APPLIED ALGORITHMS

SOLVING PROBLEM BY SEARCHING

Lecture 3: Graphs and Search Strategies (cont.)
Informed (Heuristic) Search Strategies

T.S Ha Minh Hoang
Th.S Nguyen Minh Anh

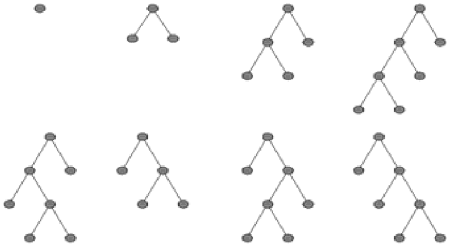
Phenikaa University

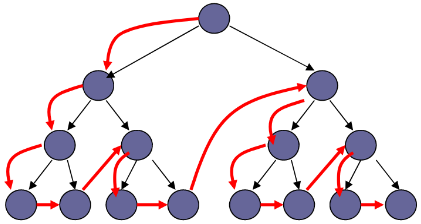
Last Update: 8th February 2023

Outline

Depth-first search (DFS)

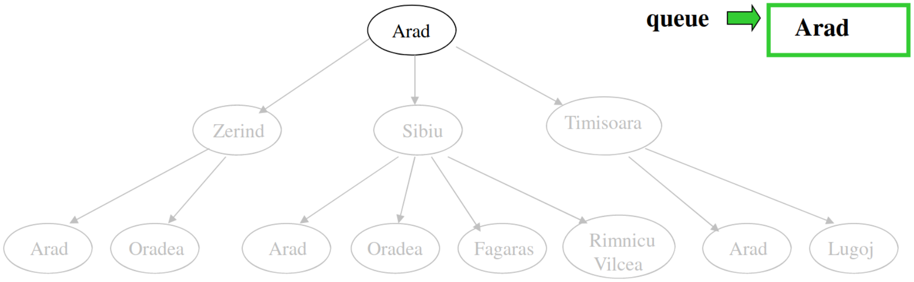
- **The deepest node is expanded first**
- Backtrack when the path cannot be further expanded



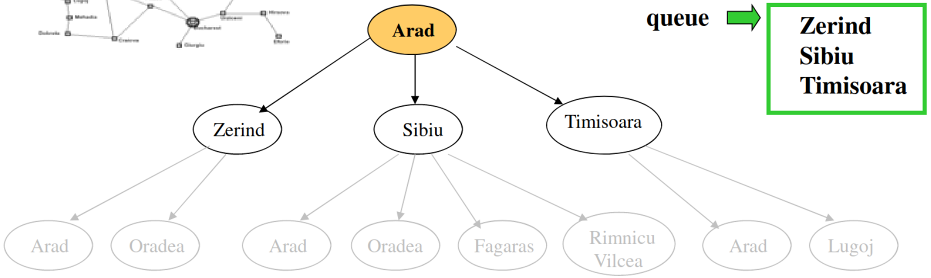


Depth-first search (DFS)

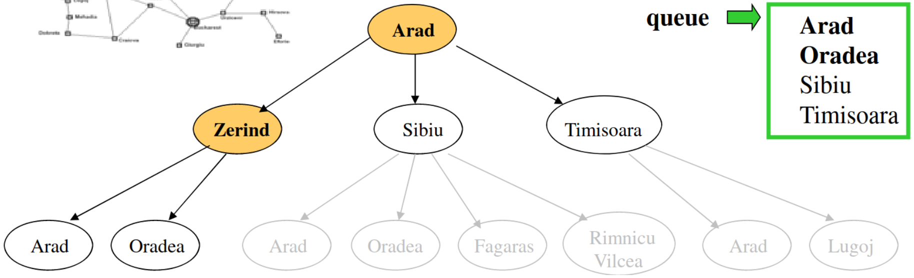
- **The deepest node is expanded first**
- Implementation: put successors to the beginning of the queue



