Problem Specification Files Documentation:

The following will explain the contents and options of the Problem Specification File. Quotation marks are used to denote the text in the file. Quotes should not be found in the input file.

**Note: All input files should be contained in a folder titled “TextFiles” and should have extension “.txt”; this folder should be located in the same directory as main.cpp**

Line 1 – The Search Strategy

* “DFS” – Depth-First Search
* “BFS” – Breadth-First Search
* “A\*” – A\* Search
* “GBFS” – Greedy Best-First Search
* “IDS” – Iterative Deepening Search
* “DLS” – Depth Limited Search

Line 2 – Depth Cutoff

* “0” – Used when Strategy is not DLS
* “(integer > 0)” – Used when strategy is DLS

Line 3 – The Exploration Cutoff

* “(integer > 0)” – represents the number of nodes to be expanded before the user is asked if they want the program to keep searching for the solution. If 0 is entered, no cutoff is used.
  + Note: some knowledge of the problem is recommended to choose an appropriate value.

Line 4 – Problem

* “MCP” - Missionaries and Cannibals Puzzle
* “Pegs” – Peg Solitaire game.

Line 5 – Problem Modifications

This is a problem specific option. Only one format will work for each problem. Formats are as follows.

* “(integer 1-6)” – Board shape for Peg Solitaire
  + These board shapes are synonymous with the description of this game found on [Wikipedia](https://en.wikipedia.org/wiki/File:Peg_Solitaire_game_board_shapes.svg).
  + Also know that your initial and final state input files must match up exactly to the board game shape you pick.
* Do not enter any value on this line for the Missionaries and Cannibals Puzzle

Example files are listed below. Bold is used to denote the “standard” game state for each puzzle.

PegSpec1.txt

DFS

0

**0**

**Pegs**

**4**

MCPSpec1.txt

DFS

0

**0**

**MCP**

PegSpec2.txt

DLS

32

1000

Pegs

4

MCPSpec2.txt

A\*

0

1500

MCP

Initial State and Goal State Files:

The initial state and goal states are represented as such:

* Pegs – A text file containing an exact visual representation of the board.
  + 1 – peg in this position
  + 0 – open hole in this position (no peg, but still a valid position for pegs)
  + X – An illegal position (out of bounds, pegs cannot jump to this position)
* MCP – A text file containing three integer values
  + 1st – Number of Missionaries present on the initial side (non-goal side)
  + 2nd – Number of Cannibals present on the initial side
  + 3rd – The capacity of the boat
    - It is assumed that the boat will start on the initial side

**Grading Criteria:**

* **Search Shell/Problem Formulation**
  + **Problem/strategy independent search algorithm?**

Yes, the search is independent from the strategy as it mainly use polymorphic call to the SearchEngine class that implements all the searches with an (Abstract) class called Node that has some pure virtual methods such as;   
 virtual bool equals(Node\* node)=0; // to check whether a given node is equivalent to another node (checking equality of state of those nodes if equal)

virtual void printState()=0; (print the state)

virtual bool goalStateTest()=0; (test equality against the goal state)

virtual vector<Node\*> successorFunction()=0; (returns all successor states wrapped in a Node)

virtual int stepCost(int move)=0; (returns the stepCost of each move)

Moreover, the Node (abstract) class has also some attributes:

Node \*prev = NULL; // to know what node expanded to this one (helps construct the search tree and the final steps of the solution)

int pathcost = 0; (store the path cost from the initial state to the state of this specific node)

int HeuristicValue =0; (holds the heuristic value to reach the goal state)

Therefore, the idea is that PegsLogicNode would extends the class Node and so MCPNode so to implement the element to constitute a Node to make a search abstract and non-related to the logic of the game.

