



Usman Institute of Technology
Department of Computer Science
Course Code: CS-321
Artificial Intelligence
Spring 2023

Lab 05

Instructor: Dr. Nasir Uddin
E-mail: nuddin@uit.edu

Objective

This experiments models problems as search problems and then explores their possible solutions using different search algorithms.

Student Information

Student Name	
Student ID	
Date	

Assessment

Marks Obtained	
Remarks	
Signature	



Usman Institute of Technology
Department of Computer Science
Course Code: CS-321
Artificial Intelligence
Spring 2023

Objective

This experiments models problems as search problems and then explores their possible solutions using different search algorithms.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work using TEAMS.

1 Search

Search is looking for a sequence of actions that reaches the goal states. A search algorithm takes a problem as input and returns a solution in the form of an action sequence. Once a solution is found, the actions it recommends can be carried out. After formulating a goal and a problem to solve, a search procedure is called to solve it. The solution is then used to guide the actions, whatever the solution recommends as the next thing to do—typically, the first action of the sequence—and then removing that step from the sequence. Once the solution has been executed, the goal is reached.

1.1 Formulating Problems

There are five components which formulate a problem as a search problem.

- state
- Actions
- Transition model
- Goal test
- Path cost

form a problem. This formulation is abstract, i.e., details are hidden. Abstraction is useful since they simplify the problem by hiding many details but still covering the most important information about states and actions (retaining the state space in simple form), therefore abstraction needs to be valid. Abstraction is called valid when the abstract solution can be expanded to more detailed world. Abstraction is useful if the actions in the solution are easier than the original problem, i.e, no further planning and searching. Construction of useful and valid abstraction is challenging.

1.2 Example Problem: 8-Puzzle Problem

The 8-puzzle is often used as test problem for new search algorithms in AI. The 8-puzzle has $9!/2 = 181,440$ reachable states and is easily solved. The 15-puzzle (on a 4×4 board) has around 1.3 trillion states, and random instances can be solved optimally in a few milliseconds by the best search algorithms. The 24-puzzle (on a 5×5 board) has around 10^{25} states, and random instances take several hours to solve optimally. The 8-puzzle, an instance of which is shown in the figure, consists of a 3×3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The object is to reach a specified goal state, such as the one shown on the right of the figure. The standard formulation is as follows:



Usman Institute of Technology
Department of Computer Science
Course Code: CS-321
Artificial Intelligence
Spring 2023

- States: Description of the location of each of the eight tiles and the blank square.
- Actions & transition model: Moving the blank; left, right, up, or down.
- Goal Test: Does the state match the goal state?
- Path Cost: Each step costs 1 unit

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

1.3 Search Schemes

Searching is the universal technique of problem solving in AI.

Infrastructure for a Search Algorithm:

Search algorithms require a data structure to keep track of the search tree that is being constructed. For each node n of the tree, we have a structure that contains four components:

- $n.STATE$: the state in the state space to which the node corresponds;
- $n.PARENT$: the node in the search tree that generated this node;
- $n.ACTION$: the action that was applied to the parent to generate the node;
- $n.PATH-COST$: the cost, traditionally denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers.