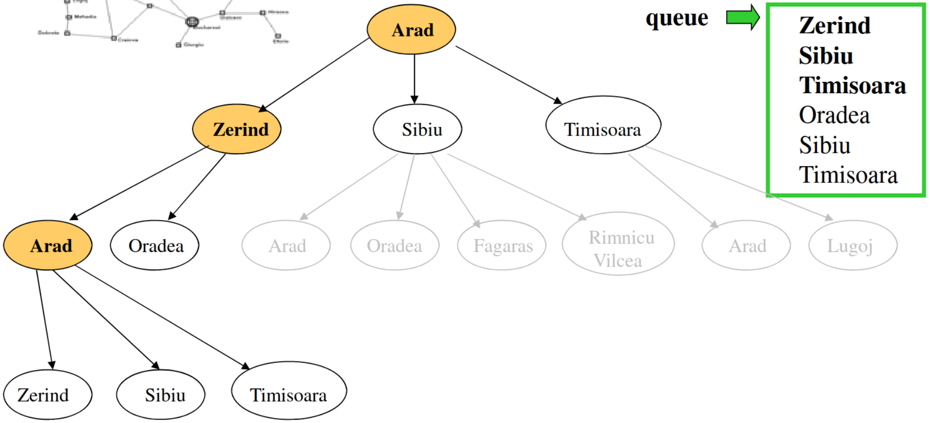
Depth-first search (DFS)



Depth-first search (DFS)



Depth-first search (DFS)



Note: Arad - Zerind - Arad cycle

Properties of depth-first search

- **Completeness: Does it always find the solution if it exists?**
- **Optimality: does it find the minimum length path ?**
- **Time complexity: ?**
- **Memory (space) complexity: ?**

Properties of depth-first search

- **Completeness: Does it always find the solution if it exists?**
  No, if we permit infinite loops.
- **Optimality: does it find the minimum length path ?**
- **Time complexity: ?**
- **Memory (space) complexity: ?**

Properties of depth-first search

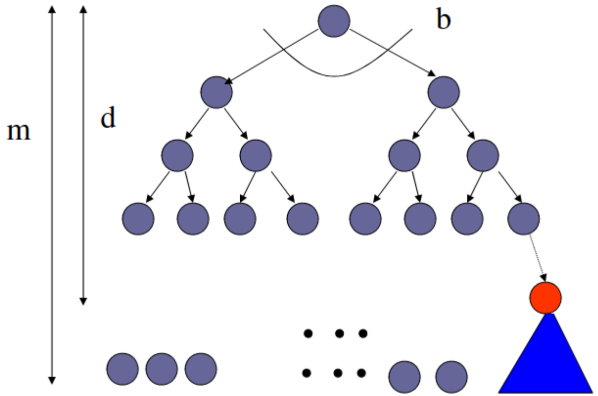
- **Completeness: Does it always find the solution if it exists?**
  No, if we permit infinite loops.
  Yes, if we prevent them
- **Optimality: does it find the minimum length path ?**

- **Time complexity: ?**
- **Memory (space) complexity: ?**

Properties of depth-first search

- **Completeness: Does it always find the solution if it exists?**
  No, if we permit infinite loops.
  Yes, if we prevent them
- **Optimality: does it find the minimum length path ?**
  No. Solution found first may not be the shortest possible.
- **Time complexity: ?**
- **Memory (space) complexity: ?**

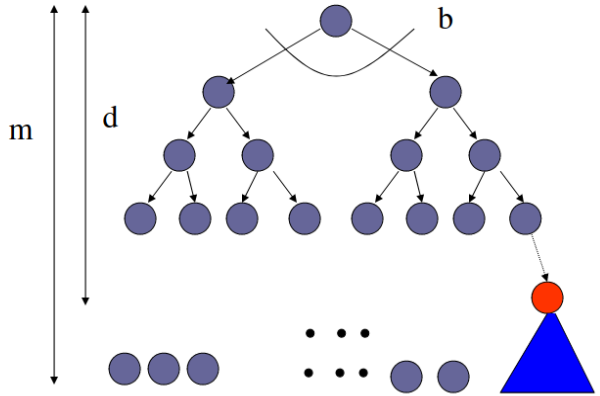
DFS - time complexity



depth	number of nodes
0	1
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
•	• •
d	2^d
•	• •
m	2^m-2^{m-d}

Complexity:

DFS - time complexity



depth	number of nodes
0	1
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
•	• •
d	2^d
•	• •
m	2^m-2^{m-d}

Complexity: $O(b^m)$

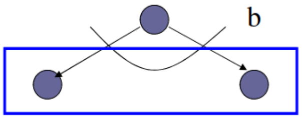
Properties of depth-first search

- **Completeness: Does it always find the solution if it exists?**
  No, if we permit infinite loops.
  Yes, if we prevent them
- **Optimality: does it find the minimum length path ?**
  No. Solution found first may not be the shortest possible.
- **Time complexity:** $O(b^m)$
  exponential in the maximum depth of the search tree m
- **Memory (space) complexity: ?**

