# Shortest Path and Informed Search

Brian C. Williams
16.410 - 13
October 27th, 2003

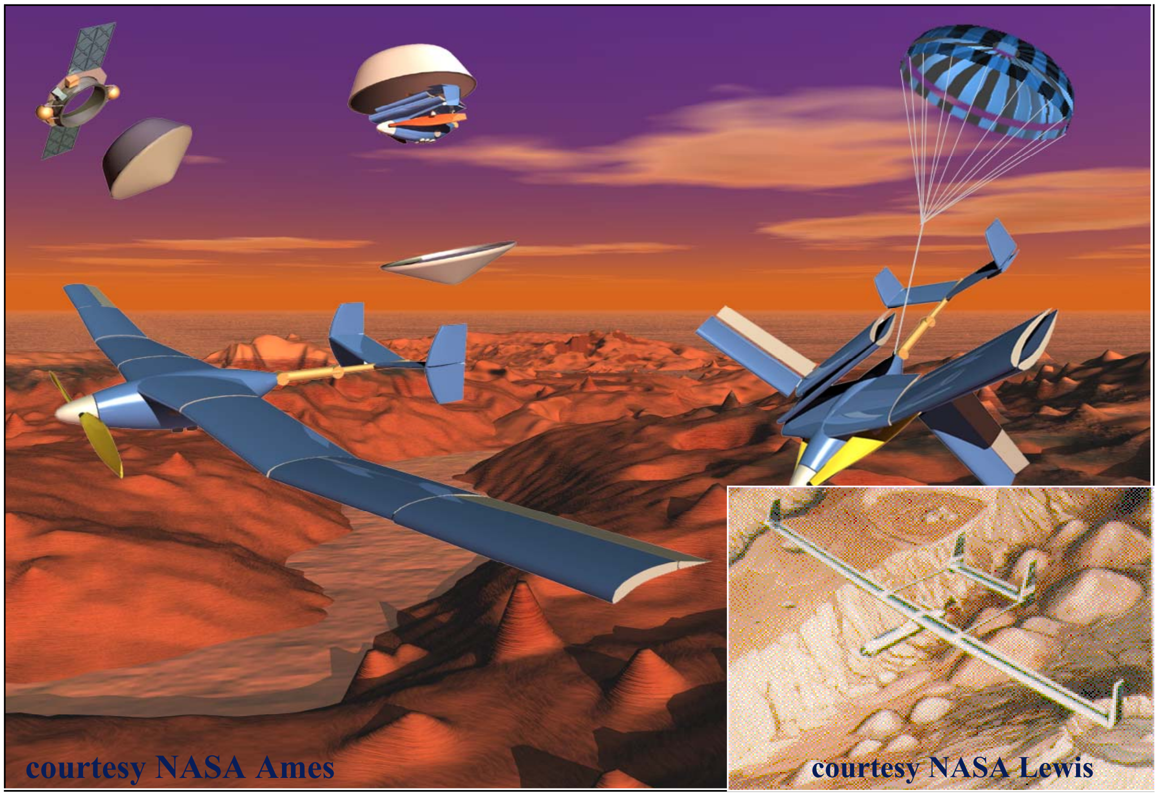
Slides adapted from:
6.034 Tomas Lozano Perez,
Winston, and Russell and Norvig AIMA



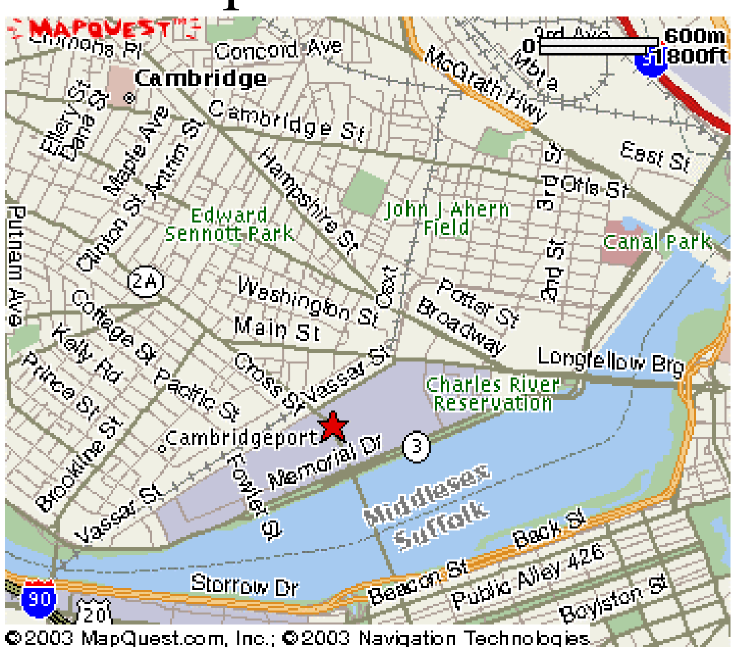
# Assignment

- Reading:
  - Shortest path:
    Cormen Leiserson & Rivest,
    "Introduction to Algorithms" Ch. 25.1-.2
  - Informed search and exploration:
    AIMA Ch. 4.1-2
- Homework:
  - Online problem set #6 due Monday November 3rd.

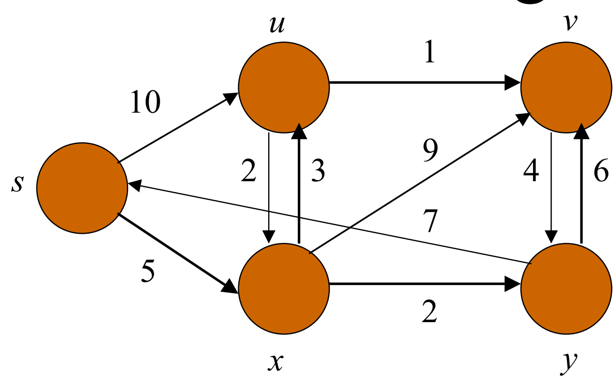
# How do we maneuver?




# Roadmaps are an effective state space abstraction



# Weighted Graphs and Path Lengths

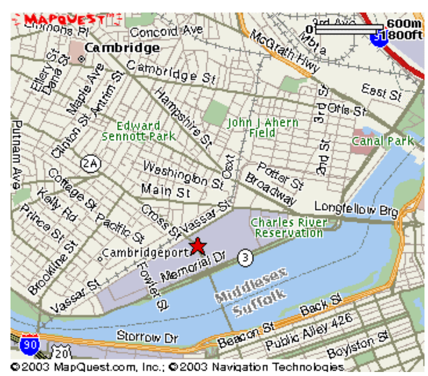



| | |
|---|---|
| Graph | G = <V, E> |
| Weight function | $w$: E $\rightarrow \Re$ |
| Path | p = < $v_o$, $v_1$, … $v_k$> |
| Path weight | $w$(p) = $\Sigma$ w($v_{i-1}$,$v_i$) |
| Shortest path weight | $\delta$(u,v) = min {$w$(p) : $u \rightarrow^{p} v$ } else $\infty$ |

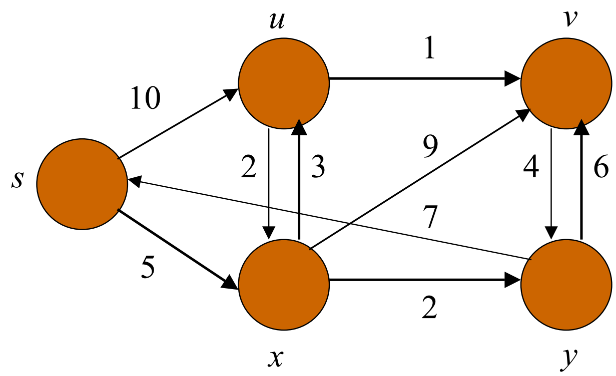


# Outline

- Creating road maps for path planning
- Exploring roadmaps: Shortest Paths
  - Single Source
    - Dijkstra;s algorithm
  - Informed search
    - Uniform cost search
    - Greedy search
    - A* search
    - Beam search
    - Hill climbing
- Avoiding adversaries
  - (Next Lecture)



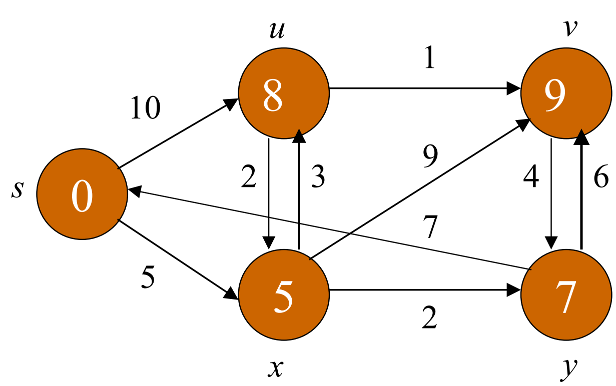

# Single Source Shortest Path




Problem: Compute shortest path to all vertices from source s



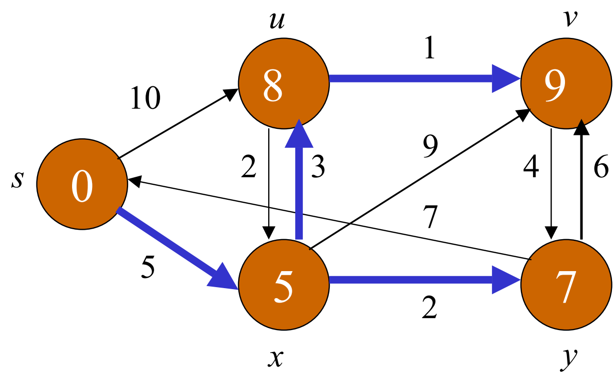
# Single Source Shortest Path




Problem: Compute shortest path to all vertices from source s

- estimate d[v] estimated shortest path length from s to v

# Single Source Shortest Path

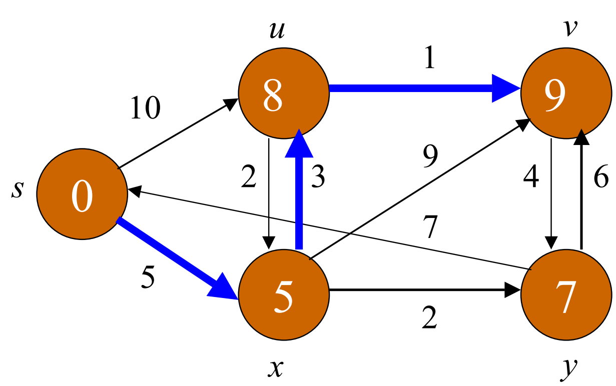



Problem: Compute shortest path to all vertices from source s

- estimate d[v] estimated shortest path length from s to v
- **predecessor π[v]** final edge of shortest path to v
  - induces shortest path tree

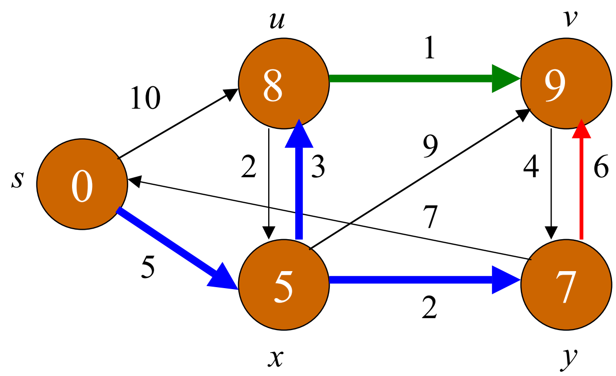


# Properties of Shortest Path




- Subpaths of shortest paths are shortest paths.
  - s $\rightarrow^p$ v = <s, x, u, v>
  - s →p u = <s, x, u>
  - s →p x = <s, x>
  - x →p v = <x, u, v>
  - x →p v = <x, u>
  - u →p v = <u, v>

# Properties of Shortest Path

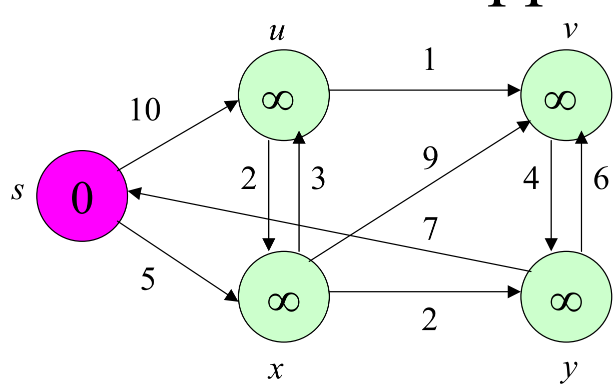



Corollary: Shortest paths are grown from shortest paths.