Given the components for a parent node, it is easy to see how to compute the necessary components for a child node. The function `CHILD-NODE` takes a parent node and an action and returns the resulting child node:

```
1  function CHILD-NODE(problem, parent, action) returns a node
2      return a node with
3          STATE = problem.RESULT(parent.STATE, action),
4          PARENT = parent, ACTION = action,
5          PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

Next, the frontier needs to be stored in such a way that the search algorithm can easily choose the next node to expand `QUEUE` according to its preferred strategy. The appropriate data structure for this is a queue. The operations on a queue are as follows:

- `EMPTY?` (queue) returns true only if there are no more elements in the queue.
- `POP` (queue) removes the first element of the queue and returns it.
- `INSERT` (element, queue) inserts an element and returns the resulting queue.



Usman Institute of Technology
Department of Computer Science
Course Code: CS-321
Artificial Intelligence
Spring 2023

1.4 Types of Search Algorithm

There are some single-player games such as tile games, Sudoku, crossword, etc. The search algorithms help you to search for a particular position in such games. There are two kinds of AI search techniques:

- Uninformed search
- Informed search

1.5 Uninformed Search

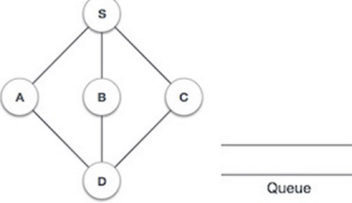
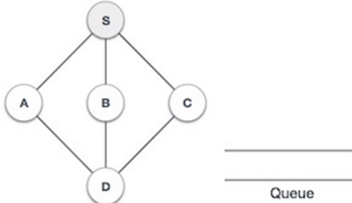
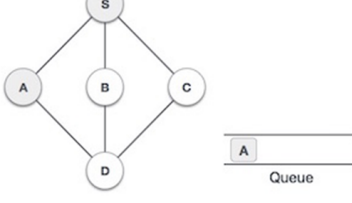
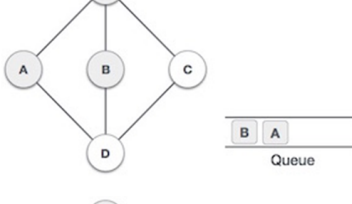
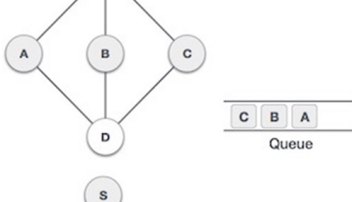
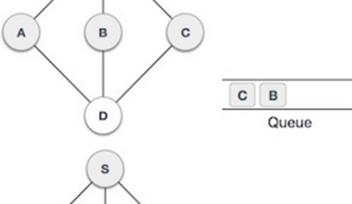
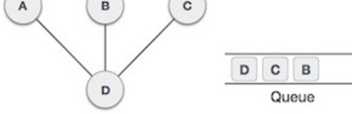
Sometimes we may not get much relevant information to solve a problem. Suppose we lost our car key and we are not able to recall where we left, we have to search for the key with some information such as in which places we used to place it. It may be our pant pocket or may be the table drawer. If it is not there then we have to search the whole house to get it. The best solution would be to search in the places from the table to the wardrobe. Here we need to search blindly with fewer clues. This type of search is called uninformed search or blind search. There are two popular AI search techniques in this category:

- Breadth first search
- Depth first search

1.6 Breadth-First Search

It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

Usman Institute of Technology
 Department of Computer Science
 Course Code: CS-321
 Artificial Intelligence
 Spring 2023

Step	Traversal	Description
1		Initialize the queue.
2		We start from visiting S (starting node), and mark it as visited.
3		We then see an unvisited adjacent node from S. In this example, we have three nodes but alphabetically we choose A, mark it as visited and enqueue it.
4		Next, the unvisited adjacent node from S is B. We mark it as visited and enqueue it.
5		Next, the unvisited adjacent node from S is C. We mark it as visited and enqueue it.
6		Now, S is left with no unvisited adjacent nodes. So, we dequeue and find A.
7		From A we have D as unvisited adjacent node. We mark it as visited and enqueue it.



Usman Institute of Technology
Department of Computer Science
Course Code: CS-321
Artificial Intelligence
Spring 2023

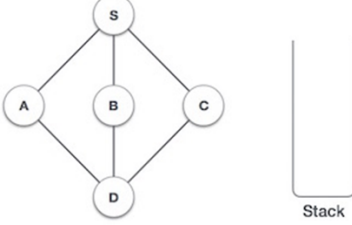
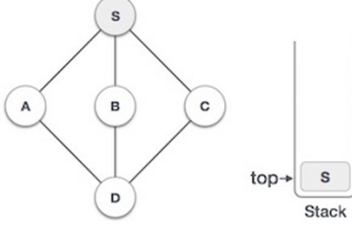
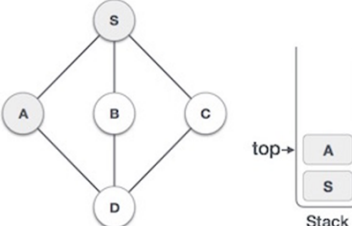
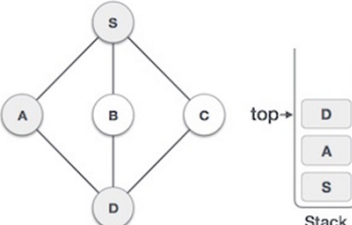
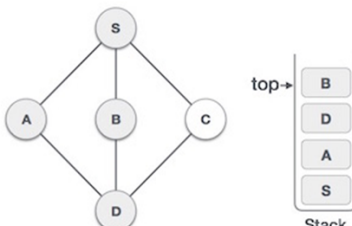
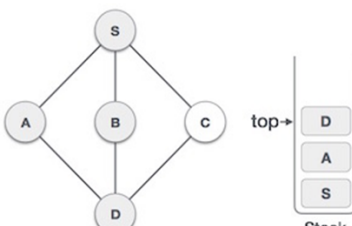
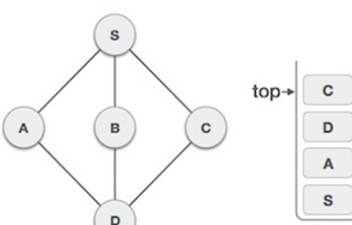
Pseudocode

