hw2 report :

1.程式架構：

n 代表當前的點

（1）backtracking with no heuristic :

push (n,0) (n,1) 每次都 pop 一組出來，如果遇到錯誤，就 pop 直到沒有錯誤，最後檢查是否符合炸彈個數需求，符合就輸出，不符合就繼續 pop 直到答案符合需求

（2）backtracking with mrv heuristic :

和 backtracking with no heuristic 最大的差別在於每次填炸彈都會影響 domain，只要有domain=1，就先從這個點開始填，要 pop 的時候必須順便修正domain

（3）backtracking with mrv and degree heuristic :

做 mrv 檢查後，有可能會出現所有 domain 都等於 2 的情況， 這時候就以 degree 檢查各點的 constraint 大小，先從 constraint 小的開始填，pop 的時候要順便恢復 domain 和 constraint(走過了沒)

（4）backtracking with mrv, degree, and lcv heuristic :

如果 mrv 和 degree 都做完了，也已經找到點了，就以 lcv 來決定要先 push (n,0) 還是 (n,1)，將0, 1分別帶入，取會造成其他點 constraint 最小（最多可能）的點，優先push，pop 的時候要順便恢復 domain 和 constraint

＊ forward checking 包含在heuristic 中

2.程式內容：

（1）main ：主程式

（2）minesweeper：原始的backtracking

（3）minesweeper_mrv：加了 mrv 的 backtracking

（4）minesweeper_degree：加了 mrv & degree 的 backtracking

（5）change_hint：放炸彈後, 要更改附近的 hint

（6）check_hint_greater_than_zero：如果有 hint <= 0, 代表不能再放炸彈了

(7) check_hint_zero：如果有 hint != 0, 代表還沒結束

(8) output_board： 把一維 vector 以二維輸出

(9) change_domain：類似 forward checking, 尋找所有 hint 的 upperbound & lowerbound, 接著把所有 domain = 1 的填上應填的數字

(10) find_domain_1：用於mrv, 優先找domain = 1, 如果都沒有就找 domain = 2

(11) find_degree：用於degree, 如果mrv都找不到 domain=1 的, 就找degree中 constraint 最大的

(12) find_constraint：用於lcv, 判斷輸入 0,1 後 constraint 大小

(13) mine_equal：檢查 mine 總量是否等於輸入的數量

(14) MRV_true：如果true, 使用mrv

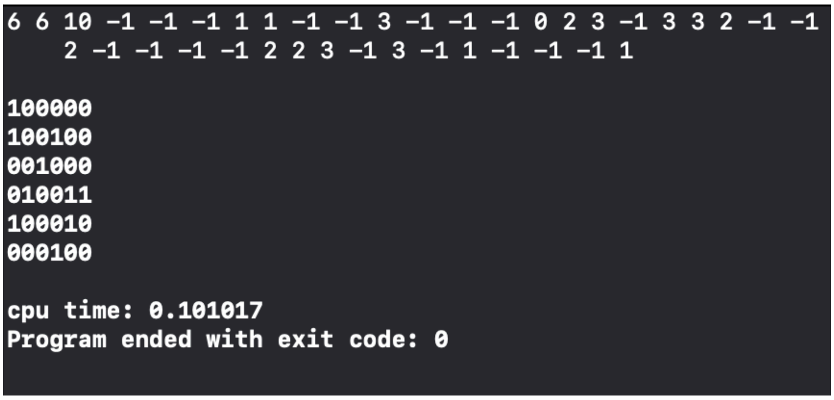
(15) MRV_degree_true：如果true, 使用mrv & degree

(16) MRV_degree_LCV_true：如果true, 使用mrv & degree & lcv

3.比較結果：

input 1:

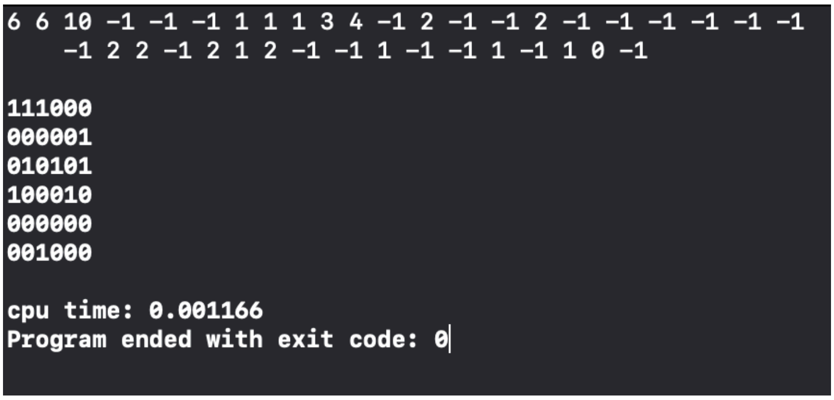
🟦	🟧	🟦	1	1	🟦
🟧	3	🟦	🟧	🟦	0
2	3	🟧	3	3	2
🟦	🟧	2	🟦	🟧	🟧
🟦	2	2	3	🟦	3
🟧	1	🟦	🟧	🟧	1



```
6 6 10 -1 -1 -1 1 1 -1 -1 3 -1 -1 -1 0 2 3 -1 3 3 2 -1 -1
    2 -1 -1 -1 -1 2 2 3 -1 3 -1 1 -1 -1 -1 1

100000
100100
001000
010011
100010
000100

cpu time: 0.101017
Program ended with exit code: 0
```

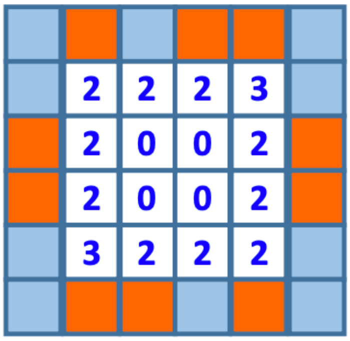
input 2:

🟧	🟧	🟦	1	1	1
3	4	🟧	2	🟦	🟧
2	🟧	🟦	🟦	🟧	🟧
🟦	🟧	2	2	🟧	2
1	2	🟧	🟦	1	🟦
🟦	1	🟦	1	0	🟦

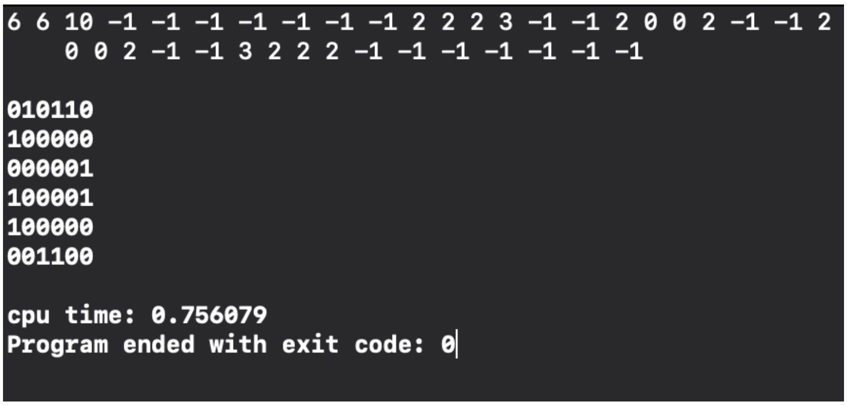


```
6 6 10 -1 -1 -1 1 1 1 3 4 -1 2 -1 -1 2 -1 -1 -1 -1 -1 -1
    -1 2 2 -1 2 1 2 -1 -1 1 -1 -1 1 -1 1 0 -1

111000
000001
010101
100010
000000
001000

cpu time: 0.001166
Program ended with exit code: 0
```

input 3:

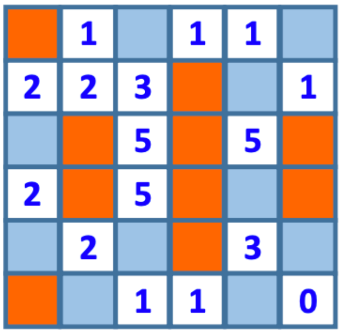


🟦	🟧	🟦	🟧	🟧	🟦
🟦	2	2	2	3	🟦
🟧	2	0	0	2	🟧
🟧	2	0	0	2	🟧
🟦	3	2	2	2	🟦
🟦	🟧	🟧	🟦	🟧	🟦

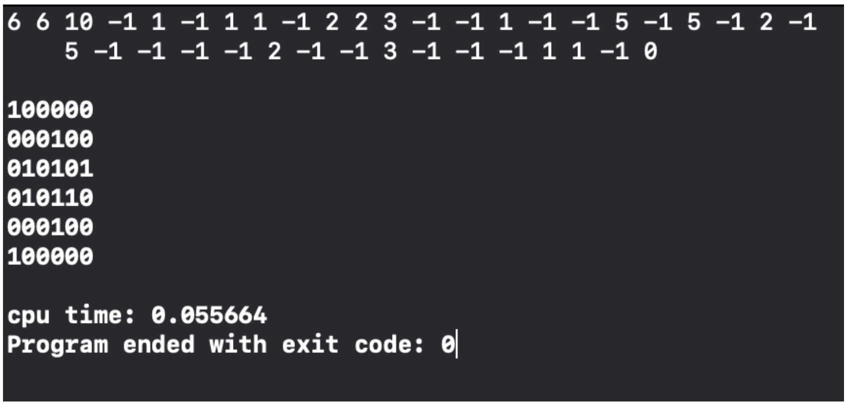


```
6 6 10 -1 -1 -1 -1 -1 -1 -1 2 2 2 3 -1 -1 2 0 0 2 -1 -1 2
    0 0 2 -1 -1 3 2 2 2 -1 -1 -1 -1 -1 -1 -1

010110
100000
000001
100001
100000
001100

cpu time: 0.756079
Program ended with exit code: 0
```

input 4:



🟧	1	🟦	1	1	🟦
2	2	3	🟧	🟦	1
🟦	🟧	5	🟧	5	🟧
2	🟧	5	🟧	🟦	🟧
🟦	2	🟦	🟧	3	🟦
🟧	🟦	1	1	🟦	0

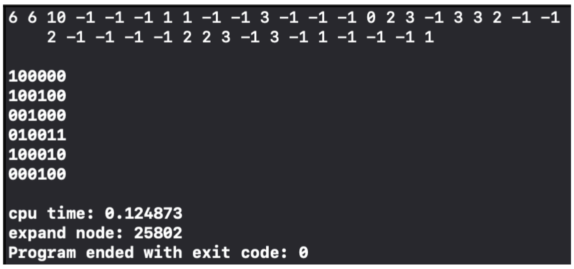


```
6 6 10 -1 1 -1 1 1 -1 2 2 3 -1 -1 1 -1 -1 5 -1 5 -1 2 -1
    5 -1 -1 -1 -1 2 -1 -1 3 -1 -1 -1 1 1 -1 0

100000
000100
010101
010110
000100
100000

cpu time: 0.055664
Program ended with exit code: 0
```

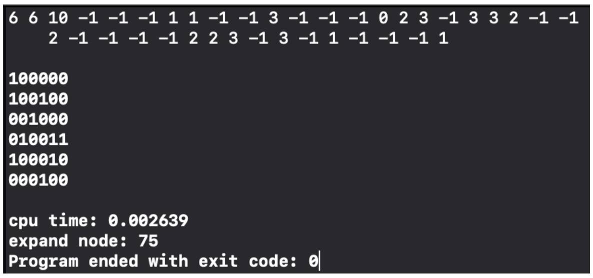
結論：結果雖然和範例些許不同，但是都是正確答案

4.比較expand node, time：

input1



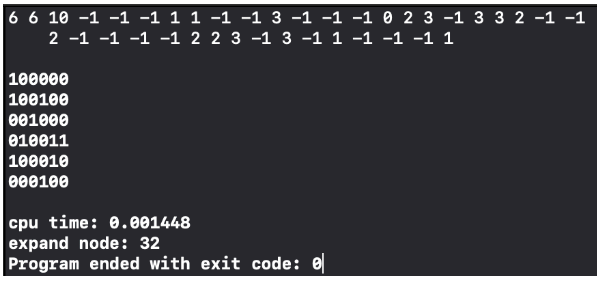
```
6 6 10 -1 -1 -1 1 1 -1 -1 3 -1 -1 -1 0 2 3 -1 3 3 2 -1 -1
    2 -1 -1 -1 -1 2 2 3 -1 3 -1 1 -1 -1 -1 1

100000
100100
001000
010011
100010
000100

cpu time: 0.124873
expand node: 25802
Program ended with exit code: 0
```



