Assignment No1 and 2:BFS Algorithm and 8 Puzzle Algorithm

CSE-0408 Summer 2021

Name:Nishu Akter
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
email:nishuakter309@gmail.com

***Abstract*—code-1:Breadth First Search (BFS) and other graph traversal techniques are widely used for measuring large unknown graphs, such as online social networks. It has been empirically observed that incomplete BFS is biased toward high degree nodes.Breadth-first search is so named because it divides the discovered and undiscovered vertices uniformly across the tree.**

code-2:A heuristic function (algorithm) or simply a heuristic is a shortcut to solving a problem when there are no exact solutions for it or the time to obtain the solution is too long. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

n

***Index Terms*—heuristic,puzzle,traverse,graph.**

I. Introduction

code-1:Breath first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighbouring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal.

code-2:We can solve Heuristic function by the eight puzzle problem. It is also known as the name of N puzzle problem or sliding puzzle problem. N-puzzle that consists of N tiles (N+1 titles with an empty tile) where N can be 8, 15, 24 and so on. in these types of problems we have given a initial state or initial configuration (Start state) and a Goal state or Goal Configuration. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order.

II. VARIANTS

code-1:The two variants of Best First Search are Greedy Best First Search and A* Best First Search. Greedy BFS:Greedy breath first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms.

A* BFS:A * BFS algorithm is a searching algorithm that searches for the shortest path between the initial and the final state. It is used in various applications, such as maps. In maps the A* BFS algorithm is used to calculate the shortest distance between the source (initial state) and the destination (final state).

code-2:Breath First Search Algorithm(Greedy search) Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of bestfirst search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function.

A* Search Algorithm: A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function.

III. BFS Code

!/usr/bin/env python
coding: utf-8

In[ ]:

from collections import defultdict
from queue import Queue

In[ ]:

class Graph():
def$_{i}nit_{(}self, directed):$
$self.graph = defaultdict(list)$
$self.directed = directed$

def add$_e dge(self, u, v):$
$if self.directed:$
$self.graph[u].append(v)$
$else:$
$self.graph[u].aparend(v)$
$self.graph[v].aparend(u)$
$def bfs(self, vertex):$
$visited = []$
$queue = Queue()$
$queue.put(vertex)$

while not queue empty():
vertex=queue.get()
if vertex in visited:
continue
print(vertex,end =” ”)
visited.aparend(vertex)

for neighbour in self.graph[vertex]:
if neighbour != None:
queue.put(neighbour)

In[ ]:

g = Graph(True)

In[4]:

g.add$_e dge('s','r')$
$g.add_e dge('s','v')$
$g.add_e dge('s','x')$
$g.add_e dge('r','t')$
$g.add_e dge('v','w')$
$g.add_e dge('x','r')$
$g.add_e dge('x','u')$
$g.add_e dge('t','x')$
$g.add_e dge('t','u')$
$g.add_e dge('t','y')$
$g.add_e dge('w','s')$
$g.add_e dge('w','y')$
$g.add_e dge('u', None)$
$g.add_e dge('y','u')$

In[5]:

g.graph

In[1]:

g.bfs(’s’)

Output:srvxtwuy

IV. Puzzle Code

include ¡bits/stdc++.h¿
using namespace std;
define N 3

struct Node

Node* parent;
int mat[N][N];

int x, y;

int cost;

int level;

int printMatrix(int mat[N][N])
for (int i = 0; i ¡ N; i++)
for (int j = 0; j ¡ N; j++)
printf(”printf(” ”);

Node* newNode(int mat[N][N], int x, int y, int newX, int newY, int level, Node* parent)
Node* node = new Node;

node-¿parent = parent;

memcpy(node-¿mat, mat, sizeof node-¿mat);

swap(node-¿mat[x][y], node-¿mat[newX][newY]);
node-¿cost = INT$_M AX;$

node-¿level = level;
node-¿x = newX;
node-¿y = newY;

return node;

int row[] = 1, 0, -1, 0 ;
int col[] = 0, -1, 0, 1 ;

int calculateCost(int initial[N][N], int final[N][N])
int count = 0;
for (int i = 0; i ¡ N; i++)
for (int j = 0; j ¡ N; j++)
if (initial[i][j] initial[i][j] != final[i][j])
count++;
return count;

int isSafe(int x, int y)
return (x ¿= 0 x ¡ N y ¿= 0 y ¡ N);

void printPath(Node* root)
if (root == NULL)
return;
printPath(root-¿parent);
printMatrix(root-¿mat);

printf(” ”);

struct comp
bool operator()(const Node* lhs, const Node* rhs) const
return (lhs-¿cost + lhs-¿level) ¿ (rhs-¿cost + rhs-¿level);

;

void solve(int initial[N][N], int x, int y,
int final[N][N])

priority$_{q}ueue < Node*, std :: vector < Node* > , comp > pq$;

Node* root = newNode(initial, x, y, x, y, 0, NULL);
root-¿cost = calculateCost(initial, final);

pq.push(root);

while (!pq.empty())

Node* min = pq.top();
pq.pop();

if (min-¿cost == 0)
printPath(min);
return;

for (int i = 0; i ¡ 4; i++)
if (isSafe(min-¿x + row[i], min-¿y + col[i]))

Node* child = newNode(min-¿mat, min-¿x,
min-¿y, min-¿x + row[i],
min-¿y + col[i],
min-¿level + 1, min);
child-¿cost = calculateCost(child-¿mat, final);

pq.push(child);

int main()

int initial[N][N]

1, 2, 3,
5, 6, 0,
7, 8, 4,
;

int final[N][N]
1, 2, 3,
5, 8, 6,
0, 7, 4

int x = 1, y = 2;

solve(initial, x, y, final);

return 0;

Output: 1 2 3
5 6 0
7 8 4

1 2 3
5 0 6
7 8 4

1 2 3
5 8 6
7 0 4

1 2 3
5 8 6
0 7 4

V. ALGORITHM FOR BFS AND PUZZLE

code-1:Let S be the root/starting node of the graph.
Step 1: Start with node S and enqueue it to the queue.
Step 2: Repeat the following steps for all the nodes in the graph.
Step 3: Dequeue S and process it.
Step 4: Enqueue all the adjacent nodes of S and process them. [END OF LOOP]
Step 6: EXIT

code-2:Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile. The empty space can only move in four directions (Movement of empty space) 1. Up
2. Down
3. Right or Left
5. The empty space cannot move diagonally and can take only one step at a time.

VI. CONCLUSION

code-1:The breadth-first search technique is a method that is used to traverse all the nodes of a graph or a tree in a breadth-wise manner. This technique is mostly used to find the shortest path between the nodes of a graph or in applications that require us to visit every adjacent node like in networks.

code-2:The heuristic function is a way to inform the search about the direction to a goal. It provides an informed way to guess which neighbor of a node will lead to a goal. There is nothing magical about a heuristic function. It must use only information that can be readily obtained about a node.

ACKNOWLEDGMENT

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.