# Lecture 5

# Least Cost and Informed Search

(Ch: 3.6, 3.6.1)

# Announcements

- Assignment 1 posted today
  - Due Monday 29th July, 11:59pm
  - You can do this assignment in pairs
  - Please carefully read and follow the instructions on cover sheet

# Lecture Overview

- Recap of Lecture 4 • Least Cost First Search • Heuristic (Informed) Search
  - Best First
  - A*
  - Branch and Bound (time permitting)

Search Strategies are different with respect to how they:

A. Check what node on a path is the goal

B. Initialize the frontier

C. Add/remove paths from the frontier

D. Check if a state is a goal

Search Strategies are different with respect to how they:

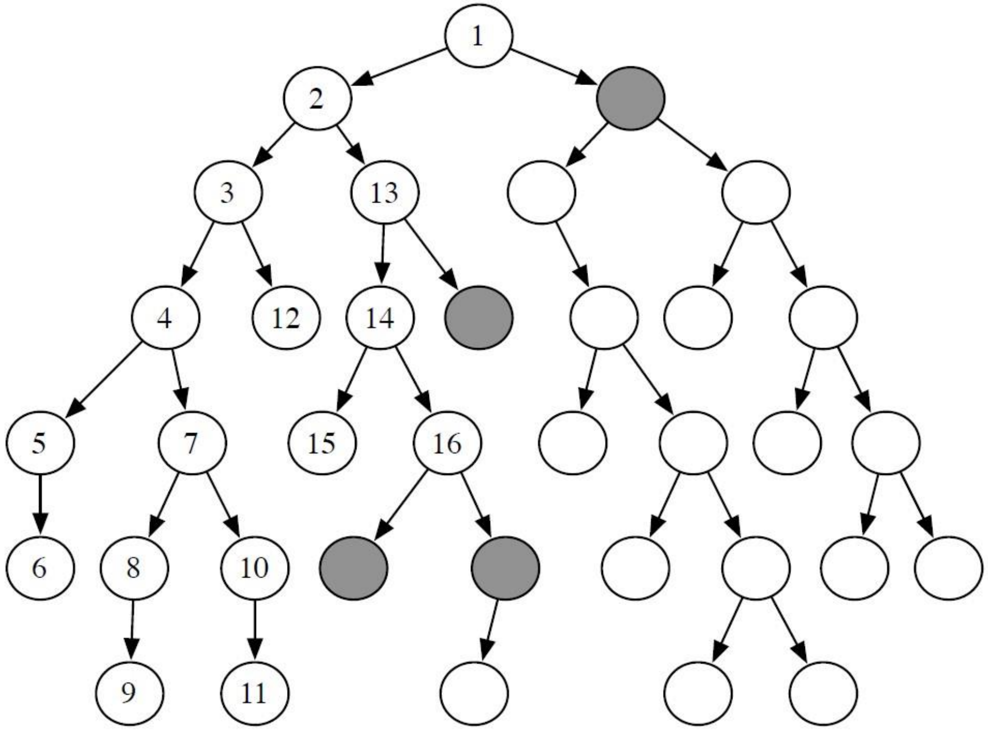
A. Check what node on a path is the goal

B. Initialize the frontier

C. Add/remove paths from the frontier

D. Check if a state is a goal

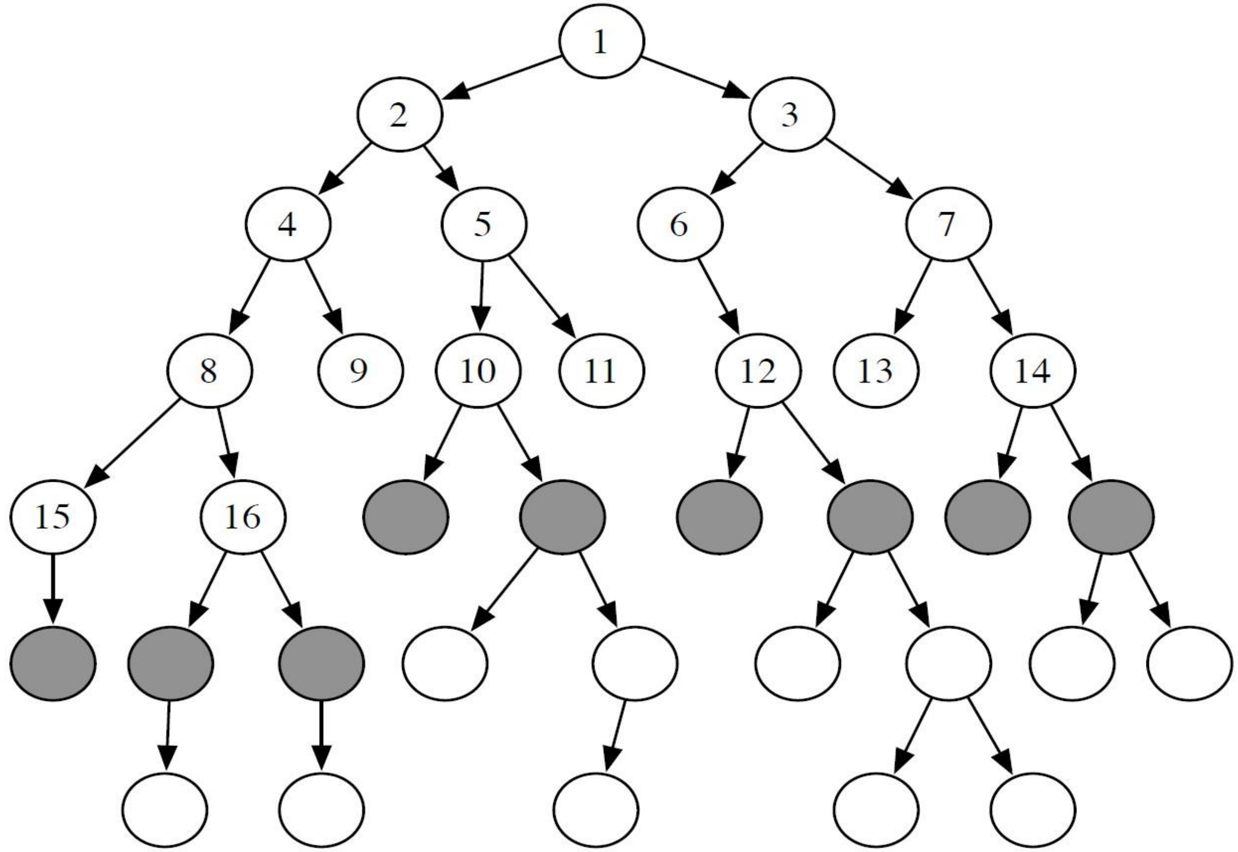
# DFS




Depth-First Search, DFS

- explores each path on the frontier until its end (or until a goal is found) before considering any other path.
- the frontier is a last-in-first-out stack

# Breadth-first search (BFS)



- BFS explores all paths of length l on the frontier, before looking at path of length l + 1
- The frontier is a first-in-first-out queue

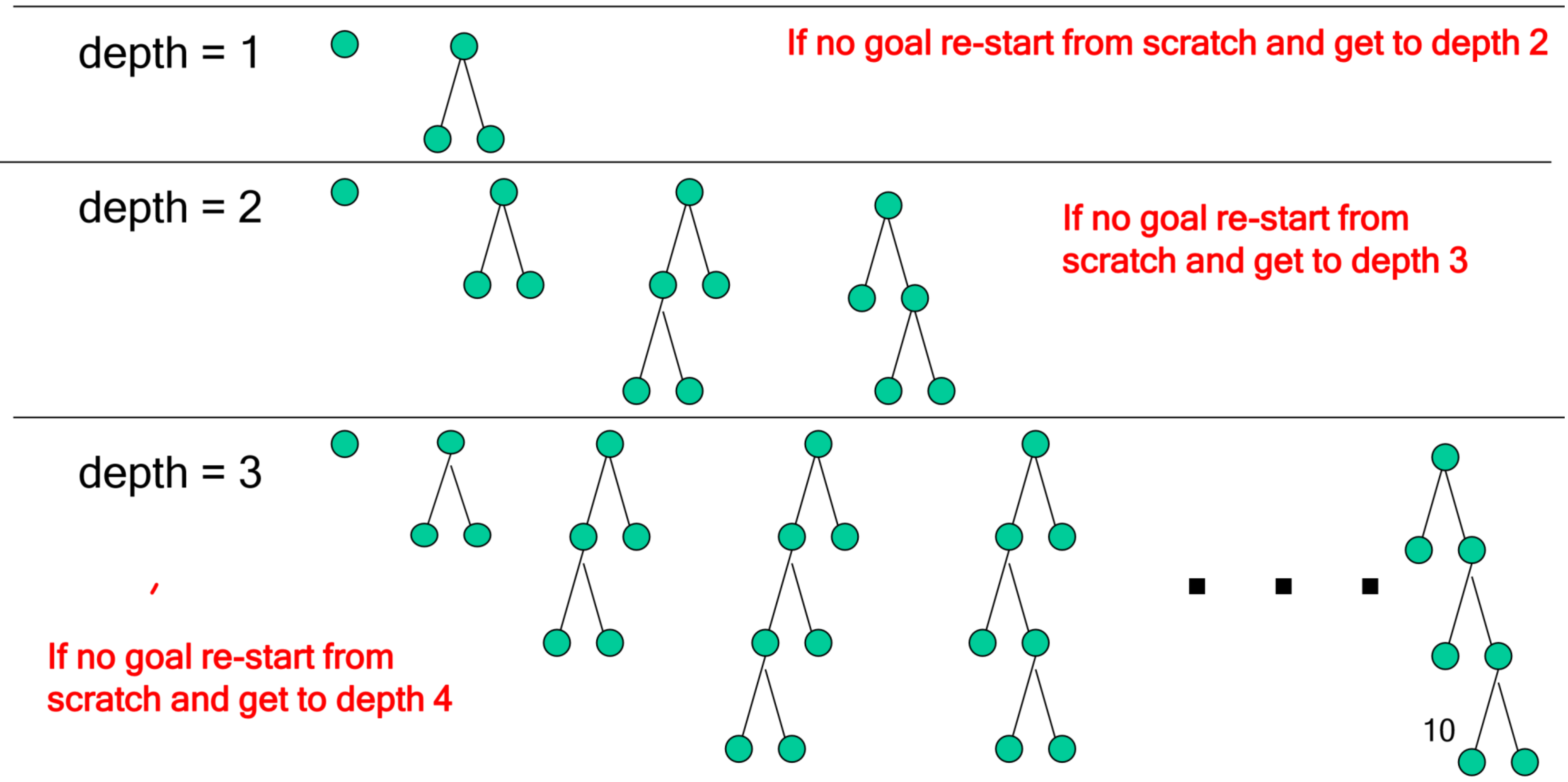
# DFS vs. BFS

| | Complete | Optimal | Time | Space |
|---|---|---|---|---|
| DFS | NO | NO | $O(b^m)$ | $O(bm)$ |
| BFS | YES | YES | $O(b^m)$ | $O(b^m)$ |

