**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

**ĐỀ TÀI:**

**TÌM HIỂU VÀ CÀI ĐẶT THUẬT TOÁN A\***

Giáo viên hướng dẫn: **MAI CƯỜNG THỌ**

Sinh viên thực hiện: **TRẦN THỊ MỸ LINH**

**VÕ LƯƠNG HOÀNG HUY**

**PHAN TRƯƠNG DUY LUẬN**

Lớp**: 60.CNTT-3**

**Năm học: 2020 - 2021**

Mục Lục

LỜI MỞ ĐẦU………………………………………………………………….…3

CHƯƠNG I: HEURISTIC VÀ NGÔN NGỮ SỬ DỤNG(C++)…………………3

I. HEURISTIC………………………………………………………………...3

1. Tổng quan về Heuristic Seach…………………………………………..3

2. Chức năng của heuristic…………………………………………………3

3.Phương pháp xây dựng thuật giải Heuristic……………………………..3

4. Ưu điểm của thuật giải Heuristic………………………………………..4

5. Ứng dụng kinh nghiệm bài toán………………………………………...4

II. LÝ THUYẾT VỀ NGÔN NGỮ SỬ DỤNG(C++)………………………….4

1. Tổng quan về ngôn ngữ C++……………………………………………..4

2. Cấu trúc một chương trình trong C++……………………………………4

3. Các kiểu dữ liệu trong C++………………………………………………6

4. Biến trong C++…………………………………………………………..6

CHƯƠNG II: CÀI ĐẶT CHƯƠNG TRÌNH VÀ THUẬT TOÁN A\*…………..7

1. Giới thiệu và cài đặt devC…………………………………………………7

2. Tìm hiểu A\*………………………………………………………………..8

2.a. Tìm Hiểu………………………………………………………………8

2.b. Ý tưởng………………………………………………………………..8

3. Thuật giải A\*………………………………………………………………9

4. Giải thuật lập trình………………………………………………………..10

CHƯƠNG III: KẾT LUẬN……………………………………………………...22

1. Kết quả đạt được…………………………………………………………..22

2. Mô phỏng trên đồ thị………………………………………………………22

3. Nhận xét……………………………………………………………………23

4. Kết luận……………………………………………………………………...24

TÀI LIỆU THAM KHẢO………………………………………………………..24

**LỜI MỞ ĐẦU**

HEURISTIC SEARCH là các kỹ thuật dựa trên kinh nghiệm để giải quyết vấn đề, học hỏi hay khám phá nhằm đưa ra một giải pháp mà không được đảm bảo là tối ưu. Với việc nghiên cứu khảo sát không có tính thực tế, các phương pháp heuristic được dùng nhằm tăng nhanh quá trình tìm kiếm với các giải pháp hợp lý thông qua các suy nghĩ rút gọn để giảm bớt việc nhận thức vấn đề khi đưa ra quyết định.

**CHƯƠNG I: HEURISTIC VÀ NGÔN NGỮ SỬ DỤNG(C++)**

1. **LÝ THUYẾT VỀ HEURISTIC:**
2. **Tổng quan về Heuristic search:**

Heuristic: là các kỹ thuật dựa trên kinh nghiệm để giải quyết vấn đề, nhằm đưa ra một giải pháp mà không được đảm bảo là tối ưu (theo Wiki)

Heuristic function: Hàm đánh giá dựa trên kinh nghiệm, dựa vào đó để xếp hạng thứ tự tìm kiếm, cách chọn hàm đánh giá quyết định nhiều đến kết quả tìm kiếm.

1. **Chức năng của Heuristic:**

Các chương trình giải quyết những vấn đề trí tuệ nhân tạo sử dụng Heuristic cơ bản theo 2 dạng:

* Vấn đề có thể không có giải pháp chính xác vì những điều không rõ ràng trong diễn đạt vấn đề hoặc trong các dữ liệu có sẵn.
* Vấn đề có thể có giải pháp chính xác, nhưng chi phí tính toán để tìm ra nó không cho phép.