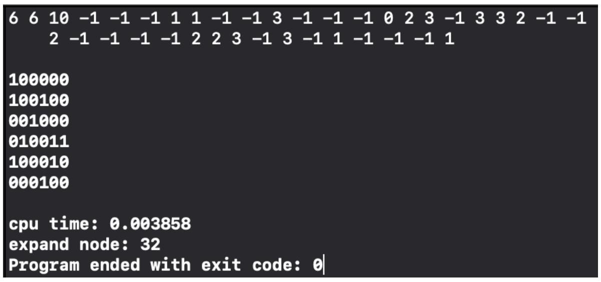
```
6 6 10 -1 -1 -1 1 1 -1 -1 3 -1 -1 -1 0 2 3 -1 3 3 2 -1 -1
    2 -1 -1 -1 -1 2 2 3 -1 3 -1 1 -1 -1 -1 1

100000
100100
001000
010011
100010
000100

cpu time: 0.002639
expand node: 75
Program ended with exit code: 0
```

backtracking: 25802(0.124873)

mrv :75(0.002639)



```
6 6 10 -1 -1 -1 1 1 -1 -1 3 -1 -1 -1 0 2 3 -1 3 3 2 -1 -1
    2 -1 -1 -1 -1 2 2 3 -1 3 -1 1 -1 -1 -1 1

100000
100100
001000
010011
100010
000100

cpu time: 0.001448
expand node: 32
Program ended with exit code: 0
```



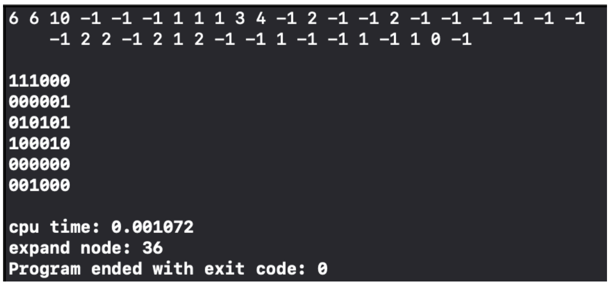
```
6 6 10 -1 -1 -1 1 1 -1 -1 3 -1 -1 -1 0 2 3 -1 3 3 2 -1 -1
    2 -1 -1 -1 -1 2 2 3 -1 3 -1 1 -1 -1 -1 1

100000
100100
001000
010011
100010
000100

cpu time: 0.003858
expand node: 32
Program ended with exit code: 0
```

mrv+degree: 32(0.001448)

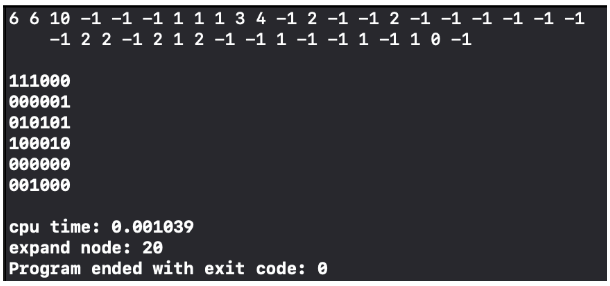
mrv+degree+lcv:32(0.003858)

---

input 2



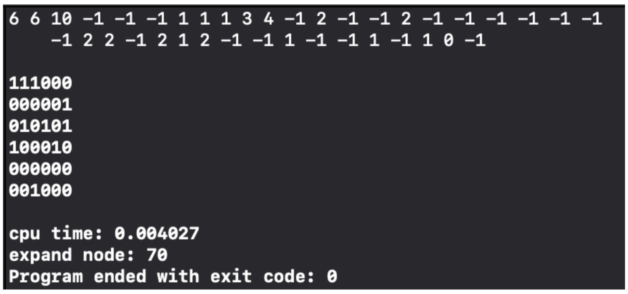
```
6 6 10 -1 -1 -1 1 1 1 3 4 -1 2 -1 -1 2 -1 -1 -1 -1 -1 -1
    -1 2 2 -1 2 1 2 -1 -1 1 -1 -1 1 -1 1 0 -1

111000
000001
010101
100010
000000
001000

cpu time: 0.001072
expand node: 36
Program ended with exit code: 0
```



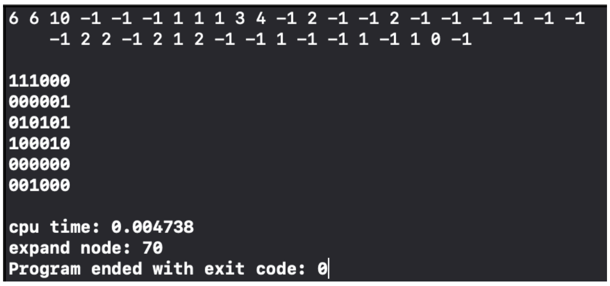
```
6 6 10 -1 -1 -1 1 1 1 3 4 -1 2 -1 -1 2 -1 -1 -1 -1 -1 -1
    -1 2 2 -1 2 1 2 -1 -1 1 -1 -1 1 -1 1 0 -1

111000
000001
010101
100010
000000
001000

cpu time: 0.001039
expand node: 20
Program ended with exit code: 0
```

backtracking: 36(0.001072)

mrv: 20(0.001039)



```
6 6 10 -1 -1 -1 1 1 1 3 4 -1 2 -1 -1 2 -1 -1 -1 -1 -1 -1
    -1 2 2 -1 2 1 2 -1 -1 1 -1 -1 1 -1 1 0 -1

111000
000001
010101
100010
000000
001000

cpu time: 0.004027
expand node: 70
Program ended with exit code: 0
```