**Key Idea:** re-compute elements of the frontier rather than saving them.

# Iterative Deepening DFS (IDS) in a Nutshell

- Use DFS to look for solutions at depth 1, then 2, then 3, etc
  - Depth-bounded depth-first search



- For depth D, ignore any paths with longer length

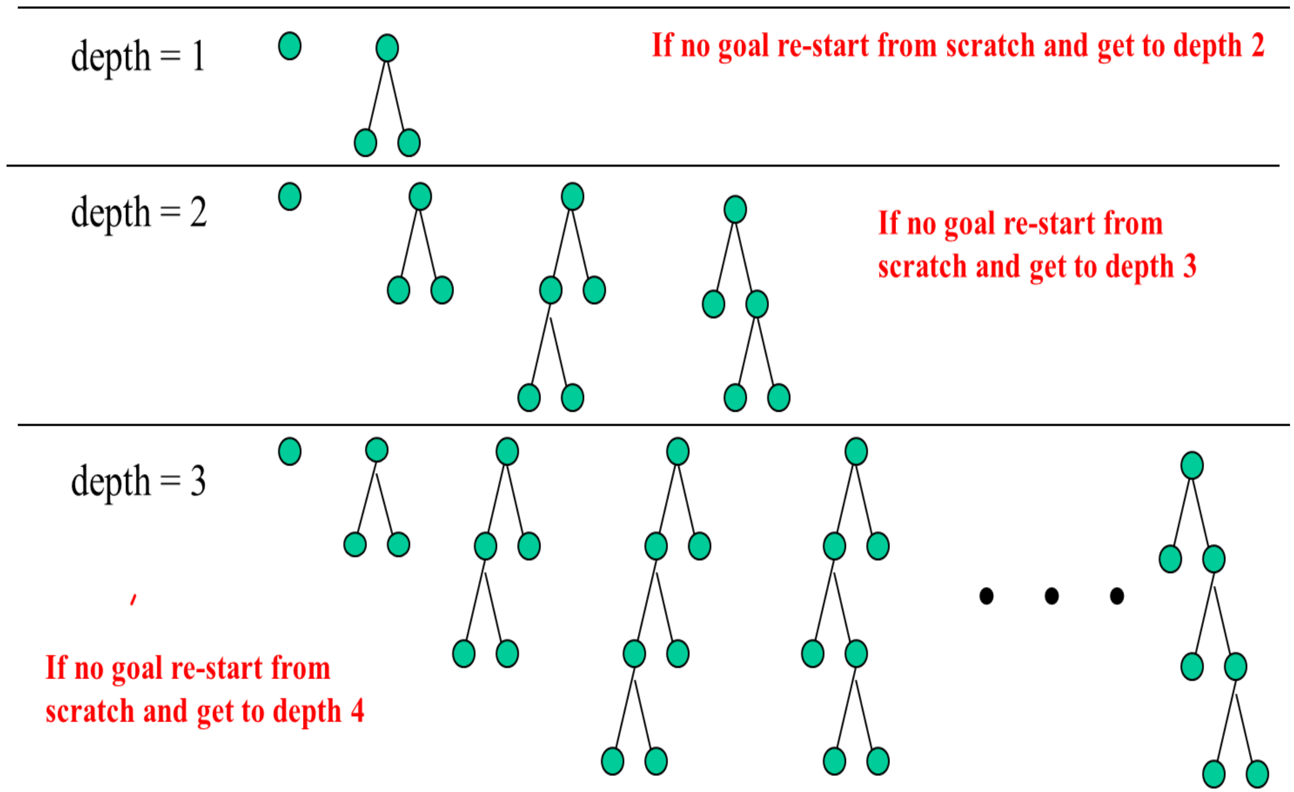
# Analysis of Iterative Deepening DFS (IDS)

- Time complexity: we showed that it is still $O(b^m)$, with limited overhead compared to BSF
- Space complexity: it does DFS

- Complete?

- Optimal?

DFS vs. BFS




| | Complete | Optimal | Time | Space |
|---|---|---|---|---|
| | | | | |

| | | | | |
|---|---|---|---|---|
| DFS | NO | NO | $O(b^m)$ | $O(bm)$ |
| BFS | YES | YES | $O(b^m)$ | $O(b^m)$ |
| IDS | YES | YES | $O(b^m)$ | $O(bm)$ |

But what if we have costs associated with search actions (arcs in the search space?

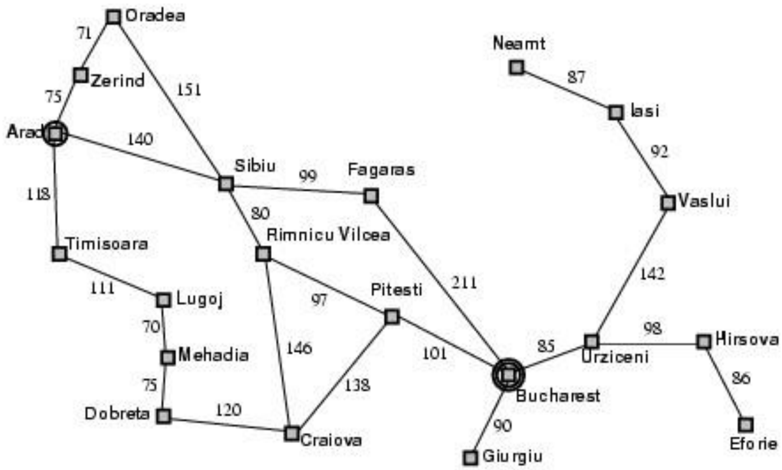
# Lecture Overview

- Recap of Lecture 4 • Least Cost First Search • Heuristic (Informed) Search
    - Best First
    - A*
    - Branch and Bound (time permitting)

# Search with Costs

Def.: The cost of a path is the sum of the costs of its arcs

$$\text{cost}(\langle n_0,\ldots,n_k \rangle) = \sum_{i=1}^{k} \text{cost}(\langle n_{i-1},n_i \rangle)$$




In this setting we usually want to find the solution that minimizes cost

Def.: A search algorithm is optimal if when it finds a solution, it is the best one: it has the lowest path cost



# Lowest-Cost-First Search (LCFS)

- Lowest-cost-first search finds the path with the lowest cost to a goal node

- At each stage, it selects the path with the lowest cost on the frontier.

- The frontier is implemented as a priority queue ordered by path cost.

Let’s see how this works in AIspace: in the Search Applet toolbar

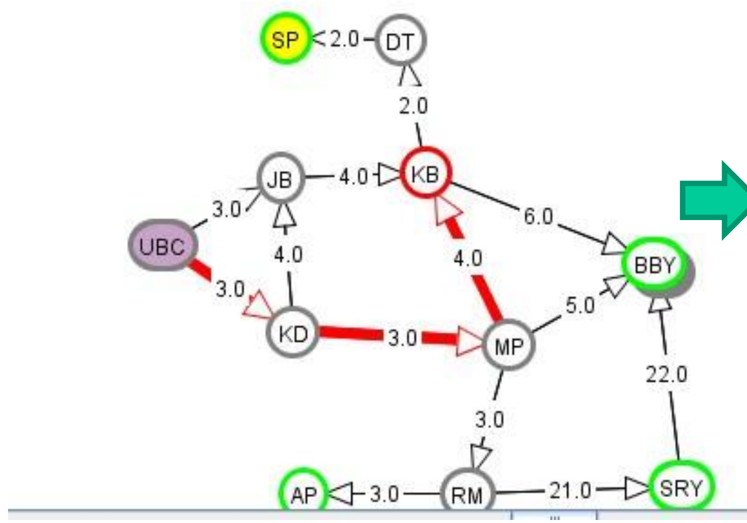
- select  the “Vancouver Neighborhood Graph” problem
- set “Search Options -> Search Algorithms” to “Lowest-Cost-First ”.
- select “Show Edge Costs” under “View”
- Create a new arc from UBC to SP with cost 20 and run LCFS

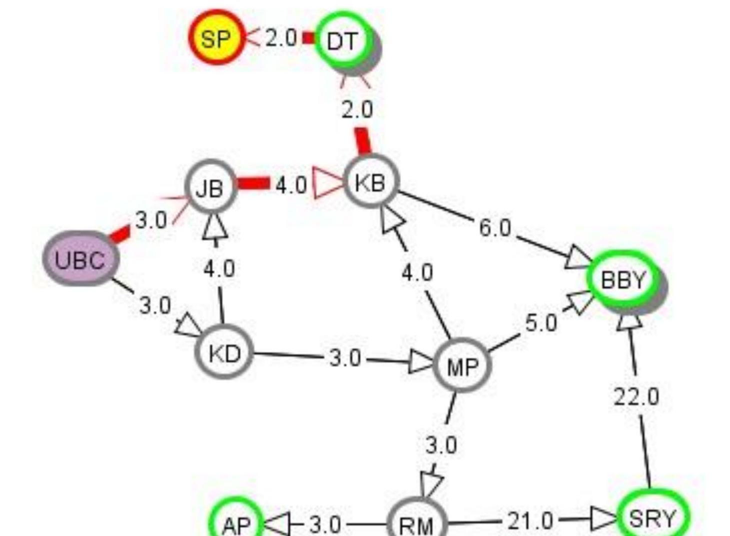


# Lowest-Cost-First Search (LCFS)

- Lowest-cost-first search finds the path with the lowest cost to a goal node

- The frontier is implemented as a priority queue ordered by path cost.




**Algorithm Selected: Lowest Cost First**

CURRENT PATH:
UBC --> KD --> MP --> KB
NEW FRONTIER:

| | | |
|---|---|---|
| Node: BBY | Path Cost: 11.0 | Path: UBC --> KD --> MP --> BBY |
| Node: KB | Path Cost: 11.0 | Path: UBC --> KD --> JB --> KB |
| Node: SP | Path Cost: 11.0 | Path: UBC --> JB --> KB --> DT --> SP |
| Node: AP | Path Cost: 12.0 | Path: UBC --> KD --> MP --> RM --> AP |
| Node: BBY | Path Cost: 13.0 | Path: UBC --> JB --> KB --> BBY |
| Node: SRY | Path Cost: 30.0 | Path: UBC --> KD --> MP --> RM --> SRY |

**Algorithm Selected: Lowest Cost First**

CURRENT PATH:
UBC --> JB --> KB --> DT --> SP (Goal)
Path to last Goal Node: UBC --> JB --> KB --> DT --> SP (Goal) Cost: 11.0
Nodes expanded: 12

NEW FRONTIER:

| | | |
|---|---|---|
| Node: AP | Path Cost: 12.0 | Path: UBC --> KD --> MP --> RM --> AP |
| Node: DT | Path Cost: 12.0 | Path: UBC --> KD --> MP --> KB --> DT |
| Node: BBY | Path Cost: 13.0 | Path: UBC --> JB --> KB --> BBY |
| Node: DT | Path Cost: 13.0 | Path: UBC --> KD --> JB --> KB --> DT |
| Node: BBY | Path Cost: 16.0 | Path: UBC --> KD --> MP --> KB --> BBY |
| Node: BBY | Path Cost: 17.0 | Path: UBC --> KD --> JB --> KB --> BBY |
| Node: SRY | Path Cost: 30.0 | Path: UBC --> KD --> MP --> RM --> SRY |

- At each stage, it selects the path with the lowest cost on the frontier.

