

Jaelle Kondohoma

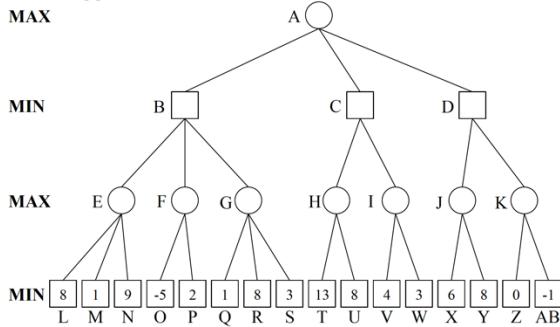
CSCE 476: Introduction to Artificial Intelligence

Homework 4

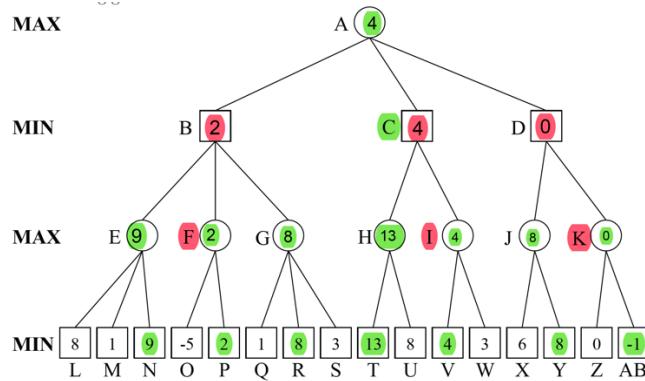
Sunday, October 3, 2021  
10:42 PM

1 Adversarial Search (Total 5 points)

Consider the following game tree:



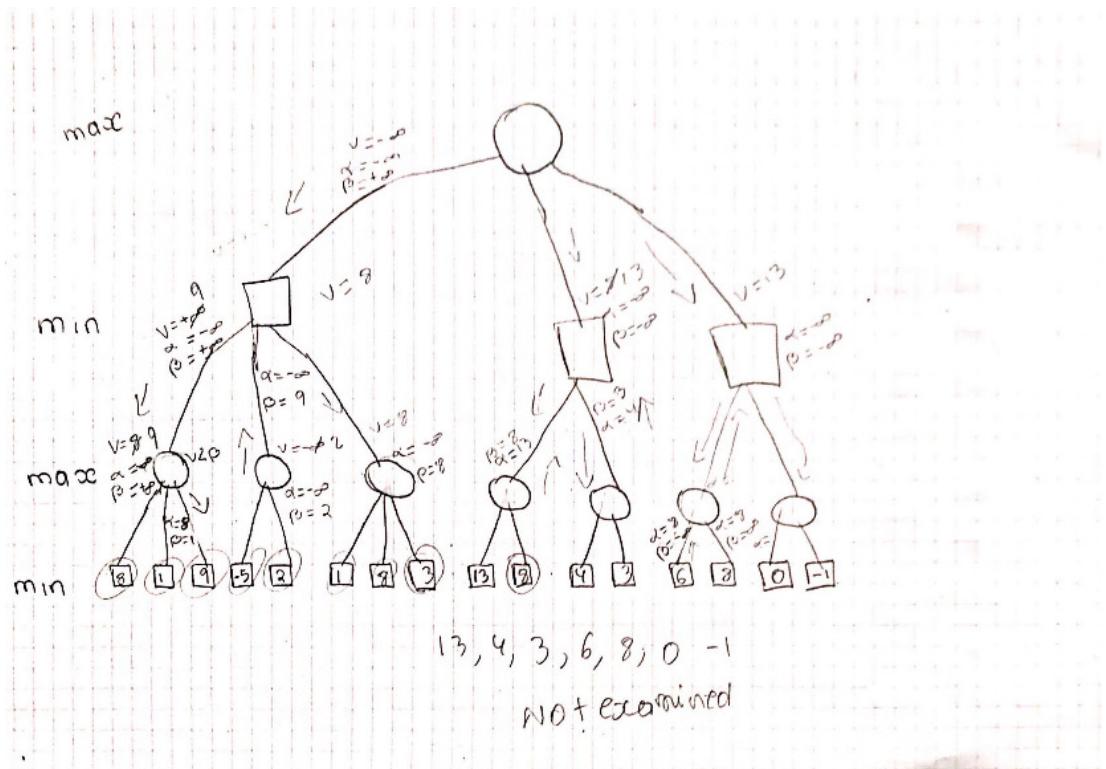
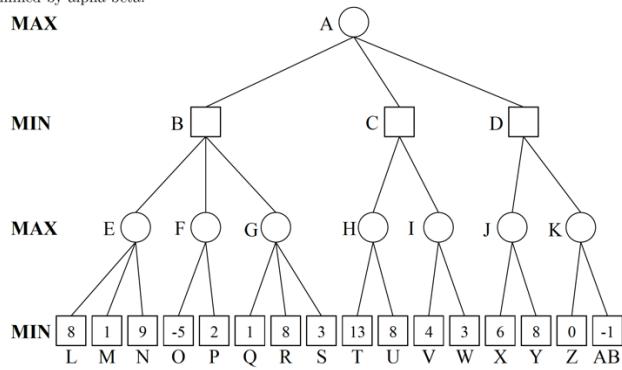
- a. Compute the minimax decision. Show your answer by writing the values at the appropriate nodes in the above tree.



