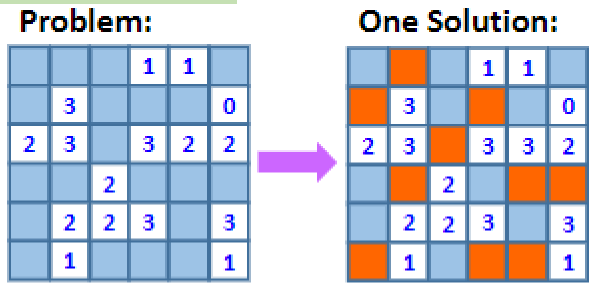
# 人工智慧概論

## 0613413 蔡怡君

**- 實驗目的：** 踩地雷，Minesweeper，給定一個 Mine Map，去找到結果，例如：



**- 實驗方法：** (depth-first search + forward checking + Heuristic)

- 使用的概念：在 explored set 和 frontier 都沒有相同 state 的 node ( 直接使用 table 去記錄這個點有沒有被 visit 過 )：將其加入 frontier !

1. **最原始 Depth First search：** 只有在 push 進去前去檢查，是否符合 local constraint 的需求，例如是否 hint 周圍的 mine 數量小於或是等於 hint mine。
2. **Depth First Search + MRV Heuristic：** 看 Domain 的關係， Domain= 1 的點比較早被 Assigned。
3. **Depth First Search + Degree Heuristic：** 看這個每一格的 Variable 跟幾個 constraint 有關，Constraint 越多的點越早被 Assigned。
4. **Depth First Search + 自定義 Heuristic：** 定義 TNT 越多的 State 應該要越先跑，。
5. **Depth First Search + forward checking：** 去檢查 Lower bound 跟 upper bound，並且條件性 Assigned。
6. **Depth First Search + forward checking + MRV Heuristic。**
7. **Depth First Search + forward checking + Degree Heuristic。**
8. **Depth First Search + forward checking + 自定義 Heuristic。**

## - 結果觀察：

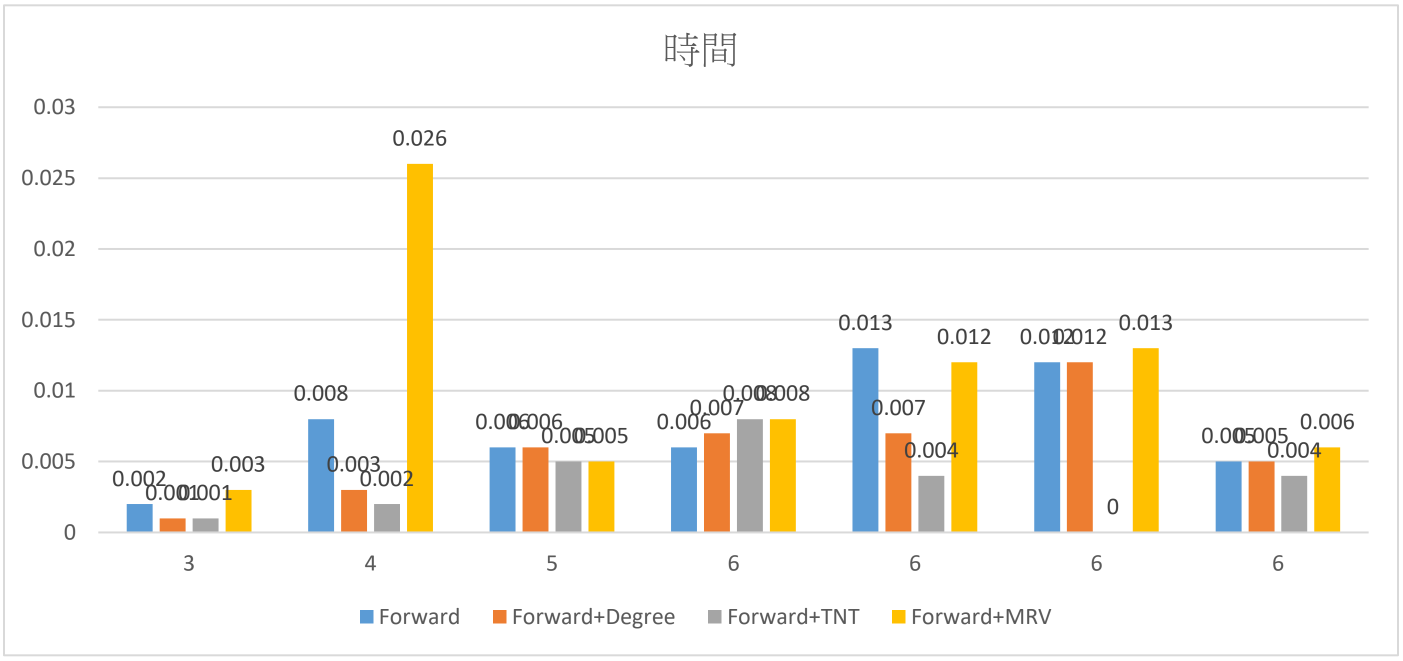
Input：

| | n= | |
|---|---|---|
| 第1筆測資 | 3 | 3 3 2 2 -1 -1 -1 -1 -1 -1 -1 -1 |
| 第2筆測資 | 4 | 4 4 5 2 -1 -1 1 -1 3 -1 -1 -1 -1 -1 -1 1 -1 1 -1 |
| 第3筆測資 | 5 | 5 5 7 2 -1 1 -1 -1 -1 -1 3 -1 1 2 -1 -1 -1 -1 -1 4 3 -1 -1 -1 2 -1 1 0 |
| 第4筆測資 | 6 | 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 |
| 第5筆測資 | 6 | 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 |
| 第6筆測資 | 6 | 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 |
| 第7筆測資 | 6 | 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 |

