人工智慧概論

0613413 蔡怡君

- 實驗目的：knight in a 8x8 chessboard

Knight 的移動只能(+-1,+-2) or (+-2,+-1) 給定兩點 find optimal path (minimum number of steps) path

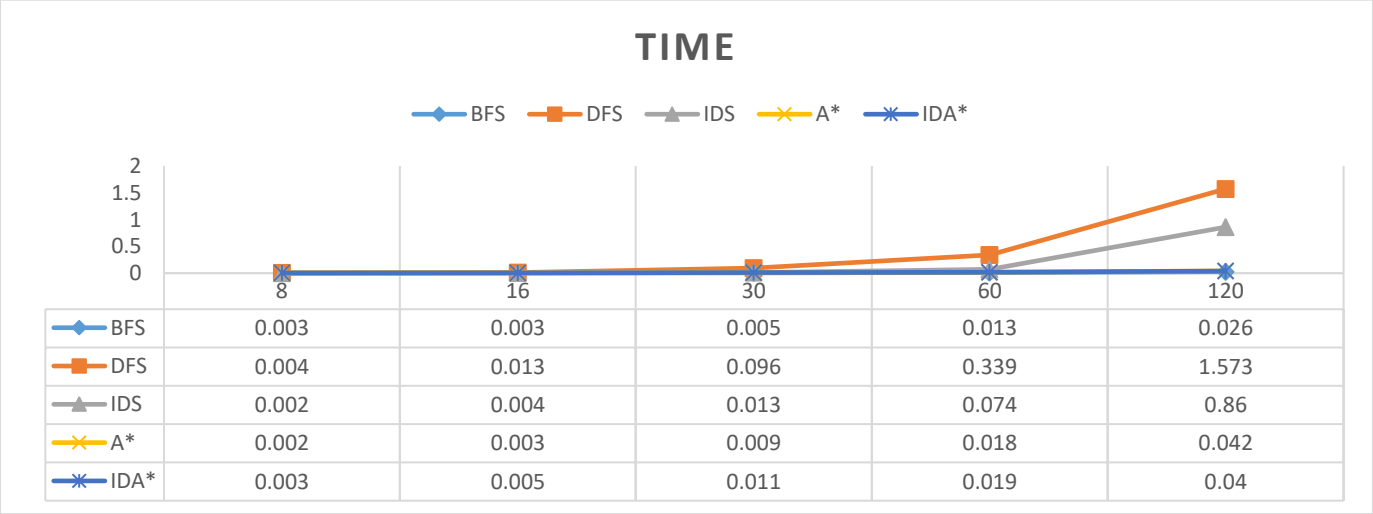
- 實驗方法：(graph-search)

- 使用的概念：在 explored set 和 frontier 都沒有相同 state 的 node (直接使用 table 去記錄這個點有沒有被 visit 過)：將其加入 frontier !

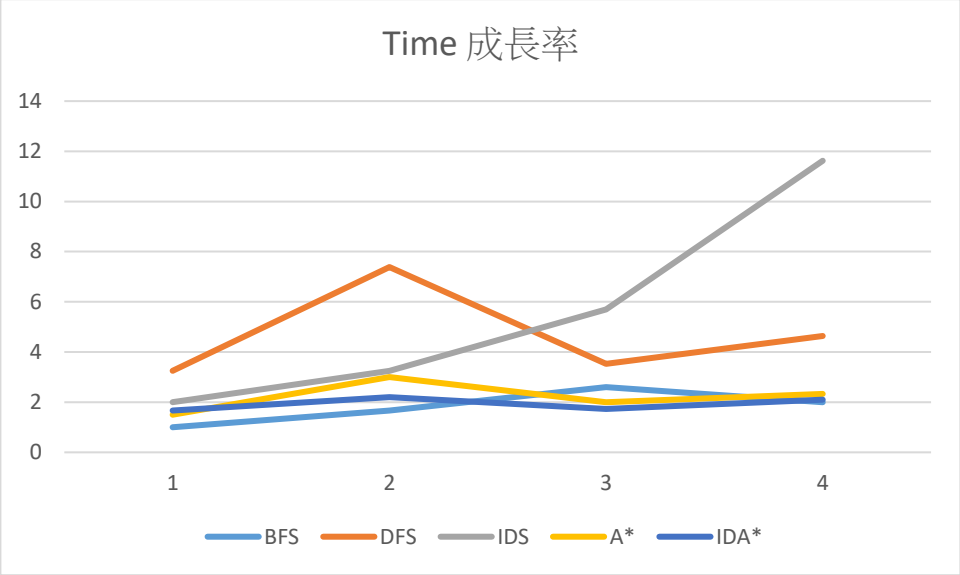
1. BFS：使用 Frontier: **FIFO queue** 能找到最佳解
2. DFS：使用 Frontier: **FIFO stack** 可以找到解答但不一定會找到最佳解
3. IDS：iteratively run DLS，使用 Frontier: **FIFO stack** 能找到最佳解，借用 BFS 的 optimal(第一個找到為最佳解)，再加上 DFS 的優點：占用空間小，但是 expand 的 node 數目比較多。
   --- 透過 Evaluation function 去選擇最佳的 node 去 expand ---
4. A*：使用 Frontier: **priority queue**，expand the lowest f(n)的 node，能找到最佳解
5. IDA*：一開始使用的是 Frontier: **priority queue**，但透過討論區發現其實應該是使用 **stack**，所以有寫兩個版本的 IDA*，再問題區有比較兩者的差別，都能找到最佳解。