- b. What move should Max choose?
- Max should choose the maximum move (the greater number)

## 2 Alpha-beta Pruning (Total 5 points)

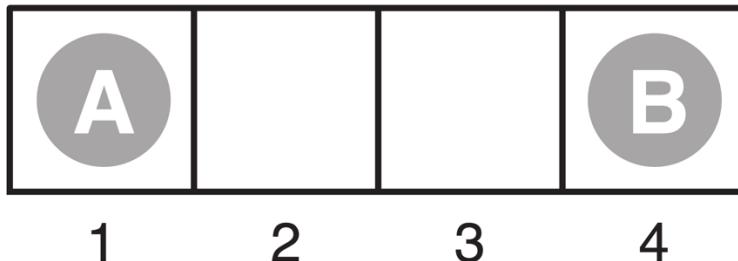
Using the *alpha-beta pruning method*, with standard left-to-right evaluation of nodes, show what nodes are *not* examined by alpha-beta.



## 3 Chapter 5, Exercise 8, Source: AIMA online site. (Total 10 points)

Consider the two-player game described in Chapter 5, Exercise 7.

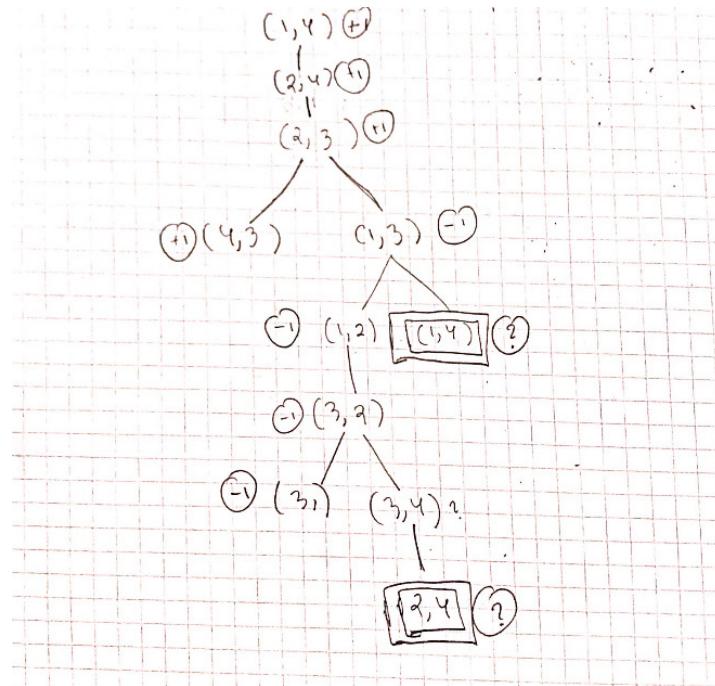
Player A moves first. The two players take turns moving, and each player must move his token to an open adjacent space in either direction. If the opponent occupies an adjacent space, then a player may jump over the opponent to the next open space if any. (For example, if A is on 3 and B is on 2, then A may move back to 1.) The game ends when one player reaches the opposite end of the board. If player A reaches space 4 first, then the value of the game to A is +1; if player B reaches space 1 first, then the value of the game to A is -1.



The starting position of a simple game.

1. Draw the complete game tree, using the following conventions:

- Write each state as  $(s_A, s_B)$ , where  $s_A$  and  $s_B$  denote the token locations.
- Put each terminal state in a square box and write its game value in a circle.
- Put *loop states* (states that already appear on the path to the root) in double square boxes. Since their value is unclear, annotate each with a "?" in a circle.

