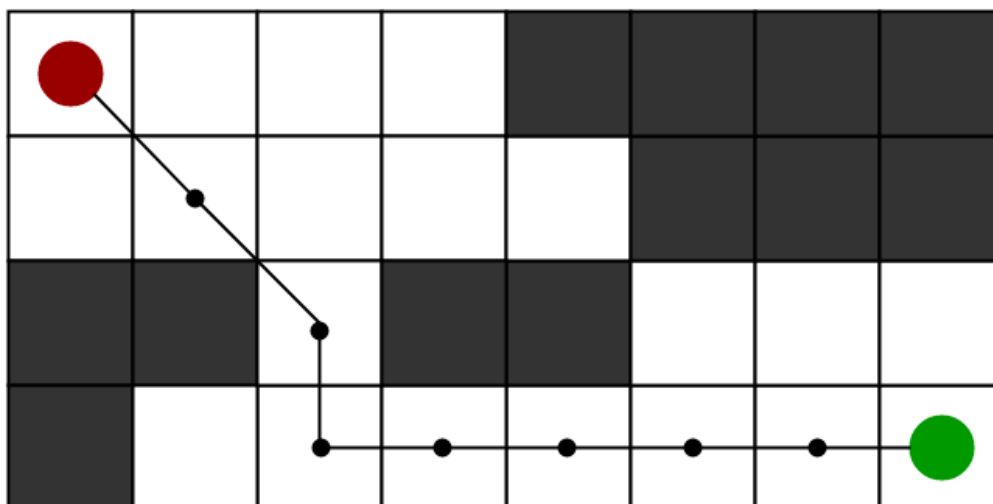




**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG TP.HCM**  
**KHOA CÔNG NGHỆ THÔNG TIN**

## **CƠ SỞ TRÍ TUỆ NHÂN TẠO**

### **PROJECT 01 – TÌM ĐƯỜNG ĐI TỐI ƯU**



**Giảng Viên: Châu Thành Đức**

**Trợ Giảng: Ngô Đình Hy, Phan Thị Phương Uyên**

TP. HCM, tháng 7 năm 2021

**Thành viên:**

STT	MSSV	Họ tên
1	19127478	Bùi Huỳnh Trung Nam (NT)
2	19127622	Ngô Trường Tuyền

**Phân công:**

Họ tên	Công việc	Tiến độ
Bùi Huỳnh Trung Nam	Read/write file bit map Hàm heuristic Testing Viết report	Hoàn thành (100%)
Ngô Trường Tuyền	A* code Hàm heuristic Testing Viết report	Hoàn thành (100%)

*“Cảm ơn sự giúp đỡ tận tình của thầy cô để nhóm có thể hoàn thành project này”*

## MUC LUC

I/ ADVERSARIAL SEARCH.....	4
1/ Định nghĩa .....	4
2/ Cấu trúc lại problem .....	5
3/ Game Tree .....	5
4/ Minimax.....	6
5/ Alpha-Beta pruning .....	7
II/ READ/WRITE FILE BITMAP .....	7
1/ Sơ lược file bitmap: .....	7
2/ Lưu trữ dữ liệu file bitmap.....	9
3/ Read Bitmap File .....	10
4/Write Bitmap File .....	11
5/ Show Bitmap Header và Information: .....	13
III/ THUẬT TOÁN A* .....	13
1/ Sơ lược heuristic, heuristic function và A*: .....	13
2/ Những biến, giá trị quan trọng hỗ trợ cho việc tìm đường: .....	13
3/ Mô tả thuật toán A*: .....	14
4/ Mã giả của thuật toán A*:.....	17
5/ Heuristic.....	18
5/1/ Heuristic 1:.....	18
5/2/ Heuristic 2:.....	19
5/3/ Heuristic 3:.....	19
IV/ HÀM MAIN (FILE MAIN.CPP).....	19
V/MỞ RỘNG .....	20
VI/ CÁCH BIÊN DỊCH/CHẠY MÃ NGUỒN .....	20
1/ Microsoft Visual Studio 2019.....	20
2/ GCC .....	21
VII/ NGUỒN THAM KHẢO.....	21

## I/ ADVERSARIAL SEARCH

### 1/ Định nghĩa

Adversarial search là một cách tìm kiếm, chúng ta xem xét vấn đề khi có các đối thủ (enemy hay opponent) thay đổi trạng thái của vấn đề trực tiếp trong mỗi tác động, vấn đề sẽ thay đổi theo tình trạng không mong muốn (agents' goal are in conflict).

Có thể có một số tình huống trong đó nhiều hơn một tác nhân (agent) đang tìm kiếm giải pháp trong cùng một không gian tìm kiếm và tình huống này thường xảy ra khi chơi trò chơi.

Môi trường có nhiều hơn một tác nhân được gọi là môi trường đa tác nhân (multi-agents), trong đó mỗi tác nhân là đối thủ của tác nhân khác và chơi với nhau. Mỗi tác nhân cần xem xét hành động của tác nhân khác và ảnh hưởng của hành động đó đối với hoạt động của chính mình, enemy và opponent thường thay đổi trạng thái vấn đề theo hướng:

- Không đoán trước được
- Có tác động xấu đến mình

Các Tìm kiếm trong đó hai hoặc nhiều người chơi có mục tiêu mâu thuẫn nhau đang cố gắng khám phá cùng một không gian tìm kiếm cho giải pháp, được gọi là adversarial search thường được gọi là games.

Hai yếu tố chính là mô hình và giải quyết trong AI là: Game được mô hình hóa thành một bài toán tìm kiếm (search problem) và hàm đánh giá heuristic.

Các loại games của AI:

	Deterministic (xác định)	Chance move (thay đổi)
Perfect information	Chess, checker, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

Perfect information (thông tin hoàn hảo): Một trò chơi có thông tin hoàn hảo là trong đó các agent có thể nhìn vào một bảng (complete board). Các agent có tất cả thông tin về trò chơi, và nó cũng có thể nhìn thấy các bước di chuyển của nhau. Ví dụ như chess, checker, go, Othello,...

Imperfect information (thông tin không hoàn hảo): Nếu agent của trò chơi không có tất cả thông tin về trò chơi và không biết chuyện gì đang xảy ra, thì loại trò chơi đó được gọi là trò chơi có thông tin không hoàn hảo, chẳng hạn như battleships, blind, tic-tac-toe,...

Deterministic (trò chơi xác định): là những trò chơi tuân theo một khuôn mẫu và bộ quy tắc chặt chẽ cho trò chơi và không có sự ngẫu nhiên nào liên quan đến chúng. Ví dụ như chess, checkers, go, tic-tac-toe,...

