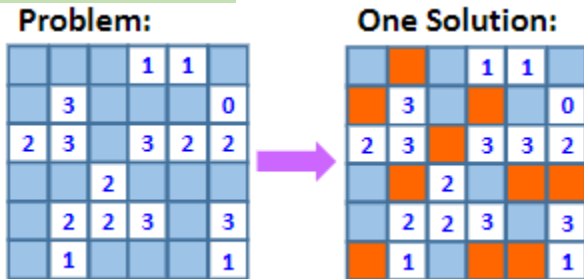


人工智慧概論

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：

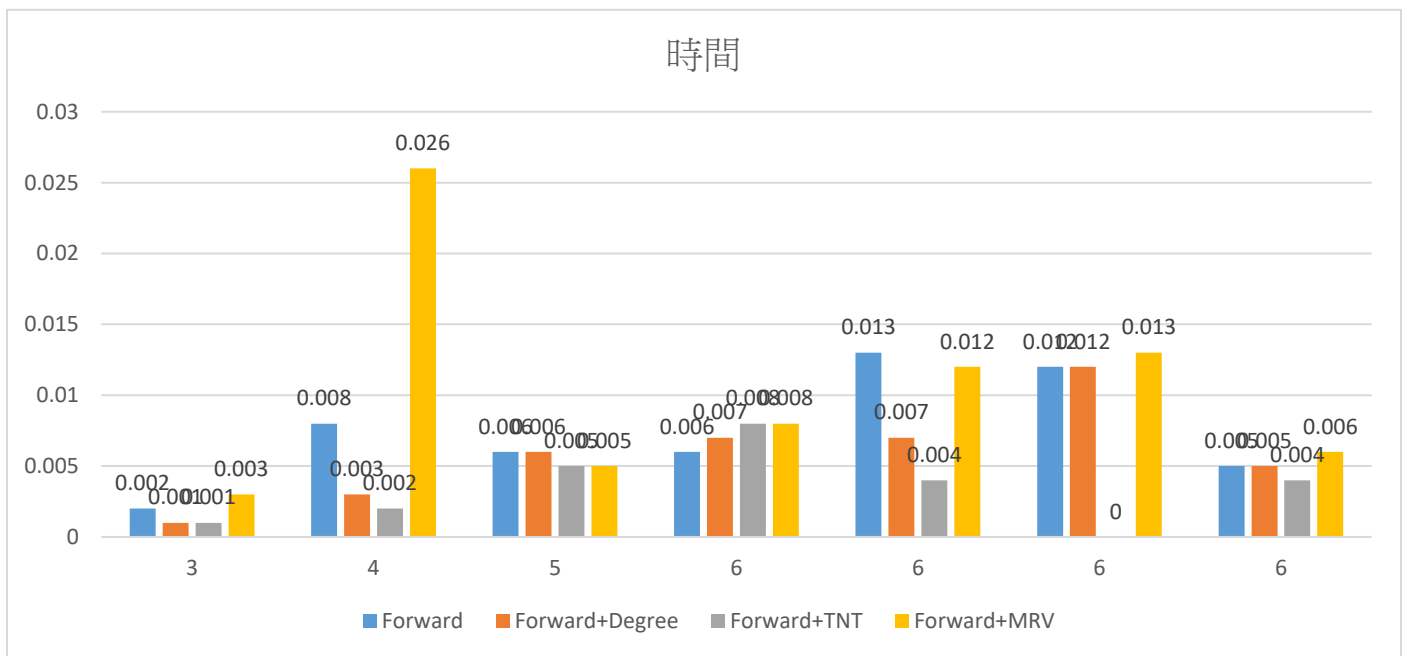
Input :

[illegible]