- 結果觀察：

Time:



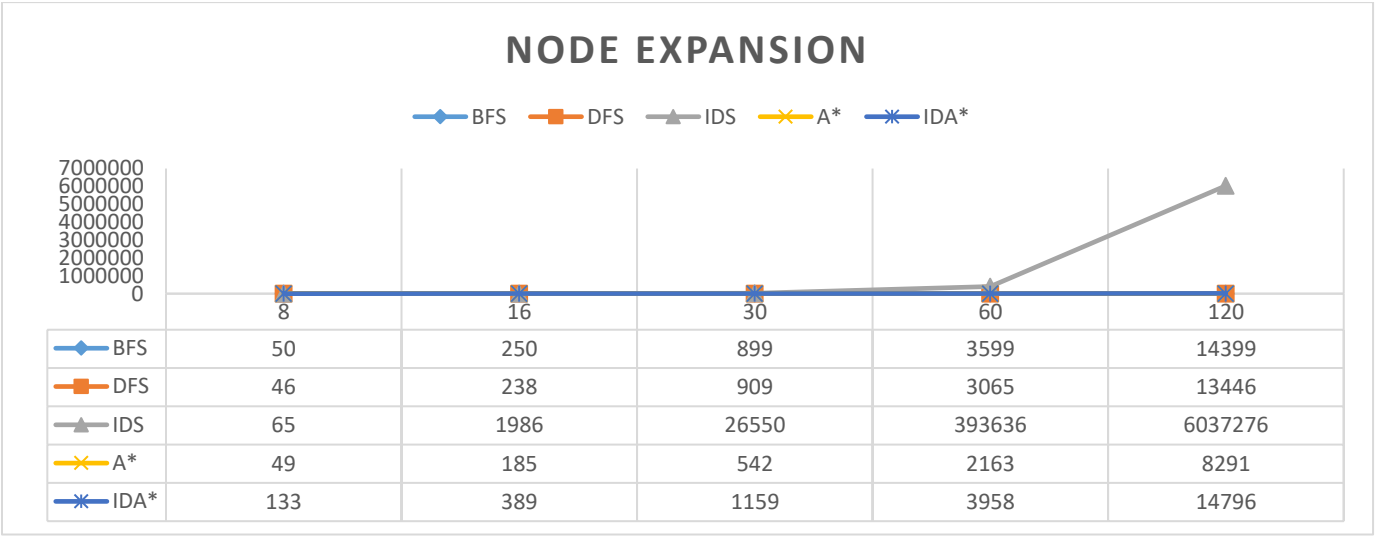
	8	16	30	60	120
BFS	0.003	0.003	0.005	0.013	0.026
DFS	0.004	0.013	0.096	0.339	1.573
IDS	0.002	0.004	0.013	0.074	0.86
A*	0.002	0.003	0.009	0.018	0.042
IDA*	0.003	0.005	0.011	0.019	0.04