DFS - memory complexity

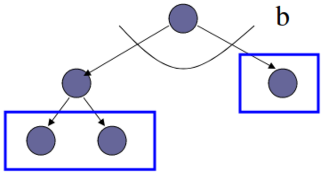
depth	number of nodes kept
0	1

DFS - memory complexity



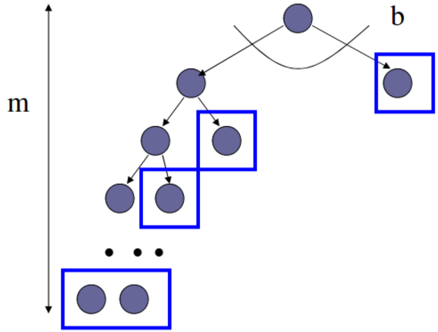
depth	number of nodes kept
0	0
1	2 = b

DFS - memory complexity



depth	number of nodes kept
0	0
1	1 = (b-1)
2	2 = b

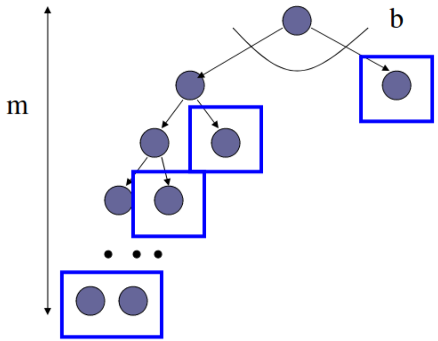
DFS - memory complexity



depth	number of nodes kept
0	0
1	1
2	1
3	1
• • •	
m	2=b

Complexity:

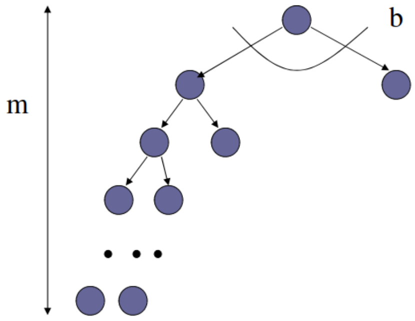
DFS - memory complexity



depth	number of nodes kept
0	0
1	1=(b-1)
2	1= (b-1)
3	1 =(b-1)
• • •	
m	2=b

Complexity: $O(bm)$

DFS - memory complexity



depth	number of nodes
0	1
1	2 = b
2	2
3	2
• • •	
m	2

Total nodes: $O(bm)$

Properties of depth-first search

- **Completeness: Does it always find the solution if it exists?**
  No, if we permit infinite loops.
  Yes, if we prevent them
- **Optimality: does it find the minimum length path ?**
  No. Solution found first may not be the shortest possible.
- **Time complexity:** $O(b^m)$
  exponential in the maximum depth of the search tree m
- **Memory (space) complexity:** $O(bm)$
  linear in the maximum depth of the search tree m

Properties of depth-first search

- **Completeness: Does it always find the solution if it exists?**
  No, if we permit infinite loops.
  Yes, if we prevent them
- **Optimality: does it find the minimum length path ?**
  No. Solution found first may not be the shortest possible.
- **Time complexity:** $O(b^m)$
  exponential in the maximum depth of the search tree m
- **Memory (space) complexity:** $O(bm)$
  the tree size we need to keep is linear in the maximum depth of the search tree m

Outline

Searching for the minimum cost path

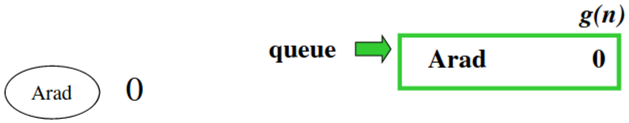
- **General minimum cost path-search problem:**
  - adds *weights* or *costs* to operators (links)
- **Search strategy:**
  "Intelligent" expansion of the search tree should be driven by the cost of the current (partially) built path
- **Implementation:**
  - **Path cost function for node** n: $g(n)$
    - Length of the path represented by the search tree branch starting at the root of the tree (initial state) to n
  - **Search strategy:**
    - Expand the leaf node with the minimum $g(n)$ first
    - Can be implemented by the priority queue