* + **Use of general goal test?**

Yes, the Node abstract class formulates that there is a need for a goal test method for any class that implements it i.e. the 2 puzzles, and in the implement of each of the classes of the games logic (PegsLogicNode.h and MCPNode.h) there is a declaration for the goal tests that compares the current state to the goal vectors (global variables) that can be found in Node.h

* + **Use of general successor function?**

As for the previous question, there is an implement for successor function for each of PegsLogic and MCP that can be found in their respective classes.

* + **Use of general path cost function?**

The same as in the previous questions, the path cost function (called stepCost(int move)) takes a certain move and returns its cost and this function is used within our successor functions. However, most of the time the function returns 1 as there is no difference in the moves.

* + **Clear representation of closed list?**

The closet list is represented by a class called ExploredSet.h that we have implemented that stores all the Node (no matter which type of course since we are using polymorphism). This class implement the methods (add, exists, size) add method adds a node to a vector to store it, exists checks within the already stores list of Node whether a similar Node with the same state exists and returns a bool and size returns the size of the ExploredSet. The ExploredSet is then used in all searches in a polymorphic manner.

* + **Clear representation of frontier (open list)?**

Data structure to be used for the frontier is depended on the search. Of course, again the used of the frontier uses polymorphism to separate the logic of the game from the search.

DFS => Stack,

BFS => Queue,

UCS=> min Priority Queue with comparator that compares only the path cost

Greedy=> min Priority Queue with comparator that compares only the heuristic value

Astar=> min Priority Queue with comparator that compares the heuristic value and path cost

* + **Good search node representation?**Yes, as stated in the first question. The abstract class Node defines a number of methods that have to be implemented by the other class that defines the puzzles logics (PegsNodeLogic, MCPNode). Moreover, it has 3 attributes as said: prev (previous Node), heuristic value from that state to the goal, and path cost from initial to that state. Find more information in the Node.h class
* **Search Progress Reporting and Control**
  + **Good solution reporting (or graceful failure)?**

Yes, for every search, as soon as it find the solution, it prints the nodes expanded, the loop detected and the path cost to that solution found, and prints state by state from the initial state to the goal state all the step or a display of intermediate state to reach the goal state  
Of course, if at the end, it does not find the solution, it prints that it fails, with the following information: loops detected, and nodes expanded.

* + **Reports number of expanded nodes?**

Yes, as stated in the first question, it reports node when it fails and succeed or when the search is interrupted after X number of nodes expanded.

* + **Reports loop detection and total number of loops detected?**

Yes, at every expansion of a node, it checks whether its successor are in the loop. If yes it displays it and add one to a counter. If not, it adds to the frontier, and says it as well. And of course, as stated earlier, the # of loops detected is displayed after failure, or solution found, or interruptions.

* + **Can stop search gracefully?**

Yes, the search can stop gracefully by specifying the # of nodes you wants to expand before being asked whether you want to continue and expanded again that number of nodes before being asked again and again. This operation is specified in this line of code (PS: if you specify zero as the # of nodes to expand, it will not ask you and expand as much as it can) : if(n\_nodes\_to\_expand\_to\_ask != 0 && n\_expension != 0 && n\_expension % n\_nodes\_to\_expand\_to\_ask == 0){

* **Interface to Specific Problems/Puzzles**
  + Good way of inputting problem-specific state information?
    - We think so! (see above)
  + Good way of providing problem-specific function information?
    - We think so! (see above)
* **Search Strategies**
  + **Good way of specifying search strategies to use?**

You can specify the problem in the Specification files of any given problem as the first argument.

For example, in the file MCPSpec5GBFS the first line is GBFS which means Greedy best first search and will therefore run the MCP problem(knows that from the rest of the arguments) with a greedy best first search. Different arguments for other search are: BFS, DFS, UCS, GBFS, A\*....

* + **Search strategies act by managing the frontier?**

Yes, every search uses a frontier that has a different data structure(Stack, Queue, min priority queue) depending on the search used (refer to the previous question for more details).

