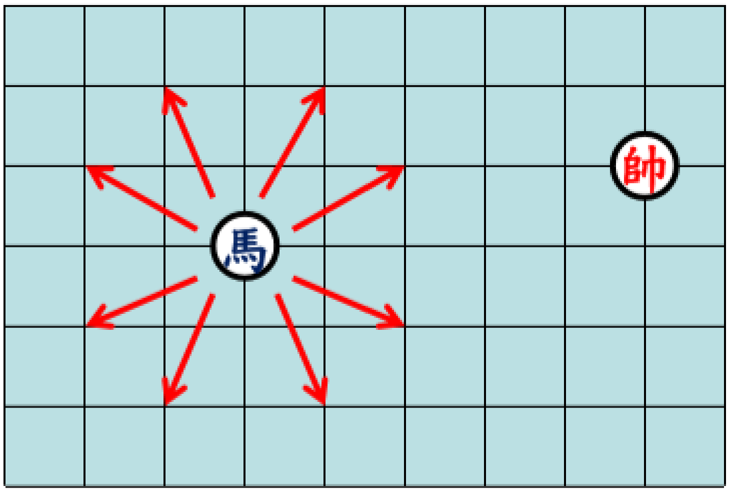
Programming Assignment #1

0611031 資工 10 謝至恆

目標:

This programming exercise concerns the move of a "knight" in a 8x8 chessboard. A knight's moves are limited to (+-1,+-2) or (+-2, +-1) from its current position. Here you need to write a program that finds the optimal (minimum number of steps) path between two given locations on the chessboard.

其實就是象棋中馬的走法，判斷要動幾步才能到達目標地點。

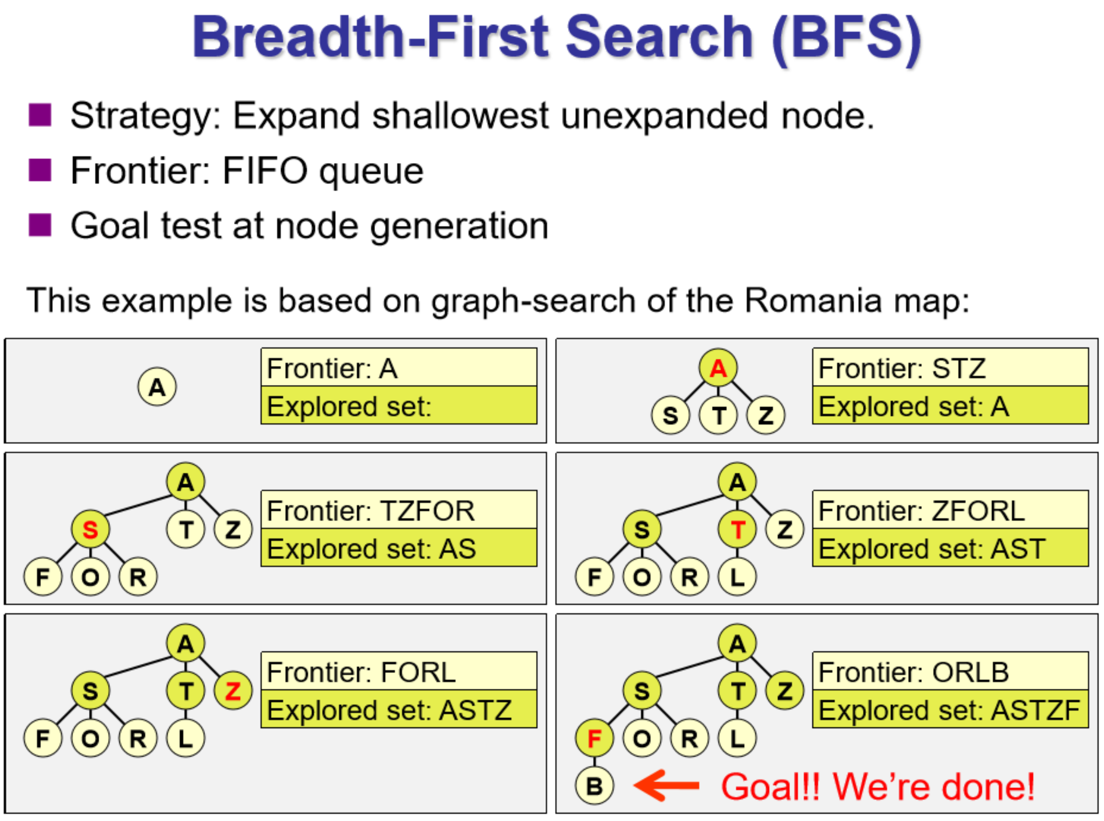


馬一共有 8 個方向可以走(在不超出邊界的情況下)

例子:

from A to B (0,0)到 (2,2) 最佳路徑 : (0,0)(2,1)(0,2)(1,0)(2,2)

使用上課學到的 BFS DFS A* IDA* 方法來達到效果。

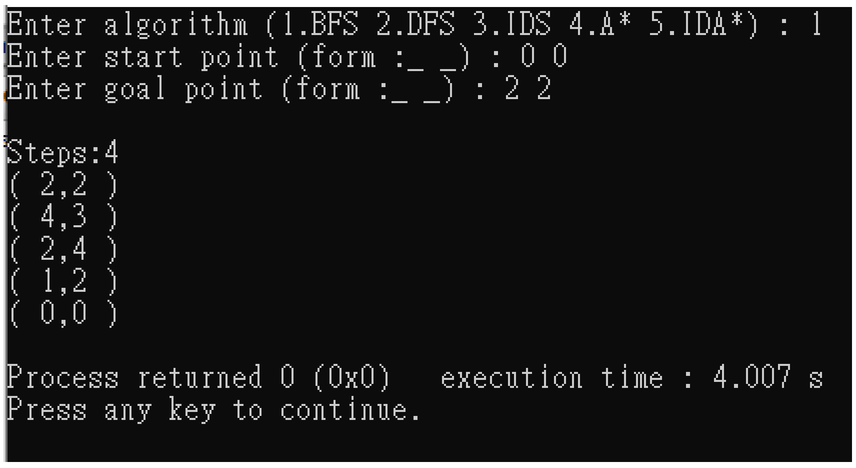

Breadth-First Search (BFS)