Trò chơi không xác định: là những trò chơi có nhiều sự kiện không thể đoán trước và có yếu tố may rủi. Yếu tố may rủi hoặc may mắn này được đưa vào bởi xúc xắc hoặc một số thẻ bài (card). Đây là những điều ngẫu nhiên và mỗi phản hồi hành động không cố định. Những trò chơi như vậy còn được gọi là trò chơi ngẫu nhiên. Ví dụ: backgammon, monopoly, poker,...

## 2/ Cấu trúc lại problem

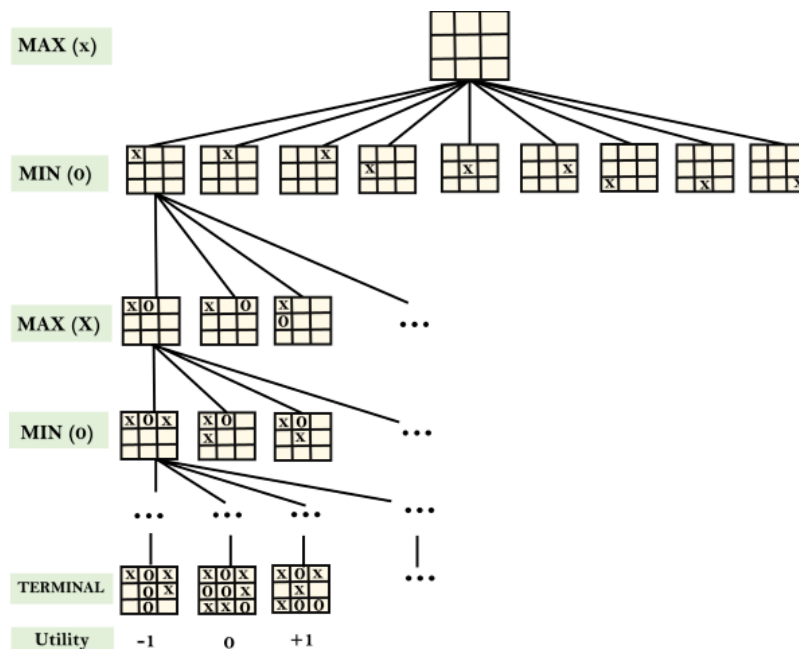
Trò chơi có thể được định nghĩa là một loại tìm kiếm trong AI có thể được chính thức hóa bằng các yếu tố sau:

- Initial state: cách trò chơi được thiết lập khi bắt đầu.
- Player(s): đưa ra người chơi nào đã di chuyển trong không gian trạng thái.
- Action(s): trả về tập hợp các bước di chuyển hợp pháp trong không gian trạng thái.
- Result(s, a): là mô hình chuyển tiếp, chỉ rõ kết quả của các chuyển động trong không gian trạng thái.
- Terminal-Test(s): kiểm tra đầu cuối là đúng nếu trò chơi kết thúc, nếu không, sai trong bất kỳ trường hợp nào. Trạng thái mà trò chơi kết thúc được gọi là trạng thái cuối.
- Utility(s, p): cung cấp giá trị số cuối cùng cho trò chơi kết thúc ở trạng thái đầu cuối s đối với người chơi p. Nó còn được gọi là hàm trả thưởng. Đối với Cờ vua, kết quả là thắng, thua hoặc hòa và giá trị kết quả của nó là +1, 0,  $\frac{1}{2}$ . Và đối với tic-tac-toe, các giá trị tiện ích là +1, -1 và 0.

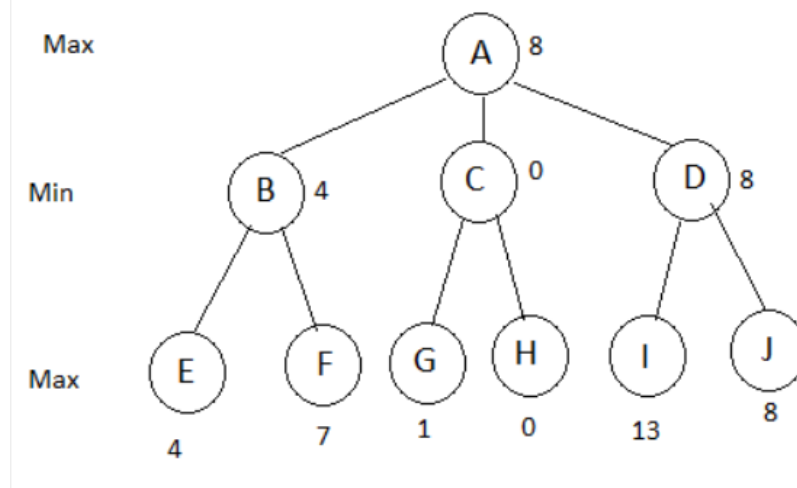
## 3/ Game Tree

Game tree là một cây trong đó các nút của cây là trạng thái trò chơi và các cạnh của cây là các bước di chuyển của người chơi. Game tree liên quan đến trạng thái ban đầu, chức năng hành động và chức năng kết quả.

- Có hai người chơi MAX và MIN.
- Người chơi có một lượt thay thế và bắt đầu với MAX.
- MAX tối đa hóa kết quả của cây trò chơi
- MIN thu nhỏ kết quả.



#### 4/ Minimax



Minimax là thuật toán đệ quy. Ở mỗi lượt chơi, máy dựa vào mảng đầu vào để “duyệt vét cạn”:

Thử lần lượt các nước đi có thể;

Với mỗi phép thử nó tiếp tục gọi vòng đệ quy mới để thử hết các lựa chọn kế tiếp của người chơi;

Với mỗi lựa chọn đó, lại tiếp tục phép thử với lượt của máy...;

=> lặp lại cho tới khi kết thúc (X thắng, hòa, O thắng).

