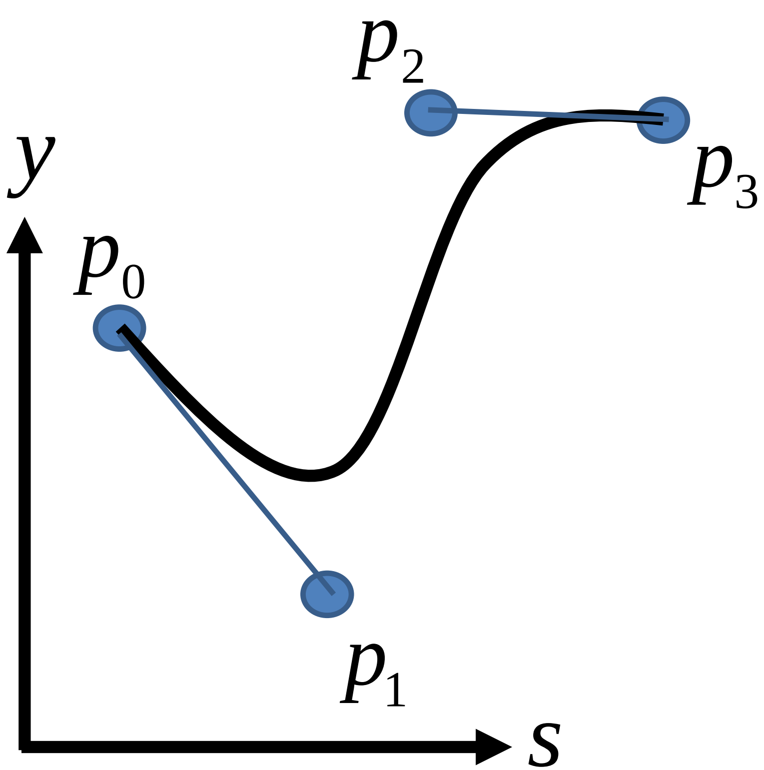
- $\mathbf{H}_{(s)}$ is given and we choose the weights h by picking $y_1, y_2, \delta_1, \delta_2$



Switch to code

Bezier Curves, revisited

- Bezier: Basis function interpretation
- Bezier Curves are similar, but we use a different set of basis functions.
  - We pick y_0, y_1, y_2, y_3 which determines the weights h on a different set of basis functions $\mathbf{H}_{(s)}$.



Basic Functions of Cubic Bezier Curve

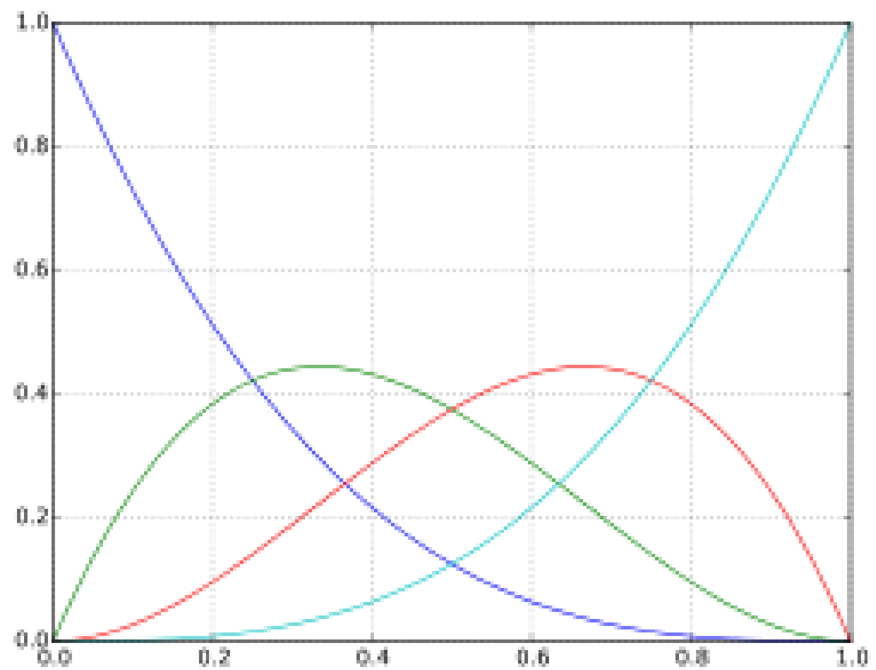
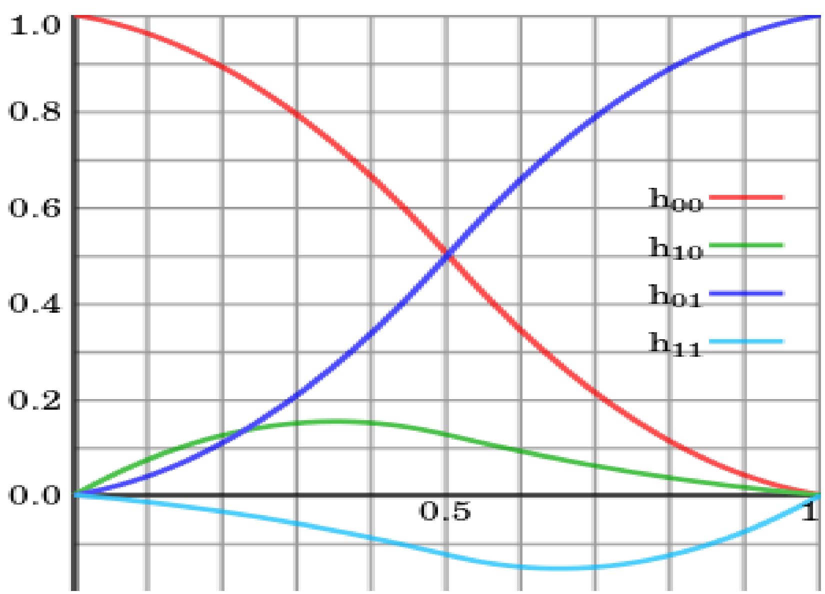
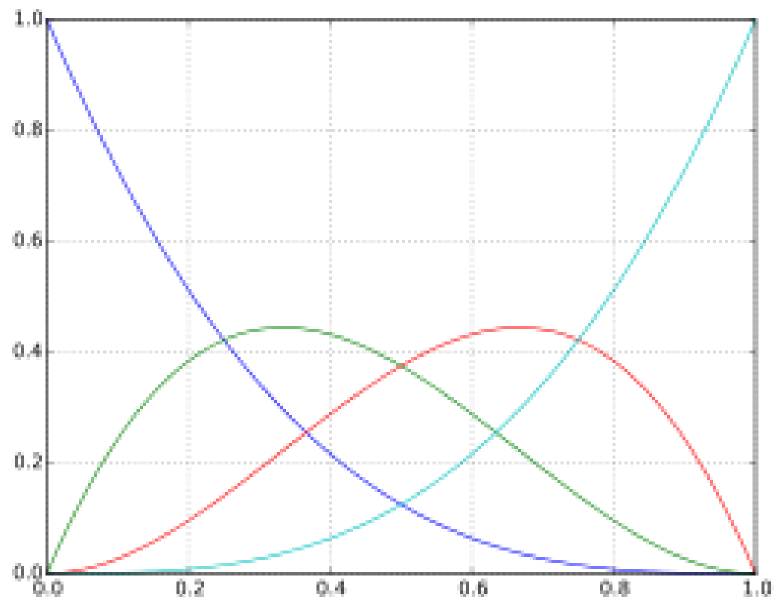