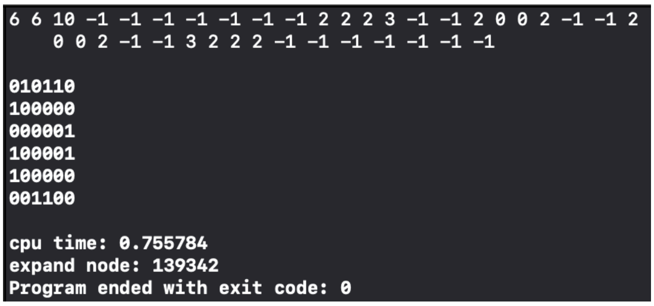
```
6 6 10 -1 -1 -1 1 1 1 3 4 -1 2 -1 -1 2 -1 -1 -1 -1 -1 -1
    -1 2 2 -1 2 1 2 -1 -1 1 -1 -1 1 -1 1 0 -1

111000
000001
010101
100010
000000
001000

cpu time: 0.004738
expand node: 70
Program ended with exit code: 0
```

mrv+degree: 70(0.004027)

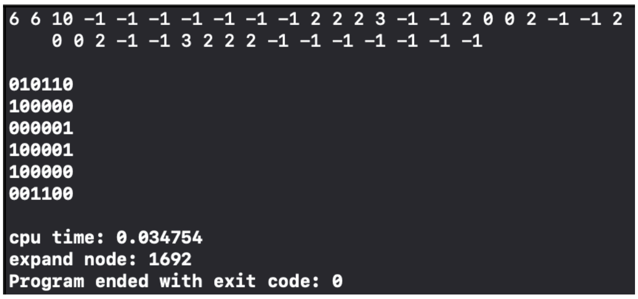
mrv+degree+lcv: 70(0.004738)

---

input 3



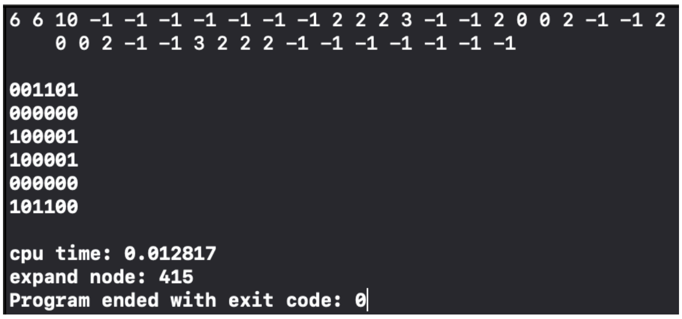
```
6 6 10 -1 -1 -1 -1 -1 -1 -1 2 2 2 3 -1 -1 2 0 0 2 -1 -1 2
    0 0 2 -1 -1 3 2 2 2 -1 -1 -1 -1 -1 -1 -1

010110
100000
000001
100001
100000
001100

cpu time: 0.755784
expand node: 139342
Program ended with exit code: 0
```



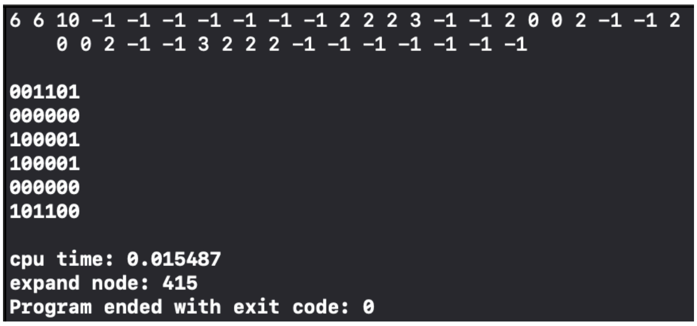
```
6 6 10 -1 -1 -1 -1 -1 -1 -1 2 2 2 3 -1 -1 2 0 0 2 -1 -1 2
    0 0 2 -1 -1 3 2 2 2 -1 -1 -1 -1 -1 -1 -1

010110
100000
000001
100001
100000
001100

cpu time: 0.034754
expand node: 1692
Program ended with exit code: 0
```

backtracking: 139342(0.755784)

mrv: 1692(0.034754)



```
6 6 10 -1 -1 -1 -1 -1 -1 -1 2 2 2 3 -1 -1 2 0 0 2 -1 -1 2
    0 0 2 -1 -1 3 2 2 2 -1 -1 -1 -1 -1 -1 -1

001101
000000
100001
100001
000000
101100

cpu time: 0.012817
expand node: 415
Program ended with exit code: 0
```

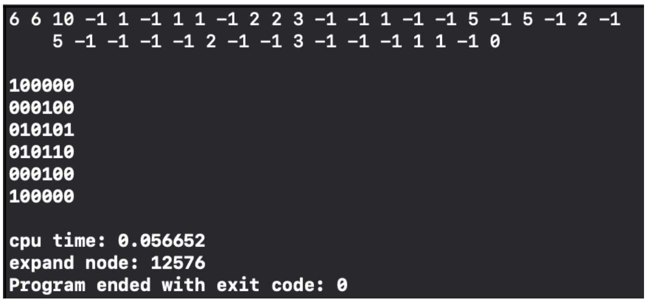


```
6 6 10 -1 -1 -1 -1 -1 -1 -1 2 2 2 3 -1 -1 2 0 0 2 -1 -1 2
    0 0 2 -1 -1 3 2 2 2 -1 -1 -1 -1 -1 -1 -1

001101
000000
100001
100001
000000
101100

cpu time: 0.015487
expand node: 415
Program ended with exit code: 0
```

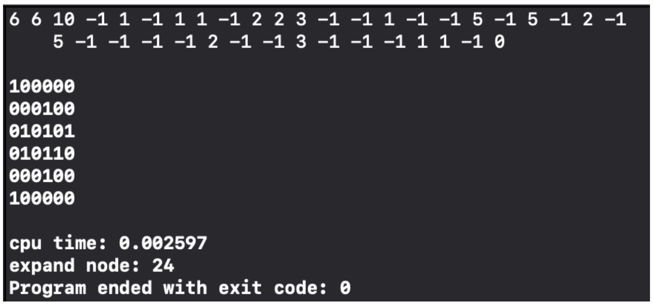
mrv+degree: 415(0.012817)　　　mrv+degree+lcv: 415(0.015487)