- The length of shortest path $s \rightarrow^{p} u \rightarrow v$
  is $\delta(s,v) = \boldsymbol{\delta(s,u)} + \boldsymbol{w(u,v)}$.
- $\forall$ **<*u,v*> $\in$ *E* $\delta$(*s,v*) $\leq$ $\delta$(*s,u*) + *w*(*u,v*)**



# Idea: Start With Upper Bound




Initialize-Single-Source*(G, s)*

**1.** **for** each vertex $v \in V[G]$

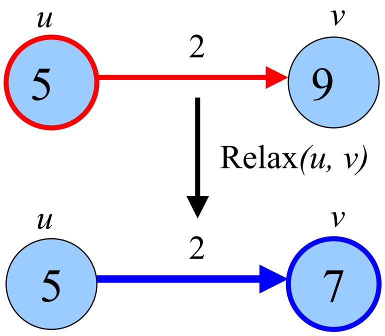
**2.** **do** $d[u] \leftarrow \infty$

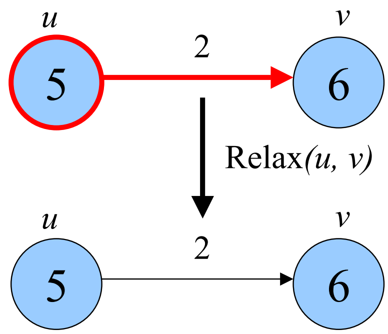
3. $\pi[v] \leftarrow NIL$

*4.* $d[s] \leftarrow 0$

**O(v)**

# Relax Bounds to Shortest Path




Relax *(u, v, w)*

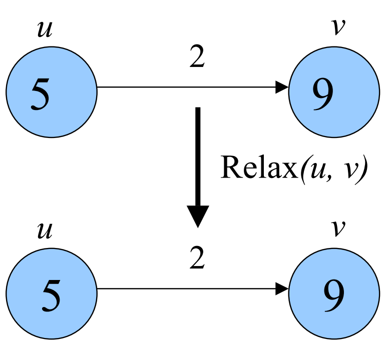
**1.** **if** $d[u] + \mathrm{w(u,v)} < d[v]$

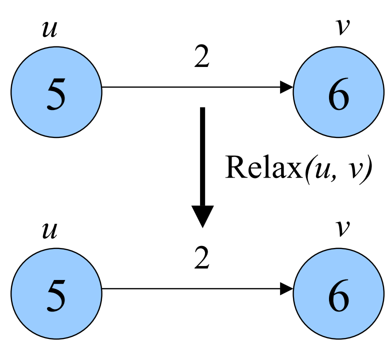
**2.** **do** $d[v] \leftarrow d[u] + \mathrm{w(u,v)}$

3. $\pi[v] \leftarrow u$



# Properties of Relaxing Bounds




After calling Relax*(u, v, w)*

- $d[u] + \mathrm{w(u,v)} \geq d[v]$

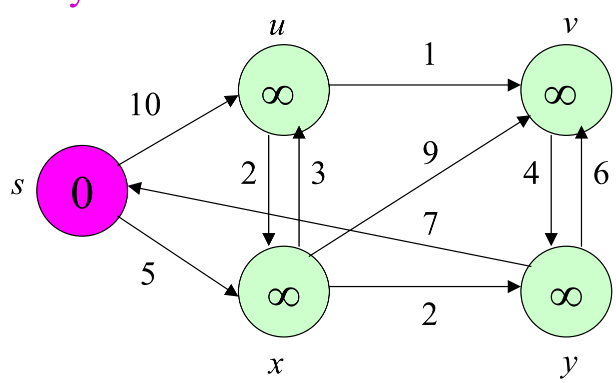
d remains a shortest path upperbound after repeated calls to Relax.

*Once* $d[v]$ is the shortest path its value persists

# Dijkstra's Algorithm

Assume all edges are non-negative.

Idea: Greedily relax out arcs of minimum cost nodes




**Q = {s,u,v,x,y}** Vertices to relax

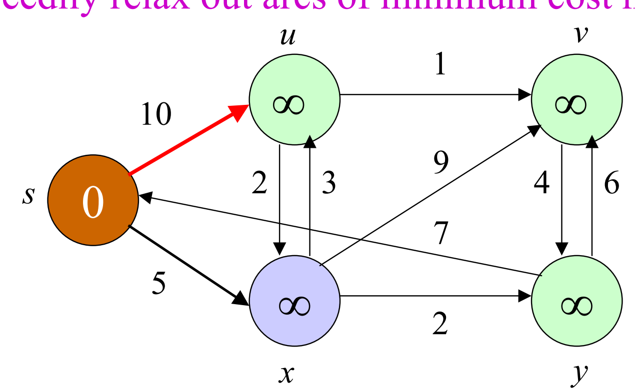
**S = {}** **Vertices with shortest path value**



# Dijkstra's Algorithm

Assume all edges are non-negative.

Idea: Greedily relax out arcs of minimum cost nodes



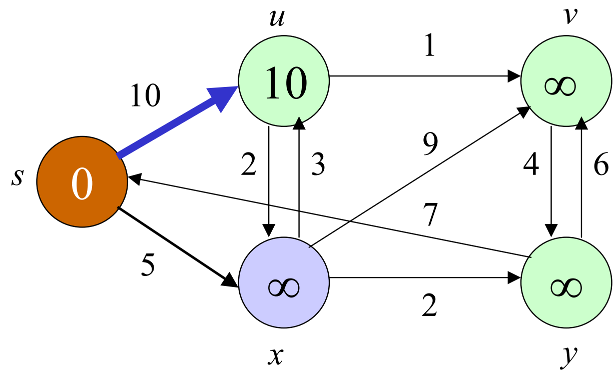

**Q = {x,y,u,v}** Vertices to relax

**S = {s}** **Vertices with shortest path value**

# Dijkstra's Algorithm

Assume all edges are non-negative.

Idea: Greedily relax out arcs of minimum cost nodes




**Q = {x,y,u,v}** Vertices to relax

**S = {s}** **Vertices with shortest path value**

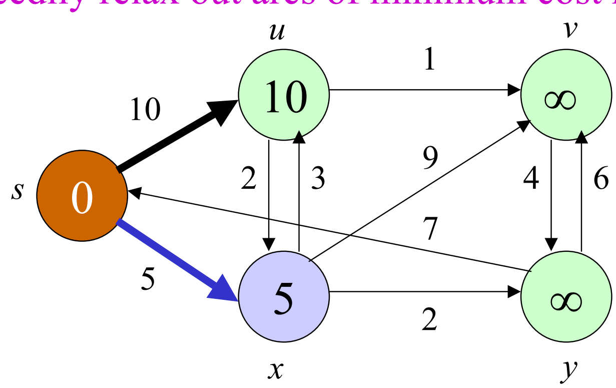
**Shortest path edge Π[v] = arrows**



# Dijkstra's Algorithm

Assume all edges are non-negative.

Idea: Greedily relax out arcs of minimum cost nodes




**Q = {x,y,u,v}** Vertices to relax

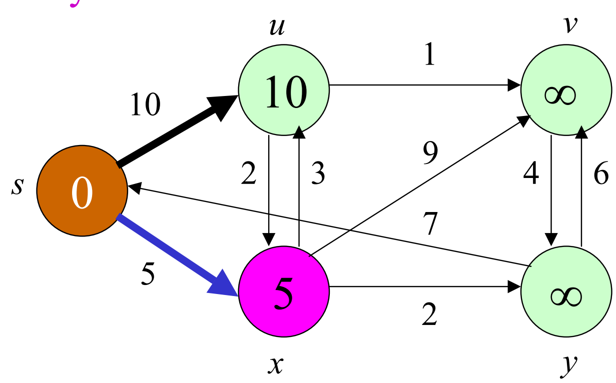
**S = {s}** **Vertices with shortest path value**

**Shortest path edge Π[v] = arrows**

# Dijkstra's Algorithm

Assume all edges are non-negative.

Idea: Greedily relax out arcs of minimum cost nodes




**Q = {x,y,u,v}** Vertices to relax

**S = {s}** **Vertices with shortest path value**

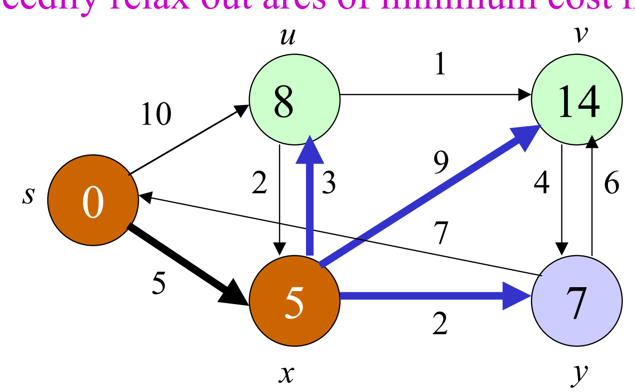
**Shortest path edge Π[v] = arrows**



# Dijkstra's Algorithm

Assume all edges are non-negative.

