BFS

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 100

int visited[MAX];

int adjacencyMatrix[MAX][MAX];

bool isGoal(int node, in$St goal) {

    return node == goal;

}

void startGraph(int nodes) {

    for (int i = 0; i < nodes; i++) {

        visited[i] = 0;

        for (int j = 0; j < nodes; j++) {

            adjacencyMatrix[i][j] = 0;

        }

    }

}

void addEdge(int start, int end) {

    if (start >= 0 && start < MAX && end >= 0 && end < MAX) {

        adjacencyMatrix[start][end] = 1;

        adjacencyMatrix[end][start] = 1; // Comment this out for directed graph

    }

}

void bfs(int startNode, int goal, int nodes) {

    int queue[MAX];

    int front = 0, rear = 0;

    int explored[MAX] = {0};

    int now;

    int flag = 0;

    queue[rear++] = startNode;

    visited[startNode] = 1;

    while (front < rear && flag==0) {

        now = queue[front++];

       printf("Doing goal test on %d\n", now);

        if (isGoal(now, goal)) {

            printf("\nGoal node %d found!\n", now);

            flag = 1;

            break;

        }

        if (flag==0){

        explored[now] = 1;

        for (int i = 0; i < nodes; i++) {

            if (adjacencyMatrix[now][i] == 1 && !visited[i] && !explored[i]) {

                printf("Added to queue %d\n", i);

                visited[i] = 1;

                queue[rear++] = i;

            }

        }

    }}

    if(flag==0){

    printf("\nGoal node %d not found.\n", goal);}

    printf("Explored nodes: ");

    for (int i = 0; i < nodes; i++) {

        if (explored[i]) {

            printf("%d ", i);

        }

    }

    printf("\n");

       printf("\nCurrent queue: ");

    for (int i = front; i < rear; i++) {

        printf("%d ", queue[i]);

    }

    printf("\n");

}

int main() {

    int start, end, startNode, goalNode, nodes, edges;

    printf("Enter the number of nodes: ");

    scanf("%d", &nodes);

    printf("Enter the number of edges: ");

    scanf("%d", &edges);

    startGraph(nodes);

    printf("Enter the edges (start end):\n");

    for (int i = 0; i < edges; i++) {

        printf("Edge %d: ", i + 1);

        scanf("%d %d", &start, &end);

        addEdge(start, end);

    }

    printf("Enter the starting node for BFS: ");

    scanf("%d", &startNode);

    printf("Enter the goal node for BFS: ");

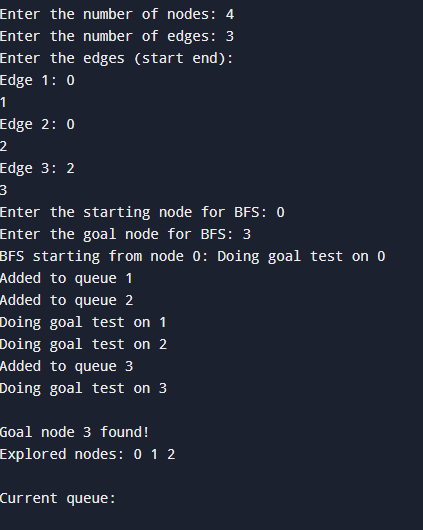
    scanf("%d", &goalNode);

    printf("BFS starting from node %d: ", startNode);

    bfs(startNode, goalNode, nodes);

    return 0;

}



DFS:

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 100

int flag = 0;

int visited[MAX];

int adjacencyMatrix[MAX][MAX];

bool isGoal(int node, int goal) {

    return node == goal;

}

void startGraph(int nodes) {

    for (int i = 0; i < nodes; i++) {

        visited[i] = 0;

        for (int j = 0; j < nodes; j++) {

            adjacencyMatrix[i][j] = 0;

        }

    }

}

void addEdge(int start, int end) {

    if (start >= 0 && start < MAX && end >= 0 && end < MAX) {

        adjacencyMatrix[start][end] = 1;

        adjacencyMatrix[end][start] = 1; // Comment this out for directed graph

    }

}

void dfs(int startNode, int goal, int nodes) {

    visited[startNode] = 1;

   printf("\nDoing goal test on %d", startNode);

    if (isGoal(startNode, goal)) {

        printf("\nGoal node %d found!\n", startNode);

        flag = 1;

        return;