Searching for the minimum cost path

- The basic algorithm for finding the minimum cost path:
  - **Dijkstra's shortest path**
- In AI, the strategy goes under the name:
  - **Uniform cost search**
- **Note:** When operator costs are all equal to 1 the uniform cost search is equivalent to the breadth first search BFS

Uniform cost search

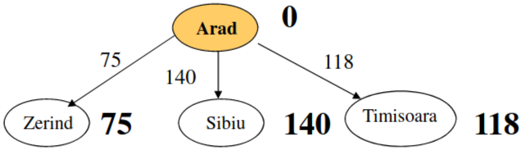
- Expand the node with the minimum path cost first
- **Implementation**: a priority queue



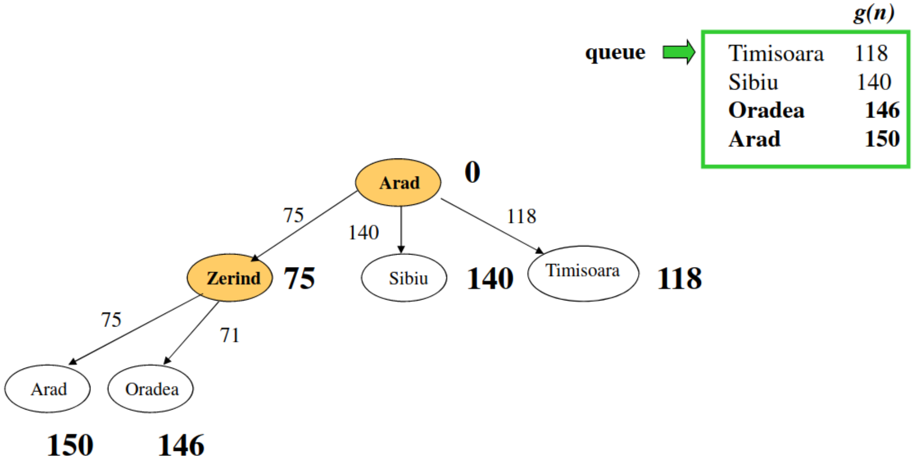
Uniform cost search

queue

	$g(n)$
Zerind	75
Timisoara	118
Sibiu	140



Uniform cost search

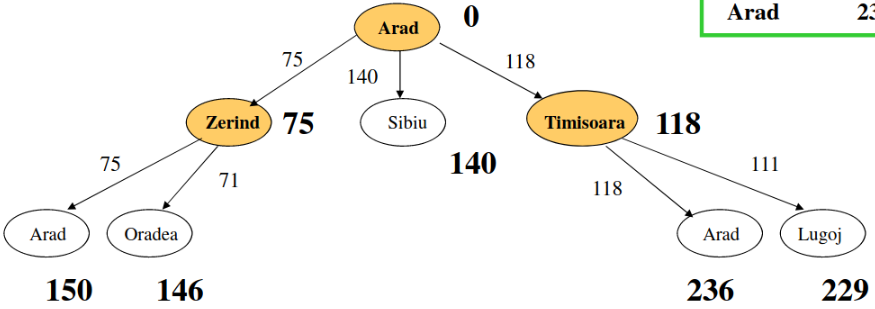


	$g(n)$
Timisoara	118
Sibiu	140
Oradea	**146**
Arad	**150**

Uniform cost search

queue

	g(n)
Sibiu	140
Oradea	146
Arad	150
Lugoj	**129**
Arad	**236**



Outline

Uninformed Search Strategies (continue)
- Searching for the minimum cost path

Informed (Heuristic) Search Strategies
- Greedy search
- A* search

Practice

Evaluation-function driven search

- A search strategy can be defined in terms of a node evaluation function
  - Similarly to the path cost for the uniform cost search
- Evaluation function
  - Denoted $f(n)$
  - Defines the desirability of a node to be expanded next
- Evaluation-function driven search:
  - expand the node (state) with the best evaluation-function value
- Implementation:
  - priority queue with nodes in the decreasing order of their evaluation function value

Uniform cost search

- Uniform cost search (Dijkstra's shortest path):
  - A special case of the evaluation-function driven search
  $f(n) = g(n)$
- **Path cost function** $g(n)$
  - path cost from the initial state to n
- **Uniform-cost search:**
  - Can handle general minimum cost path-search problem:
  - weights or costs associated with operators (links).
- **Note**: Uniform cost search relies on the problem definition only
  - It is an uninformed search method

Additional information to guide the search

- **Uninformed search methods**
  - Use only the information from the problem definition; and
  - Past explorations, e.g. cost of the path generated so far
- **Informed search methods**
  - Incorporate additional measure of a potential of a specific state to reach the goal
  - A potential of a state (node) to reach a goal is measured by a **heuristic function** $h(n)$

heuristic function is denoted $h(n)$

Best-first search

Best-first search = evaluation-function driven search

- Typically incorporates a **heuristic function**, $h(n)$, into the evaluation function $f(n)$ to guide the search.

