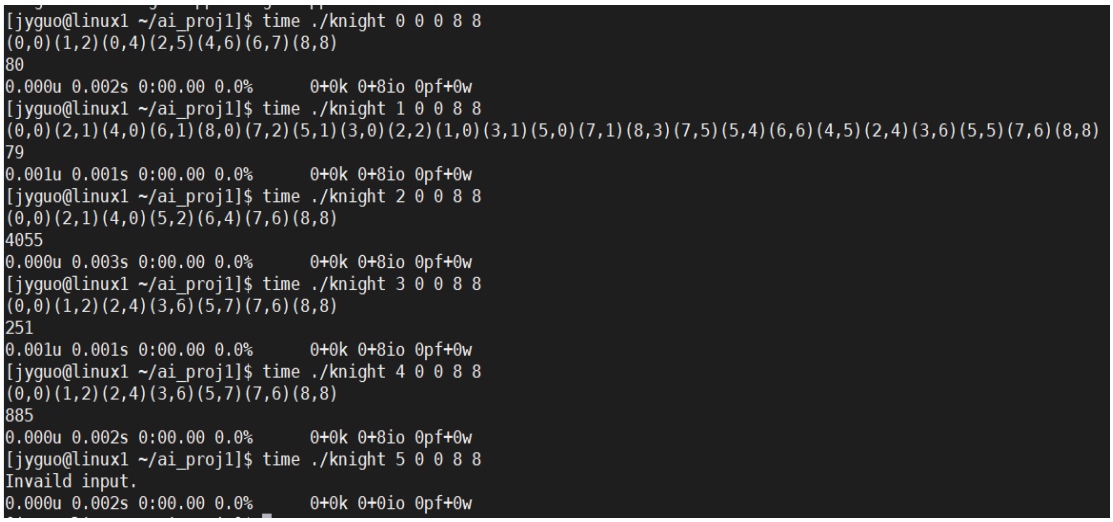
Introduction to AI: Programming Assignment #1

0612129 郭家佑

Basic demo:



In the experience, all test base on the code in appendix, and the excusable code's format is "./knight <ALGORITHSM TYPE> <STARTING X> <STARTING Y> <GOAL X> <GOAL Y>", and it would dump the path coordinate on the screen. About the <ALGORITHSM TYPE> field, 0 is for BFS, 1 is for DFS, 2 is for IDS, 3 is for A*, 4 is for IDA*. And the program also returns the number of used node in the search algorithm. To check the value, I use "echo $?" to print the return value, however, the "$?" can output bigger 255, so that it isn't reliable. So I also print on screen.

In the demo, we can firstly find out that DFS is not work for finding short path if the code is correct.

Experience result:

Firstly, I want to realize what the average cost time in difference algorithm, I used the test.sh which repeat to execute the program with input (4,4) to other point on the chessboard and sum up. And I use the "time" command to be the timer, to make the program more average. I extra run the program 10 time to make the data more

average. Result just like the picture, and the shell script also write in appendix.

```
[jyguo@linux1 ~/ai_proj1]$ time ./test.sh 0
0.092u 0.129s 0:00.56 37.5%     0+0k 0+1288io 0pf+0w
[jyguo@linux1 ~/ai_proj1]$ time ./test.sh 1
0.081u 0.133s 0:00.51 41.1%     0+0k 0+1296io 0pf+0w
[jyguo@linux1 ~/ai_proj1]$ time ./test.sh 2
0.082u 0.134s 0:00.51 41.1%     0+0k 0+1304io 0pf+0w
[jyguo@linux1 ~/ai_proj1]$ time ./test.sh 3
0.094u 0.121s 0:00.52 40.3%     0+0k 0+1296io 0pf+0w
[jyguo@linux1 ~/ai_proj1]$ time ./test.sh 4
0.078u 0.137s 0:00.52 38.4%     0+0k 0+1296io 0pf+0w
```

Sadly, I can't get the obvious different on time.