## 時間分析(Time )：

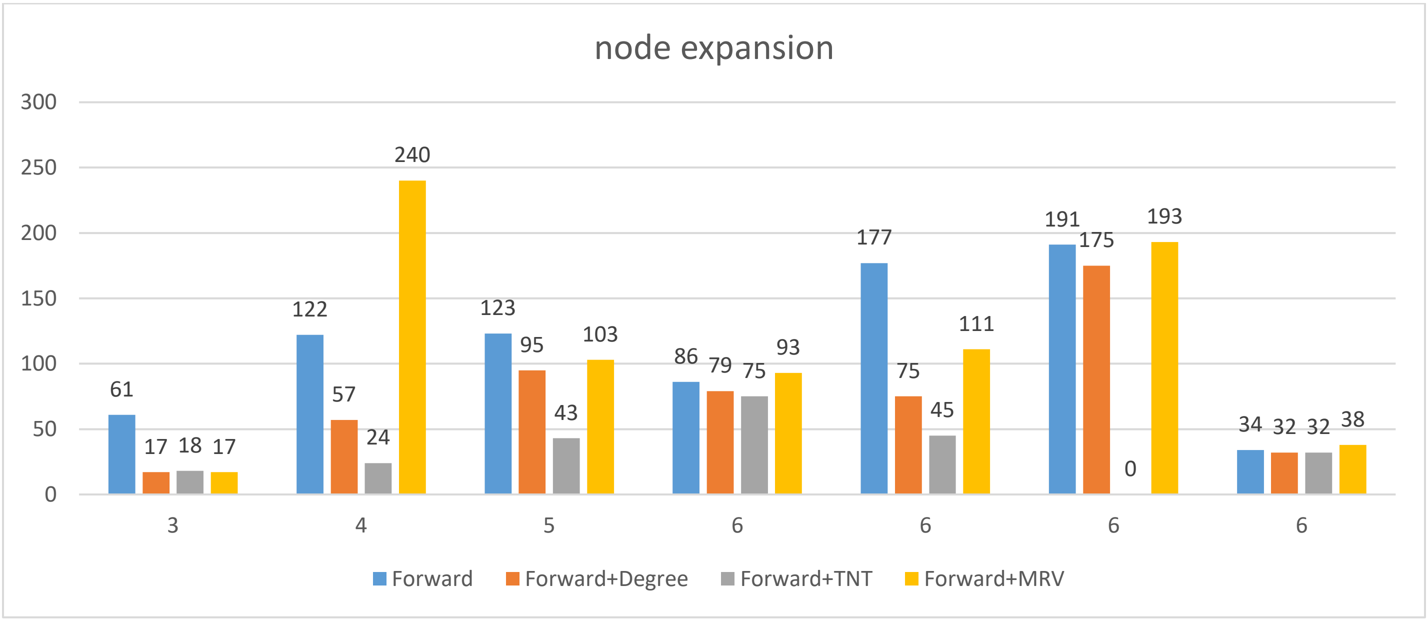
**紅色為最慢的，黃色為表現最佳的。**

| | Time | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| n= | Original | Forward | Forward+Degree | Forward+TNT | Forward+MRV | Degree | TNT | MRV |
| 3 | 0.055 | 0.002 | 0.001 | 0.001 | 0.003 | 0.003 | 0.003 | 0.021 |
| 4 | 0.005 | 0.008 | 0.003 | 0.002 | 0.026 | 2.178 | 16.623 | X |
| 5 | 0.014 | 0.006 | 0.006 | 0.005 | 0.005 | X | 0.001 | X |
| 6 | X | 0.006 | 0.007 | 0.008 | 0.008 | X | X | X |
| 6 | X | 0.013 | 0.007 | 0.004 | 0.012 | X | X | X |
| 6 | X | 0.012 | 0.012 | X | 0.013 | X | X | X |
| 6 | X | 0.005 | 0.005 | 0.004 | 0.006 | X | X | X |



## Node expansion:

| | node expand | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| n= | Original | Forward | Forward+Degree | Forward+TNT | Forward+MRV | Degree | TNT | MRV |
| 3 | 268 | 61 | 17 | 18 | 17 | 71 | 18 | 146 |
| 4 | 114 | 122 | 57 | 24 | 240 | 1921 | 4681 | X |
| 5 | 213 | 123 | 95 | 43 | 103 | X | 122 | X |
| 6 | X | 86 | 79 | 75 | 93 | X | X | X |
| 6 | X | 177 | 75 | 45 | 111 | X | X | X |
| 6 | X | 191 | 175 | X | 193 | X | X | X |
| 6 | X | 34 | 32 | 32 | 38 | X | X | X |



## 表現總結

| | |
|---|---|
| **Original** | 效率非常差，因為需要去展開所有的 node，展開到 child node 後才能去檢查。 |
| **Degree** | 表現普通。 |
| **TNT** | 在 n=4 時花費的時間較長，但是 n=5 的時候表現為最佳的。 |
| **MRV** | 只有在 n=3 的時候可以展開。 |
| **Forward** | 表現普通。 |
| **Forward + Degree** | 表現佳，但通常不是最快速的。 |
| **Forward + TNT** | 在大多的表現內是最好的，但是在 n=6 的第三個測資的時候，因為時間過於長，所以無法記錄下來。 |
| **Forward + MRV** | 表現比其他的 Forward + Heuristic 都還要差，但是可以到 n=6。 |

## - 結果分析：

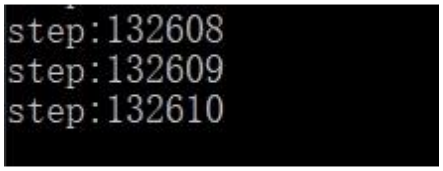
從上述的數值，看出使用 Heuristic 並不會比較好，甚至還會較糟，例如單純使用 MRV、Degree，而去**觀察 MRV 在此情況中的效率較差**，原因來自於我大多在 assigned value 的時候會直接把 Domain = 1 並且給值，所以如果用 MRV，**反而多用了沒必要的檢查**。

在此 CSP 的問題中，使用 Forward checking 來解此問題為最佳的，因為可以省去很多不必要的 node 去展開，而且如果當 lower bound = hint 的時候，可以直接將 unassigned 的值全部給 0，或是另一情況 upper bound = hint 的時候，也可以將 unassigned 的值全部給 1，可以快速的將 Domain 跟 Value 的值處理，**而最後的結果是 forward checking 跟 TNT 的結合是表現最佳的**，但 **TNT 的這個 Heuristic function** 在 n=6 的第三個測資中卻在一定的時間中，**跑不出結果**，所以 TNT 的效率具有不確定性，所以如果以**確定性**來看，會選擇 Forward checking + Degree Heuristic。

## 實驗中的遇到的問題：

**Question 1：在使用 Original DFS (C++)的時候，一直有去找原因為甚麼在 n=6 的時候，跑不出答案( 此下圖跑了 6 個小時的結果 )，然而別人用 python 寫，卻跑的出來。**

**Answer 1：** 去手動計算大概需要多少運算的時候，發現 Original 的展開 node 本來就要展開到 child 或是檢查 local constraint 符不符合時，所以展開的 node 本來就需要比較大。



```
step:132608
step:132609
step:132610
```

**Remaining Question 2：如果結合兩個、三個以上的 Heuristic function，結果會比較佳嗎？**

**Remaining Question 3：TNT(自訂 function)的問題出在哪裡？TNT 是用 priority queue 去放 state，然後 TNT 較多的先開始去展開。**

## Thing I have learned：

這次的作業學習到的很多，因為難度蠻高的，有些小 Bug 改了蠻久，然後 Constraint 類的問題，學道透過 Forward checking 的方式去處理，可以避免掉一些不必要展開的 node，而且 Forward checking 的能力比 Heuristic 還要強 （ 在 CSP 的問題上 ），在此次的問題中，會有一些的疑惑就是 Forward checking  的定義跟 Heuristic 的定義會不會有點模糊，還存在一些疑惑，像是 Forward checking 是避免掉不必要的 node，那 Heuristic 是去挑選出會造成較佳的結果嗎？AI 的作業其實都很有趣，因為都是將演算法的一些概念應用到遊戲上面，去展示最後的結果，又或是透過不同的分析去了解這個遊戲會更適合哪個演算法去解析。