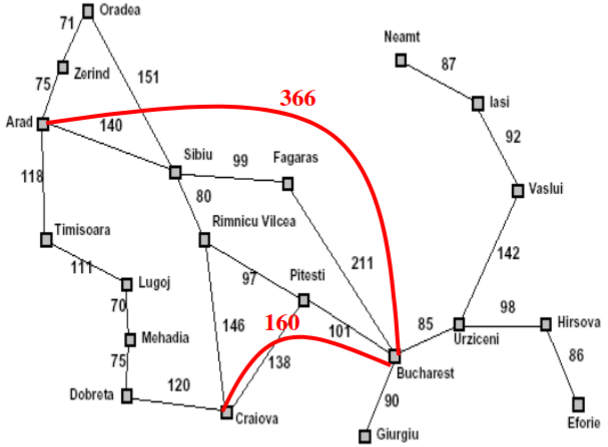
Heuristic function $h(n)$:

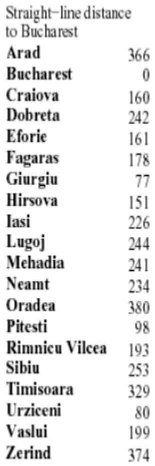
- Measures a potential of a state (node) to reach a goal
- Typically expressed in terms of some distance to a goal estimate

Example of a heuristic function:

- Assume a shortest path problem with city distances on connections
- Straight-line distances between cities give additional information we can use to guide the search

Example: traveler problem with straight-line distance information





Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Straight-line distances give an estimate of the cost of the path between the two cities

Best-first search

Best-first search = evaluation-function driven search

- Typically incorporates a **heuristic function**, $h(n)$, into the evaluation function $f(n)$ to guide the search.
- **heuristic function**: measures a potential of a state (node) to reach a goal

Special cases (differ in the design of evaluation function):