However, I output the data to the excel and sum up, and it just apart of data, the last line is the sum. The table is too big so that I choose not put entire table on the report. There are 81 point because I resize the cheeseboard to be symmetry.

	A	B	C	D	E	F	G
57	79	39	1323	209	549	216	556
58	6	6	10	25	10	9	10
59	26	53	75	99	75	79	67
60	5	5	10	17	10	17	10
61	77	36	1323	212	552	278	618
62	50	62	380	189	250	150	220
63	23	58	75	51	51	39	43
64	70	43	380	84	154	59	142
65	33	45	75	57	51	29	35
66	67	52	380	149	216	156	226
67	30	38	75	91	67	57	51
68	61	40	380	189	248	145	224
69	25	37	75	83	67	45	51
70	60	56	380	171	236	126	196
71	24	79	75	61	51	29	35
72	59	65	380	148	212	102	176
73	81	50	1323	203	543	163	503
74	68	42	380	142	206	102	176
75	32	47	75	55	51	43	43
76	66	41	380	118	222	115	198
77	29	54	75	59	51	47	51
78	65	80	380	164	200	125	202
79	28	64	75	45	51	43	43
80	64	81	380	88	158	96	176
81	78	59	1323	195	535	127	467
82	BFS	DFS	IDS	A*	IDA*	A*(with h'(x))	IDS*(with h'(x)
83	3321	3321	25225	8479	12872	7440	12415

According the data, we can find out BFS and DFS using less node.

I think it isn't accidental. Due to lookup table, the node BFS and DFS expanded would not higher than the total point on the chessboard, but the other isn't. However, in the part, modifying the heuristic function can make the nodes not need the amount

to expanded.
I make the h'(x) being max(|dx|+|dy|)/3, |dx|/2, |dy|/2). Stand for, a step only can move only 2 units' distance on one coordinate.
Like the following picture:

A*(with h'(x))	IDS*(with h'(x)
7440	12415

The nodes had been used also reduced showed in the data. Maybe there are some better heuristic function, but I still can't find out.

As our previous test, I can't compare the speed by those algorithms. So, I try to extend the cheeseboard size (defined in knight.hpp, MAPSIZEA1 stand for cheeseboard's number of line in one side). In the case I define MAPSIZEA1 301, and try "time ./knight x 100 100 150 150".

```
[jyguo@linux1 ~/ai_proj1]$ time ./knight 0 100 100 150 150
(100,100)(99,102)(100,104)(101,106)(102,108)(103,110)(104,112)(105,114)(106,116)(107,118)(108,120)(109,122)(110,124)(111,126)(112,128)(113,130)(114,132)(116,133
)(118,134)(120,135)(122,136)(124,137)(126,138)(128,139)(130,140)(132,141)(134,142)(136,143)(138,144)(140,145)(142,146)(144,147)(146,148)(148,149)(150,150)
15564
0.004u 0.002s 0:00.02 0.0%      0+0k 0+8io 0pf+0w
[jyguo@linux1 ~/ai_proj1]$ time ./knight 1 100 100 150 150
(100,100)(98,99)(96,98)(94,97)(92,96)(90,95)(88,94)(86,93)(84,92)(82,91)(80,90)(78,89)(76,88)(74,87)(72,86)(70,85)(68,84)(66,83)(64,82)(62,81)(60,80)(58,79)(56,
78)(54,77)(52,76)(50,75)(48,74)(46,73)(44,72)(42,71)(40,70)(38,69)(36,68)(34,67)(32,66)(30,65)(28,64)(26,63)(24,62)(22,61)(20,60)(18,59)(16,58)(14,57)(12,56)(10
,55)(8,54)(6,53)(4,52)(2,51)(0,50)(1,48)(0,46)(1,44)(0,42)(1,40)(0,38)(1,36)(0,34)(1,32)(0,30)(1,28)(0,26)(1,24)(0,22)(1,20)(0,18)(1,16)(0,14)(1,12)(0,10)(1,8)(
0,6)(1,4)(0,2)(1,0)(3,1)(5,0)(7,1)(9,0)(11,1)(13,0)(15,1)(17,0)(19,1)(21,0)(23,1)(25,0)(27,1)(29,0)(31,1)(33,0)(35,1)(37,0)(39,1)(41,0)(43,1)(45,0)(47,1)(49,0)(
```