- When arc costs are equal LCFS is equivalent to.

A. DFS

B. BFS

C. IDS

D. None of the Above



- When arc costs are equal LCFS is equivalent to.

A. DFS

B. BFS

C. IDS

D. None of the Above

# Analysis of Lowest-Cost Search (1)

- Is LCFS complete?
- not in general: for instance, a cycle with zero or negative arc costs could be followed forever.

see how this works in AIspace:

- e.g, add arc with cost -20 to the simple search graph from N4 to S in Simple Search Tree

- yes, as long as arc costs are strictly positive, greater than a given constant ε*

*If costs along an infinite path can become infinitively small, their sum can be finite (e.g. $\sum_{i=1}^{\infty} \frac{1}{2^i} < 1$ series ) and the path can trap LCFS



# Analysis of Lowest-Cost Search (1)

- Is LCFS complete?

- not in general: for instance, a cycle with zero or negative arc costs could be followed forever.

see how this works in AIspace:

- e.g, add arc with cost -20 to the simple search graph from N4 to S in Simple Search Tree

- yes, as long as arc costs are strictly positive, greater than a given constant ε*
- Is LCFS optimal?

*If costs along an infinite path can become infinitively small, their sum can be finite (e.g. series $\sum_{i=1}^{\infty} \frac{1}{2^i} < 1$) and the path can trap LCFS

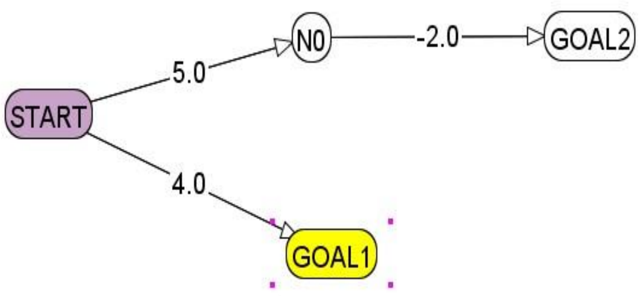
# Analysis of Lowest-Cost Search (1)

- Is LCFS complete?
- not in general: for instance, a cycle with zero or negative arc costs could be followed forever.

see how this works in AIspace:

- e.g, add arc with cost -20 to the simple search graph from N4 to S

- yes, as long as arc costs are strictly positive yes, greater than a given constant $\varepsilon$ *




- Is LCFS optimal?
- Not in general.

- Arc costs could be negative: a path that initially looks high-cost could end up getting a ``refund''.
- However, LCFS isoptimal if arc costs are guaranteed to be ≥ 0

*If costs along an infinite path can become infinitively small, their sum can be finite (e.g. series $\sum_{i=1}^{\infty} \frac{1}{2^i} < 1$)

# Analysis of Lowest-Cost Search

- Time complexity: if the maximum path length is m and the maximum branching factor is b
- The time complexity is $O(b^m)$
- In worst case, must examine every node in the tree because it generates all paths from the start that cost less than the cost of the solution

# Analysis of Lowest-Cost Search

- Space complexity
- Space complexity is $O(b^m)$:
- E.g. uniform cost: just like BFS, in worst case frontier has to store all nodes that are m-1steps away from the start node

# Summary of Uninformed Search

| | Complete | Optimal | Time | Space |
|---|---|---|---|---|
| DFS | N | N | $O(b^m)$ | $O(mb)$ |
| BFS | Y | Y (shortest) | $O(b^m)$ | $O(b^m)$ |
| IDS | Y | Y (shortest) | $O(b^m)$ | $O(mb)$ |

| | | | | |
|---|---|---|---|---|
| LCFS | Y<br>Costs > ε > 0 | Y<br>(Least Cost)<br>Costs >=0 | $O(b^m)$ | $O(b^m)$ |

Slide

# Summary of Uninformed Search (cont.)

- Why are all the search strategies seen so far are called uninformed?
- Because they do not consider any information about the states and the goals to decide which path to expand first on the frontier
- They are blind to the goal

- In other words, they are general and do not take into account the specific nature of the problem.

Slide

# Lecture Overview

- Recap of Lecture 4 • Least Cost First Search • Heuristic (Informed) Search
    - Best First

- A*
- Branch and Bound (time permitting)

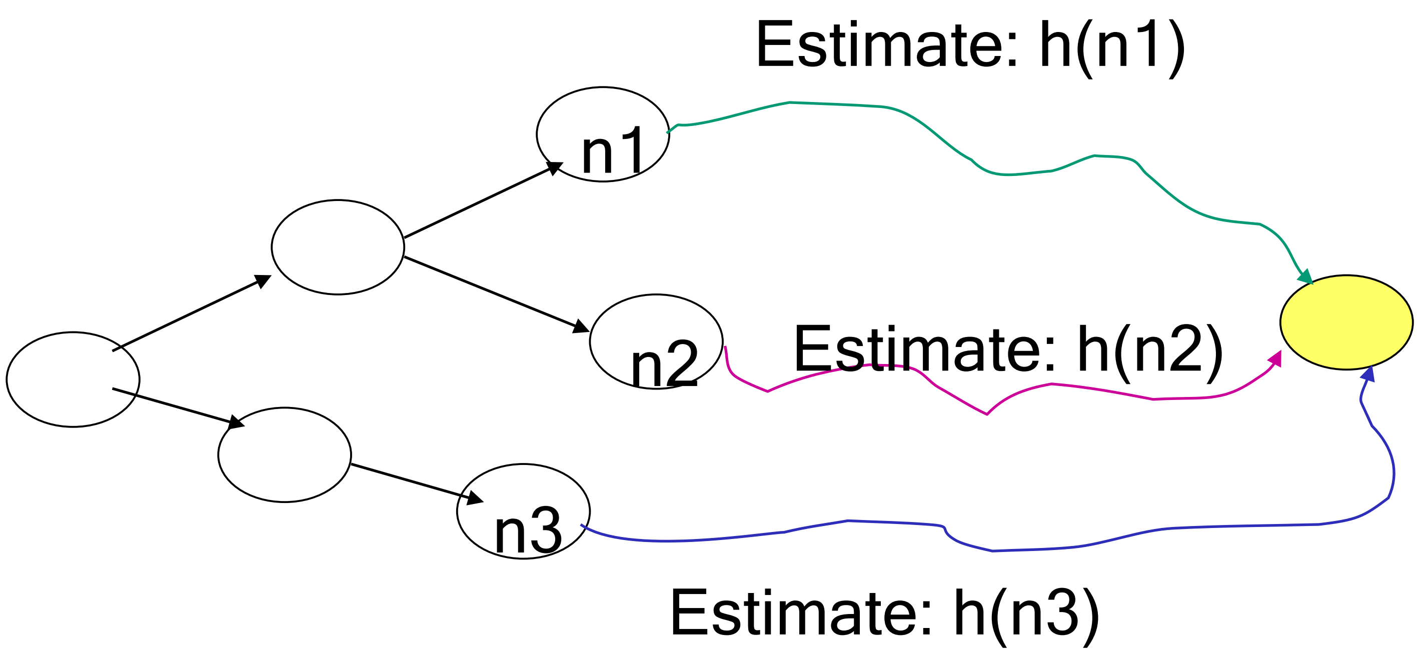
# Heuristic Search

- Blind search algorithms do not take into account the goal until they are at a goal node.
- Often there is extra knowledge that can be used to guide the search:
  - an estimate of the distance/cost from node nto a goal node.

- This estimate is called a search heuristic.

# More formally

Def.: A search heuristic h(n)is an estimate of the cost of the optimal (cheapest) path from node nto a goal node.

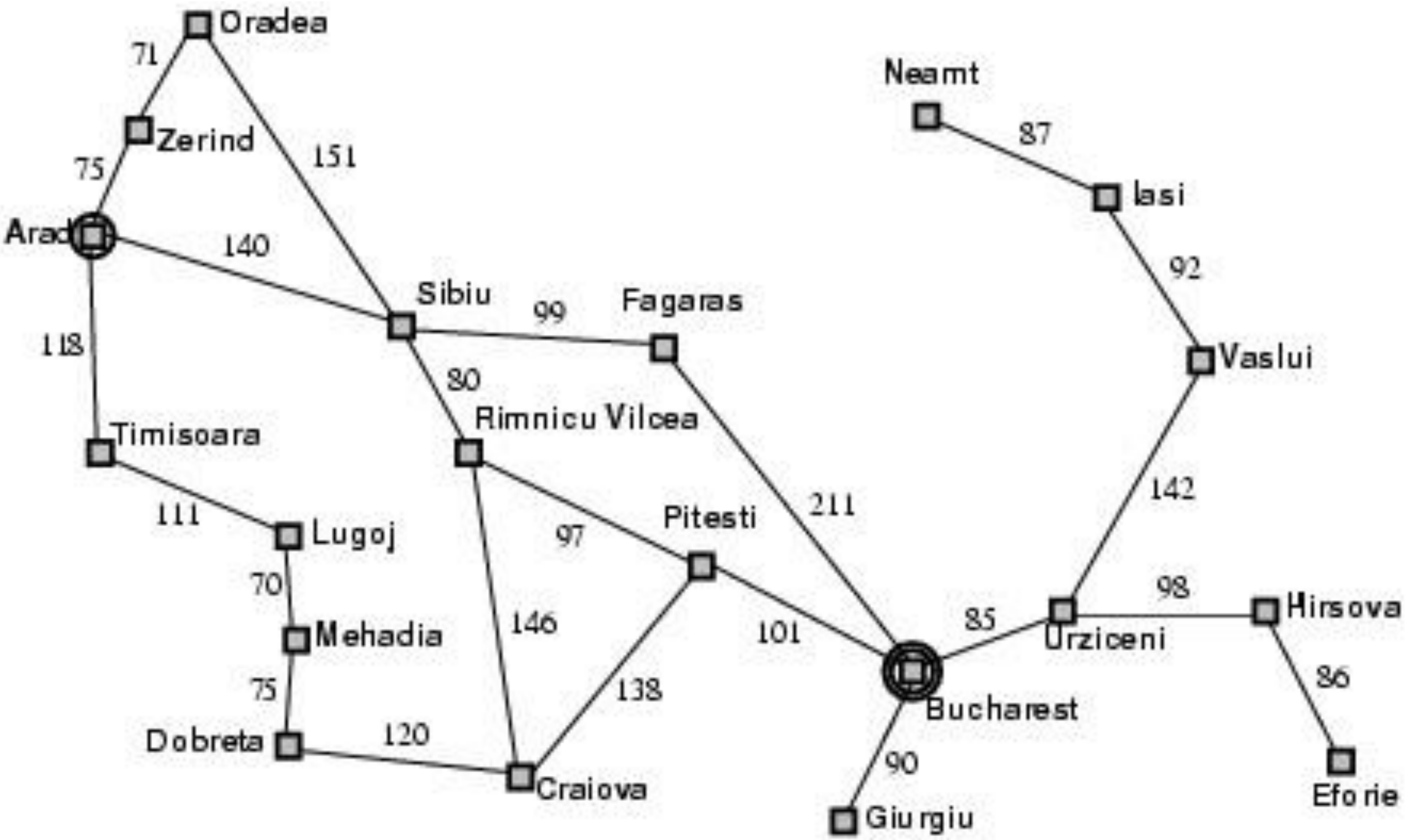


- hcan be extended to paths: $h(\langle n_0,\ldots,n_k\rangle) = h(n_k)$
- h(n)should leverage readily obtainable information (easy to compute) about a node.