時間分析(Time)：

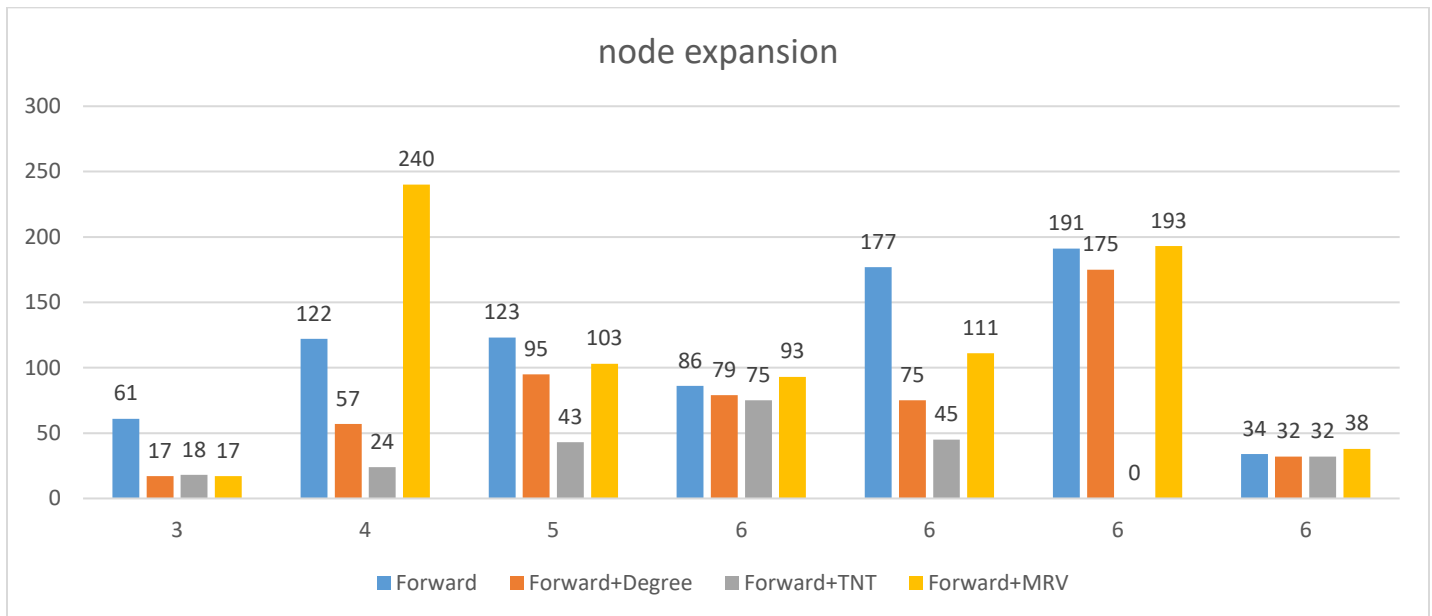
紅色為最慢的，黃色為表現最佳的。

	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，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

```
1. #include<iostream>
2. #include<stdio.h>
3. #include<stdlib.h>
4. #include<algorithm>
5. #include<vector>
6. #include<stack>
7. #include<queue>
8. #include<time.h>
9. #include<iomanip>
10. #define N 6
11. using namespace std;
12.
13. /*
14.     author : 蔡怡君
15.     content : Minesweeper using Backtrack Search
16. */
17.
18. // mine map
19. int mine[N][N],row,column;
20.
21.
22. struct node
23. {
24.     int assigned_node;
25.     int TNT;
26.     int domain[N][N]; // domain's number {0,1} = 2
27.     int value[N][N]; // value : 0 or 1 (have mines)
28. }first;
29.
30.
31. struct point
32. {
33.     int x,y;
34.     int degree;
35. }dir[8];
36.
37. vector<point>hint;
38.
39. //point **goal;
40.
41. void initial()
42. {
43.     dir[0]={-1,-1};
44.     dir[1]={-1,0};
45.     dir[2]={-1,1};
46.     dir[3]={0,-1};
47.     dir[4]={0,1};
48.     dir[5]={1,-1};
49.     dir[6]={1,0};
50.     dir[7]={1,1};
51.
52.     first.assigned_node = 0;
53.     first.TNT = 0;
54.     for(int i = 0 ; i < row ; i++){
55.         for(int j = 0 ; j < column ; j++){
56.             if(mine[i][j] == -1){
57.                 first.domain[i][j] = 2;
```

```

58.         first.value[i][j] = -1;
59.     }
60.
61.     else
62.     { // means hint
63.         point h = {i,j};
64.         hint.push_back(h);
65.         first.domain[i][j] = 1;
66.         first.value[i][j] = mine[i][j];
67.     }
68. }
69. }
70. }
71.
72. bool operator == (const node &p1,const node &p2)
73. {
74.     if(p1.TNT != p2.TNT )
75.         return false;
76.     for(int i = 0 ; i < row ; i ++){
77.         for(int j = 0 ; j < column ; j ++){
78.             if(p1.domain[i][j] != p2.domain[i][j])
79.                 return false;
80.             if(p1.value[i][j] != p2.value[i][j])
81.                 return false;
82.         }
83.     }
84.     return true;
85. }
86.
87. bool isVaild(int i,int j)
88. {
89.     if(mine[i][j] != -1) return false;
90.     return ( (i >= 0 && i < row ) && (j >= 0 && j < column) );
91. }
92.
93. node copy_node(node A)
94. {
95.     node tmp;
96.     for(int i = 0 ; i < row ;i ++){
97.         for(int j = 0 ; j < column ; j ++){
98.             tmp.domain[i][j] = A.domain[i][j];