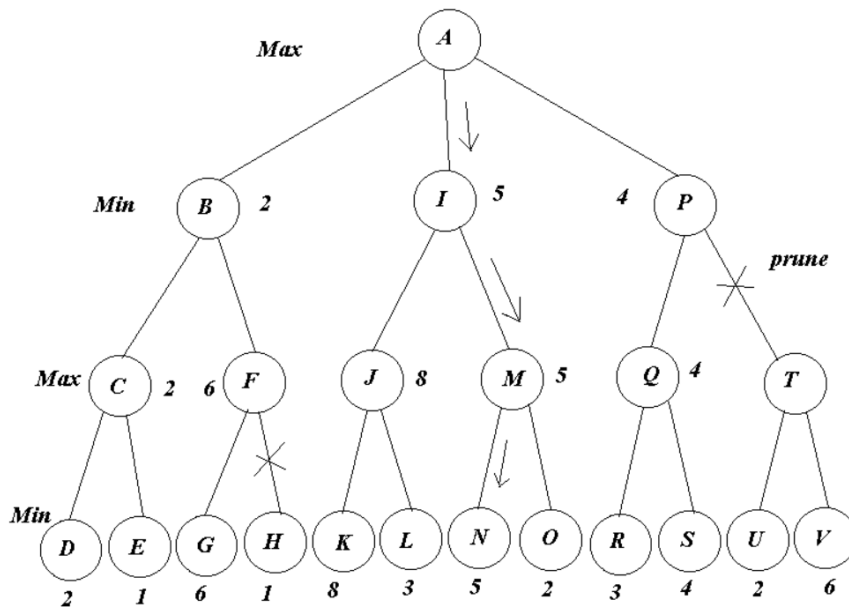
Sau khi thử hết các trường hợp, máy sẽ chọn ra nước đi nào là tối ưu nhất. Bằng cách tính điểm cho mỗi bước đi.

MAX : tấn công => chọn nước đi mà máy có điểm cao nhất

MIN : phòng thủ => chọn nước đi sao cho người chơi có điểm thấp nhất.

Lựa chọn tối ưu: [tấn công + phòng thủ] Máy tấn công thì chọn nước MAX, còn phòng thủ thì chọn MIN

## 5/ Alpha-Beta pruning



Ý tưởng giải thuật tương tự như minimax nhưng, giảm được một số nhánh không cần duyệt.

Ví dụ như hình trên, ở điểm F (max) → ta đã xét điểm đầu là 6 → nếu xét tiếp giá trị tại F sẽ là  $\geq 6$  nhưng ở trên F là B (min) với 1 nhánh C là 2  $\Rightarrow$  không cần xét các nhánh còn lại của F vì ta sẽ lấy C(2) vì  $C(2) < F(\geq 6)$ . Tương tự như vậy đến khi hoàn thành.

## II/ READ/WRITE FILE BITMAP

### 1/ Sơ lược file bitmap:

Trong đồ họa máy vi tính, **BMP**, còn được biết đến với tên tiếng Anh khác là *Windows bitmap*, là một định dạng tập tin hình ảnh khá phổ biến. Các tập tin đồ họa lưu dưới dạng BMP thường có đuôi là **.BMP** hoặc **.DIB** (*Device Independen Bitmap*).

Các thuộc tính tiêu biểu của một tập tin ảnh BMP (cũng như file ảnh nói chung) là:

- Số bit trên mỗi điểm ảnh (*bit per pixel*), thường được ký hiệu bởi **n**. Một ảnh BMP n-bit có  $2^n$  màu. Giá trị n càng lớn thì ảnh càng có nhiều màu, và càng rõ nét hơn. Giá trị tiêu biểu của n là 1 (ảnh đen trắng), 4 (ảnh 16 màu), 8 (ảnh 256 màu), 16 (ảnh 65536 màu) và 24 (ảnh 16 triệu màu) và 32 (4 tỷ màu).
- Chiều cao của ảnh (*height*), cho bởi điểm ảnh (*pixel*).
- Chiều rộng của ảnh (*width*), cho bởi điểm ảnh (*pixel*).

Cấu trúc của 1 file bitmap gồm 4 phần:

1	Bitmap Header (14 bytes): giúp nhận dạng tập tin bitmap.
2	Bitmap Information (40 bytes): lưu một số thông tin chi tiết giúp hiển thị ảnh
3	Color Palette (4*x bytes), x là số màu của ảnh: định nghĩa các màu sẽ được sử dụng trong ảnh.
4	Bitmap Data: lưu dữ liệu ảnh.

Tuy nhiên, ở project này, file bitmap được cung cấp là “map.bmp” lại có 1 số điểm khác biệt.

- Số bit per pixel  $\leq 8 \Rightarrow$  file bitmap sẽ không có Color Palette
  - Số bit per pixel khác 24  $\Rightarrow$  file bitmap sẽ không có padding (lề) trong ảnh này
  - Tổng số bytes của phần bitmap information là 108 chứ không phải 40 bytes (thêm 17 trường so với file có 40 bytes)
- $\Rightarrow$  File bitmap này thuộc version 4 (Microsoft windows 95)

Cấu trúc của file “map.bmp” là:

1	Bitmap Header (14 bytes): giúp nhận dạng tập tin bitmap.
2	Bitmap Information (108 bytes): lưu một số thông tin chi tiết giúp hiển thị ảnh
3	Bitmap Data: lưu dữ liệu ảnh.

Chi tiết các phần của file bitmap:

- Header

```
DWORD Size;      /* Size of this header in bytes */
SHORT Width;     /* Image width in pixels */
SHORT Height;    /* Image height in pixels */
WORD Planes;     /* Number of color planes */
WORD BitsPerPixel; /* Number of bits per pixel */
```

- Information:

```
DWORD Size;      /* Size of this header in bytes */
LONG Width;     /* Image width in pixels */
LONG Height;    /* Image height in pixels */
WORD Planes;     /* Number of color planes */
WORD BitsPerPixel; /* Number of bits per pixel */
DWORD Compression; /* Compression methods used */
DWORD SizeOfBitmap; /* Size of bitmap in bytes */
LONG HorzResolution; /* Horizontal resolution in pixels per meter */
LONG VertResolution; /* Vertical resolution in pixels per meter */
DWORD ColorsUsed; /* Number of colors in the image */
DWORD ColorsImportant; /* Minimum number of important colors */
/* Fields added for Windows 4.x follow this line */
```



```

DWORD RedMask;    /* Mask identifying bits of red component */
DWORD GreenMask;  /* Mask identifying bits of green component */
DWORD BlueMask;   /* Mask identifying bits of blue component */
DWORD AlphaMask;  /* Mask identifying bits of alpha component */
DWORD CSType;     /* Color space type */
LONG RedX;        /* X coordinate of red endpoint */
LONG RedY;        /* Y coordinate of red endpoint */
LONG RedZ;        /* Z coordinate of red endpoint */
LONG GreenX;      /* X coordinate of green endpoint */
LONG GreenY;      /* Y coordinate of green endpoint */
LONG GreenZ;      /* Z coordinate of green endpoint */
LONG BlueX;       /* X coordinate of blue endpoint */
LONG BlueY;       /* Y coordinate of blue endpoint */
LONG BlueZ;       /* Z coordinate of blue endpoint */
DWORD GammaRed;   /* Gamma red coordinate scale value */
DWORD GammaGreen; /* Gamma green coordinate scale value */
DWORD GammaBlue;  /* Gamma blue coordinate scale value */

```

- Data:

Data sẽ có số điểm ảnh (width\*height), mỗi điểm ảnh sẽ được lưu trữ bởi 1 số bit, ở đây là 32 bit tương ứng thành cặp (8-8-8-8 là blue-green-red-alpha). Trong đó thì sự kết hợp của 3 màu red, green, blue sẽ tạo nên màu sắc ảnh, alpha là chỉ số trong suốt của ảnh.