Idea: Greedily relax out arcs of minimum cost nodes




**Q = {y,u,v}** Vertices to relax

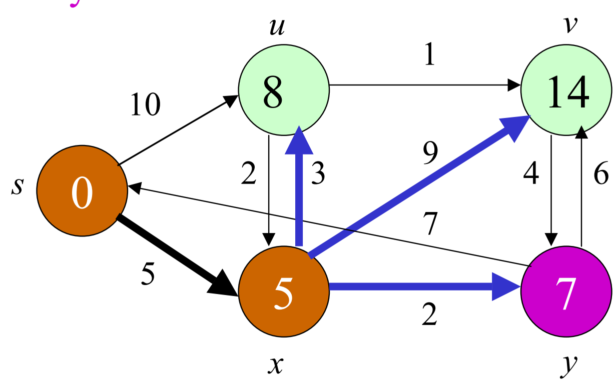
**S = {s, x}** **Vertices with shortest path value**

**Shortest path edge Π[v] = arrows**

# Dijkstra's Algorithm

Assume all edges are non-negative.

Idea: Greedily relax out arcs of minimum cost nodes




**Q = {y,u,v}** — Vertices to relax

**S = {s, x}** — **Vertices with shortest path value**

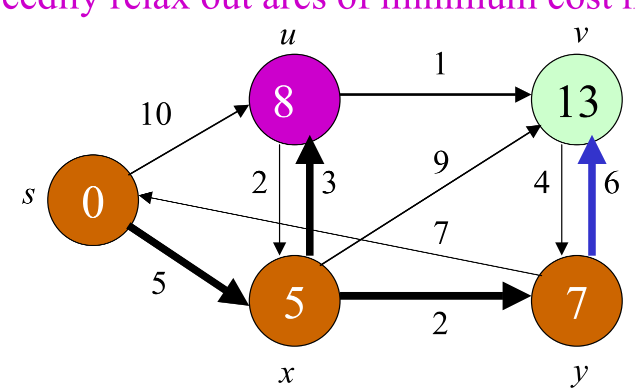
**Shortest path edge Π[v] = arrows**



# Dijkstra's Algorithm

Assume all edges are non-negative.

Idea: Greedily relax out arcs of minimum cost nodes



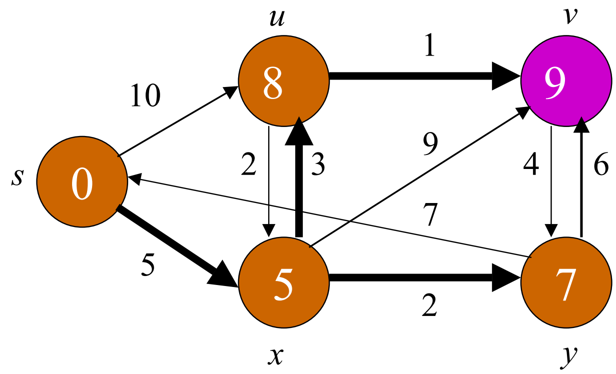

**Q = {u,v}** — Vertices to relax

**S = {s, x, y}** — **Vertices with shortest path value**

**Shortest path edge Π[v] = arrows**

# Dijkstra’s Algorithm

Assume all edges are non-negative.
Idea: Greedily relax out arcs of minimum cost nodes



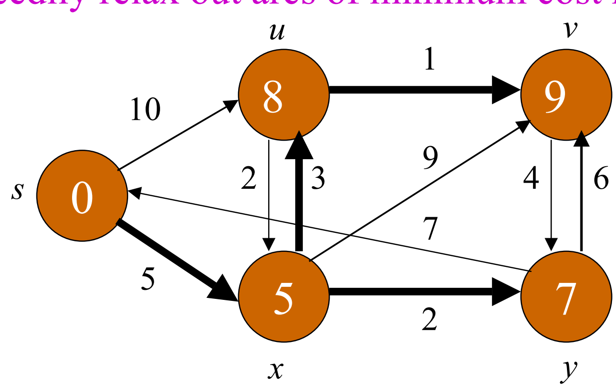

**Q = {v}**
**S = {s, x, y, u}**

Vertices to relax
**Vertices with shortest path value**
**Shortest path edge Π[v] = arrows**



# Dijkstra’s Algorithm

Assume all edges are non-negative.
Idea: Greedily relax out arcs of minimum cost nodes




**Q = {}**
**S = {s, x, y, u, v}**

Vertices to relax
**Vertices with shortest path value**
**Shortest path edge Π[v] = arrows**

# Dijkstra's Algorithm

Repeatedly select minimum cost node and relax out arcs

DIJKSTRA*(G,w,s)*

1. Initialize-Single-Source($G$, $s$) **O(V)**
2. $S \leftarrow \emptyset$
3. $Q \leftarrow V[G]$
4. **while** $Q \neq \emptyset$
5. **do** $u \leftarrow$ Extract-Min*(Q)* **O(V)** **or O(lg V) w fib heap**
6. $S \leftarrow S \cup \{u\}$
7. **for** each vertex $v \in Adj[u]$ *** O(V)**
8. **do** Relax*(u, v, w)* **O(E)**

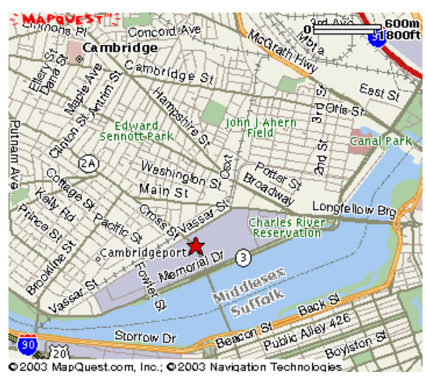
**= O(V²+E)** → $= O(V^2+E)$

**= O(VlgV+E)**



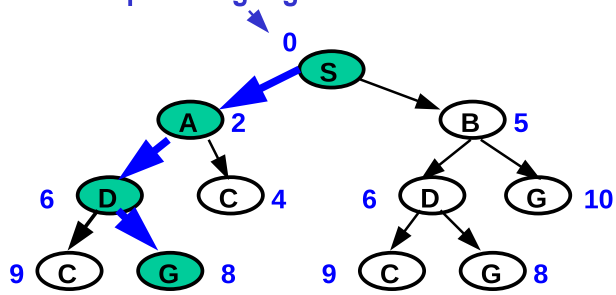
# Outline

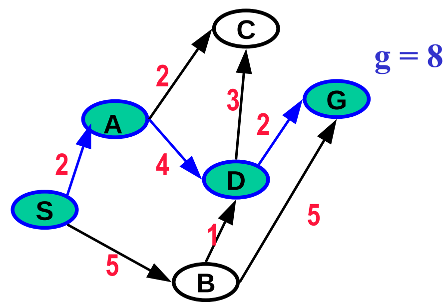
- Creating road maps for path planning
- Exploring roadmaps: Shortest Paths
  - Single Source
    - Dijkstra;s algorithm
  - Informed search
    - Uniform cost search
    - Greedy search
    - A* search
    - Beam search
    - Hill climbing
- Avoiding adversaries
  - (Next Lecture)




# Informed Search

Extend search tree nodes
to include path length g



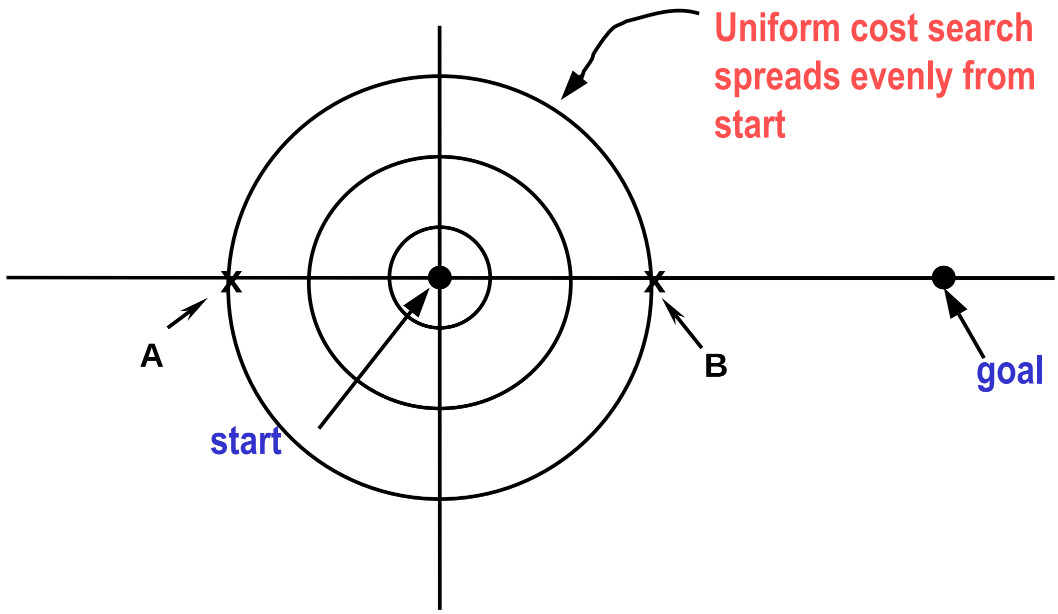

**Problem: Find the path to the goal G with the shortest path length g.**



# Classes of Search

| | | |
|---|---|---|
| **Blind** | **Depth-First** | Systematic exploration of whole tree |
| **(uninformed)** | **Breadth-First** | until the goal is found. |
| | **Iterative-Deepening** | |

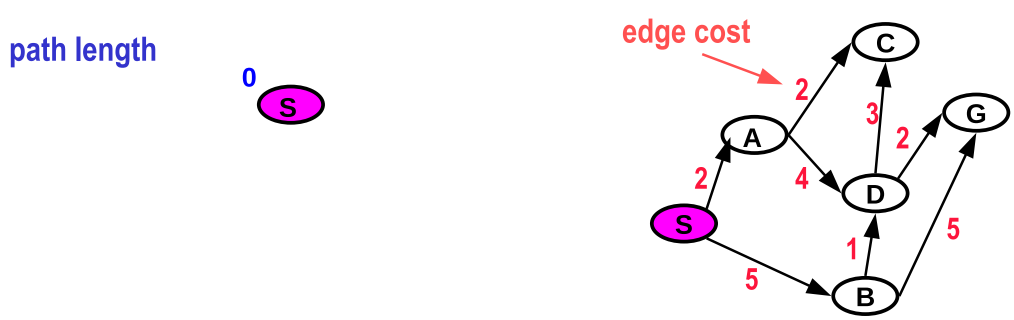
| | | |
|---|---|---|
| **Best-first** | **Uniform-cost** | Using path "length" as a measure, |
| **(informed)** | **Greedy** | finds "shortest" path. |
| | **A*** | |




Does uniform cost search find the shortest path?