Image from Wikipedia

B-Splines Curves, revisited

- B-Splines: Basis function interpretation
- B-splines divide a curve into segments, such that the resulting spline has continuity to order *n*.
  - We choose *n* and {points, tangents, knots, etc.}.
  - There is always a way to interpret what is going on (for each segment) as choosing *n*+1 weights on a set of basis functions.





Images from Wikipedia

Assignments

Homework Assignment #H1

- Online on Campus.
- Due: February 20, 11:59PM (one week)
  - Upload to Campus
- Each student does their own assignment.
  - I don’t care if you talk to each other about the homework…
  - but everyone must do the assignment on their own.
- 5 Questions:
  1. Hermite Spline
  2. Dubins Curve
  3. Coordinate Frame Transforms
  4. Queues (covered shortly)
  5. Graph Search (covered shortly)

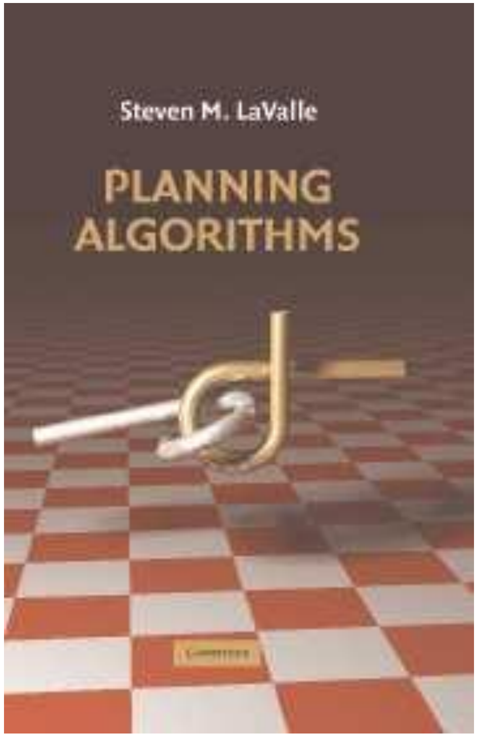
Project

- Progress?

- Who still is trying to figure out a topic?

On to new things….

Reading that Corresponds to today's lecture



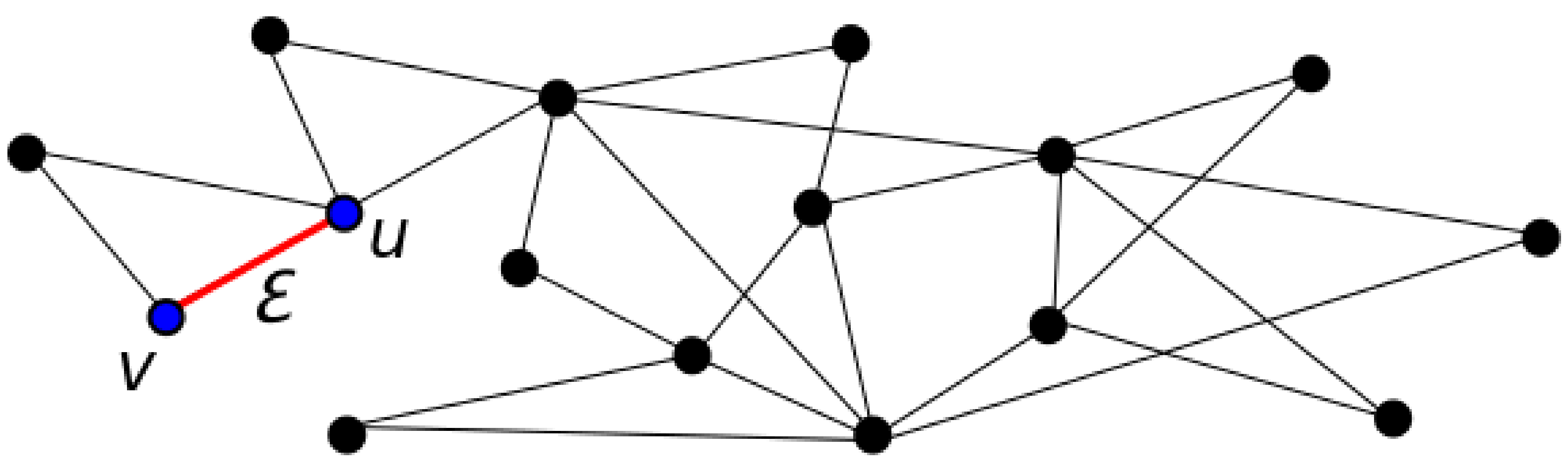
(chapters based on website version)

Chapter 2.0-2.2, Discrete Planning (first half)