99.             tmp.value[i][j] = A.value[i][j];
100.        }
101.    }
102.    tmp.assigned_node = A.assigned_node;
103.    tmp.TNT = A.TNT;
104.    return tmp;
105.}
106.
107./*
108.// local check
109.bool checkVaild(node now,int num_TNT)
110.{
111.    // global check
112.    if(now.TNT > num_TNT) return false;
113.    for(int i = 0; i < row ; i ++){
114.        for(int j = 0 ; j < column ; j ++){
115.            if(mine[i][j] != -1)
116.            { // local check 8 directions 周圍的 TNT
117.                int TNT=0;
118.                for(int k = 0; k < 8 ; k ++){

```

```

119.         if(isVaild(i + dir[k].x,j +dir[k].y))
120.             if(now.value[i + dir[k].x][j +dir[k].y ] == 1)
121.                 TNT ++;
122.         }
123.         if(TNT > mine[i][j])
124.             return false;
125.     }
126. }
127. }
128. return true;
129.}
130.*/
131.
132.bool checkVaild(node now, int num_TNT, vector<point> hhint)
133.{
134.    bool all_assigned = true;
135.    if(now.TNT > num_TNT) return false;
136.    for(int i = 0 ; i < hhint.size() ; i ++ )
137.    {
138.        // check hint 的八維
139.        int TNT = 0;
140.        for(int j = 0 ; j < 8 ; j ++){
141.            int px = hhint[i].x + dir[j].x;
142.            int py = hhint[i].y + dir[j].y;
143.            if(isVaild(px , py))
144.            {
145.                if(now.value[px][py] == 1) // 放炸彈的地方
146.                    TNT ++ ;
147.                if(now.domain[px][py] != 1) // 如果八個方位都 assigned 完
148.                    all_assigned = false;
149.            }
150.        }
151.        if(TNT > mine[hhint[i].x][hhint[i].y])
152.            return false;
153.        // 八個方位都 assigned 了 卻不符合 TNT -> false
154.        if(all_assigned == true && TNT != mine[hhint[i].x][hhint[i].y] )
155.            return false;
156.    }
157.    return true;
158.}
159.}
160.
161.void print_solution(node result)
162.{
163.    for(int i = 0 ; i < row ;i ++){
164.        for(int j = 0 ; j < column ; j ++){
165.            if(mine[i][j] != -1)
166.                cout<<mine[i][j]<<" ";
167.            else if(result.value[i][j] == 1)
168.                cout<<"*"<<" ";
169.            else
170.                cout<<" "<<" ";
171.        }
172.        cout<<endl;
173.    }
174.}
175.
176.// using for checking
177.void print_do(node result)
178.{
179.    cout<<"domain: \n";