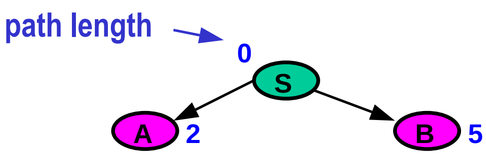
# Uniform Cost

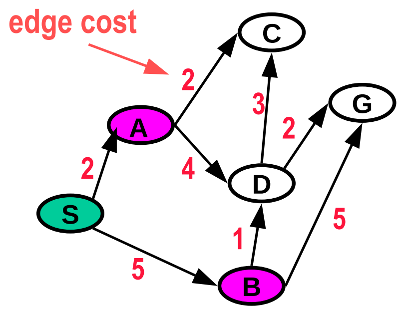



**Enumerates partial paths in order of increasing path length g.**

**May expand vertex more than once.**

# Uniform Cost



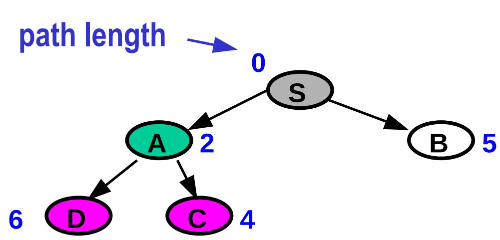



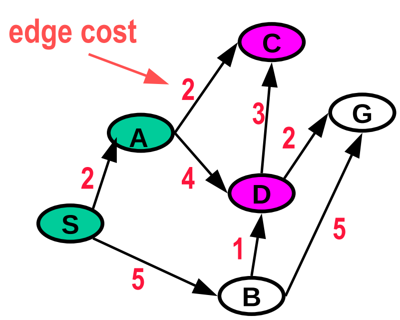

**Enumerates partial paths in order of increasing path length g.**

**May expand vertex more than once.**



# Uniform Cost

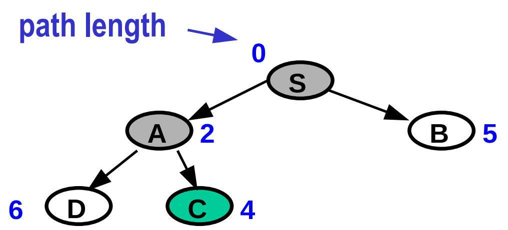


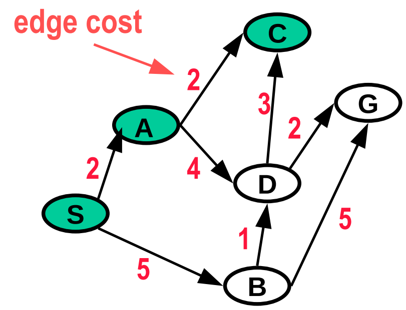




**Enumerates partial paths in order of increasing path length g.**

**May expand vertex more than once.**

# Uniform Cost



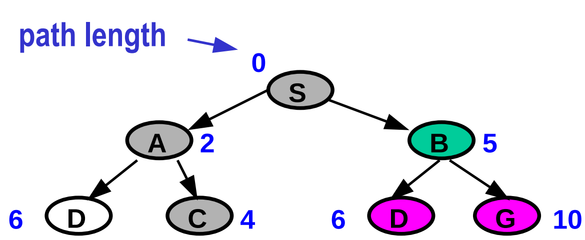



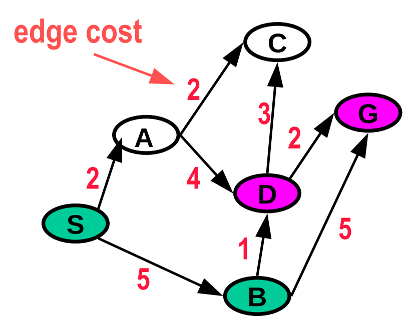

**Enumerates partial paths in order of increasing path length g.**

**May expand vertex more than once.**



# Uniform Cost

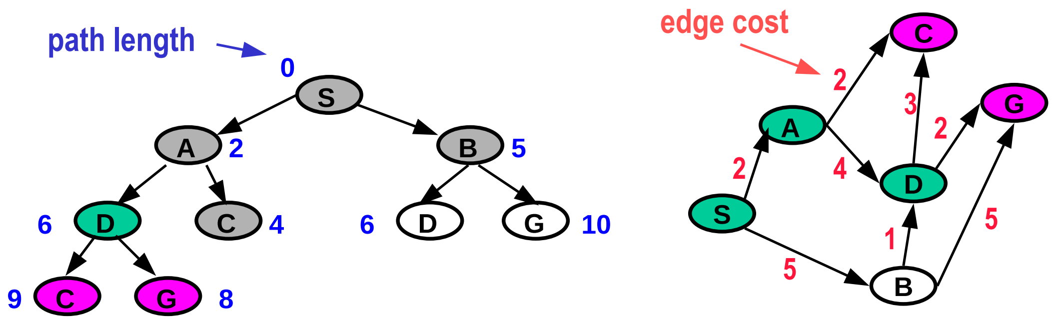



**Enumerates partial paths in order of increasing path length g.**

**May expand vertex more than once.**

# Uniform Cost

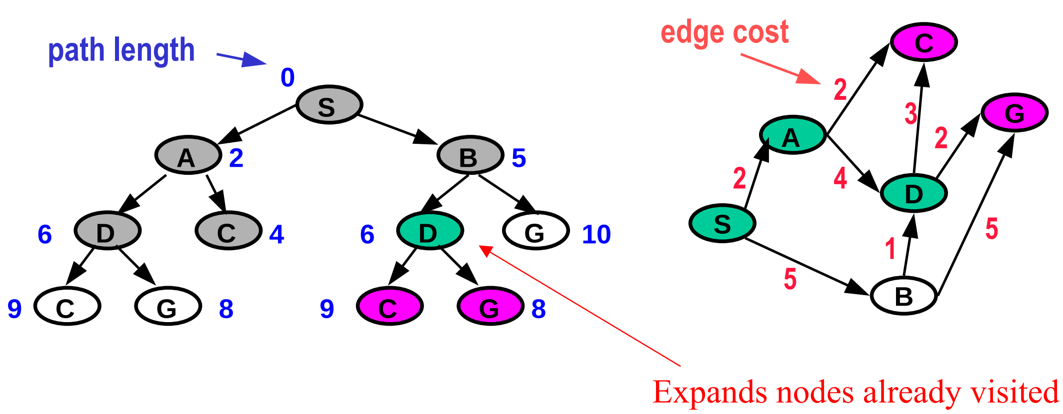



**Enumerates partial paths in order of increasing path length g.**

**May expand vertex more than once.**



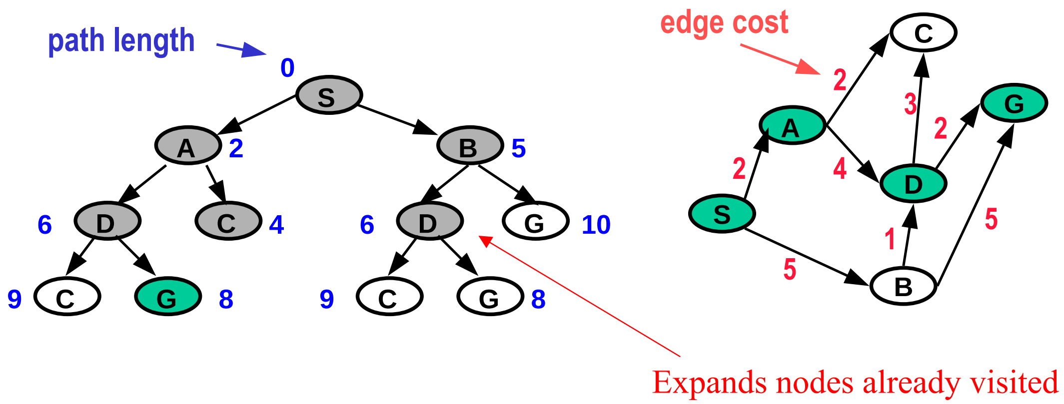
# Uniform Cost




**Enumerates partial paths in order of increasing path length g.**

**May expand vertex more than once.**

# Uniform Cost

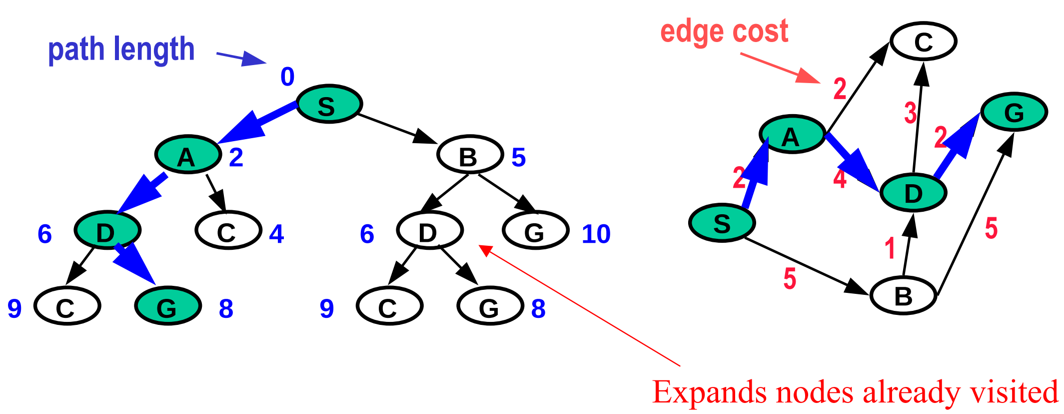



**Enumerates partial paths in order of increasing path length g.**

**May expand vertex more than once.**



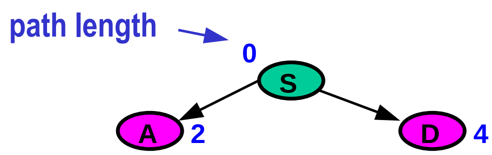
# Uniform Cost

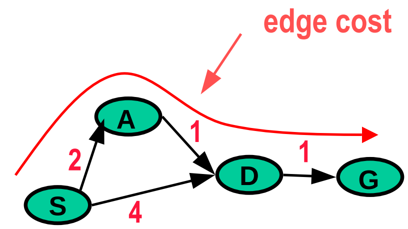



**Enumerates partial paths in order of increasing path length g.**

**May expand vertex more than once.**

# Why Expand a Vertex More Than Once?



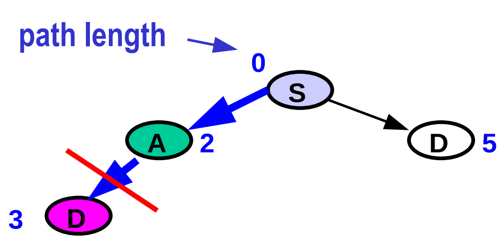



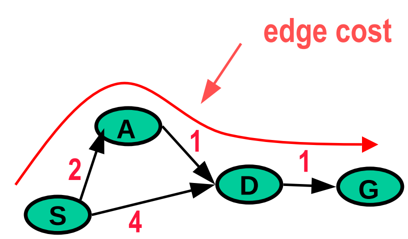

- The shortest path from S to G is (G D A S).
- D is reached first using path (D S).

Suppose we expanded only the first path to visit each vertex X?



# Why Expand a Vertex More Than Once?

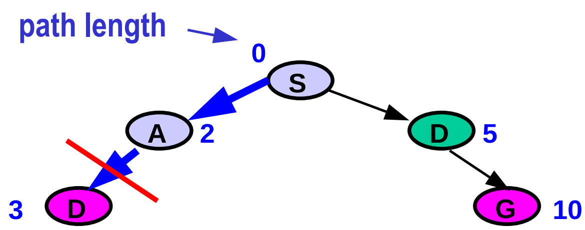


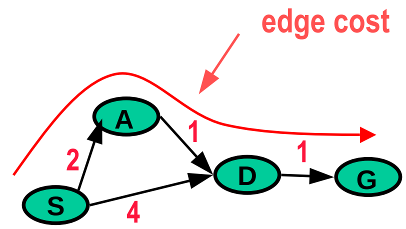




- The shortest path from S to G is (G D A S).
- D is reached first using path (D S).
- This prevents path (D A S) from being expanded.

