8 - PUZZLE

- The 8 puzzle is a simple game which consists of eight sliding tiles, labeled with pieces of a image, placed in a 3x3 squared board of nine cells.
- One of the cells is always empty, and any adjacent (horizontally and vertically) tile can be moved into the empty cell.
- The objective of the game is to start from an initial configuration and end up in a configuration which the tiles are placed in ascending number order.

Need of the Solver

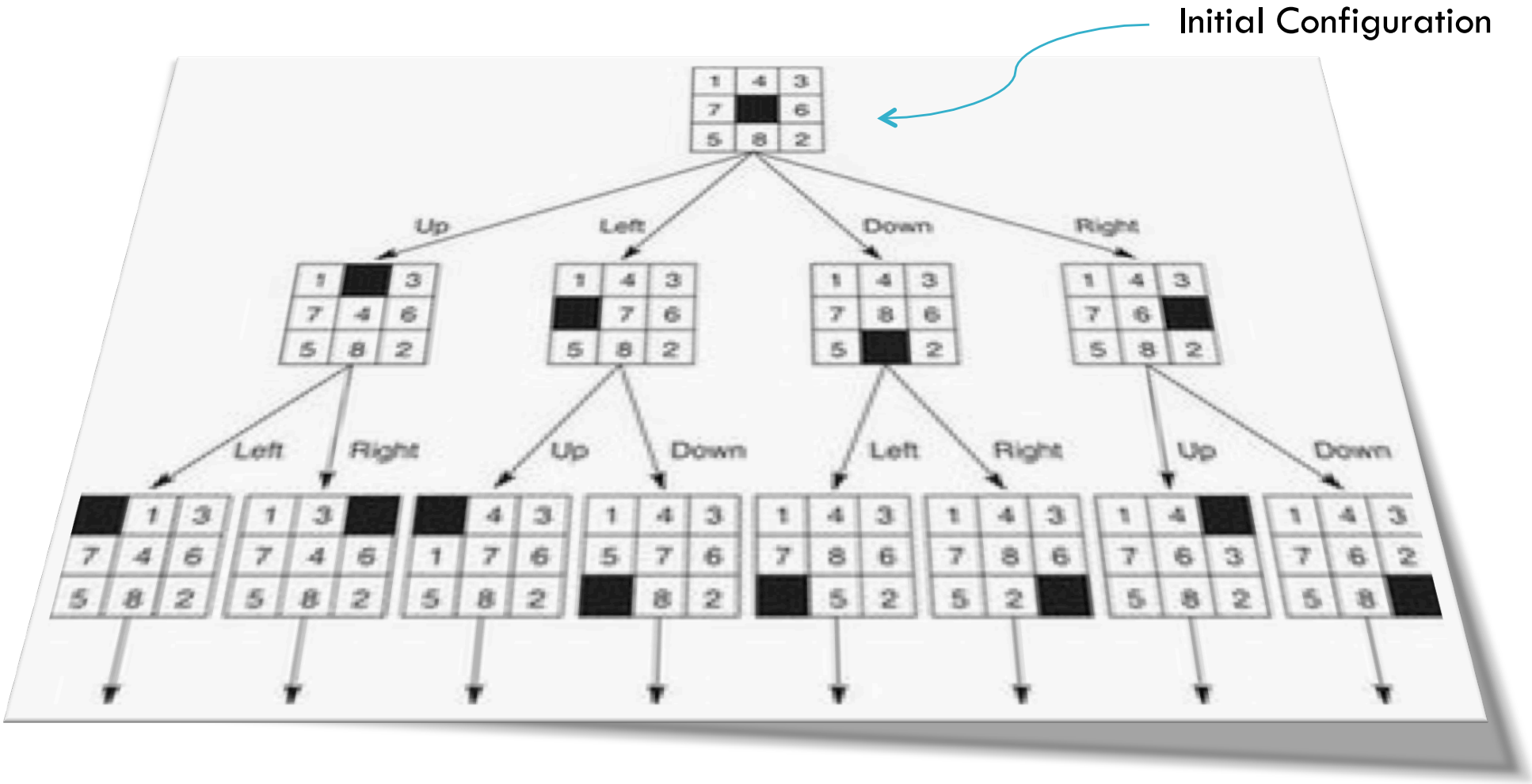
- There are many ways to solve a single puzzle, but we need the fastest way possible.
- Conventionally, the 8-puzzle can be solved storing the states as new children of a node in a tree after every possible move at that node.
- But, this makes a huge tree and includes some cycles.
- Thus, here we needs some better search “Algorithm” and “Hueristics”

Project Algorithm

- We will using **Tree Traversal** which will be done by 2 methods :
  - **A* algorithm**
  - **IDA* algorithm**
- Depth-first search is not always very successful at finding solutions that are only a few moves away. So, we are not using that.