```



```

180.     for(int i = 0 ; i < row ; i ++){
181.         for(int j = 0 ; j < column ; j ++){
182.             cout<<result.domain[i][j]<<" ";
183.         }
184.         cout<<endl;
185.     }
186. }
187.
188. // using for checking
189. void print_value(node result)
190. {
191.     cout<<"value: \n";
192.     for(int i = 0 ; i < row ; i ++){
193.         for(int j = 0 ; j < column ; j ++){
194.             if(mine[i][j] == -1 && result.value[i][j] == 1)
195.                 cout<<"*"<<" ";
196.             else
197.                 cout<<setw(2)<<result.value[i][j]<<" ";
198.         }
199.         cout<<endl;
200.     }
201. }
202.
203.
204. bool check_solution(node result)
205. {
206.     for(int i = 0; i < row ; i ++){
207.         for(int j = 0 ; j < column ; j ++){
208.             if(mine[i][j] != -1)
209.             { // local check 8 directions 周圍的TNT
210.                 int TNT=0;
211.                 for(int k = 0; k < 8 ; k ++){
212.                     if(isVaild(i + dir[k].x,j +dir[k].y))
213.                         if(result.value[i + dir[k].x][j +dir[k].y ] == 1)
214.                             TNT ++;
215.                 }
216.                 if(TNT != mine[i][j])
217.                     return false;
218.             }
219.         }
220.     }
221.     return true;
222. }
223.
224. bool isExplored(stack<node> explored , stack<node>front, node now)
225. {
226.     // 在 Explored set 裡面
227.     while(!explored.empty())
228.     {
229.         node current=explored.top();
230.         explored.pop();
231.         // 如果 current == now 一模一樣
232.         if(current == now)
233.         {
234.             //cout<<"the same"<<endl;
235.             return false;
236.         }
237.     }
238.     // 在 Frontier 裏頭
239.     while(!front.empty())
240.     {

```

```

241.     node current = front.top();
242.     front.pop();
243.     // 如果 current == now 一模一樣
244.     if(current == now)
245.     {
246.         //cout<<"the same"<<endl;
247.         return false;
248.     }
249. }
250. return true;
251.}
252.
253.bool isExplored(stack<node> explored , priority_queue<node>front, node now)
254.{
255.    // 在 Explored set 裡面
256.    while(!explored.empty())
257.    {
258.        node current=explored.top();
259.        explored.pop();
260.        // 如果 current == now 一模一樣
261.        if(current == now)
262.        {
263.            //cout<<"the same"<<endl;
264.            return false;
265.        }
266.    }
267.    // 在 Frontier 裏頭
268.    while(!front.empty())
269.    {
270.        node current = front.top();
271.        front.pop();
272.        // 如果 current == now 一模一樣
273.        if(current == now)
274.        {
275.            //cout<<"the same"<<endl;
276.            return false;
277.        }
278.    }
279.    return true;
280.}
281.
282.// original without forward checking / heuristic
283.void find_solution(int r , int c , int num)
284.{
285.    stack<node> front;
286.    stack<node> explored;
287.    int step = 1;
288.    front.push(first);
289.    //bool flag = true;
290.    while(!front.empty())
291.    {
292.        node current = front.top();
293.        explored.push(current);
294.        front.pop();
295.        //cout<<"layer9 : "<<step<<endl;
296.        for(int i = 0 ; i < r ; i++){
297.            for(int j = 0 ; j < c ; j++){
298.                if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned variable
299.                {
300.                    //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
301.                    for(int option = 0; option < 2 ; option++){

