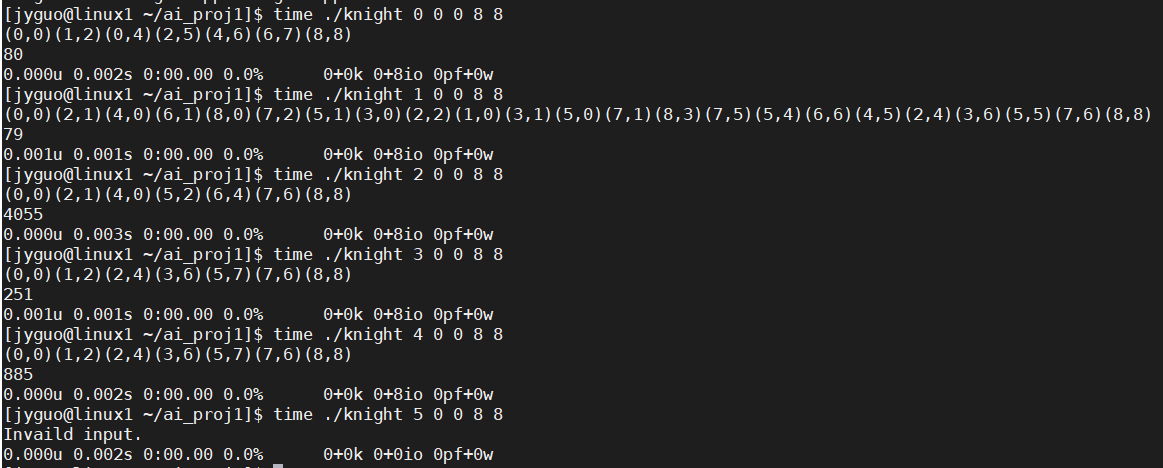
Introduction to AI: Programming Assignment #1

0612129 郭家佑

## Basic demo:

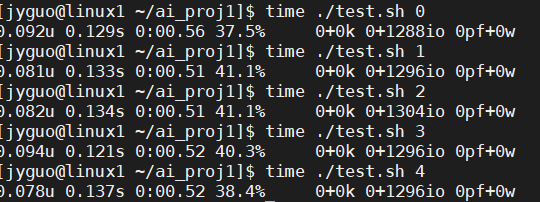


In the experience, all test base on the code in appendix, and the excusable code’s format is “./knight <ALGORITHSM TYPE> <STARTING X> <STARTING Y> <GOAL X> <GOAL Y>", and it would dump the path coordinate on the screen. About the <ALGORITHSM TYPE> field, 0 is for BFS, 1 is for DFS, 2 is for IDS, 3 is 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.

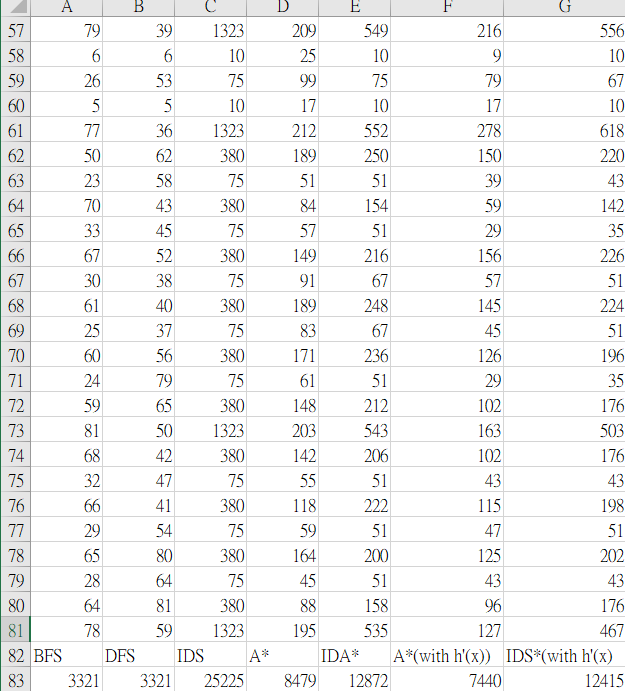
## Experience result:

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.



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.



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.