## 程式碼

```cpp
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<algorithm>
#include<vector>
#include<stack>
#include<queue>
#include<time.h>
#include<iomanip>
#define N 6
using namespace std;

/*
    author : 蔡怡君
    content : Minesweeper using Backtrack Search
*/

// mine map
int mine[N][N],row,column;


struct node
{
    int assigned_node;
    int TNT;
    int domain[N][N]; // domain's number {0,1} = 2
    int value[N][N]; // value : 0 or 1 (have mines)
}first;


struct point
{
    int x,y;
    int degree;
}dir[8];

vector<point>hint;

//point **goal;

void initial()
{
    dir[0]={-1,-1};
    dir[1]={-1,0};
    dir[2]={-1,1};
    dir[3]={0,-1};
    dir[4]={0,1};
    dir[5]={1,-1};
    dir[6]={1,0};
    dir[7]={1,1};

    first.assigned_node = 0;
    first.TNT = 0;
    for(int i = 0 ; i < row ; i ++){
        for(int j = 0 ; j < column ; j ++){
            if(mine[i][j] == -1){
                first.domain[i][j] = 2;
                first.value[i][j] = -1;
            }

            else
            {   // means hint
                point h = {i,j};
                hint.push_back(h);
                first.domain[i][j] = 1;
                first.value[i][j] = mine[i][j];
            }
        }
    }
}

bool operator == (const node &p1,const node &p2)
{
    if(p1.TNT != p2.TNT )
        return false;
    for(int i = 0 ; i < row ; i ++){
        for(int j = 0 ; j < column ; j ++){
            if(p1.domain[i][j] != p2.domain[i][j])
                return false;
            if(p1.value[i][j] != p2.value[i][j])
                return false;
        }
    }
    return true;
}

bool isVaild(int i,int j)
{
    if(mine[i][j] != -1) return false;
    return ( (i >= 0 && i < row ) && (j >= 0 && j < column) );
}

node copy_node(node A)
{
    node tmp;
    for(int i = 0 ; i < row ;i ++){
        for(int j = 0 ; j < column ; j ++){
            tmp.domain[i][j] = A.domain[i][j];
            tmp.value[i][j] = A.value[i][j];
        }
    }
    tmp.assigned_node = A.assigned_node;
    tmp.TNT = A.TNT;
    return tmp;
}

/*
// local check
bool checkVaild(node now,int num_TNT)
{
    // global check
    if(now.TNT > num_TNT) return false;
    for(int i = 0; i < row ; i ++){
        for(int j = 0 ; j < column ; j ++){
            if(mine[i][j] != -1)
            {   // local check 8 directions 周圍的 TNT
                int TNT=0;
                for(int k = 0; k < 8 ; k ++){
                    if(isVaild(i + dir[k].x,j +dir[k].y))
                        if(now.value[i + dir[k].x][j +dir[k].y ] == 1)
                            TNT ++;
                }
                if(TNT > mine[i][j])
                    return false;
            }
        }
    }
    return true;
}
*/

bool checkVaild(node now, int num_TNT, vector<point> hhint)
{
    bool all_assigned = true;
    if(now.TNT > num_TNT)     return false;
    for(int i = 0 ; i < hhint.size() ; i ++)
    {
        // check hint 的八維
        int TNT = 0;
        for(int j = 0 ; j < 8 ; j ++){
            int px = hhint[i].x + dir[j].x;
            int py = hhint[i].y + dir[j].y;
            if(isVaild(px , py))
            {
                if(now.value[px][py] == 1) // 放炸彈的地方
                    TNT ++ ;
                if(now.domain[px][py] != 1) // 如果八個方位都 assigned 完
                    all_assigned = false;
            }

        }
        if(TNT > mine[hhint[i].x][hhint[i].y])
            return false;
        // 八個方位都 assigned 了 卻不符合 TNT -> false
        if(all_assigned == true && TNT != mine[hhint[i].x][hhint[i].y] )
            return false;
    }
    return true;
}

void print_solution(node result)
{
    for(int i = 0 ; i < row ;i ++){
        for(int j = 0 ; j < column ; j ++){
            if(mine[i][j] != -1)
                cout<<mine[i][j]<<" ";
            else if(result.value[i][j] == 1)
                cout<<"*"<<" ";
            else
                cout<<" "<<" ";
        }
        cout<<endl;
    }
}

// using for checking
void print_do(node result)
{
    cout<<"domain: \n";
    for(int i = 0 ; i < row ;i ++){
        for(int j = 0 ; j < column ; j ++){
                cout<<result.domain[i][j]<<" ";
        }
        cout<<endl;
    }
}

// using for checking
void print_value(node result)
{
    cout<<"value: \n";
    for(int i = 0 ; i < row ;i ++){
        for(int j = 0 ; j < column ; j ++){
            if(mine[i][j] == -1 && result.value[i][j] == 1)
                cout<<" *"<<" ";
            else
                cout<<setw(2)<<result.value[i][j]<<" ";
        }
        cout<<endl;
    }
}


bool check_solution(node result)
{
    for(int i = 0; i < row ; i ++){
        for(int j = 0 ; j < column ; j ++){
            if(mine[i][j] != -1)
            {   // local check 8 directions 周圍的 TNT
                int TNT=0;
                for(int k = 0; k < 8 ; k ++){
                    if(isVaild(i + dir[k].x,j +dir[k].y))
                        if(result.value[i + dir[k].x][j +dir[k].y ] == 1)
                            TNT ++;
                }
                if(TNT != mine[i][j])
                    return false;
            }
        }
    }
    return true;
}

bool isExplored(stack<node> explored , stack<node>front, node now)
{
    // 在 Explored set 裡面
    while(!explored.empty())
    {
        node current=explored.top();
        explored.pop();
        // 如果 current == now 一模一樣
        if(current == now)
        {
            //cout<<"the same"<<endl;
            return false;
        }
    }
    // 在 Frontier 裏頭
    while(!front.empty())
    {
        node current = front.top();
        front.pop();
        // 如果 current == now 一模一樣
        if(current == now)
        {
            //cout<<"the same"<<endl;
            return false;
        }
    }
    return true;
}

bool isExplored(stack<node> explored , priority_queue<node>front, node now)
{
    // 在 Explored set 裡面
    while(!explored.empty())
    {
        node current=explored.top();
        explored.pop();
        // 如果 current == now 一模一樣
        if(current == now)
        {
            //cout<<"the same"<<endl;
            return false;
        }
    }
    // 在 Frontier 裏頭
    while(!front.empty())
    {
        node current = front.top();
        front.pop();
        // 如果 current == now 一模一樣
        if(current == now)
        {
            //cout<<"the same"<<endl;
            return false;
        }
    }
    return true;
}

// original without forward checking / heuristic
void find_solution(int r , int c , int num)
{
    stack<node> front;
    stack<node> explored;
    int step = 1;
    front.push(first);
    //bool flag = true;
    while(!front.empty())
    {
        node current = front.top();
        explored.push(current);
        front.pop();
        //cout<<"layer9 :"<<step<<endl;
        for(int i = 0 ; i < r ; i ++){
            for(int j = 0 ; j < c ; j++){
                if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
                {
                    //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
                    for(int option = 0; option < 2 ; option ++){
                        node now = copy_node(current);
                        now.domain[i][j] = 1;
                        now.value[i][j] = option;
                        if(option == 1)
                            now.TNT++;
                        if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
                        {
                            // 沒有被 explored 過的才能加進去！
                            if(isExplored(explored,front,now))
                            {
                                if(now.TNT == num){
                                    if(check_solution(now)){
                                        print_solution(now);
                                        cout<<"node Expand:"<<step<<endl;
                                        return;
                                    }
                                    else
                                        continue;
                                }
                                front.push(now);
                                step++;
                            }
                        }
                    }
                }
            }
        }
    }
    cout<<"No Solution!"<<endl;
}

void find_forward_solution(int r, int c, int num)
{
    stack<node> front;
    stack<node> explored;
    front.push(first);
    int step = 1;
    while(!front.empty())
    {
        node current = front.top();
        explored.push(current);
        front.pop();

        // --- forward checking

        bool flag = true;
        for(int i = 0 ; i < hint.size() ; i ++){
            int lowerbound = 0;
            int upperbound = 0;
            point no = hint[i];
            // direction
            //cout<<"forward x,y :"<< no.x<<","<<no.y<<endl;
            for(int j = 0 ; j < 8 ; j ++){
                int x = no.x + dir[j].x;