— —

input 4

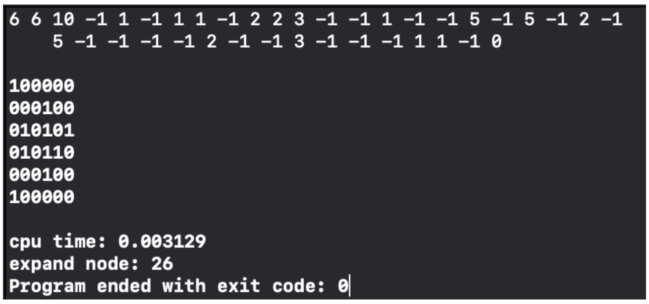


```
6 6 10 -1 1 -1 1 1 -1 2 2 3 -1 -1 1 -1 -1 5 -1 5 -1 2 -1
    5 -1 -1 -1 -1 2 -1 -1 3 -1 -1 -1 1 1 -1 0

100000
000100
010101
010110
000100
100000

cpu time: 0.056652
expand node: 12576
Program ended with exit code: 0
```

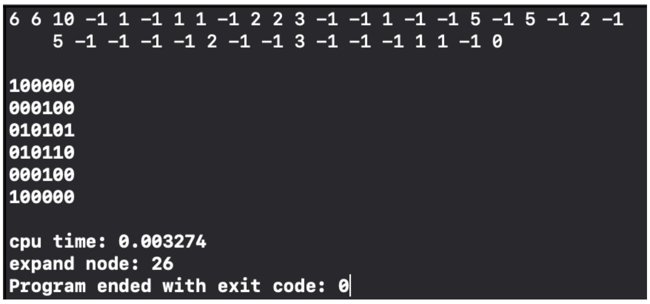


```
6 6 10 -1 1 -1 1 1 -1 2 2 3 -1 -1 1 -1 -1 5 -1 5 -1 2 -1
    5 -1 -1 -1 -1 2 -1 -1 3 -1 -1 -1 1 1 -1 0

100000
000100
010101
010110
000100
100000

cpu time: 0.002597
expand node: 24
Program ended with exit code: 0
```

backtracking: 12576(0.056652)　　　mrv: 24(0.002597)



```
6 6 10 -1 1 -1 1 1 -1 2 2 3 -1 -1 1 -1 -1 5 -1 5 -1 2 -1
    5 -1 -1 -1 -1 2 -1 -1 3 -1 -1 -1 1 1 -1 0

100000
000100
010101
010110
000100
100000

cpu time: 0.003129
expand node: 26
Program ended with exit code: 0
```



```
6 6 10 -1 1 -1 1 1 -1 2 2 3 -1 -1 1 -1 -1 5 -1 5 -1 2 -1
    5 -1 -1 -1 -1 2 -1 -1 3 -1 -1 -1 1 1 -1 0

100000
000100
010101
010110
000100
100000

cpu time: 0.003274
expand node: 26
Program ended with exit code: 0
```

mrv+degree: 26(0.003129)　　　mrv+degree+lcv: 26(0.003274)

— —

結論：

所有測資在使用 forward checking & mrv 後速度都有提升，expanding node 也大幅減少，大部分測資增加degree後expanding node 會變少，唯有第二組測資反而變多了，推測第二組測資可能從頭開始填炸彈是最有效率的，才會導致使用degree後浪費了更多次測試，至於lcv目前看起來似乎沒有太大的影響？個人認為forward checking, mrv, degree 基本上都是可以加速的方法，lcv 則視情況而定。