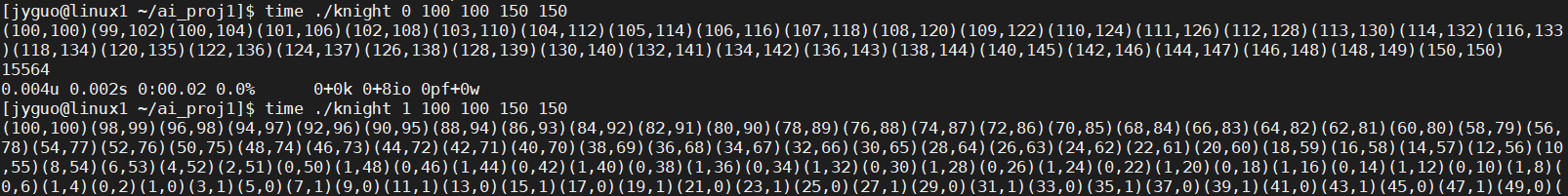
I make the h’(x) being max(|dx|+|dy|)/3, |dx|/2, |dy|/2). Stand for, a step only can move only 2 units’ distance on one coordinate.

Like the following picture:

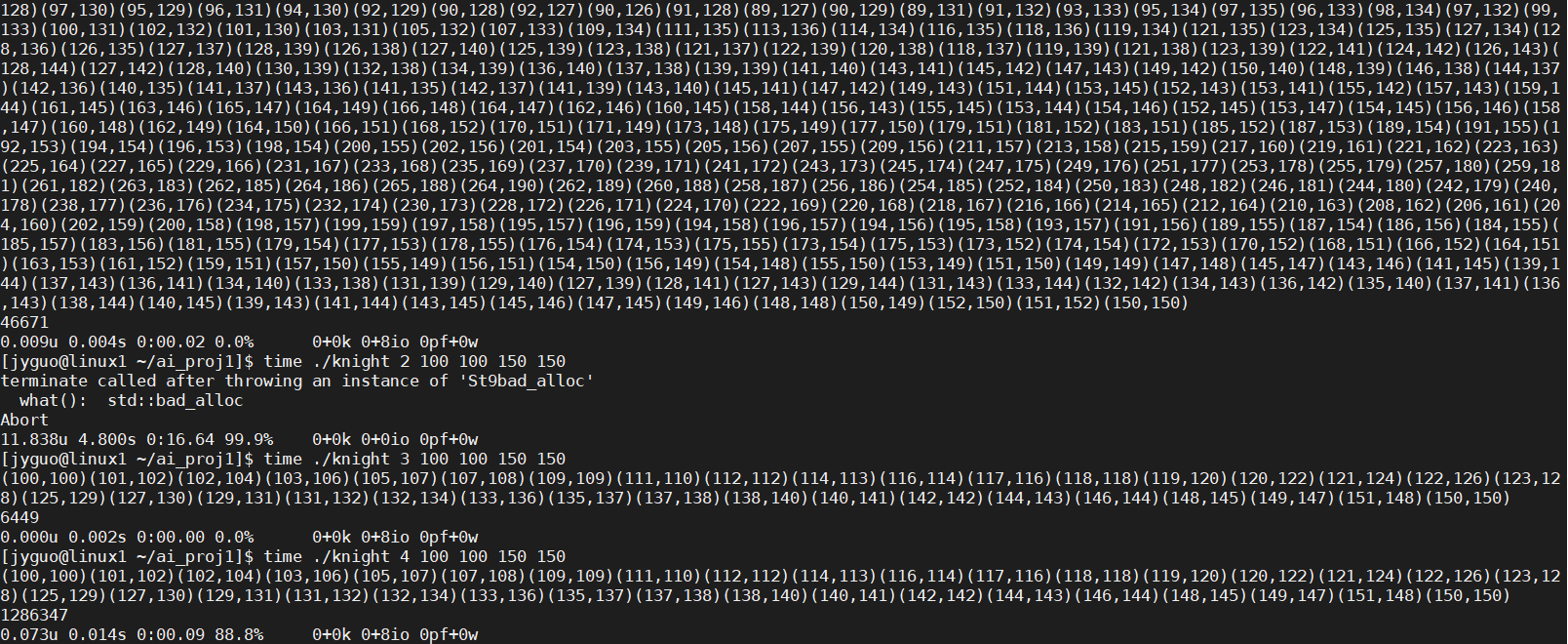


The nodes had been used also reduced showed in the data. Maybe there are some better heuristic function, but I still can’t find out.

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”.



……(a lot of node)



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(bd)

Space complexity: O(bd)

A\*:

I think in the worst case, like the heuristic function being h(x) = 0, the both complexity is bd. However, with a good heuristic function the speed would be very quick.

Time complexity: O(bd)

Space complexity: O(bd)

IDA\*:

Similar A\*, but I also think there are a huge constant compare to A\*.

Time complexity: O(bd)

Space complexity: O(bd)

## remaining questions

In the experience, the lookup table take a huge place. BFS and DFS not take (bd) 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;

}