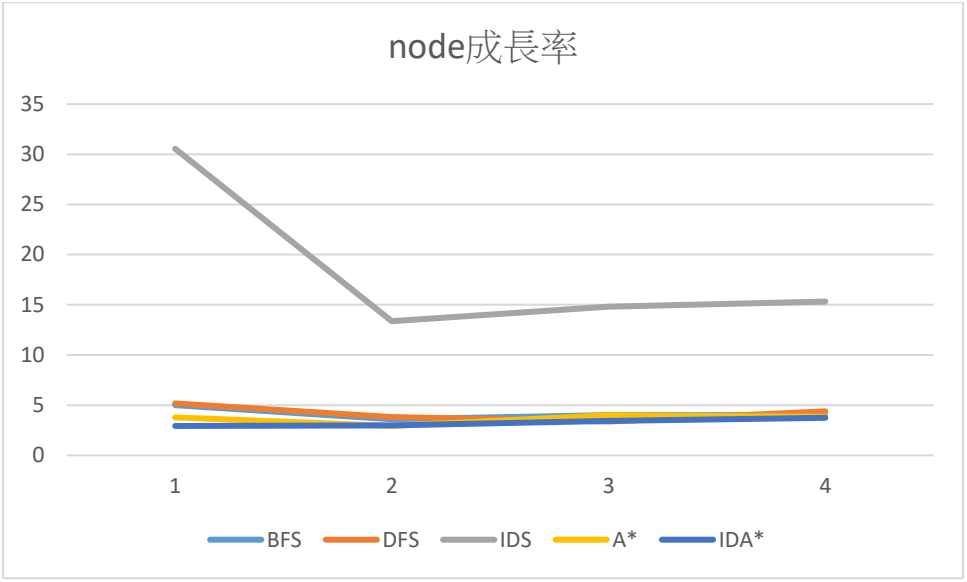
Time 成長率：

BFS	時間大約與 n 成正比。
DFS	時間大約與 n^2 成正比。
IDS	隨著 n 的成長，Time 的成長速度也越大。成長幅度比 DFS 還要大。(看 60 ->120)
A*	時間大約與 n 成正比。
IDA*	時間大約與 n 成正比。

Node expansion:



	8	16	30	60	120
BFS	50	250	899	3599	14399
DFS	46	238	909	3065	13446
IDS	65	1986	26550	393636	6037276
A*	49	185	542	2163	8291
IDA*	133	389	1159	3958	14796



Node expansion 成長率：

BFS	時間大約與 n^2 成正比。(大約 3~4 倍)
DFS	時間大約與 n^2 成正比。(大約 4 倍)
IDS	時間大約與 n^4 成正比。(大約 15~20 倍)
A*	時間大約與 n^2 成正比。(大約 3~4 倍)
IDA*	時間大約與 n^2 成正比。(大約 3~4 倍)

最佳解觀察：

```
plz input (x1,y1) (x2,y2):0 0 2 2
BFS--- Find way ---
path:(0,0) (1,2) (2,4) (4,3) (2,2)
 # of nodes expanded is 47
 Time : 0.002000s
DFS--- Find way ---
path:(0,0) (2,1) (0,2) (2,3) (0,4) (2,5) (4,6) (6,5) (5,3) (3,2) (5,1) (3,0) (2,2)
 # of nodes expanded is 46
Time : 0.004000s
IDS--- Find way ---
path:(0,0) (2,1) (0,2) (1,0) (2,2)
 # of nodes expanded is 65
Time : 0.004000s
A*--- Find way ---
path:(0,0) (1,2) (3,1) (1,0) (2,2)
 # of nodes expanded is 49
Time : 0.002000s
IDA*--- Find way ---
path:(0,0) (1,2) (2,4) (0,3) (2,2)
總共search 5次
 # of nodes expanded is 133
Time : 0.001000s
```

利用老師給的(0,0)->(2,2)，最佳 Path 為總共經過五個點，觀察答案發現只有 DFS 沒有最佳的解，其他 BFS、IDS、A*、IDA*，都能得到最佳解！或是透過其他點，也可以得到同樣的結果。

- 結果分析：

以**時間率**來看，DFS、IDS 為表現最壞的兩者，兩者的時間成長率明顯大於其他三個，再以**最佳解**來看 DFS 雖然會找到解但不一定是最佳解，所以在這個問題上 DFS 較不適合，再看到 node

expansion 的表在其中 IDS 為最高的，因為其不斷的加深 depth 去找到最佳解，所以會展開眾多 node，但是它的 node 占用記憶體不會項 BFS 這麼多，因為 IDS 每一次的 iteration 都會重新 restart 不是全部存下來。

所以目前看到**時間成長率**與 **node expansion 成長率**，IDA*與 A*與 BFS 為表現較佳者的，但是 BFS 的 space complexity 會比其他兩者還要大，因為它會展開不必要的 node，像是哩，的 space complexity 是以指數型成長。最後以 IDA*與 A*來看，兩者都是因為會透過 Evaluation function f(n) = g(n) + h(n) 去篩選排除表現較不佳的 node，所以可以得知這個 Evaluation function 在此問題中是 consistent 的，很有用的節省了時間、node expansion。**所以再這個實驗當中 IDA*跟 A*是比較推薦使用的。**