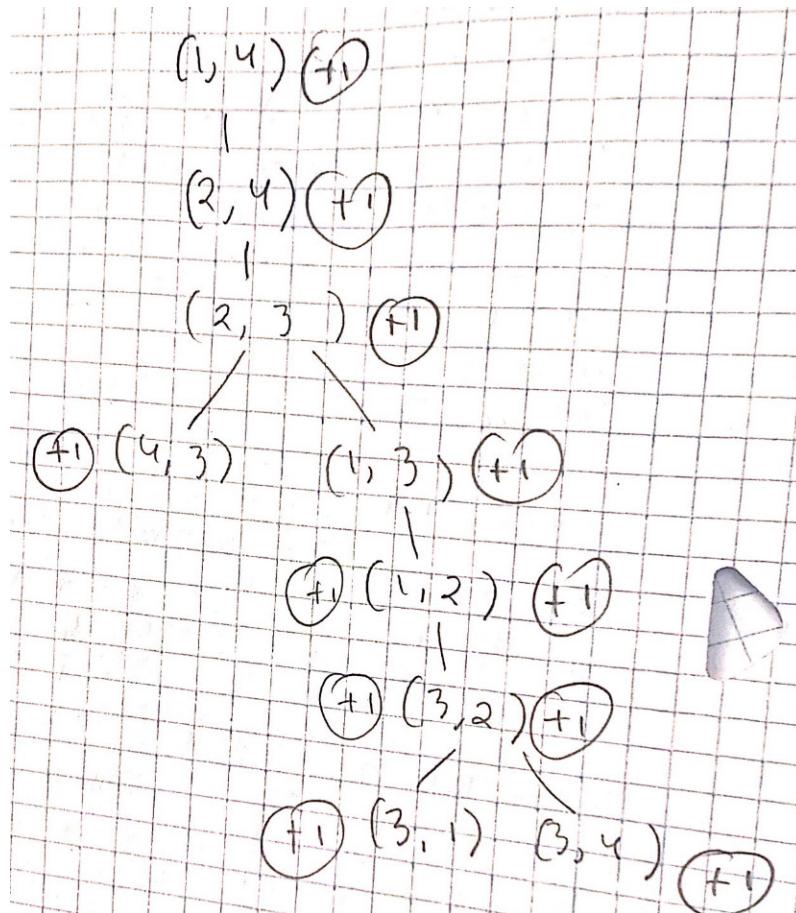


2. Now mark each node with its backed-up minimax value (also in a circle). Explain how you handled the "?" values and why.

**When we have a chance of choosing "?" values (which takes us to a node we've already visited) it will never be chosen so the backed up minimax value becomes -1**

3. Explain why the standard minimax algorithm would fail on this game tree and briefly sketch how you might fix it, drawing on your answer to part (2). Does your modified algorithm give optimal decisions for all games with loops?

Standard minimax algorithm would fail because it might get stuck in an infinite loop, we could add a variable that checks if a node has been visited already.



4. This 4-square game can be generalized to  $n$  squares for any  $n > 2$ . Prove that  $A$  wins if  $n$  is even and loses if  $n$  is odd.

#### 4 Romanian Holidays (Total 100 points)

This exercise will guide you, step by step, to implement the data structures representing Romania's map and the search algorithms for conducting search. It is mandatory to all students.

Note that in the following sections, we refer to variable and function names that may not be accepted by the programming language you choose (e.g., C, C++, and Java variables cannot contain hyphens). In these cases, just use names that are similar but acceptable by the language by, for example, using camel case instead of hyphens. Additionally, an association list is merely a list of key-value pairs.

It is not necessary to implement these structures *exactly*, but your implementation should be similar enough that the graders can clearly see the analogous variables and structures.

## 4.1 Data structures (50 points)

Create the data structures required to represent the map of Romania. Include the information about the distances between two cities linked by a road as well as the distance from any given city to Bucharest as indicated in Figure 1.

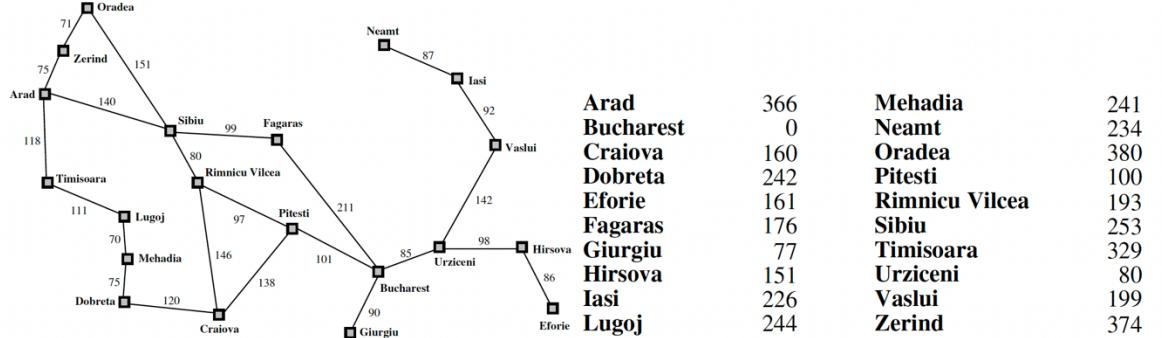


Figure 1: Map of Romania with road distances in kilometers and straight-line distances to Bucharest.

- *cities.dat*
  - Delimited file that stores information given in Figure 1.
  - Each line in the file has the following information
    - *city; neighbor<sub>1</sub>: distance; ...; neighbor<sub>i</sub>: distance; h(city)<sub>SLD</sub>*
- *City.java*
  - data structure for a city
    - name: name of city
    - neighbors: neighboring cities
    - h: value of the straight-line distance to Bucharest
    - visited: Boolean that states whether the current city has been visited or not
- Ways to access the cities
  - *ArrayList < City > allCities*
    - Stores every city object read from cities.dat
  - *HashMap < String, HashMap < String, Integer >> allCitiesHash*
    - Stores every city object read from cities.dat in key/value format where the key is the city and value is another key/value map storing neighboring city along with distance.