**3. Phương pháp xây dựng thuật giải Heuristic:**

Thuật giải Heuristic gồm 2 phần: Hàm đánh giá Heuristic và thuật toán để sử dụng nó trong tìm kiếm không gian trạng thái.

Có nhiều các để xây dựng một thuật giải Heuristic, trong đó người ta thường dựa và một số nguyên lý cơ bản như sau:

- Nguyên lý vét cạn thông minh: Trong một bài toán tìm kiếm nào đó, khi không gian tìm kiếm lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra mục tiêu

- Nguyên lý tham lam (Greedy): lấy tiêu chuẩn tối ưu (Trên phạm vi toàn cục) của bài toán để làm tiêu chuẩn chọn lựa hành động cho phạm vi cục bộ của từng bước (Hay từng giai đoạn) trong quá trình tìm kiếm lời giải

- Nguyên lý thứ tự: thực hiện hành động dựa trên một cấu trúc thứ tự hợp lý của không gian khảo sát nhằm nhanh chóng đạt được một lời giải tốt.

**4. Ưu điểm của thuật giải Heuristic:**

Thuật giải Heuristic thể hiện cách giải bài toán với các đặc tính sau:

- Thường tìm được lời giải tốt (Nhưng không chắc là lời giải tốt nhất).

- Giải bài toán theo thuật giải Heuristic thường dễ dàng và nhanh chóng đưa ra kết quả hơn so với giải thuật tối ưu, vì vậy chi phí thấp hơn.

- Thuật giải Heuristic thường thể hiện khá tự nhiên, gần gũi với cách suy nghĩ và hành động con người

**5.Ứng dụng kinh nghiệm cho bài toán cụ thể:**

Bài toán tìm đường đi ngắn nhất.

## **II.** **LÝ THUYẾT VỀ NGÔN NGỮ SỬ DỤNG (C++):**

### **Tổng quan về ngôn ngữ C++:**

C++ (đọc là "C cộng cộng" hay "xi-plus-plus", IPA: /siː pləs pləs/) là một loại ngôn ngữ lập trình bậc trung (middle-level). Đây là ngôn ngữ lập trình đa năng được tạo ra bởi Bjarne Stroustrup như một phần mở rộng của ngôn ngữ lập trình C, hoặc "C với các lớp Class", Ngôn ngữ đã được mở rộng đáng kể theo thời gian và C ++ hiện đại có các tính năng: lập trình tổng quát, lập trình hướng đối tượng, lập trình thủ tục, ngôn ngữ đa mẫu hình tự do có kiểu tĩnh, dữ liệu trừu tượng, và lập trình đa hình, ngoài ra còn có thêm các tính năng, công cụ để thao tác với bộ nhớ cấp thấp. Từ thập niên 1990, C++ đã trở thành một trong những ngôn ngữ thương mại ưa thích và phổ biến của lập trình viên

### **Cấu trúc một chương trình trong C++**

**a. Các câu lệnh**

Khai báo tên các tệp chứa những thành phần có sẵn ( các hằng chuẩn, kiểu chuẩn và hàm chuẩn) mà người sử dụng sẽ sử dụng trong chương trình.

**b. Biểu thức**

Trình biên dịch cũng có khả năng giải quyết các biểu thức. Biểu thức xác định một tính toán sẽ được thực hiện.

**Ví dụ** : tất cả chúng ta đều học rằng 2 + 3 bằng 5. Trong lập trình, chúng ta nói rằng 2 + 3 là một biểu thức đánh giá giá trị 5

**c. Hàm**

Trong C ++, các câu lệnh thường được nhóm thành các đơn vị gọi là hàm. Hàm là một tập hợp các câu lệnh thực thi tuần tự. Mỗi chương trình C ++ phải chứa một hàm đặc biệt gọi là main.Các chức năng thường được viết để làm một công việc rất cụ thể.

