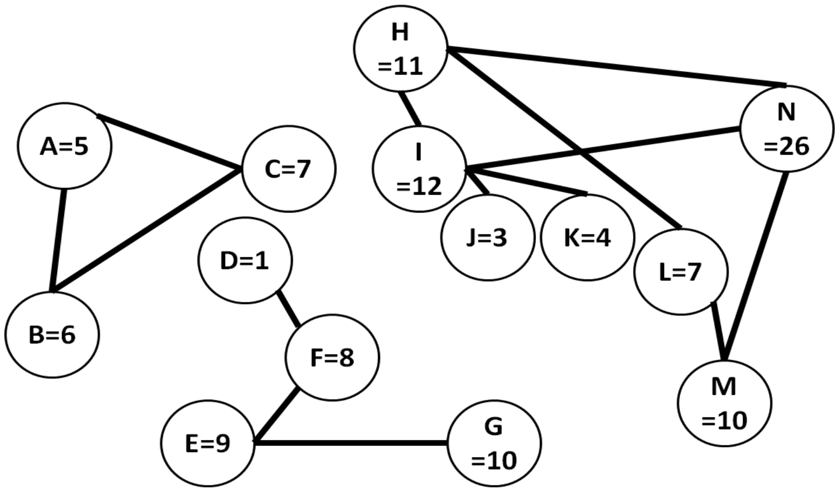
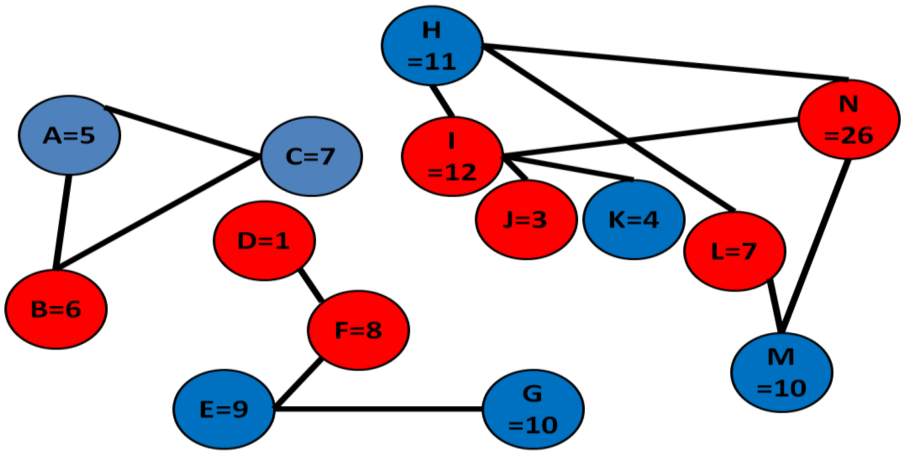
CSCI561 – Introduction to Artificial Intelligence
Instructor: Dr. K. Narayanaswamy
Assignment 3 – Adversarial Search
War simulation game
Due: 04/04/2014 11:59:59pm

1. Introduction

In this project, you will write computer bots to play the War simulation game using Greedy (Minimax with cut-off depth = 1), Minimax and Alpha-Beta pruning algorithms. The red color is Union and blue color is Confederacy. The original idea of this game is from[1]. The game is played on the graph. Each node (a city) contains resources. The goal of this game is to capture more resources than the opponent. The image below shows the initial state of the map in the simulation game, namely when no player has yet to capture any resources.



The game ends when both players cannot make any more moves. The image below shows one possible end state.



2. Tasks

In this assignment, you will write a program to implement the following algorithms for the Union player. The Confederacy always uses the Greedy algorithm.

2.1 The Union player uses the Greedy algorithm (Minimax with cut-off depth = 1);

2.2 The Union player uses the Minimax algorithm;

2.3 The Union player uses the Alpha-Beta pruning algorithm;

3. Rules and Illustrative Examples

3.1 Players

Union = Red, Confederacy = Blue

Assume that the Union player always makes the first move from the configuration provided.

