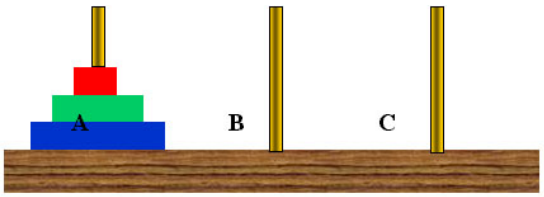
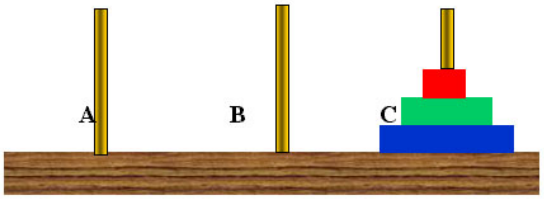
AI-Assignment#2 (Problem Solving)
SOLUTION





0) <u>EXAMPLE</u> Search Space Formulation: The Towers of Hanoi is the following problem:-

- There are three pegs, labeled A, B, and C.
- There are 3 disks on peg A. The top disk has a diameter of 1, the middle disk has a diameter of 2, and the bottom disk has a diameter of 3.
- There are no disks on peg B and C.
- You must move one disk at time and you can not place a larger disk on the top of a smaller disk.
- The problem is to get all of the disks on peg C.

(a) Design a representation for the states of this problem.

Ans: One possible answer is … Let say we have three pegs named A, B and C and three disks named 1, 2 and 3 with respective to its diameters. Thus, we can represent a state as a set of disks in legal ordering on all three pegs. So, a state is a tuple ***((A, [1,2,3]), (B, []), (C, []))***

(b) Give the initial state, goal test, successor function, and cost function from the designed representation.

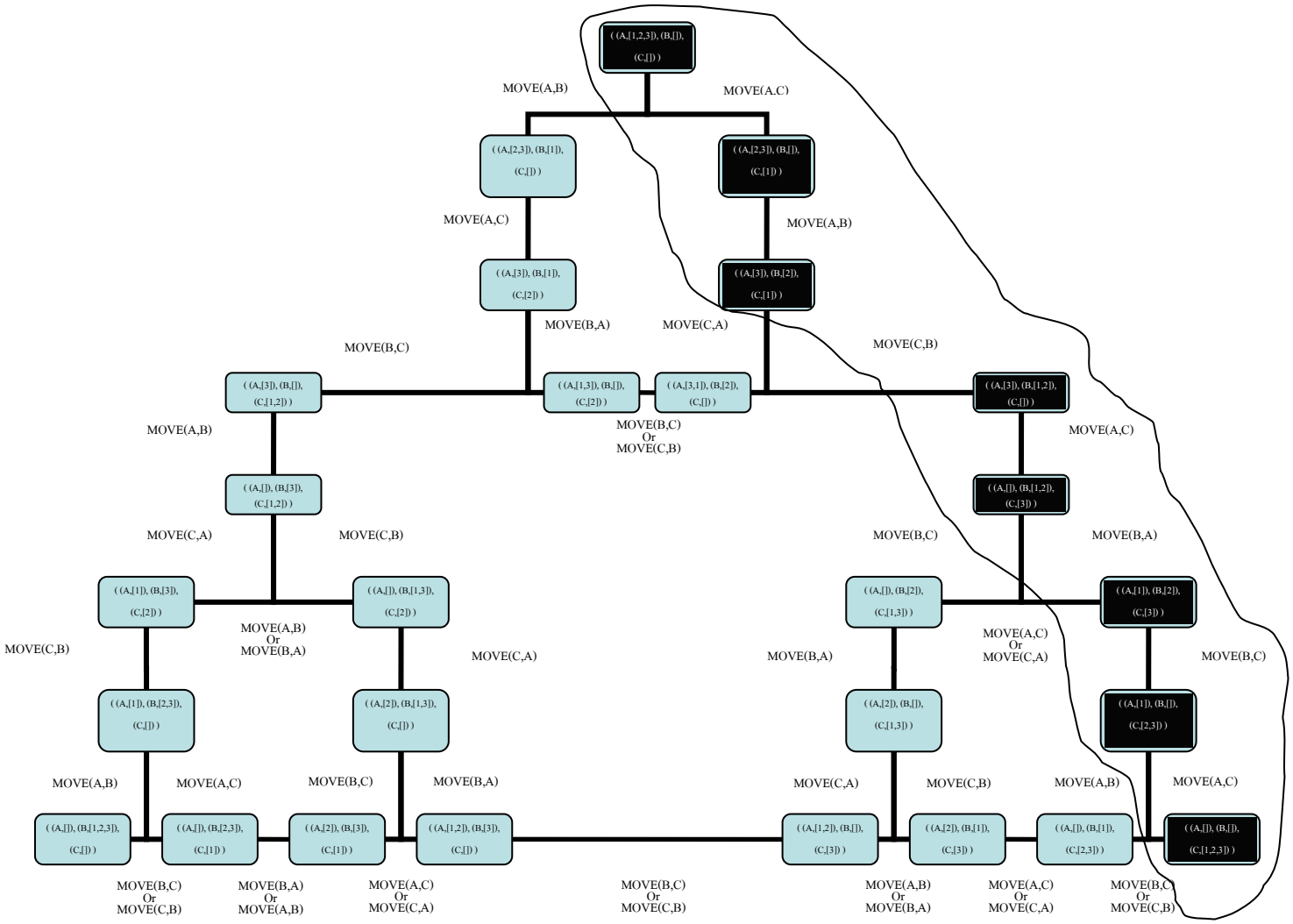
Ans: *Initial state: ((A, [1,2,3]), (B, []), (C, []))*
Goal state: ((A, []), (B, []), (C, [1,2,3]))
Successor function:
Idea: For each peg, move the top disk to either of the other two pegs, as long as the disks are not smaller. Then, an operator is MOVE(from_peg, to_peg) and we also need to check legality by

```
If (disk_on_top(from_peg) < disk_on_top(to_peg)) then
        MOVE (from_peg, to_peg).
```