Trong file “map.bmp” là kỹ thuật ảnh grayscale → với mỗi điểm ảnh sẽ có red = green = blue và alpha luôn là 255 (0xFF)

## 2/ Lưu trữ dữ liệu file bitmap

Dùng struct để lưu trữ 2 phần bitmap header và bitmap information theo nguyên tắc:

- LONG → dùng kiểu long lưu
- WORD → dùng kiểu unsigned short lưu
- DWORD → dùng kiểu unsigned int lưu

Ví dụ:

```
struct BITMAPFILEHEADER
{
    //unsigned short bfType;
    //decimal is 19778, otherwise it is
    unsigned int    fileSize;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int    startingOffsetData;
};
```

Tương tự cho bitmap information

### 3/ Read Bitmap File

Tạo 2 biến toàn cục là: BITMAPFILEHEADER bitmapHeader và BITMAPINFOHEADER bitmapInfor để lưu bitmap header và information

Ở mỗi điểm ảnh ở phần data, t phải đọc theo block (1 block = 32 bit), nếu ta dùng cách đọc file theo C là fread thì sẽ có thuộc tính size (kích cỡ của mỗi phần tử được đọc) thì sẽ dễ dàng hơn trong lúc thao tác với data:

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
```

- **ptr** – Đây là con trỏ tới một khối bộ nhớ với kích cỡ tối thiểu là size\*nmemb byte.
- **size** – Đây là kích cỡ (giá trị byte) của mỗi phần tử được đọc.
- **nmemb** – Đây là số phần tử, với mỗi phần tử có kích cỡ là size byte.
- **stream** – Đây là con trỏ tới một đối tượng FILE mà xác định một Input Stream.

Tạo hàm readBMP để đọc file bitmap để đọc và lưu bitmap header và information vào 2 biến cục bộ ở trên và data thì lưu trong 1 mảng 2 chiều width\*height.

Cách hoạt động của hàm readBMP:

- Tạo con trỏ đọc file (FILE \*fp) với kiểu đọc binary và mở file “map.bmp”
- Đọc 2 ký tự mở đầu của file bitmap → nếu đó là “BM” → tiếp tục, còn không thì dừng hàm
- Nếu đó là file bitmap, đọc và hiển thị dữ liệu của bitmap header và information bằng fread

```
unsigned short fileType;
fread(&fileType, 1, sizeof(unsigned short), fp);
if (fileType == 0x4d42) //0x4d42 = "BM"
{
    std::cout << "Opening file to read completely!" << std::endl;
    fread(&bitmapHeader, 1, sizeof(BITMAPFILEHEADER), fp);
    showBmpHead();
    fread(&bitmapInfor, 1, sizeof(BITMAPINFOHEADER), fp);
    showBmpInfoHead();
}
```

- Tiếp theo xác định dài, rộng của ảnh thông qua các thông tin vừa đọc được → tạo 1 mảng 2 chiều width\*height và 1 mảng 1 chiều kích thước 4\*width (vì 1 block có 4 phần tử → 4\*width)
- Cho chạy for bằng với height của ảnh, với mỗi lần chạy sẽ đọc dữ liệu của 1 dòng trong phần data của ảnh → lưu vào mảng 1 chiều, sau đó lấy 1 trong 3 giá trị read – green – blue gán vào mảng 2 chiều (red = blue = green)

```

for (int i = 0; i < bitmapInfor.height; i++)
{
    int count = 0;
    fread(data, sizeof(unsigned char), row_padded, fp);
    for (int j = 0; j < bitmapInfor.width * 4; j += 4)
    {
        mapPixel[i][count] = (int)data[j];
        count++;
    }
}

```

- Sau khi kết thúc vòng lặp là ta đã có được cả 3 thành phần của file bitmap được đọc, return mảng 2 chiều và đóng con trỏ file → kết thúc hàm

Kết quả:

Opening file to read completely!

```

=====FILE IDENTIFIER=====

BMP file size: 1024 byte
Reserved words (1): 0
Reserved words (2): 0
Offset bytes starting data: 122

=====BIT MAP INFORMATION HEADER=====

The size of Info header: 108
Bitmap width: 512
Bitmap height: 512
The number of planes of the image: 1
Number of bits per pixel: 32
Compression method: 3
Image size: 1048576
Horizontal resolution: 3818
Vertical resolution: 3818
Number of colors used: 0
Number of important colors: 0

```

```

*****Bit Map version 4 has been added 17 extra fields*****

Red Mask: 16711680
GreenMask: 65280
BlueMask: 255
AlphaMask: 4278190080
CSType: 1934772034
RedX: 0
RedY: 0
RedZ: 1073741824
GreenX: 0
GreenY: 0
GreenZ: 1073741824
BlueX: 0
BlueY: 0
BlueZ: 1073741824
GammaRed: 0
GammaGreen: 0
GammaBlue: 0

```

## 4/Write Bitmap File

Tạo hàm writeBMP với tham số đầu vào là mảng 2 chiều (map) (kiểu vector) và có sẵn 2 biến toàn cục bên trên, ta sẽ ghi 3 thành phần này vào file bitmap mới.

Cách hoạt động của hàm readBMP:

- Tạo 2 mảng 1 chiều chứa số byte của bitmap header và information, mỗi byte ban đầu là 0, riêng byte 2 byte đầu của bitmap header là ký hiệu “B” và “M” để nhận diện file

```

unsigned char header[14] = {
    'B', 'M',
    0, 0, 0, 0, // size in bytes
    0, 0, // app data
    0, 0, // app data
    0, 0, 0, 0 // start of data offset
};

```

Tương tự cho bitmap information

- Tạo hàm convertToLittleEndian sẽ chuyển các giá trị về kiểu lưu trữ little endian với tham số nhận vào là mảng đang xét, vị trí bắt đầu, tổng byte cần dịch (ví dụ thành phần kiểu WORD sẽ cần dịch 2 byte) và cuối cùng là giá trị để gán vào và dịch
- Với mỗi thành phần của bitmap header và information, gán cho nó bằng giá trị có sẵn trong biến cục bộ và dùng hàm convertToLittleEndian → sau cùng có được 2 mảng chứa toàn bộ thông tin, sẵn sàng ghi vào file bitmap mới.