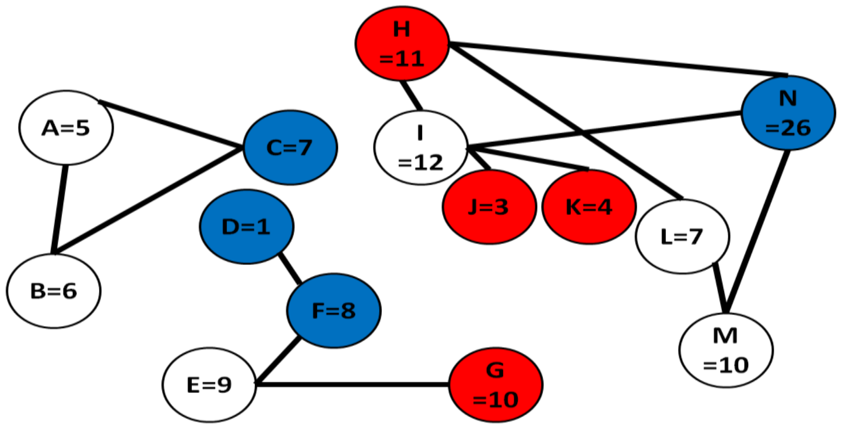
3.2 Moves

There are two kinds of moves in this game. 1) Paratroop drop; 2) Force march.

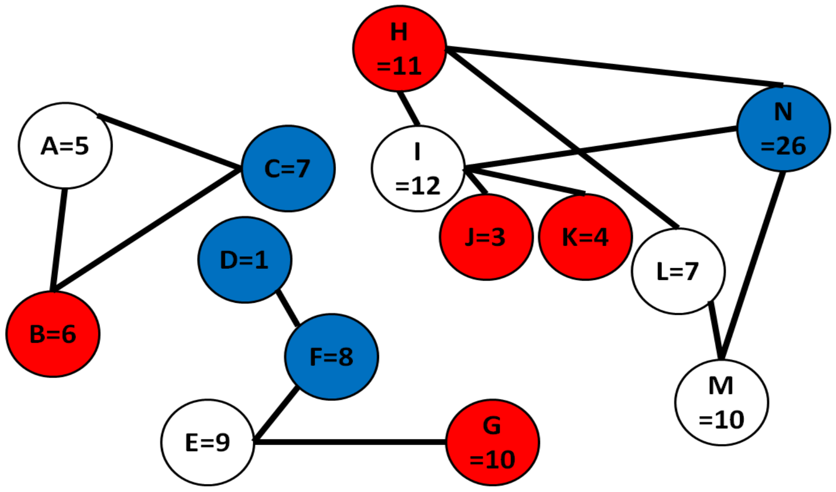
3.2.1 Paratroop drop

You can take **<u>any open space</u>** on the board with a Paratroop drop. This will create a new piece on the board for the player making the move. This move can be made as many times as one wants to during the game, but only once per turn. However, a Paratroop drop does not conquer any pieces. It simply allows one to arbitrarily place troops **<u>anywhere unoccupied on the board</u>**. Once you have done a Paratroop Drop, your turn is completed[1].

Let us say the current game state is shown in the image below and the current turn is Union's (red). You can use a Paratroop drop to any empty city (the white color). In the example below, the Union can use a Paratroop drop command to occupy ***A, B, E, I, L,*** or ***M***.



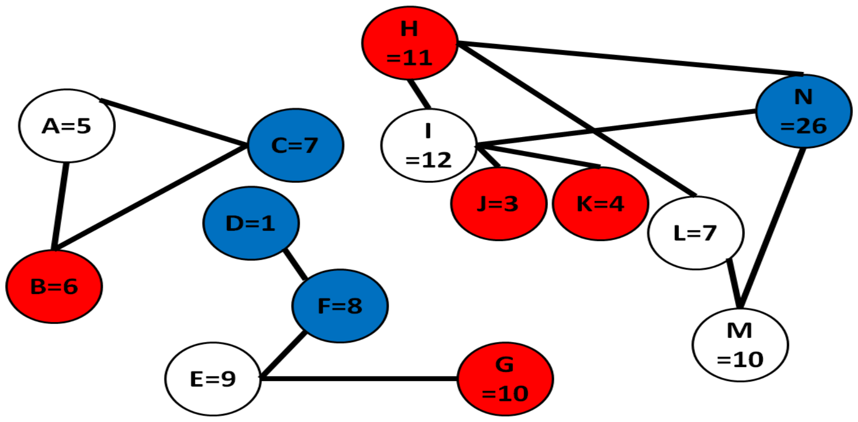
For example, if the Union uses a Paratroop drop command to occupy **B.** The game state is updated to