### 4.1.1 Tasks

#### Designing, implementing and testing the map.

- Write a function *allCitiesFromList* that takes a global variable, *allCitiesList*, and returns a list of all names of cities on the map.

```

)    /**
 * takes a global variable, allCitiesList, and returns a list of all names of
 * cities on the map.
 *
 * @param allCities
 * @return
 */
public static ArrayList<String> allCitiesFromList() {
    ArrayList<String> cities = new ArrayList<String>();
    for (City city : allCities) {
        cities.add(city.getName());
    }

    return cities;
}

```

1. Design, implement and test your map.

```

Type: class java.util.ArrayList
Size: 20

Graph Uniform Cost Search
Example: City [name=Arad, neigbours={Zerind=75, Sibiu=140, Timisoara=118}, h=366]
Path: [Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]
Cost: 418

```

2. Write a function allCitiesFromList that takes a global variable, allCitiesList, and returns a list of all names of cities on the map.

```

Oradea
Zerind
Arad
Sibiu
Timisoara
Lugoj
Mehadia
Dobreta
Craiova
Rimnicu Vilcea
Pitesti
Fagaras
Bucharest
Giurgiu
Urziceni
Hirsova
Eforie
Vaslui
Lasi
Neamt

```

- b. Write a function *allCitiesFromHtable* that takes a variable, *allCitiesHtable* and returns a list of all the structures of cities on the map.

```

/**
 * takes a variable, allCitiesHtable and returns a list of all the structures of
 * cities on the map.
 *
 * take the hash of the cities and return it back into a list
 *
 * @param allCitiesHash
 */
public static ArrayList<City> allCitiesFromHtable() {
    ArrayList<City> structures = new ArrayList<City>();

    for (Entry<String, HashMap<String, Integer>> entry : allCitiesHash.entrySet()) {
        String key = entry.getKey();
        HashMap<String, Integer> value = entry.getValue();

        for (Entry<String, Integer> cities : value.entrySet()) {
            String item = cities.getKey();
            City cite = getCityFromList(item);
            structures.add(cite);
        }
    }
    return structures;
}

```

3. Write a function `allCitiesFromHtable` that takes a variable, `allCitiesHtable` and returns a list of all the structures of cities on the map.

```

City [name=Zerind, neighbors={Oradea=71, Arad=75}, h=374]
City [name=Sibiu, neighbors={Oradea=151, Rimnicu Vilcea=80, Arad=140, Fagaras=99}, h=253]
City [name=Oradea, neighbors={Zerind=71, Sibiu=151}, h=380]
City [name=Arad, neighbors={Zerind=75, Sibiu=140, Timisoara=118}, h=366]
City [name=Urziceni, neighbors={Hirsova=98, Vaslui=142, Bucharest=85}, h=80]
City [name=Giurgiu, neighbors={Bucharest=90}, h=77]
City [name=Fagaras, neighbors={Bucharest=211, Sibiu=99}, h=176]
City [name=Pitesti, neighbors={Rimnicu Vilcea=97, Craiova=138, Bucharest=101}, h=100]
City [name=Hirsova, neighbors={Urziceni=98, Eforie=86}, h=151]
City [name=Vaslui, neighbors={Urziceni=142, Iasi=92}, h=199]
City [name=Bucharest, neighbors={Urziceni=85, Giurgiu=90, Fagaras=211, Pitesti=101}, h=0]
City [name=Zerind, neighbors={Oradea=71, Arad=75}, h=374]
City [name=Sibiu, neighbors={Oradea=151, Rimnicu Vilcea=80, Arad=140, Fagaras=99}, h=253]
City [name=Timisoara, neighbors={Arad=118, Lugoj=111}, h=329]
City [name=Dobreta, neighbors={Craiova=120, Mehadia=75}, h=242]
City [name=Lugoj, neighbors={Mehadia=70, Timisoara=111}, h=244]
City [name=Iasi, neighbors={Vaslui=92, Neamt=87}, h=226]
City [name=Vaslui, neighbors={Urziceni=142, Iasi=92}, h=199]
City [name=Neamt, neighbors={Iasi=92}, h=234]
City [name=Arad, neighbors={Zerind=75, Sibiu=140, Timisoara=118}, h=366]
City [name=Lugoj, neighbors={Mehadia=70, Timisoara=111}, h=244]
City [name=Rimnicu Vilcea, neighbors={Craiova=146, Sibiu=80, Pitesti=97}, h=193]
City [name=Craiova, neighbors={Rimnicu Vilcea=146, Dobreta=120, Pitesti=138}, h=160]
City [name=Bucharest, neighbors={Urziceni=85, Giurgiu=90, Fagaras=211, Pitesti=101}, h=0]
City [name=Hirsova, neighbors={Urziceni=98, Eforie=86}, h=151]
City [name=Rimnicu Vilcea, neighbors={Craiova=146, Sibiu=80, Pitesti=97}, h=193]
City [name=Dobreta, neighbors={Craiova=120, Mehadia=75}, h=242]
City [name=Pitesti, neighbors={Rimnicu Vilcea=97, Craiova=138, Bucharest=101}, h=100]
City [name=Craiova, neighbors={Rimnicu Vilcea=146, Dobreta=120, Pitesti=138}, h=160]
City [name=Sibiu, neighbors={Oradea=151, Rimnicu Vilcea=80, Arad=140, Fagaras=99}, h=253]
City [name=Pitesti, neighbors={Rimnicu Vilcea=97, Craiova=138, Bucharest=101}, h=100]
City [name=Urziceni, neighbors={Hirsova=98, Vaslui=142, Bucharest=85}, h=80]
City [name=Eforie, neighbors={Hirsova=86}, h=161]
City [name=Urziceni, neighbors={Hirsova=98, Vaslui=142, Bucharest=85}, h=80]
City [name=Iasi, neighbors={Vaslui=92, Neamt=87}, h=226]
City [name=Bucharest, neighbors={Urziceni=85, Giurgiu=90, Fagaras=211, Pitesti=101}, h=0]
City [name=Oradea, neighbors={Zerind=71, Sibiu=151}, h=380]
City [name=Rimnicu Vilcea, neighbors={Craiova=146, Sibiu=80, Pitesti=97}, h=193]
City [name=Arad, neighbors={Zerind=75, Sibiu=140, Timisoara=118}, h=366]
City [name=Fagaras, neighbors={Bucharest=211, Sibiu=99}, h=176]
City [name=Craiova, neighbors={Rimnicu Vilcea=146, Dobreta=120, Pitesti=138}, h=160]
City [name=Mehadia, neighbors={Dobreta=75, Lugoj=70}, h=241]
City [name=Timisoara, neighbors={Arad=118, Lugoj=111}, h=329]
City [name=Bucharest, neighbors={Urziceni=85, Giurgiu=90, Fagaras=211, Pitesti=101}, h=0]
City [name=Sibiu, neighbors={Oradea=151, Rimnicu Vilcea=80, Arad=140, Fagaras=99}, h=253]
```