Graphs

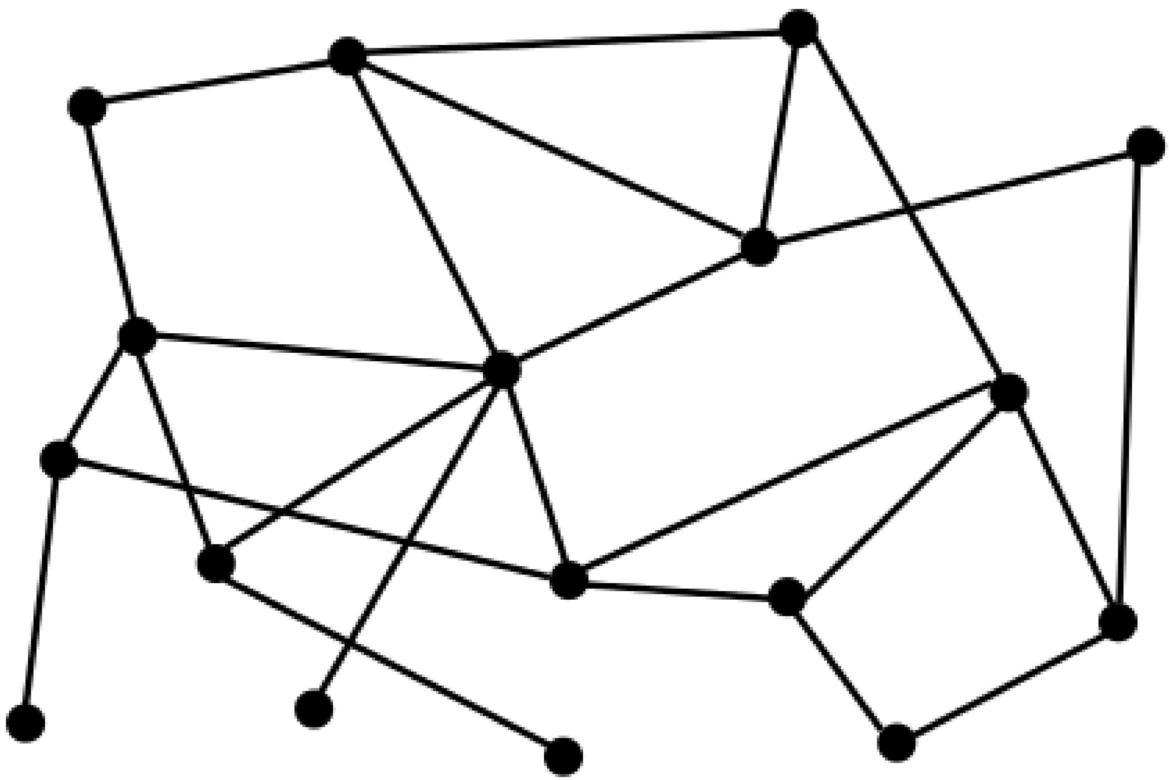
Graphs

- Node v
- Node set $\mathrm{V} = \bigcup_i \{v_i\}$
- Edge $\varepsilon = (v, u)$ where $v, u \in \mathrm{V}$
- Edge set $\mathrm{E} = \bigcup_k \{\varepsilon_k\}$
- Graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$



Undirected Graphs

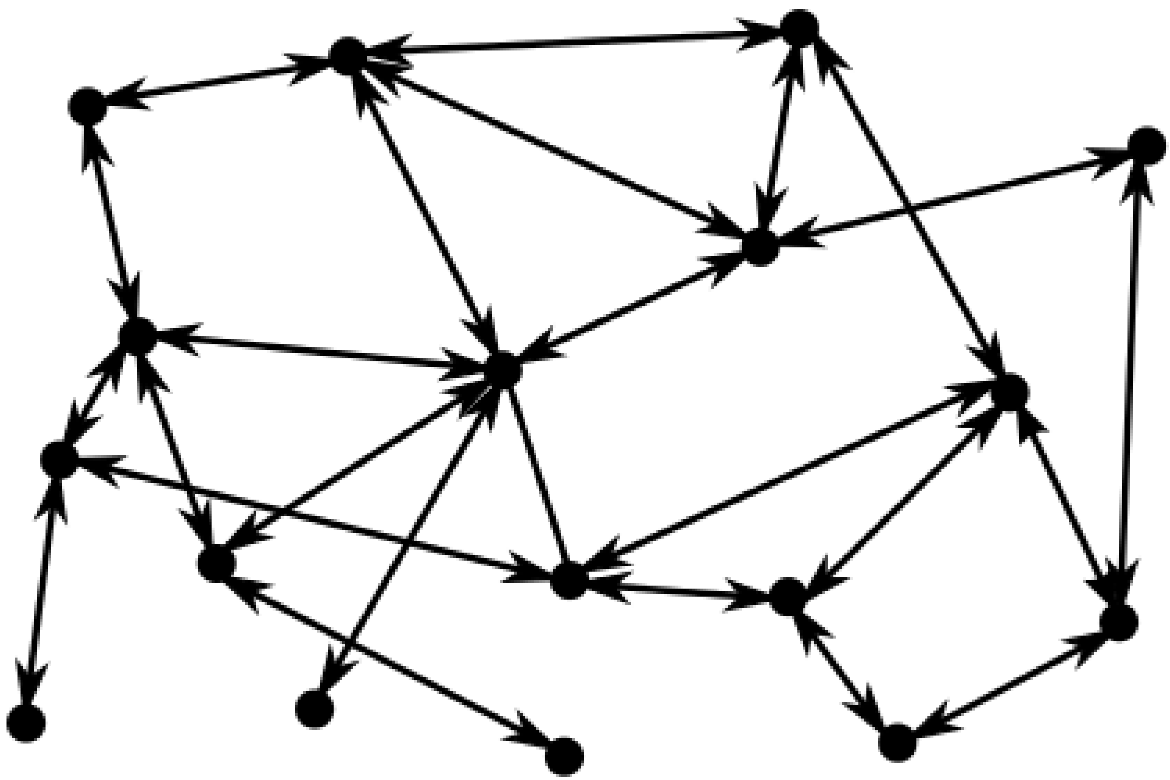
- It is possible to go either way along an edge
- Or, alternatively, $(v,u) \in \mathrm{E} \Rightarrow (u,v) \in \mathrm{E}$