```
// -----File Header
convertToLittleEndian(header, 2, 4, bitmapHeader.fileSize);
convertToLittleEndian(header, 6, 2, bitmapHeader.reserved1);
convertToLittleEndian(header, 8, 2, bitmapHeader.reserved2);
convertToLittleEndian(header, 10, 4, bitmapHeader.startingOffsetData);
```

Tương tự cho bitmap information

- Ở đây không có ràng buộc gì nên sẽ write vào file mới bằng cách ghi file binary trong C++ thay vì C như read. Tạo ofstream và bắt đầu ghi bitmap header và information:

```
std::ofstream stream(nameFile, std::ios::binary);
stream.write((char*)header, sizeof(header));
stream.write((char*)info, sizeof(info));
```





- Sau cùng, bắt đầu ghi data, cách ghi data là chạy 2 dòng for duyệt hết mảng 2 chiều, với mỗi phần tử ta sẽ tạo 1 mảng 4 phần tử tương ứng blue-green-red-alpha, và gán red = blue = green = map[i][j] (vì trong grayscale red = blue = green) và alpha = 255, sau đó ghi điểm ảnh này vào file bitmap mới.

Chú ý: có 1 số phần tử trong mảng 2 chiều sẽ có giá trị -1 (thể hiện đường đi, sẽ giải thích sau) nên khi gặp 1 điểm là -1 thì blue = green = 0, red = 255 và alpha vẫn bằng 255 (mục đích để điểm ảnh này có màu đỏ)

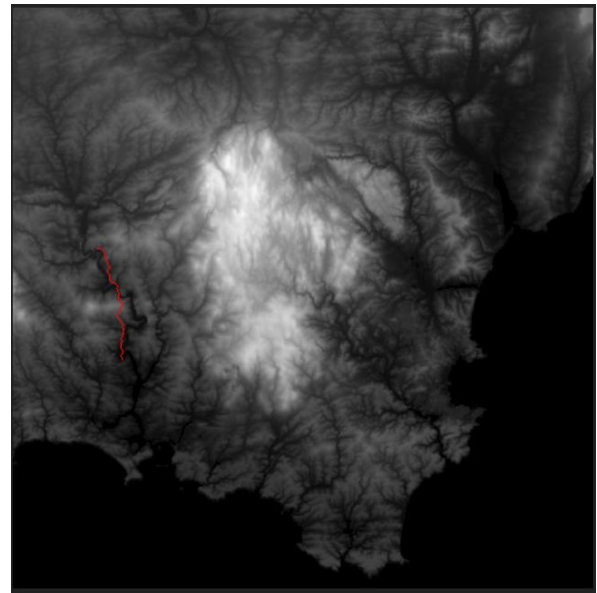
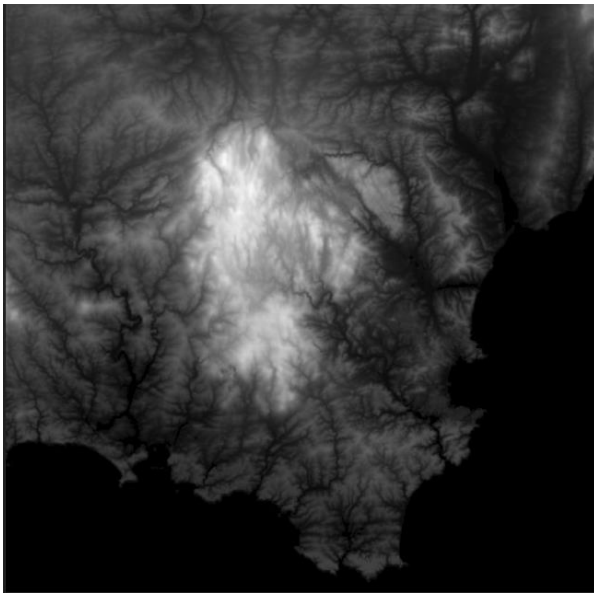
```
for (int i = 0; i < bitmapInfor.height; i++)
{
    for (int j = 0; j < bitmapInfor.width; j++)
    {
        unsigned char pixel[4]; //blue-green-red-alpha
        if (map[i][j] == -1) {
            pixel[0] = 0;
            pixel[1] = 0;
            pixel[2] = 255;
            pixel[3] = 255;
        }
        else {
            pixel[0] = map[i][j];
            pixel[1] = map[i][j];
            pixel[2] = map[i][j];
            pixel[3] = 255;
        }
        stream.write((char*)pixel, 4);
    }
}
```

- Đóng file và kết thúc hàm

Kết quả:

 map.bmp	7/5/2021 7:28 PM	BMP File	1,025 KB
 map1.bmp	7/18/2021 4:06 PM	BMP File	1,025 KB
 map2.bmp	7/18/2021 4:06 PM	BMP File	1,025 KB
 map3.bmp	7/18/2021 4:06 PM	BMP File	1,025 KB

File bitmap mới (map1/map2/map3.bmp) có kích thước hoàn toàn khớp với file cũ (1025 KB) và có thêm đường đi màu đỏ



### 5/ Show Bitmap Header và Information:

Tạo 2 hàm void, mỗi hàm sẽ in ra các thành phần của bitmap header và information thông qua 2 biến toàn cục (đã nói ở II/3)

## III/ THUẬT TOÁN A\*

### 1/ Sơ liệu heuristic, heuristic function và A\*:

- 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.
- Heuristic function là 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.
- Thuật toán A\* cùng với Greedy best-first search là những thuật toán tìm kiếm có hiểu biết (informed search).
- A\* là thuật toán cải thiện hiệu năng từ thuật toán Greedy best-first search (informed search) + UCS (uninformed search). Khi Greedy best-first search chỉ tìm đường dựa trên hàm ước lượng (heuristic function) từ điểm kế tiếp đến đích mà không tính độ dài quãng đường mà nó đã đi vì có thể ước lượng là nhỏ nhưng đường đã đi lại quá lớn.

UCS: Tìm đường dựa trên chi phí thực tế  $g(n)$ .

Greedy best-first search: Tìm đường dựa trên hàm đánh giá  $f(n)$ .

A\*: Tìm đường dựa trên hàm  $f(n) = g(n) + h(n)$ .

### 2/ Những biến, giá trị quan trọng hỗ trợ cho việc tìm đường:

Để tìm đường cần biết vị trí bắt đầu và vị trí đích. Ở đây nó là 1 điểm trong tọa độ Oxy. Mỗi điểm này sẽ có tọa độ x là tọa độ nằm trên trục ngang, y là tọa độ nằm trên trục đứng.

G là khoảng cách thực tế từ điểm bắt đầu đến điểm đang xét được tính bằng công thức theo đề cho:

$$G = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \left(\frac{1}{2} * \text{sign}(\Delta a) + 1\right) * |\Delta a|$$

Trong đó:

$$\Delta a = a(x_1, y_1) - a(x_2, y_2)$$

$a(x, y)$  là độ cao tại điểm x, y trên bản đồ.

Sign(number) function là hàm dấu. Nếu input vào là 0 thì sẽ trả ra 0, nếu là số dương thì sẽ trả ra 1 và nếu là số âm sẽ trả ra -1.

H là khoảng cách ước lượng được tính bằng hàm heuristic function. Các heuristic function được sử dụng trong project này sẽ được đề cập ở III. 4 bên dưới.

F là tổng của G và H.