實驗中的遇到的問題：

Question 1：**IDS 沒有辦法找到最佳解？**

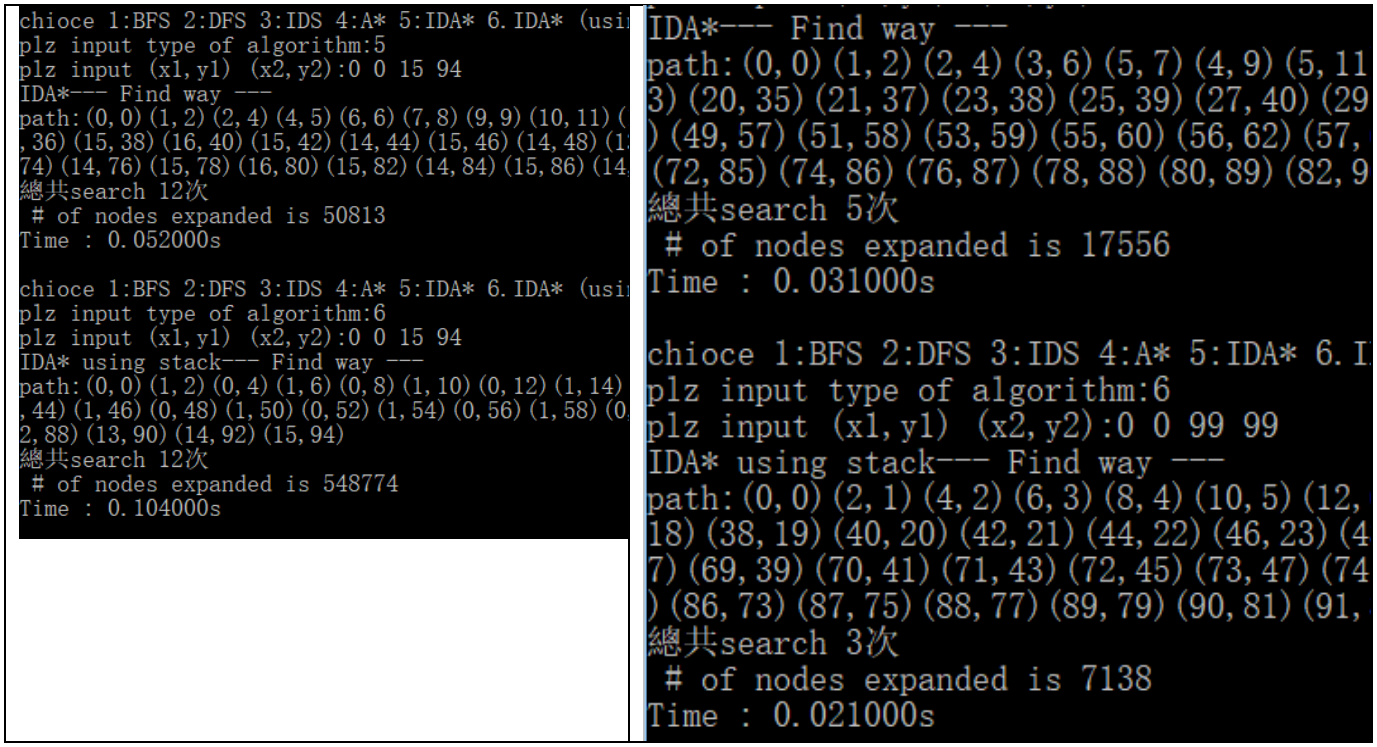
Answer 1： 因為我的 algorithm 是使用「**state 並不在 explored set，但是已經在 frontier 就丟棄的條件的話**」，在 IDS 中好像會出現 IDS 找不到最佳解的狀況。假設最佳路徑為 P1,其經過 k node，但透過 IDS 卻找不到最佳解，是因為程式方面上，先跑 Path P2，而 P2 上面也有經過 k node，所以 k node 在找到最佳解 P1 前就被丟入 frontier，所以在跑最佳解(k 點前)的時候，因為 k node 已經存在於 frontier & explored set，所以沒辦法找到最佳解。

解決方法：紀錄每一個 node 的 depth，當在產生相同 state 的時候，若新的深度小於之前記住的，就重新把其加入 Frontier，因為如果原本的 depth 是最小的話，應該要可以找到最佳解。

Question 2： 使用 priority queue 的 IDA*會比使用 stack 的 IDA*好嗎？

經過兩次測值(0,0)->(15,94)：node expansion、time 都是 priority queue 比較好，但是(0,0)->(99,99)發現又變成 stack 表現較佳。所以這部分還沒有準確的答案可以知道使用哪個方法較佳。

程式選項 5 IDA* (using priority queue)	程式選項 6 IDA*(using stack)
(0,0) -> (15,94)	(0,0) -> (99,99)



Thing I have learned：

這次的作業，可以應用在很多的手機益智遊戲方面，像是 8 - puzzle，或是下象棋，原來這些遊戲都是可以透過以前演算法所學過的 BFS、DFS 去解題，還有新教的 IDS、A*、IDA*，還有新的概念像是 evaluation function、heuristic function，去讓一個問題的解變的更明確，減少不必要的計算量、加快得到解答的速度。