MARYLAND ROBOTICS CENTER
THE INSTITUTE FOR SYSTEMS RESEARCH

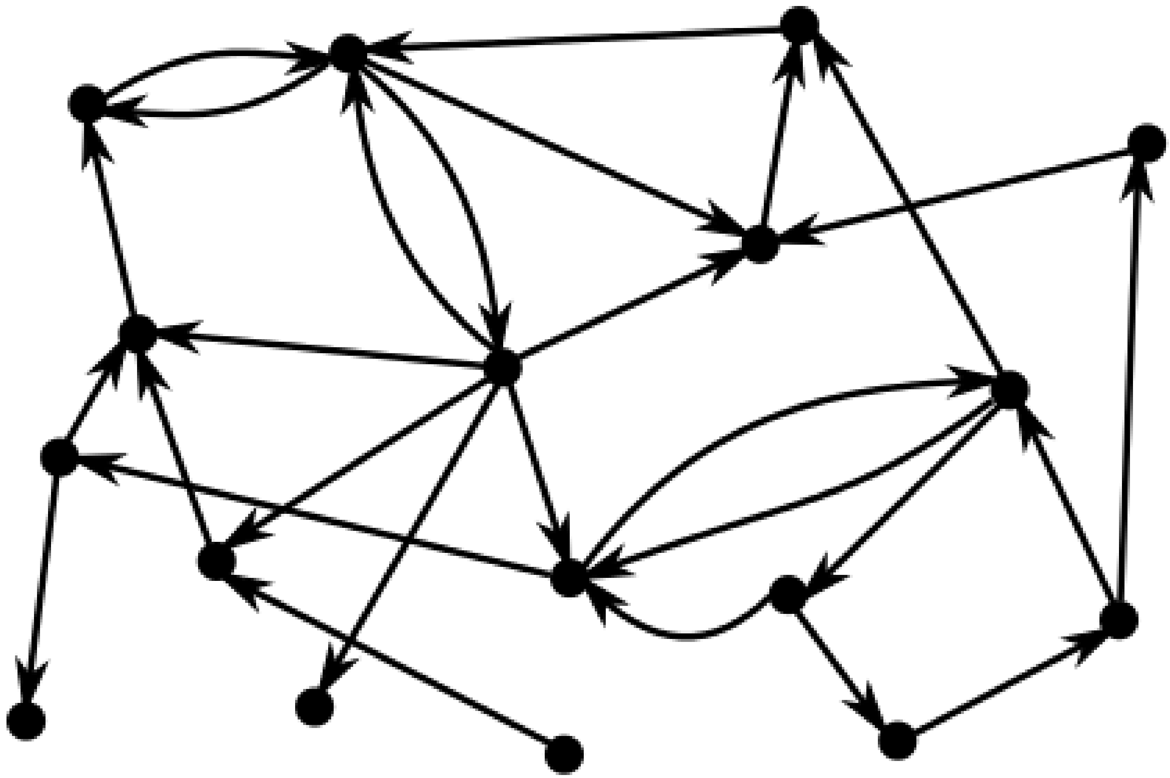
Undirected Graphs

- It is possible to go either way along an edge
- Or, alternatively, $(v,u) \in \mathrm{E} \Rightarrow (u,v) \in \mathrm{E}$



Directed Graphs

- Edges only go one way
- Or, alternatively, $(v,u) \in \mathrm{E} \not\Rightarrow (u,v) \in \mathrm{E}$