Slide

## Example: finding routes

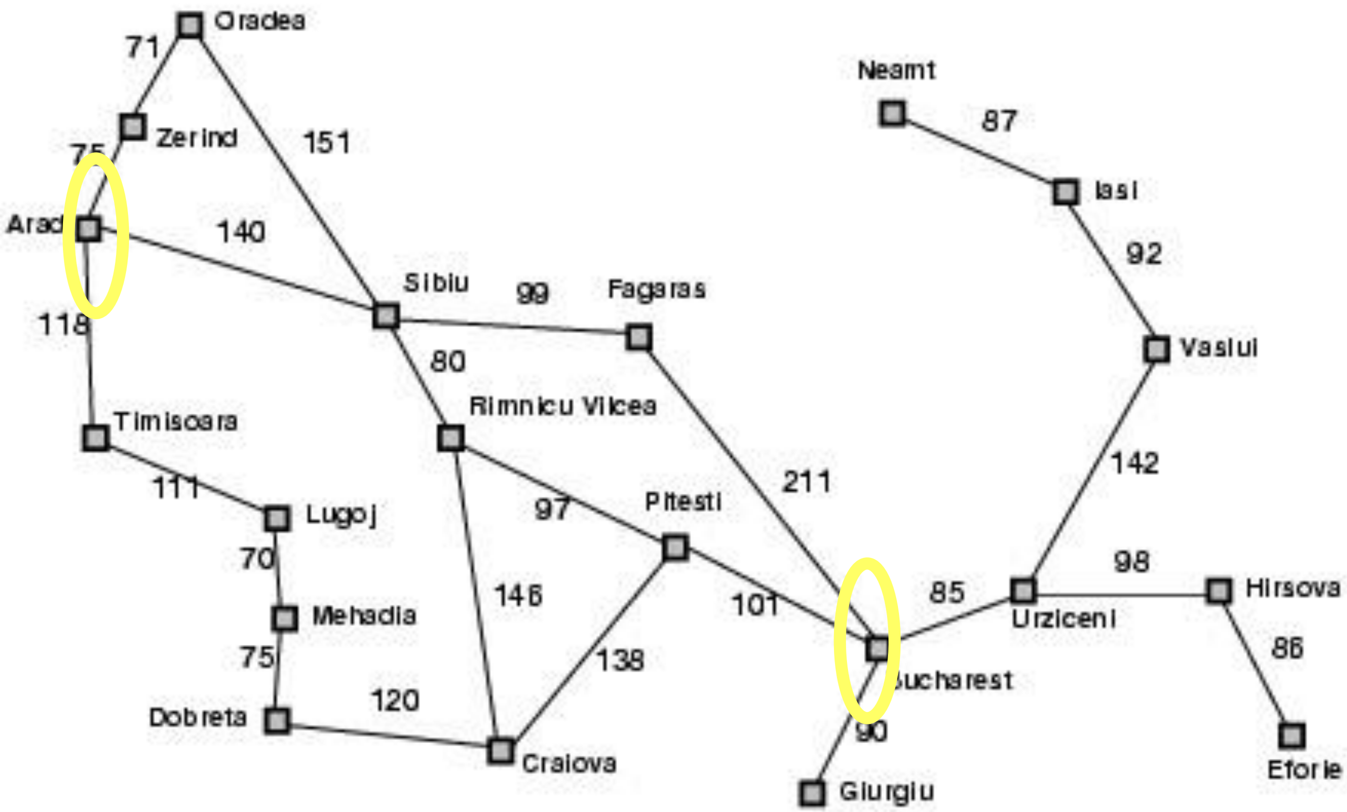
- What could we use as h(n)?




# Example: finding routes

- What could we use as h(n)? E.g., the straight-line

# (Euclidian) distance between source and goal node




Straight–line distance to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 176 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 10 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

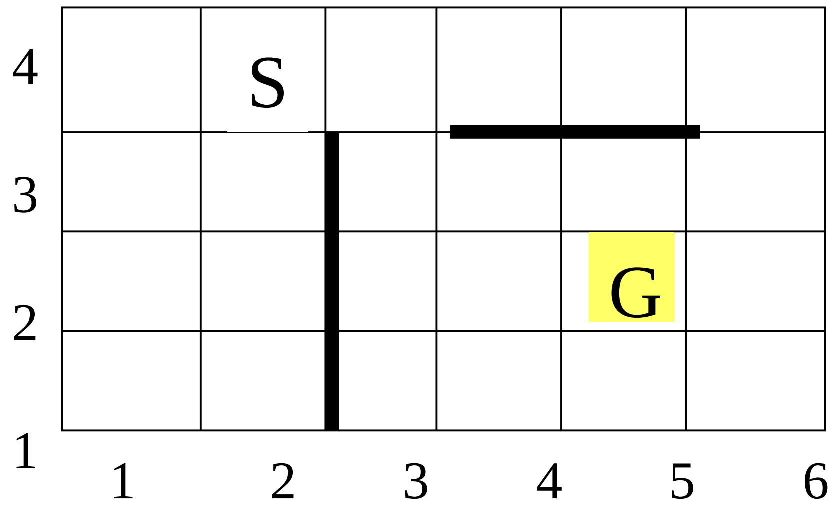
# Example 2

Search problem: robot has to find a route from start to goal location on a grid with obstacles

Actions: move up, down, left, right from tile to tile

Cost : number of moves

Possible h(n)?



# Example 2

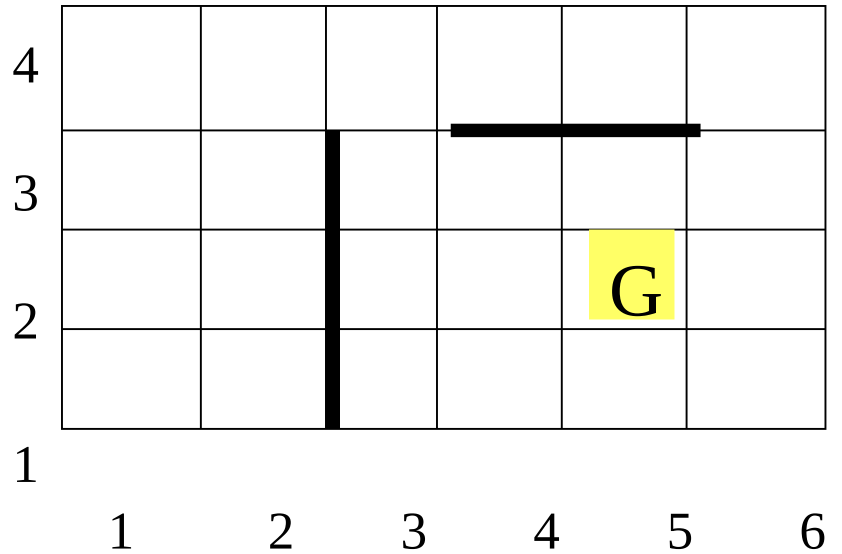
Search problem: robot has to find a route from start to goal location on a grid with obstacles

Actions: move up, down, left, right from tile to tile

Cost : number of moves

Possible h(n)? Manhattan distance ($L_1$ distance) between two points (x1, y1), (x2, y2):

- sum of the (absolute) difference of their coordinates |x2 - x1| + |y2 - y1|




Slide

# Lecture Overview

• Recap of Lecture 4 • Least Cost First Search • Heuristic (Informed) Search

- Best First
- A*
- Branch and Bound (time permitting)

# Best First Search (BestFS)

- Idea: always choose the path on the frontier with the smallest h value.
- BestFS treats the frontier as a priority queue ordered by h.
- Greedy approach: expand path whose last node seems closest to the goal - chose the solution that is locally the best.

Let's see how this works in AIspace: in the Search Applet toolbar

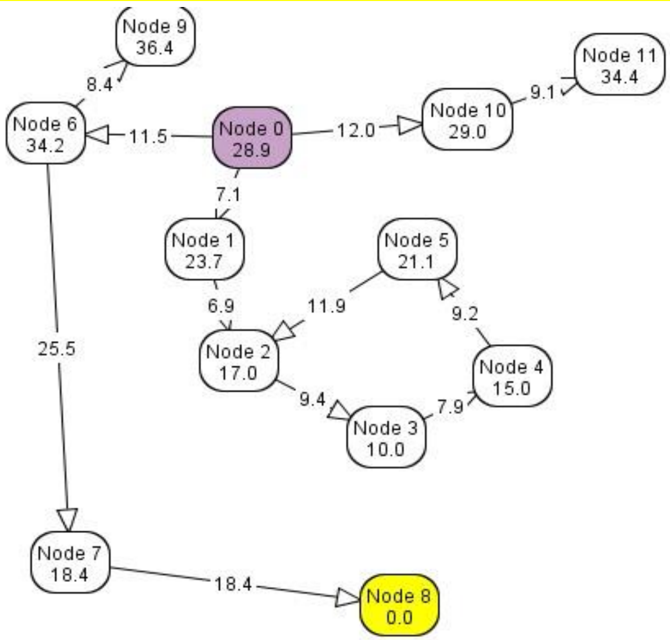
- select the "Vancouver Neighborhood Graph" problem • set "Search Options -> Search Algorithms" to "Best-First ".
- select "Show Node Heuristics" under "View"

- compare number of nodes expanded by BestFS and LCFS

# Analysis of BestFS

- Complete?