Remaining Question：

因為我在這次的演算法當中是使用的概念是：「在 explored set 和 frontier 都沒有相同 state 的 node (直接使用 table 去記錄這個點有沒有被 visit 過)：將其加入 frontier 」，所以再 Qustion1，有指出 IDS、IDA*要去多檢查之前 visit 過的深度有沒有比新 visit 的大，所以很好奇是不是其它兩種狀況的效能會比較高呢？(討論區關於 DFS 那篇裡面，我也跟他們遇到一樣的問題)。

```
1.  // 程式碼
    #include<iostream>
2.  #include<stdio.h>
3.  #include<stdlib.h>
4.  #include<math.h>
5.  #include<queue>
6.  #include<stack>
7.  #include<vector>
8.  #include<time.h>
9.  #define N 100
10. using namespace std;
11.
12. /*
13.     author : 0613413 蔡怡君
14.     content: knight in a 8*8 chessboard
15.              move only allow (+-1,+-2)or(+-2,+-1)
16.              find optimal path from 2 given locations
17. */
18.
19. struct point
20. {
21.     int x,y;
22.     int cost; // use for heuristic function
23. }p1,p2,dir[8];
24.
25. // parent 紀錄這一個 node 上面的爸爸是誰
26. // table 紀錄這一個點有沒有被 visited 過
27. // low cost using for IDA* 紀錄目前最小的 cost
28. point parent[N][N];
29. int table[N][N]={0},lowcost=1,node=1;
30.
31. void initial()
32. {
33.     dir[0]={1,2};
34.     dir[1]={1,-2};
35.     dir[2]={-1,2};
36.     dir[3]={-1,-2};
37.     dir[4]={2,1};
38.     dir[5]={2,-1};
39.     dir[6]={-2,1};
40.     dir[7]={-2,-1};
41. }
42.
43. void clear()
44. {
45.     // reset table & parent
46.     node = 0;
47.     for(int i = 0 ; i < N ; i++){
48.         for(int j = 0 ; j < N ; j++){
49.             table[i][j] = 0;
50.             parent[i][j].x = -1;
51.             parent[i][j].y = -1;
52.         }
53.     }
54.
55.
56. }
57.
58. void print_path(point now)
59. {
```

```
60.     if(parent[now.x][now.y].x == now.x && parent[now.x][now.y].y == now.y)
61.     {
62.         cout<<"("<<now.x<<","<<now.y<<")";
63.         return;
64.     }
65.     else
66.         print_path(parent[now.x][now.y]);
67.     cout<<"("<<now.x<<","<<now.y<<")";
68. }
69.
70. bool operator < (const point &p1,const point &p2){ return p1.cost < p2.cost;}
71. bool operator > (const point &p1,const point &p2){ return p1.cost > p2.cost;}
72.
73. int BFS(point p1,point p2)
74. {
75.     node = 1;
76.     queue<point>front;
77.     front.push(p1);
78.     table[p1.x][p1.y]=1;
79.     parent[p1.x][p1.y]=p1; // root 的 parent=自己
80.     while(!front.empty())
81.     {
82.         point current = front.front();
83.         front.pop();
84.         // Look 8 directions and checked whether it's visited ?
85.         for(int i = 0 ; i < 8 ; i++){
86.             point now = { current.x + dir[i].x , current.y + dir[i].y };
87.             if(now.x < 0 || now.y < 0 || now.x > N-1 || now.y > N-1) // out of index
88.                 continue;
89.             if(table[now.x][now.y] == 0) // mean 還沒被走過
90.             {
91.                 front.push(now);
92.                 node++;
93.                 table[now.x][now.y] = 1;
94.                 parent[now.x][now.y] = current;
95.                 // if we find point! print path and break.
96.                 if(now.x == p2.x && now.y == p2.y)
97.                 {
98.                     cout<<"--- Find way ---\npath:";
99.                     print_path(front.back());
100.                    return node;
101.                }
102.            }
103.        }
104.    }
105.}
106.
107.// 找到 node 時檢查 explored set & frontier (單純使用 table 去記這點是否被 visited)
108.int DFS(point p1,point p2)
109.{
110.    int i,depth[N][N] = {0}; node = 1;
111.    stack<point>front;
112.    front.push(p1);
113.    table[p1.x][p1.y] = 1;
114.    parent[p1.x][p1.y] = p1; // root 的 parent=自己
115.    while(!front.empty())
116.    {
117.        point current = front.top();
118.        front.pop();
119.        //printf("(%d,%d)",current.x,current.y);
120.        for(i = 0 ; i < 8 ; i ++){
```

```
121.            point now = { current.x + dir[i].x , current.y + dir[i].y};
122.            if(now.x < 0 || now.y < 0 || now.x > N-1 || now.y > N-1)
123.                continue;
124.            if(table[now.x][now.y] && depth[now.x][now.y] > depth[current.x][current.y]+1)
125.            {
126.                table[now.x][now.y]=0;
127.            }
128.            if( table[now.x][now.y] == 0)
129.            {
130.                table[now.x][now.y] = 1;
131.                front.push(now);
132.                parent[now.x][now.y] = current;