……(a lot of node)

```
128)(97,130)(95,129)(96,131)(94,130)(92,129)(90,128)(92,127)(90,126)(91,128)(89,127)(90,129)(89,131)(91,132)(93,133)(95,134)(97,135)(96,133)(98,134)(97,132)(99,
133)(100,131)(102,132)(101,130)(103,131)(105,132)(107,133)(109,134)(111,135)(113,136)(114,134)(116,135)(118,136)(119,134)(121,135)(123,134)(125,135)(127,134)(12
8,136)(126,135)(127,137)(128,139)(126,138)(127,140)(125,139)(123,138)(121,137)(122,139)(120,138)(118,137)(119,139)(121,138)(123,139)(122,141)(124,142)(126,143)(
128,144)(127,142)(128,140)(130,139)(132,138)(134,139)(136,140)(137,138)(139,139)(141,140)(143,141)(145,142)(147,143)(149,142)(150,140)(148,139)(146,138)(144,137
)(142,136)(140,135)(141,137)(143,136)(141,135)(142,137)(141,139)(143,140)(145,141)(147,142)(149,143)(151,144)(153,145)(152,143)(153,141)(155,142)(157,143)(159,1
44)(161,145)(163,146)(165,147)(164,149)(166,148)(164,147)(162,146)(160,145)(158,144)(156,143)(155,145)(153,144)(154,146)(152,145)(153,147)(154,145)(156,146)(158
,147)(160,148)(162,149)(164,150)(166,151)(168,152)(170,151)(171,149)(173,148)(175,149)(177,150)(179,151)(181,152)(183,151)(185,152)(187,153)(189,154)(191,155)(1
92,153)(194,154)(196,153)(198,154)(200,155)(202,156)(201,154)(203,155)(205,156)(207,155)(209,156)(211,157)(213,158)(215,159)(217,160)(219,161)(221,162)(223,163)
(225,164)(227,165)(229,166)(231,167)(233,168)(235,169)(237,170)(239,171)(241,172)(243,173)(245,174)(247,175)(249,176)(251,177)(253,178)(255,179)(257,180)(259,18
1)(261,182)(263,183)(262,185)(264,186)(265,188)(264,190)(262,189)(260,188)(258,187)(256,186)(254,185)(252,184)(250,183)(248,182)(246,181)(244,180)(242,179)(240,
178)(238,177)(236,176)(234,175)(232,174)(230,173)(228,172)(226,171)(224,170)(222,169)(220,168)(218,167)(216,166)(214,165)(212,164)(210,163)(208,162)(206,161)(20
4,160)(202,159)(200,158)(198,157)(199,159)(197,158)(195,157)(196,159)(194,158)(196,157)(194,156)(195,158)(193,157)(191,156)(189,155)(187,154)(186,156)(184,155)(
185,157)(183,156)(181,155)(179,154)(177,153)(178,155)(176,154)(174,153)(175,155)(173,154)(175,153)(173,152)(174,154)(172,153)(170,152)(168,151)(166,152)(164,151
)(163,153)(161,152)(159,151)(157,150)(155,149)(156,151)(154,150)(156,149)(154,148)(155,150)(153,149)(151,150)(149,149)(147,148)(145,147)(143,146)(141,145)(139,1
44)(137,143)(136,141)(134,140)(133,138)(131,139)(129,140)(127,139)(128,141)(127,143)(129,144)(131,143)(133,144)(132,142)(134,143)(136,142)(135,140)(137,141)(136
,143)(138,144)(140,145)(139,143)(141,144)(143,145)(145,146)(147,145)(149,146)(148,148)(150,149)(152,150)(151,152)(150,150)
46671
0.009u 0.004s 0:00.02 0.0%      0+0k 0+8io 0pf+0w
[jyguo@linux1 ~/ai_proj1]$ time ./knight 2 100 100 150 150
terminate called after throwing an instance of 'St9bad_alloc'
  what():  std::bad_alloc
Abort
11.838u 4.800s 0:16.64 99.9%    0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj1]$ time ./knight 3 100 100 150 150
(100,100)(101,102)(102,104)(103,106)(105,107)(107,108)(109,109)(111,110)(112,112)(114,113)(116,114)(117,116)(118,118)(119,120)(120,122)(121,124)(122,126)(123,12
8)(125,129)(127,130)(129,131)(131,132)(132,134)(133,136)(135,137)(137,138)(138,140)(140,141)(142,142)(144,143)(146,144)(148,145)(149,147)(151,148)(150,150)
6449
0.000u 0.002s 0:00.00 0.0%      0+0k 0+8io 0pf+0w
[jyguo@linux1 ~/ai_proj1]$ time ./knight 4 100 100 150 150
(100,100)(101,102)(102,104)(103,106)(105,107)(107,108)(109,109)(111,110)(112,112)(114,113)(116,114)(117,116)(118,118)(119,120)(120,122)(121,124)(122,126)(123,12
8)(125,129)(127,130)(129,131)(131,132)(132,134)(133,136)(135,137)(137,138)(138,140)(140,141)(142,142)(144,143)(146,144)(148,145)(149,147)(151,148)(150,150)
1286347
0.073u 0.014s 0:00.09 88.8%     0+0k 0+8io 0pf+0w
```