- c. Write two functions `getCityFromList` and `getCityFromHtable` that take the name of a city as input and return the corresponding structure (by accessing a variable, `allCitiesList` and `allCitiesHtable`, respectively).

```

/**
 * take the name of a city as input and return the corresponding structure from
 * allCities hash table
 *
 * @param city
 * @return
 */
public static Entry<String, HashMap<String, Integer>> getCityFromHtable(String city) {
    Entry<String, HashMap<String, Integer>> structure = null;
    for (Entry<String, HashMap<String, Integer>> entry : allCitiesHash.entrySet()) {
        String key = entry.getKey();
        if (city.equals(key)) {
            structure = entry;
        }
    }
    return structure;
}
```

```


    /**
     * take the name of a city as input and return the corresponding structure from
     * allCities list
     *
     * @param city
     * @return
     */
    public static City getCityFromList(String input) {

        City structure = null;
        for (City city : allCities) {
            if (city.getName().equals(input)) {
                structure = city;
            }
        }

        return structure;
    }


```

4. Write two functions `getCityFromList` and `getCityFromHtable` that take the name of a city as input and return the corresponding structure (by accessing a variable, `allCitiesList` and `allcitiesHtable`, respectively).

```


    City: Arad
        Value in allCitiesList
        City [name=Arad, neighbors={Zerind=75, Sibiu=140, Timisoara=118}, h=366]

        Value in allCitiesHtable
        Arad={Zerind=75, Sibiu=140, Timisoara=118}


```

- d. Write two functions `neighborsUsingList` and `neighborsUsingHtable` that take the name of a city as input and return the list of structures of its direct neighbors. `neighborsUsingList` and `neighborsUsingHtable` should use `getCityFromList` and `getCityFromHtable`, respectively.

```


    /**
     * take the name of a city as input and return the list of structures of its
     * direct neighbors using getCityFromList
     *
     * @param city
     */
    public static ArrayList<String> neighborsUsingList(String city) {
        ArrayList<String> neighbors = new ArrayList<String>();

        for (Entry<String, Integer> entry : getCityFromList(city).getNeighbors().entrySet()) {
            neighbors.add(entry.getKey());
        }

        return neighbors;
    }


```

```


    /**
     * take the name of a city as input and return the list of structures of its
     * direct neighbors using getCityFromHtable,
     *
     * @param city
     * @return
     */
    public static ArrayList<String> neighborsUsingHtable(String city) {
        ArrayList<String> neighbors = new ArrayList<String>();
        Entry<String, HashMap<String, Integer>> struct = getCityFromHtable(city);
        for (Entry<String, Integer> entry : struct.getValue().entrySet()) {
            neighbors.add(entry.getKey());
        }
        return neighbors;
    }


```

5. Write two functions `neighborsUsingList` and `neighborsUsingHtable` that take the name of a city as input and return the list of structures of its direct neighbors. `neighborsUsingList` and `neighborsUsingHtable` should use `getCityFromList` and `getCityFromHtable`, respectively.

```


    City: Arad
        Neighbors using list
        [Zerind, Sibiu, Timisoara]

        Neighbors using hash table
        [Zerind, Sibiu, Timisoara]


