# CSCE 420

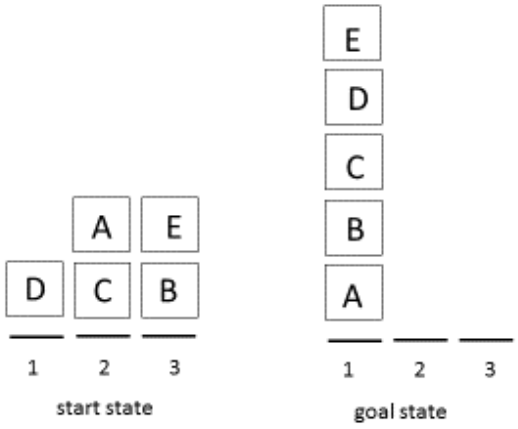
**Programing Assignment #1**

**due: Tuesday, Sept 25, 9:35am (start of class)**

Objective

The overall goal of this assignment is to implement A* search and use it to solve a classic AI search problem: the "Blocksworld". This problem involves stacking blocks into a target configuration. You are to write a program to solve random instances of this problem using your own implementation of A* search. The focus of the project, however, is to use your intuition to develop a *heuristic* that will make solving various instances of this problem more efficient (solve them in fewer iterations).

Follow the description of A* in the textbook, i.e. GraphSearch with the frontier represented as a PriorityQueue. (Your code for A* should be written from scratch by you; however, you may use an existing implementation of PriorityQueue from a library, such as in the STL.) Specifically, you will keep the queue sorted based on f(n)=g(n)+h(n). g(n) is just the path length from the root to the current node. **The main focus of this project will be on developing a heuristic function, h(n),** for this domain that estimates the distance to the goal (number of moves to solve the problem) and testing how it affects the efficiency of the search - something more sophisticated than just "number of blocks out of place".




The operator in this problem can pick up only the top block on each stack and move it to the top of any other stack. Each action has unit-cost. The objective is to find a sequence of actions that will transform the initial state into the goal state (preferably with the fewest moves, hence the shortest path to the goal.) Your program should be able to handle problems with different numbers of stacks and blocks.

This problem becomes harder as the number of blocks scales up, and cannot be solved effectively with BFS (though you can try BFS for small numbers of blocks; setting h(n)=0 in A* will emulate BFS). You will need A* search coupled with a heuristic to find solutions efficiently. The default heuristic (call it $h_0$) can be taken to be the "number of blocks out of place". Your goal is to define a better heuristic that will enable your algorithm to find solutions faster (with fewer goal tests), and to be able to solve larger problems (with more blocks). An interesting question to consider is whether having more stacks should make the average problem easier or harder to solve. What do you think?

Implementation

We will use a input format for each problem, defined as follows. Line 1 indicates the number of stacks and number of blocks (let them be N and B). Then the next N lines give the configuration of the starting state by listing the stacks on their side. Blocks (assumed to be single characters, A-Z) are listed left-to-right in order from bottom to top, with no spaces. Finally, the last N lines give the goal configuration. Here is a representation of the problem above:

```
--- blocks1.txt ---
3 5
D
CA
BE
ABCDE


------------------
```

Note the the two empty columns in the goal. The goal will not always have the blocks in order in a single stack - any goal configuration is possible.

You can assume that input files will always comply with this specification; you don't have to put a lot of error-checking in your function that reads these input files. Don't worry about checking for extra spaces or characters, or things like incorrect number of lines or duplicate block characters, etc.

Your program should take a filename as a command line argument. See an example transcript below for the problem above. *You don't have to exactly reproduce my output.*

Your program must be written in C++ using the STL. In your source code, we should be able to see a function that looks like GraphSearch, and another function that looks like "int heuristic(State)" or "State::heuristic()" or "heuristic(state,goal)" that calculates a heuristic score for a state (estimated distance to the goal).

It is very useful to put a limit on the maximum number of iterations, to control the search in case it takes too long to solve a given problem. In my implementation, I have set the cutoff at 100,000 iterations. The terminates the search for a given problem after about 5 minutes on my machine. You might have to adjust your cutoff depending on the speed (or memory) of the machine you are using.

Challenge Problem Set

A "challenge set" of 54 defined problems will be provided on which to test your program. Your objective is to develop a heuristic that will solve as many of these problems as possible. Some of them are easier to solve than others (for example, those with a solution path of 5-10, which can be found using BFS). But other problems in this challenge set are much harder, with solution paths of length >15 and will require a more sophisticated heuristic.

Tasks to do:

1. write a class for storing a problem **State**, and a **Node** in the search space
2. write code to read-in a problem file
3. implement a **goal_test** function (which checks if one state matches another)
4. implement a **successor**() function, which generates neighbor states by legal moves
5. implement **GraphSearch** (Figure 3.7 in textbook); keep it sorted by f(n)=g(n)+h(n)
6. implement a function that prints out the **solution path** once a goal node is found, which is a sequence of states on the path from the root to a goal node (if your Nodes store pointers to the parents from which they were generated, this could be a simple recursive function)
7. develop the best **heuristic function h(n)** you can, that enables your program to solve the most problems in the least time
8. at the end of a search, print out statistics, like solution path length, total # of goal tests, and maximum queue size

What to Turn in

- You will submit your code for testing using **eCampus** (https://ecampus.tamu.edu/)
- You should include a Word document with instructions on how to compile and run your program
- Include a written **description** of how your heuristic works, or what it is based on, or the logic behind it.
- include a transcript that shows some **example program traces** for your A* search.
- You should also include a **table of results** for all the problems in the challenge set, with things such as number of iterations (goal tests), maximum frontier size, mean solution path length, etc. If your program *fails* to solve some problems within the time-limit, be sure to indicate those as well.

```
> blocksearch blocks1.txt
3 stacks, 5 blocks
init state:
D
CA
BE
#####
goal state:
ABCDE


#####
iter 1, depth=0, f(n)=11.0, frontier=0
D
CA
BE
iter 2, depth=1, f(n)=11.0, frontier=5
DE
CA
B
iter 3, depth=1, f(n)=12.0, frontier=9
D
C
BEA
iter 4, depth=2, f(n)=12.0, frontier=12
DE
C
BA
...
iter 61, depth=9, f(n)=13.0, frontier=119
ABCE
D

iter 62, depth=9, f(n)=13.0, frontier=120
ABC
D
E
iter 63, depth=10, f(n)=12.0, frontier=123
ABCD

E
solution!
init
D
CA
BE
step 1, h(n)=12.000000

CA
BED
step 2, h(n)=10.000000
A
```

```
C
BED
step 3, h(n)=10.000000
AC

BED
step 4, h(n)=9.000000
AC
D
BE
step 5, h(n)=9.000000
AC
DE
B
step 6, h(n)=9.000000
A
DEC
B
step 7, h(n)=7.000000
AB
DEC

step 8, h(n)=5.000000
ABC
DE

step 9, h(n)=4.000000
ABC
D
E
step 10, h(n)=2.000000
ABCD

E
step 11, h(n)=0.000000
ABCDE


iterations=63, path_length=11, goal_tests=186, max_frontier_size=124
```