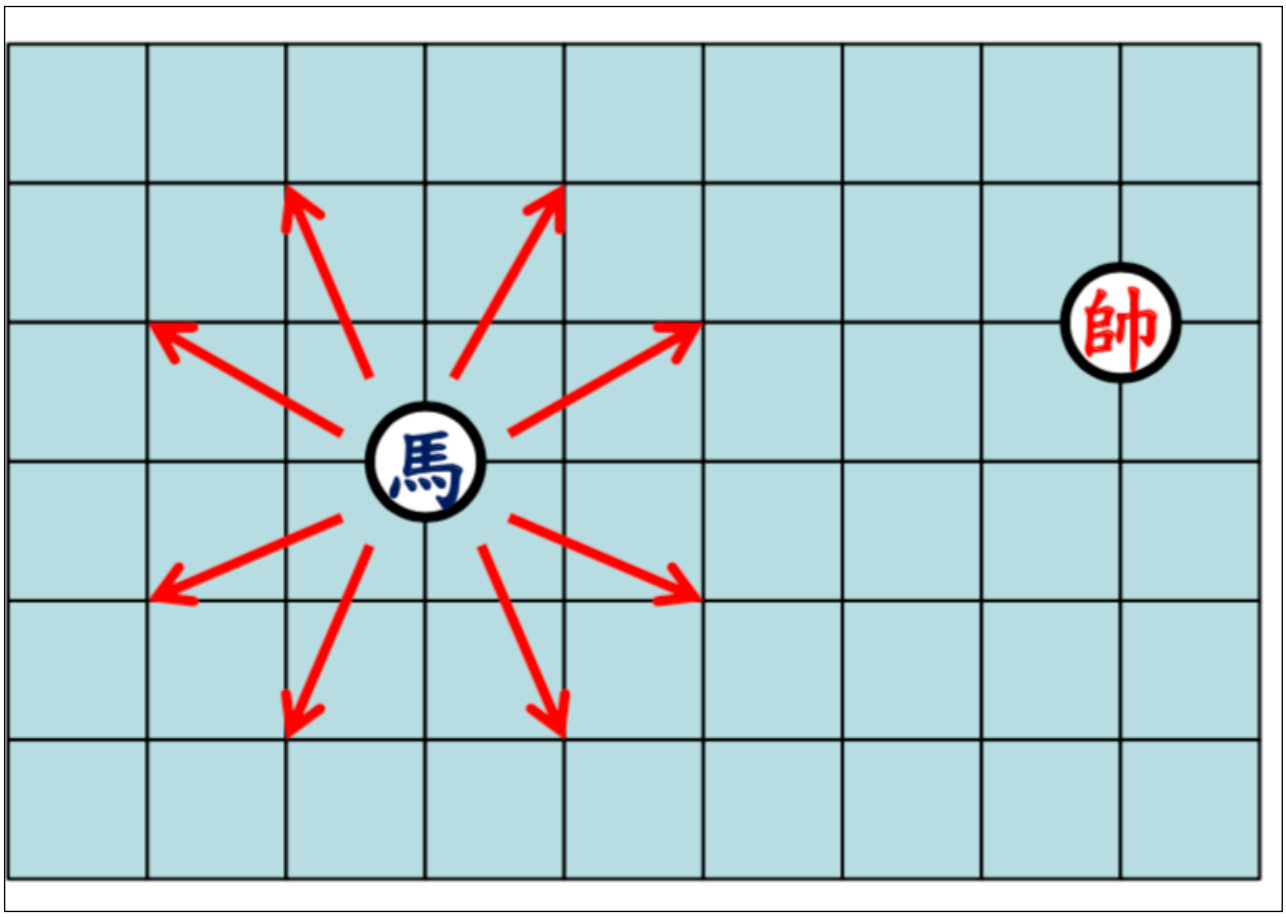
Search Algorithms

Introduction to Artificial Intelligence Project 1

0712238 Yan-Tong Lin - April 10, 2020



Task Description

Find a path for a knight from S to T where S, T is positions on 8*8 chess board. Implement BFS, DFS, IDDFS, A*, IDA* to do this.

Implementation Details

Language and Tools:

1. C++/shell script(.sh)
2. Editor: VS Code
3. Version Control: git
4. Report: Mac OS Pages

OOP style implementation for solvers

1. custom_header.hpp contains required classes and features for my program, including Solver base class and some macro definitions
2. <algorithm name>_solver.hpp covers different algorithms
3. Pointers to instances of solver are stored in vector<Solver> solvers
4. Interactive testing of search algorithms can be done by compilation of game.cpp and execution of output execution file
5. Automatic tests with shell scripts(exp.sh, tc_generate.sh, exp.cpp)

Experiments

Performance Evaluation

For evaluation of performance of algorithms,
I choose two features described as follow:

- **Node_expanded**
  - The total nodes expanded during the **whole procedure** of an algorithm.
  - As a measure of **time complexity**
- **Max_node_expanded**
  - The maximum value of nodes expanded **at the same time.**
  - As a measure of **space complexity**

The advantage of using these two features instead directly record the actual time/space spent is the calculated values are direct reflection of how good the algorithm is,

ignoring some other unrelated issues(ex: stack/recursion implementation of dfs function can cost different time and space).

