**AI Lab – 2**

**Analysis of A\* vs Best First Search Algorithm**

Name: Mokshit Oswal

Class: AI-B

Roll Number – 32

PRN: 12111304

Batch B2

1. **Best First Search**

#include <stdio.h>

#include <stdlib.h>

#define INT\_MAX\_VALUE 2147483647

int myMin(int a, int b)

{

if (a < b)

return a;

return b;

}

int myAbs(int i)

{

if (i < 0)

{

return -i;

}

return i;

}

int main()

{

int rows, columns;

printf("Enter No. of rows : ");

scanf("%d", &rows);

printf("Enter No. of columns : ");

scanf("%d", &columns);

int maze[rows][columns];

printf("Start Co-ordinates : ");

int s\_row;

scanf("%d", &s\_row);

int s\_col;

scanf("%d", &s\_col);

printf("End Co-ordinates : ");

int e\_row;

scanf("%d", &e\_row);

int e\_col;

scanf("%d", &e\_col);

int k;

printf("How many blocks you want to put? ");

scanf("%d", &k);

for (int i = 0; i < k; i++)

{

printf("Enter block Co-ordinates: ");

int k1, k2;

scanf("%d", &k1);

scanf("%d", &k2);

maze[k1][k2] = -1;

}

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < columns; j++)

{

if (maze[i][j] != -1)

{

maze[i][j] = myAbs(e\_row - i) + myAbs(e\_col - j);

}

}

}

int openlist[rows][columns];

int closedlist[rows][columns];

printf("\nHeuristics Matrix : \n");

for (int i = 0; i < rows; i++)

{

printf("----");

}

for (int i = 0; i < rows; i++)

{

printf("\n");

for (int j = 0; j < columns; j++)

{

openlist[i][j] = INT\_MAX\_VALUE;

closedlist[i][j] = 0;

printf("%d | ", maze[i][j]);

}

printf("\n");

for (int i = 0; i < rows; i++)

{

printf("----");

}

}

printf("\n");

openlist[s\_row][s\_col] = maze[s\_row][s\_col];

int r\_mat[] = {0, 0, 1, -1};

int c\_mat[] = {1, -1, 0, 0};

int x = rows \* columns;

while (x--)

{

int Min = INT\_MAX\_VALUE;

int r = 0, c = 0;

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < columns; j++)

{

if (Min > openlist[i][j] && closedlist[i][j] == 0)

{

Min = openlist[i][j];

r = i;

c = j;

}

}

}

printf("Current coordinates: %d %d\n", r, c);

if (r == e\_row && c == e\_col)

{

printf("Coordinates Found!!");

return 0;

}

openlist[r][c] = INT\_MAX\_VALUE;

closedlist[r][c] = 1;

for (int i = 0; i < 4; i++)

{

int n\_r = r + r\_mat[i];

int n\_c = c + c\_mat[i];

if (n\_r >= 0 && n\_r < rows && n\_c >= 0 && n\_c < columns && maze[n\_r][n\_c] != -1)

{

openlist[n\_r][n\_c] = maze[n\_r][n\_c];

}

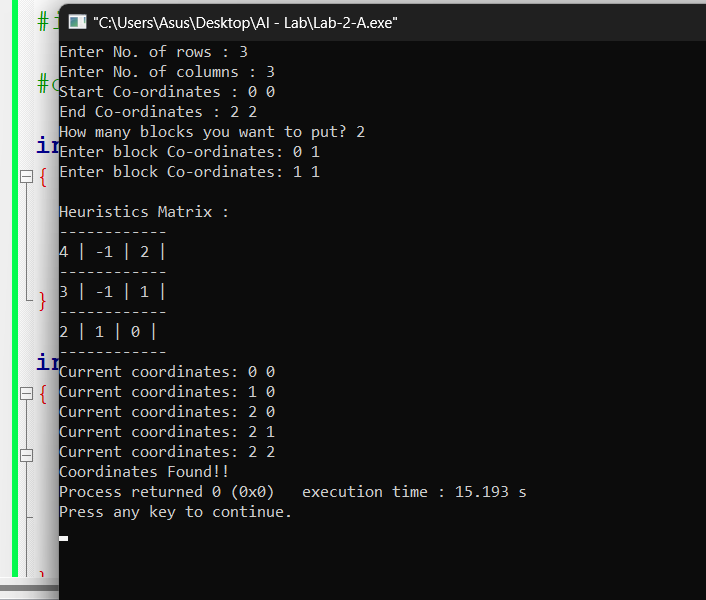
}

}

printf("Did not find any path!!");

return 0;