Cost function: number of moves

(c) Find any solution to the problem. A solution is a sequence of operator applications you have designed.

Ans: MOVE(A, C), MOVE(A, B), MOVE(C, B), MOVE(A, C), MOVE(B, A), MOVE(B, C), MOVE(A, C)



1) The **"cats and dogs Problem"** is defined as follows. Two cats and two dogs are standing with you on the left bank of a river. There is a small boat that you can ferry across with enough room for you and only one or two animals (cats or dogs). Exactly one or two animals must be with you in the boat each time it crosses the river in either direction. An indivisible action corresponds to one or two animals getting into the boat with you, moving across the river, and then all passengers disembarking the boat. The goal is to get all 4 animals and you across the river. If ever there are more dogs than cats on either side of the river, the dogs will kill the cats. Find a sequence of crossings to transport safely all the cats and dogs across the river without exposing any of the cats to be killed.

(a) Formulate this problem as a state-space search problem by giving a precise **definition of a state**, **the start state**, **the goal state** or **goal condition**, and **the operators**. Operators should be specified as "schemas" that indicate for a general state, when the operator is legal (i.e., a set of constraints on when the operator can be applied to an arbitrary state) and the description of the successor state after the operator is applied. In other words, each operator should be specified in a way that it would easily implemented in a program to solve this problem.

The possible representations can be any form. This is one of them. ☺

States representation:

We would like to represent number of cats, dogs, and boat on both sides. Thus, we will use the form of ***[L: c_l, d_l, b_l]-[R: c_r, d_r, b_r]*** *to represent states, in which L and R indicates the side of the river, c_l and c_r indicates a number of cats (0…2) on both sides, d_l and d_r indicates a number of dogs (0…2) on both sides, and b_l and b_r indicates a number of boat (0…1) on both sides.*