                int y = no.y + dir[j].y;
                if(!isVaild(x,y))
                    continue;
                if ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
                    lowerbound ++;
                else if(current.domain[x][y] == 2) // unassigned
                    upperbound ++;
            }
            upperbound += lowerbound;
            // lower bound & upper bound
            if( lowerbound > mine[no.x][no.y] )
            {
                flag = false;
                //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
                break;
            }
            else if (lowerbound == mine [no.x][no.y])
            {
                // 代表其他 unassinged 的值 都要成為 0
                for(int j = 0 ; j < 8 ; j ++){
                    int x = no.x + dir[j].x;
                    int y = no.y + dir[j].y;
                    if(!isVaild(x,y))
                        continue;
                    if(current.domain[x][y] == 2){
                        current.domain[x][y] = 1 ;
                        current.value[x][y] = 0 ;
                    }
                }
            }
            if( upperbound < mine[no.x][no.y])
            {
                flag = false;
                //cout<<"upperbound < mine[no.x][no.y]"<<endl;
                break;
            }
            else if( upperbound == mine[no.x][no.y])
            {
                // 代表其他 unassigned 的值 都要變成 1
                for(int j = 0 ; j < 8 ; j ++){
                    int x = no.x + dir[j].x;
                    int y = no.y + dir[j].y;
                    if(!isVaild(x,y))
                        continue;
                    if(current.domain[x][y] == 2){
                        current.domain[x][y] = 1 ;
                        current.value[x][y] = 1 ;
                        current.TNT++;
                    }
                }
            }
        }
        // 代表這個 state 不滿足 state 繼續 pop
        if(!flag) continue;

        // 檢查是不是答案
        if( current.TNT == num )
        {
            if(check_solution(current)){
                print_solution(current);
                cout<<"node Expand:"<<step<<endl;
                return;
            }
            continue;
        }
        // to assign value !
        for(int i = 0 ; i < r ; i ++){
            for(int j = 0 ; j < c ; j++){
                if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
                {
                    //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
                    // 去給這 value 是 1(mine) 還是 0
                    for(int option = 1; option >= 0  ; option --){
                        node now = copy_node(current);
                        now.domain[i][j] = 1;
                        now.value[i][j] = option;
                        if(option == 1)
                            now.TNT++;
                        if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
                        {
                            // 沒有被 explored 過的才能加進去！
                            if(isExplored(explored,front,now))
                            {
                                // 如果 TNT 數量 == 炸彈的數量 檢查 solution
                                if( now.TNT == num ){
                                    if(check_solution(now)){
                                        print_solution(now);
                                        cout<<"node Expand:"<<step<<endl;
                                        return;
                                    }
                                    else
                                        continue;
                                }
                                front.push(now);
                                step++;
                            }
                        }
                    }
                }
            }
        }
    }

}

// using for priority queue
bool operator < (const point &p1,const point &p2){ return p1.degree < p2.degree;}
bool operator > (const point &p1,const point &p2){ return p1.degree > p2.degree;}

// forward function with Degree Heuristic
void find_forward_solution_Degree(int r, int c, int num)
{
    stack<node> front;
    stack<node> explored;
    front.push(first);
    int step = 1;
    while(!front.empty())
    {
        node current = front.top();
        explored.push(current);
        front.pop();

        // --- priorty_queue using for degree heuristic
        priority_queue<point,vector<point>,greater<point> > de;
        for(int i = 0 ; i < r ; i ++){
            for(int j = 0 ; j < c ; j ++){
                if(mine[i][j] != -1) continue;
                point p = { i , j , 0};
                for(int k = 0 ; k < 8 ; k ++)
```

```
                    if(isVaild( i + dir[k].x , j + dir[k].y ))
                        if(mine[i + dir[k].x][j + dir[k].y] != -1)
                            p.degree ++;
                de.push(p);
            }
        }

        // --- forward checking
        bool flag = true;
        for(int i = 0 ; i < hint.size() ; i ++){
            int lowerbound = 0;
            int upperbound = 0;
            point no = hint[i];
            // direction
            //cout<<"forward x,y :"<< no.x<<","<<no.y<<endl;
            for(int j = 0 ; j < 8 ; j ++){
                int x = no.x + dir[j].x;
                int y = no.y + dir[j].y;
                if(!isVaild(x,y))
                    continue;
                if ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
                    lowerbound ++;
                else if(current.domain[x][y] == 2) // unassigned
                    upperbound ++;
            }
            upperbound += lowerbound;
            // lower bound & upper bound
            if( lowerbound > mine[no.x][no.y] )
            {
                flag = false;
                //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
                break;
            }
            else if (lowerbound == mine [no.x][no.y])
            {
                // 代表其他 unassinged 的值 都要成為 0
                for(int j = 0 ; j < 8 ; j ++){
                    int x = no.x + dir[j].x;
                    int y = no.y + dir[j].y;
                    if(!isVaild(x,y))
                        continue;
                    if(current.domain[x][y] == 2){
                        current.domain[x][y] = 1 ;
                        current.value[x][y] = 0 ;
                    }
                }
            }
            if( upperbound < mine[no.x][no.y])
            {
                flag = false;
                //cout<<"upperbound < mine[no.x][no.y]"<<endl;
                break;
            }
            else if( upperbound == mine[no.x][no.y])
            {
                // 代表其他 unassigned 的值 都要變成 1
                for(int j = 0 ; j < 8 ; j ++){
                    int x = no.x + dir[j].x;
                    int y = no.y + dir[j].y;
                    if(!isVaild(x,y))
                        continue;
                    if(current.domain[x][y] == 2){
                        current.domain[x][y] = 1 ;
                        current.value[x][y] = 1 ;
                        current.TNT++;
                    }
                }
            }
        }
        // 代表這個 state 不滿足 state 繼續 pop
        if(!flag) continue;

        // 檢查是不是答案
        if( current.TNT == num )
        {
            if(check_solution(current)){
                print_solution(current);
                cout<<"node Expand:"<<step<<endl;
                return;
            }
            continue;
        }

        // to assign value！
        while(!de.empty()){
            point n = de.top();
            de.pop();
            int i = n.x, j = n.y;

            if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
            {
                // 去給這 value 是 1(mine) 還是 0
                for(int option = 1; option >= 0  ; option --){
                    node now = copy_node(current);
                    now.domain[i][j] = 1;
                    now.value[i][j] = option;
                    if(option == 1)
                        now.TNT++;
                    // // 沒有被 explored 過的才能加進去！ 且 if 這個點有符合 17 constraints
                    if(checkVaild(now,num,hint)&&isExplored(explored,front,now))
                    {
                        // 如果 TNT 數量 == 炸彈的數量 檢查 solution
                        if( now.TNT == num ){
                            if(check_solution(now)){
                                print_solution(now);
                                cout<<"node Expand:"<<step<<endl;
                                return;
                            }
                            else
                                continue;
                        }
                        front.push(now);
                        step++;
                    }
                }
            }
        }
    }

}