* + **DF**

Yes, the DF has been implemented and the code is present in the SearchEngine.h file

* + **BF**

Yes, the BF has been implemented and the code is present in the SearchEngine.h file

* + **GBeF**

Yes, the GBef has been implemented and the code is present in the SearchEngine.h file

* + **A\***

Yes, the A\* has been implemented and the code is present in the SearchEngine.h file

* **Pegs Puzzle**
  + **State representation**: the states are represented in a 2D-Vector in which 1 means a peg, 0 a hole, and X the edge (out of bound). Here is the representation of the initial state of the 5 shapes. The size of the vector is decided by inputting the pegs shape number in the first text file for specification 5th lines

if(pegs\_shape == 4 || pegs\_shape == 1) size = 7;

else if(pegs\_shape == 2 || pegs\_shape == 5) size = 9;

else if(pegs\_shape == 3) size = 8;

else if(pegs\_shape == 6) size = 5;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pegs shape 1 | Pegs shape 2 | Pegs shape 3 | Pegs shape 4 | Pegs shape 5 |
| X X 0 0 0 X X  X 0 0 0 0 0 X  0 0 0 0 0 0 0  0 0 0 0 0 0 0  0 0 0 1 0 0 0  X 0 0 0 0 0 X  X X 0 0 0 X X | X X X 0 0 0 X X X  X X X 0 0 0 X X X  X X X 0 0 0 X X X  0 0 0 0 0 0 0 0 0  0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0  X X X 0 0 0 X X X  X X X 0 0 0 X X X  X X X 0 0 0 X X X | X X 0 0 0 X X X  X X 0 0 0 X X X  X X 0 0 0 X X X  0 0 0 0 0 0 0 0  0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0  X X 0 0 0 X X X  X X 0 0 0 X X X | X X 0 0 0 X X  X X 0 0 0 X X  0 0 0 0 0 0 0  0 0 0 1 0 0 0  0 0 0 0 0 0 0  X X 0 0 0 X X  X X 0 0 0 X X | X X X X 0 X X X X  X X X 0 0 0 X X X  X X 0 0 0 0 0 X X  X 0 0 0 0 0 0 0 X  0 0 0 0 1 0 0 0 0  X 0 0 0 0 0 0 0 X  X X 0 0 0 0 0 X X  X X X 0 0 0 X X X  X X X X 0 X X X X |

* + **Goal test**

The goal test file of each shapes is specified in the 3 arguments when executing the program. You can find these in the TextFiles directory. Here are some example of how a goalStatePegs files should be presented

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GoalStatePegs1.txt | GoalStatePegs2.txt | GoalStatePegs3.txt | GoalStatePegs4.txt | GoalStatePegs5.txt |
| X X 0 0 0 X X  X 0 0 0 0 0 X  0 0 0 0 0 0 0  0 0 0 0 0 0 0  0 0 0 1 0 0 0  X 0 0 0 0 0 X  X X 0 0 0 X X | X X X 0 0 0 X X X  X X X 0 0 0 X X X  X X X 0 0 0 X X X  0 0 0 0 0 0 0 0 0  0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0 0  X X X 0 0 0 X X X  X X X 0 0 0 X X X  X X X 0 0 0 X X X | X X 0 0 0 X X X  X X 0 0 0 X X X  X X 0 0 0 X X X  0 0 0 0 0 0 0 0  0 0 0 1 0 0 0 0  0 0 0 0 0 0 0 0  X X 0 0 0 X X X  X X 0 0 0 X X X | X X 0 0 0 X X  X X 0 0 0 X X  0 0 0 0 0 0 0  0 0 0 1 0 0 0  0 0 0 0 0 0 0  X X 0 0 0 X X  X X 0 0 0 X X | X X X X 0 X X X X  X X X 0 0 0 X X X  X X 0 0 0 0 0 X X  X 0 0 0 0 0 0 0 X  0 0 0 0 1 0 0 0 0  X 0 0 0 0 0 0 0 X  X X 0 0 0 0 0 X X  X X X 0 0 0 X X X  X X X X 0 X X X X |