In the case, although DFS expanded lots of node on path but it still finish in 0.02 second same as A* and BFS. And why IDS() terminal by accident seem like allocate too much memory. I thing it may cause by my function iteration too and the memory can't allocate anymore. I am not sure why I only use iterate function in IDS, but it may cause the situation.
And also look at IDA* the node numbers expanded to a really huge number, and compare to the excel table upper, the IDS node number could be huger.

Properties Analyzing:

BFS:
Due to the lookup table, every point on the cheeseboard would travel at mot one time, so if the cheeseboard size not change, it could be constant.
Time complexity: O(node number)
Space complexity: O(node number)

DFS:
Similar to BFS, point on cheeseboard would not re-travel in the program.
Time complexity: O(node number)
Space complexity: O(node number)

IDS:
Although with lookup table, some traveled points can be ignored in certain case, the point on table still need to repeat travel in my program. And I guess the iterative function make a huge constant due to the upper data.
Time complexity: $O(b^d)$
Space complexity: O(bd)

A*:
I think in the worst case, like the heuristic function being h(x) = 0, the both complexity is b^d. However, with a good heuristic function the speed would be very quick.
Time complexity: $O(b^d)$
Space complexity: $O(b^d)$

IDA*:
Similar A*, but I also think there are a huge constant compare to A*.
Time complexity: $O(b^d)$
Space complexity: $O(b^d)$

remaining questions

In the experience, the lookup table take a huge place. BFS and DFS not take (b^d) complexity anymore. But some of other algorithm still need lot of computation.

And I still can’t analysis BFS and A* which one is faster in the problem. In small size, five algorithm spend similar time and space to find a path make the analysis harder.

About iteration and loop to achieve these functions, I think there are only constant can affect the time and space. I know that the iteration need to additionally store some pointer and data in cache to memory, but it is not clear in detail.

What I learned

Some program designing, this is my first time to write a shell file.
Complexity analyzing, although I am not sure if I am correct or not.
Experiment making, I do the experiment when I am a freshman. But it still difficult to find out the best algorithm, at least, I tried.