- Strategy: Expand shallowest unexpanded node.
- Frontier: FIFO queue
- Goal test at node generation

This example is based on graph-search of the Romania map:

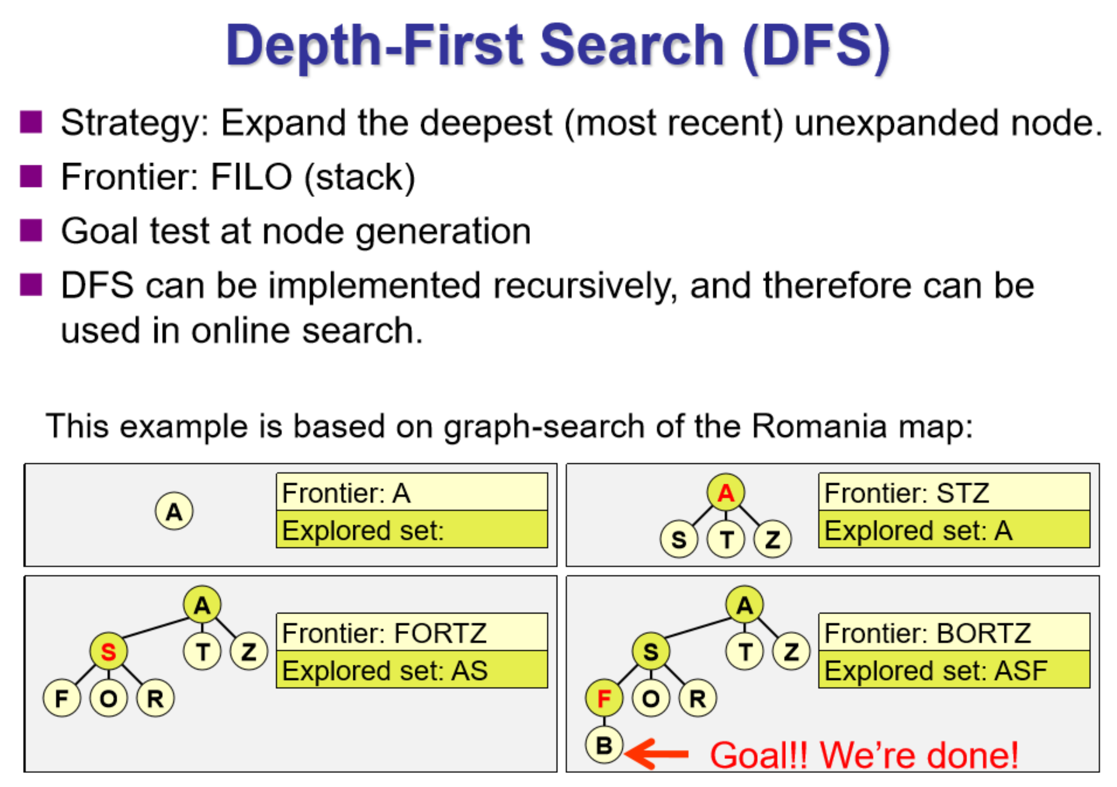
概念 ： 由淺入深 一層找完再找下一層

因為是一層一層找，所以一定會找到一個最佳解。
使用 Queue 的方式來找解，一次放一層的 queue 把 8 個方向的值做檢查，如果沒有超出棋盤範圍也沒有走到過，就 push 進 queue 裡面然後把一開始的起點 pop 掉用 while 迴圈一直找直到得到一組解為止，得到的解一定是路徑最短，但不是唯一解。可以改變 8 個方向依序的順序就可以得到不同的解。


```
Enter algorithm (1.BFS 2.DFS 3.IDS 4.A* 5.IDA*) : 1
Enter start point (form :_ _) : 0 0
Enter goal point (form :_ _) : 2 2

Steps:4
( 2,2 )
( 4,3 )
( 2,4 )
( 1,2 )
( 0,0 )

Process returned 0 (0x0)   execution time : 4.007 s
Press any key to continue.
```



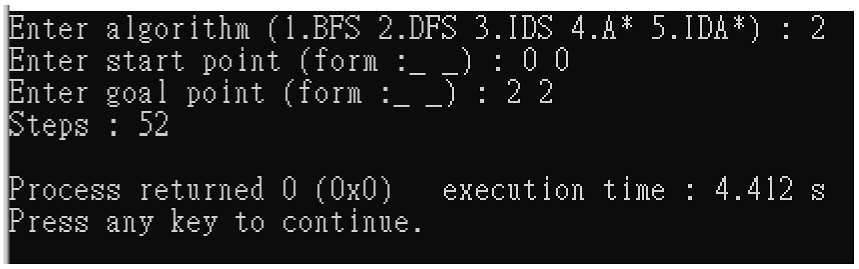
Depth-First Search (DFS)

- Strategy: Expand the deepest (most recent) unexpanded node.
- Frontier: FILO (stack)
- Goal test at node generation
- DFS can be implemented recursively, and therefore can be used in online search.