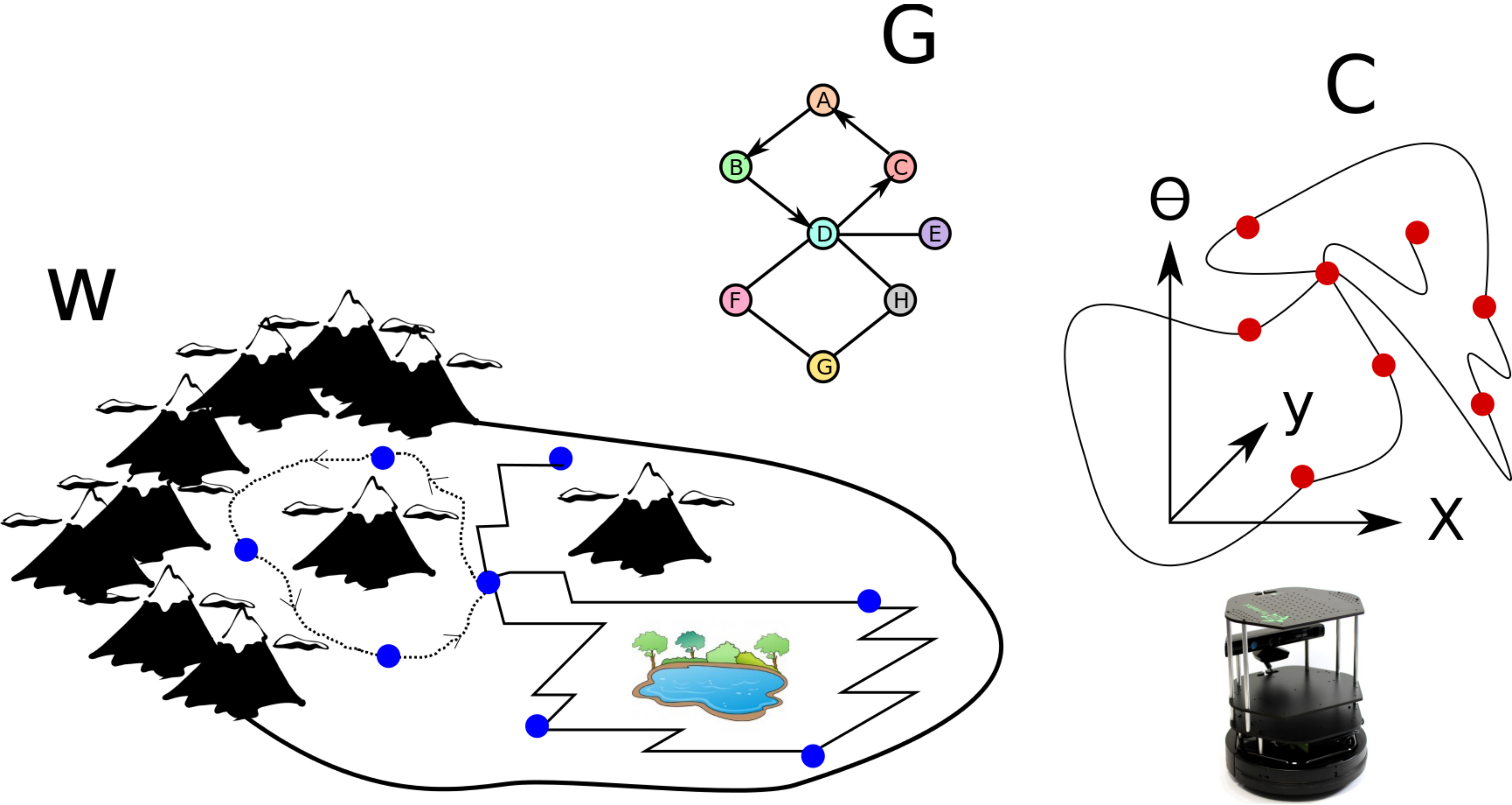


My soapbox about using graphs for motion planning

- Graphs are a theoretical tool.
- Edges represent trajectories (or control policies) that the robot can follow.
- Edges are NOT the same thing as trajectories.
  - **Edge** are a **theoretical construct** for reasoning about whether or not we can get from *u* to *v.*
  - **Trajectories** are specific **geometric curves** through the configuration space from one configuration X to another Y.
- Nodes are NOT configurations.
  - Nodes u,v ***represent*** configurations X,Y in theory and code.

My soapbox about using graphs for motion planning

- Graphs are a theoretical tool.
- Edges *represent* trajectories.
- Nodes *represent* configurations.



My soapbox about using graphs for motion planning

- That said…
- We often draw trajectories and talk about edges…
  - Edges are purely theoretical we might as well look at the trajectories they represent to build intuition.
- In fact, we often project trajectories to the workspace for visualization (even though trajectories exist in the configuration space).
- Always remember:
  - Planning happens in the configuration space.
  - Edges are not trajectories (but represent them).

Data Structures

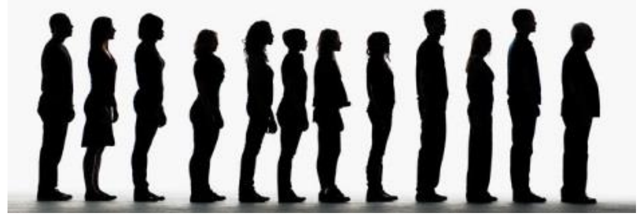
MARYLAND ROBOTICS CENTER
THE INSTITUTE FOR SYSTEMS RESEARCH

Data Structures

- Low-level
  - int, float, double, char, etc.
- Mid-level
  - array, vector, matrix
  - linked-list, double linked list.
- High level
  - Queue, Stack, Heap
  - Tree, Graph

MARYLAND ROBOTICS CENTER
THE INSTITUTE FOR SYSTEMS RESEARCH

Queue



- Technically a “First-In-First-Out Queue” or FIFO-Queue
- Q — Queue
- Q.INSERT(v) — adds node v to the “back”
  - Q.PUSH_BACK()
- Q.TOP() — returns the node at the “front”
  - Q.FRONT()
- Q.POP() — returns the node at the “front” and removes it from Q
  - Q.POP_FRONT()

MARYLAND ROBOTICS CENTER
THE INSTITUTE FOR SYSTEMS RESEARCH

Queue

- Technically a “First-In-First-Out Queue” or FIFO-Queue
- What happens:

Q.INSERT(A)
Q.INSERT(B)
Q.TOP()
Q.INSERT(C)
Q.POP()
Q.INSERT(D)
Q.POP()
Q.TOP()
Q.POP()
Q.INSERT(E)

Q.INSERT(A)
Q.POP()
Q.POP()

Stack

- Technically a “Last-In-First-Out Queue” or LIFO-Queue
- Q — Stack (LIFO-Queue)
- Q.INSERT(v) — adds node v to the “top”
  - Q.PUSH_FRONT()
- Q.TOP() — returns the node at “top”
  - Q.FRONT()
- Q.POP() — returns the node at “top” and removes it from Q
  - Q.POP_FRONT()

Image from “Art for daily life”

Stack

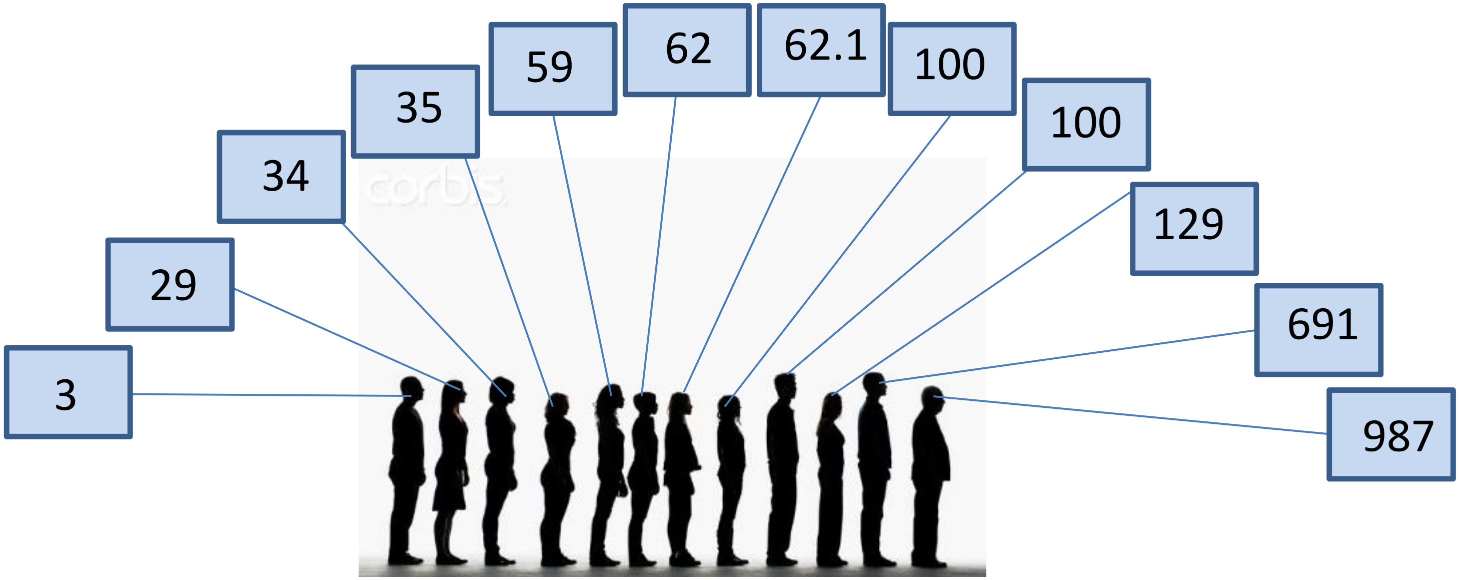
- Technically a "Last-In-First-Out Queue" or FIFO-Queue
- What happens:

```
Q.INSERT(A)
Q.INSERT(B)
Q.TOP()
Q.INSERT(C)
Q.POP()
Q.INSERT(D)
Q.POP()
Q.TOP()
Q.POP()
Q.INSERT(E)
```

```
Q.INSERT(A)
Q.POP()
Q.POP()
```