Appendix

Test.sh

```
for case_num in $@
do
	((total_expanded = 0))
	((start = $SECONDS))
#for repeat_time in {0..10}
#do
	for i in {0..8}
	do
		for j in {0..8}
		do
			./knight $case_num 4 4 $i $j >> data.txt
			#$? Cant not represent integer > 255, just for reference
			((total_expanded += $?))
		done
	done
#done
#	((total_time = $SECONDS - start))
#	echo $total_time
#	echo $total_expanded
#	comment used for timer and data alaysis.
Done
```

Makefile

```
all: knight.cpp knight.hpp main.cpp
	g++ -O2 -o knight -g knight.cpp main.cpp
clean_data:
	rm data1.txt data2.txt data3.txt data4.txt data0.txt
clean:
	rm -rf knight
```

Knight.hpp

```
#ifndef KNIGHT_HPP
#define KNIGHT_HPP

#include <iostream>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <algorithm>
#include <queue>
#include <stack>
#include <vector>
#include <fstream>
#define MAPSIZEA1 8


using namespace std;

enum color{
	white, gray, black
};

class node{
	public:
		int x, y;
		node *parent, *child;
		node(){
			parent = NULL;
			child = NULL;
		}
};

class node_valued : public node{
	public:
		int value;
		node_valued *parent, *child;
```

```
        node_valued(){
            parent = NULL;
            child = NULL;
        }

};
class node_valued_deep : public node_valued{
    public:
        int deep;
        node_valued_deep *parent, *child;
        node_valued_deep(){
            parent = NULL;
            child = NULL;
        }
};

void BFS();
void DFS();
void IDS();
void A_star();
void IDA_star();

bool IDS_visit(node*, int);
int h(node*, node*);
bool compare(node_valued*, node_valued*);

#endif
```