* + **Successor function**

The successor function is defined in the class NodePegs and is the shape for all 5 shapes(the triangular is not included as it will have another successor function that we will discuss later) and all generate a set that represents all possible node of the next moves (move hole up, left, down, right) discarding illegal state (when it hits the edgs or the hole needs two pegs adjacent to it to move)

* + **Path cost function**

The path cost function (stepCost(int move)) returns the cost of the move and it is implemented in the class of NodePegs and used in the successor method so that for every possible move that is not illegal a new node is generate with the new state and an accumulated path cost value that include the current path cost + the cost of the move (in our case, it always returns 0)

* + **Heuristics (at least one tried)**

The idea of our heuristics is that we see how many immediate possible moves are possible given the current state and count those as it serves as our heuristics. The idea is that the move possible moves you have the better it is, and you better pursue that node more than the others because the others may have pegs that are stuck in some edges or a isolated edges and therefore are not able to produce more possible state as to reach the goal state. The idea came from the fact that isolated pegs may sometimes be the dead end of the game (of course if that isolated edge is not the one represented in the goal state). Then, we negate the value before storing it in the heuristics value because the higher the heuristic value, the closer to the solution the state will be, thus for it to properly be reflected in the greedy and the A-star comparator, it needs to be negated (we pick the lowest value with a min priority queue).

* + **Game board (4)**

The game board 4 of the pegs game is represented as stated earlier in the form of a matrix, and is printed throughout the search program if you need to investigate it.

* + **Other game board**

The other game board are also represented. You only need to to run it with the adequate spec files and the adequate initial and goal state representations files.

* + **Heuristics discussion (a)**

This heuristic is neither admissible nor consistent. It does not under represent the distance to the goal, and it does not maintain consistent values for all distances to the goal. This is because the heuristic does nothing to evaluate the difference from the goal state. It is a “local” heuristic. It only looks at the current state

* + **Heuristics discussion (b)**

Running two different heuristics for GBFS:

* pegCountHeuristic: The number of nodes expanded is 1678and the # of loops detected is 2587 and the pathcost to solution 31
* totalPossibleMovesHeuristic: The number of nodes expanded is 11000and the # of loops detected is 6242
* Do you want to continue the search? if yes enter 1 otherwise 0
* // I actually stopped the search because it was taking so long

There is no gain. This heuristic is pretty bad.

* **Missionaries and Cannibals**
  + **State Representation:** The state is represented as one vector. All values in the vector represent the number of objects on the initial (non-goal) side of the river. That being said, the first position in the vector represents the number of Missionaries, and the second represents the number of Cannibals. The third is either a 1 or a 0, and just represents the presence of the boat on the initial side (1) or the goal side (0). For example, the standard state for this problem would look like this : <3, 3, 1>
  + **Goal Test:** The goal test compares the current state to the goal state input using a standard boolean equals operator for each element.

For instance [3==0 && 3 ==0 && 1 ==0 ]

