

The 8-Puzzle: Search Algorithms

Maximum number of members per group: 3 students

Deadline for submission: **September 8, 2025**

Instructions

- Your task is to write a C++ program that will solve the 8-puzzle problem using a selection of search algorithms, and their variants.
- The successors of a state are to be generated in a **FIXED** order, namely move the blank tile: **Up**, **Right**, **Down**, then **Left**. For simplicity, make node insertions into the Q, following the same order.
- An **AnimateSolution()** function has been provided that you can use to animate the sequence of moves (i.e. path) calculated by the algorithms. A **start-up program** (compiles with **g++ 15.1**) with a graphics library and routines for running multiple experiments and for generating tabulated results are available for downloading from stream.
- It is up to you to write any functions, classes or data structures that you may require. However, for each of the algorithm, there is a **specific STL data structure that is required**. You can use **cout** statements to trace the algorithms' execution.
- For each implementation of the algorithms below, include codes that will capture the following information during the algorithm's execution.
 - a. Max. Q length – e.g. 26
 - b. Path length - the number of moves to solve the puzzle, e.g. 30
 - c. Number of state expansions – e.g. 157
 - d. Actual running time in seconds (use the `clock()` function as shown in the start-up codes)
- Write your algorithm implementations inside the skeleton functions provided for the required algorithms. Do not change the names and input parameters of these skeleton functions as the batch files would refer to them. Each algorithm implementation should return the sequence of moves as a **string**. Moreover, make sure that your program runs with the supplied routines for executing multiple experiments (i.e. **batch_run**), and for generating the tabulated experiment results. Your assignments will be marked using them.

i.e.

```
string uc_explst(string const initialState, string const goalState, int &pathLength,
```

```
    int &numOfStateExpansions, int &maxQLength, float &actualRunningTime,  
    int &numOfDeletionsFromMiddleOfHeap, int &numOfLocalLoopsAvoided,  
    int &numOfAttemptedNodeReExpansions)
```

```
string aStar_ExpandedList (string const initialState, string const goalState, int &pathLength,  
    int &numOfStateExpansions, int &maxQLength, float &actualRunningTime,  
    int &numOfDeletionsFromMiddleOfHeap, int &numOfLocalLoopsAvoided,  
    int &numOfAttemptedNodeReExpansions, heuristicFunction heuristic )
```

Note that the function uses pass by reference to copy the statistical results back to the calling function

Part 1: Uniform Cost Search with the Strict Expanded List

- Use the following search node pushing sequence (for a Heap data structure): Up, Right, Down, Left
- Implement the Q container using the min **heap** data structure implementation - available in the C++ Standard Template Library (STL): use `make_heap()`, `push_heap()`, `pop_heap()`, etc.
- Define a custom min **heap** comparator that implements h-value ordering.

Part 2: A* Search Using a Strict Expanded List and Tie-Breaking in Favour of Larger g-Values

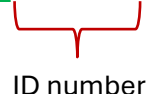
- Use the following search node pushing sequence (for a Heap data structure): Up, Right, Down, Left
- Implement the Q container using the min **heap** data structure implementation - available in the C++ Standard Template Library (STL): use `make_heap()`, `push_heap()`, `pop_heap()`, etc.
- Define a custom min **heap** comparator that implements f-cost ordering and satisfies the tie-breaking criterion.
 - a) Using the Misplaced Tiles heuristic
 - b) Using the Sum of Manhattan Distance heuristic

Part 3: Experiments and Documentation

Test your implementation of the different algorithms by performing experiments using the **5** given **(start, goal) state combinations** below. Run your program until it either returns a solution, the Q becomes empty (no solution), the computer runs out of memory, or until the program crashes. Run the program in **batch_run all** mode to run all the experiments and collect the results easily.

Tabulate the experiment results in an Excel worksheet by converting the output of the batch file into a worksheet. Ensure that the format of your tabulation matches the provided template (see [results_template.xlsx](#)). Name your Excel file using the following format: **results_ID.xlsx**

Example: (e.g., **results_20298765.xlsx**).


ID number

In addition, assign the name "**results**" to the sheet containing the experiment results. For a group submission, use one of the group member's ID numbers, but make sure to include the names and IDs of all members in the [checklist Excel file](#).

If there is no solution found for a given (start, goal states), simply leave that section blank in the table, or write 0 in each of the required statistical measure (e.g. path length, no. of state expansions, max q length, running time, etc.).

Specify under the "**Comments**" section of the tabulation of results if any of the following was observed for a given (start, goal state) combination:

- the program ran out of memory
- program crashed without any warning
- the Q turned empty; thus, allowing the program to close properly

(Start, Goal) State Combinations

Note: 0 - blank space

GOAL STATE: ((1 2 3)
(4 5 6)
(7 8 0))

Run the different algorithms on the following **START STATES:**

1. 120483765
2. 208135467
3. 704851632
4. 536407182
5. 638541720

Hints:

You can step through the search by including a **getch()** function (made available via the graphics engine provided in the start-up codes) inside your main loop to pause the program until the user presses any key.

Example Sequence:




Sequence of states and operations.

You may choose to represent states in an array, of size 9. The moves must be represented using the 'u', 'd', 'l', 'r' characters.

In notation, the sequence *s* to get to the goal from the initial state could be represented as:
s = {d,r,u,u,l,d} You may find it helpful to *cout* something similar to help debug your program.

Criteria for Marking:

- Make sure that your program compiles using **gcc 15.1** (or later), or **clang 15.0** (or later), before handing it in.
- Make sure that you submit a tabulation of all the experiment results, following the [results_template.xlsx](#) format that comes with the start-up codes package. Enter the required information in the designated cells without altering their position. This will be used to accurately analyse your implementation of the algorithms and mark your assignment. **You will lose 50% of your grade if you fail to perform the required experiments and submit this file.**
- Submit the accomplished **checklist** as part of your documentation. Please download the [checklist.xlsx](#) Excel file from our Stream site, fill-up the worksheet and **rename it** by concatenating your ID number with the word 'checklist'. Name your Excel checklist file using the following format: **checklist_ID.xlsx**
Example: (e.g., **checklist_20298765.xlsx**).


ID number

159.302 Artificial Intelligence

Assignment #1

- You can work **in a group (max. 3 members)** for this assignment.
- Copied work will be given zero marks.
- Each algorithm implementation will be evaluated based on its performance, specifically considering both accuracy and speed, when applied to the given set of start and goal state combinations.

Nothing follows.