Suppose we expanded only the first path to visit each vertex X?

# Why Expand a Vertex More Than Once?



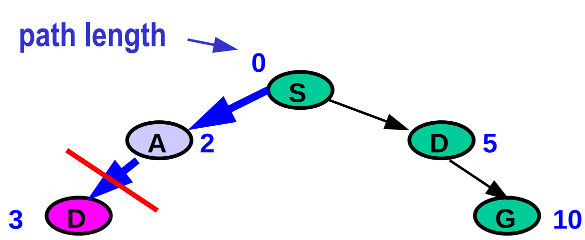



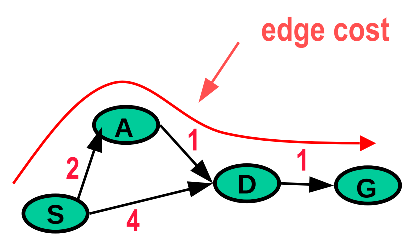

- The shortest path from S to G is (G D A S).
- D is reached first using path (D S).
- This prevents path (D A S) from being expanded.

Suppose we expanded only the first path to visit each vertex X?



# Why Expand a Vertex More Than Once?




- The shortest path from S to G is (G D A S).
- D is reached first using path (D S).
- This prevents path (D A S) from being expanded.
- The suboptimal path (G D S) is returned.

Suppose we expanded only the first path to visit each vertex X?

# Uniform Cost Search Algorithm

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.
Let g be the path length from S to N.

1. **Initialize Q with partial path (S) as only entry; ~~set Visited = ( )~~**
2. **If Q is empty, fail. Else, pick partial path N from Q with best g**
3. **If head(N) = G, return N (we've reached the goal!)**
4. **(Otherwise) Remove N from Q**
5. **Find all children of head(N) ~~not in Visited~~ and create all the one-step extensions of N to each child.**
6. **Add to Q all the extended paths;**
7. ~~**Add children of head(N) to Visited**~~
8. **Go to step 2.**



# Implementing the Search Strategies

**Depth-first:**

Pick first element of Q — Uses visited list

Add path extensions to front of Q

**Breadth-first:**

Pick first element of Q — Uses visited list

Add path extensions to end of Q

**Uniform-cost:**

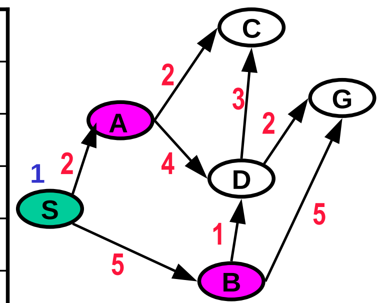
Pick first element of Q — **No visited list**

**Add path extensions to Q in order of increasing path length g.**

# Uniform Cost using BFS

Pick first element of Q;  Insert path extensions, sorted by g.

| | Q |
|---|---|
| 1 | ~~(0 S)~~ |
| 2 | (2 A S) (5 B S) |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |



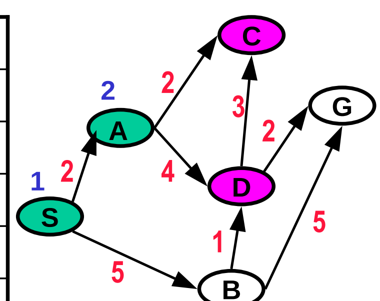

Here we:
- Insert on queue in order of g.
- Remove first element of queue.



# Uniform Cost using BFS

Pick first element of Q;  Insert path extensions, sorted by g.

| | Q |
|---|---|
| 1 | ~~(0 S)~~ |
| 2 | ~~(2 A S)~~ (5 B S) |
| 3 | (4 C A S) (5 B S) (6 D A S) |
| 4 | |
| 5 | |
| 6 | |
| 7 | |

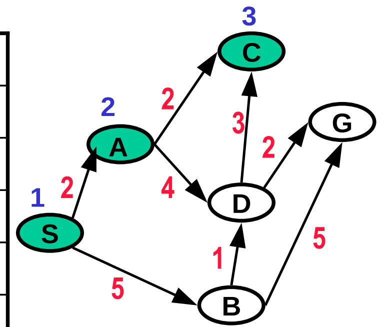



Here we:
- Insert on queue in order of g.
- Remove first element of queue.

# Uniform Cost using BFS

Pick first element of Q; Insert path extensions, sorted by g.

| | Q |
|---|---|
| 1 | ~~(0 S)~~ |
| 2 | ~~(2 A S)~~ (5 B S) |
| 3 | ~~(4 C A S)~~ (5 B S) (6 D A S) |
| 4 | (5 B S) (6 D A S) |
| 5 | |
| 6 | |
| 7 | |



Here we:

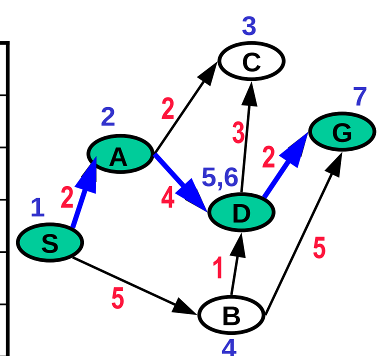
- Insert on queue in order of g.
- Remove first element of queue.



# Uniform Cost using BFS

Pick first element of Q; Insert path extensions, sorted by g.

| | Q |
|---|---|
| 1 | ~~(0 S)~~ |
| 2 | ~~(2 A S)~~ (5 B S) |
| 3 | ~~(4 C A S)~~ (5 B S) (6 D A S) |
| 4 | ~~(5 B S)~~ (6 D A S) |
| 5 | ~~(6 D B S)~~ (6 D A S) (10 G B S) |
| 6 | ~~(6 D A S)~~ (8 G D B S) (9 C D B S) (10 G B S) |
| 7 | (8 G D A S) (8 G D B S) (9 C D A S) (9 C D B S) (10 G B S) |



# Can we stop as soon as the goal is enqueued?

| | Q |
|---|---|
| 1 | ~~(0 S)~~ |
| 2 | ~~(2 A S)~~ (5 B S) |
| 3 | ~~(4 C A S)~~ (5 B S) (6 D A S) |
| 4 | ~~(5 B S)~~ (6 D A S) |
| 5 | ~~(6 D B S)~~ (6 D A S) (10 G B S) |
| 6 | ~~(6 D A S)~~ (8 G D B S) (9 C D B S) (10 G B S) |
| 7 | ~~(8 G D A S)~~ (8 G D B S) (9 C D A S) (9 C D B S) (10 G B S) |

- Other paths to the goal that are shorter **may not yet be enqueued**.
- Only **when a path is pulled off** the Q are we guaranteed that **no shorter path will be added**.
- This assumes all edges are **positive**.



# Implementing the Search Strategies

**Depth-first:**

Pick first element of Q — Uses visited list

Add path extensions to front of Q

**Breadth-first:**

Pick first element of Q — Uses visited list

Add path extensions to end of Q

**Uniform-cost:**

Pick first element of Q — No visited list

Add path extensions to Q in increasing order of path length g.

**Best-first: (generalizes uniform-cost)**

Pick first element of Q — **No visited list**

**Add path extensions in increasing order of any cost function f**

# Best-first Search Algorithm

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.
Let f be a cost function on N.

1. **Initialize Q with partial path (S) as only entry**
2. **If Q is empty, fail. Else, pick partial path N from Q with best f**
3. **If head(N) = G, return N (we've reached the goal!)**
4. **(Otherwise) Remove N from Q**
5. **Find all children of head(N) and create all the one-step extensions of N to each child.**
6. **Add to Q all the extended paths;**
7. **Go to step 2.**



# Classes of Search

| | | |
|---|---|---|
| **Blind** | **Depth-First** | Systematic exploration of whole tree |
| **(uninformed)** | **Breadth-First** | until the goal is found. |
| | **Iterative-Deepening** | |

| | | |
|---|---|---|
| **Best-first** | **Uniform-cost** | Using path "length" as a measure, |
| | **Greedy** | finds "shortest" path. |
| | **A*** | |




# Greedy Search

**Search in an order imposed by a heuristic function, measuring cost to go.**

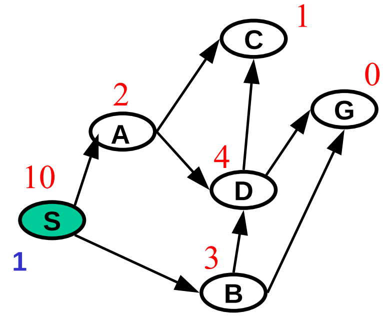
**Heuristic function h – is a function of the current node n, not the partial path s to n.**

- **Estimated distance to goal – h (n,G)**
  - **Example: straight-line distance in a road network.**
- **"Goodness" of a node – h (n)**
  - **Example: elevation**
  - **Foothills, plateaus and ridges are problematic.**

# Greedy

Pick first element of Q;  Insert path extensions, **sorted by h.**

| | Q | |
|---|---|---|
| 1 | (10 S) | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |




Heuristic values in red
Order of nodes in blue.

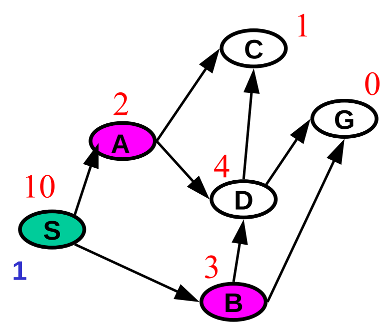
Added paths in blue; heuristic value of head is in front.



# Greedy

Pick first element of Q;  Insert path extensions, sorted by h.

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | (2 A S) (3 B S) | |
| 3 | | |
| 4 | | |
| 5 | | |



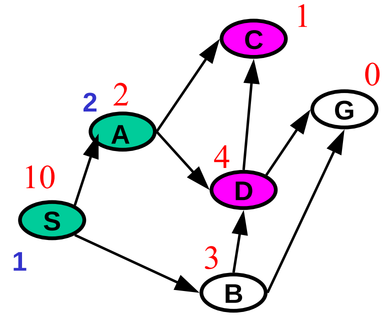

Heuristic values in red
Order of nodes in blue.

Added paths in blue; heuristic value of head is in front.

# Greedy

Pick first element of Q;  Insert path extensions, sorted by h.

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ (3 B S) | |
| 3 | (1 C A S) (3 B S) (4 D A S) | |
| 4 | | |
| 5 | | |




Added paths in blue; heuristic value of head is in front.