```

```
bool operator < (const node &p1,const node &p2){ return p1.TNT < p2.TNT;}
bool operator > (const node &p1,const node &p2){ return p1.TNT > p2.TNT;}

// forward function with TNT Heuristic
void find_forward_solution_TNT  (int r, int c, int num)
{
    priority_queue<node> front;
    stack<node> explored;
    front.push(first);
    int step =1;
    while(!front.empty())
    {
        node current = front.top();
        explored.push(current);
        front.pop();
        if(current.TNT == num)
            for(int i = 0 ; i < r; i ++)
                for(int j = 0 ; j <c ; j ++)
                    if(current.domain[i][j] == 2){
                        current.domain[i][j] = 1;
                        current.value[i][j] = 0;
                    }

        // --- forward checking
        bool flag = true;
        for(int i = 0 ; i < hint.size() ; i ++){
            int lowerbound = 0;
            int upperbound = 0;
            point no = hint[i];
            // direction
            //cout<<"forward x,y :"<< no.x<<","<<no.y<<endl;
            for(int j = 0 ; j < 8 ; j ++){
                int x = no.x + dir[j].x;
                int y = no.y + dir[j].y;
                if(!isVaild(x,y))
                    continue;
                if ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
                    lowerbound ++;
                else if(current.domain[x][y] == 2) // unassigned
                    upperbound ++;
            }
            upperbound += lowerbound;
            // lower bound & upper bound
            if( lowerbound > mine[no.x][no.y] )
            {
                flag = false;
                //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
                break;
            }
            else if (lowerbound == mine [no.x][no.y])
            {
                // 代表其他 unassinged 的值 都要成為 0
                for(int j = 0 ; j < 8 ; j ++){
                    int x = no.x + dir[j].x;
                    int y = no.y + dir[j].y;
                    if(!isVaild(x,y))
                        continue;
                    if(current.domain[x][y] == 2){
                        current.domain[x][y] = 1 ;
                        current.value[x][y] = 0 ;
                    }
                }
            }
            if( upperbound < mine[no.x][no.y])
            {
                flag = false;
                //cout<<"upperbound < mine[no.x][no.y]"<<endl;
                break;
            }
            else if( upperbound == mine[no.x][no.y])
            {
                // 代表其他 unassigned 的值 都要變成 1
                for(int j = 0 ; j < 8 ; j ++){
                    int x = no.x + dir[j].x;
                    int y = no.y + dir[j].y;
                    if(!isVaild(x,y))
                        continue;
                    if(current.domain[x][y] == 2){
                        current.domain[x][y] = 1 ;
                        current.value[x][y] = 1 ;
                        current.TNT++;
                    }
                }
            }
        }
        // 代表這個 state 不滿足 state 繼續 pop
        if(!flag) continue;

        // 檢查是不是答案
        if( current.TNT == num )
        {
            if(check_solution(current)){
                print_solution(current);
                cout<<"node Expand:"<<step<<endl;
                return;
            }
            continue;
        }

        // to assign value !
        for(int i = 0 ; i < r ; i ++){
            for(int j = 0 ; j < c ; j++){
                if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
                {
                    //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
                    // 去給這 value 是 1(mine) 還是 0
                    for(int option = 1; option >= 0  ; option --){
                        node now = copy_node(current);
                        now.domain[i][j] = 1;
                        now.value[i][j] = option;
                        if(option == 1)
                            now.TNT++;
                        if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
                        {
                            // 沒有被 explored 過的才能加進去！
                            if(isExplored(explored,front,now))
                            {
                                // 如果 TNT 數量 == 炸彈的數量 檢查 solution
                                if( now.TNT == num ){
                                    if(check_solution(now)){
                                        print_solution(now);
                                        cout<<"node Expand:"<<step<<endl;
```

```
                                            return;
                                        }
                                        else
                                            continue;
                                    }
                                    front.push(now);
                                    step++;
                                }
                            }
                        }
                    }
                }
            }
        }

}

// forward function with MRV Heuristic
void find_forward_solution_MRV(int r, int c, int num)
{
    stack<node> front;
    stack<node> explored;
    front.push(first);
    int step = 1;
    while(!front.empty())
    {
        node current = front.top();
        explored.push(current);
        front.pop();

        // --- priorty_queue using for degree heuristic
        priority_queue<point,vector<point>,greater<point> > de;
        for(int i = 0 ; i < r ; i ++){
            for(int j = 0 ; j < c ; j ++){
                point p = { i , j , current.domain[i][j] };
                de.push(p);
            }
        }

        // --- forward checking
        bool flag = true;
        for(int i = 0 ; i < hint.size() ; i ++){
            int lowerbound = 0;
            int upperbound = 0;
            point no = hint[i];
            // direction
            //cout<<"forward x,y :"<< no.x<<","<<no.y<<endl;
            for(int j = 0 ; j < 8 ; j ++){
                int x = no.x + dir[j].x;
                int y = no.y + dir[j].y;
                if(!isVaild(x,y))
                    continue;
                if ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
                    lowerbound ++;
                else if(current.domain[x][y] == 2) // unassigned
                    upperbound ++;
            }
            upperbound += lowerbound;
            // lower bound & upper bound
            if( lowerbound > mine[no.x][no.y] )
            {
                flag = false;
                //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
                break;
            }
            else if (lowerbound == mine [no.x][no.y])
            {
                // 代表其他 unassinged 的值  都要成為 0
                for(int j = 0 ; j < 8 ; j ++){
                    int x = no.x + dir[j].x;
                    int y = no.y + dir[j].y;
                    if(!isVaild(x,y))
                        continue;
                    if(current.domain[x][y] == 2){
                        current.domain[x][y] = 1 ;
                        current.value[x][y] = 0 ;
                    }
                }
            }
            if( upperbound < mine[no.x][no.y])
            {
                flag = false;
                //cout<<"upperbound < mine[no.x][no.y]"<<endl;
                break;
            }
            else if( upperbound == mine[no.x][no.y])
            {
                // 代表其他  unassigned 的值  都要變成 1
                for(int j = 0 ; j < 8 ; j ++){
                    int x = no.x + dir[j].x;
                    int y = no.y + dir[j].y;
                    if(!isVaild(x,y))
                        continue;
                    if(current.domain[x][y] == 2){
                        current.domain[x][y] = 1 ;
                        current.value[x][y] = 1 ;
                        current.TNT++;
                    }
                }
            }
        }
        // 代表這個 state 不滿足  state  繼續 pop
        if(!flag) continue;

        // 檢查是不是答案
        if( current.TNT == num )
        {
            if(check_solution(current)){
                print_solution(current);
                cout<<"node Expand:"<<step<<endl;
                return;
            }
            continue;
        }

        // to assign value !
        while(!de.empty()){
            point n = de.top();
            de.pop();
            int i = n.x, j = n.y;

            if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
            {
                // 去給這 value 是 1(mine) 還是 0
                for(int option = 1; option >= 0  ; option --){
                    node now = copy_node(current);
                    now.domain[i][j] = 1;
                    now.value[i][j] = option;
                    if(option == 1)
                        now.TNT++;
                    // // 沒有被 explored 過的才能加進去！ 且 if 這個點有符合 17 constraints
                    if(checkVaild(now,num,hint)&&isExplored(explored,front,now))
                    {
                        // 如果 TNT 數量 == 炸彈的數量 檢查 solution
                        if( now.TNT == num ){
                            if(check_solution(now)){
                                print_solution(now);
                                cout<<"node Expand:"<<step<<endl;
                                return;
                            }
                            else
                                continue;
                        }
                        front.push(now);
                        step ++;
                    }
                }
            }
        }
    }
}


int main()
{
    int num_mines;
    time_t start,end;
    double t;

    cin >> row >> column >> num_mines;

    for( int i = 0 ; i < row ; i ++ )
        for ( int j = 0 ; j < column ; j ++ )
            cin>>mine[i][j];

    initial();


    //---- original without forward & heuristic
    start = clock();
    find_solution(row,column,num_mines);
    end = clock();
    t = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time : %fs\n",t);

    //----  forward checking without heuristic
    cout<<"forward checking without heuristic\n";
    start = clock();
    find_forward_solution(row,column,num_mines);
    end = clock();
    t = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time : %fs\n",t);

    //----  forward checking with Degree heuristic
    cout<<"forward checking with Degree heuristic\n";
    start = clock();
    find_forward_solution_Degree(row,column,num_mines);
    end = clock();
    t = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time : %fs\n",t);



    //----  forward checking with uesr-defined heuristic
    cout<<"forward checking with TNT heuristic\n";
    start = clock();
    find_forward_solution_TNT(row,column,num_mines);
    end = clock();
    t = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time : %fs\n",t);

    //----  forward checking with uesr-defined heuristic
    cout<<"forward checking with MRV heuristic\n";
    start = clock();
    find_forward_solution_MRV(row,column,num_mines);
    end = clock();
    t = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time : %fs\n",t);

    return 0;
}
```