* + **Successor Function:** The successor function for the two person boat is coded in general for many boat capacities. There is an algorithm that first generates all possible travel options, then stores them in deque structure. Then the successor tests these options against the current node (add if boat is on goal side, subtract if boat is not) by seeing if any of them are invalid moves according to the laws of the game. The ones that are valid are returned as a vector of nodes.
  + **Path cost Function:** The path cost function is strictly hardcoded as a unit cost, but it does allow for change to be made for the different kinds of moves:
    - 1) No missionaries on the Goal Side
    - 2) No missionaries on the initial side
    - 3) If #C do not outnumber #M on the Goal side AND the initial side
  + **Heuristics:** One is implemented, and will be discussed later on in the report.
  + **Strategies Tried:** DF, BF, GBeF, A\* all work, and will find a solution
  + **Heuristics Discussion(a):**
    - This will remove the constraint of the cannibals not outnumbering the missionaries. So the Heuristic will just compute how many trips it will take to get everyone from the initial to the goal side just taking into regards the boat capacity, and where the boat is. The heuristic also starts by subtracting the goal state from the current state, thus increasing accuracy for all cases.
    - Admissible:   
       Definition - h(n) < h\*(n), i.e. h(n) needs to be less than the actual step cost to the goal.  
      Our heuristic is admissible due to the part of the problem that is disregarded. It just regards the number of people on the initial side, and not the ratio of missionaries to cannibals.
    - Consistent:   
       Definition - h(n) < stepcost(n’) + h(n’)  
      Our heuristic is somewhat consistent because it takes into account the number of people on the wrong side, the boat position, and the boat capacity. It is only not consistent toward the middle of the path.
    - For an example, consider the following states evaluated on the heuristic.  
       State: < 3,3,1,> Heuristic value: 9  
       State: < 2,2,0,> Heuristic value: 8  
       State: < 3,2,1,> Heuristic value: 7  
       State: < 3,0,0,> Heuristic value: 6  
       State: < 3,1,1,> Heuristic value: 5  
       State: < 1,1,0,> Heuristic value: 4  
       State: < 2,2,1,> Heuristic value: 5  
       State: < 0,2,0,> Heuristic value: 4  
       State: < 0,3,1,> Heuristic value: 3  
       State: < 0,1,0,> Heuristic value: 2  
       State: < 0,2,1,> Heuristic value: 1  
       State: < 0,0,0,> Heuristic value: 0
  + **Heuristics Discussion(b):**
    - The heuristic gives the most gains in problems with larger Missionary and Cannibal numbers. Consider the following two cases:
      * Case 1: MCPSpec5GBFS InitialStateMCP664 GoalStateMCP000
      * Using: **numberOnStartSideHeuristic**

The number of nodes expanded is 15 and the # of loops detected is 13 and the pathcost to solution 9

the solution is:

State: < 6,6,1,> Heuristic value: 12 | Path Cost: 0

State: < 4,4,0,> Heuristic value: 8 | Path Cost: 1

State: < 5,5,1,> Heuristic value: 10 | Path Cost: 2

State: < 3,3,0,> Heuristic value: 6 | Path Cost: 3

State: < 4,4,1,> Heuristic value: 8 | Path Cost: 4

State: < 0,4,0,> Heuristic value: 4 | Path Cost: 5

State: < 0,5,1,> Heuristic value: 5 | Path Cost: 6

State: < 0,1,0,> Heuristic value: 1 | Path Cost: 7

State: < 0,2,1,> Heuristic value: 2 | Path Cost: 8

State: < 0,0,0,> Heuristic value: 0 | Path Cost: 9

* + - * Case 2: MCPSpec5GBFS InitialStateMCP664 GoalStateMCP000
      * Using: **numberOnStartSideWithBoatHeuristic**

The number of nodes expanded is 12 and the # of loops detected is 11 and the pathcost to solution 9

the solution is:

State: < 6,6,1,> Heuristic value: 7 | Path Cost: 0

State: < 4,4,0,> Heuristic value: 6 | Path Cost: 1

State: < 5,5,1,> Heuristic value: 5 | Path Cost: 2

State: < 3,3,0,> Heuristic value: 4 | Path Cost: 3

State: < 4,4,1,> Heuristic value: 5 | Path Cost: 4

State: < 3,2,0,> Heuristic value: 4 | Path Cost: 5

State: < 3,3,1,> Heuristic value: 3 | Path Cost: 6

State: < 0,3,0,> Heuristic value: 2 | Path Cost: 7

State: < 0,4,1,> Heuristic value: 1 | Path Cost: 8

State: < 0,0,0,> Heuristic value: 0 | Path Cost: 9

* + - * Nodes expanded and loops detected were both less!
* **Extra Credit Search Strategy**
  + **IDS:** in searchengine.h file and its class
  + **Depth Limited search:** in searchengine.h file and its class