Initial state: *[L: 2, 2, 1] – [R: 0, 0, 0]*

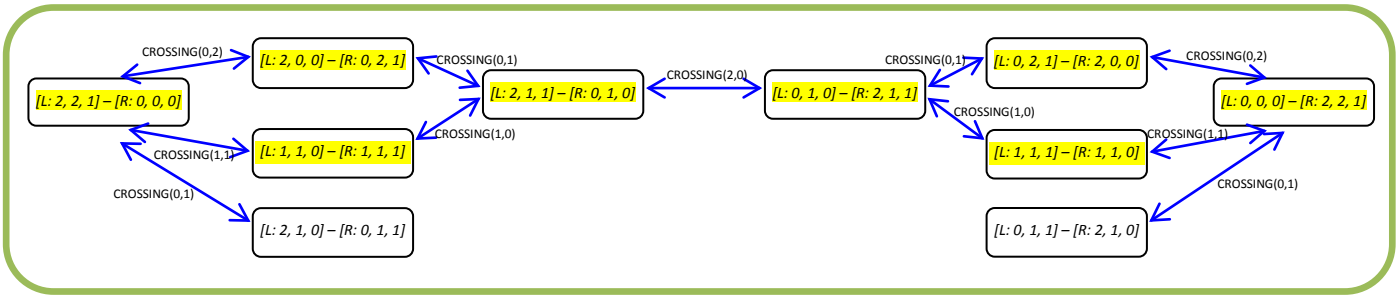
Goal state: *[L: 0, 0, 0] – [R: 2, 2, 1]*

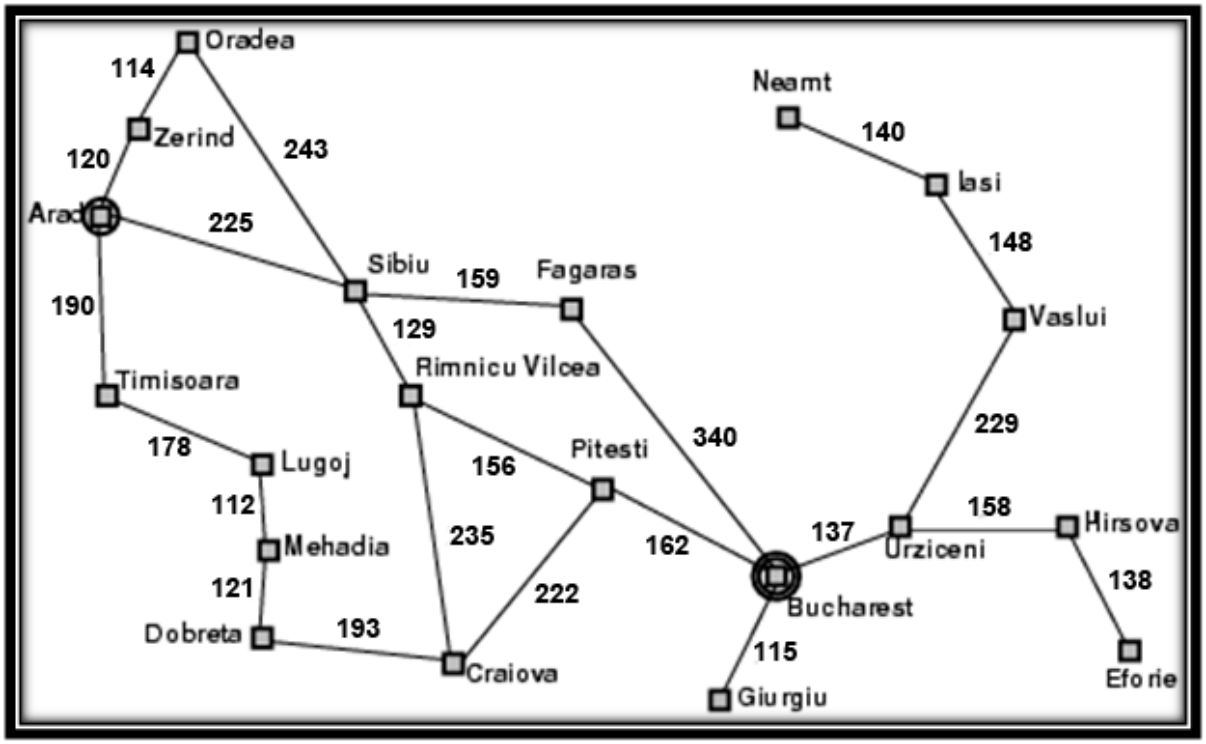
Successor function:

The successor states will be the result of performing a sequence of actions. The possible actions can be in the form of ***CROSSING(C, D)****, in which C indicates the number of cats on the boat, and D indicates the number of dogs on the boat. The restriction for the action is* ***1<= (C+D) <=2****. In addition, the restriction for the result is* ***number_of(C, L) >= number_of(D, L) when number_of(C, L) > 0,*** *and* ***number_of(C, R) >= number_of(D, R) when number_of(C, R) > 0****, in which function number_of(AnimalType, RiverSide) means number of animal who are cats or dogs in left or right side of the river.*

Cost function: *number of crossings*

(b) Show the State Space Draw the complete state-space graph that includes all nodes (and legal directed arcs connecting these nodes) for this problem. Inside each node show the state description, and label each arc with its associated operator. Highlight a path that gives a solution to the problem.





2) **<u>Uninformed Search Strategies</u>**: Please list a sequence of nodes expanded in order when perform these following search strategies in order to search for Bucharest from Arad.

<u>If set the order by expanding from **south to north (counter-clockwise)**</u>

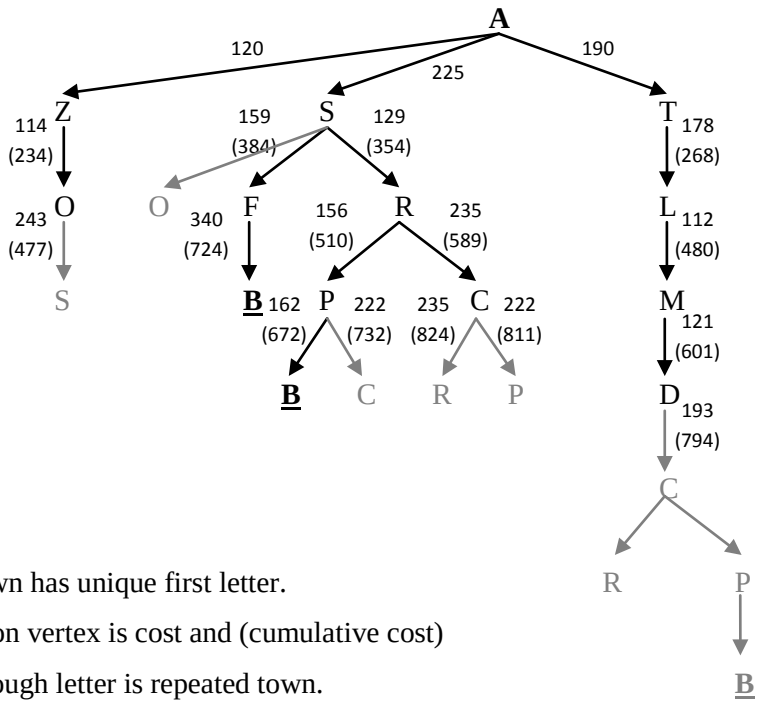
1. **<u>Breadth-first search</u>**
   Arad, Timisoara, Sibiu, Zerind, Lugoj, Rimnicu Vilcea, Fagaras, Oradea, Mehadia, Craiova, Pitesti, Bucharest
2. **<u>Uniform-cost search</u>**
   Arad(0), Zerind (120), Timisoara (190), Sibiu (225), Oradea (234), Rimnicu Vilcea (354), Lugoj (368), Fagaras (384), Mehadia (480), Pitesti (510), Craiova (589), Dorbreta (601), Bucharest(672)
3. **<u>Depth-first search</u>**
   Arad, Timisoara, Lugoj, Mehadia, Dobreta, Craiova, Pitesti, Bucharest
4. **<u>Depth-limited search</u>**
   With limit depth level = 3
   Arad(0), Timisoara (1), Lugoj (2), Mehadia (3), Sibiu (1), Rimnicu Vilcea(2), Craiova(3), Pitesti(3), Fagaras(2), Bucharest(3)
5. **<u>Iterative deepening search</u>**
   ***Level 0**: Arad(0)*
   ***Level 1**: Arad(0), Timisoara (1), Sibiu (1), Zerind(1)*
   ***Level 2**: Arad(0), Timisoara (1), Lugoj (2), Sibiu (1), Rimnicu Vilcea(2), Fagaras(2), Oradea(2), Zerind(1)*
   ***Level 3**: Arad(0), Timisoara (1), Lugoj (2), Mehadia (3), Sibiu (1), Rimnicu Vilcea(2), Craiova(3), Pitesti(3), Fagaras(2), Bucharest(3)*