This example is based on graph-search of the Romania map:

概念 ： 一條支線找到底，沒有發現解再繼續找下一條支線。

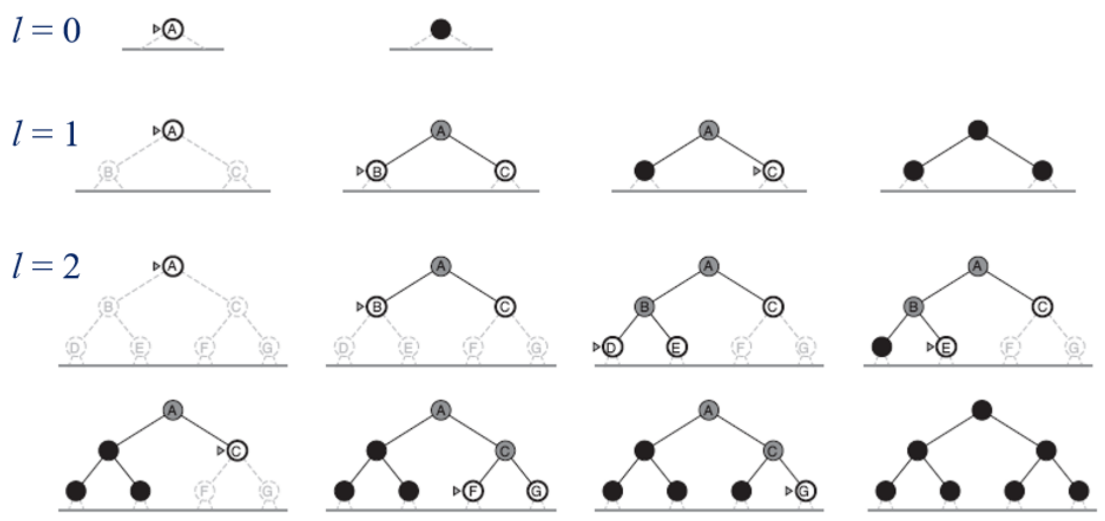
利用 BFS 得到的觀念，但是函式是用遞迴的方式來找，這樣就可以一直跑到底，看看有沒有正確解答，如果找不到就會回到上一層在換 8 個方向中還沒試過的方向繼續找，就可以實現 DFS 先往下面找的方法。如果問題有解 DFS 找的到解，但不一定是最佳解，因為他是先往深處找，有可能其他條支線有比較淺的解答，但他不會先看。



```
Enter algorithm (1.BFS 2.DFS 3.IDS 4.A* 5.IDA*) : 2
Enter start point (form :_ _) : 0 0
Enter goal point (form :_ _) : 2 2
Steps : 52

Process returned 0 (0x0)   execution time : 4.412 s
Press any key to continue.
```

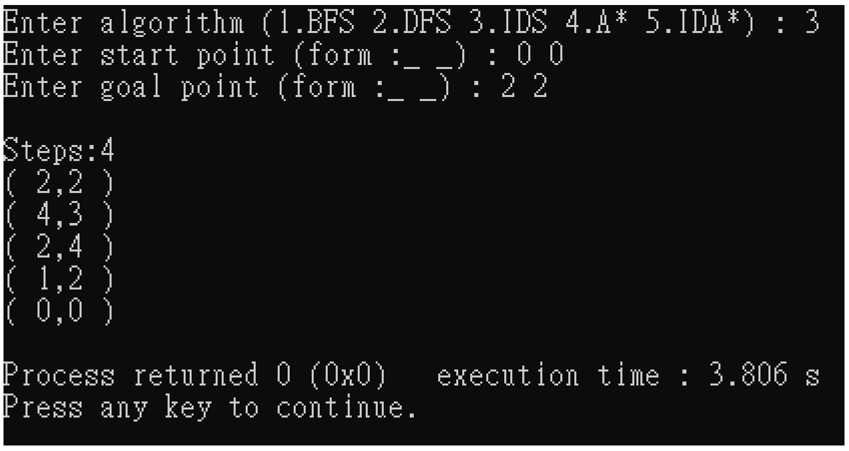
Iterative deepening search (IDS)



概念 ： 利用一層一層的 DFS 概念來達到 BFS 的效果，可以想成受限制的 DFS，每搜尋完一層後，再將深度限制加大，再繼續做下去。

逐步放寬限制深度，每次重新跑，跑到限制深度還沒找到解就放寬限制深度再重跑一次直到得解

找的方式就跟 BFS 大同小異。



```
Enter algorithm (1.BFS 2.DFS 3.IDS 4.A* 5.IDA*) : 3
Enter start point (form :_ _) : 0 0
Enter goal point (form :_ _) : 2 2

Steps:4
( 2,2 )
( 4,3 )
( 2,4 )
( 1,2 )
( 0,0 )

Process returned 0 (0x0)   execution time : 3.806 s
Press any key to continue.
```

A* search

A*搜尋演算法（A* search algorithm）是一種在圖形平面上，有多個節點的路徑，求出最低通過成本的演算法。常用於遊戲中的NPC的移動計算，或網路遊戲的BOT的移動計算上。

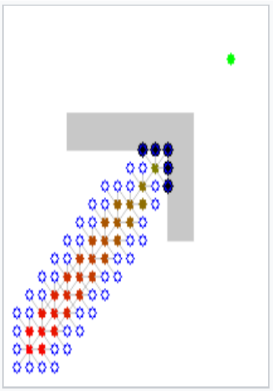
該演算法綜合了最良優先搜尋和Dijkstra演算法的優點：在進行啟發式搜尋提高演算法效率的同時，可以保證找到一條最佳路徑（基於評估函式）。

在此演算法中，如果以$g(n)$表示從起點到任意頂點n的實際距離，$h(n)$表示任意頂點n到目標頂點的估算距離（根據所採用的評估函式的不同而變化），那麼A*演算法的估算函式為：