```
1 BFS (G, s) #Where G is the graph and s is the source node
2   let Q be queue.
3   Q.enqueue(s) #Inserting s in queue until its neighbour vertices are
   marked.
4
5   mark s as visited.
6   while (Q is not empty)
7     # Removing that vertex from queue whose neighbour will be visited now
8     y=Q.dequeue()
9
10    #processing all the neighbours of v
11    for all neighbours w of v in the Graph G
12      if w is not visited
13        Q.enqueue(w) #Stores w in Q to further visit its neighbours
14        mark w as visited.
```

1.7 Depth-First Search

It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order.

Usman Institute of Technology
 Department of Computer Science
 Course Code: CS-321
 Artificial Intelligence
 Spring 2023

Step	Traversal	Description
1		Initialize the stack.
2		Mark S as visited and put it onto the stack. Explore any unvisited adjacent node from S. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.
3		Mark A as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both S and D are adjacent to A but we are concerned for unvisited nodes only.
4		Visit D and mark it as visited and put onto the stack. Here, we have B and C nodes, which are adjacent to D and both are unvisited. However, we shall again choose in an alphabetical order.
5		We choose B, mark it as visited and put onto the stack. Here B does not have any unvisited adjacent node. So, we pop B from the stack.
6		We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find D to be on the top of the stack.
7		Only unvisited adjacent node is from D is C now. So we visit C, mark it as visited and put it onto the stack.



Usman Institute of Technology
Department of Computer Science
Course Code: CS-321
Artificial Intelligence
Spring 2023