### **https://1.bp.blogspot.com/-Ps64vjXlhI4/Ww_tGKPemrI/AAAAAAAAA8M/EnAgitr13u0zglxtyoM5jfV0D5yqLdsKwCLcBGAs/s640/3.PNGCác kiểu dữ liệu trong C++**

### **Biến trong C++**

Trong C++, biến có thể được khai báo ở bất kì địa điểm nào trong chương trình trước khi sử dụng.

1. **Tên của một biến**

Bắt đầu bằng dấu gạch dưới “\_” hoặc chữ cái.Tên biến có thể là chữ hoa hoặc chữ thường như là a-z hoặc A-Z. Trong C phân biệt chữ hoa chữ thường.

Tên biến có thể bao gồm chữ cái, chữ số, và dấu gạch dưới.

Không bao gồm các kĩ tự đặc biệt như là !, %, ], $.

Không bao gồm khoảng trống.

Không được trùng tên với từ khóa.

Không nên dài quá 32 kí tự.

1. **Cách khai báo biến**

Khai báo không khởi tạo

Tên\_kiểu Tên\_biến;

Ví dụ:

int i, j; // khai báo 2 biên i, j kiểu nguyên

float x; // khai báo biến x kiểu float.

char c,d[100]; // biến kí tự c, xâu d chứa tối đa 100 kí tự.

Khai báo khởi tạo

Tên\_kiểu Tên\_biến = Giá\_trị;

Ví dụ:

int a = 5;

float b = 7.5;

string st = « nhom 1 »;

**CHƯƠNG 2:** **CÀI ĐẶT CHƯƠNG TRÌNH VÀ THUẬT TOÁN A\***

1. **Giới thiệu và cài đặt Dev C:**

Dev-C++ là một môi trường phát triển tích hợp tự do(IDE) được phân phối dưới hình thức giấy phép Công cộng GNU hỗ trợ việc lập trình bằng C/C++. Nó cũng nằm trong bộ trình dịch mã nguồn mở MinGW. Chương trình IDE này được viết bằng ngôn ngữ Delphi.

Dự án phát triển Dev-C++ được lưu trữ trên SourceForge. Dev-C++ nguyên được phát triển bởi một lập trình viên có tên là Colin Laplace và chỉ chạy trên hệ điều hành Microsoft Windows.

Bloodshed Dev-C++ là một Môi trường Phát triển Tích hợp (IDE) có hỗ trợ đầy đủ tính năng cho ngôn ngữ lập trình C/C++. Nó sử dụng trình MinGW của GCC (Bộ trình dịch GNU) làm trình biên dịch. Dev-C++ cũng có thể được dùng kết hợp với Cygwin hay bất kỳ trình dịch nền tảng GCC nào khác.

* Cài đặt chúng ta chỉ download về và dùng thôi.

**2. Tìm hiểu A\***

* 1. **Tìm hiểu**

A\* là giải thuật tìm kiếm trong đồ thị, tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.

Từ trạng thái hiện tại A\* xây dựng tất cả các đường đi có thể đi dùng hàm ước lược khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. A\* luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế.

A\* lưu giữ một tập các đường đi qua đồ thị, từ đỉnh bắt đầu đến đỉnh kết thúc, tập các đỉnh có thể đi tiếp được lưu trong tập Open.

A\* là một thuật toán rất thông minh, được Google Maps sử dụng trong bài toán tìm đường đi ngắn nhất thời gian thực!