## 單純使用 Heuristic function 的 code：

```
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<algorithm>
#include<vector>
#include<stack>
#include<queue>
#include<time.h>
#include<iomanip>
#define N 6
using namespace std;

/*
    author : 蔡怡君
    content : Minesweeper using Backtrack Search
*/

// mine map
int mine[N][N],row,column;


struct node
{
    int assigned_node;
    int TNT;
    int domain[N][N]; // domain's number {0,1} = 2
    int value[N][N]; // value : 0 or 1 (have mines)
}first;


```

```cpp
struct point
{
    int x,y;
    int degree;
}dir[8];

vector<point>hint;

//point **goal;

void initial()
{
    dir[0]={-1,-1};
    dir[1]={-1,0};
    dir[2]={-1,1};
    dir[3]={0,-1};
    dir[4]={0,1};
    dir[5]={1,-1};
    dir[6]={1,0};
    dir[7]={1,1};

    first.assigned_node = 0;
    first.TNT = 0;
    for(int i = 0 ; i < row ; i ++){
        for(int j = 0 ; j < column ; j ++){
            if(mine[i][j] == -1){
                first.domain[i][j] = 2;
                first.value[i][j] = -1;
            }

            else
            {   // means hint
                point h = {i,j};
                hint.push_back(h);
                first.domain[i][j] = 1;
                first.value[i][j] = mine[i][j];
            }
        }
    }
}

bool operator == (const node &p1,const node &p2)
{
    if(p1.TNT != p2.TNT )
        return false;
    for(int i = 0 ; i < row ; i ++){
        for(int j = 0 ; j < column ; j ++){
            if(p1.domain[i][j] != p2.domain[i][j])
                return false;
            if(p1.value[i][j] != p2.value[i][j])
                return false;
        }
    }
    return true;
}

bool isVaild(int i,int j)
{
    if(mine[i][j] != -1) return false;
    return ( (i >= 0 && i < row ) && (j >= 0 && j < column) );
}