```

```

302.         node now = copy_node(current);
303.         now.domain[i][j] = 1;
304.         now.value[i][j] = option;
305.         if(option == 1)
306.             now.TNT++;
307.         if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
308.         {
309.             // 沒有被 explored 過的才能加進去 !
310.             if(isExplored(explored,front,now))
311.             {
312.                 if(now.TNT == num){
313.                     if(check_solution(now)){
314.                         print_solution(now);
315.                         cout<<"node Expand:"<<step<<endl;
316.                         return;
317.                     }
318.                     else
319.                         continue;
320.                 }
321.                 front.push(now);
322.                 step++;
323.             }
324.         }
325.     }
326. }
327. }
328. }
329. }
330. cout<<"No Solution!"<<endl;
331. }
332.
333. void find_forward_solution(int r, int c, int num)
334. {
335.     stack<node> front;
336.     stack<node> explored;
337.     front.push(first);
338.     int step = 1;
339.     while(!front.empty())
340.     {
341.         node current = front.top();
342.         explored.push(current);
343.         front.pop();
344.
345.         // --- forward checking
346.
347.         bool flag = true;
348.         for(int i = 0 ; i < hint.size() ; i++){
349.             int lowerbound = 0;
350.             int upperbound = 0;
351.             point no = hint[i];
352.             // direction
353.             //cout<<"forward x,y : "<< no.x<<" , "<<no.y<<endl;
354.             for(int j = 0 ; j < 8 ; j++){
355.                 int x = no.x + dir[j].x;
356.                 int y = no.y + dir[j].y;
357.                 if(!isVaild(x,y))
358.                     continue;
359.                 if ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
360.                     lowerbound ++;
361.                 else if(current.domain[x][y] == 2) // unassigned
362.                     upperbound ++;

```

```

363.     }
364.     upperbound += lowerbound;
365.     // lower bound & upper bound
366.     if( lowerbound > mine[no.x][no.y] )
367.     {
368.         flag = false;
369.         //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
370.         break;
371.     }
372.     else if (lowerbound == mine [no.x][no.y])
373.     {
374.         // 代表其他 unassigned 的值 都要成為 0
375.         for(int j = 0 ; j < 8 ; j ++){
376.             int x = no.x + dir[j].x;
377.             int y = no.y + dir[j].y;
378.             if(!isVaild(x,y))
379.                 continue;
380.             if(current.domain[x][y] == 2){
381.                 current.domain[x][y] = 1 ;
382.                 current.value[x][y] = 0 ;
383.             }
384.         }
385.     }
386.     if( upperbound < mine[no.x][no.y])
387.     {
388.         flag = false;
389.         //cout<<"upperbound < mine[no.x][no.y]"<<endl;
390.         break;
391.     }
392.     else if( upperbound == mine[no.x][no.y])
393.     {
394.         // 代表其他 unassigned 的值 都要變成 1
395.         for(int j = 0 ; j < 8 ; j ++){
396.             int x = no.x + dir[j].x;
397.             int y = no.y + dir[j].y;
398.             if(!isVaild(x,y))
399.                 continue;
400.             if(current.domain[x][y] == 2){
401.                 current.domain[x][y] = 1 ;
402.                 current.value[x][y] = 1 ;
403.                 current.TNT++;
404.             }
405.         }
406.     }
407. }
408. // 代表這個 state 不滿足 state 繼續 pop
409. if(!flag) continue;
410.
411. // 檢查是不是答案
412. if( current.TNT == num )
413. {
414.     if(check_solution(current)){
415.         print_solution(current);
416.         cout<<"node Expand:"<<step<<endl;
417.         return;
418.     }
419.     continue;
420. }
421. // to assign value !
422. for(int i = 0 ; i < r ; i ++){
423.     for(int j = 0 ; j < c ; j++){

```

```

424.         if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned variable
425.         {
426.             //cout<<"fill in (x,y)"<<j<<" "<<i<<endl;
427.             // 去給這 value 是 1(mine) 還是 0
428.             for(int option = 1; option >= 0 ; option --){
429.                 node now = copy_node(current);
430.                 now.domain[i][j] = 1;
431.                 now.value[i][j] = option;
432.                 if(option == 1)
433.                     now.TNT++;
434.                 if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
435.                 {
436.                     // 沒有被 explored 過的才能加進去！
437.                     if(isExplored(explored,front,now))
438.                     {
439.                         // 如果 TNT 數量 == 炸彈的數量 檢查 solution
440.                         if( now.TNT == num ){
441.                             if(check_solution(now)){
442.                                 print_solution(now);
443.                                 cout<<"node Expand:"<<step<<endl;
444.                                 return;
445.                             }
446.                             else
447.                                 continue;
448.                         }
449.                         front.push(now);
450.                         step++;
451.                     }
452.                 }
453.             }
454.         }
455.     }
456. }
457. }
458.
459. }
460.
461. // using for priority queue
462. bool operator < (const point &p1,const point &p2){ return p1.degree < p2.degree;}
463. bool operator > (const point &p1,const point &p2){ return p1.degree > p2.degree;}
464.
465. // forward function with Degree Heuristic
466. void find_forward_solution_Degree(int r, int c, int num)
467. {
468.     stack<node> front;
469.     stack<node> explored;
470.     front.push(first);
471.     int step = 1;
472.     while(!front.empty())
473.     {
474.         node current = front.top();
475.         explored.push(current);
476.         front.pop();
477.
478.         // --- priority_queue using for degree heuristic
479.         priority_queue<point,vector<point>,greater<point> > de;
480.         for(int i = 0 ; i < r ; i ++){
481.             for(int j = 0 ; j < c ; j ++){
482.                 if(mine[i][j] != -1) continue;
483.                 point p = { i , j , 0};
484.                 for(int k = 0 ; k < 8 ; k ++){