}



1. **A\***

**Code:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define INT\_MAX 2147483647**

**int min(int a, int b) {**

**return (a < b) ? a : b;**

**}**

**int rows, columns;**

**int\* minimum(int openlist[rows][columns], int closedlist[rows][columns], int maze[rows][columns]) {**

**int Min = INT\_MAX;**

**int r = -1, c = -1;**

**for (int i = 0; i < rows; i++) {**

**for (int j = 0; j < columns; j++) {**

**if (Min > openlist[i][j] && closedlist[i][j] == 0) {**

**Min = openlist[i][j];**

**r = i;**

**c = j;**

**}**

**}**

**}**

**printf("Current coordinates: %d %d\n", r, c);**

**int\* ans = (int\*)malloc(2 \* sizeof(int));**

**ans[0] = r;**

**ans[1] = c;**

**return ans;**

**}**

**int abs(int i) {**

**return (i < 0) ? -i : i;**

**}**

**int main() {**

**int i, j;**

**printf("Enter No. of rows: ");**

**scanf("%d", &rows);**

**printf("Enter No. of columns: ");**

**scanf("%d", &columns);**

**int maze[rows][columns];**

**printf("Start Coordinates: ");**

**int s\_row;**

**scanf("%d", &s\_row);**

**int s\_col;**

**scanf("%d", &s\_col);**

**printf("End Coordinates: ");**

**int e\_row;**

**scanf("%d", &e\_row);**

**int e\_col;**

**scanf("%d", &e\_col);**

**printf("Enter Heuristic Matrix (Costs to reach the destination):\n");**

**for (i = 0; i < rows; i++) {**

**for (j = 0; j < columns; j++) {**

**scanf("%d", &maze[i][j]);**

**if (maze[i][j] == -1) {**

**// Treat -1 in the heuristic matrix as a block by setting it to INT\_MAX**

**maze[i][j] = INT\_MAX;**

**}**

**}**

**}**

**int openlist[rows][columns];**

**int closedlist[rows][columns];**

**printf("Heuristic Matrix : \n");**

**for (i = 0; i < rows; i++) {**

**for (j = 0; j < columns; j++) {**

**openlist[i][j] = INT\_MAX;**

**closedlist[i][j] = 0;**

**printf("%d | ", maze[i][j]);**

**}**

**printf("\n-------------------------------\n");**

**}**

**openlist[s\_row][s\_col] = 0; // Initialize start node with cost 0**

**int r\_mat[] = {0, 0, 1, -1};**

**int c\_mat[] = {1, -1, 0, 0};**

**int x = rows \* columns;**

**while (x--) {**

**int\* r\_p = minimum(openlist, closedlist, maze);**

**int r = r\_p[0], c = r\_p[1];**

**if (r == e\_row && c == e\_col) {**

**printf("Coordinates Found!!");**

**break;**

**}**

**if (r == -1 || c == -1) {**

**continue;**

**}**

**closedlist[r][c] = 1;**

**for (int i = 0; i < 4; i++) {**

**int n\_r = r + r\_mat[i];**

**int n\_c = c + c\_mat[i];**

**if (n\_r >= 0 && n\_r < rows && n\_c >= 0 && n\_c < columns && closedlist[n\_r][n\_c] == 0) {**

**int new\_cost = openlist[r][c] + 1; // Assuming a cost of 1 for each step**

**if (new\_cost < openlist[n\_r][n\_c]) {**

**openlist[n\_r][n\_c] = new\_cost;**

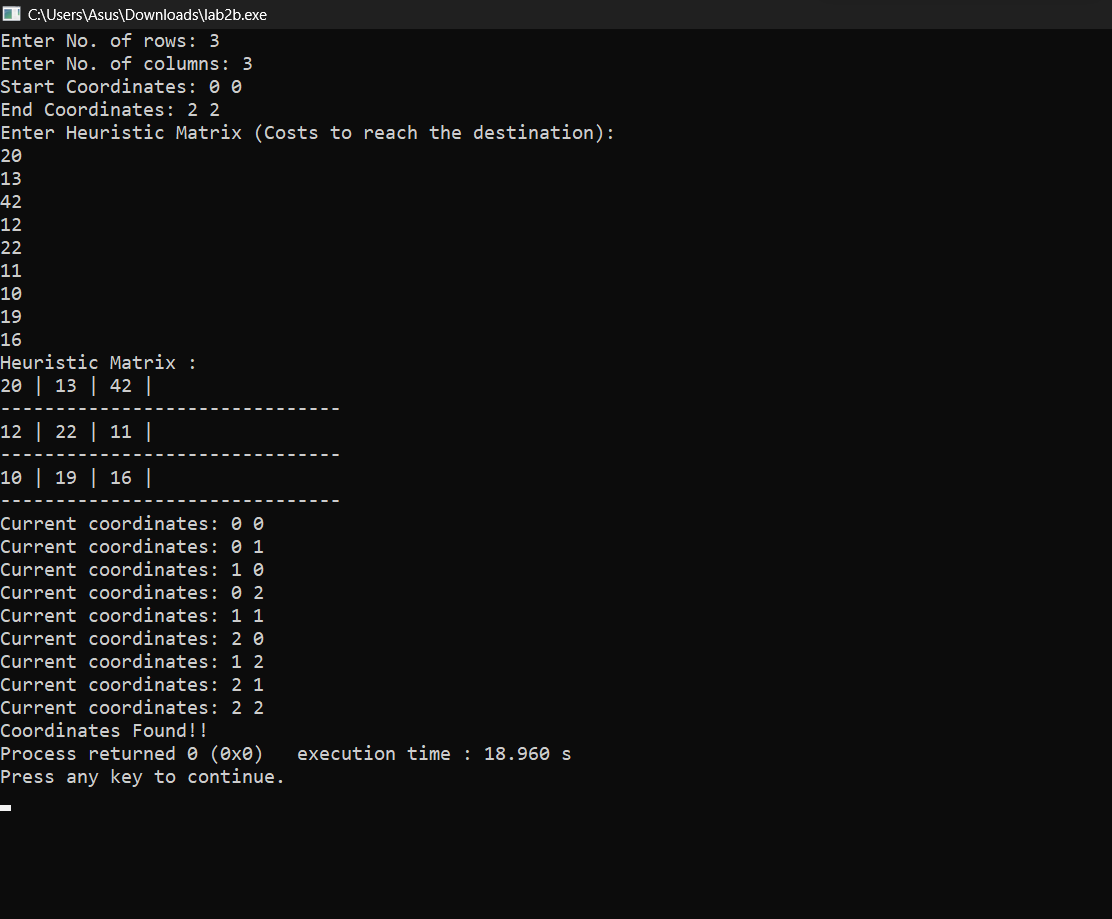
**}**

**}**

**}**

**}**

**return 0;**

**}  
  
**