- Greedy search: $f(n) = h(n)$
- A* algorithm: $f(n) = g(n) + h(n)$

Outline

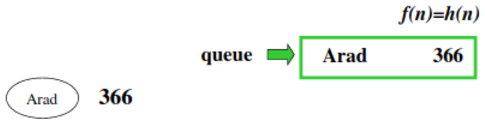
Greedy search method

- Evaluation function is equal to the heuristic function

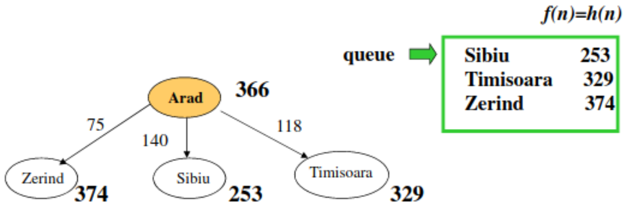
$$f(n) = h(n)$$

- **Idea**: the node that seems to be the closest to the goal is expanded first

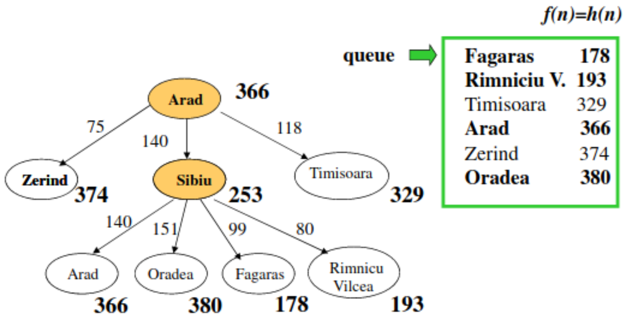
Greedy search method: demo



Greedy search method: demo

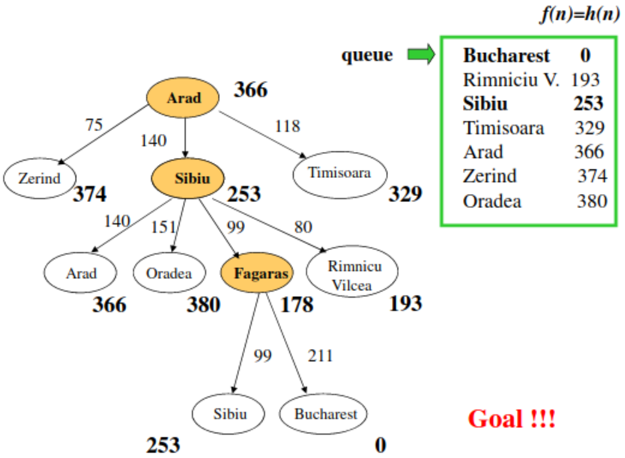


Greedy search method: demo



Fagaras	**178**
Rimniciu V.	**193**
Timisoara	329
Arad	**366**
Zerind	374
Oradea	**380**

Greedy search method: demo



Bucharest	**0**
Rimniciu V.	193
Sibiu	**253**
Timisoara	329
Arad	366
Zerind	374
Oradea	380

Goal !!!

Outline

A* search

- The problem with the **greedy search** is that it can keep expanding paths that are already very expensive.
- The problem with the **uniform-cost search** is that it uses only past exploration information (path cost), no additional information is utilized

A* search

A* Idea: to provide the algorithm a **"brain"** that can produce **estimation/approximation**

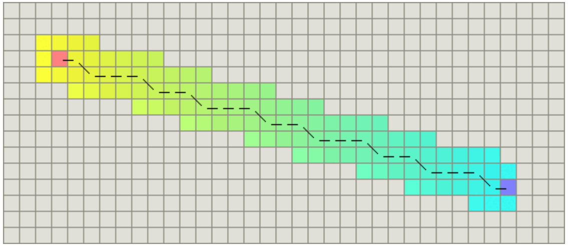
- **A* search**
  $f(n) = g(n) + h(n)$
  $g(n)$ - is the cost of the path from the start node to n
  $h(n)$ - is a heuristic function that estimates the cost of the cheapest path from n to the goal
  $f(n)$ - estimated cost of the best path that continues from n

Approximation functions/heuristics

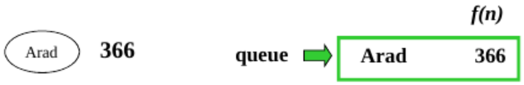
Several typically three approximation heuristics $h(n)$:

- Manhattan Distance
- Euclidean Distance
- Diagonal Distance: If your map allows diagonal movement you need a different heuristic.

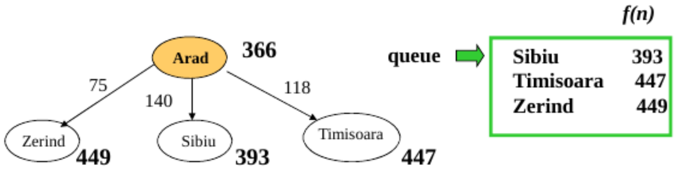
```
function heuristic(node) =
dx = abs(node.x - goal.x)
dy = abs(node.y - goal.y)
return D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)
```