Thứ tự ưu tiên cho một đường đi đươc quyết định bởi hàm Heuristic được đánh giá f(x) = g(x) + h(x)

* g(x) là chi chi phí của đường đi từ điểm xuất phát cho đến thời điểm hiện tại.
* h(x) là hàm ước lượng chi phí từ đỉnh hiện tại đến đỉnh đích f(x) thường có giá trị càng thấp thì độ ưu tiên càng cao.
  1. **Ý tưởng:**

Xét bài toán tìm đường, A\* sẽ xây dựng tăng dần các tuyến đường từ đỉnh xuất phát đến khi nó tìm thấy đường đi chạm đến đích. Để xác định khả năng dẫn đến đích, A\* sử dụng một đánh giá heuristic về khoảng cách từ một điểm bất kì cho trước đến đích. Trong ví dụ về bài toán tìm đường đi giao thông  này thì đánh giá heuristic là khoảng cách đường chim bay từ điểm cho trước đến điểm đích.

A\* đảm bảo tính đầy đủ và tối ưu, nó luôn tìm thấy đường đi ngắn nhất nếu tồn tại một đường đi như thế. Đầy đủ và tối ưu hơn các thuật toán tìm đường đi khác ở chỗ nó không chỉ ước lượng khoảng cách còn lại (nhờ đánh giá heuristic) mà còn tính khoảng cách đã đi qua để tìm được đường đi ngắn nhất.

## **3. Thuật giải A\***

1. Open: tập các trạng thái đã được sinh ra nhưng chưa được xét đến.
2. Close: tập các trạng thái đã được xét đến.
3. Cost(p, q): là khoảng cách giữa p, q.
4. g(p): khoảng cách từ trạng thái đầu đến trạng thái hiện tại p.
5. h(p): giá trị được lượng giá từ trạng thái hiện tại đến trạng thái đích.
6. f(p) = g(p) + h(p)
   * Bước 1:
     + Open: = {s}
     + Close: = {}
   * Bước 2: while (Open !={})
     + Chọn trạng thái (đỉnh) tốt nhất p trong Open (xóa p khỏi Open).
     + Nếu p là trạng thái kết thúc thì thoát.
     + Chuyển p qua Close và tạo ra các trạng thái kế tiếp q sau p.
       - Nếu q đã có trong Open
         * Nếu g(q) > g(p) + Cost(p, q)

g(q) = g(p) + Cost(p, q)

f(q) = g(q) + h(q)

prev(q) = p (đỉnh cha của q là p)

* + - * Nếu q chưa có trong Open
        + g(q) = g(p) + cost(p, q)
        + f(q) = g(q) + h(q)
        + prev(q) = p
        + Thêm q vào Open
      * Nếu q có trong Close
        + Nếu g(q) > g(p) + Cost(p, q)

Bỏ q khỏi Close

Thêm q vào Open

* + Bước 3: Không tìm được.

**4.Giải thuật lập trình:**

Dùng 2 file Input1.txt và Input2.txt để lưu các trọng số trên cây đồ thị.

File Input1.txt lưu giá trị h của mỗi node mà đề bài cho còn file Input2.txt lưu dưới dạng ma trận lưu khoảng cách giữa 2 điểm nếu giữa 2 điểm không có đoạn nối đánh 0 (tức khoảng cách giữa hai đỉnh này bằng không hoặc không có đoạn nối 2 đỉnh này)

Trong đó:

* 11 là số đỉnh
* Ma trận kề ở dưới lưu mỗi liên hệ giữa 2 đỉnh và độ dài 2 đỉnh đó trong đồ thị theo thứ tự của các đỉnh.

Sau đó tạo 1 file main.cpp lưu đoạn code dưới này và chạy chương trình. Chương trình cho kết quả thứ tự các node đi qua từ điểm bắt đầu đến điểm kết thúc.