appendix:

```
//
//  main.cpp
//  hw2
//
//  Created by 蕭楚澔 on 2020/4/27.
//  Copyright © 2020 Bob. All rights reserved.
//

#include <iostream>
#include <vector>
#include <math.h>
#include <stack>
#include <time.h>

using namespace std;

stack<pair<int, bool>> s;
//決定要使用的 h funtion
bool MRV_true = false;
bool MRV_degree_true = false;
bool MRV_degree_LCV_true = false;

//檢查 mine 總量是否等於輸入的數量
bool mine_equal(vector<int> mine, int m){
    int ans = 0;
    for(int i=0; i<mine.size(); i++){
        ans+=mine[i];
    }
    if(ans == m){
        return true;
    }
    else{
        return false;
    }
}

//用於lcv, 判斷輸入 0,1 後 constraint 大小
int find_constraint(vector<int> mine, vector<bool> hint, vector<int> domain, int place, bool b){
    int ans = 0;
    domain[place] = -1;
    if(b == 0){
        mine[place] = -2;
    }
    else{
        mine[place] = -3;
    }
    int t=sqrt(mine.size());
    vector<int> v;
    int arr[t][t];
    for(int i=0; i<t; i++){
        for(int j=0; j<t; j++){
            arr[i][j] = i*t+j;
        }
    }
    for(int i=0; i<mine.size(); i++){
        v.clear();
        if(hint[i] == true){
            int x=(i/t);
            int y=(i%t);
            if(y-1 >= 0){
                v.push_back(arr[x][y-1]);
            }
            if(y+1 < t){
                v.push_back(arr[x][y+1]);
            }
            if(x-1 >= 0){
                v.push_back(arr[x-1][y]);
            }
            if(x-1 >= 0 && y-1 >= 0){
                v.push_back(arr[x-1][y-1]);
            }
            if(x-1 >= 0 && y+1 < t){
                v.push_back(arr[x-1][y+1]);
            }
            if(x+1 < t){
                v.push_back(arr[x+1][y]);
            }
            if(x+1 < t && y-1 >= 0){
                v.push_back(arr[x+1][y-1]);
            }
            if(x+1 < t && y+1 < t){
                v.push_back(arr[x+1][y+1]);
            }
            int upperbound = 0;
            int lowerbound = 0;
            for(int j=0; j<v.size(); j++){
                if(domain[v[j]] != 3 && domain[v[j]] != -1){
                    upperbound+=1;
                }
                else if(mine[v[j]] == -3){
                    mine[i]--;
                }
            }

            if(upperbound == mine[i]){
                for(int k=0; k<v.size(); k++){
                    if(domain[v[k]] == 2){
                        domain[v[k]] = 1;
                    }
                }
            }
            else if(lowerbound == mine[i]){
                for(int k=0; k<v.size(); k++){
                    if(domain[v[k]] == 2){
                        domain[v[k]] = 0;
                    }
                }
            }
        }
    }
    for(int i=0; i<domain.size(); i++){
        if(hint[i] == false && domain[i] == 0){
            ans++;
        }
        else if(hint[i] == false && domain[i] == 1){
            ans++;
        }
        else if(hint[i] == false && domain[i] == 2){
            ans+=2;
        }
    }
    return ans;
}

//用於degree, 如果mrv都找不到 domain=1 的, 就找degree中 constraint 最大
的
int find_degree(vector<int> domain, vector<bool> hint, vector<int>
degree, vector<bool> &sign){
    for(int i=0; i<domain.size(); i++){
        if(domain[i] == 0){
            return i;
        }
    }
    for(int i=0; i<domain.size(); i++){
        if(domain[i] == 1){
            return i;
        }
    }
    int max = -1;
    int ans = (int)degree.size();
    for(int i=0; i<degree.size(); i++){
        if(domain[i] == 2 && degree[i] > max && sign[i] == false){
            max = degree[i];
            ans = i;
        }
    }
    if(ans<degree.size()){
        sign[ans] = true;
    }
    return ans;
}
```

```
//用於mrv, 優先找domain = 1, 如果都沒有就找 domain = 2
int find_domain_1(vector<int> domain){
    for(int i=0; i<domain.size(); i++){
        if(domain[i] == 0){
            return i;
        }
    }
    for(int i=0; i<domain.size(); i++){
        if(domain[i] == 1){
            return i;
        }
    }
    for(int i=0; i<domain.size(); i++){
        if(domain[i] == 2){
            return i;
        }
    }
    return (int)domain.size();
}
```

```
//類似 forward checking, 尋找所有 hint 的 upperbound & lowerbound, 接著把所有 domain = 1 的填上應填的數字
void change_domain(vector<int> mine, vector<bool> hint, vector<int> &domain){
    int t=sqrt(mine.size());
    vector<int> v;
    int arr[t][t];
    for(int i=0; i<t; i++){
        for(int j=0; j<t; j++){
            arr[i][j] = i*t+j;
        }
    }
    for(int i=0; i<mine.size(); i++){
        v.clear();
        if(hint[i] == true){
            int x=(i/t);
            int y=(i%t);
            if(y-1 >= 0){
                v.push_back(arr[x][y-1]);
            }
            if(y+1 < t){
                v.push_back(arr[x][y+1]);
            }
            if(x-1 >= 0){
                v.push_back(arr[x-1][y]);
            }
            if(x-1 >= 0 && y-1 >= 0){
                v.push_back(arr[x-1][y-1]);
            }
            if(x-1 >= 0 && y+1 < t){
                v.push_back(arr[x-1][y+1]);
            }
            if(x+1 < t){
                v.push_back(arr[x+1][y]);
            }
            if(x+1 < t && y-1 >= 0){
                v.push_back(arr[x+1][y-1]);
            }
            if(x+1 < t && y+1 < t){
                v.push_back(arr[x+1][y+1]);
            }
            int upperbound = 0;
            int lowerbound = 0;
            for(int j=0; j<v.size(); j++){
                if(domain[v[j]] != 3 && domain[v[j]] != -1){
                    upperbound+=1;
                }
            }

            if(upperbound == mine[i]){
                for(int k=0; k<v.size(); k++){
                    if(domain[v[k]] == 2){
                        domain[v[k]] = 1;
                    }
                }
            }
            else if(lowerbound == mine[i]){
                for(int k=0; k<v.size(); k++){
                    if(domain[v[k]] == 2){
                        domain[v[k]] = 0;
                    }
                }

            }
        }
    }
}

//把一維 vector 以二維輸出
void output_board(vector<int> v){
    int t=(int)v.size();
    t = sqrt(t);
    for(int i=0; i<t; i++){
        for(int j=0; j<t; j++){
            cout<<v[t*i+j];
        }
        cout<<endl;
    }
}

//如果有 hint != 0, 代表還沒結束
```

```
bool check_hint_zero(vector<int> mine, vector<bool> hint){
    bool ans=true;
    for(int i=0; i<hint.size(); i++){
        if(hint[i] == true && mine[i] != 0){
            ans=false;
            break;
        }
    }
    return ans;
}

//如果有 hint <= 0，代表不能再放炸彈了
bool check_hint_greater_than_zero(vector<int> mine, vector<bool> 
hint){
    bool ans=true;
    for(int i=0; i<hint.size(); i++){
        if(hint[i] == true && mine[i] < 0){
            ans=false;
            break;
        }
    }
    return ans;
}

//放炸彈後，要更改附近的 hint
void change_hint(vector<int> &mine, vector<bool> hint, int p, bool 
b){
    int t=sqrt(mine.size());
    vector<int> v;
    int arr[t][t];
    for(int i=0; i<t; i++){
        for(int j=0; j<t; j++){
            arr[i][j] = i*t+j;
        }
    }
    int x=(p/t);
    int y=(p%t);
    if(y-1 >= 0){
        v.push_back(arr[x][y-1]);
    }
    if(y+1 < t){
        v.push_back(arr[x][y+1]);
    }
    if(x-1 >= 0){
        v.push_back(arr[x-1][y]);
    }