```
struct Point {
    //Point coordinates, here is calculated according to C++ array for convenience,
    //x represents horizontal row, y represents vertical column
    int x, y;
    double F, G, H; //F = G + H
    Point* parent; //The coordinates of parent, no pointer is used here, thus simplifying the code
    Point(int _x, int _y) : x(_x), y(_y), F(0), G(0), H(0), parent(NULL) {} // variable initialization
};
```

\_map là ma trận biểu diễn dữ liệu thực tế có thể là 1 bản đồ địa hình lưu độ cao hoặc là một mê cung.

\_openList tạm gọi là tập mở để lưu lại những điểm xung quanh điểm đang xét mà có thể đi đến được (ở đây theo yêu cầu đề bài là  $|\Delta a| = |a(x_1, y_1) - a(x_2, y_2)| \leq m$ , m là khoảng cách có thể đi được giữa vị trí (độ cao))

\_closeList tạm gọi là tập đóng để lưu những điểm đã đi qua. Những điểm đã đi qua rồi chúng ta sẽ không đi lại nữa.

\_numOfInteractions lưu lại số điểm đã tương tác trong quá trình tìm đường.

```
std::vector<std::vector<int>> _map; //Matrix
std::list<Point*> _openList; //Open the list
std::list<Point*> _closeList; //Close the list
int _numOfInteractions = 0; //Initially does not interaction
```

### 3/ Mô tả thuật toán A\*:

- Đầu tiên chúng ta sẽ thêm điểm bắt đầu vào \_openList để bắt đầu quá trình tìm đường.

```
//Place the starting point, copy and open a node, isolate inside and outside
_openList.push_back(new Point(startPoint.x, startPoint.y));
```

- Kiểm tra nếu \_openList không rỗng thì sẽ thực hiện các bước sau:

- Lấy điểm có  $f(n)$  nhỏ nhất trong `_openList` thông qua hàm `getLeastFPoint()`. Sau đó xóa điểm đó trong `_openList` (điểm đã đi qua sẽ không xét lại) và thêm nó vào `_closeList`.

```
auto curPoint = getLeastFPoint(); //Find the point with the smallest F value
_openList.remove(curPoint); //Remove from the open list
_closeList.push_back(curPoint); //Put into the close list
```

Giả sử điểm đầu tiên có  $F$  nhỏ nhất, duyệt qua tất cả các điểm trong `_openList` nếu có điểm nào có  $F$  nhỏ hơn hoặc bằng  $F$  tại điểm đó thì cập nhật lại điểm có  $F$  lớn nhất.

```
Point* Astar::getLeastFPoint() {
    if (!_openList.empty()) {
        auto resPoint = _openList.front();
        for (auto& point : _openList) {
            if (point->F < resPoint->F
                || ((point->F - resPoint->F) < EPSILON && (resPoint->F - point->F) < EPSILON)) {
                resPoint = point;
            }
        }
        return resPoint;
    }
    return NULL;
}
```

- Duyệt qua 8 điểm xung quanh điểm đang xét bằng hàm `getSurroundPoints(...)` và lấy những điểm thỏa yêu cầu thông qua hàm `isCanReach(...)`. Biến `values` được truyền vào hàm `getSurroundPoints` là một vector dùng để lưu lại giá trị được xử lý từ vị trí của điểm đã đi qua nhằm phục vụ cho việc tính số điểm đã tương tác, vì chỉ cần điểm đó nằm trong `_map` và là 1 trong 8 điểm xung quanh điểm đang xét thì sẽ tính vào điểm tương tác nên không thể dùng tập `_openList` và `_closeList` để tính.

```
//1, find the grid that can be passed among the eight grids around
auto surroundPoints = getSurroundPoints(curPoint, limitedDistance, values);
```

```
std::vector<Point*> Astar::getSurroundPoints(const Point* point, int limitedDistance, std::vector<int>& values) {
    std::vector<Point*> surroundPoints;

    for (int x = point->x - 1; x <= point->x + 1; x++)
        for (int y = point->y - 1; y <= point->y + 1; y++)
            if (isCanReach(point, new Point(x, y), limitedDistance, values))
                surroundPoints.push_back(new Point(x, y));

    return surroundPoints;
}
```

Bên trong hàm `isCanReach(...)`: để xét 1 điểm lân cận tạm gọi là target có đi được hay không.

Nếu điểm target này nằm ngoài `_map` hoặc là điểm đã đi đến và đang tìm điểm lân cận hoặc nằm trong tập `_closeList`, tức là đã đi qua rồi thì sẽ return false, không xét nó nữa và đương nhiên sẽ không tính nó 1 lần nữa vào số điểm tương tác.



```
//If the point coincides with the current node, exceeds the map, or is in the closed list then returns false
if (target->y < 0 || target->y > _map.size() - 1
    || target->x < 0 || target->x > _map[0].size() - 1
    || target->x == point->x && target->y == point->y
    || isInList(_closeList, target)) {
    return false;
}
```

Nếu điểm target này nằm bên trong \_map và không thỏa yêu cầu của đề là  $|x_1 - x_2| \leq 1$ ,  $|y_1 - y_2| \leq 1$  và  $|\Delta a| \leq m$  thì sẽ tính vị trí tương đối của nó trên \_map bằng công thức: vị trí tương đối = max(số dòng, số cột) \* tọa độ theo Oy + tọa độ theo Ox.

Nếu nó target chưa có trong vector values thì sẽ thêm nó vào và tăng số điểm tương tác lên 1. Và return về giá trị false, không đi đến nó được.

Sở dĩ lấy max(số dòng, số cột) của \_map là vì giá trị max này sẽ đảm bảo là vị trí tương đối của 1 điểm trong \_map này là duy nhất trong vector values. Và tính theo cách này cũng rất tối ưu vì chỉ cần 1 vector với số lượng phần tử là số điểm đã tương tác rồi, không gây dư thừa, tốn bộ nhớ nhiều.