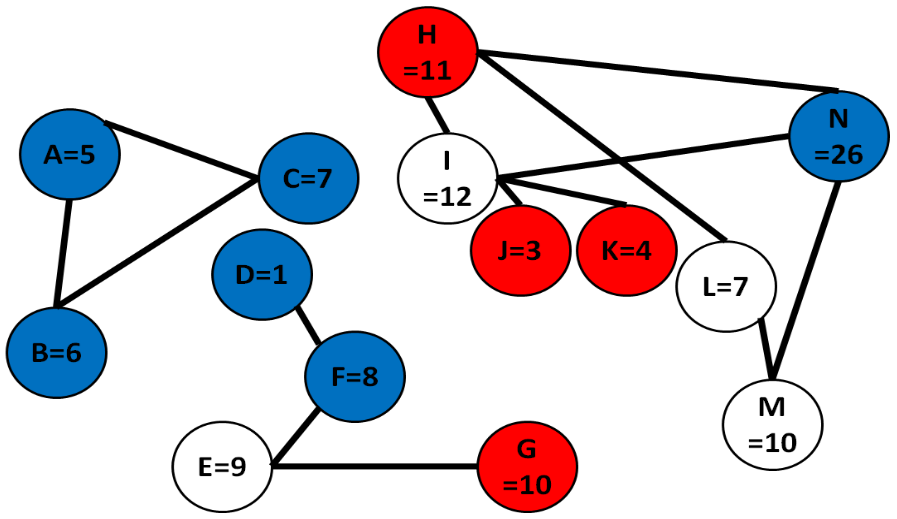
3.2.1 Force march

In any of your turns, with a Force march command, you can take an unoccupied city next to any of the cities that you have already occupied. The city you have originally held is still occupied. Thus, you get to create a new piece for yourself by using the Force march command. A Force march move can also trigger immediate "conquering" consequences as follows: **any of your opponent's cities adjacent to the city you have taken using the Force march command is also turned to your side** (you conquer those cities as the consequence of the initial Force march move). **A Force march move can be used even if it does not trigger additional conquering consequences as just described.** Once you have made this move, your turn is over[1].

Let us say the current game state is shown in the image below and the current turn is the Confederacy's (blue). You can use the Force march command to take the unoccupied city next to any city of yours . In the following example, the Confederacy can use the Force march command to occupy ***A, E, I,*** or ***M***. Note that the Confederacy cannot obtain **L** by the Force march command because the Confederacy does not occupy any city which is adjacent to **L**.

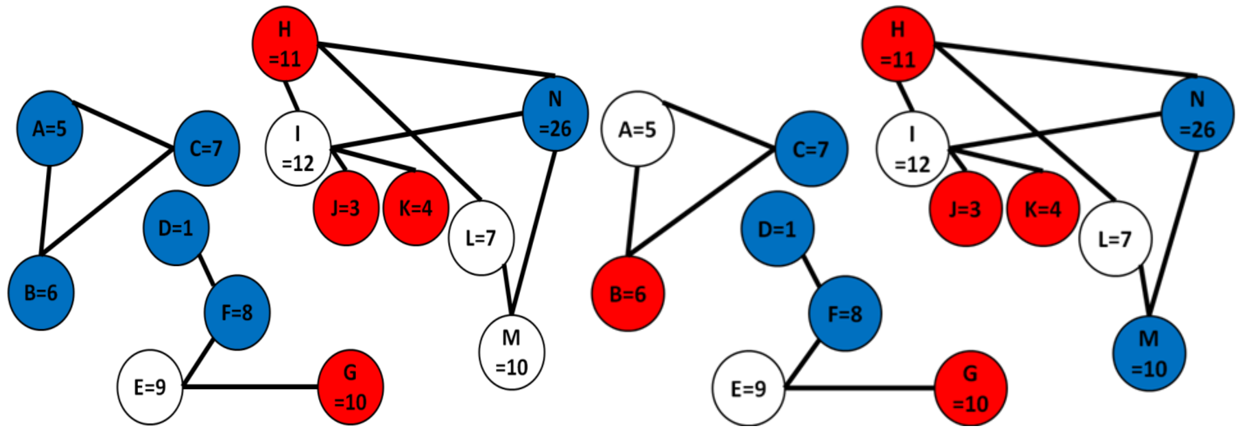


For example, if the Confederacy uses the Force march command to occupy **A.** The game state is updated to