$$f(n) = g(n) + h(n)$$

這個公式遵循以下特性：

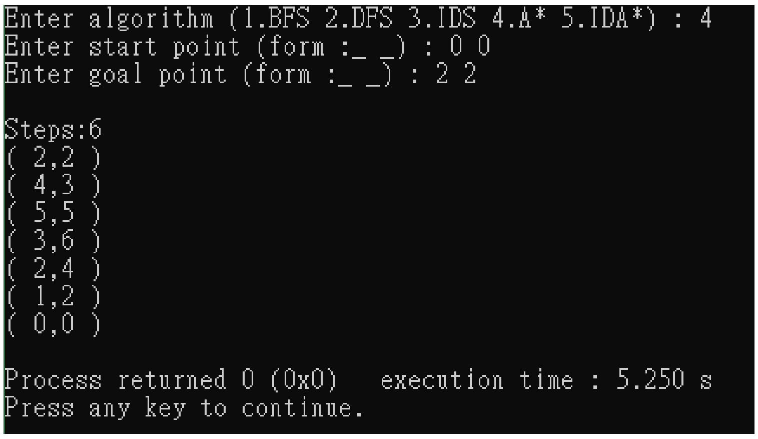
- 如果$g(n)$為0，即只計算任意頂點n到目標的評估函式$h(n)$，而不計算起點到頂點n的距離，則演算法轉化為使用貪心策略的最良優先搜尋，速度最快，但可能得不出最佳解；
- 如果$h(n)$不大於頂點n到目標頂點的實際距離，則一定可以求出最佳解，而且$h(n)$越小，需要計算的節點越多，演算法效率越低，常見的評估函式有——歐幾里得距離、曼哈頓距離、切比雪夫距離；
- 如果$h(n)$為0，即只需求出起點到任意頂點n的最短路徑$g(n)$，而不計算任何評估函式$h(n)$，則轉化為單源最短路徑問題，即Dijkstra演算法，此時需要計算最多的頂點；


A*搜尋演算法的演示圖

概念 ： 起始到目標的距離可以拆成 起始點到任一點的距離+此任一點到終點的距離。也就是 **f(n) = g(n) +h(n)**

可以利用這個方法來簡化問題

Heuristic function: A simple heuristic function to use is floor((|dx|+|dy|)/3), where (dx,dy) is the vector from the current location to the goal, and floor(v) is the largest integer equal to or less than v.
利用給的 Heuristic function 下去把 8 個方向的值先求出一個值，再從小到大去執行對應的方向，再利用 BFS 方式迴圈，有點像 greedy algorithm 的方式來求解，最後不一定會得到最佳解，但會比 DFS 接近很多。

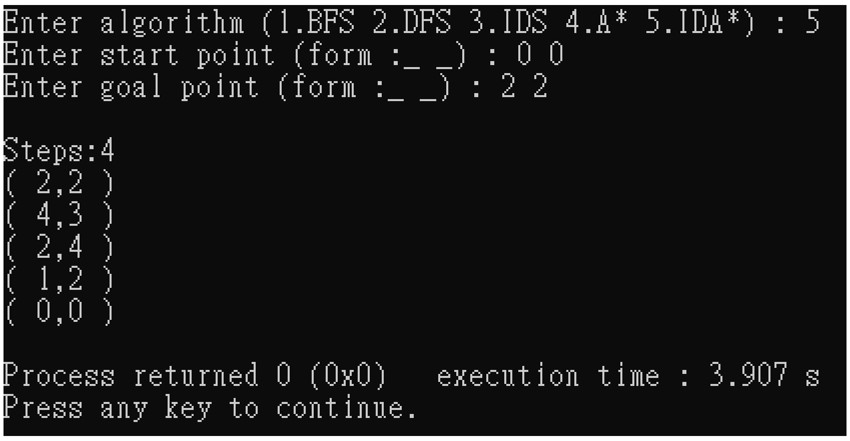

```
Enter algorithm (1.BFS 2.DFS 3.IDS 4.A* 5.IDA*) : 4
Enter start point (form :_ _) : 0 0
Enter goal point (form :_ _) : 2 2

Steps:6
( 2,2 )
( 4,3 )
( 5,5 )
( 3,6 )
( 2,4 )
( 1,2 )
( 0,0 )

Process returned 0 (0x0)   execution time : 5.250 s
Press any key to continue.
```

Iterative deepening A* (IDA*)

概念 ： 是對狀態空間的搜尋策略。它重複地執行一個有深度限制的深度優先搜尋，每次執行結束後，它增加深度并迭代，直到找到目標狀態。可以理解成疊加的 A*



```
Enter algorithm (1.BFS 2.DFS 3.IDS 4.A* 5.IDA*) : 5
Enter start point (form :_ _) : 0 0
Enter goal point (form :_ _) : 2 2

Steps:4
( 2,2 )
( 4,3 )
( 2,4 )
( 1,2 )
( 0,0 )

Process returned 0 (0x0)   execution time : 3.907 s
Press any key to continue.
```

觀察 ：

BFS 可以準確得到最佳解，A*是估計一個可能能求到最佳解的值再下去找，雖然不一定是最佳但也很接近了，DFS 就是一條路窮舉，如果全部展開也可以找到最佳解，但如果是先找到就停就很難找到最佳解了，如果數據很大就很沒效率。