<u>If set the order by expanding from **North to South (clockwise)**</u>

1. **<u>Breadth-first search</u>**
   Arad, Zerind, Sibiu, Timisoara, Oradea, Fagaras, Rimnicu Vilcea, Lugoj, Bucharest
2. **<u>Uniform-cost search</u>**
   Arad(0), Zerind (120), Timisoara (190), Sibiu (225), Oradea (234), Rimnicu Vilcea (354), Lugoj (368), Fagaras (384), Mehadia (480), Pitesti (510), Craiova (589), Dorbreta (601), Bucharest(672)
3. **<u>Depth-first search</u>**
   Arad, Zerind, Oradea, Sibiu, Fagaras, Bucharest
4. **<u>Depth-limited search</u>**
   With limit depth level = 3
   Arad(0), Zerind (1), Oradea (2), Sibiu (1), Fagaras (2), Bucharest(3)
5. **<u>Iterative deepening search</u>**
   ***Level 0**: Arad(0)*
   ***Level 1**: Arad(0), Zerind (1), Sibiu (1), Timisoara (1)*
   ***Level 2**: Arad(0), Zerind (1), Oradea(2), Sibiu (1), Fagaras(2), Rimnicu Vilcea(2), Timisoara (1) , Lugoj (2)*
   ***Level 3**: Arad(0), Zerind (1), Oradea(2), Sibiu (1), Fagaras(2), Bucharest(3)*

One of the best answers for the second question is from a student named **Kunthawat Suthambut.**

Uniformed Search Strategies



* Every town has unique first letter.

* Number on vertex is cost and (cumulative cost)

* Strikethrough letter is repeated town.

1. **Breadth-first search**:

A – T – S – Z – L – R – F – O – M – C – P – **B**	* Right to left, ignore repeated town.
A – Z – S – T – O – F – R – L – **B**	* Left to right, ignore repeated town.

2. **Uniform-cost search**:

$\mathbf{A_0} - Z_{120} - T_{190} - S_{225} - O_{234} - R_{354} - L_{368} - F_{384} - M_{480} - P_{510} - C_{589} - D_{601} - \mathbf{B_{672}}$

* Ignore repeated town with more cumulative cost

* Subscript number is cumulative cost.

3. **Depth-first search**:

$\mathbf{A_0} - T_1 - L_2 - M_3 - D_4 - C_5 - P_6 - \mathbf{B_7}$	* Right to Left, ignore repeated town.
$\mathbf{A_0} - Z_1 - O_2 - S_1 - F_2 - \mathbf{B_3}$	* Left to right, ignore repeated town.

4. **Depth-limited search**: With limit depth level = 3

$\mathbf{A_0} - T_1 - L_2 - M_3 - S_1 - R_2 - C_3 - P_3 - F_2 - \mathbf{B_3}$	* Right to Left, ignore repeated town.
$\mathbf{A_0} - Z_1 - O_2 - S_1 - F_2 - \mathbf{B_3}$	* Left to right, ignore repeated town.

5. **Iterative deepening search**:

(0) $\mathbf{A_0}$

(1) $\mathbf{A_0} - T_1 - S_1 - Z_1$

(2) $\mathbf{A_0} - T_1 - L_2 - S_1 - R_2 - F_2 - O_2 - Z_1$

(3) $\mathbf{A_0} - T_1 - L_2 - M_3 - S_1 - R_2 - C_3 - P_3 - F_2 - \mathbf{B_3}$

* Right to left, ignore repeated town.

(0) $\mathbf{A_0}$

(1) $\mathbf{A_0} - Z_1 - S_1 - T_1$

(2) $\mathbf{A_0} - Z_1 - O_2 - S_1 - F_2 - R_2 - T_1 - L_2$

(3) $\mathbf{A_0} - Z_1 - O_2 - S_1 - F_2 - \mathbf{B_3}$

* Left to right, ignore repeated town.