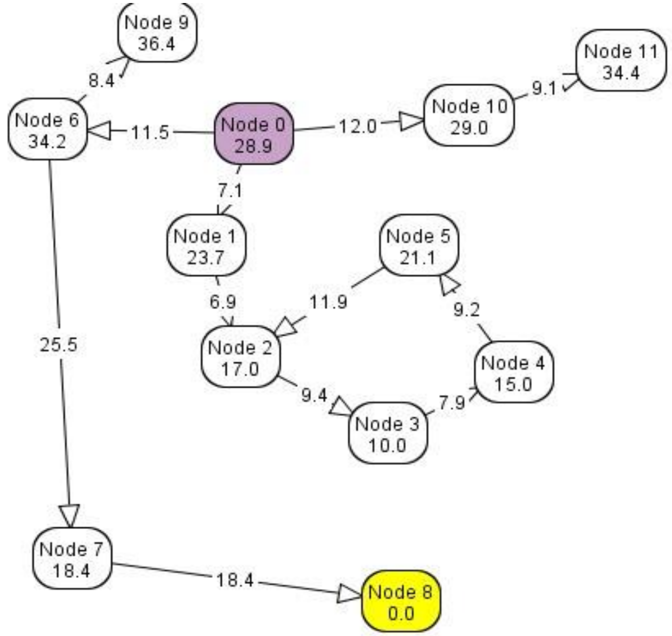
A. YES B. NO




- Complete?

# Analysis of BestFS

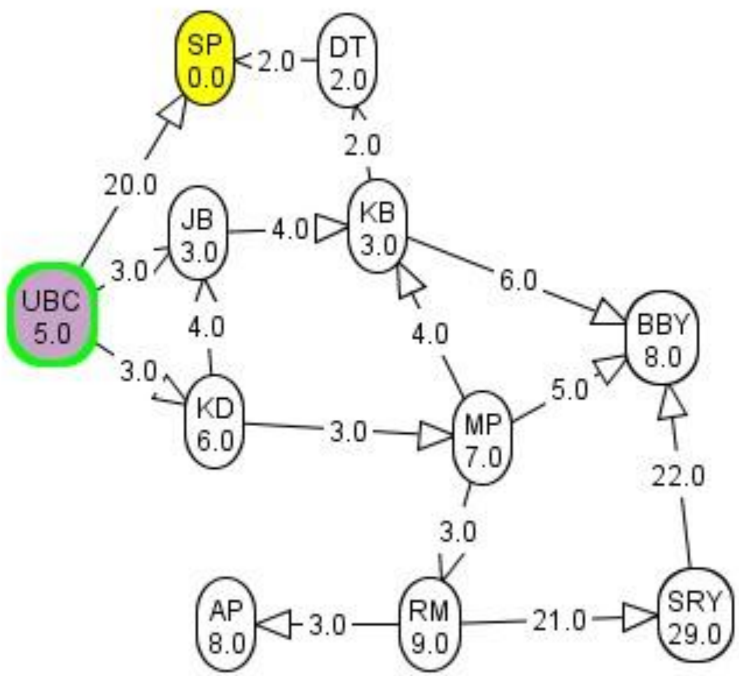
B. NO




- See the “misleading heuristics demo” example in AISPACE
- Optimal?

# Analysis of BestFS

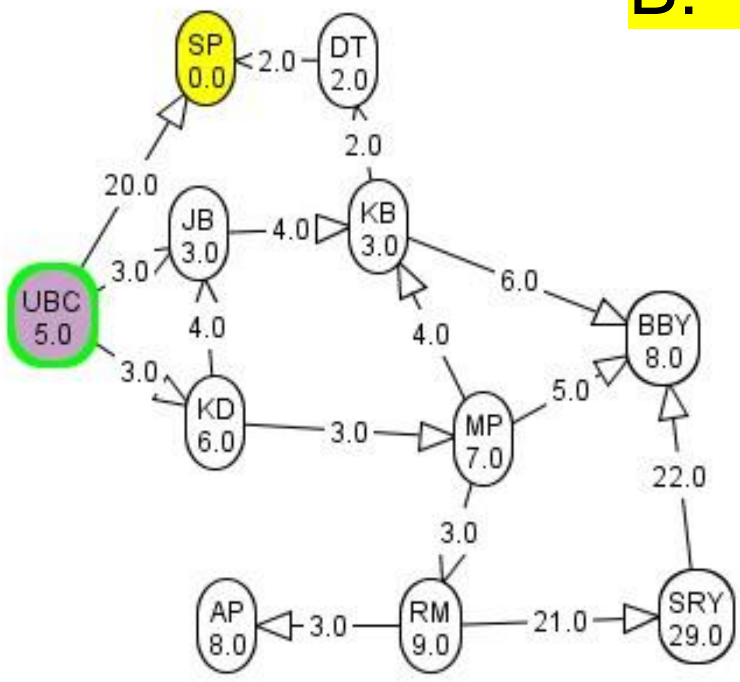
A. YES    B. NO




- Optimal?

# Analysis of BestFS

B. NO




- Try this example in AISPACE or example "ex-best.txt" from schedule page (save it and then load using "load from file" option)

# Analysis of BestFS

- Time and space Complexity: $O(b^m)$
- Worst case (bad h(n)): has to explore all nodes and keep related partial paths on the frontier

- Why would one want to use Best First Search ?
- Because if the heuristics is good it can find the solution very fast.
- For another example, see AIspace, Delivery problem graph with C1 linked to o123 (cost 3.0)

# What's Next?

- Thus, having estimates of the distance to the goal can speed things up a lot
- but by itself it can also mislead the search (i.e. Best First Search)

- On the other hand, taking only path costs into account allows LCSF to find the optimal solution
- but the search process is still uniformed as far as distance to the goal goes.

## How can we more effectively use h(p) and cost(p)?

iclicker.

Shall we select from the frontier the path pwith:

A. Lowest cost(p) – h(p)

B. Highest cost(p) – h(p)

C. Highest cost(p)+h(p)

D. Lowest cost(p)+h(p)



How can we more effectively use h(p) and cost(p)?

Shall we select from the frontier the path pwith:

A. Lowest cost(p) – h(p)

B. Highest cost(p) – h(p)

C. Highest cost(p)+h(p)

D. Lowest cost(p)+h(p)

# Lecture Overview

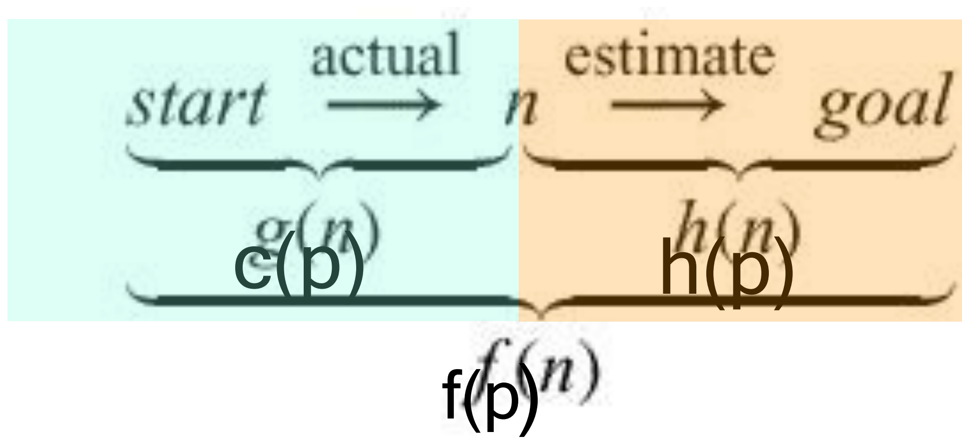
• Recap of Lecture 4 • Least Cost First Search • Heuristic (Informed) Search

- Best First
- A*
- Branch and Bound (time permitting)

# A* Search

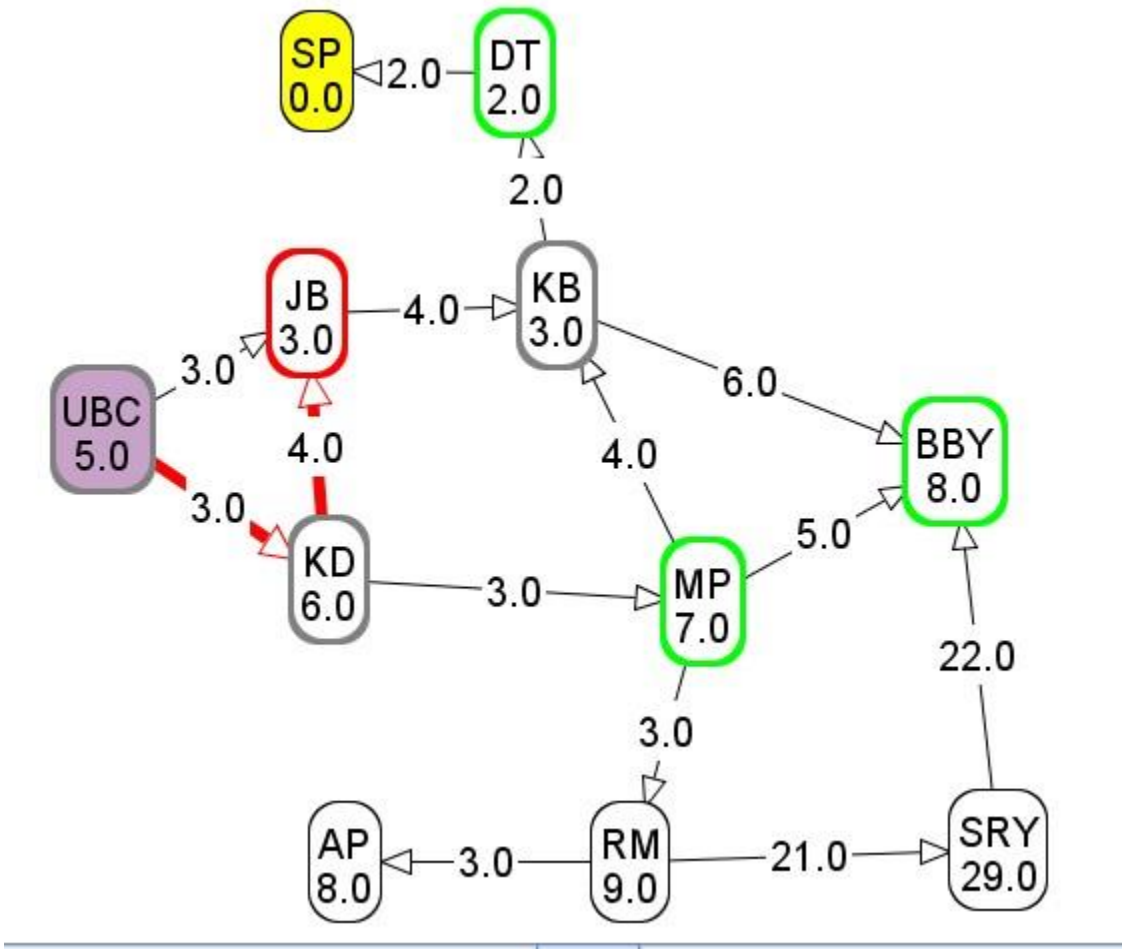
- A* search takes into account both

- the cost of the path pto a node c(p)
- the heuristic value of that path h(p)(i.e. the h value of the node n at the end of p)
- Let f(p) = c(p) + h(p).
- f(p)is an estimate of the cost of a path from the start to a goal via p




A* always chooses the path on the frontier with the lowest estimated distance from the start to a goal node constrained to go via that path.