Experiment Designs

- Compare **performance of different algorithms** in 8*8 board with random-generated test data(experiment 1)
  - Test cases = 150 random Ss, Ts on 8*8 board, evaluated with mean, medium and maximum node_expanded/max_node_expanded
- Compare **growth of time and space** to **board size** with random-generated test data, for each algorithm(experiment 2)
- Compare **implementation of styles of IDDFS's DFS**(experiment 3)

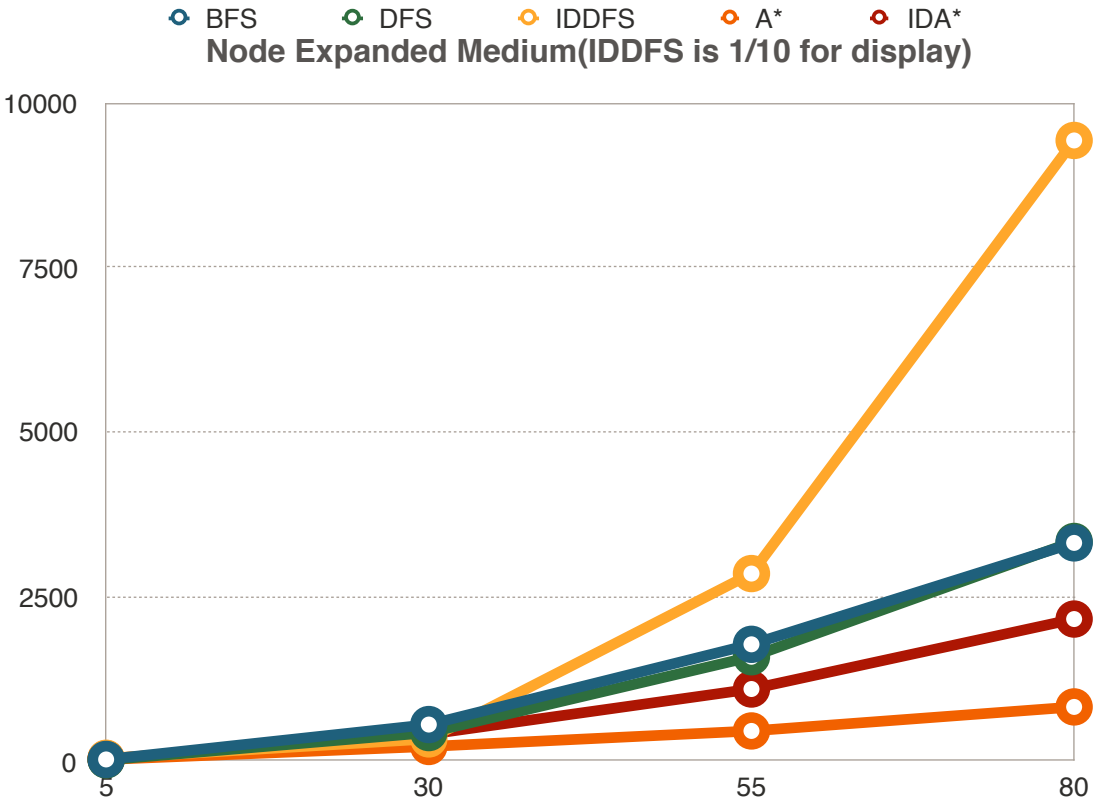
Experiment Results

1. performance of different algorithms in 8*8 board(150 random test cases)

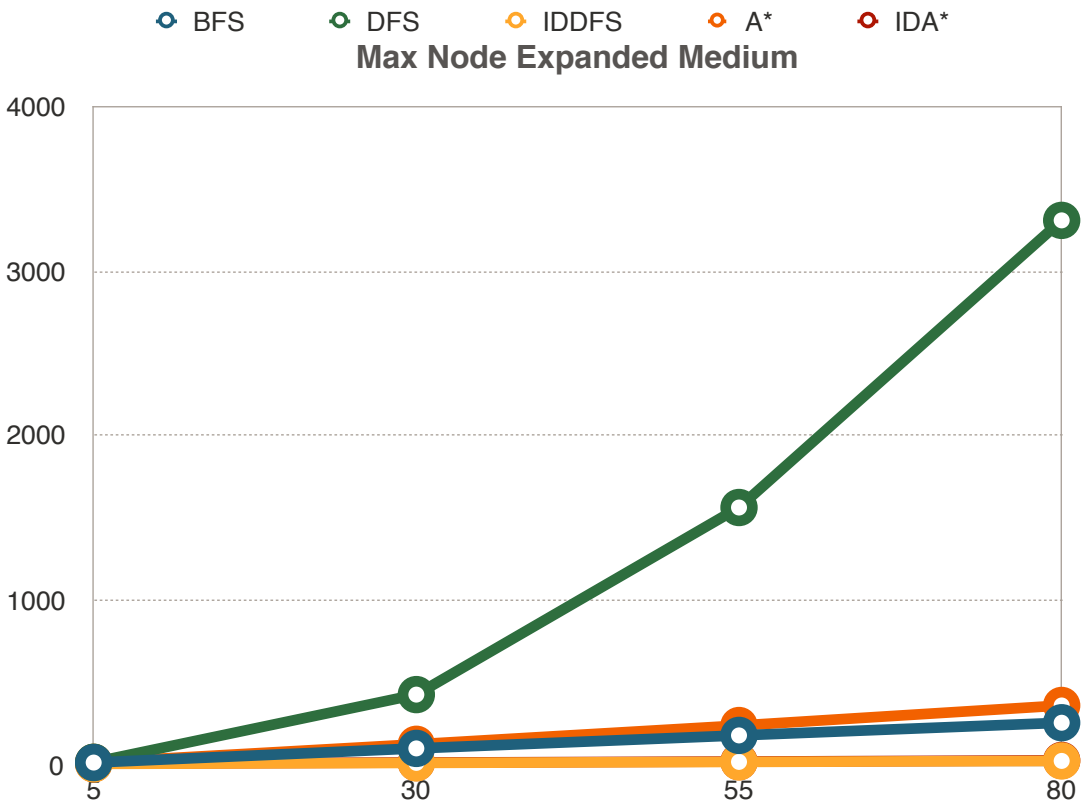
ALGO:time/memory	Mean	Medium	Maximum
BFS	50.2567/27.3467	58/29	64/35
DFS	31.22/29.9267	31/30	64/56
IDDFS	123.193/4.76667	120/5	321/7
A*	38.7267/26.1133	40/28	62/41
IDA*	63.1267/6.03333	59/6	184/10

