HW1

0616038 蔡雨恩

Comparsion between the different algorithms:

BFS:

優先建立離起始狀態最近的狀態，適用於轉移成本是固定值。

DFS:

優先建立離起始狀態最遠的狀態，適用於轉移成本是固定值。

IDS:

DLS 的改良版本，若每次放寬的量極少時，可達到類似於 BFS 的功能。

A*:

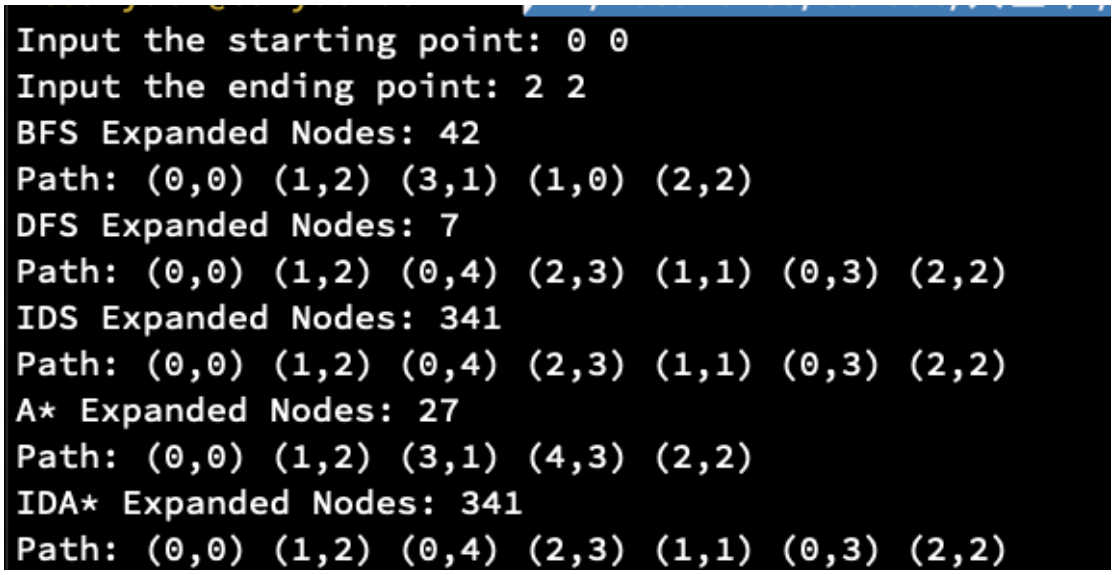
In this homework, we use heuristic function ((|dx|+|dy|)/3).

由小到大建立 g(n)+h(n)

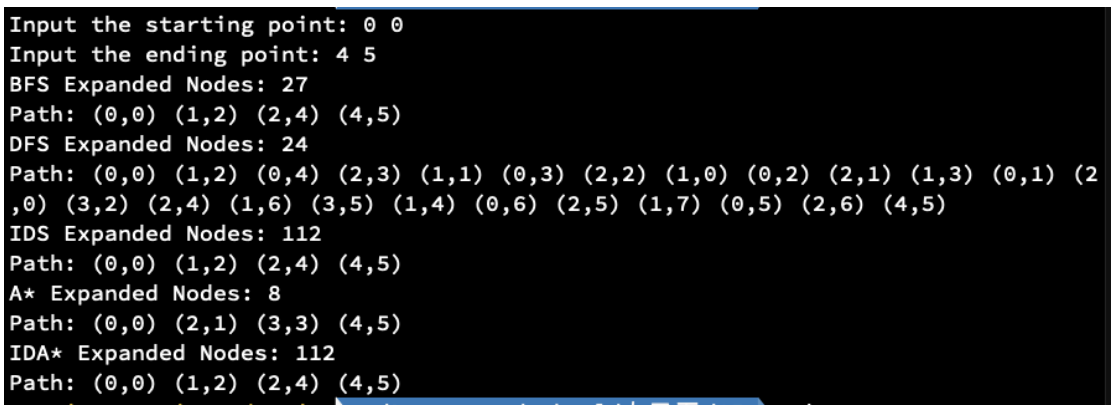
IDA*:

由小到大建立 g(n)+h(n)，每次漸漸放寬 g(n)+h(n)的限制。

Experiment 1:



Experiment 2:



Experiment 3:

```
Input the starting point: 0 0
Input the ending point: 7 5
BFS Expanded Nodes: 54
Path: (0,0) (1,2) (3,3) (5,4) (7,5)
DFS Expanded Nodes: 54
Path: (0,0) (1,2) (0,4) (2,3) (1,1) (0,3) (2,2) (1,0) (0,2) (2,1) (1,3) (0,1) (2
,0) (3,2) (2,4) (1,6) (3,5) (1,4) (0,6) (2,5) (1,7) (0,5) (2,6) (4,5) (3,3) (5,2
) (4,0) (6,1) (5,3) (4,1) (6,0) (7,2) (5,1) (3,0) (4,2) (3,4) (1,5) (2,7) (4,6)
(6,5) (4,4) (3,6) (5,5) (4,3) (3,1) (5,0) (6,2) (5,4) (7,5)
IDS Expanded Nodes: 774
Path: (0,0) (1,2) (0,4) (2,3) (4,2) (5,4) (7,5)
A* Expanded Nodes: 5
Path: (0,0) (2,1) (4,2) (6,3) (7,5)
IDA* Expanded Nodes: 774
Path: (0,0) (1,2) (0,4) (2,3) (4,2) (5,4) (7,5)
```

Although the chessboard isn't too big, it will still be a little bit slow when using DFS so we need to be concern of this problem. I think BFS and DFS have better performance in this homework comparing to IDS, A* and IDA*. I learned about the difference between the five algorithms and when should I use these different algorithms and how to implement them.