```
1 DFS-iterative(G, s): # Where G is graph and s is the source vertex
2   let S be stack
3   S.push(s) #Inserting s on stack
4   mark s as visited.
5   while (S is not empty):
6     #pop a vertex from stack to visit next
7     y=S.top()
8     S.pop()
9     #Push all the neighbours of v into the stack that are not visited
10    for all neighbours w of v in Graph G:
11      if w is not visited:
12        S.push(w)
13        mark w as visited
14
15 DFS-recursive(G, s):
16   mark s as visited
17   for all neighbours w of s in Graph G:
18     if w is not visited:
19       DFS-recursive(G, w)
```

1.8 Informed Search

A search strategy which searches the most promising branches of the state-space first can:

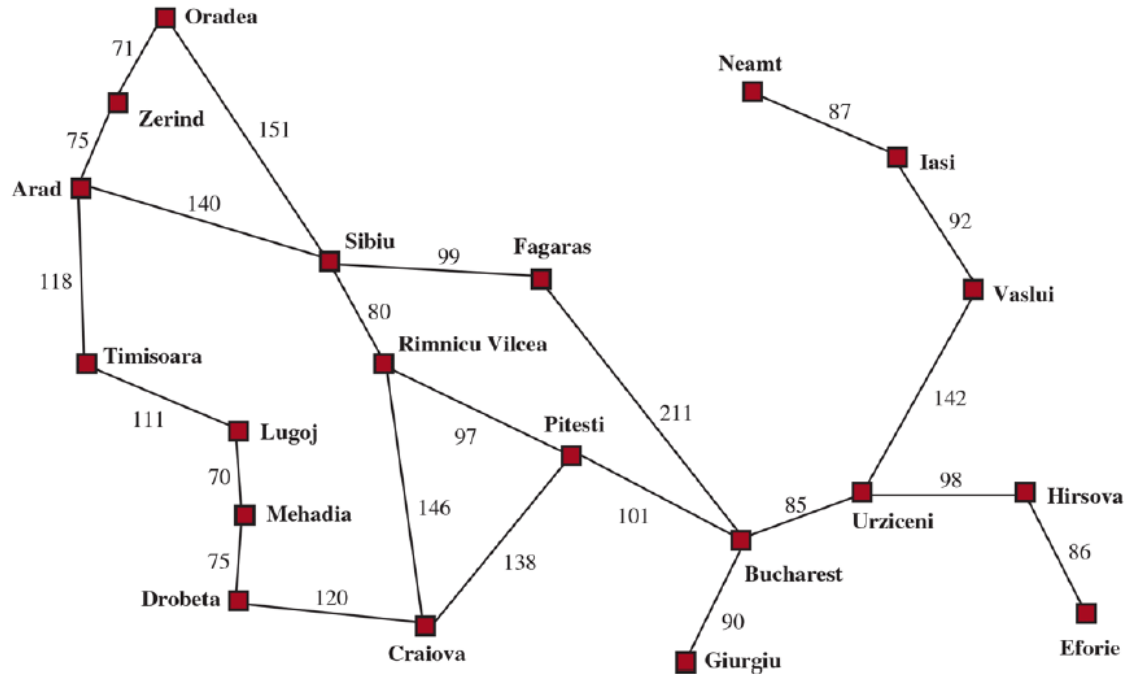
- find a solution more quickly,
- find solutions even when there is limited time available,
- often find a better solution, since more profitable parts of the state-space can be examined, while ignoring the unprofitable parts.

A search strategy which is better than another at identifying the most promising branches of a search-space is said to be more informed.

1.9 Greedy Best-First Search

Greedy (Best-First) search is similar in spirit to Depth-First Search. It keeps exploring until it has to back up due to a dead end. Greedy search is not complete and not optimal, but is often fast and efficient, depending on the heuristic function h find a solution more quickly.

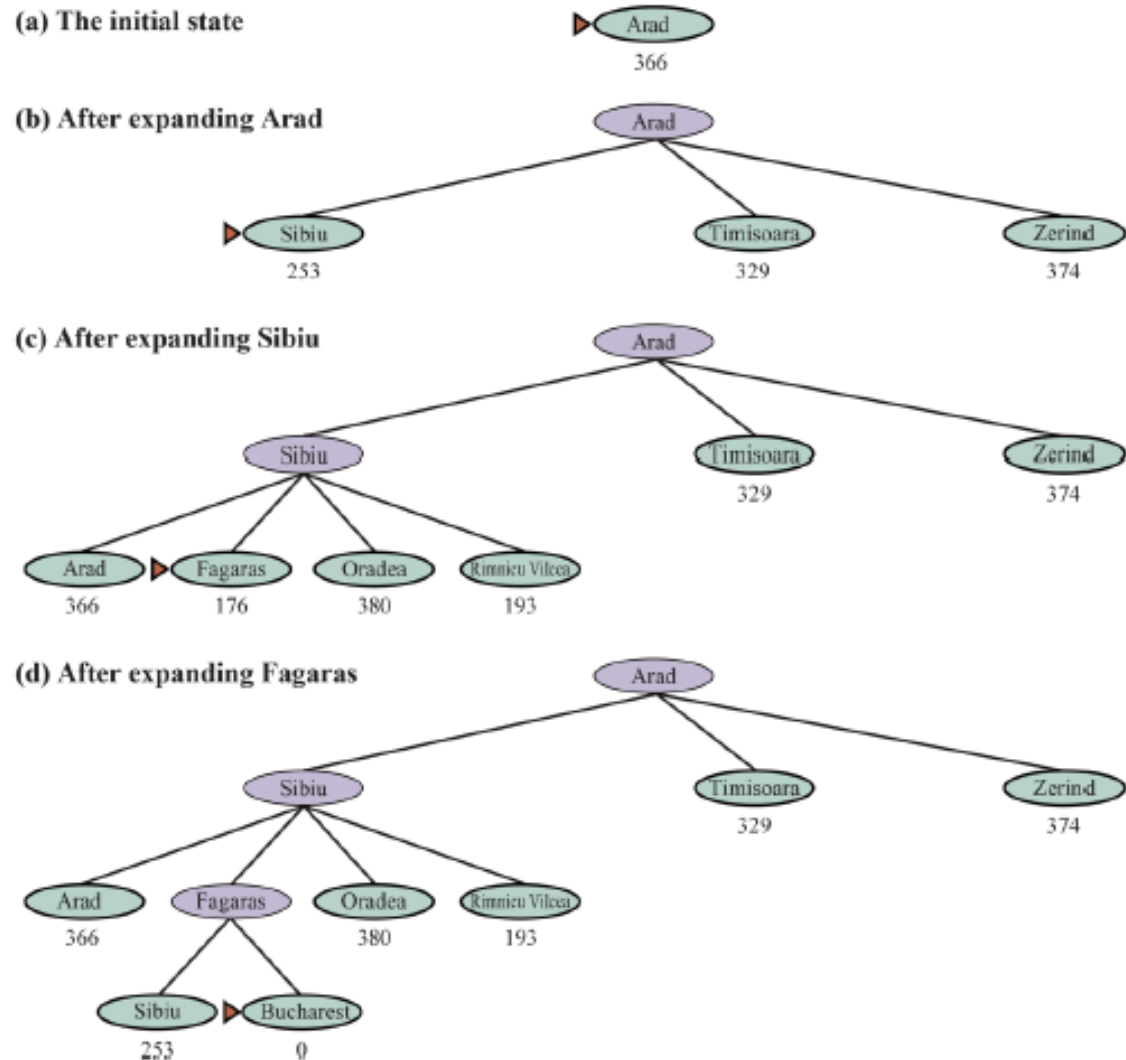
Usman Institute of Technology
 Department of Computer Science
 Course Code: CS-321