心得 ：

這次的作業我學到了 queue 這個函式，也複習很多之前的工具，讓我更了解上課的內容，以及可以用來做那些運用，各個方法的優缺點。

問題 ：

不過我還是有點不太懂 Iterative deepening 的意義，感覺就是換一個想法做一樣的事，不過我感覺還是 BFS 最直觀也最好用

未來的想法 ：

感覺很多問題都可以用類似的想法，做樹狀圖用這些方法來解決問題，我再想可不可以用 A*來實作一個當作投資建議的參考值。

```
#include <iostream>
```

```
#include <iostream>
#include <queue>
#include <math.h>
#include <algorithm>
using namespace std;

//節點的結構
struct node {
    int cur,x,y,step;
    node *pre;
};
void BFS(int s,int s1,int g,int g1,int step){
    //先做一 NODE 把起始點存入
    node *start=new node;
    start->x=s;
    start->y=s1;
    start->step=0;

    //TABLE 用來檢查下一個準備要做的點有沒有走過，走過代表有
人更早就走過了，沒必要再做一次
    vector<node> table;

    //用來存還剩下要做的 NODE
    queue<node> memory;
    memory.push(*start);
    while(memory.front().x!=g || memory.front().y!=g1){
        int x1,y1;
        node *parent = new node;
        *parent=memory.front();
        table.push_back(memory.front());
        memory.pop();
        /*----------------------------方向 1----------------------------*/
```

```
}

node *next = new node;
next->x=parent->x+1;
next->y=parent->y+2;
next->pre=parent;
next->step=parent->step+1;

int check=0;
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
        memory.push(*next);
}
/*---------------------------方向 2---------------------------*/
next->x=parent->x+1;
next->y=parent->y-2;
next->pre=parent;
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
        memory.push(*next);
}
check=0;
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
/*---------------------------方向 3---------------------------*/
next->x=parent->x+2;
next->y=parent->y+1;
next->pre=parent;
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
        memory.push(*next);
}
check=0;
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
/*--------------------------方向 4----------------------------*/
next->x=parent->x+2;
next->y=parent->y-1;
next->pre=parent;
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
        memory.push(*next);
}
check=0;
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
```

```
/*-------------------------方向 5----------------------------*/

next->x=parent->x-1;
next->y=parent->y+2;
next->pre=parent;
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
        memory.push(*next);
}
check=0;
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
/*------------------------方向 6-----------------------------*/

next->x=parent->x-1;
next->y=parent->y-2;
next->pre=parent;
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
        memory.push(*next);
}
check=0;
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
/*------------------------方向 7-----------------------------*/
```

```
        next->x=parent->x-2;
        next->y=parent->y+1;
        next->pre=parent;
        for(int i=0;i<table.size();i++){
            if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
            }
        }
        if(check==0){
            memory.push(*next);
        }
        check=0;
        if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
            check=1;
        }
        /*-----------------------方向 8-------------------------------*/
        next->x=parent->x-2;
        next->y=parent->y-1;
        next->pre=parent;
        check=0;
        if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
            check=1;
        }
        for(int i=0;i<table.size();i++){
            if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
            }
        }
        if(check==0){
            memory.push(*next);
        }

    }
    node result;
    result=memory.front();
```

```
        cout <<endl<<"Steps:"<< memory.front().step << endl;
        while(result.pre){
                cout << "( " <<     result.x << "," << result.y<<" )"<<endl;
                result=*result.pre;
        }
        cout << "( " <<     result.x << "," << result.y<<" )"<<endl;

}
int ans=0;
vector<node> path;
int DFS(int s,int s1,int g,int g1,int step){
        if(ans==1){
                return 0;
        }
        node *start=new node;
        start->x=s;
        start->y=s1;
        start->step=step;

        path.push_back(*start);

        node *next =new node;
        next->x=start->x+1;
        next->y=start->y+2;
        next->pre=start;
        next->step=step+1;

        //下面的 if  判斷有沒有解和超出邊界，else if 判斷是不是到了

else  進入下一層

        if(next->x >8 || next->x<0 || next->y >8 || next->y <0 || ans==1){
        }
        else if(next->x==g && next->y==g1){
                ans=1;
                cout << "Steps : "<<next->step <<endl;
        }
        else{
```

```
        int check=0;
        for(int i=0;i<path.size();i++){
                if(next->x==path[i].x && next->y==path[i].y){
                        check=1;
                        break;
                }
        }
                if(check==0){
                        DFS(next->x,next->y,g,g1,next->step);
        }
}


next->x=start->x+1;
next->y=start->y-2;
next->pre=start;
next->step=step+1;

if(next->x >8 || next->x<0 || next->y >8 || next->y <0 || ans==1){
}
else if(next->x==g && next->y==g1){
        ans=1;
        cout << "Steps : "<<next->step <<endl;
}
else{
        int check=0;
        //cout << 2 <<endl;
        for(int i=0;i<path.size();i++){
                if(next->x==path[i].x && next->y==path[i].y){
                        check=1;
                        break;
                }
        }
                if(check==0){
                        DFS(next->x,next->y,g,g1,next->step);
        }
}
```

```
next->x=start->x+2;
next->y=start->y+1;
next->pre=start;
next->step=step+1;

if(next->x >8 || next->x<0 || next->y >8 || next->y <0 || ans==1){
}
else if(next->x==g && next->y==g1){
	ans=1;
	cout << "Steps : "<<next->step <<endl;
}
else{
	int check=0;
	//cout << 3 <<endl;
	for(int i=0;i<path.size();i++){
		if(next->x==path[i].x && next->y==path[i].y){
			check=1;
			break;
		}
	}
		if(check==0){
			DFS(next->x,next->y,g,g1,next->step);
	}
}

next->x=start->x+2;
next->y=start->y-1;
next->pre=start;
next->step=step+1;

if(next->x >8 || next->x<0 || next->y >8 || next->y <0 || ans==1){
}
else if(next->x==g && next->y==g1){
	ans=1;
	cout << "Steps : "<<next->step <<endl;
}
else{
	int check=0;
```

```
		//cout << 4 <<endl;
		for(int i=0;i<path.size();i++){
			if(next->x==path[i].x && next->y==path[i].y){
				check=1;
				break;
			}
		}
			if(check==0){
				DFS(next->x,next->y,g,g1,next->step);
		}
	}

	next->x=start->x-1;
	next->y=start->y+2;
	next->pre=start;
	next->step=step+1;

	if(next->x >8 || next->x<0 || next->y >8 || next->y <0 || ans==1){
	}
	else if(next->x==g && next->y==g1){
		ans=1;
		cout << "Steps : "<<next->step <<endl;
	}
	else{
		int check=0;
		//cout << 5 <<endl;
		for(int i=0;i<path.size();i++){
			if(next->x==path[i].x && next->y==path[i].y){
				check=1;
				break;
			}
		}
			if(check==0){
				DFS(next->x,next->y,g,g1,next->step);
		}
	}

	next->x=start->x-1;
```

```
next->y=start->y-2;
next->pre=start;
next->step=step+1;

if(next->x >8 || next->x<0 || next->y >8 || next->y <0 || ans==1){
}
else if(next->x==g && next->y==g1){
        ans=1;
        cout << "Steps : "<<next->step <<endl;
}
else{
        int check=0;
        //cout << 6 <<endl;
        for(int i=0;i<path.size();i++){
                if(next->x==path[i].x && next->y==path[i].y){
                        check=1;
                        break;
                }
