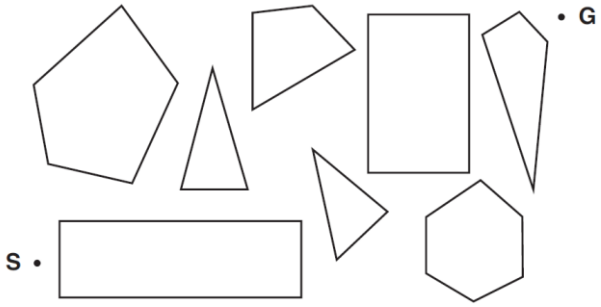
CSC 421 Artificial Intelligence: Assignment 1

Q1. (5 pts)

a. (2 pts) Consider the ***n*-rook problem**: Place *n* rooks on an n by n chess board so that no rook attacks another. Consider a search formulation (similar to the 8-queens example) where states are all the possible arrangements of *m* rooks ($0 \leq m \leq n$), one per column in the leftmost *m* columns, with no rook attacking another. The actions are adding a rook to any square in the leftmost empty column such that it is not attacked by any other rook. Let *S* be the set of states for the above formulation. **What is the size of S?**

b. (3 pts) Consider the ***n*-queen problem**: Place *n* queens on an n by n chess board so that no queen attacks another. Consider a search formulation (similar to the above) where states are all the possible arrangements of *m* queens ($0 \leq m \leq n$), one per column in the leftmost *m* columns, with no queen attacking another. The actions are adding a queen to any square in the leftmost empty column such that it is not attacked by any other queen. Let *S* be the set of states for the above formulation. **Show that the size of S is at least $\sqrt[3]{n!}$**

Q2. (5 pts) Consider the problem of finding the shortest path between two points on a plane that has convex polygonal obstacles as shown in figure on the right. S and G are the start and goal states. This is an idealization of the problem that a robot has to solve to navigate in a crowded environment.



a. (1 pts) Suppose the state space consists of all positions (x, y) in the plane. **How many states are there? How many paths are there to the goal?**

b. (4 pts) Explain briefly why the shortest path from one polygon vertex to any other in the scene must consist of straight-line segments joining some of the vertices of the polygons. Define a good state space now. **How large is this state space?**

Q3. (5 pts) Which of the following are true and which are false? Explain your answers.

a. (1 pts) Depth-first search always expands at least as many nodes as A∗ search with an admissible heuristic.

b. (1 pts) h(n) = 0 is an admissible heuristic for the 8-puzzle.

c. (1 pts) Breadth-first search is complete even if zero step costs are allowed.

d. (2 pts) Assume that a rook can move on a chessboard any number of squares in a straight line, vertically or horizontally, but cannot jump over other pieces. Manhattan distance is an admissible heuristic for the problem of moving the rook from square A to square B in the smallest number of moves.

Q4. (5 pts) Use the straight-line heuristic function (see slides) to solve the Romania route problem (starting from **Timisoara** going to **Bucharest**) efficiently using A* TreeSearch and A* GraphSearch. Report statistics. Draw (or print out) the final search tree, and add at each node its path cost value (*g*), the heuristic function value (*h*), their sum (f=g+h), and the order of expansion, e.g. the root will be 0, then the next node expanded will be 1, etc.
You can solve this by hand on paper, or create a PrintTree method (extending the code provided) which takes the list of all created (search-tree) nodes and prints them in a nested form, e.g.

```
Timisoara(g=0.0, h=329.0, f=329.0) order=0
  Arad(g=118.0, h=366.0, f=484.0) order=3
    …
  Lugoj(g=111.0, h=244.0, f=355.0) order=1
    …
```

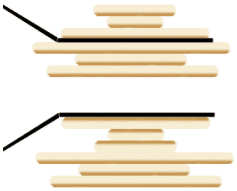
Q5. (5 pts) Complete the accompanying base code. It has implementations for several search strategies, but not all, e.g. iterative deepening is not implemented (you should implement it). Look for "TODO" comments.

Q6. (15 pts) Consider the following problems. Model them as search problems, implement them using the accompanying code and find a solution using all the search algorithms that work on the particular problem. Report solution cost and number of expansions for each search algorithm that is able to (quickly) terminate on a problem. For each problem, find a heuristic function that improves the performance of A* compared to UCS. See the implemented NPuzzle problem in the accompanying code for an example.

a. (5 pts) **Wolf-goat-cabbage problem**
You are on the bank of a river with a boat, a cabbage, a goat, and a wolf. Your task is to get everything to the other side. Restrictions are as follows.
1. only you can handle the boat
2. when you're in the boat, there is only space for one more item
3. you can't leave the goat alone with the wolf, nor with the cabbage (or something will be eaten).

b. (5 pts) **Missionaries and cannibals problem**
Three missionaries and three cannibals seek to cross a river, say from the left bank to the right bank. A boat that may be navigated by any combination of one or two people is available on their side of the river. If cannibals outnumber the missionaries on either side of the river at any time, the cannibals will indulge in their anthropophagic tendencies and do away with the missionaries. Find a sequence of boat trips that will permit all the missionaries and cannibals to cross the river safely.

c. (5 pts) **Pancake Sorting Problem**
Given a stack of pancakes of various sizes, can you sort them into a stack of decreasing sizes, largest on bottom to smallest on top? You have a spatula with which you can flip the top i pancakes. This is shown below for i = 3; on the top the spatula grabs the first three pancakes; on the bottom we see them flipped:



How many flips will it take to get the whole stack sorted? This is an interesting problem that Bill Gates has written about. A reasonable heuristic for this problem is the *gap heuristic*: if we look at neighboring pancakes, if, say, the 2nd smallest is next to the 3rd smallest, that's good; they should stay next to each other. But if the 2nd smallest is next to the 4th smallest, that's bad: we will require at least one move to separate them and insert the 3rd smallest between them. The gap heuristic counts the number of neighbors that have a gap like this. In our specification of the problem, pancakes are ranked by size: the smallest is 0, the 2nd smallest 1, and so on, and the representation of a state is a tuple of these rankings, from the top to the bottom pancake. Thus the goal state is always (0, 1, ..., *n*-1) and the initial (top) state in the diagram above is (1, 0, 3, 5, 2, 4).

What to submit.
One PDF file with the solutions to Q1-Q5. In the same file put: the code you added for Q5 (only your part, not the rest of code which is provided) and the output printout for Q6, a, b, c.
One zip file with the complete code for all programming parts.
So, in total you need to submit two files.

If you do the assignment in a **group**, write the **names** and **ids** of **group members** in the **submission textbox** as well as in the **PDF file** submitted. If the names of group members have not been written in those two places, we will not consider it a group submission but only individual.