    if(x-1 >= 0 && y-1 >= 0){
        v.push_back(arr[x-1][y-1]);
    }
    if(x-1 >= 0 && y+1 < t){
        v.push_back(arr[x-1][y+1]);
    }
    if(x+1 < t){
        v.push_back(arr[x+1][y]);
    }
    if(x+1 < t && y-1 >= 0){
        v.push_back(arr[x+1][y-1]);
    }
    if(x+1 < t && y+1 < t){
        v.push_back(arr[x+1][y+1]);
    }

    if(b == 1){
        for(int i=0; i<v.size(); i++){
            if(hint[v[i]] == true){
                mine[v[i]]--;
            }
        }
    }
    else{
        for(int i=0; i<v.size(); i++){
            if(hint[v[i]] == true){
                mine[v[i]]++;
            }
        }
    }
}

//原始的backtracking
int minesweeper(vector<int> &mine,  vector<bool> hint, int place, bool bomb){
    if(hint[place] == true){
        place++;
        return place;
    }
    else{
        mine[place]=bomb;
        change_hint(mine, hint, place, bomb);
        if(check_hint_greater_than_zero(mine, hint) == true){
            place++;
            return place;
        }
        else{
            return place;
        }
    }
}

//加了 mrv 的 backtracking
int minesweeper_mrv(vector<int> &mine, vector<bool> hint, int place, bool bomb, vector<int> &domain){
    if(hint[place] == true){
        place++;
        return place;
    }
    else if(domain[place] == 1 || domain[place] == 0){
        mine[place] = bomb;
        if(bomb == 1){
            change_hint(mine, hint, place, bomb);
        }
        change_domain(mine, hint, domain);
        domain[place] = -1;
        int ans = find_domain_1(domain);
        return ans;
    }
    else{
        mine[place]=bomb;
        change_hint(mine, hint, place, bomb);
        change_domain(mine, hint, domain);
        domain[place] = -1;
        if(check_hint_greater_than_zero(mine, hint) == true){
            int ans = find_domain_1(domain);
            return ans;
        }
        else{
            return place;
        }
    }
}

//加了 mrv & degree 的 backtracking
int minesweeper_degree(vector<int> &mine, vector<bool> hint, int
place, bool bomb, vector<int> &domain, vector<int> degree, 
vector<bool> &sign){
    if(hint[place] == true){
        place++;
        return place;
    }
    else if(domain[place] == 1 || domain[place] == 0){
        mine[place] = bomb;
        if(bomb == 1){
            change_hint(mine, hint, place, bomb);
        }
        change_domain(mine, hint, domain);
        domain[place] = -1;
        sign[place] = true;
        int ans = find_degree(domain, hint, degree, sign);
        return ans;
    }
    else{
        mine[place]=bomb;
        change_hint(mine, hint, place, bomb);
        change_domain(mine, hint, domain);
        domain[place] = -1;
        sign[place] = true;
        if(check_hint_greater_than_zero(mine, hint) == true){
            int ans = find_degree(domain, hint, degree, sign);
            return ans;
        }
        else{
            return place;
        }
    }
}

int main(int argc, const char * argv[]) {
    //計算時間
    clock_t start, end;
    double cpu_time_used;
    start = clock();

    int boardsizex;
    int boardsizey;
    int mines;
    cin>>boardsizex>>boardsizey>>mines;
    vector<int> mine;
    for(int i=0; i<boardsizex; i++){
        for(int j=0; j<boardsizey; j++){
            int t;
            cin>>t;
            mine.push_back(t);
        }
    }
    vector<bool> hint;
    for(int i=0; i<mine.size(); i++){
        if(mine[i]!=-1){
            hint.push_back(true);
        }
        else{
            hint.push_back(false);
        }
    }

    vector<int> h_mrv;
    for(int i=0; i<hint.size(); i++){
        if(hint[i] == true){
            h_mrv.push_back(3);
        }
        else{
            h_mrv.push_back(2);
        }
    }

    vector<int> h_degree;
    for(int i=0; i<hint.size(); i++){
        h_degree.push_back(-1);
    }
    int t = sqrt(hint.size());
    int arr[t][t];
    for(int i=0; i<t; i++){
        for(int j=0; j<t; j++){
            arr[i][j] = i*t+j;
        }
    }
    for(int i=0; i<hint.size(); i++){
        if(hint[i] == true){
            int x=(i/t);
            int y=(i%t);
            if(y-1 >= 0 && hint[arr[x][y-1]] == false){
                h_degree[arr[x][y-1]] += 1;
            }
            if(y+1 < t && hint[arr[x][y+1]] == false){
                h_degree[arr[x][y+1]] += 1;
            }
            if(x-1 >= 0 && hint[arr[x-1][y]] == false){
                h_degree[arr[x-1][y]] += 1;
            }
            if(x-1 >= 0 && y-1 >= 0 && hint[arr[x-1][y-1]] == 
false){
                h_degree[arr[x-1][y-1]] += 1;
            }
            if(x-1 >= 0 && y+1 < t && hint[arr[x-1][y+1]] == 
false){
                h_degree[arr[x-1][y+1]] += 1;
            }
            if(x+1 < t && hint[arr[x+1][y]] == false){
                h_degree[arr[x+1][y]] += 1;
            }
            if(x+1 < t && y-1 >= 0 && hint[arr[x+1][y-1]] == 
false){
                h_degree[arr[x+1][y-1]] += 1;
            }
            if(x+1 < t && y+1 < t && hint[arr[x+1][y+1]] == false)
{
                h_degree[arr[x+1][y+1]] += 1;
            }
        }
    }

    vector<bool> sign;
    for(int i=0; i<mine.size(); i++){
        if(mine[i] != -1){
            sign.push_back(true);
        }
        else{
            sign.push_back(false);
        }
    }

    vector<int> h_lcv;
    for(int i=0; i<h_mrv.size(); i++){
        h_lcv.push_back(h_mrv[i]);
    }

    int place = 0;
    bool bomb = 0;
    pair<int, bool> p;
    p.first = place;
    p.second = 0;
    s.push(p);
    p.first = place;
    p.second = 1;
    s.push(p);
    bool flag = 0;
    int count = 0;

    if(MRV_true == false && MRV_degree_true == false && 
MRV_degree_LCV_true == false){
        while(place < mine.size()){
            count++;
            if(flag == 0){
                place = s.top().first;
                bomb = s.top().second;
                s.pop();
            }
            else{
                flag = 0;
            }
            int temp;
            temp = minesweeper(mine, hint, place, bomb);
            if(temp != place){
                place = temp;
            }
            else{
                place = s.top().first;
                bomb = s.top().second;
                s.pop();
                temp = minesweeper(mine, hint, place, bomb);
                place = temp;
            }

            if(place == mine.size() && check_hint_zero(mine, hint) 
== true){
                break;
            }
            else if(place != mine.size() && hint[place] == true){
                p.first = place;
                p.second = 0;
                s.push(p);
            }
            else if(place != mine.size()){
                p.first = place;
                p.second = 0;
                s.push(p);
                p.first = place;
                p.second = 1;
                s.push(p);
            }
            else{
                place = s.top().first;
                bomb = s.top().second;
                s.pop();
                flag = 1;
            }
        }
    }
    else if(MRV_true == true){
        stack<int> walkthrough;
        while(place < mine.size()){
            count++;
            if(flag == 0){
                place = s.top().first;
                bomb = s.top().second;