        }
                if(check==0){
                        DFS(next->x,next->y,g,g1,next->step);
        }
}

next->x=start->x-2;
next->y=start->y+1;
next->pre=start;
next->step=step+1;

if(next->x >8 || next->x<0 || next->y >8 || next->y <0 || ans==1){
}
else if(next->x==g && next->y==g1){
        ans=1;
        cout << "Steps : "<<next->step <<endl;
}
else{
        int check=0;
        //cout << 7 <<endl;
        for(int i=0;i<path.size();i++){
            if(next->x==path[i].x && next->y==path[i].y){
                check=1;
                break;
            }
        }
            if(check==0){
                DFS(next->x,next->y,g,g1,next->step);
        }
    }

    next->x=start->x-2;
    next->y=start->y-2;
    next->pre=start;
    next->step=step+1;

    if(next->x >8 || next->x<0 || next->y >8 || next->y <0 || ans==1){
    }
    else if(next->x==g && next->y==g1){
        ans=1;
        cout << "Steps : "<<next->step <<endl;
    }
    else{
        int check=0;
        //cout << 8 <<endl;
        for(int i=0;i<path.size();i++){
            if(next->x==path[i].x && next->y==path[i].y){
                check=1;
                break;
            }
        }
            if(check==0){
                DFS(next->x,next->y,g,g1,next->step);
        }
    }
    return 0;
}
```

```
int IDS_ans=0;
vector<node> IDS_path;
int IDS(int s,int s1,int g,int g1,int step,int level){
        node *start=new node;
        start->x=s;
        start->y=s1;
        start->step=0;

        vector<node> table;

        queue<node> memory;
        memory.push(*start);
        while(memory.front().x!=g || memory.front().y!=g1){
                int x1,y1;
                node *parent = new node;
                *parent=memory.front();
                table.push_back(memory.front());
                memory.pop();
                node *next = new node;
                next->x=parent->x+1;
                next->y=parent->y+2;
                next->pre=parent;
                next->step=parent->step+1;

                int check=0;
                if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
                        check=1;
                }
                if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
                        check=1;
                }
                for(int i=0;i<table.size();i++){
                        if(next->x==table[i].x && next->y==table[i].y){
                                check=1;
                                break;
                        }
                }
                if(check==0){
```

```
        memory.push(*next);
}
next->x=parent->x+1;
next->y=parent->y-2;
next->pre=parent;
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
        memory.push(*next);
}
check=0;
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
next->x=parent->x+2;
next->y=parent->y+1;
next->pre=parent;
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
        memory.push(*next);
}
check=0;
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
next->x=parent->x+2;
next->y=parent->y-1;
next->pre=parent;
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
        memory.push(*next);
}
check=0;
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
next->x=parent->x-1;
next->y=parent->y+2;
next->pre=parent;
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
        memory.push(*next);
}
check=0;
if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
        check=1;
}
next->x=parent->x-1;
next->y=parent->y-2;
next->pre=parent;
for(int i=0;i<table.size();i++){
        if(next->x==table[i].x && next->y==table[i].y){
                check=1;
                break;
        }
}
if(check==0){
		memory.push(*next);
	}
	check=0;
	if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
		check=1;
	}
	next->x=parent->x-2;
	next->y=parent->y+1;
	next->pre=parent;
	for(int i=0;i<table.size();i++){
		if(next->x==table[i].x && next->y==table[i].y){
			check=1;
			break;
		}
	}
	if(check==0){
		memory.push(*next);
	}
	check=0;
	if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
		check=1;
	}
	next->x=parent->x-2;
	next->y=parent->y-1;
	next->pre=parent;
	check=0;
	if(next->x >8 || next->x<0 || next->y >8 || next->y <0){
		check=1;
	}
	for(int i=0;i<table.size();i++){
		if(next->x==table[i].x && next->y==table[i].y){
			check=1;
			break;
		}
	}
	if(check==0){
		memory.push(*next);
	}
```

```
        }
        node result;
        result=memory.front();
        cout <<endl<<"Steps:"<< memory.front().step << endl;
        while(result.pre){
                cout << "( " <<    result.x << "," << result.y<<" )"<<endl;
                result=*result.pre;
        }
        cout << "( " <<    result.x << "," << result.y<<" )"<<endl;

}

int A(int s,int s1,int g,int g1,int step){
        node *start=new node;
        start->x=s;
        start->y=s1;
        start->step=0;

        vector<node> table;

        queue<node> memory;
        memory.push(*start);
        while(memory.front().x!=g || memory.front().y!=g1){
                node *parent = new node;
                *parent=memory.front();
                table.push_back(memory.front());
                memory.pop();

                int x1[8],y1[8];
                float heuristic_number[8];
                x1[0]=memory.front().x+1;
                x1[1]=memory.front().x+1;
                x1[2]=memory.front().x+2;
                x1[3]=memory.front().x+2;
                x1[4]=memory.front().x-1;
                x1[5]=memory.front().x-1;
                x1[6]=memory.front().x-2;
```

```
            x1[7]=memory.front().x-2;
            y1[0]=memory.front().y+2;
            y1[1]=memory.front().y-2;
            y1[2]=memory.front().y+1;
            y1[3]=memory.front().y-1;
            y1[4]=memory.front().y+2;
            y1[5]=memory.front().y-2;
            y1[6]=memory.front().y+1;
            y1[7]=memory.front().y-1;
            for(int i=0;i<8;i++){
                heuristic_number[i]=(abs(g-x1[i])+abs(g1-y1[i]))/3;
            }
            sort(heuristic_number,heuristic_number+8);
            node *next = new node;
            for(int i=0;i<8;i++){
                if((abs(g-x1[0])+abs(g1-y1[0]))/3==heuristic_number[i]){
                    next->x=parent->x+1;
                    next->y=parent->y+2;
                    next->pre=parent;
                    next->step=parent->step+1;

                    int check=0;
                    if(next->x >8 || next->x<0 || next->y >8 || next->y
    <0){
                        check=1;
                    }
                    for(int i=0;i<table.size();i++){
                        if(next->x==table[i].x && next->y==table[i].y){
                            check=1;
                            break;
                        }
                    }
                    if(check==0){
                        memory.push(*next);
                    }
                }
                else if((abs(g-x1[1])+abs(g1-
    y1[1]))/3==heuristic_number[i]){
```

```
                next->x=parent->x+1;
                next->y=parent->y-2;
                next->pre=parent;
                next->step=parent->step+1;

                int check=0;
                if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                    check=1;
                }
                for(int i=0;i<table.size();i++){