# Computing f-values



f value of ubc → kd

A.6

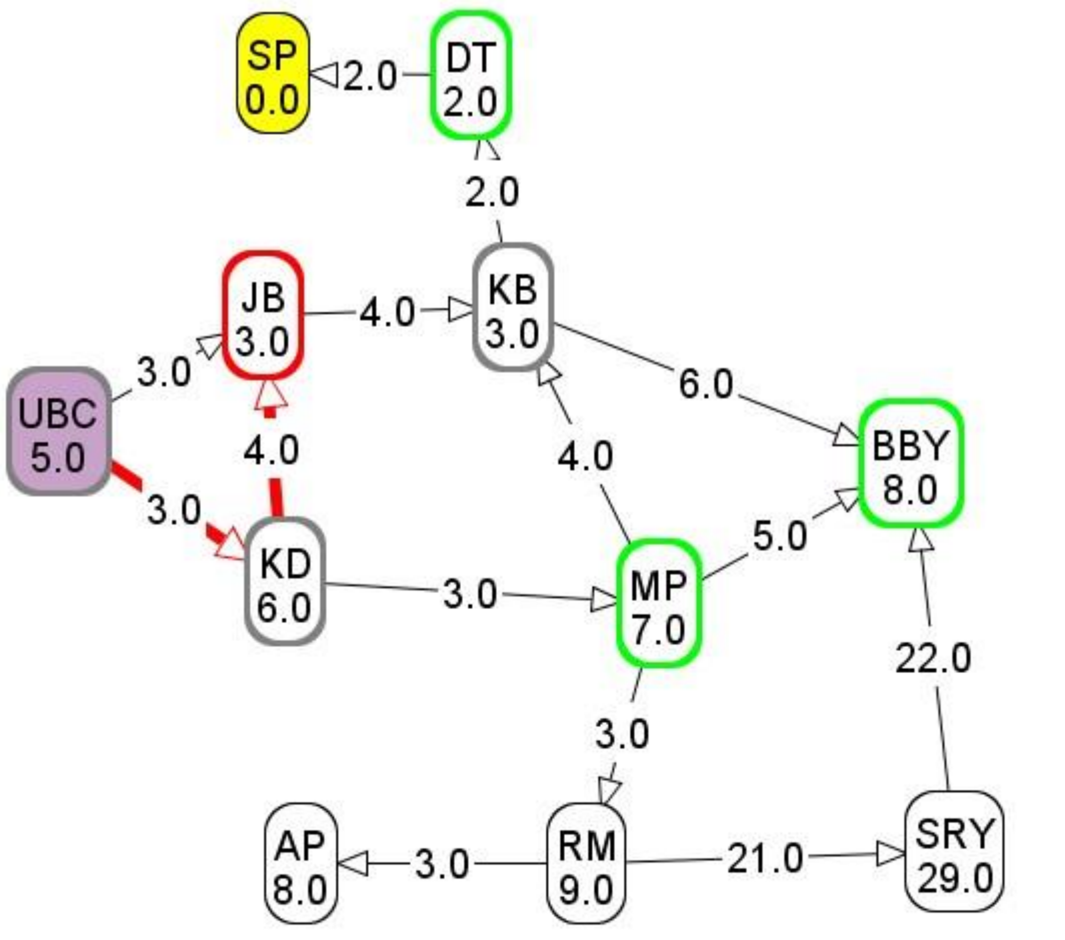
B. 9

C.10

→ jb?

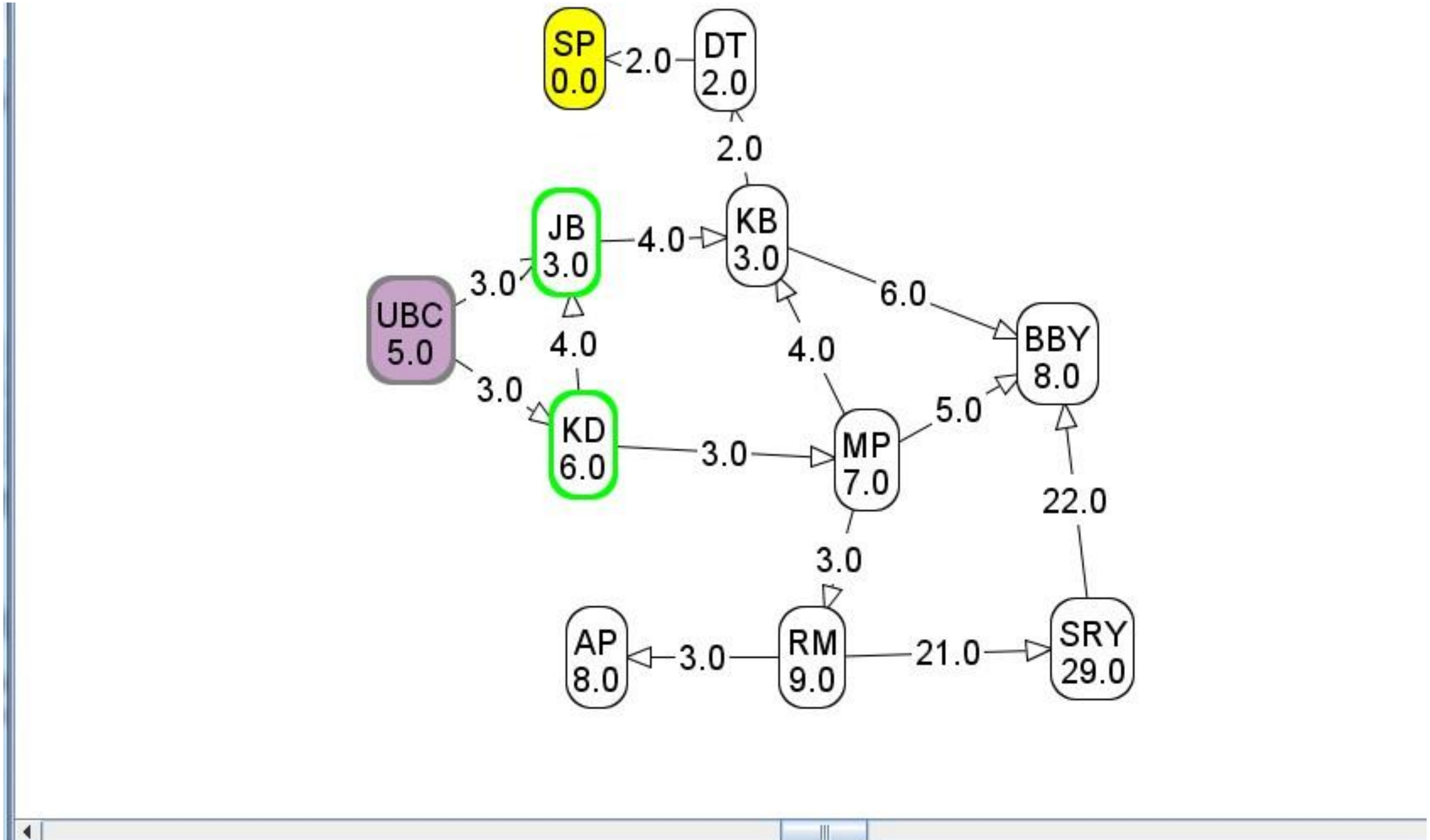
D. 11

# Computing f-valeues



f value of ubc → kd → jb?

C. 10




**Algorithm Selected: A***

PREVIOUS PATH2:

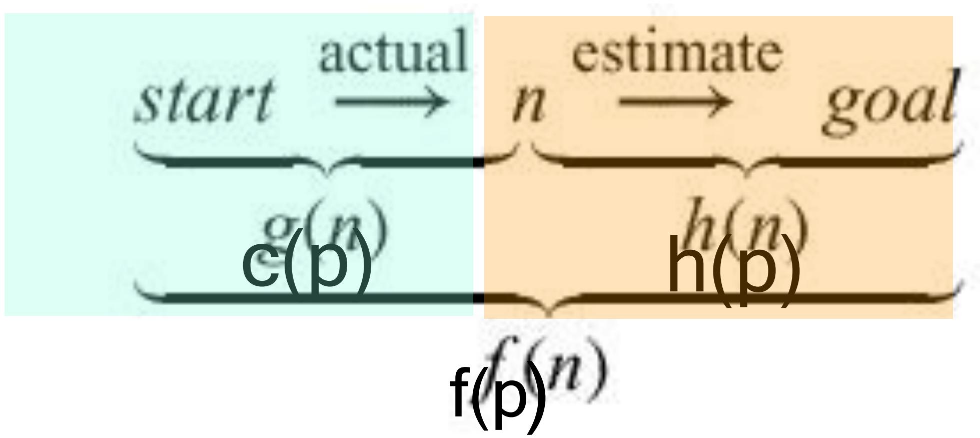
UBC

NEW FRONTIER:

Node: JB Path Cost: 3.0 h(JB): 3.0 f-value: 6.0 Path: UBC --> JB

# A* Search

- A* search takes into account both
- the cost of the path to a node c(p) • the heuristic value of that path h(p).
- Let f(p) = c(p) + h(p).
- f(p)is an estimate of the cost of a path from the start to a goal vi



A* always chooses the path on the frontier with the lowest estimated distance from the start to a goal node constrained to go via that path.

Compare A* and LCFS on the Vancouver graph

# Optimality of A*

A* is complete (finds a solution, if one exists) and optimal (finds the optimal path to a goal) if

- arc costs are > $\varepsilon$ > 0
- h(n) is admissible

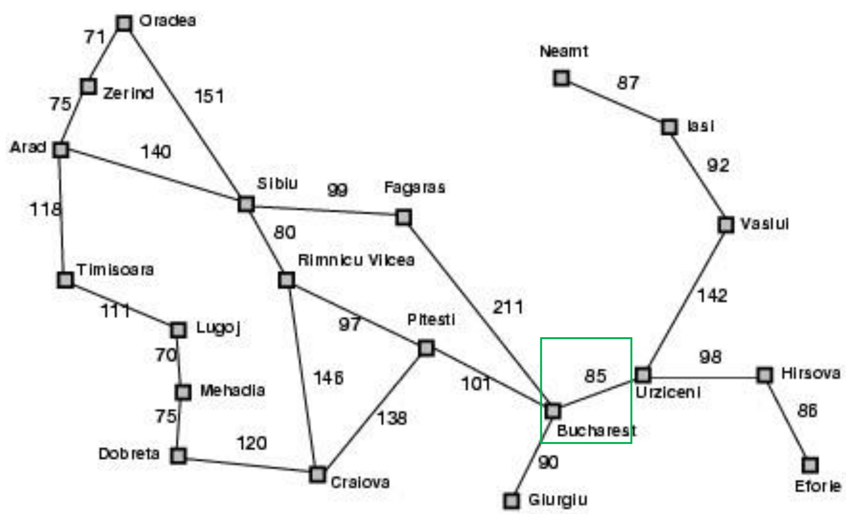
The book also mentions explicitly that the branching factor b has to be finite, which we have been assuming by default (without this condition even BFS would not be complete)

# Admissibility of a heuristic

Def.:

Let c(n) denote the cost of the optimal path from node n to any goal node. A search heuristic h(n) is called admissible if h(n) ≤ c(n) for all nodes n, i.e. if for all nodes it is an underestimate of the cost to any goal.