//phần input

11

60 53 36 35 35 19 26 38 23 0 7

//input2

11

0 11 0 0 0 0 0 15 0 0 0

11 0 9 0 0 0 0 0 0 0 0

0 9 0 1 0 0 0 0 0 0 0

0 0 1 0 2 0 0 0 0 0 0

0 0 0 2 0 11 0 0 0 0 0

0 0 0 0 11 0 16 0 0 0 5

0 0 0 0 0 16 0 3 0 0 7

15 0 0 0 0 0 3 0 7 0 0

0 0 0 0 0 0 0 7 0 29 0

0 0 0 0 0 0 0 0 29 0 7

0 0 0 0 0 5 7 0 0 7 0

//phần chương trình

#include<conio.h>

#include <fstream>

#include <iostream>

using namespace std;

struct Node

{

int index; // so thu tu

int g; // khoang cach tu dinh ban dau den dinh hien ta

int f; // f = h + g;

int h; // duong di ngan nhat

int color; // danh dau dinh di qua

int parent; // dinh cha

};

int a[100][100];

Node p[100];

Node Open[100];

Node Close[100];

void ReadInputFile1(int \*b, int &n)

{

fstream fs1("Input1.txt");

if (!fs1.is\_open())

{

cout << "Khong the mo duoc file!";

}

else

{

fs1 >> n;

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

{

fs1 >> b[i];

}

}

fs1.close();

}

void ReadInputFile2(int a[100][100], int &n, int &start, int &finsh)

{

fstream fs2("Input2.txt");

if (!fs2.is\_open())

{

cout << "Khong the mo duoc file!";

}

else

{

fs2 >> n >> start >> finsh;

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

{

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

fs2 >> a[i][j];

}

}

fs2.close();

}

void RhowMatrix(int a[100][100], int n)

{

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

{

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

{

cout << a[i][j] << "\t";

}

cout << "\n";

}

}

int Count(int n, Node \*Open)

{

int count = 0;

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

{

if (Open[i].color == 1)

count++;

}

return count;

}

int Find(int n, Node \*Open)

{

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

if (Open[i].color == 1)

return i;

return -1;

}

int FindMin(int n, Node \*Open)

{

int minIndex = Find(n, Open);

int min = Open[minIndex].f;

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

{

if (Open[i].f < min && Open[i].color == 1)

{

minIndex = i;

min = Open[i].f;

}

}

return minIndex;

}

void Init(int n, int \*b)

{

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

{

p[i].index = i;

p[i].color = 0;

p[i].g = b[i];

p[i].parent = 0;

p[i].f = p[i].g;

p[i].h = 0;

}

}

int FindPoint(int n, Node \*q, int o)

{

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

if (q[i].index == o)

return i;

return -1;

}

void AStar(int a[100][100], int n, int start, int finsh, int b[])

{

int l = 0;

Open[l] = p[start];

Open[l].color = 1;

Open[l].f = Open[l].h + Open[l].g;

l++;

int w = 0;

while (Count(l, Open) != 0) // kiem tra xem tap Open co con phan tu nao khong

{

int k = FindMin(n, Open); // tim vi tri nho nhat trong Open

Open[k].color = 2; // cho diem tim duoc vao Close

Close[w] = Open[k];

Close[w].color = 2;

w++;

p[FindPoint(n, p, Open[k].index)].color = 2;

if (FindPoint(n, p, Open[k].index) == finsh)

{

cout << "Duong di qua la" << endl;

cout << finsh << "\t";

int y = FindPoint(w, Close, finsh);

int u = Close[y].parent;

while (u != start)

{

y = FindPoint(w, Close, u);

u = Close[y].parent;

cout << u << "\t";

}

break;

}

else

{

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

{

if (a[FindPoint(n, p, Open[k].index)][i] != 0 && p[i].color == 0) // neu chua co trong Open va Close

{

Open[l] = p[i];

Open[l].color = 1;

Open[l].h = a[FindPoint(n, p, Open[k].index)][i] + Open[k].h; // tinh h khoang cach ngan nhat tu dinh bat dau den dinh hien tai

Open[l].f = Open[l].g + Open[l].h;

Open[l].parent = FindPoint(n, p, Open[k].index);

p[i].color = 1;

l++;

}

if (a[FindPoint(n, p, Open[k].index)][i] != 0 && p[i].color == 1) // neu dinh da co trong Open

{

int h = FindPoint(l, Open, p[i].index);