                    if(next->x==table[i].x && next->y==table[i].y){
                        check=1;
                        break;
                    }
                }
                if(check==0){
                    memory.push(*next);
                }
            }
            else if((abs(g-x1[2])+abs(g1-
y1[2]))/3==heuristic_number[i]){
                next->x=parent->x+2;
                next->y=parent->y+1;
                next->pre=parent;
                next->step=parent->step+1;

                int check=0;
                if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                    check=1;
                }
                for(int i=0;i<table.size();i++){
                    if(next->x==table[i].x && next->y==table[i].y){
                        check=1;
                        break;
                    }
                }
```

```
                        if(check==0){
                                memory.push(*next);
                        }
                }
                else if((abs(g-x1[3])+abs(g1-
y1[3]))/3==heuristic_number[i]){
                        next->x=parent->x+2;
                        next->y=parent->y-1;
                        next->pre=parent;
                        next->step=parent->step+1;

                        int check=0;
                        if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                                check=1;
                        }
                        for(int i=0;i<table.size();i++){
                                if(next->x==table[i].x && next->y==table[i].y){
                                        check=1;
                                        break;
                                }
                        }
                        if(check==0){
                                memory.push(*next);
                        }
                }
                else if((abs(g-x1[4])+abs(g1-
y1[4]))/3==heuristic_number[i]){
                        next->x=parent->x-1;
                        next->y=parent->y+2;
                        next->pre=parent;
                        next->step=parent->step+1;

                        int check=0;
                        if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                                check=1;
                        }
```

```
				for(int i=0;i<table.size();i++){
					if(next->x==table[i].x && next->y==table[i].y){
						check=1;
						break;
					}
				}
				if(check==0){
					memory.push(*next);
				}
			}
			else if((abs(g-x1[5])+abs(g1-
y1[5]))/3==heuristic_number[i]){
				next->x=parent->x-1;
				next->y=parent->y-2;
				next->pre=parent;
				next->step=parent->step+1;

				int check=0;
				if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
					check=1;
				}
				for(int i=0;i<table.size();i++){
					if(next->x==table[i].x && next->y==table[i].y){
						check=1;
						break;
					}
				}
				if(check==0){
					memory.push(*next);
				}
			}
			else if((abs(g-x1[6])+abs(g1-
y1[6]))/3==heuristic_number[i]){
				next->x=parent->x-2;
				next->y=parent->y+1;
				next->pre=parent;
				next->step=parent->step+1;
				int check=0;
				if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
					check=1;
				}
				for(int i=0;i<table.size();i++){
					if(next->x==table[i].x && next->y==table[i].y){
						check=1;
						break;
					}
				}
				if(check==0){
					memory.push(*next);
				}
			}
			else if((abs(g-x1[7])+abs(g1-
y1[7]))/3==heuristic_number[i]){
				next->x=parent->x-2;
				next->y=parent->y-1;
				next->pre=parent;
				next->step=parent->step+1;

				int check=0;
				if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
					check=1;
				}
				for(int i=0;i<table.size();i++){
					if(next->x==table[i].x && next->y==table[i].y){
						check=1;
						break;
					}
				}
				if(check==0){
					memory.push(*next);
				}
			}
			}
		}
		node result;
		result=memory.front();
		cout <<endl<<"Steps:"<< memory.front().step << endl;
		while(result.pre){
			cout << "( " <<	result.x << "," << result.y<<" )"<<endl;
			result=*result.pre;
		}
		cout << "( " <<	result.x << "," << result.y<<" )"<<endl;
}

int solution=0;
int IDA(int s,int s1,int g,int g1,int step,int level){
		node *start=new node;
		start->x=s;
		start->y=s1;
		start->step=0;
		cout << "level : "<<level <<endl;
		vector<node> table;
		cout << "times"<<endl;
		queue<node> memory;
		memory.push(*start);
		while(memory.front().x!=g || memory.front().y!=g1){
			if(memory.front().step>level){
				memory.pop();
			}
			node *parent = new node;
			*parent=memory.front();
			table.push_back(memory.front());
			memory.pop();
			cout << "run" <<endl;
			int increase=0;
			int x1[8],y1[8];
			float heuristic_number[8];
			x1[0]=memory.front().x+1;
			x1[1]=memory.front().x+1;
			x1[2]=memory.front().x+2;
		x1[3]=memory.front().x+2;
		x1[4]=memory.front().x-1;
		x1[5]=memory.front().x-1;
		x1[6]=memory.front().x-2;
		x1[7]=memory.front().x-2;
		y1[0]=memory.front().y+2;
		y1[1]=memory.front().y-2;
		y1[2]=memory.front().y+1;
		y1[3]=memory.front().y-1;
		y1[4]=memory.front().y+2;
		y1[5]=memory.front().y-2;
		y1[6]=memory.front().y+1;
		y1[7]=memory.front().y-1;
		for(int i=0;i<8;i++){
			heuristic_number[i]=(abs(g-x1[i])+abs(g1-y1[i]))/3;
		}
		sort(heuristic_number,heuristic_number+8);
		node *next = new node;
		for(int i=0;i<8;i++){
			if((abs(g-x1[0])+abs(g1-y1[0]))/3==heuristic_number[i]){
				next->x=parent->x+1;
				next->y=parent->y+2;
				next->pre=parent;
				next->step=parent->step+1;

				int check=0;
				if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
					check=1;
				}
				for(int i=0;i<table.size();i++){
					if(next->x==table[i].x && next->y==table[i].y){
						check=1;
						break;
					}
				}
				if(check==0&&next->step<=level){
					memory.push(*next);
```

```
                                        increase++;
                                }
                        }
                        else if((abs(g-x1[1])+abs(g1-
y1[1]))/3==heuristic_number[i]){
                                next->x=parent->x+1;
                                next->y=parent->y-2;
                                next->pre=parent;
                                next->step=parent->step+1;

                                int check=0;
                                if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                                        check=1;
                                }
                                for(int i=0;i<table.size();i++){
                                        if(next->x==table[i].x && next->y==table[i].y){
                                                check=1;
                                                break;
                                        }
                                }
                                if(check==0&&next->step<=level){
                                        memory.push(*next);
                                        increase++;
                                }
                        }
                        else if((abs(g-x1[2])+abs(g1-
y1[2]))/3==heuristic_number[i]){
                                next->x=parent->x+2;
                                next->y=parent->y+1;
                                next->pre=parent;
                                next->step=parent->step+1;