1. DFS value is because of it is bound to have a "Knight Tour for 8*8 board.
2. The comparison between BFS/A*, IDDFS/IDA* can show heuristic is doing its job
3. Notice that IDDFS memory ~= IDA memory is for their max depth ~= shortest path length ~= max node expanded at the same time
4. IDDFS and IDA* have higher time constant for adding up previous nodes.

2. performance of different algorithms in N*N board



1. We can see IDDFS seems to have a lot higher time complexity compared to other algorithm, the guess of O(n^4) may be right. (94274 >> 3313, but not exponential)
2. A* and IDA* generally visit less total node than other algorithms, which is even more significant when the search space is large enough.
3. BFS and DFS have similar time complexity O(#position of board).
4. Overall, BFS, DFS, A* : O(#position of board), IDDFS : O(square of positions) with small constant(conjecture), IDA*: O(square of positions) with even smaller constant.



1. DFS is O(# positions of board) for Knight Tour again.
2. DFS(~3000up) >> BFS, DFS(~300) > IDDFS, IDA*(22, 25) in case N=80
3. For A* and BFS, since branching factor rapidly shrinks from 8 to small value for this task, and the depth is not too large , the max node expanded is not as bad as I thought(I checked my code again for ensuring this is not because of bug). And actually, max node expanded is actually bounded by O(#states expanded before reaching T).

3. implementation of styles of IDDFS's DFS, see discussion for more details

Experiment is conducted on test case:
Board = 39*39 (for no knight tour)
S, T = (0, 0), (38, 37)

BFS, baseline(AC)

```
input solver id:
0
input start coordinate(0-indexed)(ex: 0 7):
0 0 38 37
input destination coordinate(0-indexed)(ex: 0 7):
tha path found by algorithm BFS is of length 26
the past is listed as follow:
(0, 0) (1, 2) (2, 4) (3, 6) (4, 8) (5, 10) (6, 12) (7, 14) (8, 16) (9, 18)
(10, 20) (11, 22) (12, 24) (14, 25) (16, 26) (18, 27) (20, 28) (22, 29) (24, 30) (26, 31)
(28, 32) (30, 33) (32, 34) (34, 35) (36, 36) (38, 37)
1521 137
1521 137
```

IDDFS, DFS without considering relaxation(WA)

```
input solver id:
2
input start coordinate(0-indexed)(ex: 0 7):
0 0
input destination coordinate(0-indexed)(ex: 0 7):
38 37
tha path found by algorithm IDDFS is of length 36
the past is listed as follow:
(0, 0) (1, 2) (2, 4) (3, 6) (4, 8) (5, 10) (6, 12) (7, 14) (8, 16) (9, 18)
(10, 20) (11, 22) (12, 24) (13, 26) (14, 28) (15, 30) (16, 32) (17, 34) (18, 36) (19, 38)
(20, 36) (21, 38) (22, 36) (23, 38) (24, 36) (25, 38) (26, 36) (27, 38) (28, 36) (29, 38)
(30, 36) (31, 38) (32, 36) (34, 37) (36, 38) (38, 37)
20751 37
20751 37
```

IDDFS, DFS with loop detection(TLE)

```
input destination coordinate(0-indexed)(ex: 0 7):
maxd: 1
maxd: 2
maxd: 3
maxd: 4
maxd: 5
maxd: 6
maxd: 7
maxd: 8
maxd: 9
maxd: 10
maxd: 11
maxd: 12
TLE break at 10 secondsTime taken by the solver is :36.499671 sec
Segmentation fault: 11
```

IDDFS, DFS with relaxation detection(AC, with less memory cost(O(# path)), higher total expanded node(close to my guess is O(n^4))

```
id 6:, algorithm :IDDFS2
input solver id:
6
input start coordinate(0-indexed)(ex: 0 7):
0 0 38 37
input destination coordinate(0-indexed)(ex: 0 7):
tha path found by algorithm IDDFS2 is of length 26
the past is listed as follow:
(0, 0) (1, 2) (2, 4) (3, 6) (4, 8) (5, 10) (6, 12) (7, 14) (8, 16) (9, 18)
(10, 20) (11, 22) (12, 24) (14, 25) (16, 26) (18, 27) (20, 28) (22, 29) (24, 30) (26, 31)
(28, 32) (30, 33) (32, 34) (34, 35) (36, 36) (38, 37)
73133 27
73133 27
```