    }

    for (int i = 0; i < nodes; i++) {

        if (adjacencyMatrix[startNode][i] == 1 && visited[i] == 0 && flag == 0) {

            dfs(i, goal, nodes);

        }

    }

}

int main() {

    int start, end, startNode, goalNode, nodes, edges;

    printf("Enter the number of nodes: ");

    scanf("%d", &nodes);

    printf("Enter the number of edges: ");

    scanf("%d", &edges);

    startGraph(nodes);

    printf("Enter the edges (start end):\n");

    for (int i = 0; i < edges; i++) {

        printf("Edge %d: ", i + 1);

        scanf("%d %d", &start, &end);

        addEdge(start, end);

    }

    printf("Enter the starting node for DFS: ");

    scanf("%d", &startNode);

    printf("Enter the goal node for DFS: ");

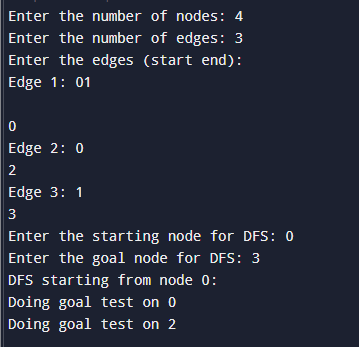
    scanf("%d", &goalNode);

    printf("DFS starting from node %d: ", startNode);

    dfs(startNode, goalNode, nodes);

    return 0;

}



A-Star

n = int(input("Enter number of nodes"))

adjMat = [[0] \* n for \_ in range(n)]

e = int(input("Enter number of edges"))

for \_ in range(e):

    s = int(input("Label of start node"))

    e = int(input("label of end node"))

    w = int(input("Enter weight"))

    adjMat[s][e] = w

    adjMat[e][s] = w #comment out for directed graph

goal = int(input("What is goal node?"))

print("I am trying to print goal node", goal)

h = []

for i in range(n):

    h.append(int(input("What is h("+ str(i)+ ")?")))

start = int(input("What is start node?"))

now = start

dis = 0

visited = [0] \* n

flag = 0

while flag == 0 and 0 in visited:

    visited[now] = 1

    loop\_flag = 0

    print("Unvisited neighbours of ", now, ", with their f = g + h are:")

    mini = h[start]

    min\_indx = -1

    for i in range(n):

        if adjMat[now][i] != 0 and visited[i] == 0:

            loop\_flag = 1

            print("h(",i,") = ",h[i])

            print("g(",i,") = ",dis," + ",adjMat[now][i]," = ",dis + adjMat[now][i])

            print("f(", i, ") = ",h[i]+dis+adjMat[now][i])

            if mini > h[i]+adjMat[now][i]:

                mini = h[i]+adjMat[now][i]

                min\_indx = i

    print("Goal test on ", now)

    #if loop\_flag == 0:

     #   break

    if now == goal:

        print("Goal node found. Distance is ", dis)