Code:

```
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>
#include <queue>
#include <string.h>
#include <bits/stdc++.h>

using namespace std;

struct point {
	int x, y;
}st, ed;

bool operator<(const point &a, const point &b) {
	if(a.x != b.x) return a.x<b.x;
	else return a.y<b.y;
}

int dis[8][8];
```

```
point from[8][8];

vector<point> circle( point p ) {
	vector<point> ans;
	point nxt;

	for(int i=-1;i<=1;i+=2) {
		for(int j=-2;j<=2;j+=4) {
			int dx, dy;
			dx = i;
			dy = j;
			nxt.x = p.x + dx;
			nxt.y = p.y + dy;
			if( ! (nxt.x < 0 || nxt.y < 0 || nxt.x >= 8 || nxt.y >= 8) ) {
				ans.push_back(nxt);
			}
			swap(dx, dy);
			nxt.x = p.x + dx;
			nxt.y = p.y + dy;
			if( ! (nxt.x < 0 || nxt.y < 0 || nxt.x >= 8 || nxt.y >= 8) ) {
				ans.push_back(nxt);
			}
		}
	}

	return ans;
}


vector<point> vv, path;
int dfs(point p, int d, int id) {
	if(d == id) return 1;
	// cout<<"dfs "<<p.x<<' '<<p.y<<' '<<d<<' '<<id<<endl;
	dis[p.x][p.y] = d;
	if(p.x == ed.x && p.y == ed.y) {
		path = vv;
		return 1;
	}
```

```
	vector<point> v = circle(p);
	int cnt = 1;
	for(int i=0;i<v.size();i++){
		point nxt = v[i];
		if(dis[nxt.x][nxt.y] != -1) continue;
		vv.push_back(nxt);
		// cout<<"nxt = "<<nxt.x<<' '<<nxt.y<<endl;
		cnt += dfs(nxt, d+1, id);
		dis[nxt.x][nxt.y] = -1;
		vv.pop_back();
		if(!path.empty()) return cnt;
	}
	return cnt;
}

int heuristic(point p) {
	return - ( abs(p.x - ed.x) + abs(p.y - ed.y) ) / 3;
}

int hSort( pair<int, point> &a, pair<int, point> &b ) {
	return a.first < b.first;
}

int idAStar(point p, int d,int id) {
	if( d == id ) return 1;
	dis[p.x][p.y] = d;
	if(p.x == ed.x && p.y == ed.y) {
		path = vv;
		return 1;
	}

	vector<point> v1 = circle(p);
	vector<pair<int, point> > v;

	for(int i=0;i<v1.size();i++) {
		v.push_back(make_pair(heuristic(v1[i]), v1[i]));
	}
```

```
        sort(v.begin(), v.end(), hSort);


        int cnt = 1;
        for(int i=0;i<v.size();i++){
                point nxt = v[i].second;
                if(dis[nxt.x][nxt.y] != -1) continue;
                vv.push_back(nxt);
                // cout<<"nxt = "<<nxt.x<<' '<<nxt.y<<endl;
                cnt += dfs(nxt, d+1, id);
                dis[nxt.x][nxt.y] = -1;
                vv.pop_back();
                if(!path.empty()) return cnt;
        }

        return cnt;
}

int main(){

        int graphSize;

        cout<<"Input the starting point: ";
        cin>>st.x>>st.y;
        cout<<"Input the ending point: " ;
        cin>>ed.x>>ed.y;

        queue<point> q;
        memset(dis, -1, sizeof(dis));
        graphSize = 0;

        q.push(st);
        while(!q.empty()) {
                point p = q.front();
                q.pop();
                graphSize ++ ;
```

```
        if( p.x == ed.x && p.y == ed.y ) break;

        vector< point > v = circle(p);

        for(int i=0;i<v.size();i++) {
            point nxt = v[i];
            if( dis[nxt.x][nxt.y] == -1 || (dis[nxt.x][nxt.y] > dis[p.x][p.y] + 1) ){
                dis[nxt.x][nxt.y] = dis[p.x][p.y] + 1;
                from[nxt.x][nxt.y] = p;
                q.push(nxt);
            }
        }
}

point p = ed;
path.push_back(p);
while( p.x != st.x || p.y != st.y ) {
        //cout<<"yo "<<p.x<<' '<<p.y<<endl;
        p = from[p.x][p.y];
        path.push_back(p);
}
for(int i=0;i<path.size()/2;i++){
        swap(path[i], path[path.size()-1-i]);
}

cout<<"BFS Expanded Nodes: "<<graphSize<<endl;

cout<<"Path: ";
for(int i=0;i<path.size();i++) {
        cout<<"("<<path[i].x<<','<<path[i].y<<") ";
}
cout<<endl;

path.clear();
vv.clear();
vv.push_back(st);
memset(dis, -1, sizeof(dis));
graphSize = dfs(st, 0, 2147483647);
```

```
cout<<"DFS Expanded Nodes: "<<graphSize<<endl;

cout<<"Path: ";
for(int i=0;i<path.size();i++) {
	cout<<"("<<path[i].x<<','<<path[i].y<<") ";
}
cout<<endl;

graphSize = 0;
path.clear();
vv.clear();
vv.push_back(st);
memset(dis, -1, sizeof(dis));
for(int lim = 1; true ; lim+=3) { // increase limit by 3
	graphSize += dfs(st, 0, lim);
	if(!path.empty()) break;
}

cout<<"IDS Expanded Nodes: "<<graphSize<<endl;

cout<<"Path: ";
for(int i=0;i<path.size();i++) {
	cout<<"("<<path[i].x<<','<<path[i].y<<") ";
}
cout<<endl;


graphSize = 0;
path.clear();
memset(dis, -1, sizeof(dis));

priority_queue< pair<int, point> > pq;

dis[st.x][st.y] = 0;
pq.push( make_pair(heuristic(st), st) );

while(!pq.empty()) {
```

```
        point p = pq.top().second;
        pq.pop();
        graphSize ++ ;

        if(p.x == ed.x && p.y == ed.y) break;

        vector< point > v = circle(p);

        for(int i=0;i<v.size();i++) {
            point nxt = v[i];
            if(dis[nxt.x][nxt.y] == -1 || dis[nxt.x][nxt.y] > dis[p.x][p.y] + 1) {
                dis[nxt.x][nxt.y] = dis[p.x][p.y] + 1;
                from[nxt.x][nxt.y] = p;
                pq.push( make_pair(heuristic(nxt) - dis[nxt.x][nxt.y], nxt) );
            }
        }
}

p = ed;
path.push_back(p);
while( p.x != st.x || p.y != st.y ) {
        //cout<<"yo "<<p.x<<' '<<p.y<<endl;
        p = from[p.x][p.y];
        path.push_back(p);
}
for(int i=0;i<path.size()/2;i++){
        swap(path[i], path[path.size()-1-i]);
}

cout<<"A* Expanded Nodes: "<<graphSize<<endl;

cout<<"Path: ";
for(int i=0;i<path.size();i++) {
        cout<<"("<<path[i].x<<','<<path[i].y<<") ";
}
cout<<endl;

graphSize = 0;
```

```
	path.clear();
	vv.clear();
	vv.push_back(st);
	memset(dis, -1, sizeof(dis));
	for(int lim = 1; true ; lim+=3) { // increase limit by 3
		graphSize += idAStar(st, 0, lim);
		if(!path.empty()) break;
	}

	cout<<"IDA* Expanded Nodes: "<<graphSize<<endl;

	cout<<"Path: ";
	for(int i=0;i<path.size();i++) {
		cout<<"("<<path[i].x<<','<<path[i].y<<") ";
	}
	cout<<endl;


	return 0;
}
```