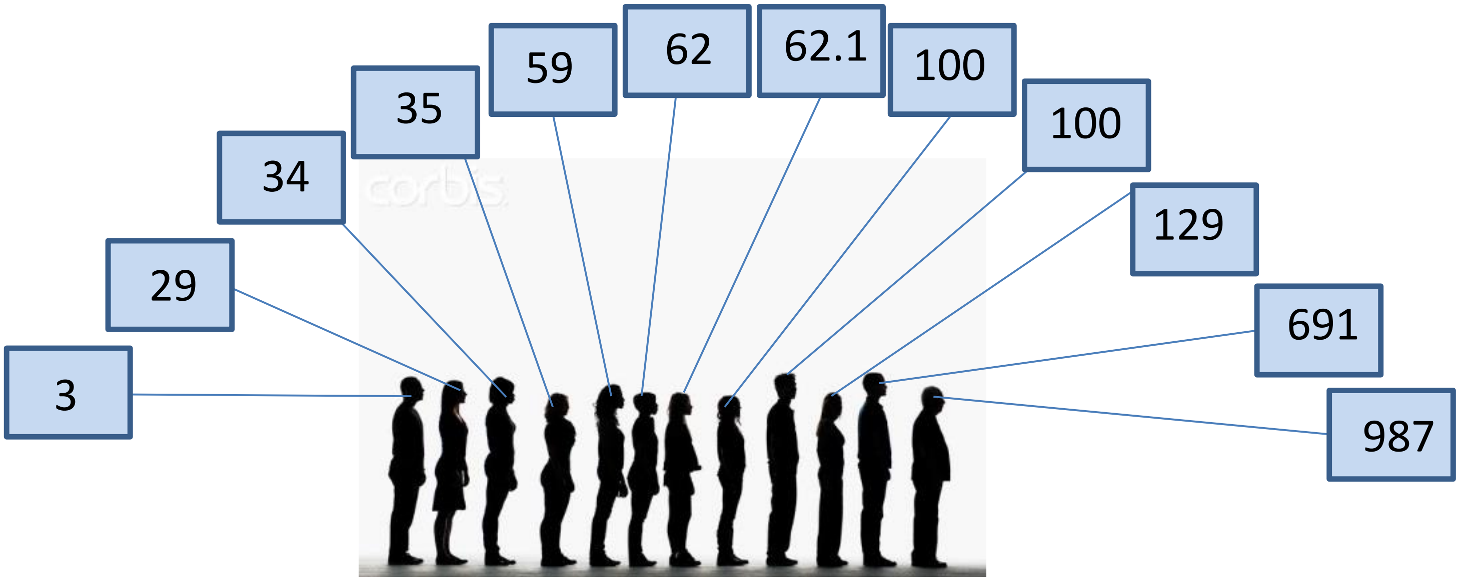
Image from "Art for daily life"

Priority Queue (Priority Heap)



- Nodes are sorted by key values, v.KEY
  - Min-Priority Queue vs. Max-Priority Queue
- Q.INSERT(v) adds node v where it goes
- Q.TOP() returns node with min/max key
- Q.POP() returns & removes the node at "top"
- Q.REMOVE(v) returns & removes node v
- Q.UPDATE(v) updates position of v (after key change)

Priority Queue (Priority Heap)



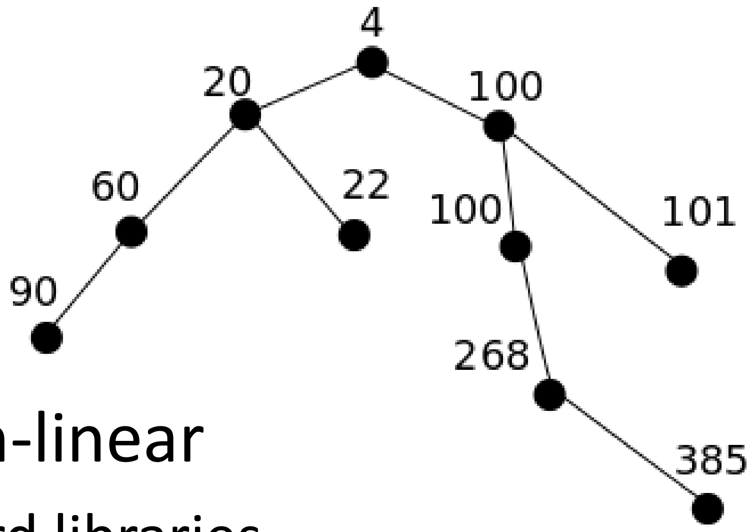
- Assume max-priority queue:

```
A.KEY = 5;     Q.INSERT(A);
B.KEY = 3;     Q.INSERT(B);
C.KEY = 10;    Q.INSERT(C);
B.KEY = 7;     Q.UPDATE(B);
Q.TOP();
A.KEY = 12;    Q.UPDATE(A);
```

```
Q.REMOVE(C);
Q.POP();
D.KEY = 4;     Q.INSERT(D);
B.KEY = 2;     Q.UPDATE(B);
Q.REMOVE(TOP());
Q.POP();
```

Priority Queue (Priority Heap)

- Why the name Heap?



- Fast data structures are non-linear
  - A good reason to use standard libraries….