- Example: is the straight-line distance (SLD) admissible?




| Straight–line distance to Bucharest | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 176 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 10 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

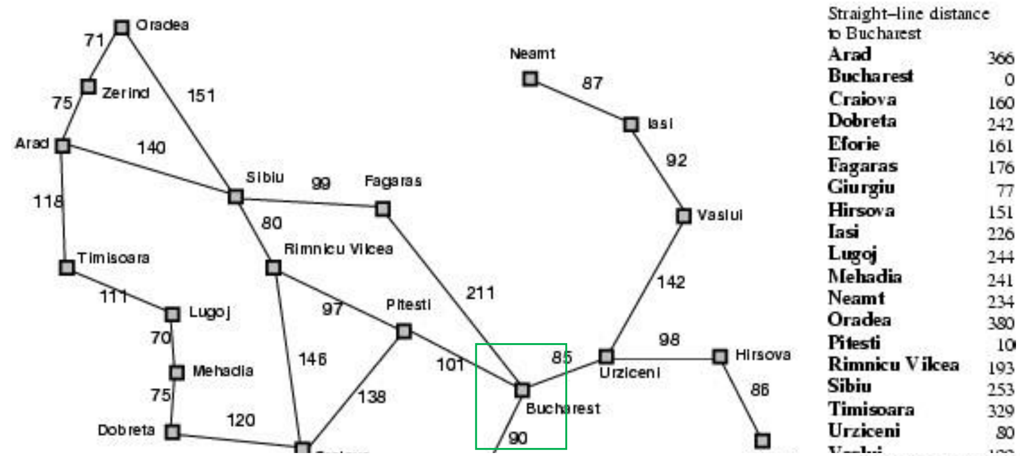
YES

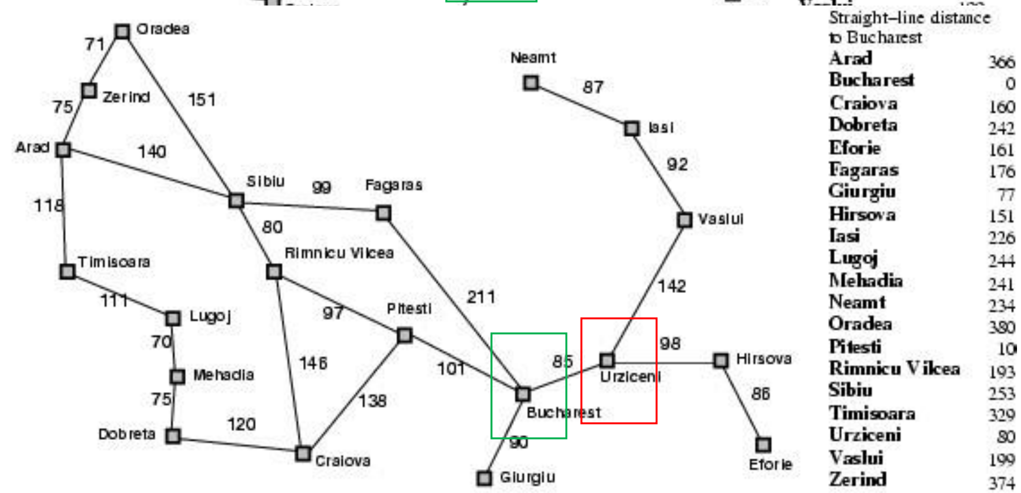
# Admissibility of a heuristic

- Example: is the straight-line distance admissible?

  - The shortest distance between two points is a line.

example:



# Admissibility of a heuristic

Def.:

Let c(n) denote the cost of the optimal path from node n to any goal node. A search heuristic h(n) is called admissible if h(n) ≤ c(n) for all nodes n, i.e. if for all nodes it is an underestimate of the cost to any goal.

the goal is Urzizeni (red box), but all we know is the straight-line distances (sld) to Bucharest (green box)

- Possible h(n) = sld(n, Bucharest) + cost(Bucharest, Urzineni)
- Admissible?

A. Yes

B. No

C. It depends

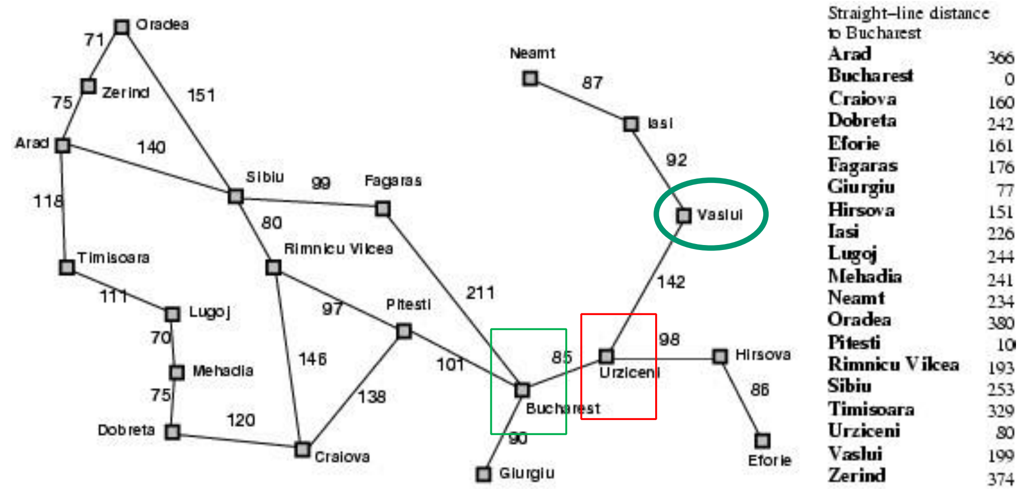
# Admissibility of a heuristic

Def.:

Let $c(n)$ denote the cost of the optimal path from node $n$ to any goal node. A search heuristic $h(n)$ is called admissible if $h(n) \le c(n)$ for all nodes $n$, i.e. if for all nodes it is an underestimate of the cost to any goal.

example:

the goal is Urzizeni (red box), but all we know is the straight-line distances to Bucharest (green box)




- Possible h(n) = sld(n, Bucharest) + cost(Bucharest, Urzineni)
- Admissible?

NO

- Actual cost of going from Vastul to Urzineni is shorter than this estimate

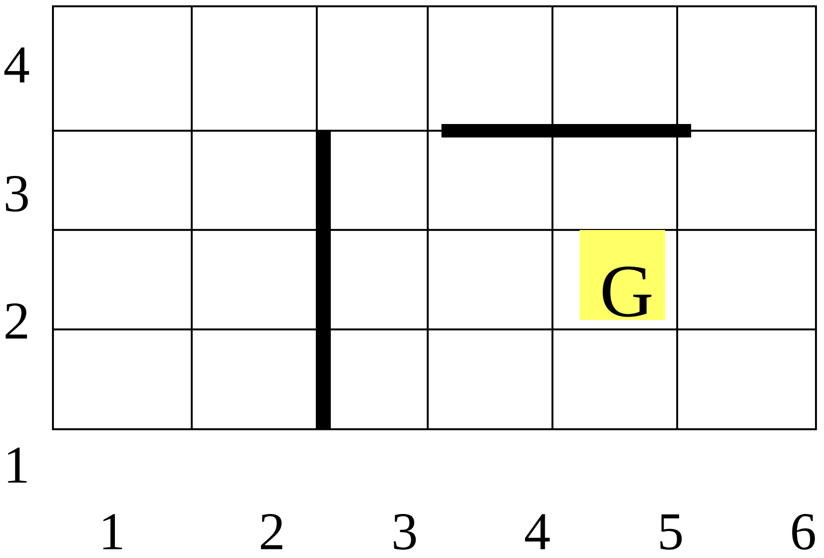
60

# Example 2

- Search problem: robot has to find a route from start to goal location on a grid with obstacles

- Actions: move up, down, left, right from tile to tile
- Cost : number of moves
- Possible h(n)? Manhattan distance ($L_1$ distance) between two points (x1, y1), (x2, y2):
- sum of the (absolute) difference of their coordinates |x2 - x1| + |y2 - y1|

ADMISSIBLE?




Slide

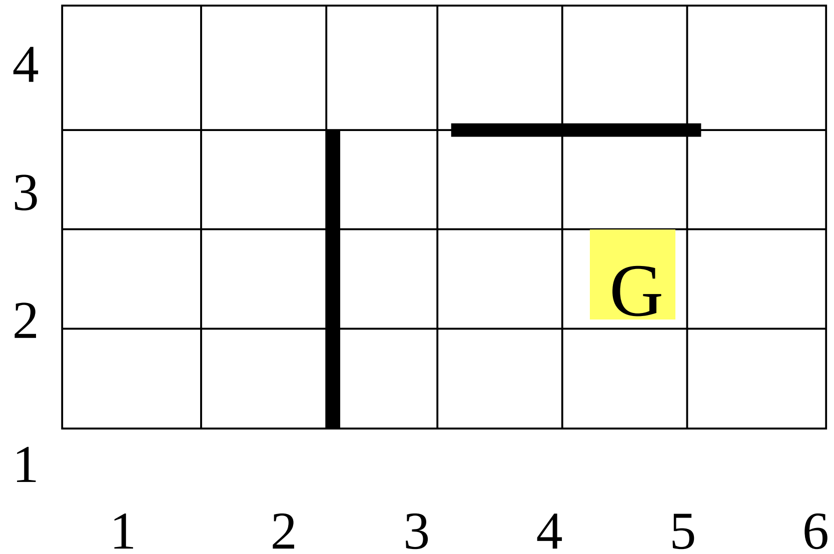
# Example 2

- Search problem: robot has to find a route from start to goal location on a grid with obstacles
- Actions: move up, down, left, right from tile to tile
- Cost : number of moves
- Possible h(n)? Manhattan distance ($L_1$ distance) between two points (x1, y1), (x2, y2):
- sum of the (absolute) difference of their coordinates

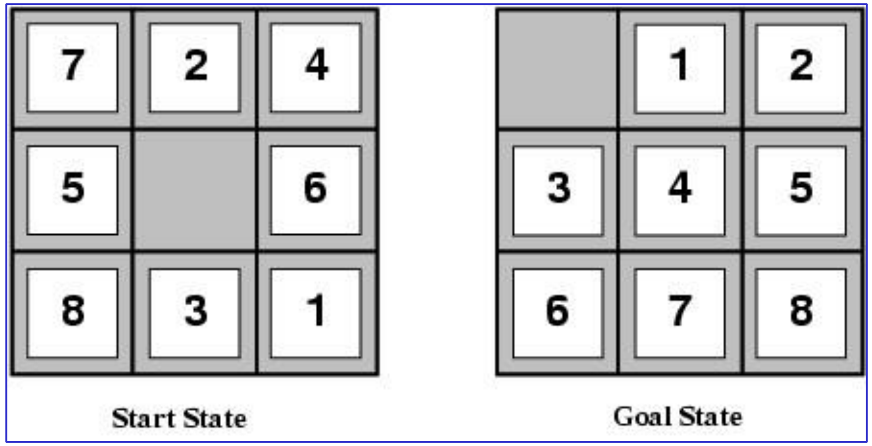
$$|x2 - x1| + |y2 - y1|$$

ADMISSIBLE?