Node tempNode = p[i];

tempNode.color = 1;

tempNode.h = a[FindPoint(n, p, Open[k].index)][i] + Open[k].h;

tempNode.parent = k;

tempNode.f = tempNode.h + tempNode.g;

if (tempNode.f < Open[h].f) // neu f trang thai hien tai be hon trang thai cap nhat truoc do

Open[h] = tempNode;

}

if (a[FindPoint(n, p, Open[k].index)][i] != 0 && p[i].color == 2) // neu dinh da co trong Close

{

int h = FindPoint(l, Close, p[i].index);

Node tempNode = p[i];

tempNode.color = 1;

tempNode.h = a[FindPoint(n, p, Open[k].index)][i] + Open[k].h;

tempNode.parent = k;

tempNode.f = tempNode.h + tempNode.g;

if (tempNode.f < Close[h].f) // neu f trang thai hien tai be hon trang thai truoc do

{

Open[l] = tempNode; // them vao Open

Close[h].color = 1; // danh dau dinh do thuoc Open

l++;

}

}

}

}

}

}

int main()

{

int n;

int start;

int finish;

int b[100];

ReadInputFile1(b, n);

ReadInputFile2(a, n, start, finish);

Init(n, b);

cout << "Dinh bat dau" << endl;

cout << start << endl;

cout << "Dinh ket thuc" << endl;

cout << finish << endl;

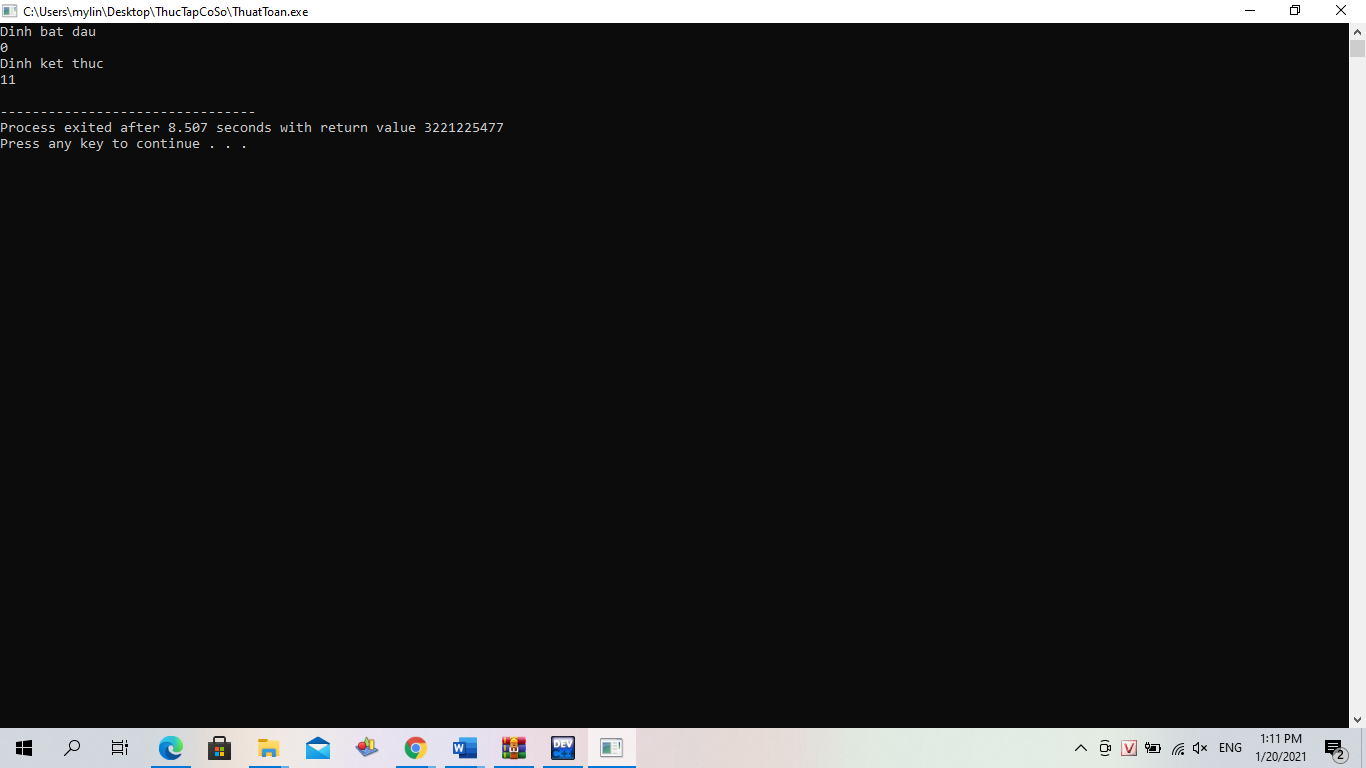
AStar(a, n, start, finish, b);