	Binary	Binomial	Fibonacci	Brodal
TOP()	O(1)	O(log n)	O(1)	O(1)
POP()	O(log n)	O(log n)	O(log n)	O(log n)
UPDATE(v)	O(log n)	O(log n)	O(1)	O(1)
INSERT(v)	O(log n)	O(1)	O(1)	O(1)
REMOVE(v)	O(log n)	O(log n)	O(log n)	O(log n)

Graph Search

Simple Forward Graph Search Algorithms

- Breadth-First Search
  - Queue (FIFO) as subroutine
- Depth First Search
  - Stack (LIFO) as subroutine
- Best-First Search
  - Priority Queue as subroutine
    - Dijkstra’s Algorithm
  - Heuristics can make “best” more accurate
    - A* Algorithm

Forward Graph Search: Preliminaries

- "start" node in V
- "goal" node in V
  - Sometimes we have multiple goals (don't care which we get)
- Each node maintains a list of is neighbors
  - v.Neighbors = $\bigcup \{u \mid (v,u) \in \mathrm{E}\}$
  - array or a list in practice
- v.NextNeighbor()
  - "iterator" function over v.Neighbors
  - each time we call this we get a new neighbor of v
  - after returning each neighbor once, it always returns Ø.
- v.parent
  - "parent pointer" that remembers search discovery order
  - the set of all parent pointers defines a **search-tree**!

Forward Graph Search: Preliminaries

- UNVISITED
  - a set that initially contains all nodes
  - nodes are removed when they are visited
- Algorithmic notation: "←" vs. "="
  - "←" assignment opperator
    - A←B
    - sets the value of A to be the same as that in B
  - "=" boolean function that checks for equality
    - A=B
    - returns **true** if a and b have the same value
    - returns **false** if a and be have different values

Simple Forward Graph Search: Psudocode

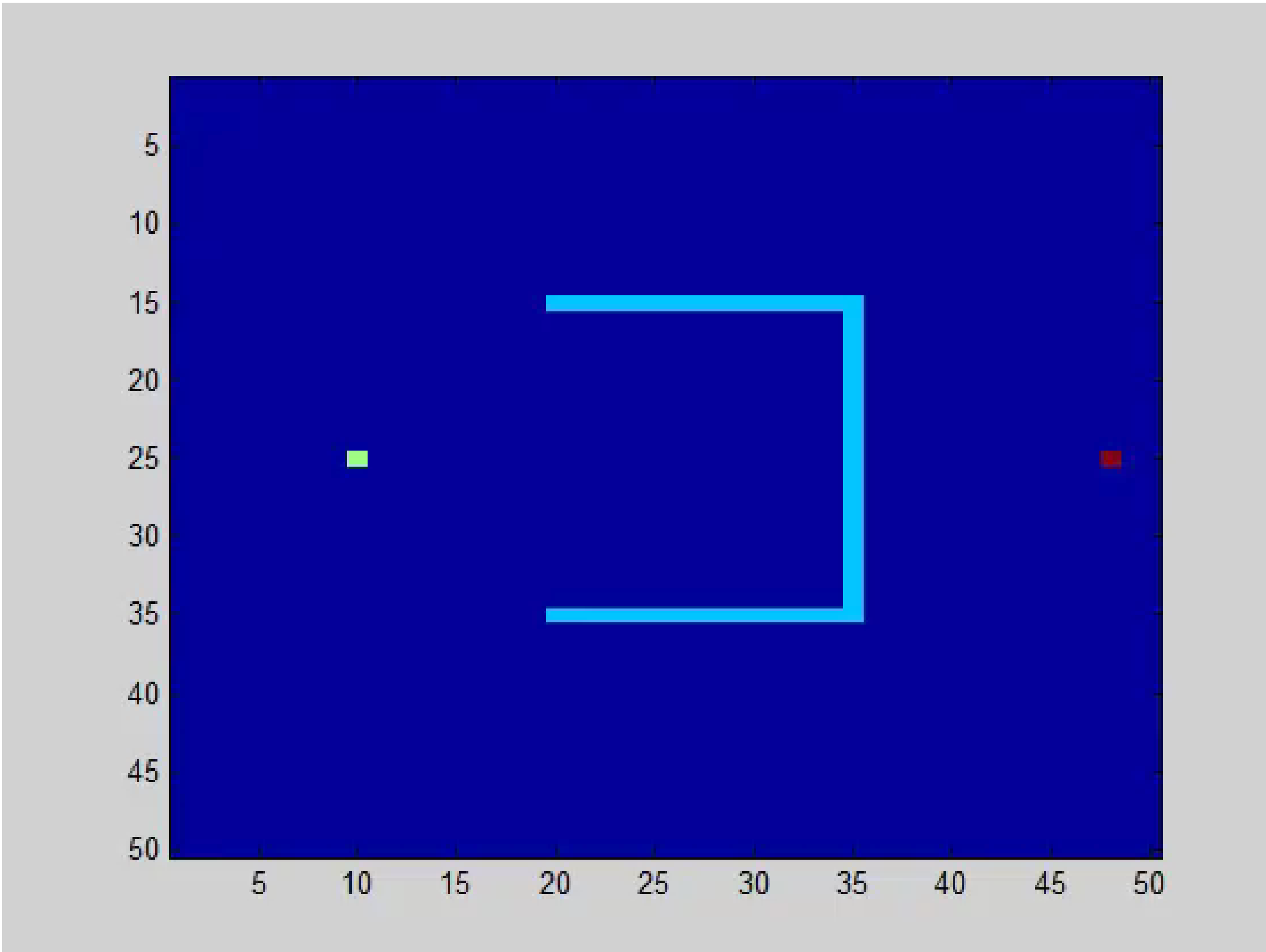
```
Function GraphSearch(G,start, goal, Q)
UNVISITED ←V\{start}
Q.INSERT(start)
while Q.TOP() ≠ Ø
    v ← Q.TOP()
    u ← v.NextNeighbor()
    if u = Ø
        Q.POP()
    else if u∈INVISITED
        UNVISITED ← UNVISITED \ {u}
        u.parent ← v
        Q.INSERT(u)
    if u = goal
        return SUCCESS
return FAILURE
```

Breadth-First Search (BFS)

- Q is a FIFO-Queue
- The **search tree** (of parent pointers) goes as wide as possible before going deeper.
- Example on board ….