                                int check=0;
                                if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                                        check=1;
                                }
                                for(int i=0;i<table.size();i++){
                                        if(next->x==table[i].x && next->y==table[i].y){
                                                check=1;
                                                break;
                                        }
                                }
                                if(check==0&&next->step<=level){
                                        increase++;
                                        memory.push(*next);
                                }
                        }
                        else if((abs(g-x1[3])+abs(g1-
y1[3]))/3==heuristic_number[i]){
                                next->x=parent->x+2;
                                next->y=parent->y-1;
                                next->pre=parent;
                                next->step=parent->step+1;

                                int check=0;
                                if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                                        check=1;
                                }
                                for(int i=0;i<table.size();i++){
                                        if(next->x==table[i].x && next->y==table[i].y){
                                                check=1;
                                                break;
                                        }
                                }
                                if(check==0&&next->step<=level){
                                        increase++;
                                        memory.push(*next);
                                }
                        }
                        else if((abs(g-x1[4])+abs(g1-
y1[4]))/3==heuristic_number[i]){
                                next->x=parent->x-1;
                                next->y=parent->y+2;
```

```
                                next->pre=parent;
                                next->step=parent->step+1;

                                int check=0;
                                if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                                        check=1;
                                }
                                for(int i=0;i<table.size();i++){
                                        if(next->x==table[i].x && next->y==table[i].y){
                                                check=1;
                                                break;
                                        }
                                }
                                if(check==0&&next->step<=level){
                                        increase++;
                                        memory.push(*next);
                                }
                        }
                        else if((abs(g-x1[5])+abs(g1-
y1[5]))/3==heuristic_number[i]){
                                next->x=parent->x-1;
                                next->y=parent->y-2;
                                next->pre=parent;
                                next->step=parent->step+1;

                                int check=0;
                                if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                                        check=1;
                                }
                                for(int i=0;i<table.size();i++){
                                        if(next->x==table[i].x && next->y==table[i].y){
                                                check=1;
                                                break;
                                        }
                                }
                                if(check==0&&next->step<=level){
```

```
                                increase++;
                                memory.push(*next);
                        }
                }
                else if((abs(g-x1[6])+abs(g1-
y1[6]))/3==heuristic_number[i]){
                        next->x=parent->x-2;
                        next->y=parent->y+1;
                        next->pre=parent;
                        next->step=parent->step+1;

                        int check=0;
                        if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                                check=1;
                        }
                        for(int i=0;i<table.size();i++){
                                if(next->x==table[i].x && next->y==table[i].y){
                                        check=1;
                                        break;
                                }
                        }
                        if(check==0&&next->step<=level){
                                increase++;
                                memory.push(*next);
                        }
                }
                else if((abs(g-x1[7])+abs(g1-
y1[7]))/3==heuristic_number[i]){
                        next->x=parent->x-2;
                        next->y=parent->y-1;
                        next->pre=parent;
                        next->step=parent->step+1;

                        int check=0;
                        if(next->x >8 || next->x<0 || next->y >8 || next->y
<0){
                                check=1;
```

```
                    }
                    for(int i=0;i<table.size();i++){
                        if(next->x==table[i].x && next->y==table[i].y){
                            check=1;
                            break;
                        }
                    }
                    if(check==0&&next->step<=level){
                        increase++;
                        memory.push(*next);
                    }
                }

            }
            if(increase==0){
                cout<<"break"<<endl;
                break;
            }
        }
        if(memory.front().x==g && memory.front().y==g1){
            node result;
            result=memory.front();
            cout <<endl<<"Steps:"<< memory.front().step << endl;
            while(result.pre){
                cout << "( " <<    result.x << "," << result.y<<" )"<<endl;
                result=*result.pre;
            }
            solution=1;
            cout << "( " <<    result.x << "," << result.y<<" )"<<endl;
        }
        else{
            return 0;
        }
}
int main(){
        int type;
        cout << "Enter algorithm (1.BFS 2.DFS 3.IDS 4.A* 5.IDA*) : ";
        cin >> type;
```

```
	int start[2],goal[2];
	cout << "Enter start point (form :_ _) : ";
	cin >> start[0] >> start[1];
	cout << "Enter goal point (form :_ _) : ";
	cin >> goal[0] >> goal[1];
	//cout <<start[0] << start[1] << goal[0] << goal[1];
	int i=0;
	switch(type){
		case 1:
			BFS(start[0],start[1],goal[0],goal[1],0);
			break;
		case 2:
			DFS(start[0],start[1],goal[0],goal[1],0);
			break;
		case 3:
			IDS(start[0],start[1],goal[0],goal[1],0,0);
			break;
		case 4:
			A(start[0],start[1],goal[0],goal[1],0);
			break;
		case 5:
			IDA(start[0],start[1],goal[0],goal[1],0,0);
			while(solution==0){
				i++;
				IDA(start[0],start[1],goal[0],goal[1],0,i);
			}
			break;
		default:
			break;
	}
	return 0;
}
```