```

```
node copy_node(node A)
{
    node tmp;
    for(int i = 0 ; i < row ;i ++){
        for(int j = 0 ; j < column ; j ++){
            tmp.domain[i][j] = A.domain[i][j];
            tmp.value[i][j] = A.value[i][j];
        }
    }
    tmp.assigned_node = A.assigned_node;
    tmp.TNT = A.TNT;
    return tmp;
}

bool checkVaild(node now, int num_TNT, vector<point> hhint)
{
    bool all_assigned = true;
    if(now.TNT > num_TNT)     return false;
    for(int i = 0 ; i < hhint.size() ; i ++)
    {
        // check hint 的八維
        int TNT = 0;
        for(int j = 0 ; j < 8 ; j ++){
            int px = hhint[i].x + dir[j].x;
            int py = hhint[i].y + dir[j].y;
            if(isVaild(px , py))
            {
                if(now.value[px][py] == 1) // 放炸彈的地方
                    TNT ++ ;
                if(now.domain[px][py] != 1) // 如果八個方位都 assigned 完
                    all_assigned = false;
            }

        }
        if(TNT > mine[hhint[i].x][hhint[i].y])
            return false;
        // 八個方位都 assigned 了 卻不符合 TNT -> false
        if(all_assigned == true && TNT != mine[hhint[i].x][hhint[i].y] )
            return false;
    }
    return true;
}

void print_solution(node result)
{
    for(int i = 0 ; i < row ;i ++){
        for(int j = 0 ; j < column ; j ++){
            if(mine[i][j] != -1)
                cout<<mine[i][j]<<" ";
            else if(result.value[i][j] == 1)
                cout<<"*"<<" ";
            else
                cout<<" "<<" ";
        }
        cout<<endl;
    }
}

// using for checking
void print_do(node result)
{
    cout<<"domain: \n";
    for(int i = 0 ; i < row ;i ++){
        for(int j = 0 ; j < column ; j ++){
                cout<<result.domain[i][j]<<" ";
        }
        cout<<endl;
    }
}

// using for checking
void print_value(node result)
{
    cout<<"value: \n";
    for(int i = 0 ; i < row ;i ++){
        for(int j = 0 ; j < column ; j ++){
            if(mine[i][j] == -1 && result.value[i][j] == 1)
                cout<<" *"<<" ";
            else
                cout<<setw(2)<<result.value[i][j]<<" ";
        }
        cout<<endl;
    }
}


bool check_solution(node result)
{
    for(int i = 0; i < row ; i ++){
        for(int j = 0 ; j < column ; j ++){
            if(mine[i][j] != -1)
            {   // local check 8 directions 周圍的 TNT
                int TNT=0;
                for(int k = 0; k < 8 ; k ++){
                    if(isVaild(i + dir[k].x,j +dir[k].y))
                        if(result.value[i + dir[k].x][j +dir[k].y ] == 1)
                            TNT ++;
                }
                if(TNT != mine[i][j])
                    return false;
            }
        }
    }
    return true;
}

bool isExplored(stack<node> explored , stack<node>front, node now)
{
    // 在 Explored set 裡面
    while(!explored.empty())
    {
        node current=explored.top();
        explored.pop();
        // 如果 current == now 一模一樣
        if(current == now)
        {
            //cout<<"the same"<<endl;
            return false;
        }
    }
    // 在 Frontier 裏頭
    while(!front.empty())
    {
        node current = front.top();
        front.pop();
        // 如果 current == now 一模一樣
        if(current == now)
        {
            //cout<<"the same"<<endl;
            return false;
        }
    }
    return true;
}

bool isExplored(stack<node> explored , priority_queue<node>front, node now)
{
    // 在 Explored set 裡面
    while(!explored.empty())
    {
        node current=explored.top();
        explored.pop();
        // 如果 current == now 一模一樣
        if(current == now)
        {
            //cout<<"the same"<<endl;
            return false;
        }
    }
    // 在 Frontier 裏頭
    while(!front.empty())
    {
        node current = front.top();
        front.pop();
        // 如果 current == now 一模一樣
        if(current == now)
        {
            //cout<<"the same"<<endl;
            return false;
        }
    }
    return true;
}

// original without forward checking / heuristic
void find_solution(int r , int c , int num)
{
    stack<node> front;
    stack<node> explored;
    int step = 1;
    front.push(first);
    //bool flag = true;
    while(!front.empty())
    {
        node current = front.top();
        explored.push(current);
        front.pop();
        //cout<<"layer9 :"<<step<<endl;
        for(int i = 0 ; i < r ; i ++){
            for(int j = 0 ; j < c ; j++){
                if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
                {
                    //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
                    for(int option = 0; option < 2 ; option ++){
                        node now = copy_node(current);
                        now.domain[i][j] = 1;
                        now.value[i][j] = option;
                        if(option == 1)
                            now.TNT++;
                        if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
                        {
                            // 沒有被 explored 過的才能加進去！
                            if(isExplored(explored,front,now))
                            {
                                if(now.TNT == num){
                                    if(check_solution(now)){
                                        print_solution(now);
                                        cout<<"node Expand:"<<step<<endl;
                                        return;
                                    }
                                    else
                                        continue;
                                }
                                front.push(now);
                                step++;
                            }
                        }
                    }
                }
            }
        }
    }
    cout<<"No Solution!"<<endl;
}

void find_forward_solution(int r, int c, int num)
{
    stack<node> front;
    stack<node> explored;
    front.push(first);
    int step = 1;
    while(!front.empty())
    {
        node current = front.top();
        explored.push(current);
        front.pop();

        // --- forward checking

        bool flag = true;
        for(int i = 0 ; i < hint.size() ; i ++){
            int lowerbound = 0;
            int upperbound = 0;
            point no = hint[i];
            // direction
            //cout<<"forward x,y :"<< no.x<<","<<no.y<<endl;
            for(int j = 0 ; j < 8 ; j ++){
                int x = no.x + dir[j].x;
                int y = no.y + dir[j].y;
                if(!isVaild(x,y))
                    continue;
                if ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
                    lowerbound ++;
                else if(current.domain[x][y] == 2) // unassigned
```

```
                    upperbound ++;
            }
            upperbound += lowerbound;
            // lower bound & upper bound
            if( lowerbound > mine[no.x][no.y] )
            {
                flag = false;
                //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
                break;
            }
            else if (lowerbound == mine [no.x][no.y])
            {
                // 代表其他 unassinged 的值 都要成為 0
                for(int j = 0 ; j < 8 ; j ++){
                    int x = no.x + dir[j].x;
                    int y = no.y + dir[j].y;
                    if(!isVaild(x,y))
                        continue;
                    if(current.domain[x][y] == 2){
                        current.domain[x][y] = 1 ;
                        current.value[x][y] = 0 ;
                    }
                }
            }
            if( upperbound < mine[no.x][no.y])
            {
                flag = false;
                //cout<<"upperbound < mine[no.x][no.y]"<<endl;
                break;
            }
            else if( upperbound == mine[no.x][no.y])
            {
                // 代表其他 unassigned 的值 都要變成 1
                for(int j = 0 ; j < 8 ; j ++){
                    int x = no.x + dir[j].x;
                    int y = no.y + dir[j].y;
                    if(!isVaild(x,y))
                        continue;
                    if(current.domain[x][y] == 2){
                        current.domain[x][y] = 1 ;
                        current.value[x][y] = 1 ;
                        current.TNT++;
                    }
                }
            }
            //cout<<"TNT:"<<current.TNT<<endl;
            //print_value(current);
        }
        // 代表這個 state 不滿足 state 繼續 pop
        if(!flag) continue;

        // 檢查是不是答案
        if( current.TNT == num )
        {
            if(check_solution(current)){
                print_solution(current);
                cout<<"node Expand:"<<step<<endl;
                return;
            }
            continue;
        }
        // to assign value！
        for(int i = 0 ; i < r ; i ++){
            for(int j = 0 ; j < c ; j++){
                if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
                {
                    //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
                    // 去給這 value 是 1(mine) 還是 0
                    for(int option = 1; option >= 0  ; option --){
                        node now = copy_node(current);
                        now.domain[i][j] = 1;
                        now.value[i][j] = option;