```
//If the point is unsatisfied conditions,
//then it is checked whether it have interacted, if the answer is true(have interacted), do nothing
//otherwise add it to the values vector, increase number of interaction by 1
//return false
if (abs(target->x - point->x) > 1 || abs(target->y - point->y) > 1
    || abs(_map[point->x][point->y] - _map[target->x][target->y]) > limitedDistance) {
    int value = ((_map.size() > _map[0].size() ? _map.size() : _map[0].size()) * target->y) + target->x;

    if (!checkValueExists(values, value)) {
        values.push_back(value);
        _numOfInteractions++;
    }

    return false;
}
```

Nếu điểm target vượt qua 2 trường hợp trên thì tương tự ta sẽ kiểm tra target có nằm trong vector values chưa, nếu chưa sẽ thêm vào và tăng 1 đơn vị. Sau đó return true, tức là điểm này là điểm đi được.

- Sau khi có được các điểm lân cận, sẽ duyệt qua tất cả các điểm đó và tính giá trị G, H, F.

Nếu điểm đó không nằm trong tập \_openList thì sẽ gán điểm parent (cha) của nó là điểm hiện tại đã đi tới và đang tìm các điểm lân cận (để phục vụ cho việc truy xuất ngược lại đường đi), tính và cập nhật lại các giá trị G, H, F, sau đó thêm nó vào tập mở (điểm này có thể đi đến được).

```
if (!isInList(_openList, target)) {
    target->parent = curPoint;

    target->G = calcG(curPoint, target);
    target->H = calcH(target, &endPoint, type);
    target->F = calcF(target);

    _openList.push_back(target);
}
```

Ngược lại điểm đó đã nằm trong tập \_openList thì sẽ gán tính lại giá trị G (giá trị H không cần tính lại vì đây không phải là điểm mới), nếu giá trị G mới nhỏ hơn G cũ thì sẽ cập nhật lại G, F và cập nhật lại điểm parent của.



```

else {
    float tempG = calcG(curPoint, target);
    if (tempG < target->G) {
        target->parent = curPoint;
        target->G = tempG;
        target->F = calcF(target);
    }
}

```

Cuối cùng là kiểm tra xem điểm lân cận đó có phải là điểm đích hay không nếu phải sẽ trả về điểm đích luôn mà không cần làm gì nữa.

```

Point* resPoint = isInList(_openList, &endPoint);
if (resPoint) {
    //Return the node pointer in the list,
    //do not use the original endpoint pointer passed in,
    //because a deep copy has occurred
    return resPoint;
}

```

- Chạy cho đến khi tập \_openList là rỗng mà vẫn chưa thấy điểm đích thì sẽ trả về NULL tức là không có đường đi
- Sau khi đi đến được điểm đích, chúng ta sẽ lưu đường đi vào danh sách path, bằng cách lấy lần lượt điểm parent bắt đầu từ điểm đích thêm vào đầu danh sách path.

```

Point* result = findPath(startPoint, endPoint, limitedDistance, type);
std::list<Point*> path;

//Return path, if no path is found, return empty list
while (result) {
    path.push_front(result);
    result = result->parent;
}

```

- Một việc quan trọng nữa là sẽ cần phải xóa 2 tập \_openList và \_closeList để giải phóng bộ nhớ.
- Sau khi đi được đến điểm đích, ta sẽ có được G của điểm đích, đó cũng chính là tổng quãng đường đã đi.

#### 4/ Mã giả của thuật toán A\*:

*function findPath(startPoint, endPoint, limitedDistance) return a point*

*add startPoint to \_openList*

*while \_openList is not empty*

*curPoint = get the point has lowest f(n) value in \_openList*

*remove curPoint from \_openList*

*add curPoint to \_closeList*

*surroundPoints = all the adjacent points*

*for each target in surroundPoints*

*if target is not in \_openList then*

```

    target->parent = curPoint
    target->G = calculate G value of target point
    target->H = calculate H value of target point
    target->F = calculate F value of target point
    add target to _openList
else
    tempG = calculate G values of target point
    if tempG < target->G then
        target->parent = curPoint
        target->G = tempG
        target->F = calculate F value of target point

    resPoint = check whether end point is in _openList
    if resPoint is in _openList then return resPoint (found end point, stop
algorithm)

    return NULL (did not find end point, stop algorithm)
function getPath(startPoint, endPoint, limitedDistance) return a point
    path := an empty list
    result := findPath(startPoint, endPoint, limitedDistance)
    while result is not empty
        path.push_front(result)
        result := result->parent

```

**Chú ý:** Ngoài ra, trong code sẽ thêm 1 số tham số, mục đích là chạy hàm A\* 3 lần với mỗi lần là 1 hàm heuristic (có 3 hàm heuristic). Phần trọng tâm thuật toán A\* vẫn theo như mã giả bên trên.

## 5/ Heuristic

### 5/1/ Heuristic 1:

Hàm heuristic này được dựa trên Diagonal distance. Chi phí cho việc đi theo đường chéo được giả sử là khoảng cách cho phép có thể đi được giữa hai độ cao (limitedDistance). Chi phí cho việc đi theo phương ngang là  $0.5 * \text{limitedDistance}$ .

Chỉ khi nào đi theo đường chéo thì giá trị của biến diagonal sẽ bằng 1, còn khi đi theo phương ngang thì sẽ luôn bằng 0, vì lấy min của deltaX và deltaY, mà khi đi theo phương ngang thì hoặc

là giá trị của tọa độ X theo đổi hoặc là giá trị của tọa độ Y thay đổi, nên luôn có hoặc  $\Delta X = 0$ , hoặc  $\Delta Y = 0$ .

Tương tự, giá trị của biến straight sẽ là tổng  $\Delta X$  và  $\Delta Y$ . Khi đi theo phương ngang thì nó sẽ luôn là 1.

Heuristic này sẽ lấy chi phí đi theo đường chéo hoặc đường ngang.

Khi đi theo đường chéo  $\Delta X = 1$ ,  $\Delta Y = 1$ ,  $\Delta X + \Delta Y = 2$ ,  $\Delta X + \Delta Y - (2 * \Delta X)$  sẽ = 0 khi đó chỉ còn lại chi phí đi theo đường chéo.

Tương tự, khi đi theo phương ngang thì  $\Delta X = 0$ ,  $\Delta Y = 1$ ,  $\Delta X + \Delta Y - (2 * \Delta X) = 1$ , chỉ còn lại chi phí đi theo phương ngang.