                s.pop();
            }
            else{
                while(walkthrough.top() != place){
                    if(mine[walkthrough.top()] == 1){
                        change_hint(mine, hint, walkthrough.top(), 
0);
                    }
                    mine[walkthrough.top()] = -1;
                    h_mrv[walkthrough.top()] = 2;
                    walkthrough.pop();
                }
                if(mine[walkthrough.top()] == 1){
                    change_hint(mine, hint, walkthrough.top(), 0);
                }
                mine[walkthrough.top()] = -1;
                h_mrv[walkthrough.top()] = 1;
                walkthrough.pop();
                flag = 0;
            }
            walkthrough.push(place);
            int temp;
            temp = minesweeper_mrv(mine, hint, place, bomb, 
h_mrv);
            if(temp != place){
                place = temp;
            }
            else{
                place = s.top().first;
                bomb = s.top().second;
                s.pop();
                temp = minesweeper_mrv(mine, hint, place, bomb, 
h_mrv);
                place = temp;
            }

            if(place == mine.size() && check_hint_zero(mine, hint) 
== true){
                break;
            }
            else if(place != mine.size() && hint[place] == true){
                p.first = place;
                p.second = 0;
                s.push(p);
            }
            else if(place != mine.size() && h_mrv[place] == 1){
                p.first = place;
                p.second = 1;
                s.push(p);
            }
            else if(place != mine.size() && h_mrv[place] == 0){
                p.first = place;
                p.second = 0;
                s.push(p);
            }
            else if(place != mine.size()){
                p.first = place;
                p.second = 0;
                s.push(p);
                p.first = place;
                p.second = 1;
                s.push(p);
            }
            else{
                place = s.top().first;
                bomb = s.top().second;
                s.pop();
                flag = 1;
            }
        }
    }
    else if(MRV_degree_true == true){
        stack<int> walkthrough;
        while(place < mine.size()){
            count++;
            if(flag == 0){
                place = s.top().first;
                bomb = s.top().second;
                s.pop();
```

```
      }
      else{
        while(walkthrough.top() != place){
          if(mine[walkthrough.top()] == 1){
            change_hint(mine, hint, walkthrough.top(), 
0);
          }
          mine[walkthrough.top()] = -1;
          h_mrv[walkthrough.top()] = 2;
          sign[walkthrough.top()] = false;
          walkthrough.pop();
        }
        if(mine[walkthrough.top()] == 1){
          change_hint(mine, hint, walkthrough.top(), 0);
        }
        mine[walkthrough.top()] = -1;
        h_mrv[walkthrough.top()] = 1;
        sign[walkthrough.top()] = false;
        walkthrough.pop();
        flag = 0;
      }
      walkthrough.push(place);
      int temp;
      temp = minesweeper_degree(mine, hint, place, bomb, 
h_mrv, h_degree, sign);
      if(temp != place){
        place = temp;
      }
      else{
        place = s.top().first;
        bomb = s.top().second;
        s.pop();
        cout<<place<<" "<<bomb<<endl;
        place = temp;
      }
```

```
      if(place == mine.size() && check_hint_zero(mine, hint) 
== true && mine_equal(mine, mines)){
        break;
      }
      else if(place != mine.size() && hint[place] == true){
        p.first = place;
        p.second = 0;
        s.push(p);
      }
      else if(place != mine.size() && h_mrv[place] == 1){
        p.first = place;
        p.second = 1;
        s.push(p);
      }
      else if(place != mine.size() && h_mrv[place] == 0){
        p.first = place;
```

```
                p.second = 0;
                s.push(p);
            }
            else if(place != mine.size()){
                p.first = place;
                p.second = 0;
                s.push(p);
                p.first = place;
                p.second = 1;
                s.push(p);
            }
            else{
                place = s.top().first;
                bomb = s.top().second;
                s.pop();
                flag = 1;
            }
        }
```

```
    }
    else{
        stack<int> walkthrough;
        while(place < mine.size()){
            count++;
            if(flag == 0){
                place = s.top().first;
                bomb = s.top().second;
                s.pop();
            }
            else{
                while(walkthrough.top() != place){
                    if(mine[walkthrough.top()] == 1){
                        change_hint(mine, hint, walkthrough.top(), 0);
                    }
                    mine[walkthrough.top()] = -1;
                    h_mrv[walkthrough.top()] = 2;
                    sign[walkthrough.top()] = false;
                    walkthrough.pop();
                }
                if(mine[walkthrough.top()] == 1){
                    change_hint(mine, hint, walkthrough.top(), 0);
                }
                mine[walkthrough.top()] = -1;
                h_mrv[walkthrough.top()] = 1;
                sign[walkthrough.top()] = false;
                walkthrough.pop();
                flag = 0;
            }
            walkthrough.push(place);
            int temp;
```

```
      temp = minesweeper_degree(mine, hint, place, bomb, h_mrv, h_degree, sign);
      if(temp != place){
        place = temp;
      }
      else{
        place = s.top().first;
        bomb = s.top().second;
        s.pop();
        cout<<place<<" "<<bomb<<endl;
        place = temp;
      }

      if(place == mine.size() && check_hint_zero(mine, hint) == true && mine_equal(mine, mines)){
        break;
      }
      else if(place != mine.size() && hint[place] == true){
        p.first = place;
        p.second = 0;
        s.push(p);
      }
      else if(place != mine.size() && h_mrv[place] == 1){
        p.first = place;
        p.second = 1;
        s.push(p);
      }
      else if(place != mine.size() && h_mrv[place] == 0){
        p.first = place;
        p.second = 0;
        s.push(p);
      }
      else if(place != mine.size()){
        int a0 = find_constraint(mine, hint, h_mrv, place, bomb);
        int a1 = find_constraint(mine, hint, h_mrv, place, bomb);
        if(a0>a1){
          p.first = place;
          p.second = 1;
          s.push(p);
          p.first = place;
          p.second = 0;
          s.push(p);
        }
        else{
          p.first = place;
          p.second = 0;
          s.push(p);
          p.first = place;
          p.second = 1;
          s.push(p);
```

```
                }
            }
            else{
                place = s.top().first;
                bomb = s.top().second;
                s.pop();
                flag = 1;
            }
        }
    }

    cout<<endl;
    output_board(mine);

    end = clock();
    cpu_time_used = (double)(end-start)/CLOCKS_PER_SEC;

    cout<<endl<<"cpu time: "<<cpu_time_used<<endl;
    cout<<"expand node: "<<count<<endl;

    return 0;
}
```