IDA*, DFS with relaxation (AC, better than IDDFS in case of time, by a lot)

```
id 7:, algorithm :IDA2
input solver id:
7
input start coordinate(0-indexed)(ex: 0 7):
0 0 38 37
input destination coordinate(0-indexed)(ex: 0 7):
tha path found by algorithm IDA2 is of length 26
the past is listed as follow:
(0, 0) (1, 2) (2, 4) (3, 6) (4, 8) (5, 10) (6, 12) (7, 14) (8, 16) (9, 18)
(10, 20) (11, 22) (12, 24) (14, 25) (16, 26) (18, 27) (20, 28) (22, 29) (24, 30) (26, 31)
(28, 32) (30, 33) (32, 34) (34, 35) (36, 36) (38, 37)
410 27
410 27
```

Facts about DFS algorithm:
There are actually multiple strategy when it comes to adding new node to frontier

1. Discard when in frontier or explored set
2. Discard when cannot do “relaxation” or create “loop”
   - When adopting another path which is shorter to the original path to the node, we call this a relaxation to the node. If the definition is not clear for you, please refer to textbooks’ sections about shortest path.

Further discussion is covered in Discussion section.

Discussion

DFS implementation in IDDFS task

1. Discard when in **frontier or explored set**
   - Will **not necessarily find shortest path** for IDDFS/IDA*
     - Since the final position maybe reachable in depth limit if relaxation if allowed.
   - Time complexity is **O(# of states)**, in this case O(n^2), since every state is explored exactly once.(Here state definition is position of knight, which is independent of path taken, depth, etc.)
2. Discard when **cannot do "relaxation" or form loop**
   - **Will find shortest path for IDDFS/IDA***
   - Time complexity will change, since the original complexity is O(# of states) is based on no revisit.
   - **Implementation and conjecture about time complexity**
     - **Detect loop**(only check visited points on current path, my implementation)
       - I believe that it is of **exponential complexity** for it is actually doing enumeration of all paths.
       - As depth getting bigger, branching factor is getting smaller
       - Since the possible next move is bounded by 8 for knight
       - Let N be # of positions: the complexity is O(N*(N/b1)*(N/b2)…1) ~= **O(N!/8^N)**?
       - On 39*39 board, max depth = 11, is limit of this version of IDDFS(wait for 10 seconds up, still no result)
     - **Check relaxation** availability
       - I am not quite sure about the time complexity in this scenario.
       - One guess is that we consider the new sate defined as {depth, position}, there are O(BoardSize^2) = O(n^4) such sates, relaxation can be seen as transition between new states, each time decrease depth by at least 1, means there are at most O(n^2)/O(1) relaxation for each position => **O(n^4)**.
     - According to the experiment 3, I believe that my inference on time complexity is correct or close to the answer.(**loop detection => exponential time, relaxation detection => polynomial time**)

DFS algorithm's relation to the "Knight Tour" problem

1. There is two implementation of DFS
2. Actually, in some boards, for DFS function, it is always possible to go from any S to any T **without backtracking**. (reference: https://en.wikipedia.org/wiki/Knight%27s_tour#Existence)(original paper: Allen J. Schwenk (1991). "Which Rectangular Chessboards Have a Knight's Tour?" (PDF). *Mathematics Magazine*: 325–

332.) **In such cases, DFS time and space complexity is both O(d) where O(d) is actually O(n^2), i.e. the size of whole board**.

3. For the cases in which there is no knight tour, it's still plausible to find a path for DFS, but for IDDFS, it requires some modification(to the version with "relaxation"), or the path found by algorithm won't be shortest path.

4. By the way, these facts are discovered for **DFS didn't get TLE but IDDFS get TLE when board side length - n is even cases**.

Conclusion

1. The complexity measurement we take generally reflects the conjecture I made.
2. A* and IDA* with good heuristic can dramatically reduce nodes expanded by providing "good explore direction".
3. Modified version DFS implementation is required for graph search with loop for IDDFS to find shortest path.

Remaining Questions and Future Work

1. Do both evaluation of system time/memory usage and node_expanded/ max_node_expanded. Compare their correlation.
2. Further discussion about IDDFS tine complexity on task with loop, should provide proof.
3. Try other heuristic functions.
4. Make the test data more robust (cover some special cases).
5. Test for extreme data(prime, 1*1e9, 2*1e9, extremely large umber)
6. Try bi-directional search algorithms.
7. Implement parallel searching