133.                depth[now.x][now.y] = depth[current.x][current.y] + 1;
134.                node++;
135.                if(now.x == p2.x && now.y == p2.y) // means arrive
136.                {
137.                    cout<<"--- Find way ---\npath:";
138.                    print_path(front.top());
139.                    return node;
140.                }
141.            }
142.        }
143.    }
144.    return 0;
145.}
146.
147.// -- Depth-limited Search
148.int DLS(point p1,point p2,int d)
149.{
150.    // add depth[][] to memorize depth of the node
151.    int i, depth[N][N]={0};
152.    node = 1;
153.    stack<point>front;
154.    front.push(p1);
155.    table[p1.x][p1.y]=1;
156.    parent[p1.x][p1.y] = p1; // root 的 parent=自己
157.    //printf("\n\ndepth limit:%d\n",d);
158.    while(!front.empty())
159.    {
160.        point current = front.top();
161.        front.pop();
162.        // control depth
163.        if(depth[current.x][current.y] == d)
164.            continue;
165.        for(i = 0 ; i < 8 ; i ++){
166.            point now = { current.x + dir[i].x , current.y + dir[i].y};
167.            if(now.x < 0 || now.y < 0 || now.x > N-1 || now.y > N-1)
168.                continue;
169.            // 新的點 之前被走過但之前的深度比較大 現今走的 depth 比較小 所以要再丟入 frontier 一次
170.            if(table[now.x][now.y] && depth[now.x][now.y] > depth[current.x][current.y]+1)
171.            {
172.                table[now.x][now.y]=0;
173.            }
174.            if( table[now.x][now.y] == 0)
175.            {
176.                table[now.x][now.y] = 1;
177.                front.push(now);
178.                parent[now.x][now.y] = current;
179.                depth[now.x][now.y] = depth[current.x][current.y]+1; // depth = 原來的點 +1
180.                node++;
181.                if(now.x == p2.x && now.y == p2.y) // means arrive
```

```
                {
                    cout<<"--- Find way ---\npath:";
                    print_path(front.top());
                    return node;
                }
                // printf("depth=%d. (%d,%d)",depth[now.x][now.y],now.x,now.y);
            }
        }
    }
    return 0; // if it's not found return 0
}

// iterative-deepening search
int IDS(point p1,point p2)
{
    // i for control depth
    int result = 0, i = 1 ,lowcost = 1, sum =0;
    while(1){
        clear();
        result = DLS(p1,p2,i++);
        sum += node;
        if(result != 0)
            return sum;
    }
    return 0;
}

int h(point n,point p2)
{
    int x,y;
    x = (n.x > p2.x) ? n.x - p2.x : p2.x - n.x;
    y = (n.y > p2.x) ? n.y - p2.y : p2.y - n.y;
    if(n.x == p2.x && n.y == p2.y)
    {
        x=-6666; y=-6666;
    }
    return floor((x+y)/3);
}

// A*
int Astar(point p1,point p2)
{
    int depth[N][N] = {0}; node=1;
    priority_queue<point,vector<point>,greater<point> >front;
    front.push(p1);
    table[p1.x][p1.y]=1;
    parent[p1.x][p1.y]=p1;
    while(!front.empty())
    {
        point current = front.top();
        front.pop();
        for(int i = 0 ; i < 8 ; i ++){
            point now = { current.x + dir[i].x, current.y + dir[i].y};
            if(now.x < 0 || now.y < 0 || now.x > N-1 || now.y > N-1)
                continue;
            if(table[now.x][now.y] == 0)
            {
                table[now.x][now.y] = 1;
                depth[now.x][now.y] = depth[current.x][current.y] + 1;
                parent[now.x][now.y] = current;
                node ++;
                now.cost = depth[now.x][now.y] + h(now,p2);
```

```
                front.push(now);
                //printf("depth=%d. (%d,%d) cost=%d\n",depth[now.x][now.y],now.x,now.y,now.cost)
    ;
                if ( now.x == p2.x && now.y == p2.y )
                {
                    cout<<"--- Find way ---\npath:";
                    print_path(front.top());
                    return node;
                }
                //cout<<"now.cost"<<depth[now.x][now.y]<<"+"<<h(now,p2)<<"="<<now.cost<<endl;
            }
        }
    }
}

// using for IDA*
// Deepening Limited A*

int DLAstar(point p1,point p2)
{
    // cost = 第一次跑 DLAstar 的 low cost;
    int depth[N][N] = {0},cost = lowcost;
    node = 1;
    bool flag = false;