Please note that when Confederacy uses the Force march command to occupy **A.** B is conquered as the consequence of the Force march move because B is occupied by the Union (Red) and adjacent to A. A Force march move can also be done without any consequence of conquering any additional other cities.

Let us compare these two situations using the two below images. The left image is after the Union makes a Force march move to occupy A. The right image is after the Union makes a Force march move to occupy M.

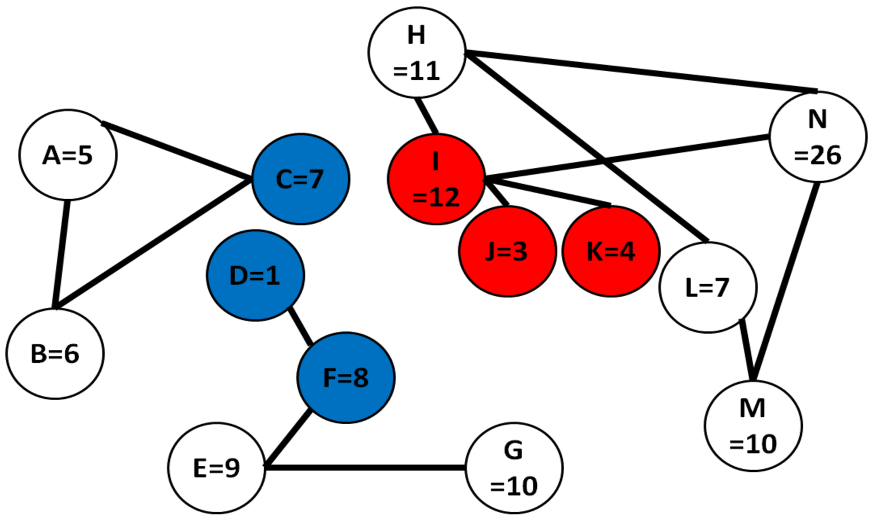


3.3 Evaluation functions

The evaluation function can be computed by

$$E(s) = \text{Resource\_your\_cities} - \text{Resource\_opponent\_cities}$$

For example, if the current state is:

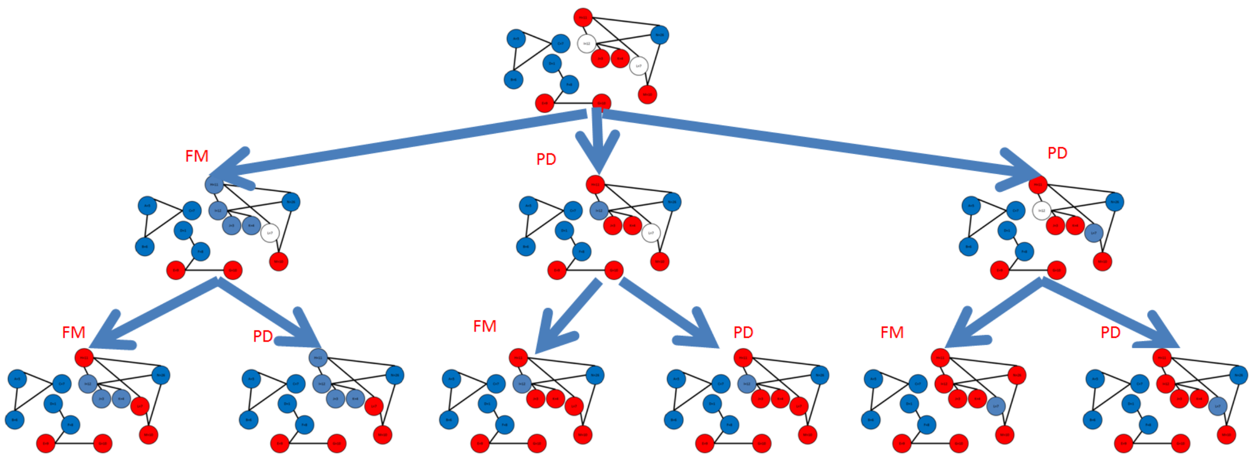


Union   : E(s) = (12+3+4) - (7+1+8)   = 3

Confederacy : E(s) = (7+1+8) - (12+3+4) = -3

3.4 Search tree

Suppose the current player is the Confederacy. Below is an example of the game search tree. You need to implement Greedy, Minimax and Alpha-Beta pruning on this tree. The Greedy search is a special case of Minimax where the cut-off depth is always only 1. We will discuss the details in the next subsection.

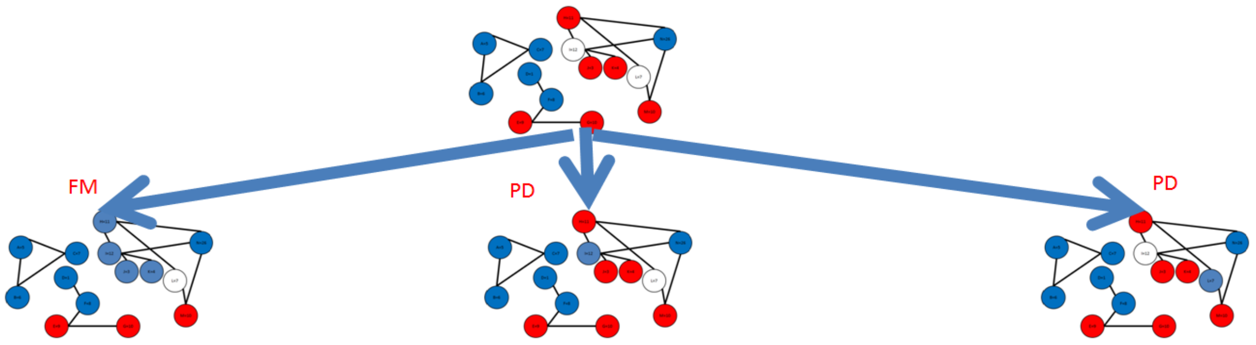


There are two empty cities **I** and **L**. The Confederacy can use Force march (FM) to occupy only one city that is **I.** It cannot be **L to be occupied** because the Confederacy has not occupied any cities that are adjacent to **L**. However, it can use the paratroop drop command to occupy either **I** or **L.** Thus, there are three possible actions for the Confederacy to play in this state.

3.5 Greedy Search

It is a special case of Minimax. The cut-off depth is always 1. Thus, the algorithm is very simple. You only need to pick the action which has **the highest evaluation value**. In this example, using Force march to occupy **I** has the highest value. Thus, Greedy will return using Force march to occupy **I** as an action.

You do not need to explicitly create a tree for this algorithm because the depth is always one. Instead, you can simply make a list of actions, then, choose the action that leads you to the state which has highest value. Try to debug your game mechanisms (moves, end game, state updates) with this task as it is the simplest one.



3.6 Minimax and Alpha-Beta pruning

Pseudo codes in Figure 5.3 and 5.7 in AIMA are used to produce the examples of input/desired output posted on the Blackboard for Minimax and Alpha-Beta. Thus, to eliminate variations in log files, figure 5.3 in AIMA must be used to implement Minimax algorithm. Figure 5.7 in AIMA must be used to implement Alpha-Beta algorithm.

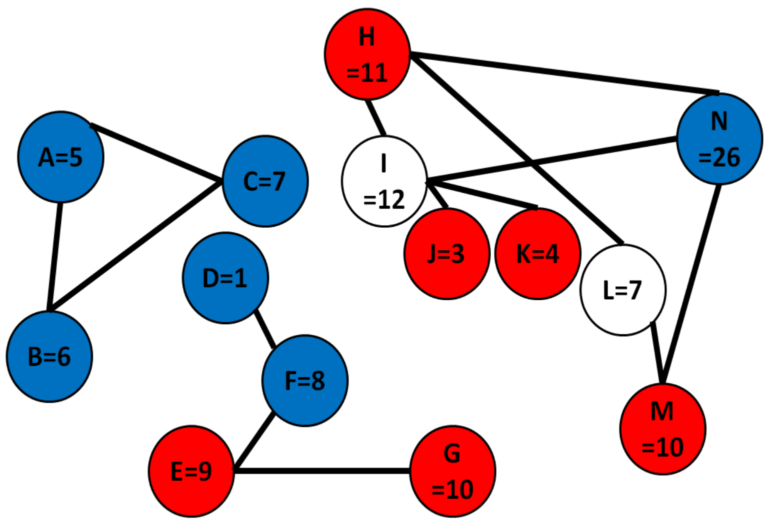
3.7 Tie Breaking and Expand order

When you expand a node, use these rules:

1) Expand "Force march" before "Paratroop drop"

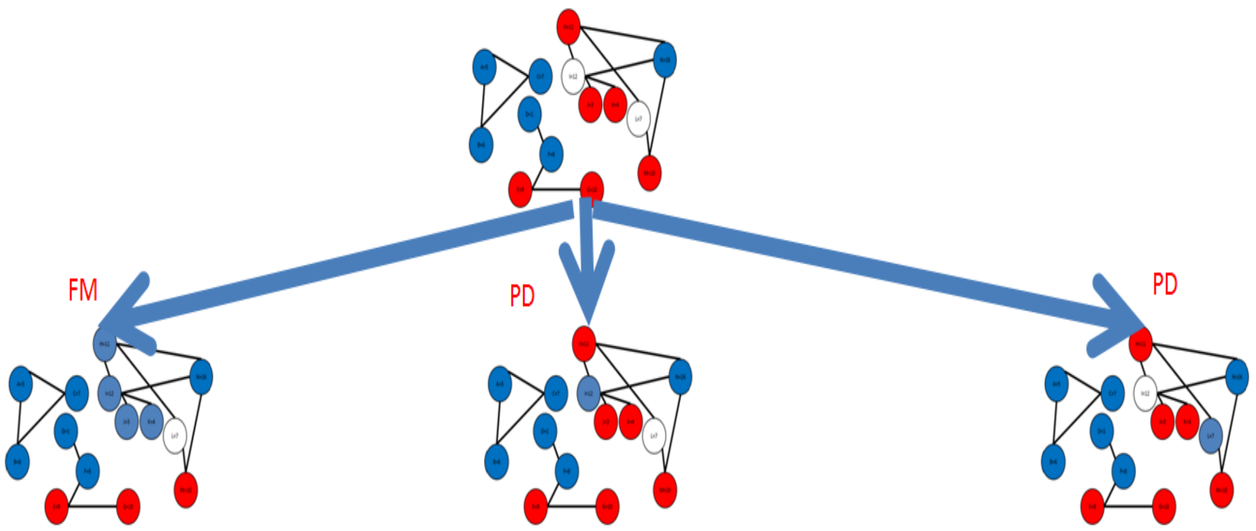
2) Expand the destination city in alphabetical order.

For example, if the current player is the Confederacy and current game state is ,



There are only three available actions. 1. Paratroop drop to L 2. Paratroop drop to I 3. Force march to I.

According to the expanding rule, you need to choose Force March action first. Thus, you expand the Force march to I first. Now, we have two more actions to expand 1. Paratroop drop to L 2. Paratroop drop to I. Both of them are Paratroop drop action. Thus, we use the second rule of expanding. We need to expand Paratroop drop to I before Paratroop drop to L because I is before L in alphabetical order. Thus, your tree should be like:



4. Inputs

There are three inputs for your programs:

4.1 Which task to perform: there are three possible values

1 ➔ The Union player uses the Greedy algorithm (Minimax with cut-off depth = 1);

2 ➔ The Union player uses the Minimax algorithm;

3 ➔ The Union player uses the Alpha-Beta pruning algorithm;

Remember: The Confederacy always uses the Greedy algorithm.