How to use Tree Traversal

- A Tree is be constructed which stores the states of the board with its root as **initial state.**
  - The next level nodes are possible states of the board from the parent node.
- The path through which final state is found at minimum depth is the best possible solution.


Initial Configuration
Up
Left
Down
Right
Left
Right
Up
Down
Left
Right
Up
Down

A* algorithm

- A* algorithm is a search algorithm which explores a graph by expanding the most promising node chosen according to a function f(x) [distance plus cost heuristic] such that f(x) = g(x) + h(x), where
  - g(x) = the path-cost function, which is the cost from the starting node to the current node.
  - h(x) = an admissible "heuristic estimate" of the distance to the goal.

IDA* algorithm

- Many 8-puzzles cannot be solved efficiently with the A* algorithm, since it generates too many new states and consumes a lot of memory maintaining these lists. To solve such puzzles, Iterative-Deepening-A* (IDA*) can be used. Like the A* algorithm, it finds optimal solutions when paired with an admissible heuristic but is much more efficient with respect to space. IDA* is described as follows:
  - Set *threshold* equal to the heuristic evaluation of the initial state.
  - Conduct a depth-first search, pruning a branch when the cost of the latest node exceeds *threshold*. If a solution is found during the search, return it.
  - If no solution is found during the current iteration, increment *threshold* by the minimum amount it was exceeded, and go back to the previous step.

Checking Solvability

- To check the solvability of the given initial configuration, count the numbers smaller than every number occurring after it and find sum for all digits in input configuration(except 0 or blank).
- For eg 215384607
  for 2 – 1is smaller
  for 1 – no number is smaller than it
  for 5 – 3 and 4 are smaller and so on…
  Total count 1+0+2+0+3+0+0+0 = 6
- If total count is even, than the puzzle is solvable, otherwise it is not solvable.

Code Description

- The Code is divided into 3 parts :
  - MainWindow class (main file)
  - A* class implementation
  - IDA* class implementation

These files are included in MainWindow class

- Depending upon the Algorithm chosen, corresponding object is created in MainWindow class.

MainWindow Class

- A MainWindow class with
  - Public functions
    - MainWindow(QWidget *parent = 0);
    - ~MainWindow();
  - Private functions
    - Ui::MainWindow *ui;
    - void Load_images();
    - queue<string> stk;
    - a algoA;
    - ida algolda;
    - void moveHelper(int a);
  - public slots :
    - void handler();
    - void movebmp();
    - void menuA(), menuIDA();
    - void h0(), h1();
    - void combo_algo();
    - void combo_ht();
    - void myrefresh();
    - void update();
    - void nextMove();

A* Class

- A SearchTree class with
  - structTreeNode which contains links to
    - neighbors
    - vector state
    - integers cost, index, distance.
  - Stack storing states
  - Functions
    - isEmpty()
    - insert(TreeNode*)
    - void makeRoot(vector<int>)
    - build(string input, int h)
    - read(int algo, string conf)
    - print(tree_node*)
    - checkSuccess (tree_node*)
    - costCalculator(tree_node*)
    - sort()
    - makeMove(tree_node*, tree node*, char)
    - checksolvability()
    - queue<string> a;