Yes. Manhattan distance is the shortest path between any two tiles of the grid given the actions available and no walls. Including the walls will force the agent to take some extra steps to avoid them



# Heuristic Function for 8-puzzle




An admissible heuristic for the 8-puzzle is?

A. Number of misplaced tiles plus number of correctly place tiles

B. Number of misplaced tiles

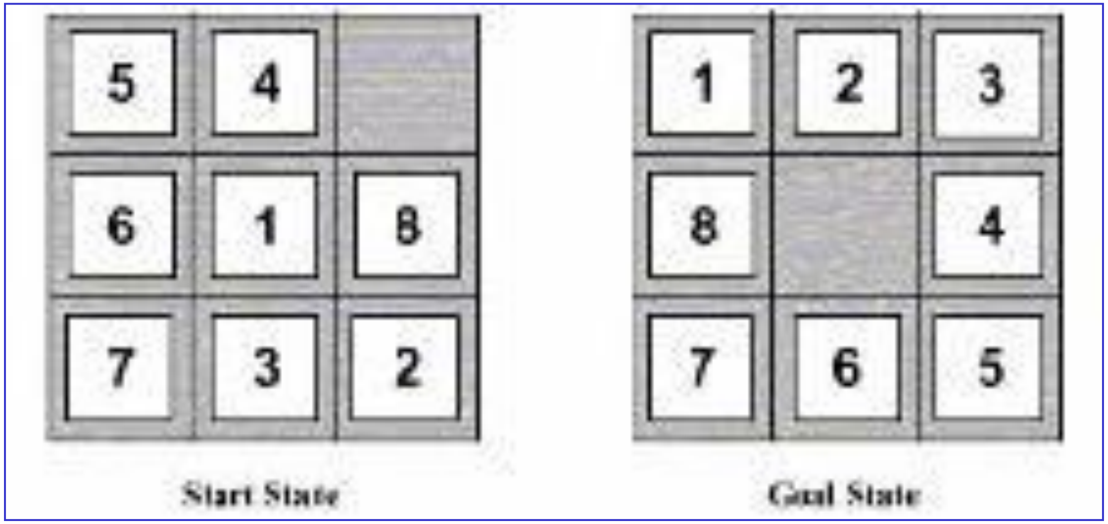
C. Number of correctly placed tiles

D. None of the above

# Example 3: Eight Puzzle

- Admissible h(n):

  Number of Misplaced Tiles: One needs at least that many moves to get the board in the goal state

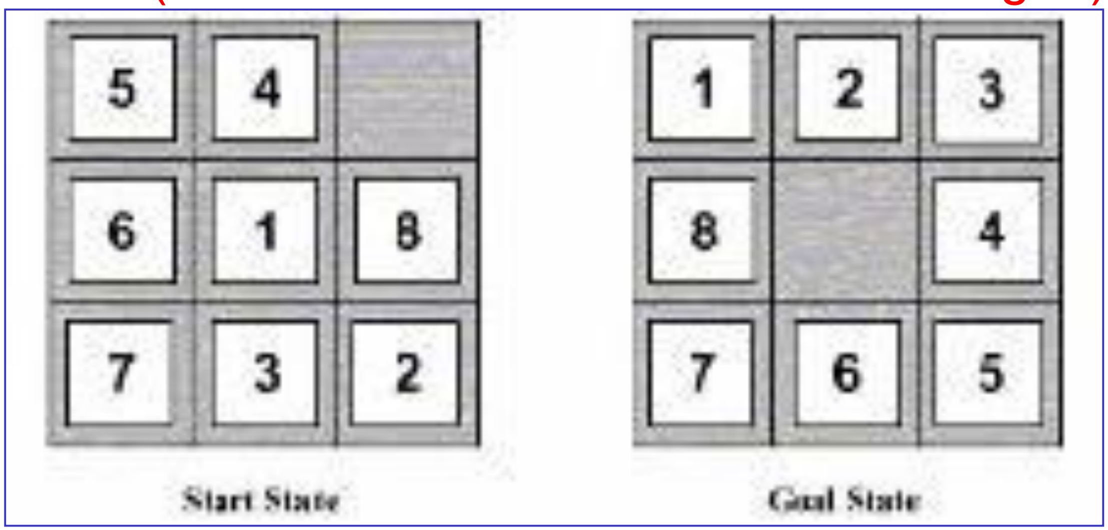


# Example 3: Eight Puzzle

A and C clearly generate overestimates (e.g. when all tiles are in the correct position with respect to the goal above, except for 4 which is in the center)

- Another possible h(n):

Sum of number of moves between each tile's current position and its goal position (we can move over other tiles in the grid)

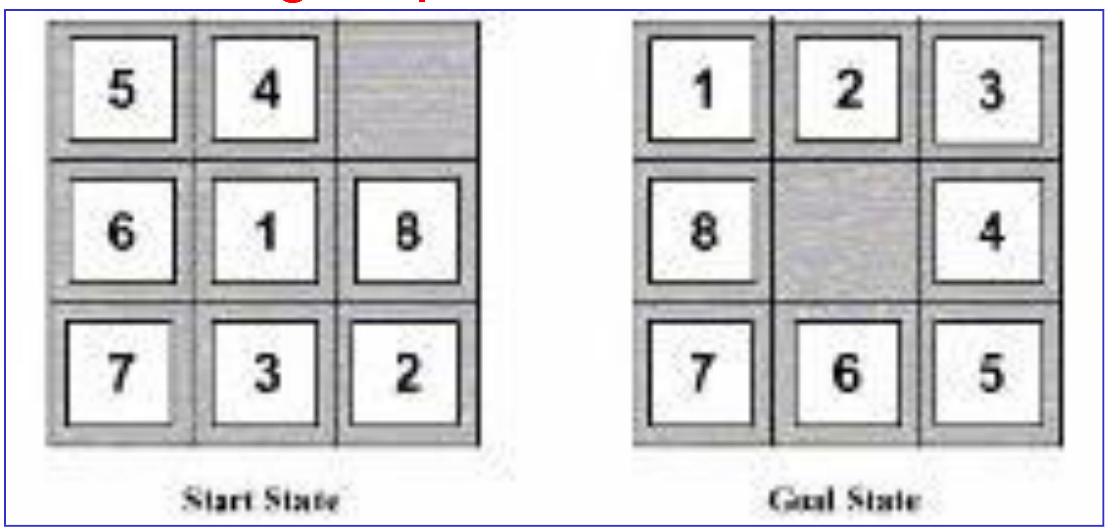



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

# Example 3: Eight Puzzle

Sum (

- Another possible h(n):
  Sum of number of moves between each tile's current position and its goal position




1 2 3 4 5 6 7 8

sum ( 2 3 3 2 4 2 0 2) = 18

# Example 3: Eight Puzzle

Admissible?

A. Yes

B. No

C. It depends

# How to Construct an Admissible Heuristic

- Identify relaxed version of the problem:
  - where one or more constraints have been dropped
  - problem with fewer restrictions on the actions
- Grid world: the agent can move through walls
- Driver: the agent can move straight
- 8 puzzle:
  - "number of misplaced tiles":
    tiles can move everywhere and occupy same spot as others
  - "sum of moves between current and goal position":
    tiles can ……………………..

Why does this lead to an admissible heuristic?

- The problem only gets easier!
- Less costly to solve it

# How to Construct a Heuristic (cont.)

- You should identify constraints which, when dropped, make the problem easy to solve
- important because heuristics are not useful if they're as hard to solve as the original problem!

This was the case in our examples

Robot: allowing the agent to move through walls. Optimal solution to this relaxed problem is Manhattan distance

Driver: allowing the agent to move straight. Optimal solution to this relaxed problem is straight-line distance

8puzzle: tiles can move anywhere. Optimal solution to this relaxed problem is number of misplaced tiles

## Learning Goal for Search

Apply basic properties of search algorithms:

- completeness, optimality, time and space complexity

| | Complete | Optimal | Time | Space |
|---|---|---|---|---|
| DFS | N (Y if no cycles) | N | $O(b^m)$ | O(mb) |
| BFS | Y | Y | $O(b^m)$ | $O(b^m)$ |

| | | | | |
|---|---|---|---|---|
| IDS | Y | Y | $O(b^m)$ | $O(mb)$ |
| LCFS (when arc costs available) | Y Costs > ε | Y Costs >=0 | $O(b^m)$ | $O(b^m)$ |
| Best First (when havailable) | | | | |

# Learning Goal for Search

Apply basic properties of search algorithms:

- completeness, optimality, time and space complexity

| | Complete | Optimal | Time | Space |
|---|---|---|---|---|
| DFS | N | N | $O(b^m)$ | $O(mb)$ |

| | | | | |
|---|---|---|---|---|
| | (Y if no cycles) | | | |
| BFS | Y | Y | $O(b^m)$ | $O(b^m)$ |
| IDS | Y | Y | $O(b^m)$ | $O(mb)$ |
| LCFS (when arc costs available) | Y Costs > ε | Y Costs >=0 | $O(b^m)$ | $O(b^m)$ |
| Best First (when havailable) | N | N | $O(b^m)$ | $O(b^m)$ |

# Learning Goals for Search (up to today)

- Apply basic properties of search algorithms:

- completeness, optimality
- time and space complexity
- Select the most appropriate search algorithms for specific problems.
- Depth-First Search  vs. Bredth-First Search vs. Iterative Deepening vs. Least-Cost-First Search, Best First  Search
- Define/read/write/trace/debug different search algorithms
- Construct heuristic functions and discuss their admissibility for specific search problems

# TO DO

- Do practice exercises 3C and 3D
- Read Ch 3.7 (More sophisticated search)

- Start working on Assignment 1! You can already do many of the questions