```
double Astar::calcHByHeuristic1(Point* point, Point* end, int limitedDistance) {
    int deltaX = abs(end->x - point->x);
    int deltaY = abs(end->y - point->y);

    int diagonal = deltaX < deltaY ? deltaX : deltaY;
    int straight = deltaX + deltaY;

    return (double)((limitedDistance * diagonal) + ((limitedDistance / 2) * (straight - (2 * diagonal))));
}
```

### 5/2/ Heuristic 2:

Hàm heuristic này dựa trên Euclidean distance, với chi phí đường đi được giả sử là khoảng cách cho phép có thể đi được giữa hai độ cao (limitedDistance).

Euclidean distance giữa 2 điểm trong không gian Eculid là độ dài của đoạn thẳng nối 2 điểm đó.

$$D(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

Ở đây, sẽ nhân thêm với chi phí.

```
double Astar::calcHByHeuristic2(Point* point, Point* end, int limitedDistance) {
    return (double)(sqrt((end->x - point->x) * (end->x - point->x) + (end->y - point->y) * (end->y - point->y)) * limitedDistance);
}
```

### 5/3/ Heuristic 3:

Hàm heuristic này sẽ ước lượng chi phí đường đi, nếu đi theo phương ngang thì chi phí sẽ bằng với khoảng cách cho phép có thể đi được giữa hai độ cao (limitedDistance), nếu đi theo đường chéo thì chi phí sẽ là 1.25 lần khoảng cách cho phép có thể đi được giữa hai độ cao.

```
double Astar::calcHByHeuristic3(Point* point, Point* end, int limitedDistance) {
    int deltaX = abs(end->x - point->x);
    int deltaY = abs(end->y - point->y);

    return (double)(limitedDistance * (deltaX + deltaY)) + ((-0.75 * limitedDistance) * (deltaX < deltaY ? deltaX : deltaY));
}
```

## IV/ HÀM MAIN (FILE MAIN.CPP)

Ở hàm này ta sẽ:

- Đọc dữ liệu của file bit map và lưu vào 1 mảng 2 chiều
- Tạo 1 mảng vector 2 chiều với giá trị y như mảng 2 chiều bên trên
- Gán các giá trị cần thiết cho class Astar
- Chạy dòng for, loop 3 lần, mỗi lần là 1 hàm heuristic, sau khi có được path, ta sẽ thay đổi giá trị những ô là path thành -1 (vì hàm write bmp sẽ cho những điểm này thành màu đỏ)

nếu có giá trị -1). Với mỗi lần chạy → in ra file bmp mới và file txt. Sau mỗi lần loop, gán mảng vector lại bằng mảng 2 chiều ban đầu và tiếp tục

## V/MỞ RỘNG

### Ưu điểm:

- Tổng thể code chạy cả 3 hàm heuristic và xuất ra 3 file bitmap mới với một khoảng thời gian cho phép (khá nhanh)
- Tìm được đường đi theo thuật toán A\* và tương tác với một số điểm chấp nhận được (số điểm tương tác không quá nhiều)
- Code dễ hiểu, ngắn gọn
- Các vấn đề trong bài toàn đa số nhóm tự giải quyết

### Nhược điểm:

- Chưa có các cải tiến nhiều ở hàm heuristic so với các hàm đã có sẵn
- Với m quá nhỏ ( $m < 5$ ) thì thuật toán sẽ không tìm được đường đi do chênh lệch pixel không  $\leq m$
- Chỉ cấu trúc đọc được file bitmap version 4, nếu các file bitmap không phải version 4 (số byte của information là 40 chứ không phải 108) thì thuật toán sẽ đọc file sai và có lỗi.
- Chỉ dùng cho grayscale vì pixel = red = green = blue

### Cải tiến:

- Có thể tăng độ ưu tiên của g hoặc h tùy tình huống để thuật toán tối ưu hơn
- Ở bài này có thể chú trọng vào độ chênh lệch pixel → đường đi sẽ có xu hướng nằm trong vùng màu đen hoặc trắng (chênh lệch độ cao ít → hợp lý để di chuyển)
- Có thể kết hợp với 1 số thuật toán tìm kiếm khác để tối ưu hơn

## VI/ CÁCH BIÊN DỊCH/CHẠY MÃ NGUỒN

Mã nguồn gồm 3 file là Astart.h, Astart.cpp và main.cpp. Ngoài ra file bitmap và input.txt mặc định tên là “map.bmp” và phải đặt chung directory của các file .cpp và .h

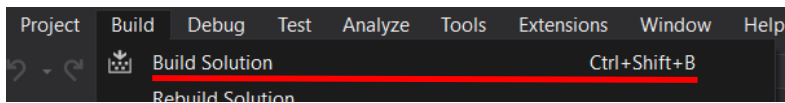
Tạo độ trong file input.txt không nên có khoảng cách vd: (xx,yy)

Astar.cpp	7/18/2021 5:15 PM	C++ Source	19 KB
Astar.h	7/18/2021 5:14 PM	C/C++ Header	7 KB
input.txt	7/18/2021 3:51 PM	Text Document	1 KB
main.cpp	7/18/2021 5:14 PM	C++ Source	2 KB
map.bmp	7/5/2021 7:28 PM	BMP File	1,025 KB

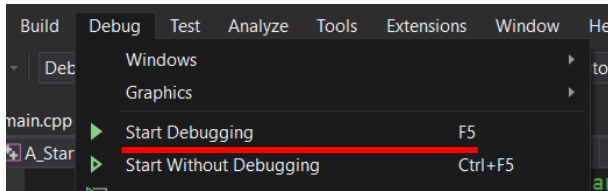
### 1/ Microsoft Visual Studio 2019

Nhóm chủ yếu code trên Microsoft Visual Studio nên việc đặt mã nguồn vào IDE này giúp cho code chạy thuận lợi hơn.

Về cách biên dịch: build solution hoặc Ctrl+Shift+B



Về cách chạy mã nguồn: Start Debugging hoặc F5



(phím tắt có thể khác tùy phiên bản)

## 2/ GCC

```
C:\Users\Admin\source\repos\A_Start Algorithm\A_Start Algorithm>g++ main.cpp Astar.cpp Astar.h
C:\Users\Admin\source\repos\A_Start Algorithm\A_Start Algorithm>a.exe
Opening file to read completely!

=====BIT MAP FILE HEADER=====

BMP file size: 1024 byte
Reserved words (1): 0
Reserved words: (2) 0
```

## VII/ NGUỒN THAM KHẢO

[https://www.tutorialspoint.com/dip/grayscale\\_to\\_rgb\\_conversion.htm](https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm)

<https://www.fileformat.info/format/bmp/egff.htm>

<http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>

~~~~~ THE END ~~~~~