Appendix A : Code

game,cpps

```
#include "custom_header.hpp"
#include "bfs_solver.hpp"
#include "dfs_solver.hpp"
#include "iddfs_solver.hpp"
#include "astar_solver.hpp"
#include "idastar_solver.hpp"
#include "dfsv2_solver.hpp"
#include "iddfs_v2_solver.hpp"
#include "idastar_v2_solver.hpp"

signed main()
{
    //interaction with cerr
    cerr << "Introduction to AI 2020 spring HW1 by yan-tong lin"
<< endl;
    cerr << "solver type list" << endl;
    //list of solver names
    vector<Solver*> solvers;
    solvers.pb(new BFS());
    solvers.pb(new DFS());
    solvers.pb(new IDDFS());
    solvers.pb(new Astar());
    solvers.pb(new IDA());
    solvers.pb(new DFS2());
    solvers.pb(new IDDFS2());
    solvers.pb(new IDA2());
    N = 10; //effective test by dfs
    cerr << "input board size: ";
    cin >>  N;
    //iddfs test segment fault expected?, shouldnt it be TLE?
    rep(i, 0, solvers.size()) cerr << "id " << i << ":,
algorithm :" << solvers[i]->name << endl;
    cerr << "input solver id:" << endl;
    int id; int sx, sy, tx, ty;
```

```
    cin >> id;
    while(!(id>=0 && id < solvers.size()))
    {
        cerr << "input solver id in range:" << endl;
        cin >> id;
    }
    solvers[id]->init();
    cerr << "input start coordinate(0-indexed)(ex: 0 7):" << endl;
    cin >> sx >> sy;
    cerr << "input destination coordinate(0-indexed)(ex: 0 7):" <<
endl;
    cin >> tx >> ty;
    auto dat = solvers[id]->solve(mp(sx, sy), mp(tx, ty));
    vector<pii>& path = dat.X;
    int nodes_expanded = dat.Y;
    cerr << "tha path found by algorithm " << solvers[id]->name <<
" is of length " << path.size() << endl;
    cerr << "the past is listed as follow:\n";
    cerr << path;
    cerr << nodes_expanded << " " << solvers[id]-
>max_node_expanded << endl;

    //for real output
    cout << nodes_expanded << " " << solvers[id]-
>max_node_expanded << endl;

    return 0;
}
```

custom_header.hpp

```
#ifndef CUSTOM_H
#define CUSTOM_H

//headers and namespaces
#include <bits/stdc++.h>
//#include "matplotlibcpp.h", not available on Mac(or complicated)
using namespace std;
```

```
//hyper params
int N = 8;
#define C_TL 10.0 //time limit

//useful macros
#define rep(i, s, t) for(int i = s, _t = (t); i < _t; i++)
#define pb push_back
#define debug(x) std::cout << #x << ": " << x << endl

template<class T>
using Board = vector<vector<T>>; //elastic, init size with init(),
which uses mutable N

//Timer
// S: start(hidden), T : end (hidden), D : calc delta by hidden, P
print car, R : all wrapped
// C: end and calc by end
#define TIMER_S auto ___st_time =
chrono::high_resolution_clock::now();
#define TIMER_T auto ___ed_time =
chrono::high_resolution_clock::now();
#define TIMER_D(var) double var =
chrono::duration_cast<chrono::nanoseconds>(___ed_time -
___st_time).count(); var *= 1e-9;
#define TIMER_C(var) TIMER_T TIMER_D(var)
#define TIMER_P(var) cout << "Time taken by the solver is :" <<
fixed << var << setprecision(9);  cout << " sec" << endl;
#define TIMER(xxx) TIMER_S xxx TIMER_C(_tt) TIMER_P(_tt)

//pair<int, int> make useful
#define pii pair<int, int>
#define X first
#define Y second
#define cor2(point) point.X, point.Y
#define cor(point) point.X][point.Y
#define mp make_pair
pii operator+(const pii&x, const pii&y) { return mp(x.X+y.X,
x.Y+y.Y);}
```

```
pii operator-(const pii&x, const pii&y) { return mp(x.X-y.X, x.Y-
y.Y);}
int manhattan_distance(const pii&x, const pii&y){return abs(x.X-
y.X) + abs(x.Y-y.Y);}

//moving
pii dxdy[8] =
{
    {1,2},{-1,2},{1,-2},{-1,-2},
    {2,1},{-2,1},{2,-1},{-2,-1}
};

bool inrange(pii x)
{
    return (x.X>=0) && (x.Y>=0) && (x.X<N) && (x.Y<N);
}

//overload output method for specific types
std::ostream& operator<<(std::ostream& os, pii& p) //tested
{
    os << "("<<  p.first << ", " <<  p.second << ")";
    return os;
}

std::ostream& operator<<(std::ostream& os, vector<pii>& path) //
tested
{
    rep(i, 0, path.size()) os <<  path[i] << " \n"[i==path.size()-
1 || (i+1)%10==0];
    return os;
}

template<class T>
std::ostream& operator<<(std::ostream& os, Board<T>& b) //tested
{
    rep(i, 0, N)
    {
        rep(j, 0, N) os << b[i][j] << " ";
        os << endl;
```

```
    }
    return os;
}

//below is Solver, parent of different solvers
class Solver
{
private:
public:
    string name;
    int max_node_expanded;
    int cur_node_expanded;
    int node_expanded;
    int node;
    Board<pii> vis;
    //TL is time limit
    virtual pair<vector<pii>, int> solve(pii x, pii y, double
TL=C_TL) = 0;
    void init();
    void print();
    void construct_path(vector<pii> &path, pii t);
};

void Solver::construct_path(vector<pii> &path, pii t)
{
    pii cur = t;
    while(1)
    {
        if(vis[cor(cur)] == cur) { path.pb(cur); break;}
        path.pb(cur);
        cur = vis[cor(cur)];
    }
    reverse(path.begin(), path.end());
    return;
}

void Solver::init() //init visboard
{
    vis = vector<vector<pii>>(N, vector<pii>(N)); //elastic board
```

```
    rep(i, 0, N) fill(vis[i].begin(), vis[i].end(), pii(-1, -1));
    //cout << name << " solver initialized." << endl;
}

void Solver::print()
{
    cout << "printing " <<   name << " board\n";
    cout << vis;
}


#endif
```