        flag = 1

        break

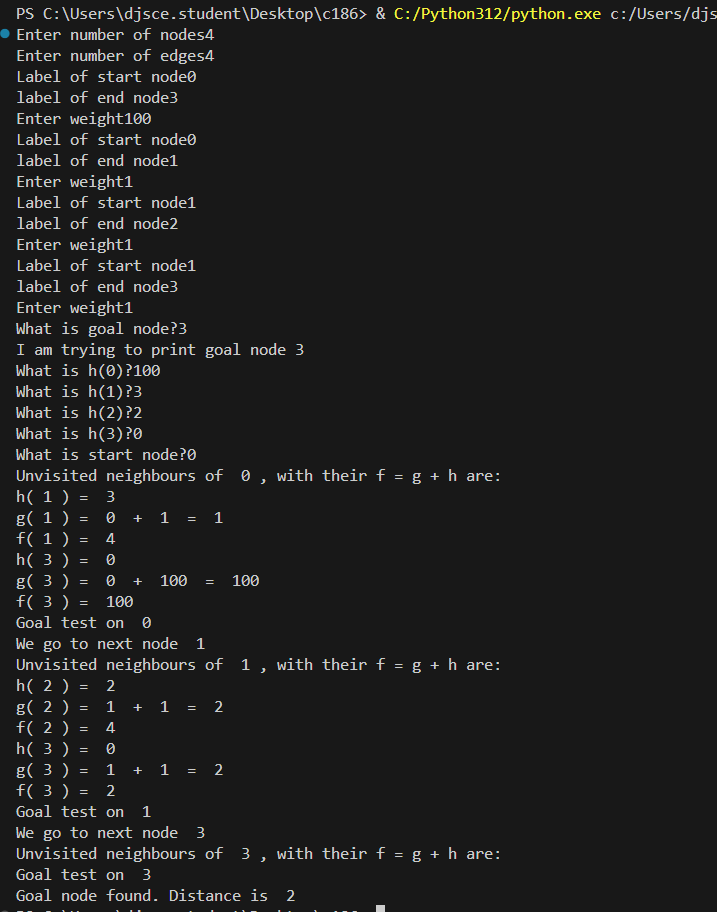
    print("We go to next node ", min\_indx)

    dis += adjMat[now][min\_indx]

    now = min\_indx

if flag==0:

    print("Unsuccessful search")



Hill Climbing

def neighbours(now):

    n = len(now)

    num\_empty = 0

    for i in range(n):

        if now[i] == []:

            num\_empty += 1

    ans = []

    marked = 0

    for i in range(n):

        if now[i] == []:

            break

        else:

            for j in range(n):

                if j != i:

                    new\_state = [stack.copy() for stack in now]

                    block = new\_state[i].pop()

                    new\_state[j].append(block)

                    ans.append(new\_state)

    return ans

def find\_below(goal):

    below = {}

    for i in range(len(goal)):

        for j in range(len(goal[i])):

            if j==0:

                below[goal[i][j]] = 0

            else:

                below[goal[i][j]] = goal[i][j-1]

    return below

def heuristic(now, below):

    ans = 0

    for i in range(len(now)):

        for j in range(len(now[i])):

            if j==0:

                if below[now[i][j]] == 0:

                    ans += 1

                else:

                    ans -= 1

            else:

                if below[now[i][j]] == now[i][j-1]:

                    ans += 1

                else:

                    ans -= 1

    return ans

def hill\_climbing(start, below):

    now = start

    while True:

        n = len(neighbours(now))

        max\_till\_now = -1

        for i in range(n):

            if max\_till\_now < heuristic(neighbours(now)[i], below):

                    max\_till\_now = heuristic(neighbours(now)[i], below)

                    max\_indx = i

        if max\_till\_now > heuristic(now, below):

            print("We found a bigger node. Go to ", neighbours(now)[max\_indx] )

            now = neighbours(now)[max\_indx]

        else:

            print("No bigger neighbour. End here")

            return

n = int(input("Enter number of stacks"))

initial = [[] for \_ in range(n)]

final = [[] for \_ in range(n)]

for i in range(n):

    m = int(input("For initial state, how big is is stack"+ str(i+1)))

    for j in range(m):

        initial[i].append(input("Write label of block (bottom to top)"))

    m = int(input("For final state, how big is is stack" + str( i+1)))

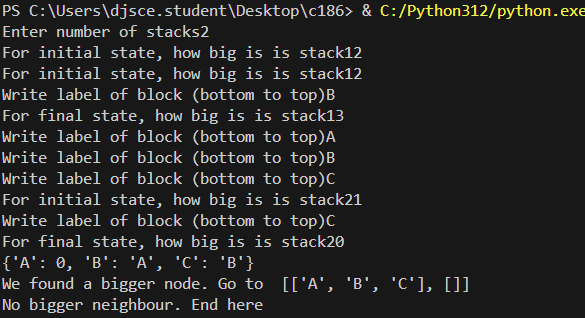
    for j in range(m):

        final[i].append(input("Write label of block (bottom to top)"))

below = find\_below(final)

print(below)

hill\_climbing(initial, below)