 Artificial Intelligence
 Spring 2023



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Values of h_{SLD} —straight-line distances to Bucharest.

Usman Institute of Technology
 Department of Computer Science
 Course Code: CS-321
 Artificial Intelligence
 Spring 2023

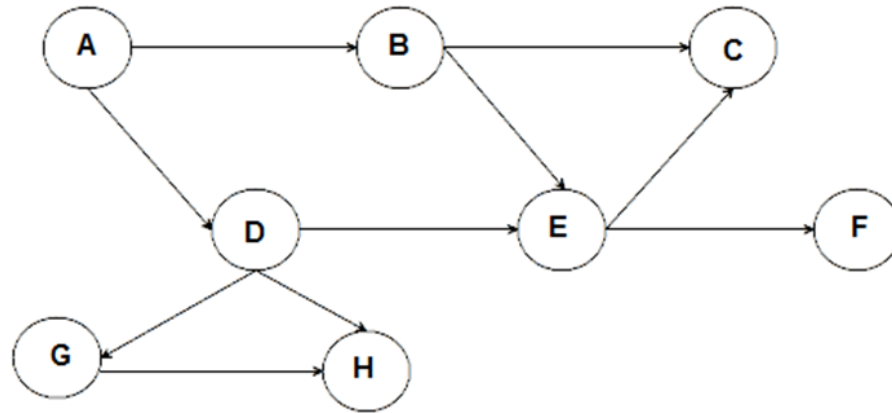


Stages in a greedy best-first tree-like search for Bucharest with the straight-line distance heuristic h_{SLD} . Nodes are labeled with their h -values.

Exercise 1

Solve depth first search and breadth first search of following graph starting from node A and reaching goal node G:

Usman Institute of Technology
 Department of Computer Science
 Course Code: CS-321
 Artificial Intelligence
 Spring 2023



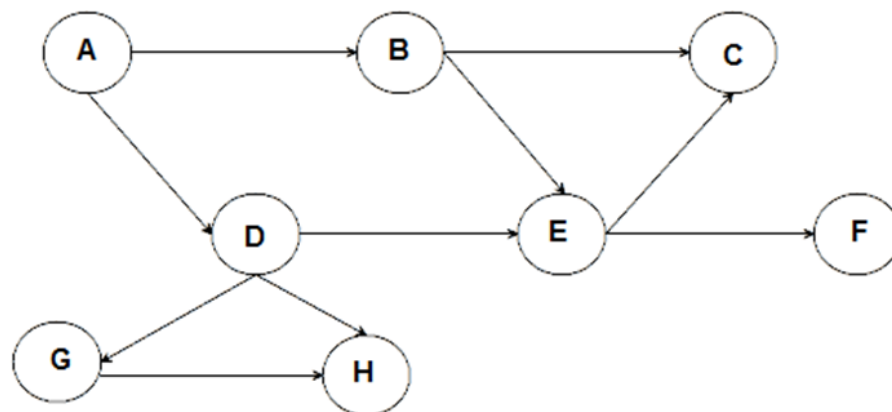
Exercise 2

Solve 8- Puzzle problem:

1. Consist of 3×3 board with eight numbered tiles and a blank space.
2. A tile adjacent to the blank space can slide into the space.
3. The objective is to reach a specified goal state, such as the one shown in the figure.
 Your game always starts with any four boxes (of your choice) already filled in any specific order, again of your choice.

Exercise 3

Solve Greedy Best-First Search of the following graph:



Note:

Don't use packages such as `bigtree` library to create and print tree. Nevertheless, you should learn how to use these packages so that you are well prepared for future tasks. You can read more about it from the following website.

<https://towardsdatascience.com/python-tree-implementation-with-bigtree-13cdabd77adc#3e14>