```

```

485.         if(isVaild( i + dir[k].x , j + dir[k].y ))
486.             if(mine[i + dir[k].x][j + dir[k].y] != -1)
487.                 p.degree ++;
488.         de.push(p);
489.     }
490. }
491.
492. // --- forward checking
493. bool flag = true;
494. for(int i = 0 ; i < hint.size() ; i ++){
495.     int lowerbound = 0;
496.     int upperbound = 0;
497.     point no = hint[i];
498.     // direction
499.     //cout<<"forward x,y : "<< no.x<<" "<<no.y<<endl;
500.     for(int j = 0 ; j < 8 ; j ++){
501.         int x = no.x + dir[j].x;
502.         int y = no.y + dir[j].y;
503.         if(!isVaild(x,y))
504.             continue;
505.         if ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
506.             lowerbound ++;
507.         else if(current.domain[x][y] == 2) // unassigned
508.             upperbound ++;
509.     }
510.     upperbound += lowerbound;
511.     // lower bound & upper bound
512.     if( lowerbound > mine[no.x][no.y] )
513.     {
514.         flag = false;
515.         //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
516.         break;
517.     }
518.     else if (lowerbound == mine [no.x][no.y])
519.     {
520.         // 代表其他 unassigned 的值 都要成為 0
521.         for(int j = 0 ; j < 8 ; j ++){
522.             int x = no.x + dir[j].x;
523.             int y = no.y + dir[j].y;
524.             if(!isVaild(x,y))
525.                 continue;
526.             if(current.domain[x][y] == 2){
527.                 current.domain[x][y] = 1 ;
528.                 current.value[x][y] = 0 ;
529.             }
530.         }
531.     }
532.     if( upperbound < mine[no.x][no.y])
533.     {
534.         flag = false;
535.         //cout<<"upperbound < mine[no.x][no.y]"<<endl;
536.         break;
537.     }
538.     else if( upperbound == mine[no.x][no.y])
539.     {
540.         // 代表其他 unassigned 的值 都要變成 1
541.         for(int j = 0 ; j < 8 ; j ++){
542.             int x = no.x + dir[j].x;
543.             int y = no.y + dir[j].y;
544.             if(!isVaild(x,y))
545.                 continue;

```

```

546.         if(current.domain[x][y] == 2){
547.             current.domain[x][y] = 1 ;
548.             current.value[x][y] = 1 ;
549.             current.TNT++;
550.         }
551.     }
552. }
553. }
554. // 代表這個 state 不滿足 state 繼續 pop
555. if(!flag) continue;
556.
557. // 檢查是不是答案
558. if( current.TNT == num )
559. {
560.     if(check_solution(current)){
561.         print_solution(current);
562.         cout<<"node Expand:"<<step<<endl;
563.         return;
564.     }
565.     continue;
566. }
567.
568. // to assign value !
569. while(!de.empty()){
570.     point n = de.top();
571.     de.pop();
572.     int i = n.x, j = n.y;
573.
574.     if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned variable
575.     {
576.         // 去給這 value 是 1(mine) 還是 0
577.         for(int option = 1; option >= 0 ; option --){
578.             node now = copy_node(current);
579.             now.domain[i][j] = 1;
580.             now.value[i][j] = option;
581.             if(option == 1)
582.                 now.TNT++;
583.             // // 沒有被 explored 過的才能加進去！ 且 if 這個點有符合 17 constraints
584.             if(checkVaild(now,num,hint)&&isExplored(explored,front,now))
585.             {
586.                 // 如果 TNT 數量 == 炸彈的數量 檢查 solution
587.                 if( now.TNT == num ){
588.                     if(check_solution(now)){
589.                         print_solution(now);
590.                         cout<<"node Expand:"<<step<<endl;
591.                         return;
592.                     }
593.                     else
594.                         continue;
595.                 }
596.                 front.push(now);
597.                 step++;
598.             }
599.         }
600.     }
601. }
602. }
603.
604. }
605.
606.