    priority_queue<point,vector<point>,greater<point> >front;
    front.push(p1);
    table[p1.x][p1.y]=1;
    parent[p1.x][p1.y]=p1;
    while(!front.empty())
    {
        point current = front.top();
        front.pop();
        for(int i = 0 ; i < 8 ; i ++){
            point now = { current.x + dir[i].x, current.y + dir[i].y};
            if(now.x < 0 || now.y < 0 || now.x > N-1 || now.y > N-1)
                continue;
            if(table[now.x][now.y] == 0)
            {
                table[now.x][now.y] = 1;
                depth[now.x][now.y] = depth[current.x][current.y] + 1;
                parent[now.x][now.y] = current;
                now.cost = depth[now.x][now.y] + h(now,p2);
                node ++;
                // push 之前要看這個點的 f() 是否超過 limited f()？
                if(now.cost > cost)
                {
                    // 紀錄第一次 大於 low cost 的 flag 要用於下一次
                    if(!flag){ lowcost = now.cost; flag = true; }
                    // cout<<"the step lowcost:"<<lowcost<<endl;}
                    //cout<<"超過 limit 不 push 進去 -";
                }
                else
                    front.push(now);
                //printf("(%d,%d) cost=%d\n",now.x,now.y,now.cost);
                if ( now.x == p2.x && now.y == p2.y )
                {
                    cout<<"--- Find way ---\npath:";
                    print_path(front.top());
                    return node;
                }
```

```
304.            }
305.        }
306.    }
307.    return 0;
308.}
309.
310.int IDAstar(point p1,point p2)
311.{
312.    int result = 0, i = 1, sum =0;
313.    lowcost = 1;
314.    while(1)
315.    {
316.        clear();
317.        result = DLAstar(p1,p2);
318.        sum += node; i++;
319.        //cout<<"第"<<i++<<"次 try"<<"lower cost:"<<lowcost<<endl;
320.        if(result != 0)
321.        {
322.            cout<<"\n 總共 search "<<i<<"次";
323.            return sum;
324.        }
325.    }
326.
327.}
328.
329.
330.int DLAstar_stack(point p1,point p2)
331.{
332.    // cost = 第一次跑 DLAstar 的 low cost;
333.    int depth[N][N] = {0},cost = lowcost;
334.    node = 1;
335.    bool flag = false;
336.
337.    stack<point>front;
338.    front.push(p1);
339.    table[p1.x][p1.y]=1;
340.    parent[p1.x][p1.y]=p1;
341.    while(!front.empty())
342.    {
343.        point current = front.top();
344.        front.pop();
345.        for(int i = 0 ; i < 8 ; i ++){
346.            point now = { current.x + dir[i].x, current.y + dir[i].y};
347.            if(now.x < 0 || now.y < 0 || now.x > N-1 || now.y > N-1)
348.                continue;
349.            // 新的點 之前被走過但之前的深度比較大 現今走的 depth 比較小 所以要再丟入 frontier 一次
350.            if(table[now.x][now.y] && depth[now.x][now.y] > depth[current.x][current.y]+1)
351.                table[now.x][now.y]=0;
352.            if(table[now.x][now.y] == 0)
353.            {
354.                table[now.x][now.y] = 1;
355.                depth[now.x][now.y] = depth[current.x][current.y] + 1;
356.                parent[now.x][now.y] = current;
357.                now.cost = depth[now.x][now.y] + h(now,p2);
358.                node ++;
359.                // push 之前要看這個點的 f() 是否超過 limited f()？
360.                if(now.cost > cost)
361.                {
362.                    // 紀錄第一次 大於 low cost 的 flag 要用於下一次
363.                    if(!flag){ lowcost = now.cost; flag = true;  }
364.                    //cout<<"the step lowcost:"<<lowcost<<endl;
```

```
365.                     //cout<<"超過 limit 不 push 進去 -";
366.                 }
367.                 else
368.                     front.push(now);
369.                 //printf("(%d,%d) cost=%d\n",now.x,now.y,now.cost);
370.                 if ( now.x == p2.x && now.y == p2.y )
371.                 {
372.                     cout<<"--- Find way ---\npath:";
373.                     print_path(front.top());
374.                     return node;
375.                 }
376.             }
377.         }
378.     }
379.     return 0;
380. }
381.
382. int IDAstar_stack(point p1,point p2)
383. {
384.     int result = 0, i = 1, sum =0;
385.     lowcost = 1;
386.     while(1)
387.     {
388.         clear();
389.         result = DLAstar_stack(p1,p2);
390.         sum += node; i++;