4.2 The cut off depth

More details in AIMA section 5.4.2

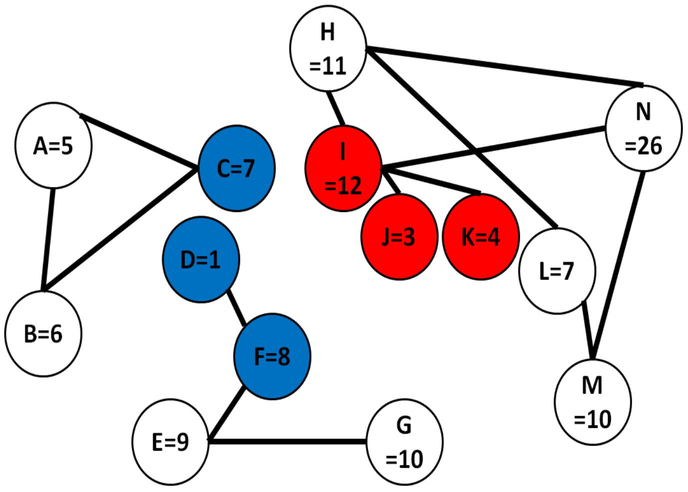
4.3 The map file:

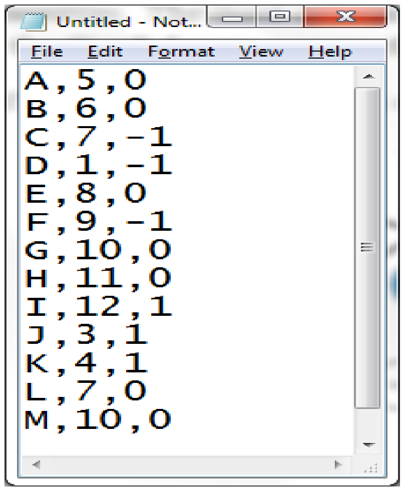
The map is provided in the form of an adjacency list. The graph is assumed to be un-weighted and undirected. Each line represents an edge of a graph. There are two columns per line separated by comma. These two columns represent the nodes of an edge.

4.4 The initial board configuration:

There are three columns per line separated by comma. The number of line represent number of cities (node). The node name is already sorted in alphabetical order. The first column is a city name. The second is its resources. The third column is who already occupy it. 1 means the Union occupy the city, -1 means the Confederacy occupy the city. 0 means that no one occupy the city.

For example, if the initial configuration of the board is on the left image, the right image show the values in the initial board file.

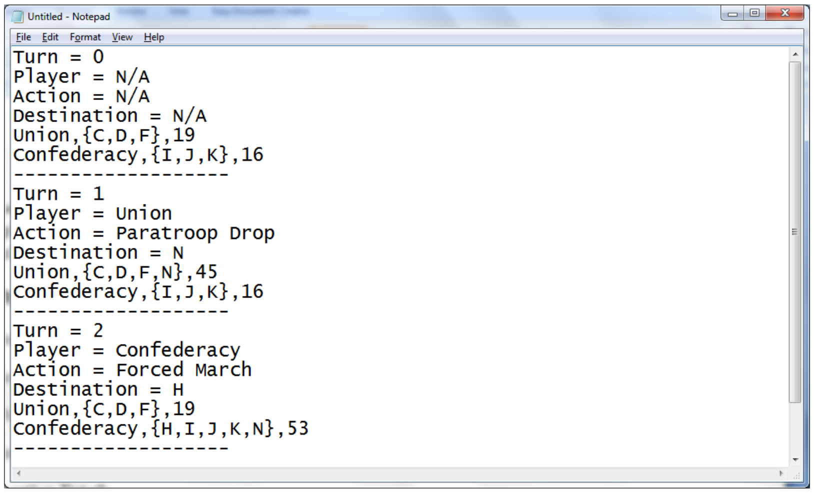




```
A,5,0
B,6,0
C,7,-1
D,1,-1
E,8,0
F,9,-1
G,10,0
H,11,0
I,12,1
J,3,1
K,4,1
L,7,0
M,10,0
```

5. Output

5.1 Moves: List the current player and his action in each turn. Also, provide the cities that Union and Confederacy occupy with sum of resources values after taking actions. For example,



```
Turn = 0
Player = N/A
Action = N/A
Destination = N/A
Union,{C,D,F},19
Confederacy,{I,J,K},16
-------------------
Turn = 1
Player = Union
Action = Paratroop Drop
Destination = N
Union,{C,D,F,N},45
Confederacy,{I,J,K},16
-------------------
Turn = 2
Player = Confederacy
Action = Forced March
Destination = H
Union,{C,D,F},19
Confederacy,{H,I,J,K,N},53
-------------------
```

5.2 Logs of the first step: Show complete traverse logs of the first step. Please see the complete examples in output1_tlog_t1.txt, output1_tlog _t2.txt, output1_ tlog_t3.txt, output2_tlog_t1.txt, output2_tlog_t2.txt, output2_tlog_t3.txt.