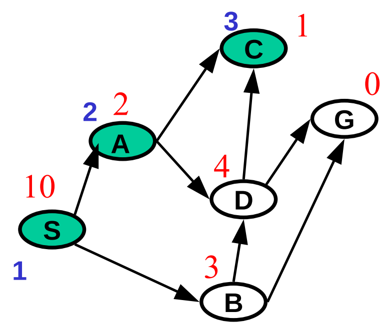
Heuristic values in red
Order of nodes in blue.



# Greedy

Pick first element of Q;  Insert path extensions, sorted by h.

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ (3 B S) | |
| 3 | ~~(1 C A S)~~ (3 B S) (4 D A S) | |
| 4 | (3 B S) (4 D A S) | |
| 5 | | |



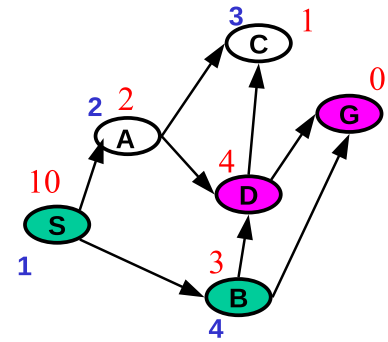

Added paths in blue; heuristic value of head is in front.

Heuristic values in red
Order of nodes in blue.

# Greedy

Pick first element of Q; Insert path extensions, sorted by h.

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ (3 B S) | |
| 3 | ~~(1 C A S)~~ (3 B S) (4 D A S) | |
| 4 | ~~(3 B S)~~ (4 D A S) | |
| 5 | (0 G B S) (4 D A S) (4 D B S) | |




Heuristic values in red
Order of nodes in blue.

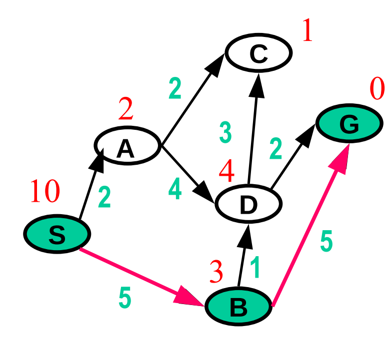
Added paths in blue; heuristic value of head is in front.



# Greedy

Pick first element of Q; Insert path extensions, sorted by h.

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ (3 B S) | |
| 3 | ~~(1 C A S)~~ (3 B S) (4 D A S) | |
| 4 | ~~(3 B S)~~ (4 D A S) | |
| 5 | $\boxed{\text{(0 G B S)}}$ (4 D A S) ) (4 D B S) | |




Heuristic values in red
Edge cost in green.

Added paths in blue; heuristic value of head is in front.

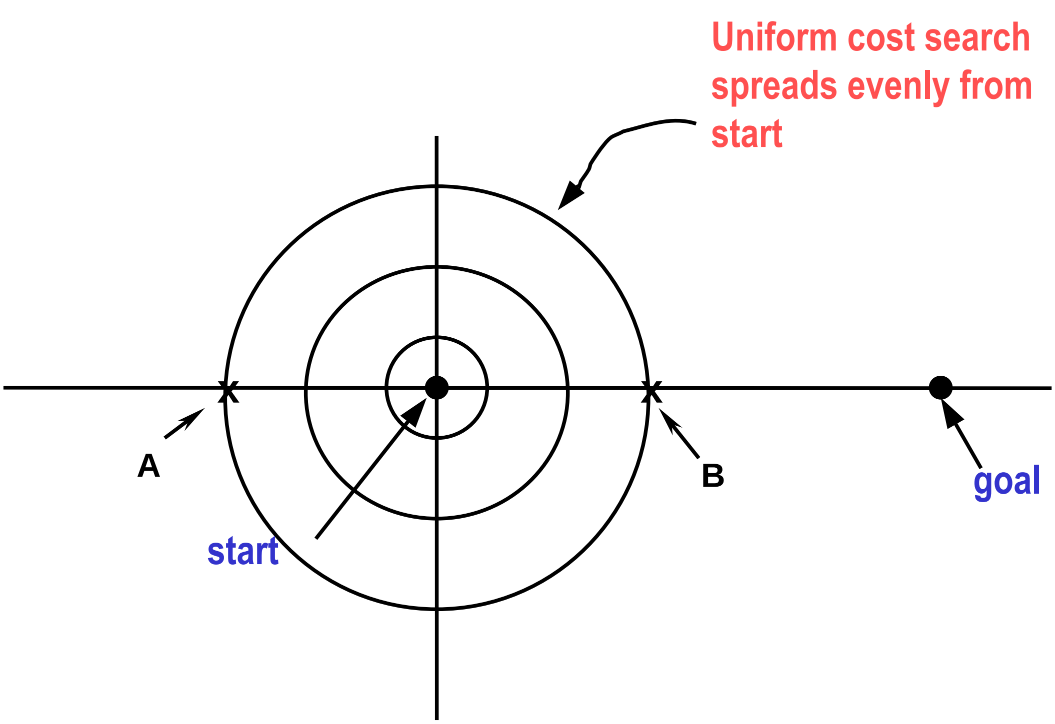
Was the shortest path produced?

# Classes of Search

| | | |
|---|---|---|
| **Blind** | **Depth-First** | Systematic exploration of whole tree |
| **(uninformed)** | **Breadth-First** | until the goal is found. |
| | **Iterative-Deepening** | |

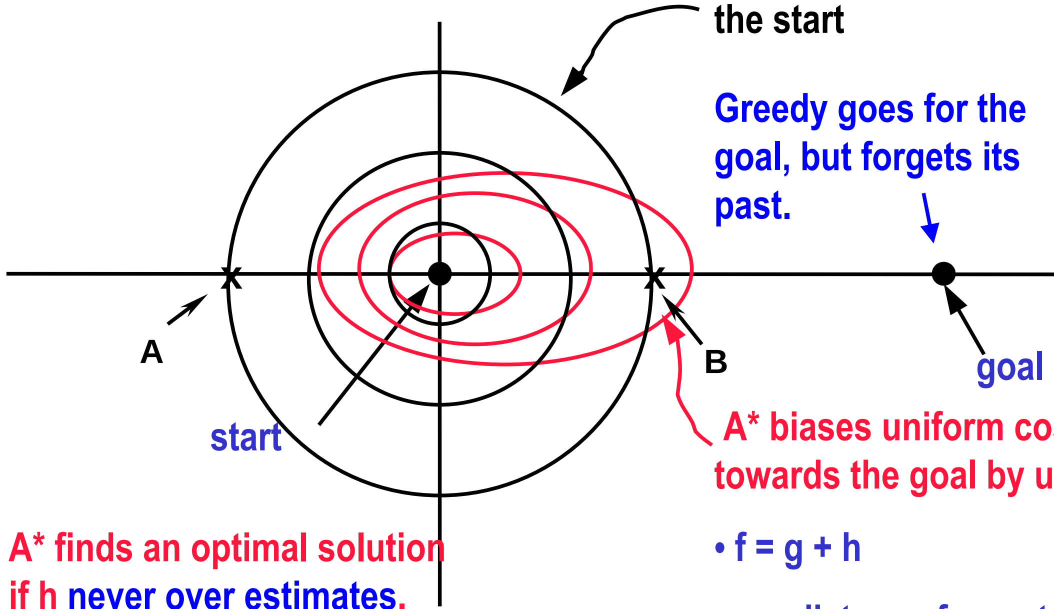
| | | |
|---|---|---|
| **Best-first** | **Uniform-cost** | Using path "length" as a measure, |
| | **Greedy** | finds "shortest" path. |
| | **A*** | |





Uniform cost search spreads evenly from the start

Greedy goes for the goal, but forgets its past.




A* biases uniform cost towards the goal by using h

- f = g + h
- g = distance from start
- h = estimated distance to goal.

A* finds an optimal solution if h never over estimates.

Then h is called "admissible"



# Simple Optimal Search Algorithm

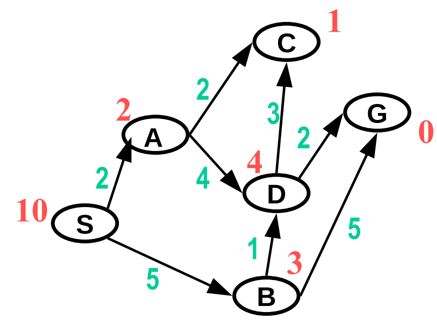
## BFS + Admissible Heuristic

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.
Let f = g + h be an admissible heuristic function

1. **Initialize Q with partial path (S) as only entry;**
2. **If Q is empty, fail. Else, use f to pick "best" partial path N from Q**
3. **If head(N) = G, return N (we've reached the goal)**
4. **(Otherwise) Remove N from Q;**
5. **Find all the descendants of head(N) and create all the one-step extensions of N to each descendant.**
6. **Add to Q all the extended paths.**
7. **Go to step 2.**

# In the example, is h an admissible heuristic?

- A is ok
- B is ok
- C is ok
- D is too big, needs to be ≤ 2
- S is too big, can always use 0 for start




Heuristic Values of h in Red

Edge cost in Green

**A* finds an optimal solution if h never over estimates.**

**Then h is called "admissible"**



# Admissible heuristics for 8 puzzle?

S

| | | |
|---|---|---|
| 6 | 2 | 8 |
| | 3 | 5 |
| 4 | 7 | 1 |

⇨

G

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

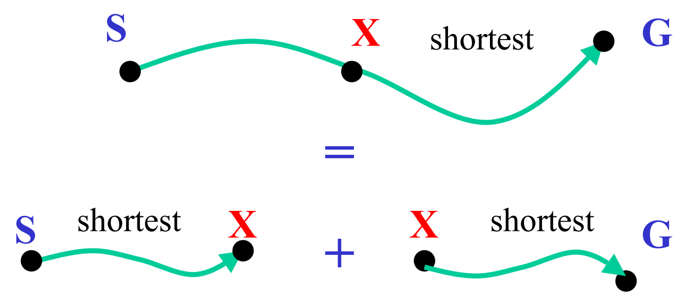
**What is the heuristic?**

- An underestimate of number of moves to the goal.