bfs_solver.hpp

```
#ifndef BFS_H
#define BFS_H

#include "custom_header.hpp"

class BFS : public Solver
{
public:
    //string name; this will cause the name be empty string
    BFS();
    pair<vector<pii>, int> solve(pii x, pii y, double TL);
};

BFS::BFS()
{
    name = "BFS";
}

pair<vector<pii>, int> BFS::solve(pii s, pii t, double TL)
{
    node_expanded = 0;
    cur_node_expanded = 0;
    max_node_expanded = 0;
```

```
    vector<pii> path;
    queue<pii> q;
    node_expanded++;
    cur_node_expanded++;
    q.push(s);
    vis[cor(s)] = s;
    while(!q.empty())
    {
        cur_node_expanded--;
        pii cur = q.front(); q.pop();
        if(cur == t) break;
        for(auto &di : dxdy)
        {
            pii nxt = cur + di;
            if(!inrange(nxt) || vis[cor(nxt)] != pii(-1,-1))
continue;
            //if(nxt == t)
            node_expanded++;
            cur_node_expanded++;
            max_node_expanded = max(max_node_expanded,
cur_node_expanded);
            q.push(nxt);
            vis[cor(nxt)] = cur;
        }
    }
    //print();
    Solver::construct_path(path, t);

    return mp(path, node_expanded);
}

#endif
```

dfsv2_solver.hpp

```
#ifndef DFS2_H
#define DFS2_H

//dfs with relaxation check
```

```cpp
#include "custom_header.hpp"

class DFS2 : public Solver
{
private:
    int INF;
public:
    //string name; this will cause the name be empty string
    DFS2();
    bool dfs2(int d, pii s, pii t);
    vector<pii> path;
    vector<vector<int>> dep; //relaxtion based
    pair<vector<pii>, int> solve(pii x, pii y, double TL);
};

DFS2::DFS2()
{
    name = "DFS2_v2";
}

bool DFS2::dfs2(int d, pii s, pii t) //s = current, p = from, d =
current depth, t =
{
    bool ret = false;
    if(s == t) return true;
    for(auto &di : dxdy)
    {
        pii nxt = s + di;
        //nxt is not in range or no relexation, init d = INF, so
if no visit will explore
        //dep != INF => no relaxation version(visit once)
        //dep <= d+1 => relaxation version
        if(!inrange(nxt)||dep[cor(nxt)]!=INF) continue;
        path.pb(nxt);
        node_expanded++;
        cur_node_expanded++;
        max_node_expanded = max(max_node_expanded,
cur_node_expanded);
```

```
        dep[cor(nxt)] = d+1;
        ret |= dfs2(d+1, nxt, t);
        if(ret) break; //success, no pop_back this path
        path.pop_back();
        cur_node_expanded--;
    }
    return ret;
}

pair<vector<pii>, int> DFS2::solve(pii s, pii t, double TL)
{
    node_expanded = 0;
    max_node_expanded = 0;
    cur_node_expanded = 0;
    path.clear();
    INF = N*N + N;
    dep = vector<vector<int>>(N, vector<int>(N, INF));
    //init();
    //start dfs
    node_expanded++;
    cur_node_expanded++;
    max_node_expanded = max(max_node_expanded, cur_node_expanded);
    dep[cor(s)] = 0;
    path.pb(s);
    dfs2(0, s, t);
    return mp(path, node_expanded);
}
#endif
```

iddfs2_solver.hpp

```
#ifndef IDDFS2_H
#define IDDFS2_H

#include "custom_header.hpp"

class IDDFS2 : public Solver
{
```

```
private:
public:
    vector<pii> path;
    vector<vector<int>> dep; //relaxtion based
    //string name; this will cause the name be empty string
    IDDFS2();
    bool dfs2(int d, pii s, pii t, int maxd);
    void init_dfs();
    pair<vector<pii>, int> solve(pii x, pii y, double TL);
};

IDDFS2::IDDFS2()
{
    name = "IDDFS2";
}

bool IDDFS2::dfs2(int d, pii s, pii t, int maxd) //s = current, p
= from, d = current depth, t =
{
    bool ret = false;
    if(d > maxd) return false;
    if(s == t) return true;
    for(auto &di : dxdy)
    {
        pii nxt = s + di;
        //nxt is not in range or no relexation, init d = INF, so
if no visit will explore
        if(!inrange(nxt)||dep[cor(nxt)] <= d+1) continue;
        path.pb(nxt);
        node_expanded++;
        cur_node_expanded++;
        max_node_expanded = max(max_node_expanded,
cur_node_expanded);
        dep[cor(nxt)] = d+1;
        ret |= dfs2(d+1, nxt, t, maxd);
        if(ret) break; //success, no pop_back this path
        path.pop_back();
        cur_node_expanded--;
    }
```

```
    return ret;
}

void IDDFS2::init_dfs()
{
    path.clear();
    int INF = N*N + N;
    dep = vector<vector<int>>(N, vector<int>(N, INF));
}


pair<vector<pii>, int> IDDFS2::solve(pii s, pii t, double TL)
{
    node_expanded = 0;
    max_node_expanded = 0;
    cur_node_expanded = 0;
    int maxd = 0;
    init_dfs();
    path.pb(s);
    node_expanded++;
    cur_node_expanded++;
    max_node_expanded = max(max_node_expanded, cur_node_expanded);
    dep[cor(s)] = 0;
    while(!dfs2(0, s, t, maxd) && maxd <= 100000)
    {
        init_dfs();
        dep[cor(s)] = 0;
        path.pb(s);
        maxd++;
    }
    //assert(path.size() == maxd+1); // s -d++- 1 - 2 - 3 - d++ t
    return mp(path, node_expanded);
}

#endif
```

astar_solver.hpp

```cpp
#ifndef Astar_H
#define Astar_H

#include "custom_header.hpp"

class Astar : public Solver
{
private:
    std::function<int(const pii&, const pii&)> h;
public:
    //string name; this will cause the name be empty string
    Astar();
    pair<vector<pii>, int> solve(pii x, pii y, double TL);
    //int h1(pii x); //heuristic
};

Astar::Astar()//std::function<int(pii)> _h)
{
    h = [&](const pii &s, const pii &t){
        return manhattan_distance(s, t)/3;
    };
    name = "Astar";
}

pair<vector<pii>, int> Astar::solve(pii s, pii t, double TL)
{
    TIMER_S
    node_expanded = 0;
    cur_node_expanded = 0;
    max_node_expanded = 0;
    vector<pii> path;
    priority_queue<pair<int, pair<int, int>>> pq; //bigger first
out
    //note: negate the sum of heuristic + distance now, to
arvhieve smaller first out
    //origially, nxt_h = cur_h - h(cur, t) + 1 + h(nxt, t)
    //now, nxt_h = cur_h - 1 + h(cur, t) - h(nxt, t)
```

```
    pq.push(mp(-h(s,t), s));
    node_expanded++;
    cur_node_expanded++;
    vis[cor(s)] = s;
    while(!pq.empty())
    {
        pair<int, pii> cur = pq.top(); pq.pop();
        cur_node_expanded--;
        if(cur.Y == t) break;
        for(auto &d : dxdy)
        {
            pii nxtp = cur.Y + d;
            if(!inrange(nxtp) || vis[cor(nxtp)] != pii(-1,-1))
continue;
            //if(nxt == t)
            pair<int, pii> nxt = mp(cur.X-1+h(cur.Y, t)-h(nxtp,
t), nxtp);
            pq.push(nxt);
            node_expanded++;
            cur_node_expanded++;
            max_node_expanded = max(max_node_expanded,
cur_node_expanded);
            vis[cor(nxtp)] = cur.Y;
        }
    }
    TIMER_C(_t)
    TIMER_P(_t)
    Solver::construct_path(path, t);

    return mp(path, node_expanded);
}

#endif
```

idastar_v2_solver.hpp

```cpp
#ifndef IDA2_H
#define IDA2_H

#include "custom_header.hpp"

class IDA2 : public Solver
{
private:
    std::function<int(const pii&, const pii&)> h;
public:
    vector<pii> path;
    vector<vector<int>> dep; //relaxtion based
    //string name; this will cause the name be empty string
    IDA2();
    int dfs2(int d, pii s, pii t, int maxd);
    void init_dfs();
    pair<vector<pii>, int> solve(pii x, pii y, double TL);
};

IDA2::IDA2()
{
    h = [&](const pii &s, const pii &t){
        return manhattan_distance(s, t)/3;
    };
    name = "IDA2";
}

int IDA2::dfs2(int d, pii s, pii t, int maxd) //s = current, p =
from, d = current depth, t =
{
    int ret = INT_MAX;
    if(d > maxd) return d;
    if(s == t) return -1;
    for(auto &di : dxdy)
    {
        pii nxt = s + di;
        if(!inrange(nxt)) continue;
```

```
        int nxtd = d + 1 - h(s,t) + h(nxt,t);
        //nxt no relexation, no explore
        if(dep[cor(nxt)] <= nxtd) continue;
        path.pb(nxt);
        node_expanded++;
        cur_node_expanded++;
        max_node_expanded = max(max_node_expanded,
cur_node_expanded);
        dep[cor(nxt)] = nxtd;
        ret = min(ret, dfs2(nxtd, nxt, t, maxd));
        if(ret == -1) break; //success, no pop_back this path
        path.pop_back();
        cur_node_expanded--;
    }
    return ret;
}

void IDA2::init_dfs()
{
    path.clear();
    int INF = N*N + N;
    dep = vector<vector<int>>(N, vector<int>(N, INF));
}


pair<vector<pii>, int> IDA2::solve(pii s, pii t, double TL)
{
    node_expanded = 0;
    max_node_expanded = 0;
    cur_node_expanded = 0;
    int maxd = h(s,t);
    while(1)
    {
        init_dfs();
        dep[cor(s)] = h(s,t);
        node_expanded++;
        cur_node_expanded++;
        max_node_expanded = max(max_node_expanded,
cur_node_expanded);
```

```
        path.pb(s); //cleared before, no need pop
        int res = dfs2(h(s,t), s, t, maxd);
        if(res == -1) break;
        else if(res == INT_MAX) {cout << "no solution" << endl;
break;}
        maxd = res;
    }
    assert(path.size() == maxd+1); // s -d++- 1 - 2 - 3 - d++ t
    return mp(path, node_expanded);
}

#endif
```

