PAGE NO :
DATE :

Week - 4

→ Iterative deepening algo :-

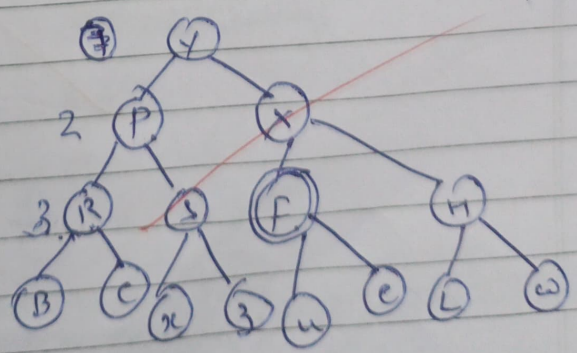
Write depth limit search function which would perform dfs till given max_limit

call the limit search function from range(0, max_lim)

```
goal = global_var
func IDDFS(Graph, limit), start)
    for depth in 0 to limit :
        result = DFS(start, depth, 0)
        if result:
            return result
        else :
            return NONE

func . DFS(root, limit, depth)
    if root == goal
        return root
    if depth == limit+1 : return
    for child in root.children:
        DFS(child, limit, depth+1)
```

Tree :-

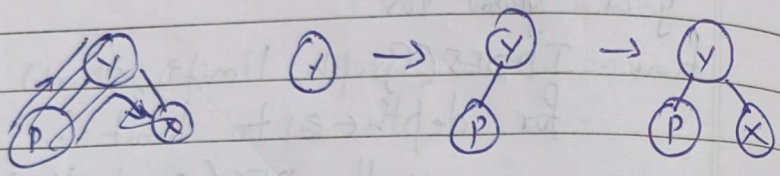


PAGE NO :
DATE :

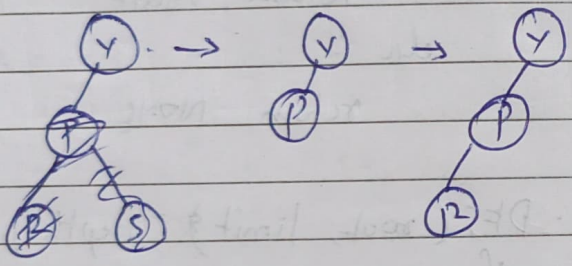
A* 8 puzzle

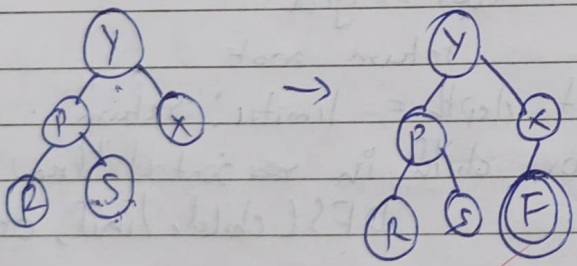
At max-depth = 1

At max-depth = 2



At max-depth = 3





found.

PAGE NO :
DATE :

→ A* 8 puzzle

dir = [(0,-1), (0,1), (1,0), (-1,0)]

goal_state = "global_val"

((2,8,1), (0,4,3), (7,6,5))

g = dictionary of goal position

~~def manhatten(curr):~~

~~def A*(start, goal):~~

vis = set()

h = heap ; h ← (manhatten(start), (start, 0))

v ← (start)

func manhatten (curr):

for i in curr:

for j in i:

v = curr[i][j]

s += abs(i - g[v]) + abs(j - g[v])

func moves (curr, g):

x, y = locate (curr)

for dir in dir:

nx, ny = x + d[0], y + d[1]

if 0 <= nx <= 3 and 0 <= ny <= 3

curr1 = swap(curr[x][y], curr[nx][ny])

f = g + 1 + manhatten(curr1)

h ← (f, (curr1, g))

func

def a_star (curr, g)

if curr v ← curr

if curr = goal :

return

c = heap.pop()

moves (curr, g)

c = heap.pop()

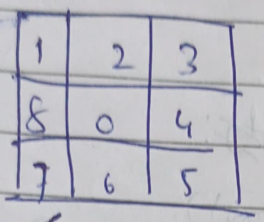
a_start (c[0], c[1])

PAGE NO :
DATE :

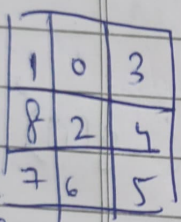
Start input Start start

a-star(start, 0)

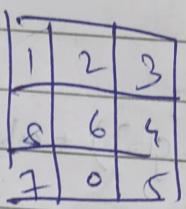
Sol^n



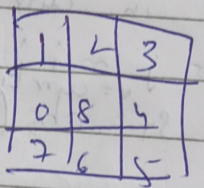
- f(n) = g(n) + manhatten(n)
  for initial g = 0
  g = No of steps



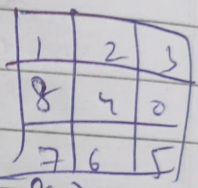
f(n) = 1 + manhatten (curr)



f(n) = 1 + manhatten (curr)



f(n) = 1 + manhatten (curr)



f(n) = 1 + manhatten (curr)

Solved
13/10/24