**Programming Assignment-1**

**1. N – Slidding Puzzle problem**

**a)** using BFS : (UnInformed Search Technique)

* It traverses level order and when child generation if we encounter with goal node, it will give us goal path rather than at time of selection and expansion (reduces space complexity).

**b)** using DFS: (UnInformed Search Technique)

* It traverses level order and when child generation if we encounter with goal node, it will give us goal path rather than at time of selection and expansion.
* It is not optimal because of Loops.
* Sometimes, for small inputs also it may exceed the Maximum Recusion Depth.

**c)** using A\*: (Informed Search Technique)

* f(n)=g(n)+h(n) where g(n)=cost to reach to n ; h(n)=heuristic function=manhattan distance.
* Here it will selects minimum f(n) value state everytime.
* It will give Goal Node path at the time of selection and expansion instead of at state generation. So, it is giving optimal path cost.

**d)** using IDA\*: (Informed Search Technique)

* f(n)=g(n)+h(n) where g(n)=cost to reach to n ; h(n)=heuristic function=manhattan distance.
* Here we have to give *maxBound,* so that when the goal is not present or our program is in infinite searching condition it will stop when it reached it.
* Here if we don’t find goal node within the limited *f(n)* value, we will clear *frontier* and *explored lists* and we increase our limit *f(n) value*  and again we will do searching.
* It is like IDS but implementing *f(n) value*.It is enqueing more same nodes again, if we don’t find Goal state.
* Memory is not restricted.
* May not find solution.

**Ex1:** input(*Ex1\_in.txt*) (complete output in *Ex1\_out.txt* in specified folder)

[2, 0, 3, 4] [1,2,3,4]

[1, 5, 6, 7] --> [5,6,7,8]

[9, 11, 12, 8] [9,10,11,12]

[13, 10, 14, 15] [13,14,15,0]

**Ex2:**

[ 0, 1 ]

[ 2, 3 ] --> Not possible

*For increasing it will take so much time to give Goal State.*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **BFS** | | **DFS** | | | **A\*** | | | **IDA\*** | |
|  | Ex1 | Ex2 | | Ex1 | Ex2 | | Ex1 | Ex2 | Ex1(given limit=4, maxbound=1000) | Ex2(given limit=4,maxbound=100) |
| Optimal | Yes | Yes | | No | No | | Yes | Yes | Yes | Yes |
| Solution Found | Yes | Yes | | No(Maximum Recursion Exceeded) | Yes | | Yes | Yes | Yes | Yes |
| Frontier Size(when goal found) | 5131 | 2 | | 298(when terminated) | 6 | | 25 |  | 24 |  |
| Explored Nodes(when goal found) | 3172(pathcost=11) | 12 | | 201 | 12 | | 12(Pathcost=11) |  | 12(pathcost=11) | 12 |
| Execution time | 0:00:10 sec | 0:00:00 sec  (<1sec) | | - | 0:00:00 sec  (<1sec) | | 0:00:00 sec  (<1sec) | 0:00:00 sec  (<1sec) | 0:00:00 sec  (<1sec) | 0:00:00 sec  (<1sec) but more than A\* |

**2**. **NxN board coloring such that no two-neighbours are same color: using BFS and DFS search algorithms.**

**Ex1**:

**BFS**:input size=n=4 (Random generation) --> same input to **DFS**(Ex1\_in.txt)

(complete output in Ex1\_out.txt)

**BFS O/P DFS O/P**

**(EX\_1.out.txt) (EX\_1.out.txt)**

[1, 1, 1, 3] [2, 1, 3, 1] [1, 2, 1, 3]

[2, 2, 1, 3] --> [1, 2, 1, 3] [2, 1, 3, 1]

[1, 3, 1, 1] [3, 1, 2, 1] [1, 3, 1, 2]

[1, 2, 2, 4] [1, 2, 1, 4] [4, 1, 2, 1]

**Ex2**: **(EX\_2.out.txt) (EX\_2.out.txt)**

[1, 1, 3, 3, 2] [2, 1, 3, 1, 2] terminated due to Maximum

[2, 2, 1, 4, 4] [4, 2, 1, 3, 4] Recursion Depth Exceeded.

[4, 4, 2, 1, 1] --> [1, 4, 2, 4, 1]

[4, 2, 4, 1, 2] [4, 2, 4, 1, 2]

[1, 4, 1, 4, 1] [1, 4, 1, 4, 1]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **BFS** | | | **DFS** | |
|  | Ex1 | Ex2 | Ex1 | | Ex2 |
| Optimal | Yes | Yes | No | | No |
| Solution Found | Yes | Yes | Yes | | No(Maximum Recursion Depth Exceeded) |
| Frontier nodes(when goal state found) | 2968 Nodes | 9879 Nodes | 1385 Nodes | | 6386(when terminated) |
| Explored nodes(when goal state found) | 167(pathcost=4) | 365(pathcost=4) | 82 (pathcost=82) | | 199(when terminated) |
| Execution time | 0:00:03 sec | 0:00:16 sec | 0:00:07 sec | | More than BFS |