**Examples:**

1. Number of misplaced tiles (7)
2. Sum of Manhattan distance of each tile to its goal location (17)

# A* Incorporates the Dynamic Programming Principle




**The shortest path from S through X to G**
**= shortest path from S to X + shortest path from X to G.**

- **Idea: when shortest from S to X is found, ignore other S to X paths.**
  - **When BFS dequeues the first partial path with head node X, this path is the shortest path from S to X.**
- **Given the first path to X, we don't need to extend other paths to X; delete them.**



# Simple Optimal Search Algorithm

## How do we add dynamic programming?

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.
**Let f = g + h be an admissible heuristic function**

1. **Initialize Q with partial path (S) as only entry;**
2. **If Q is empty, fail. Else, use f to pick the "best" partial path N from Q**
3. **If head(N) = G, return N (we've reached the goal)**
4. **(Else) Remove N from Q;**
5. **Find all children of head(N) and create all the one-step extensions of N to each child.**
6. **Add to Q all the extended paths.**
7. **Go to step 2.**

# A* Optimal Search Algorithm

## BFS + Dyn Prog + Admissible Heuristic

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.
**Let f = g + h be an admissible heuristic function**

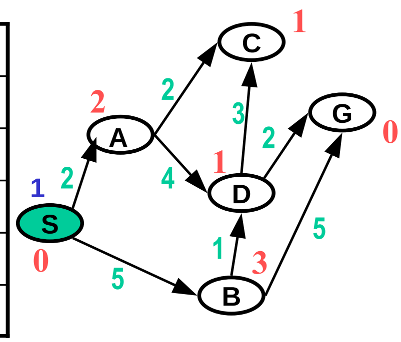
1. **Initialize Q with partial path (S) as only entry; set Expanded = ( )**
2. **If Q is empty, fail. Else, use f to pick "best" partial path N from Q**
3. **If head(N) = G, return N** **(we've reached the goal)**
4. **(Else) Remove N from Q;**
5. **if head(N) is in Expanded, go to step 2, otherwise add head(N) to Expanded.**
6. **Find all the children of head(N) (not in Expanded) and create all the one-step extensions of N to each child.**
7. **Add to Q all the extended paths.**
8. **Go to step 2.**



# A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q; Insert path extensions, sorted by path length + heuristic.

| | Q | Expanded |
|---|---|---|
| 1 | (0 S) | |
| | | |
| | | |
| | | |
| | | |



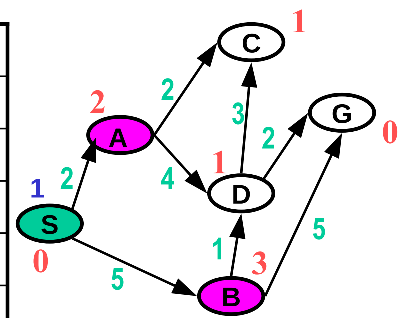

**Heuristic Values of g in Red**
**Edge cost in Green**

**Added paths in blue; cost f at head of each path.**

# A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q; Insert path extensions, sorted by path length + heuristic.

| | Q | Expanded |
|---|---|---|
| 1 | ~~(0 S)~~ | |
| 2 | | S |
| | | |
| | | |
| | | |




**Heuristic Values of g in Red**

**Edge cost in Green**

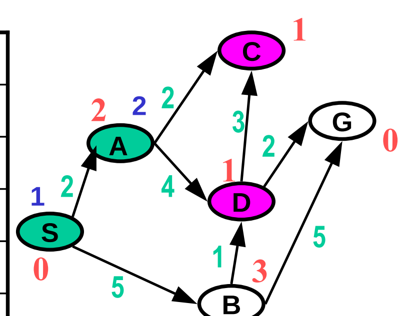
**Added paths in blue; cost f at head of each path**



# A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q; Insert path extensions, sorted by path length + heuristic.

| | Q | Expanded |
|---|---|---|
| 1 | ~~(0 S)~~ | |
| 2 | ~~(4 A S)~~ (8 B S) | S |
| 3 | | S A |
| | | |
| | | |




**Heuristic Values of g in Red**

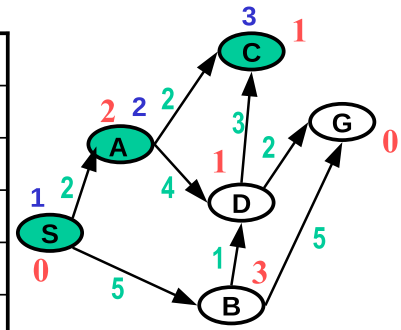
**Edge cost in Green**

**Added paths in blue; cost f at head of each path**

# A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q;  Insert path extensions, sorted by path length + heuristic.

| | Q | Expanded |
|---|---|---|
| 1 | (~~0 S~~) | |
| 2 | (~~4 A S~~) (8 B S) | S |
| 3 | (~~5 C A S~~) (7 D A S) (8 B S) | S A |
| 4 | | S A C |
| | | |




**Heuristic Values of g in Red**

**Edge cost in Green**

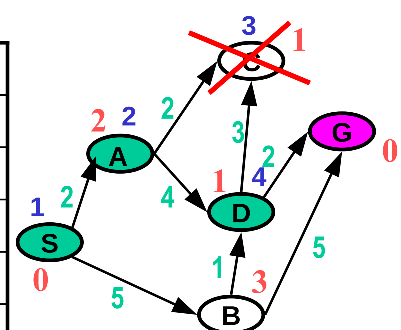
**Added paths in blue; cost f at head of each path**



# A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q;  Insert path extensions, sorted by path length + heuristic.

| | Q | Expanded |
|---|---|---|
| 1 | (~~0 S~~) | |
| 2 | (~~4 A S~~) (8 B S) | S |
| 3 | (~~5 C A S~~) (7 D A S) (8 B S) | S A |
| 4 | (~~7 D A S~~) (8 B S) | S A C |
| 5 | | S A C D |




**Heuristic Values of g in Red**

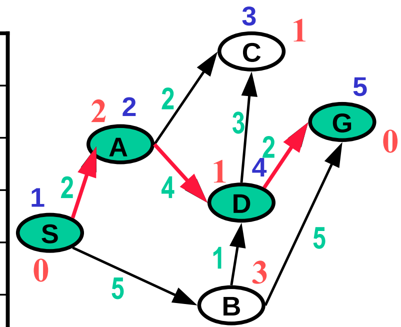
**Edge cost in Green**

**Added paths in blue; cost f at head of each path**

# A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q;  Insert path extensions, sorted by path length + heuristic.

| | Q | Expanded |
|---|---|---|
| 1 | ~~(0 S)~~ | |
| 2 | ~~(4 A S)~~ (8 B S) | S |
| 3 | ~~(5 C A S)~~ (7 D A S) (8 B S) | S A |
| 4 | ~~(7 D A S)~~ (8 B S) | S A C |
| 5 | $\boxed{\text{(8 G D A S)}}$ (8 B S) | S A C D |




**Heuristic Values of g in Red**
**Edge cost in Green**

**Added paths in blue; cost f at head of each path**



# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| **Depth-First** | $b^m$ | $b*m$ | Yes | No |
| **Breadth-First** | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| **Best-First** | | | | |
| **Beam (beam width = k)** | | | | |
| **Hill-Climbing (no backup)** | | | | |
| **Hill-Climbing (backup)** | | | | |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**

# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| Depth-First | $b^m$ | $b*m$ | Yes | No |
| Breadth-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| Best-First | $b^{d+1}$ | $b^{d+1}$ | | |
| Beam (beam width = k) | | | | |
| Hill-Climbing (no backup) | | | | |
| Hill-Climbing (backup) | | | | |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**



# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| Depth-First | $b^m$ | $b*m$ | Yes | No |
| Breadth-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| Best-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| Beam (beam width = k) | | | | |
| Hill-Climbing (no backup) | | | | |
| Hill-Climbing (backup) | | | | |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**

# Classes of Search

| | | |
|---|---|---|
| **Blind** | **Depth-First** | Systematic exploration of whole tree |
| **(uninformed)** | **Breadth-First** | until the goal is found. |
| | **Iterative-Deepening** | |

| | | |
|---|---|---|
| **Best-first** | **Uniform-cost** | Uses path "length" measure. Finds |
| | **Greedy** | "shortest" path. |
| | **A*** | |

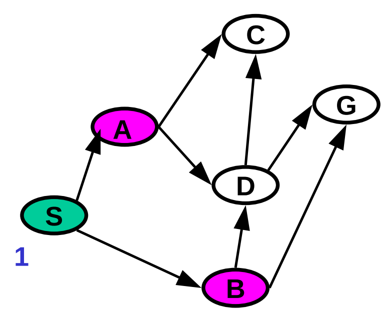
| | |
|---|---|
| **Variants** | **Beam** |
| | **Hill-Climbing (w backup)** |
| | **ID A*** |



# Hill-Climbing

Pick **first element** of Q; **Replace Q** with **extensions** (sorted by **heuristic value**)

| | Q | |
|---|---|---|
| 1 | (~~10~~ S) | |
| 2 | (2 A S) (3 B S) | |
| 3 | | |
| 4 | | |




**Heuristic Values**

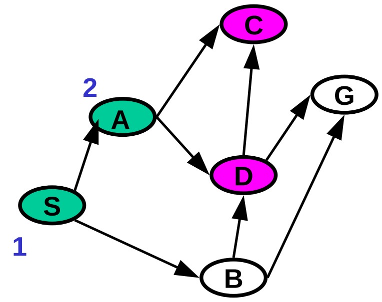
| | | |
|---|---|---|
| A=2 | C=1 | S=10 |
| B=3 | D=4 | G=0 |

**Added paths in blue; heuristic value of head is in front.**

# Hill-Climbing

Pick **first element** of Q; **Replace Q** with **extensions** (sorted by **heuristic value**)

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ (3 B S) Removed | |
| 3 | (1 C A S) (4 D A S) | |
| 4 | | |




**Heuristic Values**

| | | |
|---|---|---|
| A=2 | C=1 | S=10 |
| B=3 | D=4 | G=0 |

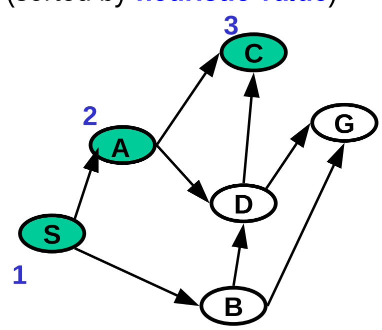
**Added paths in blue; heuristic value of head is in front.**



# Hill-Climbing

Pick **first element** of Q; **Replace Q** with **extensions** (sorted by **heuristic value**)

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ (3 B S) | |
| 3 | ~~(1 C A S)~~ (4 D A S) | |
| 4 | ( ) | |




**Fails to find a path!**

**Heuristic Values**

| | | |
|---|---|---|
| A=2 | C=1 | S=10 |
| B=3 | D=4 | G=0 |

**Added paths in blue; heuristic value of head is in front.**

# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| Depth-First | $b^m$ | $b*m$ | Yes | No |
| Breadth-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| Best-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| Beam (beam width = k) | | | | |
| Hill-Climbing (no backup) | | | | |
| Hill-Climbing (backup) | | | | |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**



# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| Depth-First | $b^m$ | $b*m$ | Yes | No |
| Breadth-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| Best-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| Beam (beam width = k) | | | | |
| Hill-Climbing (no backup) | $b*m$ | | | |
| Hill-Climbing (backup) | | | | |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**

# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| Depth-First | $b^m$ | $b*m$ | Yes | No |
| Breadth-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| Best-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| Beam (beam width = k) | | | | |
| Hill-Climbing (no backup) | $b*m$ | b | | |
| Hill-Climbing (backup) | | | | |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**



# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

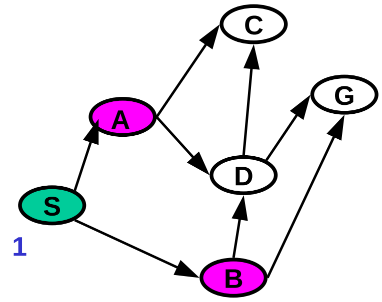
| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| Depth-First | $b^m$ | $b*m$ | Yes | No |
| Breadth-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| Best-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| Beam (beam width = k) | | | | |
| Hill-Climbing (no backup) | $b*m$ | b | No | No |
| Hill-Climbing (backup) | | | | |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**