8-Queens problem

#include <stdio.h>

#include <stdlib.h>

struct arr{

int array[100];

};

void write(struct arr new, int n){

printf("Solution:\n");

for (int i=0; i<n; i++){

printf("In row number %d, queen is in column number %d\n", i+1, new.array[i]+1); }

}

int place(int k, int i, int x[]) {

for (int j=0; j<k; j++){

if (x[j]==i){

return 0; }

if (abs(x[j] - i)==abs(j-k)){

return 0;

} }

return 1;

}

struct arr Nqueens(int k, int n,int x[]){

struct arr new;

for (int j=0; j<k; j++){

new.array[j] = x[j];

}

for (int i=0; i<n; i++){

if (place(k, i, x) == 1){

new.array[k] = i;

if (k==n-1){

write(new, n);}

else{

Nqueens(k+1, n, new.array);

}

}

}

}

int main() {

int n;

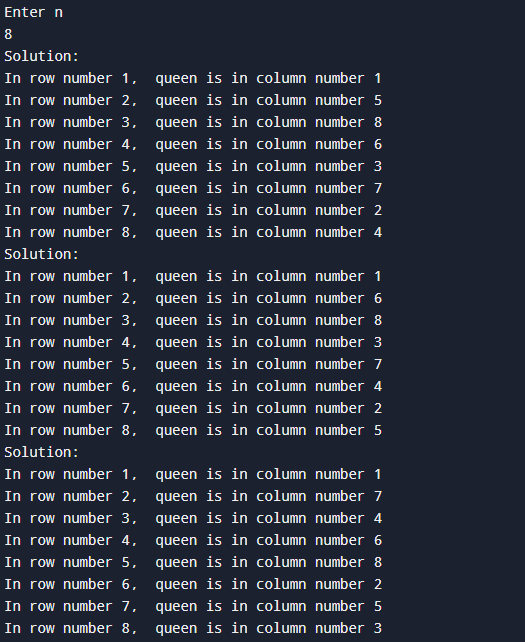
int x[100];

printf("Enter n\n");

scanf("%d",&n);

Nqueens(0, n, x);

return 0;}



Family tree in PROLOG:

Knowledge base:

/\* Facts \*/

male(jack).

male(oliver).

male(ali).

male(james).

male(simon).

male(harry).

female(helen).

female(sophie).

female(jess).

female(lily).

parent\_of(jack,jess).

parent\_of(jack,lily).

parent\_of(helen, jess).

parent\_of(helen, lily).

parent\_of(oliver,james).

parent\_of(sophie, james).

parent\_of(jess, simon).

parent\_of(ali, simon).

parent\_of(lily, harry).

parent\_of(james, harry).

/\* Rules \*/

father\_of(X,Y):- male(X),

parent\_of(X,Y).

mother\_of(X,Y):- female(X),

parent\_of(X,Y).

grandfather\_of(X,Y):- male(X),

parent\_of(X,Z),

parent\_of(Z,Y).

grandmother\_of(X,Y):- female(X),

parent\_of(X,Z),

parent\_of(Z,Y).

sister\_of(X,Y):- %(X,Y or Y,X)%

female(X),

father\_of(F, Y), father\_of(F,X),X \= Y.

sister\_of(X,Y):- female(X),

mother\_of(M, Y), mother\_of(M,X),X \= Y.

aunt\_of(X,Y):- female(X),

parent\_of(Z,Y), sister\_of(Z,X),!.

brother\_of(X,Y):- %(X,Y or Y,X)%

male(X),

father\_of(F, Y), father\_of(F,X),X \= Y.

brother\_of(X,Y):- male(X),

mother\_of(M, Y), mother\_of(M,X),X \= Y.

uncle\_of(X,Y):-

parent\_of(Z,Y), brother\_of(Z,X).

ancestor\_of(X,Y):- parent\_of(X,Y).

ancestor\_of(X,Y):- parent\_of(X,Z),

ancestor\_of(Z,Y).

Queries:

