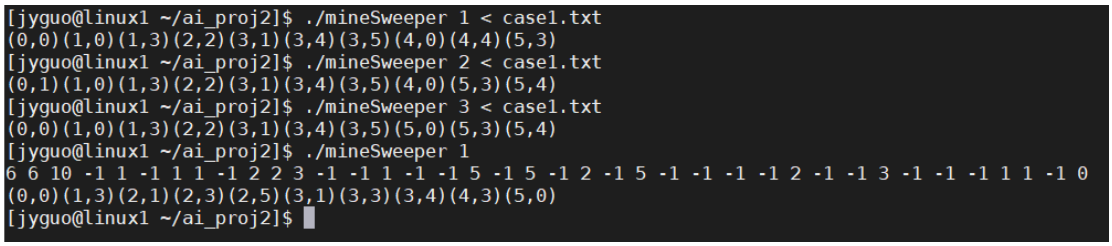
Programming Assignment #2

0612129 郭家佑

Demo:



This is a simple demo, and the parameter in the argv is stand for different kind of heuristics using.

1 -> MRV

2 -> Degree heuristic

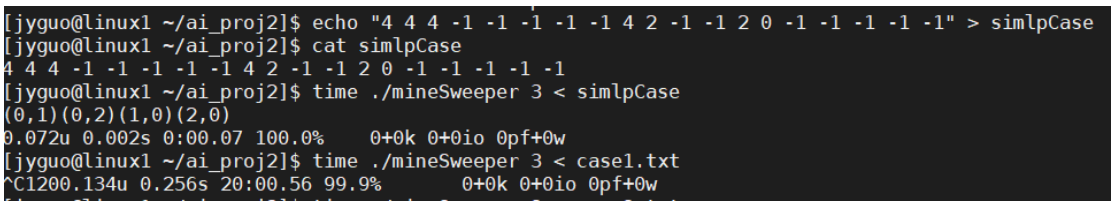
3 -> LCV

After inputting the constrain, the screen would print the position of the mines. However, if the solution is not existed, it would try almost all of possible and print nothing.

Detail inside:

First, I would take a look of LCV, unlike MRV and degree heuristic, it is a heuristic to choose what value should assign to the variable first. So that it need to compute all of the case (in the default case, is 16*2) initially and hard to choose a good value to avoid waste. In example 1 of the assignment given, if the (0, 0) and (0, 1) both are regarded as mines due to the domain of other variable is not changed, but the system would try a lot of situation and it would know that (0, 0) and (0, 1) couldn't having same value at last.

In the picture, we can find out it can solve the small case. But it need to spend more than 20 min on the example 1.



However, the LCV is not need to be used barely. It can combine with MRV and LCV,

that is, MRV mark those variable only have one value in its domain, and using LCV to choose what value should choose from remain variable witch have two value in its domain.
And this picture it case using LCV with MRV (with parameter 3), compare to other kind of heuristics.

```
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 1 < case1.txt
(0,0)(1,0)(1,3)(2,2)(3,1)(3,4)(3,5)(4,0)(4,4)(5,3)
0.002u 0.001s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 2 < case1.txt
(0,1)(1,0)(1,3)(2,2)(3,1)(3,4)(3,5)(4,0)(5,3)(5,4)
5.343u 0.003s 0:05.35 99.8%     0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 3 < case1.txt
(0,0)(1,0)(1,3)(2,2)(3,1)(3,4)(3,5)(5,0)(5,3)(5,4)
5.720u 0.002s 0:05.73 99.8%     0+0k 0+0io 0pf+0w
```

Sadly, LCV+MRV(in the following of report, I would use LCV to represent MRV+ LCV) is also slower than barely using MRV and I think it is due to the LCV need large computation to choose a point is mine or not, but it isn't such useful.
Anyway, I using "#ifndef" and "#ifdef" in "mineSweeper.cpp" to parted two kind of LCV and you can comment the line "#define LCV_WITH_MRV 1" in "minesweeper.hpp".

Next, in the program, I using the forward checking in usual. In "mineSweeper.cpp", there are two function, "testMidCondition" and "testEndCondition", and the "testMidCondition" used for checking the mines position is true or not for all of constrain, before the state that mines in the map are not all flagged.
Otherwise, there are some checking in the program, like checking how many of mine in the table. And, in the MRV case, the function similar to testMidCondition is combined when choosing variable to assign. So that the MRV without forwarding checking is a bit hard to do. So that I checking the LCV without forwarding checking.
The first diagram is the normal case named as" testCase".

```
[jyguo@linux1 ~/ai_proj2]$ time ./testCase  3 < case1.txt
(0,0)(1,0)(1,3)(2,2)(3,1)(3,4)(3,5)(5,0)(5,3)(5,4)
5.700u 0.003s 0:05.71 99.8%     0+0k 928+0io 1pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 3 < case2.txt
(0,0)(0,1)(0,2)(1,5)(2,1)(2,3)(2,5)(3,0)(3,4)(5,2)
0.065u 0.001s 0:00.06 100.0%    0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 3 < case3.txt
(0,2)(0,3)(1,5)(2,0)(2,5)(3,0)(4,5)(5,0)(5,2)(5,3)
104.102u 0.013s 1:44.12 99.9%   0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 3 < case4.txt
(0,0)(1,3)(2,1)(2,3)(2,5)(3,1)(3,3)(3,4)(4,3)(5,0)
0.002u 0.001s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
```

And the second one is commented the "testMidCondition" function.

```
[jyguo@linux1 ~/ai_proj2]$ time ./mineSweeper 3 < case1.txt
(0,0)(1,0)(1,3)(2,2)(3,1)(3,4)(3,5)(4,4)(5,0)(5,3)
7.163u 0.000s 0:07.16 100.0%    0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ vim mineSweeper.cpp
[jyguo@linux1 ~/ai_proj2]$ time ./mineSweeper 3 < case2.txt
(0,0)(0,1)(0,2)(1,5)(2,1)(2,3)(2,5)(3,0)(3,4)(5,2)
0.095u 0.001s 0:00.10 90.0%     0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./mineSweeper 3 < case3.txt
(0,2)(0,3)(1,5)(2,0)(2,5)(3,0)(4,5)(5,0)(5,2)(5,3)
115.640u 0.010s 1:55.68 99.9%   0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./mineSweeper 3 < case4.txt
(0,0)(1,3)(2,1)(2,3)(2,5)(3,1)(3,3)(3,4)(4,3)(5,0)
0.002u 0.002s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
```

Obviously, the case with the forward checking is more quick in most case.

```
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 2 < case1.txt
(0,1)(1,0)(1,3)(2,2)(3,1)(3,4)(3,5)(4,0)(5,3)(5,4)
5.184u 0.002s 0:05.19 99.8%     0+0k 928+0io 1pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 2 < case2.txt
(0,0)(0,1)(0,2)(1,5)(2,1)(2,3)(2,5)(3,0)(3,4)(5,2)
0.838u 0.000s 0:00.84 98.8%     0+0k 0+0io 0pf+0w
```

```
[jyguo@linux1 ~/ai_proj2]$ time ./mineSweeper 2 < case1.txt
(0,1)(1,0)(1,3)(2,2)(3,1)(3,4)(3,5)(4,0)(5,3)(5,4)
668.645u 0.036s 11:08.77 99.9%  0+0k 928+0io 1pf+0w
```

Similar test of degree heuristic. The different being more obviously.

Test in different size, first, I test 6*6 matrix which made from “ramdon.c”. In fact, the position set as a mine with 25%, and it could be more or less.

```
[jyguo@linux1 ~/ai_proj2]$ cat case1_s
case1_size6.txt  case1_size8.txt
[jyguo@linux1 ~/ai_proj2]$ cat case1_size6.txt
6 6 11
1 -1 1 1 -1 -1
-1 5 -1 2 -1 -1
-1 -1 -1 -1 1 -1
3 5 -1 -1 1 -1
1 -1 4 -1 3 2
-1 2 -1 2 -1 -1
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 1 < case1_size6.txt
(1,0)(1,2)(2,0)(2,1)(2,2)(3,2)(3,3)(4,1)(5,2)(5,4)(5,5)
0.002u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 2 < case1_size6.txt
(1,0)(1,2)(2,0)(2,1)(2,2)(3,2)(3,3)(4,1)(5,2)(5,4)(5,5)
0.073u 0.001s 0:00.07 100.0%    0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 3 < case1_size6.txt
(1,0)(1,2)(2,0)(2,1)(2,2)(3,2)(3,3)(4,1)(5,2)(5,4)(5,5)
37.960u 0.004s 0:37.98 99.9%    0+0k 0+0io 0pf+0w
```

In severe case, the output is similar to example case given.

Then testing about 7*7 size, sadly the LCV spending a long time to run. And the other case both can finish in few sec. I tested 10 random case, and the result showing that MRV finished 7 case in few sec, degree heuristic finish 5 case in few sec and LCV only finish 1 case in few second. About 8*8 size, in 10 cases, the MRV only finish 2 case and the degree heuristic only finish 4 case.

About 9*9 size, in 10 cases, all of function can’t finish any case in 30 sec.

The following picture is part of test.

```
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 1 < case1_size7.txt
(0,2)(0,3)(1,0)(1,3)(1,4)(2,0)(2,2)(3,3)(3,6)(5,0)(5,1)(5,2)(5,4)(6,0)(6,4)(6,5)
1.081u 0.128s 0:01.21 99.1%     0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 2 < case1_size7.txt
^C12.268u 0.001s 0:12.27 99.9%  0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 3 < case1_size7.txt
^C15.084u 0.001s 0:15.09 99.9%  0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 1 < case2_size7.txt
^C9.612u 0.565s 0:10.19 99.8%   0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 1 < case3_size7.txt
(0,0)(0,5)(1,6)(2,1)(2,6)(3,0)(4,1)(5,0)(5,3)(5,5)(6,2)(6,6)
0.001u 0.002s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 2 < case2_size7.txt
(0,3)(2,1)(2,4)(2,5)(4,4)(5,0)(5,1)(5,2)(5,6)(6,0)
0.001u 0.001s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 3 < case2_size7.txt
^C13.625u 0.002s 0:13.63 99.9%  0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 1 < case3_size7.txt
(0,0)(0,5)(1,6)(2,1)(2,6)(3,0)(4,1)(5,0)(5,3)(5,5)(6,2)(6,6)
0.001u 0.002s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 2 < case3_size7.txt
(0,0)(0,5)(1,6)(2,1)(2,6)(3,0)(4,1)(5,0)(5,3)(5,5)(6,2)(6,6)
0.413u 0.000s 0:00.41 100.0%    0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 3 < case3_size7.txt
^C8.946u 0.006s 0:08.96 99.7%   0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 1 < case4_size7.txt
(0,0)(0,1)(0,2)(1,0)(1,4)(1,5)(2,2)(4,4)(4,5)(5,1)(5,5)(6,1)(6,3)
0.239u 0.017s 0:00.25 96.0%     0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 2 < case4_size7.txt
^C10.447u 0.000s 0:10.45 99.9%  0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 3 < case4_size7.txt
^C6.768u 0.002s 0:06.78 99.7%   0+0k 0+0io 0pf+0w
```

Algorithms and heuristic

First, if I don't use any heuristic to backtracking, the complexity is O(2^n) which n is variable number, in the assignment assigned as -1.

Although I am not sure the precise complexity with heuristic, different heuristic also show that they suit in different case.

MRV suits to solve the constrains in case which are dispersed, specially about the case whose variable's domains are singletons.
And the degree heuristic suit to solve the concentrated constrain like the example 3 which constrain concentrated in center.

There are two 8*8 case solved in the following picture. And they show their convenience on the situation.

```
8 8 13
-1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 1 -1
3 -1 3 1 -1 1 -1 2
-1 -1 -1 1 -1 -1 -1 -1
1 2 2 1 -1 2 -1 -1
-1 -1 1 -1 3 -1 2 -1
-1 -1 -1 -1 -1 -1 2 -1
-1 -1 1 -1 2 -1 -1 -1
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 1 < case2_size8.txt
^C15.570u 0.592s 0:16.16 100.0% 0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 2 < case2_size8.txt
(0,0)(0,1)(0,3)(1,0)(1,7)(2,1)(3,1)(3,2)(3,6)(5,5)(6,3)(6,5)(7,0)
0.002u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
```

```
[jyguo@linux1 ~/ai_proj2]$ cat case3_size8.txt
8 8 20
-1 2 -1 2 -1 -1 2 -1
-1 2 -1 -1 -1 -1 -1 -1
-1 -1 1 -1 3 -1 -1 -1
1 -1 2 3 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 3 -1 -1 1 -1
-1 -1 3 4 -1 2 1 1
1 -1 -1 2 -1 1 -1 -1
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 1 < case3_size8.txt
(0,0)(0,2)(0,4)(0,5)(0,7)(2,3)(2,5)(2,6)(2,7)(3,4)(3,6)(3,7)(4,0)(4,2)(5,2)(5,4)(5,7)(6,1)(7,2)(7,4)
0.089u 0.014s 0:00.10 90.0%     0+0k 0+0io 0pf+0w
[jyguo@linux1 ~/ai_proj2]$ time ./testCase 2 < case3_size8.txt
^C41.636u 0.001s 0:41.64 99.9%  0+0k 0+0io 0pf+0w
```

remaining questions and future investigation

Few day ago, I test some case with MRV and get the bad_alloc() in few second. But until today, I don't have the problem anymore and I am not sure why. And I also wonder the server I use to test those case is not such stable and the "time" command is not such reliable and produce enough information. But I think the virtue machine could be less stable and I don't know what kind of data should be collected and what is useful tool in window system.
At lease, I know there are perf and gprof command in linux.

About my program, I using stack and also using recursion to finish this assignment in the SAME time. So that it meaning that I spend some space and time on not necessary part. But I don't think I can change code in a short time and the function is complete. And it is part of future investigation about how to make the code better maybe shorter or faster.

Appendix

Now I know that VS code can print the text with color.

Makefile:

```
all: main.cpp mineSweeper.cpp
    g++ -O2 -g -o mineSweeper main.cpp mineSweeper.cpp
```

minesweeper.cpp

```
#include "mineSweeper.hpp"
extern int board_size_x, board_size_y;
extern int variable_num;
extern int mines_num;
int runTime;

void func(vector<vector<int>>& map, int mode){
    //using stack to store different state with vector STL
    stack<vector<position_state>> state;
    stack<vector<position_state>>& state_ref = state;

    //first state to put into stack
    vector<position_state> first;
    vector<position_state>& first_ref = first;

    //making a lookup table to let me know how the 2D map's position st
ate
    //store in the linear vector, if there are no correspond part, stor
e
    //with -1.
    vector<vector<int>> lookup = map;
    vector<vector<int>> &lookup_ref = lookup;
    int index = 0;
    for(int i=0; i < board_size_x; i++){
        for(int j=0; j< board_size_y; j++){
            if(lookup[i][j] == -1) lookup[i][j] = index++;
            else lookup[i][j] = -1;
        }
    }
```

```
    //complete the first state
    for(int i=0; i < board_size_x; i++){
        for(int j=0; j< board_size_y; j++){
            if(map[i][j] == -1) {
                position_state tmp(i, j);
                first.push_back(tmp);
            }
        }
    }
    state.push(first);

    //using different heuristic function with parameter
    switch(mode){
        case 1:
            MRV(map, state_ref, lookup_ref);
            break;
        case 2:
            DH(map, state_ref, lookup_ref);
            break;
        case 3:
            LCV(map, state_ref, lookup_ref);
            break;
        default:
            break;
    }

    //if the state return successfully, print all of position of mine.
    //In the loop, if there are no answer, it would print nothing.
    for(int i = 0; i<variable_num; i++){
        if(state.top()[i].state == isMine){
            cout << "(" << state.top()[i].x << "," << state.top()[i].y
<< ")";
        }
    }
    cout << endl;
}

int MRV(vector<vector<int>>& map,
```

```
        stack<vector<position_state>>& state,
        vector<vector<int>>& lookup){

    //copy the newest state
    vector<position_state> priority;
    priority.assign(state.top().begin(), state.top().end());

    //To check what position only have one domain, that is,
    //it must be mine or nothing in the position.
    //The reflect flag to make sure the state would different
    //between the last state.
    int reflesh_flag = 0;
    for(int i=0; i<board_size_x; i++){
        for(int j=0; j<board_size_y; j++){
            if(map[i][j] == -1) continue;
            int notSure_num = 0;
            int isMine_num = 0;
            for(int test=0; test<8; test++){
                int test_x = i + surround_x[test];
                int test_y = j + surround_y[test];
                if(test_x >= 0 && test_x < board_size_x &&
                        test_y >= 0 && test_y < board_size_y &&
                        map[test_x][test_y] == -1
                  ){
                    if(priority[lookup[test_x][test_y]].state == notSur
e) notSure_num++;
                    else if(priority[lookup[test_x][test_y]].state == i
sMine) isMine_num++;
                }
            }
            if(map[i][j] - isMine_num == 0){
                for(int test=0; test<8; test++){
                    int test_x = i + surround_x[test];
                    int test_y = j + surround_y[test];
                    if(test_x >= 0 && test_x < board_size_x &&
                            test_y >= 0 && test_y < board_size_y &&
                            map[test_x][test_y] == -1
                      ){
```

```
                        if(priority[lookup[test_x][test_y]].state == no
tSure){
                            priority[lookup[test_x][test_y]].state = is
ntMine;
                            reflesh_flag = 1;

                        }
                    }
                }
            }else if(map[i][j] - isMine_num == notSure_num){
                for(int test=0; test<8; test++){
                    int test_x = i + surround_x[test];
                    int test_y = j + surround_y[test];
                    if(test_x >= 0 && test_x < board_size_x &&
                            test_y >= 0 && test_y < board_size_y &&
                            map[test_x][test_y] == -1
                      ){
                        if(priority[lookup[test_x][test_y]].state == no
tSure) {
                            priority[lookup[test_x][test_y]].state = is
Mine;
                            reflesh_flag = 1;
                        }
                    }
                }
            }else if(map[i][j] - isMine_num < 0){
                state.pop();
                return 0;
            }
        }
    }

    //Forward checking
    int isMine_num = 0;
    for(int i=0; i<variable_num; i++){
        if(priority[i].state == isMine) isMine_num++;
    }
    if(isMine_num == mines_num){
```

```
        if(!testEndCondition(map, priority, lookup)){
            state.pop();
            return 0;
        }
        state.push(priority);
        return 1;
    } else if(isMine_num > mines_num){
        state.pop();
        return 0;
    }

    //if the new state is same as the last one,
    //choose ramdon position as mine due to the
    //remind positiin's domain is all 1 and 0,
    //I using the "notSure" state to represent.
    if(reflesh_flag){
        state.push(priority);
        if(MRV(map, state, lookup)) return 1;
    }else{
        for(int i=0; i<variable_num; i++){
            if(priority[i].state == notSure){
                priority[i].state = isMine;
                state.push(priority);
                if(MRV(map, state, lookup)){
                    return 1;
                }else {
                    priority[i].state = notSure;
                }
            }
        }
    }
    return 0;
}

int DH(vector<vector<int>>& map,
        stack<vector<position_state>>& state,
        vector<vector<int>>& lookup){
    //Full name is degree heuristic.
```

```
    //in the function, choosing the position has most constrains.

    //degree isnt represent the new state but different degreee,
    //that is, how much constrain effect the point.
    vector<position_state> degree;
    degree.assign(state.top().begin(), state.top().end());
    vector<position_state>& test_state = degree;

    //forward checking first
    int isMine_num = 0;
    for(int i=0; i<variable_num; i++){
        if(degree[i].state == isMine) isMine_num++;
    }
    if(isMine_num > mines_num){
        return 0;
    }else if(isMine_num == mines_num){
        if(testEndCondition(map, test_state, lookup)) return 1;
        else return 0;
    }else {
        if(testMidCondition(map, test_state, lookup) == 0) return 0;
    }

    //compute the constrain in different position
    //And if there are 0 constrain, regrad as a point
    //almost impossible to be a mine.
    for(int i=0; i<board_size_x; i++){
        for(int j=0; j<board_size_y; j++){
            if(map[i][j] != -1){
                for(int test = 0; test < 8; test++){
                    int test_x = i + surround_x[test];
                    int test_y = j + surround_y[test];
                    if(test_x >= 0 && test_x < board_size_x &&
                            test_y >= 0 && test_y < board_size_y &&
                            map[test_x][test_y] == -1){
                        degree[lookup[test_x][test_y]].value += map[i][
j];
                        if(map[i][j] == 0) degree[lookup[test_x][test_y
]].value = INT_MIN;
```

```
                    }
                }
            }
        }
    }
    //using heap to choose position effected by most constrain.
    make_heap(degree.begin(), degree.end(), stateHeapMax());

    //Try the position with the heap one by one.
    vector<position_state> new_state = state.top();
    new_state.assign(state.top().begin(), state.top().end());
    while(!degree.empty()){
        position_state tmp = degree.front();
        if(tmp.state != isMine){
            mine_state original_state = new_state[lookup[tmp.x][tmp.y]]
.state;
            new_state[lookup[tmp.x][tmp.y]].state = isMine;
            state.push(new_state);
            if(DH(map, state, lookup)){
                return 1;
            }else{
                new_state[lookup[tmp.x][tmp.y]].state = original_state;
                state.pop();
            }
        }
        pop_heap(degree.begin(),degree.end(), stateHeapMax()); degree.p
op_back();
    }
    return 0;
}


int LCV(vector<vector<int>>& map,
        stack<vector<position_state>>& state,
        vector<vector<int>>& lookup){

    vector<position_state> corrent_state;
    vector<position_state>& test_state = corrent_state;
```

```
    corrent_state.assign(state.top().begin(), state.top().end());

    //forward checking
    int isMine_num = 0;
    for(int i=0; i<variable_num; i++){
        if(corrent_state[i].state == isMine) isMine_num++;
    }

    if(isMine_num > mines_num){
        return 0;
    }else if(isMine_num == mines_num){
        if(testEndCondition(map, test_state, lookup)) return 1;
        else return 0;
    }else {
        if(testMidCondition(map, test_state, lookup) == 0) return 0;
    }

    //Compute the upper bound and low bound on constrain
    vector<vector<int>> upperbound(board_size_x, vector<int>(board_size
_y));
    vector<vector<int>> lowerbound(board_size_x, vector<int>(board_size
_y));
    for(int i=0; i<board_size_x; i++){
        for(int j=0; j<board_size_y; j++){
            if(map[i][j] == -1) continue;
            int notSure_num = 0;
            int isMine_num = 0;
            for(int test=0; test<8; test++){
                int test_x = i + surround_x[test];
                int test_y = j + surround_y[test];
                if(test_x >= 0 && test_x < board_size_x &&
                        test_y >= 0 && test_y < board_size_y &&
                        map[test_x][test_y] == -1
                  ){
                    if(corrent_state[lookup[test_x][test_y]].state == n
otSure) notSure_num++;
                    else if(corrent_state[lookup[test_x][test_y]].state
 == isMine) isMine_num++;
```

```
                }
            }
            upperbound[i][j] = isMine_num + notSure_num;
            lowerbound[i][j] = isMine_num;
        }
    }

    vector<position_state> priority;

//compute how much do each values effected other
//Parted in different case to test.
    for(int i=0; i<variable_num; i++){
        if (corrent_state[i].state != notSure) continue;
        int handle_x = corrent_state[i].x;
        int handle_y = corrent_state[i].y;

        int effect_surround_num_upper = 0;
        int effect_surround_num_lower = 0;
        mine_state corrent_point_state = notSure;

        //a value in domain change at most effect 5*5 size of point.
        vector<vector<int>> effect_to_isMine (5, vector<int>(5, 0));
        vector<vector<int>> effect_to_isntMine (5, vector<int>(5, 0));

//differe kind, comment the line if LCV_WITH_MRV in .hpp to use.
//The most important of them is checking
//This is:
//map[test_x][test_y] - upperbound[test_x][test_y] == -1
//map[test_x][test_y] - lowerbound[test_x][test_y] == 1
//Two case

#ifndef LCV_WITH_MRV
        for(int test = 0; test < 8; test++){
            int test_x = handle_x + surround_x[test];
            int test_y = handle_y + surround_y[test];

            if(test_x < 0 || test_x >= board_size_x ||
                    test_y < 0 || test_y >= board_size_y ||
```

```
                    map[test_x][test_y] == -1) continue;

            if(map[test_x][test_y] > upperbound[test_x][test_y]){
                corrent_point_state = errorCondition;
            }else if(map[test_x][test_y] == upperbound[test_x][test_y])
{
                if(corrent_point_state == notSure) corrent_point_state
= isMine;
                else if(corrent_point_state == isntMine) corrent_point_
state = errorCondition;
            }else if(map[test_x][test_y] - upperbound[test_x][test_y] =
= -1){
                for(int test_block = 0; test_block<8; test_block++){
                    int test_block_x = test_x + surround_x[test_block];
                    int test_block_y = test_y + surround_y[test_block];

                    if(test_block_x >= 0 && test_block_x < board_size_x
 &&
                            test_block_x >= 0 && test_block_y < board_s
ize_y &&
                            map[test_block_x][test_block_y] == -1 &&
                            corrent_state[lookup[test_block_x][test_blo
ck_y]].state == notSure
                          ){
                            effect_to_isMine[2 + surround_x[test] + surroun
d_x[test_block]][2 + surround_y[test] + surround_y[test_block]] = 1;
                    }
                }
            }

            if(map[test_x][test_y] < lowerbound[test_x][test_y]){
                corrent_point_state = errorCondition;
            }else if(map[test_x][test_y] == lowerbound[test_x][test_y])
{
                if(corrent_point_state == notSure) corrent_point_state
= isntMine;
                else if(corrent_point_state == isMine) corrent_point_st
ate = errorCondition;
```

```
            }else if(map[test_x][test_y] - lowerbound[test_x][test_y] =
= 1){
                for(int test_block = 0; test_block<8; test_block++){
                    int test_block_x = test_x + surround_x[test_block];
                    int test_block_y = test_y + surround_y[test_block];

                    if(test_block_x >= 0 && test_block_x < board_size_x
 &&
                            test_block_x >= 0 && test_block_y < board_s
ize_y &&
                            map[test_block_x][test_block_y] == -1 &&
                            corrent_state[lookup[test_block_x][test_blo
ck_y]].state == notSure
                          ){
                            effect_to_isntMine[2 + surround_x[test] + surro
und_x[test_block]][2 + surround_y[test] + surround_y[test_block]] = 1;
                    }
                }
            }

            if(corrent_point_state == errorCondition) break;
        }

#endif
#ifdef LCV_WITH_MRV
        for(int test = 0; test < 8; test++){
            int test_x = handle_x + surround_x[test];
            int test_y = handle_y + surround_y[test];

            if(test_x < 0 || test_x >= board_size_x ||
                test_y < 0 || test_y >= board_size_y ||
                map[test_x][test_y] == -1) continue;

            if(map[test_x][test_y] >= upperbound[test_x][test_y]){
                if(corrent_point_state == notSure) corrent_point_state
= isntMine;
                else if(corrent_point_state == isMine) corrent_point_st
ate = errorCondition;
```

```
                if(map[test_x][test_y] == upperbound[test_x][test_y]){
                    for(int test_block = 0; test_block<8; test_block++)
{
                        int test_block_x = test_x + surround_x[test_blo
ck];
                        int test_block_y = test_y + surround_y[test_blo
ck];

                        if(test_block_x >= 0 && test_block_x < board_si
ze_x &&
                                test_block_x >= 0 && test_block_y < boa
rd_size_y &&
                                map[test_block_x][test_block_y] == -
1 &&
                                corrent_state[lookup[test_block_x][test
_block_y]].state == notSure
                            ){
                              corrent_state[lookup[test_block_x][test_blo
ck_y]].state = isMine;
                        }
                    }
                }
            }else if(map[test_x][test_y] - upperbound[test_x][test_y] =
= -1){
                for(int test_block = 0; test_block<8; test_block++){
                    int test_block_x = test_x + surround_x[test_block];
                    int test_block_y = test_y + surround_y[test_block];

                    if(test_block_x >= 0 && test_block_x < board_size_x
 &&
                            test_block_x >= 0 && test_block_y < board_s
ize_y &&
                            map[test_block_x][test_block_y] == -1 &&
                            corrent_state[lookup[test_block_x][test_blo
ck_y]].state == notSure
                        ){
```

```
                        effect_to_isMine[2 + surround_x[test] + surroun
d_x[test_block]][2 + surround_y[test] + surround_y[test_block]];
                    }
                }
            }

            if(map[test_x][test_y] <= lowerbound[test_x][test_y]){
                if(corrent_point_state == notSure) corrent_point_state
= isMine;
                else if(corrent_point_state == isntMine) corrent_point_
state = errorCondition;

                if(map[test_x][test_y] == lowerbound[test_x][test_y]){
                    for(int test_block = 0; test_block<8; test_block++)
{
                        int test_block_x = test_x + surround_x[test_blo
ck];
                        int test_block_y = test_y + surround_y[test_blo
ck];

                        if(test_block_x >= 0 && test_block_x < board_si
ze_x &&
                                test_block_x >= 0 && test_block_y < boa
rd_size_y &&
                                map[test_block_x][test_block_y] == -
1 &&
                                corrent_state[lookup[test_block_x][test
_block_y]].state == notSure
                           ){
                             corrent_state[lookup[test_block_x][test_blo
ck_y]].state = isntMine;
                        }
                    }
                }
            }else if(map[test_x][test_y] - lowerbound[test_x][test_y] =
= 1){
                for(int test_block = 0; test_block<8; test_block++){
                    int test_block_x = test_x + surround_x[test_block];
```

```
                    int test_block_y = test_y + surround_y[test_block];

                    if(test_block_x >= 0 && test_block_x < board_size_x
 &&
                            test_block_x >= 0 && test_block_y < board_s
ize_y &&
                            map[test_block_x][test_block_y] == -1 &&
                            corrent_state[lookup[test_block_x][test_blo
ck_y]].state == notSure
                        ){
                          effect_to_isntMine[2 + surround_x[test] + surro
und_x[test_block]][2 + surround_y[test] + surround_y[test_block]];
                    }
                }
            }
            if(corrent_point_state == errorCondition) break;
        }
#endif
        for(int m=0; m<5; m++) for(int n=0; n<5; n++) if(effect_to_isnt
Mine[m][n]) effect_surround_num_lower++;
        for(int m=0; m<5; m++) for(int n=0; n<5; n++) if(effect_to_isMi
ne[m][n]) effect_surround_num_upper++;

        //compute what value need to try first by a heap
        if(corrent_point_state == errorCondition) continue;
        else{
            if(corrent_point_state == notSure || corrent_point_state ==
 isMine){
                position_state tmp(handle_x, handle_y);
                tmp.value = effect_surround_num_lower;
                tmp.state = isMine;
                priority.push_back(tmp);
            }

            if(corrent_point_state == notSure || corrent_point_state ==
 isntMine){
                position_state tmp(handle_x, handle_y);
                tmp.value = effect_surround_num_upper;
```

```
                tmp.state = isntMine;
                priority.push_back(tmp);
            }
        }
    }
    //make the heap to make more ramdon
    random_shuffle ( priority.begin(), priority.end());
    //making heap
    make_heap(priority.begin(), priority.end(), stateHeapMin());
    //try
    while(!priority.empty()){
        runTime--;
        vector<position_state> new_state = corrent_state;
        if (new_state[lookup[priority.front().x][priority.front().y]].s
tate == notSure)
            new_state[lookup[priority.front().x][priority.front().y]] =
 priority.front();
        pop_heap(priority.begin(),priority.end(), stateHeapMin()); prio
rity.pop_back();
        state.push(new_state);

        if(LCV(map, state, lookup)) return 1;
        else state.pop();

    }
    return 0;
}


//Two funtion to check
//This one to check reaching end case or not
int testEndCondition(vector<vector<int>>& map,
        vector<position_state>& test_array,
        vector<vector<int>>& lookup
        ){
    for(int i=0; i<board_size_x; i++){
        for(int j=0; j<board_size_y; j++){
            if(map[i][j] == -1) continue;
            int isMine_num = 0;
```

```
            for(int test=0; test<8; test++){
                int test_x = i + surround_x[test];
                int test_y = j + surround_y[test];
                if(test_x >= 0 && test_x < board_size_x &&
                        test_y >= 0 && test_y < board_size_y &&
                        map[test_x][test_y] == -1
                  ){
                    if(test_array[lookup[test_x][test_y]].state == isMi
ne) isMine_num++;
                }
            }
            if(map[i][j] !=  isMine_num) return 0;
        }
    }
    return 1;
}

//This one use for forward checking
int testMidCondition(vector<vector<int>>& map,
        vector<position_state>& test_state,
        vector<vector<int>>& lookup
        ){
    for(int i=0; i<board_size_x; i++){
        for(int j=0; j<board_size_y; j++){
            if(map[i][j] == -1) continue;
            int isMine_num = 0;
            for(int test=0; test<8; test++){
                int test_x = i + surround_x[test];
                int test_y = j + surround_y[test];
                if(test_x >= 0 && test_x < board_size_x &&
                        test_y >= 0 && test_y < board_size_y &&
                        map[test_x][test_y] == -1
                  ){
                    if(test_state[lookup[test_x][test_y]].state == isMi
ne) isMine_num++;
                }
            }
            if(map[i][j] - isMine_num < 0){
```

```
                return 0;
            }
        }
    }
    return 1;
}
```

mineSweeper.hpp

```
#include <iostream>
#include <vector>
#include <stack>
#include <array>
#include <algorithm>
#include <list>
#include <bits/stdc++.h>
#define LCV_WITH_MRV 1
using namespace std;

const int surround_x[8] = {1, 1, 0, -1, -1, -1, 0, 1};
const int surround_y[8] = {0, 1, 1, 1, 0, -1, -1, -1};

enum mine_state{
    notSure, isMine, isntMine, errorCondition
};

class position_state{
    public:
        int x, y;
        int value;
        mine_state state;
        position_state(int init_x, int init_y){
            state = notSure;
            x = init_x;
            y = init_y;
            value = 0;
        }
};
```

```
struct stateHeapMax{
    bool operator()(const position_state& x, const position_state& y){
        return x.value < y.value;
    }
};

struct stateHeapMin{
    bool operator()(const position_state& x, const position_state& y){
        return x.value >= y.value;
    }
};

void func(vector<vector<int>>&, int);
int MRV(vector<vector<int>>&,
        stack<vector<position_state>>&,
        vector<vector<int>>&
        );
int DH(vector<vector<int>>&,
        stack<vector<position_state>>&,
        vector<vector<int>>&
        );
int LCV(vector<vector<int>>&,
        stack<vector<position_state>>&,
        vector<vector<int>>&
        );
int testEndCondition(vector<vector<int>>&,
        vector<position_state>&,
        vector<vector<int>>&
        );
int testMidCondition(vector<vector<int>>&,
        vector<position_state>&,
        vector<vector<int>>&
        );
```

random.c (use for make matrix)

```
#include <stdio.h>
#include <string.h>
```

```c
#include <stdlib.h>
#include <time.h>
const int surround_x[8] = {1, 1, 0, -1, -1, -1, 0, 1};
const int surround_y[8] = {0, 1, 1, 1, 0, -1, -1, -1};
int main(int argc, char** argv){
    srand( time(NULL) );
    int size_x = atoi(argv[1]);
    int size_y = (argc == 3) ? atoi(argv[2]) : size_x;

    int map[size_x][size_y];
    int mine_num = 0;

    for(int i=0; i<size_x; i++){
        for(int j=0; j<size_y; j++){
            int x = rand() % 4;
            if(!x){
                map[i][j] = -1;
                mine_num++;
            }
            else map[i][j] = 0;
        }
    }

    for(int i=0; i<size_x; i++){
        for(int j=0; j<size_y; j++){
            int x = rand() % 2;
            if(!map[i][j] && x){
                for(int test = 0 ; test<8; test++){
                    if(i+surround_x[test] >= 0 && i+surround_x[test] <
size_x
                    && j+surround_y[test] >= 0 && j+surround_y[test] <
size_y
                    && map[i+surround_x[test]][j+surround_y[test]] == -
1
                    ) map[i][j]++;
                }
            }
        }
```

```
    }
    for(int i=0; i<size_x; i++){
        for(int j=0; j<size_y; j++){
            if(!map[i][j]) map[i][j] = -1;
        }
    }

    printf("%d %d %d\n", size_x, size_y, mine_num);

    for(int i=0; i<size_x; i++){
        for(int j=0; j<size_y; j++){
            printf("%d ", map[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```