A* search: demo

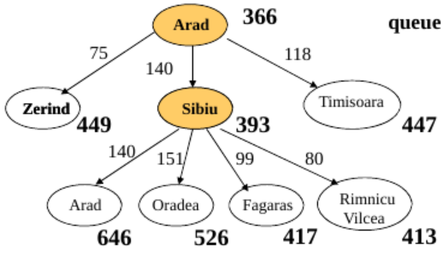


A* search: demo



Sibiu	393
Timisoara	447
Zerind	449

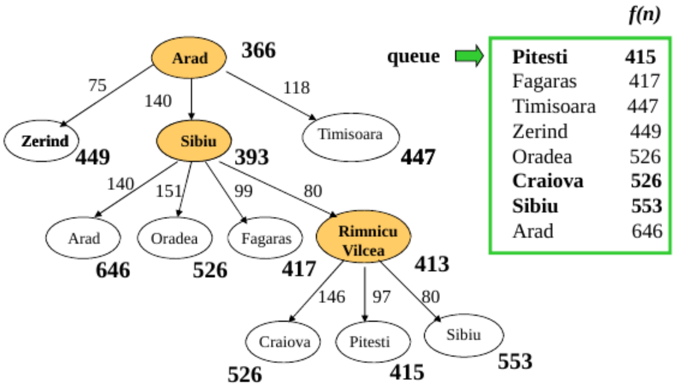
A* search: demo



queue

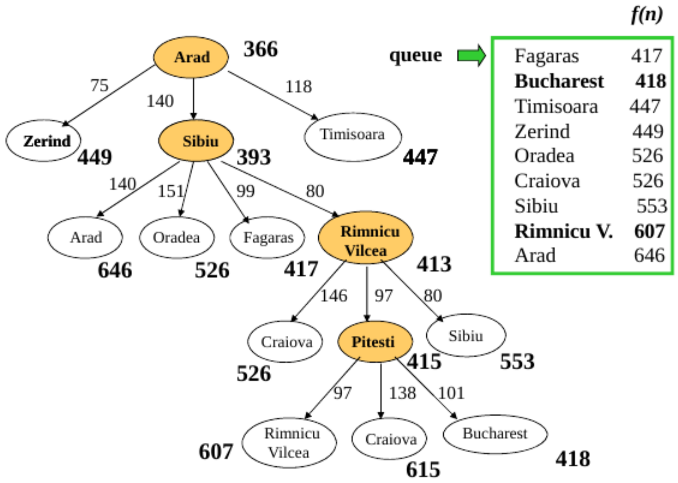
	f(n)
Rimnicu V.	**413**
Fagaras	**417**
Timisoara	447
Zerind	449
Oradea	**526**
Arad	**646**

A* search: demo



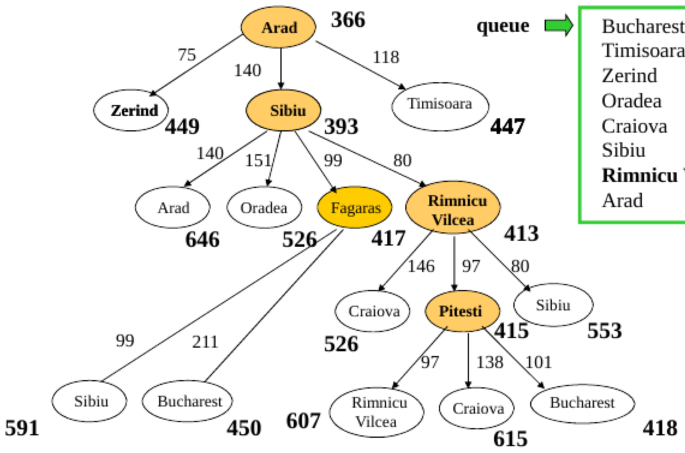
queue	f(n)
Pitesti	**415**
Fagaras	417
Timisoara	447
Zerind	449
Oradea	526
Craiova	**526**
Sibiu	**553**
Arad	646

A* search: demo



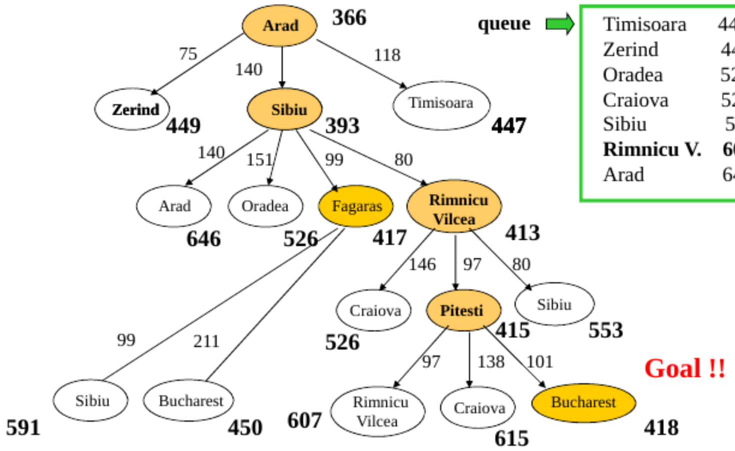
	f(n)
Fagaras	417
Bucharest	**418**
Timisoara	447
Zerind	449
Oradea	526
Craiova	526
Sibiu	553
Rimnicu V.	**607**
Arad	646

A* search: demo



	f(n)
Bucharest	418
Timisoara	447
Zerind	449
Oradea	526
Craiova	526
Sibiu	553
Rimnicu V.	**607**
Arad	646

A* search: demo

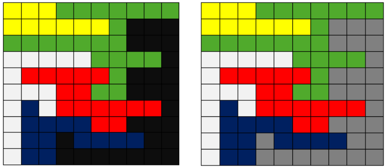


	f(n)
Timisoara	447
Zerind	449
Oradea	526
Craiova	526
Sibiu	553
Rimnicu V.	**607**
Arad	646

Outline

Exercise 1: Flood fill

Consider the following matrix to the left - if the start node is (3, 9), target color is “BLACK” and replacement color is “GREY”, the floodfill algorithm looks for all nodes in the matrix that are connected to the start node by a path of the target color and changes them to the replacement color.