tc_generate.sh

```
#echo "hi" > test.txt
#x0=$((1+$RANDOM % 100))
#echo $((1+$RANDOM % 100)) >> test.txt
#echo $x0 >> test.txt

#g++ -o out -std=c++17 game.cpp

algoN=4
maxN=$1
deltaN=$2
testN=$3

rm -rf testcases
mkdir testcases

for board_size in $(seq 5 ${deltaN} ${maxN});
do
    echo "" > ./testcases/N${board_size}.txt

    for test_id in $(seq 1 1 ${testN});
    do
            x1=$((0+$RANDOM % $board_size))
            x2=$((0+$RANDOM % $board_size))
```

```
            y1=$((0+$RANDOM % $board_size))
            y2=$((0+$RANDOM % $board_size))
            echo $board_size $x1 $x2 $y1 $y2 >> ./testcases/
N${board_size}.txt
            #run ./out < board_size algo_id $
            #echo $board_size 1 $x1 $x2 $y1 $y2 | ./out >>
testout.txt
    done
done
```

exp.cpp

```
#include "custom_header.hpp"
//#include "test_case_generator.hpp"

#include "bfs_solver.hpp"
#include "dfs_solver.hpp"
#include "iddfs_solver.hpp"
#include "astar_solver.hpp"
#include "idastar_solver.hpp"
#include "dfsv2_solver.hpp"
#include "iddfs_v2_solver.hpp"
#include "idastar_v2_solver.hpp"
#include <fstream>

//0-indexed
int algoN = 4;
int minN = 5;
int maxN = 50;
int deltaN = 5;
int testN = 100;

string result_directory = "./results/";
string testcase_directory = "./testcases/";

void test_solver(Solver* solver)
{
    //RTCG gen; should not regen, should test on same dataset
```

```
    for(int ni = minN; ni <= maxN; ni += deltaN)
    {
        ifstream is;
        ofstream fs;
        //cerr << testcase_directory + "N" + to_string(ni) +
".txt" << endl;
        is.open(testcase_directory + "N" + to_string(ni) + ".txt");
        fs.open(result_directory + solver->name + "_N" +
to_string(ni) + ".csv");
        assert(is.is_open());
        assert(fs.is_open());
        vector<int> time_statistics(testN);
        vector<int> mem_statistics(testN);
        double sum_time = 0.0;
        double sum_mem = 0.0;
        rep(i, 0, testN)
        {
            //is >> N chages board size!!
            pii s, t;
            is >> N >> s.X >> s.Y >> t.X >> t.Y;
            solver->init(); //N is set and use to init N size board
            auto ret = solver->solve(s, t);
            time_statistics[i] = solver->node_expanded;
            mem_statistics[i] = solver->max_node_expanded;
            sum_time += solver->node_expanded;
            sum_mem += solver->max_node_expanded;
            fs << solver->node_expanded << " "<< solver-
>max_node_expanded << endl;
        }
        sort(time_statistics.begin(), time_statistics.end());
        sort(mem_statistics.begin(), mem_statistics.end());
        fs << "node_expanded_min: " << time_statistics[0] << endl;
        fs << "max_node_expanded_min: " << mem_statistics[0] <<
endl;
        fs << "node_expanded_max: " << time_statistics[testN-1] <<
endl;
        fs << "max_node_expanded_max: " << mem_statistics[testN-1]
<< endl;
```

```
        fs << "node_expanded_medium: " << time_statistics[testN/2]
<< endl;
        fs << "max_node_expanded_medium: " << mem_statistics[testN/
2] << endl;
        fs << "node_expanded_average: " << sum_time/double(testN)
<< endl;
        fs << "max_node_expanded_average: " << sum_mem/
double(testN) << endl;

        fs.close();
    }
    return;
}

signed main()
{
    //list of solver names
    cin >> maxN >> deltaN >> testN; //passing hyper param from
stdin
    vector<Solver*> solvers;
    solvers.pb(new BFS());
    solvers.pb(new DFS2());
    solvers.pb(new IDDFS2());
    solvers.pb(new Astar());
    solvers.pb(new IDA2());
    //rep(i, 0, solvers.size()) cout << "id " << i << ":,
algorithm :" << solvers[i]->name << endl;
    //cout << "input solver id:" << endl;
    int n_solver = solvers.size();
    rep(i, 0, n_solver)
    {
        test_solver(solvers[i]);
    }

    return 0;
}
```

exp.sh

```
# usage
#notice algoN should change

algoN=4
maxN=$1
deltaN=$2
testN=$3

# algoN independent
./tc_generate.sh $maxN $deltaN $testN

rm -rf results
mkdir results

# exp
g++ -o tmpout -std=c++17 exp.cpp
echo $maxN $deltaN $testN | ./tmpout
rm tmpout
```

Appendix B: References

- https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/
- https://en.wikipedia.org/wiki/Knight%27s_tour
- Lecture Note Set 2
- New e3 discussion forum