                        if(option == 1)
                            now.TNT++;
                        if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
                        {
                            // 沒有被 explored 過的才能加進去！
                            if(isExplored(explored,front,now))
                            {
                                // 如果 TNT 數量 == 炸彈的數量 檢查 solution
                                if( now.TNT == num ){
                                    if(check_solution(now)){
                                        print_solution(now);
                                        cout<<"node Expand:"<<step<<endl;
                                        return;
                                    }
                                    else
                                        //cout<<"**************************************"<<endl;

                                    continue;
                                }
                                front.push(now);
                                step++;
                            }
                        }
                    }
                }
            }
        }
    }

}

// using for priority queue
bool operator < (const point &p1,const point &p2){ return p1.degree < p2.degree;}
bool operator > (const point &p1,const point &p2){ return p1.degree > p2.degree;}

// forward function with Degree Heuristic
void find_forward_solution_Degree(int r, int c, int num)
{
    stack<node> front;
    stack<node> explored;
    front.push(first);
    int step = 1;
    while(!front.empty())
    {
        node current = front.top();
        explored.push(current);
        front.pop();

        // --- priorty_queue using for degree heuristic
        priority_queue<point,vector<point>,greater<point> > de;
        for(int i = 0 ; i < r ; i ++){
            for(int j = 0 ; j < c ; j ++){
                if(mine[i][j] != -1) continue;
                point p = { i , j , 0};
                for(int k = 0 ; k < 8 ; k ++)
                    if(isVaild( i + dir[k].x , j + dir[k].y ))
                        if(mine[i + dir[k].x][j + dir[k].y] != -1)
                            p.degree ++;
                de.push(p);
            }
        }
        // to assign value !
        while(!de.empty()){
            point n = de.top();
            de.pop();
            int i = n.x, j = n.y;

            if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
            {
                // 去給這 value 是 1(mine) 還是 0
                for(int option = 1; option >= 0  ; option --){
                    node now = copy_node(current);
                    now.domain[i][j] = 1;
                    now.value[i][j] = option;
                    if(option == 1)
                        now.TNT++;
                    // // 沒有被 explored 過的才能加進去！ 且 if 這個點有符合 17 constraints
                    if(checkVaild(now,num,hint)&&isExplored(explored,front,now))
                    {
                        // 如果 TNT 數量 == 炸彈的數量 檢查 solution
                        if( now.TNT == num ){
                            if(check_solution(now)){
                                print_solution(now);
                                cout<<"node Expand:"<<step<<endl;
                                return;
                            }
                            else
                                continue;
                        }
                        front.push(now);
                        step++;
                    }
                }
            }
        }
    }

}


bool operator < (const node &p1,const node &p2){ return p1.TNT < p2.TNT;}
bool operator > (const node &p1,const node &p2){ return p1.TNT > p2.TNT;}

// forward function with TNT Heuristic
void find_forward_solution_TNT  (int r, int c, int num)
{
    priority_queue<node> front;
    stack<node> explored;
    front.push(first);
    int step =1;
    while(!front.empty())
    {
        node current = front.top();
        explored.push(current);
        front.pop();
        if(current.TNT == num)
            for(int i = 0 ; i < r; i ++)
                for(int j = 0 ; j <c ; j ++)
                    if(current.domain[i][j] == 2){
                        current.domain[i][j] = 1;
                        current.value[i][j] = 0;
                    }

        // to assign value !
        for(int i = 0 ; i < r ; i ++){
            for(int j = 0 ; j < c ; j++){
                if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
                {
                    //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
                    // 去給這 value 是 1(mine) 還是 0
                    for(int option = 1; option >= 0  ; option --){
                        node now = copy_node(current);
                        now.domain[i][j] = 1;
                        now.value[i][j] = option;
                        if(option == 1)
                            now.TNT++;
                        if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
                        {
                            // 沒有被 explored 過的才能加進去！
                            if(isExplored(explored,front,now))
                            {
                                // 如果 TNT 數量 == 炸彈的數量 檢查 solution
                                if( now.TNT == num ){
                                    if(check_solution(now)){
                                        print_solution(now);
                                        cout<<"node Expand:"<<step<<endl;
                                        return;
                                    }
                                    else
                                        continue;
                                }
                                front.push(now);
                                step++;
                            }
                        }
                    }
                }
            }
        }
    }

}

// forward function with MRV Heuristic
void find_forward_solution_MRV(int r, int c, int num)
{
    stack<node> front;
    stack<node> explored;
    front.push(first);
    int step = 1;
    while(!front.empty())
    {
        node current = front.top();
        explored.push(current);
        front.pop();

        // --- priorty_queue using for degree heuristic
        priority_queue<point,vector<point>,greater<point> > de;
        for(int i = 0 ; i < r ; i ++){
            for(int j = 0 ; j < c ; j ++){
                point p = { i , j , current.domain[i][j] };
                de.push(p);
            }
        }


        // to assign value！
        while(!de.empty()){
            point n = de.top();
            de.pop();
            int i = n.x, j = n.y;

            if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
            {
                // 去給這 value 是 1(mine) 還是 0
                for(int option = 1; option >= 0  ; option --){
                    node now = copy_node(current);
                    now.domain[i][j] = 1;
                    now.value[i][j] = option;
                    if(option == 1)
                        now.TNT++;
                    // // 沒有被 explored 過的才能加進去！ 且 if 這個點有符合 17 constraints
                    if(checkVaild(now,num,hint)&&isExplored(explored,front,now))
                    {
                        // 如果 TNT 數量 == 炸彈的數量 檢查 solution
                        if( now.TNT == num ){
                            if(check_solution(now)){
                                print_solution(now);
                                cout<<"node Expand:"<<step<<endl;
                                return;
                            }
                            else
                                continue;
                        }
                        front.push(now);
                        step ++;
                    }
                }
            }
        }
    }
}


int main()
{
    int num_mines;
    time_t start,end;
    double t;

    cin >> row >> column >> num_mines;

    for( int i = 0 ; i < row ; i ++ )
```

```
        for ( int j = 0 ; j < column ; j ++ )
            cin>>mine[i][j];

    initial();

    /*
    //---- original without forward & heuristic
    start = clock();
    find_solution(row,column,num_mines);
    end = clock();
    t = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time : %fs\n",t);   */

    //----  forward checking without heuristic
    cout<<"forward checking without heuristic\n";
    start = clock();
    find_forward_solution(row,column,num_mines);
    end = clock();
    t = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time : %fs\n",t);


    //----   with Degree heuristic
    cout<<"with Degree heuristic\n";
    start = clock();
    find_forward_solution_Degree(row,column,num_mines);
    end = clock();
    t = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time : %fs\n",t);



    //----  with uesr-defined heuristic
    cout<<"with TNT heuristic\n";
    start = clock();
    find_forward_solution_TNT(row,column,num_mines);
    end = clock();
    t = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time : %fs\n",t);

    //----   with MRV heuristic
    cout<<"with MRV heuristic\n";
    start = clock();
    find_forward_solution_MRV(row,column,num_mines);
    end = clock();
    t = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time : %fs\n",t);

    return 0;
}
```