return 0;

}

**CHƯƠNG III: KẾT LUẬN**

**1. Kết quả đạt được**



**2.Mô phỏng trên đồ thị**

**A picture containing device, accessory

Description automatically generated**

h(A) = 60, h(B) = 53, h(C) = 36, h(D) = 35, h(E) = 35, h(F) = 19, h(G) = 16, h(H) = 38, h(I) = 23, h(J) = 0, h(K) = 7

* Đỉnh bắt đầu A.
* Đỉnh kết thúc K.
* Ước lượng khoảng cách từ đỉnh hiện tại cho đến đỉnh kết thúc f(x)=g(x)+h(x) trong đó g là khoảng cách ngắn nhất từ đỉnh hiện tại đến đích. Ví dụ: f(A) = 0 + 60.

**3. Nhận xét**

**Ưu điểm**

### - Thường tìm được lời giải tốt (Nhưng không chắc là lời giải tốt nhất).

- Giải bài toán theo thuật giải Heuristic thường dễ dàng và nhanh chóng đưa ra kết quả hơn so với giải thuật tối ưu, vì vậy chi phí thấp hơn.

- Thuật giải Heuristic thường thể hiện khá tự nhiên, gần gũi với cách suy nghĩ và hành động con người.

### **Nhược điểm**

A\* rất linh động nhưng vẫn gặp một khuyết điểm cơ bản - giống như chiến lược tìm kiếm chiều rộng - đó là tốn khá nhiều bộ nhớ để lưu lại những trạng thái đã đi qua

**4.KẾT LUẬN**

Heuristic không phải lúc nào cũng tìm ra giải pháp tối ưu nhất, nhưng nó có thể tìm ra một trong những giải phát tốt nhất trong thời gian ngắn với không gian bộ nhớ nhỏ.

**TÀI LIỆU THAM KHẢO:**

**https://hvitclan.vn/blog/bai-3-cac-phuong-phap-tim-kiem-co-su-dung-thong-tin-188?fbclid=IwAR3\_rpZI9FtPpH-CzXyldFYBDrB4ADrSWwbALIue\_GB3UHpxU1ay\_W4ejvY**

**https://bloglaptrinh2016.wordpress.com/2016/06/09/a-thuat-toan-tim-kiem-a/?fbclid=IwAR2YOy45ZtDNE4RDPP0y8s2\_-Kygwytc-nfWM4sE514rHBmZucG7UEEIuL4**

**https://labs.septeni-technology.jp/algorithm/algorithm-cac-thuat-toan-tim-kiem-trong-ai/?fbclid=IwAR0uQBhPZb2DqiRxLRvP40Q5gFtzCbR\_w\_vsalCow9xZjQfyL2F3RzSUs9Y**

**https://cuuduongthancong.com/atc/28/tri-tue-nhan-tao---cac-phuong-phap-giai-quyet-van-de-co-ban?src=list&fbclid=IwAR15B0b05EKS08YyhX5ydVkCqz2LkdvhhdaVuVAM6R-m6tt7pjdaQKXfhbY**