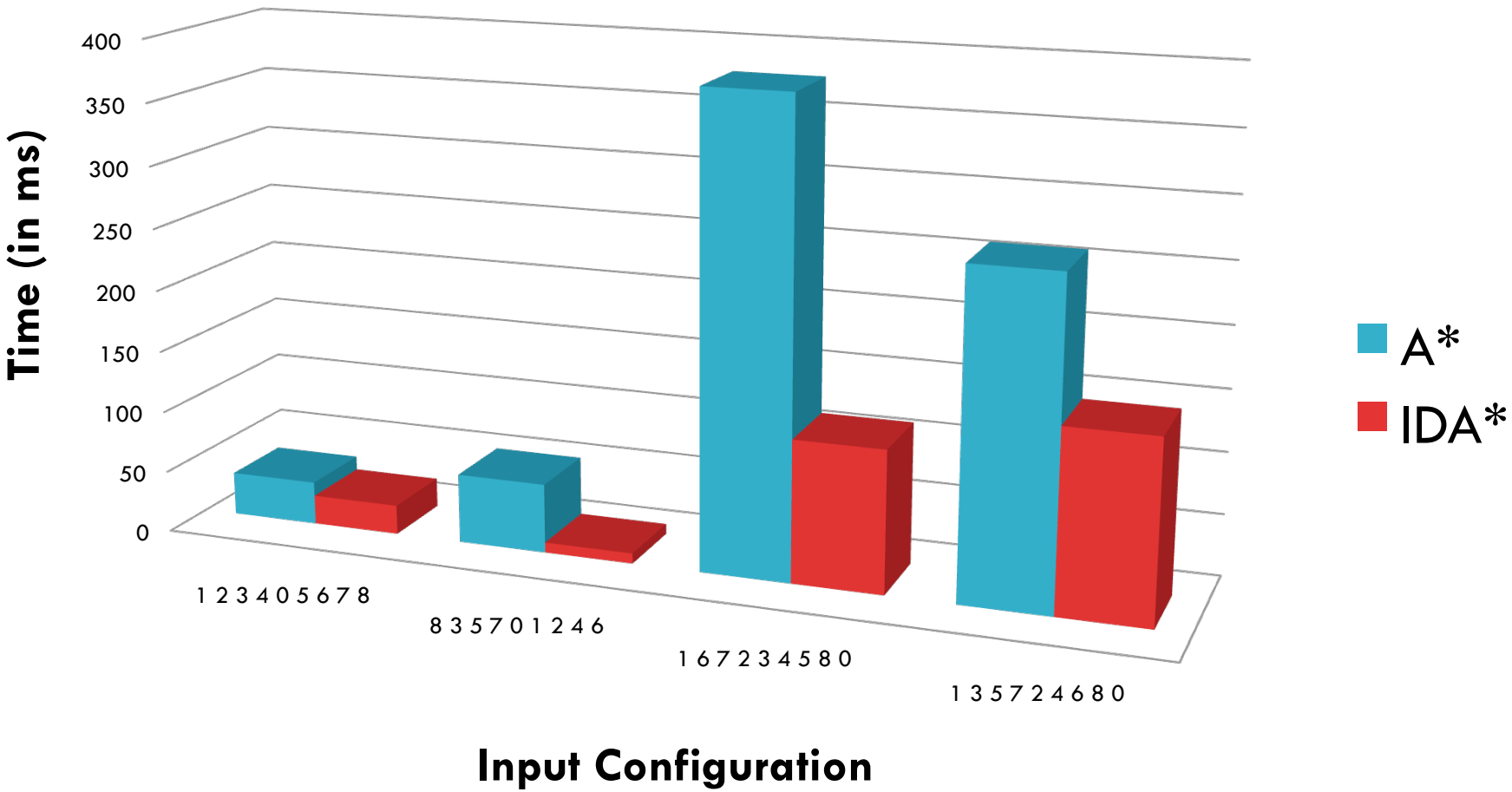
IDA* Class

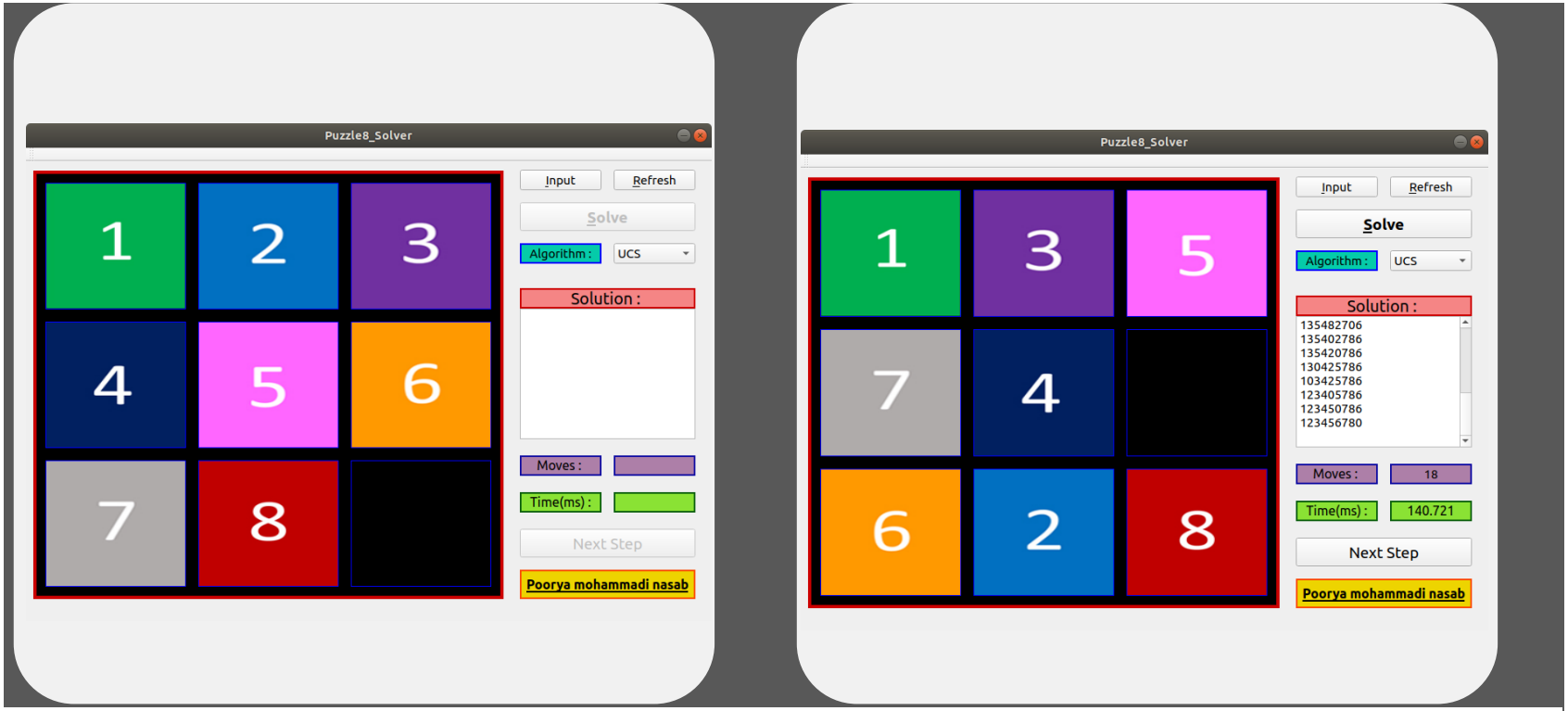
- A SearchTree class with
  - structTreeNode which contains links to
    - neighbors
    - vector state
    - integers cost, index, distance.
  - Stack storing states
  - Functions
    - isEmpty()
    - insert(TreeNode*)
    - makeRoot(vector<int>)
    - build(string input, int h)
    - read(int algo, string conf)
    - print(tree_node*)
    - checkSuccess (tree_node*)
    - costCalculator(tree_node*)
    - sort()
    - makeMove(tree_node*, tree node*, char)
    - checksolvability()
    - queue<string> a;

Statistics of Solution Time

	No of moves	Without distance		With distance	
		*A**	*IDA**	*A**	*IDA**
1 2 3 4 0 5 6 7 8	14	771.182	262.190	34.937	23.761
8 3 5 7 0 1 2 4 6	18	Cannot Solve	Cannot Solve	56.251	8.300
1 6 7 2 3 4 5 8 0	20	Cannot Solve	Cannot Solve	379.334	115.239
1 6 7 2 3 4 5 0 8	21	Cannot Solve	Cannot Solve	1590.290	400.174

Statistics with Manhattan distance





GUI Package

We have used Qt package for development of the Graphical User Interface of the program.

GUI Description

- We have used Qt package for development of the Graphical User Interface of the program.
- Initially, We Started with EzWindows Library, But later to introduce Menubar, Textbox, Pushbutton and Drop-Down List, we switched to Qt Package

Qt Package

- In Qt, We have following objects:
  - QMainWindow
  - QWidget
  - QPushButton
  - QLabel
  - QPixmap
  - QTimer
  - QString

Work Distribution

- 100050033 - 55%
  - Algorithms – A*, IDA*
  - Heuristics – Manhattan Distance, Number of misplaced tiles
  - Tried manual method of solving 8-puzzle
  - Helped in GUI
- 100050034 – 45%
  - GUI – Qt, EzWindows
  - Connecting GUI with algorithm classes
  - Helped in algorithms
  - Prepared Slides