```

- e. Using `allCitiesHtable`, write a function `neighborsWithinD` that takes the name of a city `myCity` and a number `distance`, then returns, for all direct neighbors within distance from `myCity` ( $\leq$ ), an association list of the structures of the neighbors of `myCity` and their distance to `myCity`.

```


/**
 * Using allCitiesHtable,a function that takes the name of a city myCity and a
 * number distance, then returns, for all direct neighbors within distance from
 * myCity (LEQ), an association list of the structures of the neighbors of
 * myCity and their distance to myCity
 *
 * @return
 */
public static HashMap<String, Integer> neighborsWithinD(String myCity, int distance) {
    HashMap<String, Integer> neighbors = new HashMap<String, Integer>();
    Entry<String, HashMap<String, Integer>> city = getCityFromHtable(myCity);
    HashMap<String, Integer> everyNeighbor = city.getValue();
    for (Entry<String, Integer> entry : everyNeighbor.entrySet()) {
        String cite = entry.getKey();
        int dist = entry.getValue();
        if (distance <= dist) {
            neighbors.put(cite, dist);
        }
    }
    return neighbors;
}


```

6. Using allCitiesHtable, write a function neighborsWithinD that takes the name of a city myCity and a number distance, then returns, for all direct neighbors within distance from myCity (less than or equal), an association list of the structures of the neighbors of myCity and their distance to myCity

```


City: Arad
Neighbors within 204 km
NONE


```

- f. Using *allCitiesHtable*, write a function *neighborsP* that takes the name of two cities *cityOne* and *cityTwo*, and returns the distance between them if they are directly connected or *nil* if they are not.

```


/**
 * Using allCitiesHtable, a function that takes the name of two cities cityOne
 * and cityTwo, and returns the distance between them if they are directly
 * connected or nil if they are not
 *
 * @return
 */
public static int neighborsP(String cityOne, String cityTwo) {
    int dist = 0;
    HashMap<String, Integer> neighbors = allCitiesHash.get(cityOne);
    for (Entry<String, Integer> entry : neighbors.entrySet()) {
        if (entry.getKey().equals(cityTwo)) {
            dist = entry.getValue();
        }
    }
    return dist;
}


```

7. Using `allCitiesHtable`, write a function `neighborsP` that takes the name of two cities `cityOne` and `cityTwo`, and returns the distance between them if they are directly connected or `None` if they are not.

Example: Arad and Hirsova  
distance: 0

## 4.2 Implementing Search (50 points)

You are asked to implement search the following search strategies, first as a TREE-SEARCH then as a GRAPH-SEARCH:

- Any uninformed search strategy of your choice, 10 points
- A Greedy search strategy, and 10 points
- An A\* search strategy. 10 points

### a. Uninformed search strategy: Uniform Cost Search

→ Uniform-cost expands first lowest-cost node on the fringe

Uniform Cost Tree Search (Uninformed)						
City Name	#nodes visited	Path to Bucharest		Total cost of path	CPU time	
Arad	53	[Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		418	398500 ns	
Bucharest	1	[Bucharest]		0	0 ns	
Craiova	8	[Craiova, Pitesti, Bucharest]		239	177000 ns	
Dobreta	21	[Dobreta, Craiova, Pitesti, Bucharest]		359	445000 ns	
Eforie	6	[Eforie, Hirsova, Urziceni, Bucharest]		269	80000 ns	
Fagaras	5	[Fagaras, Bucharest]		211	84000 ns	
Giurgiu	2	[Giurgiu, Bucharest]		90	28000 ns	
Hirsova	5	[Hirsova, Urziceni, Bucharest]		183	69000 ns	
Iasi	11	[Iasi, Vaslui, Urziceni, Bucharest]		319	158000 ns	
Lugoj	73	[Lugoj, Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		504	2292000 ns	
Mehadia	46	[Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		434	878000 ns	
Neamt	12	[Neamt, Iasi, Vaslui, Urziceni, Bucharest]		411	93000 ns	
Oradea	45	[Oradea, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		429	979000 ns	
Pitesti	3	[Pitesti, Bucharest]		101	19000 ns	
Rimnicu Vilcea	8	[Rimnicu Vilcea, Pitesti, Bucharest]		198	67000 ns	
Sibiu	17	[Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		278	169000 ns	
Timisoara	91	[Timisoara, Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		536	2465000 ns	
Urziceni	2	[Urziceni, Bucharest]		85	10000 ns	
Vaslui	6	[Vaslui, Urziceni, Bucharest]		227	37000 ns	
Zerind	93	[Zerind, Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		493	2879000 ns	