6 Program Specifications

6.1 Your program must be written in either Java or C++.

6.2 Your program MUST compile and run on aludra.usc.edu

6.3 Write your own code. Files will be compared and any cheating will be reported.

6.4 Your program name must be "war" (without quotation mark).

7. Execution Details

Your program will be tested on aludra.usc.edu on unseen input files that meet the input file specifications. Your program will receive 6 arguments, 4 for inputs and 2 for outputs.

7.1 C/C++

The grader will execute this command:

```
war -t <task> -d < cut_off_depth> -m<map_file>  -i <init > -op <output_path> -ol <output_log>
```

Example:

```
war -t 1 -d 3 -m map1.txt -i init1.txt-op output1_moves_greedy.txt -ol output1_tlog_ greedy.txt
```

7.2 JAVA

The grader will execute this command:

```
java war -t <task> -d < cut_off_depth> -m<map_file>  -i <init > -op <output_path> -ol <output_log>
```

Example:

```
java war -t 1 -d 3 -m map1.txt -i init1.txt-op output1_moves_greedy.txt -ol output1_tlog_ greedy.txt
```

7.3 Arguments:

7.3.1 <task> : there are 3 possible values 1, 2 and 3.

7.3.2 < cut_off_depth >: the cut-off depth

7.3.3 <map_file>: location of a map file.

7.3.4 <init>: location of a initial board configuration file.

7.3.5 <output_path>: location of an path output.

7.3.6 <output_log>: location of an traverse log output.

Thus, you should interpret the example:

```
war -t 1 -d 3 -m map1.txt -i init1.txt-op output1_moves_greedy.txt -ol output1_tlog_ greedy.txt
```

The first task is chosen. The cut-off depth is 3. The location of the map file is same as your program and its name is map1.txt. The location of the initial board configuration file is same as your program and its name is init1.txt. The location of path output file is same as your program and its name is output1_moves_greedy.txt. Finally, the location of traverse log output file is same as your program and its name is output1_tlog_ greedy.txt.

8.1 Programming (90 pts):

Your program must implement the three search algorithms.

8.1.1 Correct outputs for task 1: 20 points

8.1.2 Correct outputs for task 2: 30 points

8.1.3 Correct outputs for task 3: 30 points

8.1.4 Your analysis of similarities/differences in terms playing performance/runtime/number of iterations between task1, task2 and task3. Your explanation must be included as part of readme.txt: 10 points

8.2 Readme.txt (10 pts):

8.2.1 A brief description of the program structure and any other issues you think will be helpful for the grader to understand your program. (5 pts)

8.2.2 Instructions on how to compile and execute your code. (5 pts)

8.2.3 Please include your name, student ID and email address on the top.

8.2.4 You must submit a program in order to get any credit for the Readme.txt. In short, if you submit ONLY a Readme.txt file you will get 0.

8.2.5 Remember to also include the explanation of outputs (Part 8.1.4)

9. Submission Guidelines

Your program files will all be submitted via blackboard. You MUST follow the guidelines below. Failure to do so will incur a -25 point penalty.

9.1 Compress and zip ALL your homework files (this includes the Readme.txt and all source files) into one .zip file. Note that only .zip file extensions are allowed. Other compression extensions such as .tar, rar, or 7z will NOT be accepted.

9.2 Name your zip file as follows: firstname_lastname.zip. For example, John_Smith.zip would be a correct file name.

9.3 To submit your assignment, simply select the appropriate assignment link from the Assignments subsection of the course blackboard website. Upload your zip file and click submit (clicking send is not enough).

Please make sure ALL source files are included in your zip file when submitted. Errors in submission will be assessed –25 points. A program that does not compile as submitted will be given 0 points.

Only your FINAL submission will be graded.

For policies on late submissions, please see the Syllabus from the course home page.

References

[1]. http://www.cool-ai.com/minimax.alpha-beta/minimax.alpha-beta.homework.pdf