Features

- [x] Input files are given in the command line, through a path to a designated directory. Same for the output directory and the number of solutions. The command line also specifies the tiomeout.

```
python m.py Inputs Outputs 10 60
```

- [x] Parsing the input file.
- [x] Generating successors.
- [x] Calculating the cost for a move as `1 + (N-K)/N`, where `N` is the number of characters of the type we are removing and `K` is the number of characters we are removing through this move.
- [x] Testing to check if the goal state was reached.
- [x] 4 heuristics:
  - [x] **simple heuristc**:

    ```
    h = 1 if state is non terminal
    h = 0 if state is terminal
    ```

    Although simple, this heuristic is valid because the cost of a move ranges between `[1,2)`. Beacause of this, the cost of the solution cannot be less than `1`, so the heuristic is valid.
  - [x] **first heuristic**: this heuristic takes advantage of the fact that the cost of every move ranges between [1,2)

    ```
    h = number of characters in the current state
    ```

    In the best case scenario we will be able to move the characters in such a way that we eliminate all the characters of one type in one move. This heuristic assumes that we will be able to do that for every state, and calculates the cost of this best case scenario.
  - [x] **second heuristic**: this heuristic works because it estimates haow fragmented the matrix is.

    ```
    h = 1 - K/N, where N is the number of characters of the type we are removing and K is the number of characters we are removing through this move.


    Suppose we have a matrix of n rows and m columns.
    The matrix contains c characters.
    The number of characters for each character is p_i, where i is the index of the character.
    Obviously, p_1+p_2 +...+ p_c = n*m
    Each character has f_i fragments, where i is the index of the character.
    Each fragment has size s_i_j, where i is the index od the character and j is the index of the fragment.
    Following this heuristing, the estimated cost for this matrix is:
    - 1-s_1_1/p_1 + 1-s_1_2/p_1 + ... + 1-s_1_(f_1)/p_1 for character 1. We know that s_1_1 + s_1_2 + ... + s_1_(f_1) = p_1
    ```

```
  so the final cost is f_1 -1
- the final cost for character2 is f_2 -1
- same for the rest of them
So, the final estimated cost is f_1 + f_2 + ... + f_c - c.
```

- [x] **invalid heuristic**: this heuristic assumes that we remove each zone on its won, without moving the characters

```
h = 1 + (N-K)/N,  where N is the number of characters of the type we are removing and K is
the number of characters we are removing through this move.
```

Lets' prove that it's invalid:

```
                             optimal solution
aaa                          aaa                ###                ###
bbb                          bbb  => cost 1 =>  aaa  => cost 1 =>  ###
aaa                          aaa                aaa                ###
estimated h:                 optimal solution cost:
2-1/2 + 1 + 2-1/2 = 4            2

4 > 2
```

- [x] input files:
  - [x] a file with no solutions:
  - [x] a file where the initial state is also final
  - [x] a file with small solution lengths
  - [x] a file that timeouts an algorithm: `input4.in` timeouts on Iterative deepening A* when timeOut is set to 6s, but gives a solution for every other algorithm.
  - [x] at least one of these files should result in a subobtimal result for the invalid heuristic

optimized A* with second heuristic	optimized A* with invalid heuristic
5	12.83

- [x] for each solution output:
  - [ ] id of every node in solution
  - [x] solution length
  - [x] cost
  - [x] time needed to find the solution
  - [x] maximum number of nodes generated at any point in time
  - [x] total number of nodes generated
- [x] validations:
  - [x] checking if input data is correct

    - [x] checking if a state could result in a goal state or not
- [x] table

Table for input 4

algorithm	solution length	cost	time	maximum number of nodes	total number of nodes
uniform cost search	7	6.44	0.84	715	874
A* - obvious heuristic	7	6.44	0.85	2028	874
A* - first heuristic	11	12.83	0.57	2028	608
A* - second heuristic	7	6.33	0.45	2028	443
A* - invalid heuristic	10	11.33	0.27	2028	261
optimized A* - obvious heuristic	6	5.00	1.90	1233	1592
optimized A* - first heuristic	7	6.56	0.41	367	402
optimized A* - second heuristic	6	5.00	0.78	588	741
optimized A* - invalid heuristic	11	12.83	0.17	142	153
iterative deepening A* - obvious heuristic	6	5.00	6.86	850	6004
iterative deepening A* - first heuristic	6	5.00	8.52	4738	6004
iterative deepening A* - second heuristic	6	5.00	6.81	4738	6004
iterative deepening A* - invalid heuristic	6	5.00	6.71	4738	6004

Table for input 3

algorithm	solution length	cost	time	maximum number of nodes	total number of nodes
uniform cost search	5	4.75	0.09	159	210
A* - obvious heuristic	5	4.75	0.11	294	210
A* - first heuristic	7	6.56	0.41	367	402
A* - second heuristic	5	4.75	0.03	294	66
A* - invalid heuristic	6	6.08	0.03	294	75

algorithm	solution length	cost	time	maximum number of nodes	total number of nodes
optimized A* - obvious heuristic	5	4.50	0.10	108	178
optimized A* - first heuristic	5	4.50	0.03	65	89
optimized A* - second heuristic	5	4.50	0.16	99	160
optimized A* - invalid heuristic	5	4.50	0.04	52	64
iterative deepening A* - obvious heuristic	5	4.50	0.33	198	745
iterative deepening A* - first heuristic	5	4.50	0.35	198	745
iterative deepening A* - second heuristic	5	4.50	0.38	198	745
iterative deepening A* - invalid heuristic	5	4.50	0.33	198	745

Conclusions

- A good heuristic can improve the cost of the first solution by 50% (A* invalid heuristic vs A* second heuristic).
- IDA* seems to be the most consistent at finding the minimum possible cost, although it takes the longest time to do so and generates the most nodes. For the other algorithms, they rarely find the best solution first.
- When comparing A* with optimized A*, we can see consistent better results for optimized A* whilst the running times for both of them are comparable.
- Because the simple heuristic is valid and always has value `1`, it leaves the final say in which node gets expanded to `g`, aka the cost from start to current node. Because the cost of a move tends to favor moves that remove as many characters of one type as possible, algorithms using the simple heuristic still get pretty good results.