Uniform Cost Graph Search (Uninformed)						
City Name	#nodes visited	Path to Bucharest		Total cost of path	CPU time	
Arad	14	[Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		418	1092000 ns	
Bucharest	1	[Bucharest]		0	0 ns	
Craiova	8	[Craiova, Pitesti, Bucharest]		239	366000 ns	
Dobreta	10	[Dobreta, Craiova, Pitesti, Bucharest]		359	370000 ns	
Eforie	4	[Eforie, Hirsova, Urziceni, Bucharest]		269	218000 ns	
Fagaras	4	[Fagaras, Bucharest]		211	117000 ns	
Giurgiu	2	[Giurgiu, Bucharest]		90	44000 ns	
Hirsova	4	[Hirsova, Urziceni, Bucharest]		183	100000 ns	
Iasi	5	[Iasi, Vaslui, Urziceni, Bucharest]		319	145000 ns	
Lugoj	15	[Lugoj, Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		504	389000 ns	
Mehadia	12	[Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		434	316000 ns	
Neamt	5	[Neamt, Iasi, Vaslui, Urziceni, Bucharest]		411	131000 ns	
Oradea	12	[Oradea, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		429	303000 ns	
Pitesti	3	[Pitesti, Bucharest]		101	62000 ns	
Rimnicu Vilcea	6	[Rimnicu Vilcea, Pitesti, Bucharest]		198	133000 ns	
Sibiu	11	[Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		278	218000 ns	
Timisoara	16	[Timisoara, Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		536	466000 ns	
Urziceni	2	[Urziceni, Bucharest]		85	39000 ns	
Vaslui	5	[Vaslui, Urziceni, Bucharest]		227	91000 ns	
Zerind	18	[Zerind, Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		493	515000 ns	

### b. Greedy Search strategy: Best First Search

Greedy best-first search chooses the node  $n$  closest to the goal such as  $h(n)$  is minimal

Greedy Best First Tree Search					
City Name	#nodes visited	Path to Bucharest		Total cost of path	CPU time
Arad	4	[Arad, Sibiu, Fagaras, Bucharest]	[Bucharest]	450	47000 ns
Bucharest	1			0	0 ns
Craiova	3	[Craiova, Pitesti, Bucharest]		239	41000 ns
Dobreta	4	[Dobreta, Craiova, Pitesti, Bucharest]		359	39000 ns
Eforie	4	[Eforie, Hirsova, Urziceni, Bucharest]		269	48000 ns
Fagaras	2		[Fagaras, Bucharest]	211	17000 ns
Giurgiu	2		[Giurgiu, Bucharest]	90	38000 ns
Hirsova	3		[Hirsova, Urziceni, Bucharest]	183	35000 ns
Iasi	4		[Iasi, Vaslui, Urziceni, Bucharest]	319	26000 ns
Lugoj	6	[Lugoj, Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		504	38000 ns
Mehadia	5	[Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		434	72000 ns
Neamt	5		[Neamt, Iasi, Vaslui, Urziceni, Bucharest]	411	45000 ns
Oradea	4		[Oradea, Sibiu, Fagaras, Bucharest]	461	98000 ns
Pitesti	2		[Pitesti, Bucharest]	101	11000 ns
Rimnicu Vilcea	3		[Rimnicu Vilcea, Pitesti, Bucharest]	198	34000 ns
Sibiu	3		[Sibiu, Fagaras, Bucharest]	310	16000 ns
Timisoara	7	[Timisoara, Lugoj, Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		615	125000 ns
Urziceni	2		[Urziceni, Bucharest]	85	9000 ns
Vaslui	3		[Vaslui, Urziceni, Bucharest]	227	33000 ns
Zerind	5		[Zerind, Arad, Sibiu, Fagaras, Bucharest]	525	46000 ns

Greedy Best First Graph Search					
City Name	#nodes visited	Path to Bucharest		Total cost of path	CPU time
Arad	4	[Arad, Sibiu, Fagaras, Bucharest]	[Bucharest]	450	38000 ns
Bucharest	1			0	0 ns
Craiova	3	[Craiova, Pitesti, Bucharest]		239	24000 ns
Dobreta	4	[Dobreta, Craiova, Pitesti, Bucharest]		359	33000 ns
Eforie	4	[Eforie, Hirsova, Urziceni, Bucharest]		269	33000 ns
Fagaras	2		[Fagaras, Bucharest]	211	15000 ns
Giurgiu	2		[Giurgiu, Bucharest]	90	19000 ns
Hirsova	3		[Hirsova, Urziceni, Bucharest]	183	25000 ns
Iasi	4		[Iasi, Vaslui, Urziceni, Bucharest]	319	65000 ns
Lugoj	6	[Lugoj, Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		504	49000 ns
Mehadia	5	[Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		434	43000 ns
Neamt	5		[Neamt, Iasi, Vaslui, Urziceni, Bucharest]	411	37000 ns
Oradea	4		[Oradea, Sibiu, Fagaras, Bucharest]	461	36000 ns
Pitesti	2		[Pitesti, Bucharest]	101	15000 ns
Rimnicu Vilcea	3		[Rimnicu Vilcea, Pitesti, Bucharest]	198	65000 ns
Sibiu	3		[Sibiu, Fagaras, Bucharest]	310	23000 ns
Timisoara	7	[Timisoara, Lugoj, Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		615	58000 ns
Urziceni	2		[Urziceni, Bucharest]	85	13000 ns
Vaslui	3		[Vaslui, Urziceni, Bucharest]	227	23000 ns
Zerind	5		[Zerind, Arad, Sibiu, Fagaras, Bucharest]	525	86000 ns