* **Extra Credit Pegs Puzzle**
  + **State representation**

**We represent the triangle puzzle the same way as the other board shapes i.e. a 2D vector. However, we rotate the shape of by 45 degrees to the left as to represent it as a square. Here is a matrix representation of the goal state:**

**1 0 0 0 0**

**0 0 0 0 X**

**0 0 0 X X**

**0 0 X X X**

**0 X X X X**

**Technically speaking, or class&objects speaking, we have created a class called PegsTriangleNode that is found in the file PegsLogicNode.h that extend the PegsNode (which itself extend the abstract class Node by the way) and it override only two methods which are successor function and step cost**

* + **Goal state**

**As we did for all puzzles, we get the goal state from the text that is specified in the 3rd argument when executing the program. And that goal state is then stored in a vector that is present in Node.h and you can test the whether a triangular Pegs shape node has reached the goal state by using goalTest method that is present in the NodePegs class (because the triangular shapes will have the same way of comparing the goalstate with its own state to see whether it has reached the goal) in other words, we did NOT override that method**

* + **Successor function**

**However, for the successor function, we had to override that method so to have a specific method for generating all next nodes because the triangular shapes have more possible moves (in a square transitioned way) than the other shapes(for more details about the moves, please refer to the PegsTriangleNode class).**

* + **Path cost function**

**The stepCost function has also been overriding as there is more moves (please refer to the successor function) than there is in the other pegs shapes.**

* + **Heuristic**

**We have kept the same logic for the heuristic(for more details please refer to the heuristics of the Pegs game in general) however, we had to override the heuristic as there is two more possible moves when we translate a triangular shape in a 2D vector by rotating.**

* **Extra Credit Missionaries and Cannibals**
  + The following list the command arguments that you will need to input after the executable (if running on command line). Otherwise, if you want to pick a specific text file in our folder to use, type it in verbatim, BUT *leave off the “.txt”*
  + If you want to compile:
    - g++ -std=c++11 -o A2 main.cpp
  + If you want to run:
    - ./A2 MCPSpec1DFS InitialStateMCP222 GoalStateMCP000
  + Flexible representation for 2-, 3-, 4-person boat?
    - Yes. Our program also allows for other inputs besides just couples
  + Successor function for 3 person boat?
    - Same as 2 person boat (explanation provided above)
  + Input provided for 2-person boat, 2 couples?
    - MCPSpec1DFS **InitialStateMCP222** GoalStateMCP000
  + Input provided for 2-person boat, 4 couples? (no solution)
    - MCPSpec1DFS **InitialStateMCP442** GoalStateMCP000
  + Input provided for 3-person boat 2 couples?
    - MCPSpec1DFS **InitialStateMCP223** GoalStateMCP000
  + Input provided for 3-person boat 3 couples?
    - MCPSpec1DFS **InitialStateMCP333** GoalStateMCP000
  + Input provided for 3-person boat 4 couples?
    - MCPSpec1DFS **InitialStateMCP443** GoalStateMCP000
  + Input provided for 3-person boat 5 couples?
    - MCPSpec1DFS **InitialStateMCP553** GoalStateMCP000
  + Input provided for 3-person boat 6 couples?
    - MCPSpec1DFS **InitialStateMCP663** GoalStateMCP000
  + Successor function for 4 person boat?
    - Yes. Successor function is the same for an n-person boat.
  + Input provided for 4-person boat different numbers of couples?  
    (2 is trivial; show 3, 4, 5, 6)
    - MCPSpec1DFS **InitialStateMCP334** GoalStateMCP000
    - MCPSpec1DFS **InitialStateMCP444** GoalStateMCP000
    - MCPSpec1DFS **InitialStateMCP554** GoalStateMCP000
    - MCPSpec1DFS **InitialStateMCP664** GoalStateMCP000
  + Question: With a 4-person boat, **why** can any number of couples cross?
    - Because it allows for even numbers to cross to the other side while an even number row the boat. Thus there never has to be anything besides equal numbers of Missionaries and Cannibals, on both sides of the river.