Question:

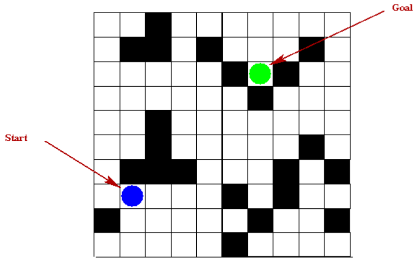
Propose the idea to perform the floodfill algorithm that takes three parameters: a start node, a target color, and a replacement color.

Exercise 2:

- How would you describe the initial "configuration"?
- Find a configuration that meets the goal (where the arm tip is on the goal). How would you technically describe this particular final configuration?
- See if you can describe a few intermediate configurations.

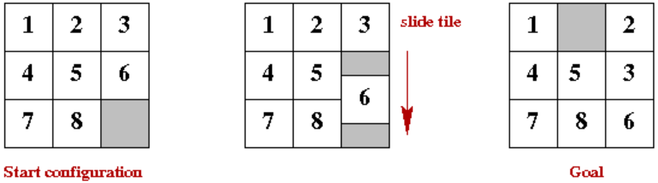
The maze problem

- The maze is an N x N grid of cells.
- Each cell is either closed (prohibited) or not.
- There is a start cell and a goal cell.
- At each step, the allowable actions are: take one step (one cell) in one of four directions North, South, East, West.
- For example, the following sequence works for the bellow instance of the problem: W, N, N, N, N, N, E, E, E, N, N, E, E, E, S, S
- Secondary goal: find an efficient (short) sequence.



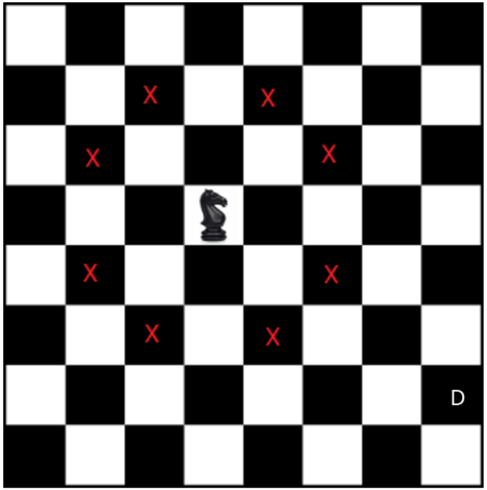
The N-puzzle problem

- The example shows an 8-puzzle.
- There are 8 tiles in 9 spaces.
- A "move" (action) consists of moving a tile into the blank spot (leaving another blank spot).
- The objective: find a sequence of actions to go from the initial configuration to the goal.
- The algorithmic problem: write an algorithm to do so.
- Secondary goal: find an efficient (short) sequence.



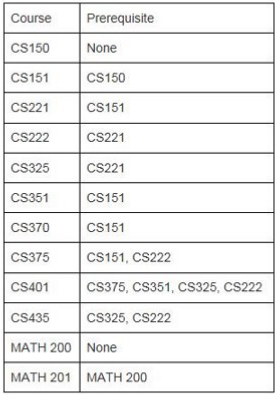
Chess Knight Problem

Given a chessboard, find the shortest distance (minimum number of steps) taken by a knight to reach a given destination from a given source.



Exercise 3:

Given a list of courses and prerequisites for a CS degree.
(a) Draw a directed acyclic graph (DAG) that represents the precedence among the courses.
(b) Give a topological sort of the graph.
(c) Find an order in which all the classes can be taken
(d) Determine the length of the longest path in the DAG. How did you find it? What does this represent?



Course	Prerequisite
CS150	None
CS151	CS150
CS221	CS151
CS222	CS221
CS325	CS221
CS351	CS151
CS370	CS151
CS375	CS151, CS222
CS401	CS375, CS351, CS325, CS222
CS435	CS325, CS222
MATH 200	None
MATH 201	MATH 200