### c. $A^*$ Search: Best First $A^*$ search

$A^*$  search chooses the least-cost solution

$$\text{solution cost } f(n) \left\{ \begin{array}{l} g(n): \text{cost from root to a given node } n \\ + \\ h(n): \text{cost from the node } n \text{ to the goal node} \end{array} \right.$$

such as  $f(n) = g(n) + h(n)$  is minimal

Best First A* Tree Search						
City Name	#nodes visited	Path to Bucharest		Total cost of path	CPU time	
Arad	6	[Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		418	52000 ns	
Bucharest	1	[Bucharest]		0	0 ns	
Craiova	3	[Craiova, Pitesti, Bucharest]		239	20000 ns	
Dobreta	5	[Dobreta, Craiova, Pitesti, Bucharest]		359	29000 ns	
Eforie	4	[Eforie, Hirsova, Urziceni, Bucharest]		269	22000 ns	
Fagaras	2	[Fagaras, Bucharest]		211	11000 ns	
Giurgiu	2	[Giurgiu, Bucharest]		90	9000 ns	
Hirsova	3	[Hirsova, Urziceni, Bucharest]		183	17000 ns	
Iasi	4	[Iasi, Vaslui, Urziceni, Bucharest]		319	22000 ns	
Lugoj	11	[Lugoj, Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		504	136000 ns	
Mehadia	8	[Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		434	56000 ns	
Neamt	5	[Neamt, Iasi, Vaslui, Urziceni, Bucharest]		411	27000 ns	
Oradea	6	[Oradea, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		429	51000 ns	
Pitesti	2	[Pitesti, Bucharest]		101	11000 ns	
Rimnicu Vilcea	3	[Rimnicu Vilcea, Pitesti, Bucharest]		198	34000 ns	
Sibiu	5	[Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		278	49000 ns	
Timisoara	12	[Timisoara, Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		536	163000 ns	
Urziceni	2	[Urziceni, Bucharest]		85	9000 ns	
Vaslui	3	[Vaslui, Urziceni, Bucharest]		227	17000 ns	
Zerind	9	[Zerind, Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		493	67000 ns	

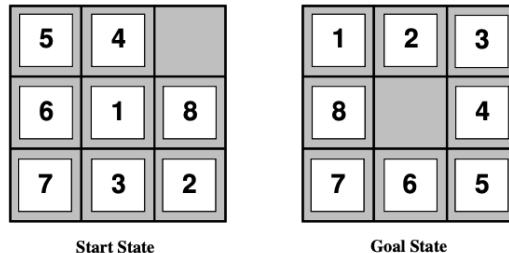
Best First A* Graph Search						
City Name	#nodes visited	Path to Bucharest		Total cost of path	CPU time	
Arad	6	[Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		418	145000 ns	
Bucharest	1	[Bucharest]		0	0 ns	
Craiova	3	[Craiova, Pitesti, Bucharest]		239	51000 ns	
Dobreta	5	[Dobreta, Craiova, Pitesti, Bucharest]		359	92000 ns	
Eforie	4	[Eforie, Hirsova, Urziceni, Bucharest]		269	75000 ns	
Fagaras	2	[Fagaras, Bucharest]		211	27000 ns	
Giurgiu	2	[Giurgiu, Bucharest]		90	24000 ns	
Hirsova	3	[Hirsova, Urziceni, Bucharest]		183	50000 ns	
Iasi	4	[Iasi, Vaslui, Urziceni, Bucharest]		319	143000 ns	
Lugoj	8	[Lugoj, Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		504	169000 ns	
Mehadia	6	[Mehadia, Dobreta, Craiova, Pitesti, Bucharest]		434	116000 ns	
Neamt	5	[Neamt, Iasi, Vaslui, Urziceni, Bucharest]		411	95000 ns	
Oradea	6	[Oradea, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		429	140000 ns	
Pitesti	2	[Pitesti, Bucharest]		101	25000 ns	
Rimnicu Vilcea	3	[Rimnicu Vilcea, Pitesti, Bucharest]		198	51000 ns	
Sibiu	5	[Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		278	93000 ns	
Timisoara	12	[Timisoara, Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		536	282000 ns	
Urziceni	2	[Urziceni, Bucharest]		85	24000 ns	
Vaslui	3	[Vaslui, Urziceni, Bucharest]		227	52000 ns	
Zerind	9	[Zerind, Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]		493	384000 ns	

Source code submitted on Handin

## 5 Eight-Piece Sliding Puzzle (Bonus 80 points)

The goal is to implement A\* search for solving the Eight-Piece Sliding Puzzle Problem with the two admissible heuristics: the displaced tile and the Manhattan distance heuristics.

$$\text{8-puzzle: } \begin{cases} h_1(n) = \text{number of misplaced tiles} \\ h_2(n) = \text{total Manhattan distance} \end{cases}$$



I think I have a solution (displaced tiles) for the bonus but I'm not able to generate random solvable states. Source code for problem 5 is also submitted