Knight.cpp

```
#include "knight.hpp"
extern int expanded;
extern node start, goal;
int x_move[8] = {-2, -1, 1, 2, 2, 1, -1, -2};
int y_move[8] = {1, 2, 2, 1, -1, -2, -2, -1};
//using map to store expended points
//color is just for look clear
```

```
int map[MAPSIZEA1][MAPSIZEA1];

void BFS(){
    for (int j=0; j<MAPSIZEA1; j++) for (int k=0; k<MAPSIZEA1; k++) map[j][k]=0;
    map[start.x][start.y] = black;
    queue<node*> Q;
    Q.push(&start); expanded++;
    if(start.x == goal.x && start.y == goal.y) return ;
    while (!Q.empty()){
        node* handle = Q.front();
        Q.pop();
        for (int i=0; i<8; i++){
            node *tmp = new node;
            tmp->x = handle->x + x_move[i];
            tmp->y = handle->y + y_move[i];
            if( tmp->x >= 0 &&    tmp->x < MAPSIZEA1 &&
                tmp->y >= 0 &&    tmp->y < MAPSIZEA1 &&
                map[tmp->x][tmp->y] == white){
                tmp->parent = handle;
                Q.push(tmp); expanded++;
                if (tmp->x == goal.x && tmp->y == goal.y){
                    node *trace = tmp;
                    while(trace->parent != NULL){
                        trace->parent->child    = trace;
                        trace = trace->parent;
                    }
                    trace = &start;
                    while(trace != NULL){
                        cout << "(" << trace->x << "," << trace->y << ")";
                        trace = trace->child;
                    }
                    cout << endl;
                    return ;
                }
                map[tmp->x][tmp->y] = gray;
            }else delete(tmp);
        }
        map[handle->x][handle->y] = black;
```

```
    }
}

void DFS(){
    for (int j=0; j<MAPSIZEA1; j++) for (int k=0; k<MAPSIZEA1; k++) map[j][k]=0;
    map[start.x][start.y] = black;
    stack<node*> S;
    S.push(&start); expanded++;
    if(start.x == goal.x && start.y == goal.y) return ;
    while (!S.empty()){
        node* handle = S.top();
        S.pop();
        for (int i=0; i<8; i++){
            node *tmp = new node;
            tmp->x = handle->x + x_move[i];
            tmp->y = handle->y + y_move[i];
            if( tmp->x >= 0 &&    tmp->x < MAPSIZEA1 &&
                tmp->y >= 0 &&    tmp->y < MAPSIZEA1 &&
                map[tmp->x][tmp->y] == white){
                tmp->parent = handle;
                S.push(tmp); expanded++;
                if (tmp->x == goal.x && tmp->y == goal.y){
                    node *trace = tmp;
                    while(trace->parent != NULL){
                        trace->parent->child    = trace;
                        trace = trace->parent;
                    }
                    trace = &start;
                    while(trace != NULL){
                        cout << "(" << trace->x << "," << trace->y << ")";
                        trace = trace->child;
                    }
                    cout << endl;
                    return ;
                }
                map[tmp->x][tmp->y] = gray;
            }else delete(tmp);
        }
```

```
            map[handle->x][handle->y] = black;
        }
    }
    int deep_limit;
    void IDS(){
        int end_flag = 0;
        for(int i=1; i<=(MAPSIZEA1-1)*(MAPSIZEA1-1); i++){
            deep_limit = i;
            node *s = new node; expanded++;
            s->x = start.x;
            s->y = start.y;
            for (int j=0; j<MAPSIZEA1; j++) for (int k=0; k<MAPSIZEA1; k++) map[j][k]=0;
            if(IDS_visit(s, 1)){
                node* visit = s;
                while(visit){
                    cout << "(" << visit->x << "," << visit->y << ")";
                    visit = visit->child;
                }
                cout << endl;
                end_flag = 1;
                break;
            }
            if (end_flag) break;
        }
    }

    bool IDS_visit(node *s, int deep){
        if(s->x == goal.x && s->y == goal.y){
            return 1;
        }else if (deep == deep_limit) return 0;
        map[s->x][s->y] = deep;
        int in_route = 0;
        for (int i=0; i<8; i++){
            node *tmp = new node;
            tmp->x = s->x + x_move[i];
            tmp->y = s->y + y_move[i];
            if( tmp->x >= 0 &&    tmp->x < MAPSIZEA1 &&
                tmp->y >= 0 &&    tmp->y < MAPSIZEA1 &&
```

```
                (deep < map[tmp->x][tmp->y] ||
                map[tmp->x][tmp->y] == white)){
                expanded++;
                tmp->parent = s;
                if (IDS_visit(tmp, deep+1)){
                        in_route = 1;
                        s->child = tmp;
                        }
                }
        }
    if(!in_route) {
        delete s;
    }
    return in_route;
}
int h(node *n1, node *n2){
    int dis_x = abs(n1->x - n2->x);
    int dis_y = abs(n1->y - n2->y);
//original
    return (dis_x + dis_y)/3;
//self define 01
    return max((dis_x + dis_y)/3, max(dis_y/2, dis_x/2));
}

bool compare(node_valued* n1, node_valued* n2){
    return (n1->value > n2->value);
}

void A_star(){
    for (int j=0; j<MAPSIZEA1; j++) for (int k=0; k<MAPSIZEA1; k++) map[j][k]=0;
    node_valued *first = new node_valued; expanded++;
    first->x = start.x;
    first->y = start.y;
    first->value = h(&start, &goal);

    vector<node_valued*> considered;
    considered.push_back(first);
    make_heap(considered.begin(), considered.end(), compare);
```

```
	while(!considered.empty()){
		node_valued *handle = considered.front();
		pop_heap(considered.begin(), considered.end(), compare);
		if(handle->x == goal.x && handle->y == goal.y){
			while(handle->parent){
				handle->parent->child = handle;
				handle = handle->parent;
			}
			while(handle){
				cout << "(" << handle->x << "," << handle->y << ")";
				handle = handle->child;
			}
			cout << endl;
			break;
		}
		considered.pop_back();
		for(int i=0; i<8; i++){
			node_valued *tmp = new node_valued;
			tmp->x = handle->x + x_move[i];
			tmp->y = handle->y + y_move[i];
			tmp->parent = handle;
			if( tmp->x >= 0 &&	tmp->x < MAPSIZEA1 &&
				tmp->y >= 0 &&	tmp->y < MAPSIZEA1){
				expanded++;
				tmp->value = handle->value - h(handle, &goal) + h(tmp, &goal) +
1;
					if(map[tmp->x][tmp->y] == white || map[tmp->x][tmp->y] >
tmp->value){
						map[tmp->x][tmp->y] = tmp->value;
						considered.push_back(tmp);
						push_heap(considered.begin(), considered.end(),
compare);
					}else{
						delete tmp;
					}
//				considered.push_back(tmp);
//				push_heap(considered.begin(), considered.end(), compare);
```

```
                }else delete tmp;
            }
        }
}

void IDA_star(){
        for(int deep=0; deep<=(MAPSIZEA1-1)*(MAPSIZEA1-1); deep++){
            for (int j=0; j<MAPSIZEA1; j++) for (int k=0; k<MAPSIZEA1; k++) map[j][k]=0;
            node_valued_deep *first = new node_valued_deep; expanded++;
            first->x = start.x;
            first->y = start.y;
            first->value = h(&start, &goal);
            first->deep = 0;

            vector<node_valued_deep*> considered;
            considered.push_back(first);
            make_heap(considered.begin(), considered.end(), compare);

            while(!considered.empty()){
                node_valued_deep *handle = considered.front();
                pop_heap(considered.begin(), considered.end(), compare);
                if(handle->x == goal.x && handle->y == goal.y){
                    while(handle->parent){
                        handle->parent->child = handle;
                        handle = handle->parent;
                    }
                    while(handle){
                        cout << "(" << handle->x << "," << handle->y << ")";
                        handle = handle->child;
                    }
                    cout << endl;
                    return ;
                }
                considered.pop_back();
                for(int i=0; i<8; i++){
                    node_valued_deep *tmp = new node_valued_deep;
                    tmp->x = handle->x + x_move[i];
                    tmp->y = handle->y + y_move[i];
```

```
                        tmp->parent = handle;
                        tmp->deep = handle->deep + 1;
                        if(     tmp->deep <= deep &&
                                tmp->x >= 0 &&    tmp->x < MAPSIZEA1 &&
                                tmp->y >= 0 &&    tmp->y < MAPSIZEA1 ){
                                expanded++;
                                tmp->value = handle->value - h(handle, &goal) + h(tmp,
&goal) + 1;

                                if(map[tmp->x][tmp->y] == white || map[tmp->x][tmp->y] >
tmp->value){
                                        map[tmp->x][tmp->y] = tmp->value;
                                        considered.push_back(tmp);
                                        push_heap(considered.begin(), considered.end(),
compare);
                                }else{
                                        delete tmp;
                                }
                        }else delete tmp;
                }
                //delete handle;
        }
    }
}
```