391.         //cout<<"第"<<i++<<"次 try"<<"lower cost:"<<lowcost<<endl;
392.         if(result != 0)
393.         {
394.             cout<<"\n 總共 search "<<i<<"次";
395.             return sum;
396.         }
397.     }
398.
399. }
400.
401.
402. // only using for 印 node/ time
403. void check(point p1,point p2)
404. {
405.     time_t start,end;
406.     double t;
407.     cout<<"BFS";
408.     start = clock();
409.     cout<<"\n # of nodes expanded is "<<BFS(p1,p2)<<endl;
410.     end = clock();
411.     t = ((double)(end-start))/CLOCKS_PER_SEC;
412.     printf(" Time : %fs\n",t);
413.     clear();
414.
415.     cout<<"DFS";
416.     start = clock();
417.     cout<<"\n # of nodes expanded is "<<DFS(p1,p2)<<endl;
418.     end = clock();
419.     t = ((double)(end-start))/CLOCKS_PER_SEC;
420.     printf("Time : %fs\n",t);
421.     clear();
422.
423.     cout<<"IDS";
424.     start = clock();
425.     cout<<"\n # of nodes expanded is "<<IDS(p1,p2)<<endl;
```

```
426.     end = clock();
427.     t = ((double)(end-start))/CLOCKS_PER_SEC;
428.     printf("Time : %fs\n",t);
429.     clear();
430.
431.     cout<<"A*";
432.     start = clock();
433.     cout<<"\n # of nodes expanded is "<<Astar(p1,p2)<<endl;
434.     end = clock();
435.     t = ((double)(end-start))/CLOCKS_PER_SEC;
436.     printf("Time : %fs\n",t);
437.     clear();
438.
439.     cout<<"IDA*";
440.     start = clock();
441.     cout<<"\n # of nodes expanded is "<<IDAstar(p1,p2)<<endl;
442.     end = clock();
443.     t = ((double)(end-start))/CLOCKS_PER_SEC;
444.     printf("Time : %fs\n",t);
445.     clear();
446. }
447.
448. int main()
449. {
450.     int type;
451.     initial();
452.     cout<<"--- Welcome to 8-kight ---\n";
453.     time_t start,end;
454.     double t;
455.     while(1)
456.     {
457.         cout<<"\n\nchioce 1:BFS 2:DFS 3:IDS 4:A* 5:IDA* 6.IDA* (using stack)\n";
458.         cout<<"plz input type of algorithm:";
459.         cin>>type;
460.         cout<<"plz input (x1,y1) (x2,y2):";
461.         cin>>p1.x>>p1.y>>p2.x>>p2.y;
462.         clear();
463.         //check(p1,p2);
464.         switch(type)
465.         {
466.             case 1:
467.                 cout<<"BFS";
468.                 start = clock();
469.                 cout<<"\n # of nodes expanded is "<<BFS(p1,p2)<<endl;
470.                 end = clock();
471.                 t = ((double)(end-start))/CLOCKS_PER_SEC;
472.                 printf(" Time : %fs",t);
473.                 break;
474.             case 2:
475.                 cout<<"DFS";
476.                 start = clock();
477.                 cout<<"\n # of nodes expanded is "<<DFS(p1,p2)<<endl;
478.                 end = clock();
479.                 t = ((double)(end-start))/CLOCKS_PER_SEC;
480.                 printf("Time : %fs",t);
481.                 break;
482.             case 3:
483.                 cout<<"IDS";
484.                 start = clock();
485.                 cout<<"\n # of nodes expanded is "<<IDS(p1,p2)<<endl;
486.                 end = clock();
487.                 t = ((double)(end-start))/CLOCKS_PER_SEC;
```

```
                printf("Time : %fs",t);
                break;
            case 4:
                cout<<"A*";
                start = clock();
                cout<<"\n # of nodes expanded is "<<Astar(p1,p2)<<endl;
                end = clock();
                t = ((double)(end-start))/CLOCKS_PER_SEC;
                printf("Time : %fs",t);
                break;
            case 5:
                cout<<"IDA*";
                start = clock();
                cout<<"\n # of nodes expanded is "<<IDAstar(p1,p2)<<endl;
                end = clock();
                t = ((double)(end-start))/CLOCKS_PER_SEC;
                printf("Time : %fs",t);
                break;
            case 6:
                cout<<"IDA* using stack";
                start = clock();
                cout<<"\n # of nodes expanded is "<<IDAstar_stack(p1,p2)<<endl;
                end = clock();
                t = ((double)(end-start))/CLOCKS_PER_SEC;
                printf("Time : %fs",t);
                break;
            default:
                cout<<"----- warning:wrong input -----\n";
        }
    }
    return 0;
}
```