# Hill-Climbing (with backup)

Pick first element of Q; **Add** path extensions (sorted by heuristic value) **to front of Q**

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | (2 A S) (3 B S) | |
| 3 | | |
| 4 | | |
| 5 | | |




**Heuristic Values**

**A=2 C=1 S=10**

**B=3 D=4 G=0**

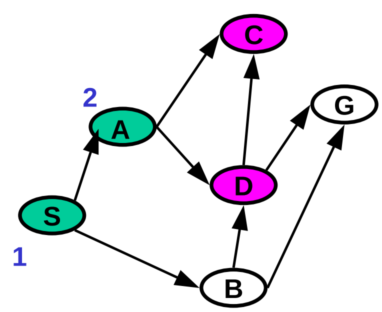
**Added paths in blue; heuristic value of head is in front.**



# Hill-Climbing (with backup)

Pick first element of Q; **Add** path extensions (sorted by heuristic value) **to front of Q**

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ (3 B S) | |
| 3 | (1 C A S) (4 D A S) (3 B S) | |
| 4 | All new nodes before old | |
| 5 | | |




**Heuristic Values**

**A=2 C=1 S=10**

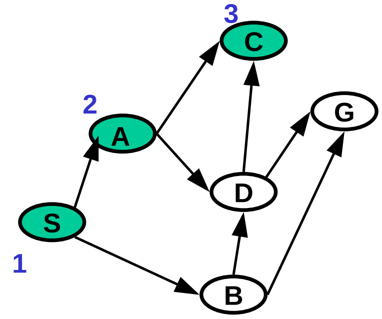
**B=3 D=4 G=0**

**Added paths in blue; heuristic value of head is in front.**

# Hill-Climbing (with backup)

Pick first element of Q; **Add** path extensions (sorted by heuristic value) **to front of Q**

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ (3 B S) | |
| 3 | ~~(1 C A S)~~ (4 D A S) (3 B S) | |
| 4 | (4 D A S) (3 B S) | |
| 5 | | |




**Heuristic Values**

**A=2 C=1 S=10**

**B=3 D=4 G=0**

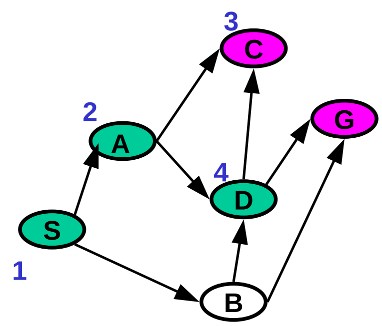
**Added paths in blue; heuristic value of head is in front.**



# Hill-Climbing (with backup)

Pick first element of Q; **Add** path extensions (sorted by heuristic value) **to front of Q**

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ (3 B S) | |
| 3 | ~~(1 C A S)~~ (4 D A S) (3 B S) | |
| 4 | ~~(4 D A S)~~ (3 B S) | |
| 5 | (0 G D A S) (1 C A S) (3 B S) | |




**Heuristic Values**

**A=2 C=1 S=10**

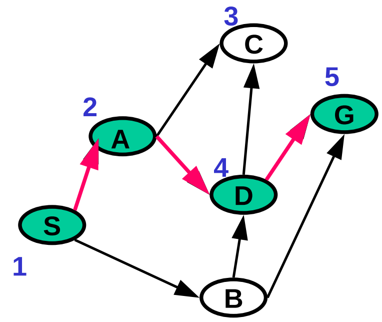
**B=3 D=4 G=0**

**Added paths in blue; heuristic value of head is in front.**

# Hill-Climbing (with backup)

Pick first element of Q; **Add** path extensions (sorted by heuristic value) **to front of Q**

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ (3 B S) | |
| 3 | ~~(1 C A S)~~ (4 D A S) (3 B S) | |
| 4 | ~~(4 D A S)~~ (3 B S) | |
| 5 | (0 G D A S) (1 C A S) (3 B S) | |




**Heuristic Values**

| | | |
|---|---|---|
| A=2 | C=1 | S=10 |
| B=3 | D=4 | G=0 |

**Added paths in blue; heuristic value of head is in front.**



# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| Depth-First | $b^m$ | $b*m$ | Yes | No |
| Breadth-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| Best-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| Beam (beam width = k) | | | | |
| Hill-Climbing (no backup) | $b*m$ | b | No | No |
| Hill-Climbing (backup) | | | | |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**

# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| **Depth-First** | $b^m$ | $b*m$ | Yes | No |
| **Breadth-First** | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| **Best-First** | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| **Beam (beam width = k)** | | | | |
| **Hill-Climbing (no backup)** | $b*m$ | $b$ | No | No |
| **Hill-Climbing (backup)** | $b^m$ | $b*m$ | Yes | No |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**



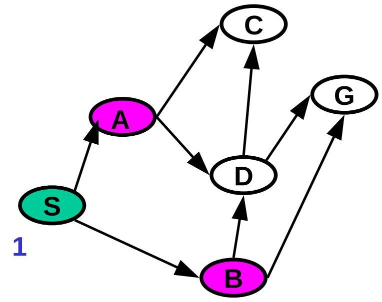
# Classes of Search

| | | |
|---|---|---|
| **Blind (uninformed)** | **Depth-First**<br>**Breadth-First**<br>**Iterative-Deepening** | Systematic exploration of whole tree until the goal is found. |
| **Best-first** | **Uniform-cost**<br>**Greedy**<br>**A*** | Uses path "length" measure. Finds "shortest" path. |
| **Variants** | **Beam**<br>**Hill-Climbing (w backup)**<br>**ID A*** | |

# Beam

**Expand all Q elements; Keep** the **k best extensions** (sorted by **heuristic value**)

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | | |
| | | |




Idea: Incrementally expand the k best paths

**Heuristic Values**

| | | |
|---|---|---|
| A=2 | C=1 | S=10 |
| B=3 | D=4 | G=0 |

**Added paths in blue; heuristic value of head is in front.**

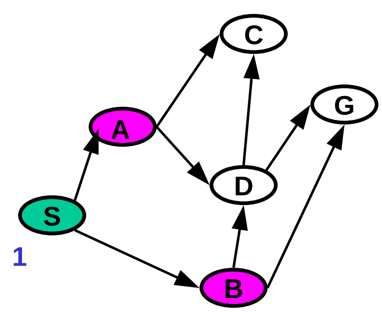
**Let k = 2**



# Beam

**Expand all Q elements; Keep** the **k best extensions** (sorted by **heuristic value**)

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | (2 A S) (3 B S) | |
| | | |




Idea: Incrementally expand the k best paths

**Heuristic Values**

| | | |
|---|---|---|
| A=2 | C=1 | S=10 |
| B=3 | D=4 | G=0 |

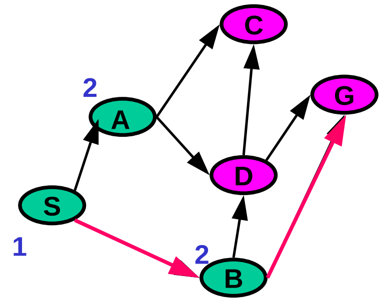
**Added paths in blue; heuristic value of head is in front.**

**Let k = 2**

# Beam

**Expand all Q elements; Keep** the **k best extensions** (sorted by **heuristic value**)

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ ~~(3 B S)~~ | |
| 3 | (0 G B S) (1 C A S) ~~(4 D A S) (4 D B S)~~ **Keep k best** | |




Idea: Incrementally expand the k best paths

**Heuristic Values**

| | | |
|---|---|---|
| A=2 | C=1 | S=10 |
| B=3 | D=4 | G=0 |

**Added paths in blue; heuristic value of head is in front.**

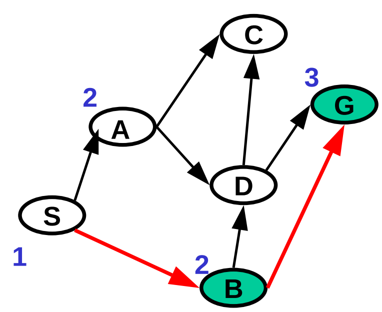
**Let k = 2**



# Beam

**Expand all Q elements; Keep** the **k best extensions** (sorted by **heuristic value**)

| | Q | |
|---|---|---|
| 1 | ~~(10 S)~~ | |
| 2 | ~~(2 A S)~~ ~~(3 B S)~~ | |
| 3 | $\boxed{\text{(0 G B S)}}$ (1 C A S) ~~(4 D A S) (4 D B S)~~ **Keep k best** | |




Idea: Incrementally expand the k best paths

**Heuristic Values**

| | | |
|---|---|---|
| A=2 | C=1 | S=10 |
| B=3 | D=4 | G=0 |

**Added paths in blue; heuristic value of head is in front.**

**Let k = 2**

# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| Depth-First | $b^m$ | b*m | Yes | No |
| Breadth-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| Best-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| Beam (beam width = k) | | | | |
| Hill-Climbing (no backup) | b*m | b | No | No |
| Hill-Climbing (backup) | $b^m$ | b*m | Yes | No |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**



# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| Depth-First | $b^m$ | b*m | Yes | No |
| Breadth-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| Best-First | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| Beam (beam width = k) | | k*b | | |
| Hill-Climbing (no backup) | b*m | b | No | No |
| Hill-Climbing (backup) | $b^m$ | b*m | Yes | No |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**

# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| **Depth-First** | $b^m$ | $b*m$ | Yes | No |
| **Breadth-First** | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| **Best-First** | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| **Beam (beam width = k)** | $k*b*m$ | $k*b$ | | |
| **Hill-Climbing (no backup)** | $b*m$ | $b$ | No | No |
| **Hill-Climbing (backup)** | $b^m$ | $b*m$ | Yes | No |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**



# Cost and Performance

**Searching a tree with branching factor b, solution depth d, and max depth m**

| Search Method | Worst Time | Worst Space | Guaranteed to find a path? | Optimal? |
|---|---|---|---|---|
| **Depth-First** | $b^m$ | $b*m$ | Yes | No |
| **Breadth-First** | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes for unit edge cost |
| **Best-First** | $b^{d+1}$ | $b^{d+1}$ | Yes | Yes if uniform cost or A* w admissible heuristic. |
| **Beam (beam width = k)** | $k*b*m$ | $k*b$ | No | No |
| **Hill-Climbing (no backup)** | $b*m$ | $b$ | No | No |
| **Hill-Climbing (backup)** | $b^m$ | $b*m$ | Yes | No |

**Worst case time is proportional to number of nodes visited**
**Worst case space is proportional to maximal length of Q**

# Outline

- Creating road maps for path planning
- Exploring roadmaps: Shortest Path
  - Single Source
    - Dijkstra;s algorithm
  - Informed search
    - Uniform cost search
    - Greedy search
    - A* search
    - Beam search
    - Hill climbing
- Avoiding adversaries
  - (Next Lecture)