Main.cpp

```
#include "knight.hpp"

int expanded = 0;
node start, goal;

int main(int argc, char** argv){
    if(argc != 6){
        cout << "Usage: <ALGORITHSM TYPE> <STARTING X> <STARTING Y> <GOAL
X> <GOAL Y>."<< endl;
        exit(1);
```

```
}
int type = atoi(argv[1]);
start.x = atoi(argv[2]);
start.y = atoi(argv[3]);
goal.x = atoi(argv[4]);
goal.y = atoi(argv[5]);
if(type < 0 || type > 4
	|| start.x < 0 || start.x >= MAPSIZEA1
	|| start.y < 0 || start.y >= MAPSIZEA1
	|| goal.x < 0 || goal.x >= MAPSIZEA1
	|| goal.y < 0 || goal.y >= MAPSIZEA1 ){
	cout << "Invaild input." << endl;
	exit(2);
}
switch (type){
	case 0:
		BFS();
		break;
	case 1:
		DFS();
		break;
	case 2:
		IDS();
		break;
	case 3:
		A_star();
		break;
	case 4:
		IDA_star();
		break;
}
ofstream fout;
char str[] = "datax.txt";
str[4] = argv[1][0];
fout.open (str, ios::app);
fout << expanded << endl;
cout << expanded << endl;
return 0;
```

```
}
```