MARYLAND ROBOTICS CENTER
THE INSTITUTE FOR SYSTEMS RESEARCH

Breadth First Search (BFS): ANIMATION



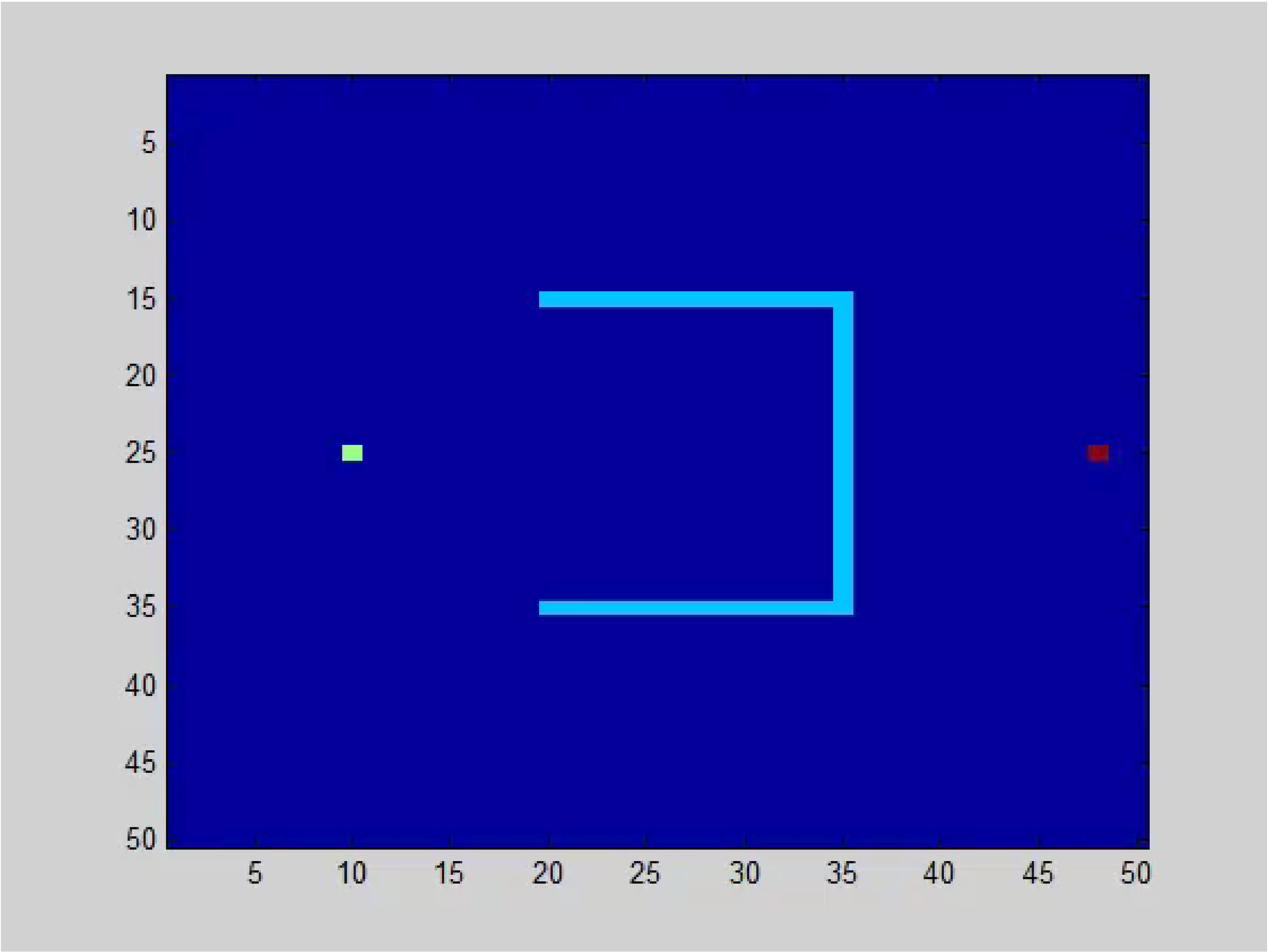
(Courtesy: Petr Svec)

Slide by S.K. Gupta

Depth-First Search (DFS)

- Q is a Stack (LIFO-Queue)
- The **search tree** (of parent pointers) goes as deep as possible before going deeper.
- Example on board....

Depth First Search (DFS): ANIMATION



(Courtesy: Petr Svec)

Best-First Search (BFS): More Preliminaries

- Q is a Priority Queue
- Only useful if we have some notion of "best"
  - Distance and cost are widely used in path/motion planning
  - "distance" e.g., Euclidian distance along trajectories
  - "cost" e.g., fuel required to go along trajectories
  - etc.
- Cost(v,u)
  - returns the cost of moving
    - from: the configuration space point represented by v
    - to: the configuration space point represented by u
  - Note, sometimes for (u,v) $\notin$ V we define Cost(u,v) = ∞

Best-First Search (BFS): More Preliminaries

- UNVISITED, OPEN, CLOSED
  - Sometimes this is UNVISITED, LIVE, DEAD
- UNVISITED
  - Node is not in search tree
  - (has not been “touched” yet)
- OPEN
  - Node is on the fringe of the search tree
  - It is currently in the priority queue
- CLOSED
  - Node is internal to the search tree
  - Node has been removed from the priority queue

Dijkstra's Psudocode

```
Function DijkstraAlgorithm(G,start, goal, Q)
    UNVISITED ←V\{start}
    Q.INSERT(start)
    while Q.TOP() ≠ Ø                                  // and goal ∉ CLOSED
        v ← Q.TOP()
        u ← v.NextNeighbor()
        if u = Ø
            Q.POP()                                    //;  OPEN ← OPEN \ {v}
                                                       //;  CLOSED ← CLOSED ∪ {v}
        else if  u∈Unvisited or  u.costToStart  > v.costToStart + Cost(v,u)
            UNVISITED ← UNVISITED\{u}                  //; OPEN← OPEN ∪ {u}
            u.parent ← v
            u.costToStart =v.costToStart + Cost(v,u)
            Q.INSERT(u)
        if u = goal
            return SUCCESS
    return FAILURE
```

Best-First Search: Dijkstra’s

- Example on board….

Best-First Search: Dijkstra's

- So… hmm…. What's the difference between
- Depth-First Search and Dijkstra's algorithm?

Bibliography for this Lecture

- LaValle, S. M. Planning Algorithms, 2006