```

```

607. bool operator < (const node &p1, const node &p2){ return p1.TNT < p2.TNT;}
608. bool operator > (const node &p1, const node &p2){ return p1.TNT > p2.TNT;}
609.
610. // forward function with TNT Heuristic
611. void find_forward_solution_TNT (int r, int c, int num)
612. {
613.     priority_queue<node> front;
614.     stack<node> explored;
615.     front.push(first);
616.     int step =1;
617.     while(!front.empty())
618.     {
619.         node current = front.top();
620.         explored.push(current);
621.         front.pop();
622.         if(current.TNT == num)
623.             for(int i = 0 ; i < r; i ++){
624.                 for(int j = 0 ; j < c ; j ++){
625.                     if(current.domain[i][j] == 2){
626.                         current.domain[i][j] = 1;
627.                         current.value[i][j] = 0;
628.                     }
629.
630.                     // --- forward checking
631.                     bool flag = true;
632.                     for(int i = 0 ; i < hint.size() ; i ++){
633.                         int lowerbound = 0;
634.                         int upperbound = 0;
635.                         point no = hint[i];
636.                         // direction
637.                         //cout<<"forward x,y : "<< no.x<<" "<<no.y<<endl;
638.                         for(int j = 0 ; j < 8 ; j ++){
639.                             int x = no.x + dir[j].x;
640.                             int y = no.y + dir[j].y;
641.                             if(!isVaild(x,y))
642.                                 continue;
643.                             if ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
644.                                 lowerbound ++;
645.                             else if(current.domain[x][y] == 2) // unassigned
646.                                 upperbound ++;
647.                         }
648.                         upperbound += lowerbound;
649.                         // lower bound & upper bound
650.                         if( lowerbound > mine[no.x][no.y] )
651.                         {
652.                             flag = false;
653.                             //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
654.                             break;
655.                         }
656.                         else if (lowerbound == mine [no.x][no.y])
657.                         {
658.                             // 代表其他 unassigned 的值 都要成為 0
659.                             for(int j = 0 ; j < 8 ; j ++){
660.                                 int x = no.x + dir[j].x;
661.                                 int y = no.y + dir[j].y;
662.                                 if(!isVaild(x,y))
663.                                     continue;
664.                                 if(current.domain[x][y] == 2){
665.                                     current.domain[x][y] = 1 ;
666.                                     current.value[x][y] = 0 ;
667.                                 }

```



```

668.     }
669.     }
670.     if( upperbound < mine[no.x][no.y])
671.     {
672.         flag = false;
673.         //cout<<"upperbound < mine[no.x][no.y]"<<endl;
674.         break;
675.     }
676.     else if( upperbound == mine[no.x][no.y])
677.     {
678.         // 代表其他 unassigned 的值 都要變成 1
679.         for(int j = 0 ; j < 8 ; j ++){
680.             int x = no.x + dir[j].x;
681.             int y = no.y + dir[j].y;
682.             if(!isVaild(x,y))
683.                 continue;
684.             if(current.domain[x][y] == 2){
685.                 current.domain[x][y] = 1 ;
686.                 current.value[x][y] = 1 ;
687.                 current.TNT++;
688.             }
689.         }
690.     }
691. }
692. // 代表這個 state 不滿足 state 繼續 pop
693. if(!flag) continue;
694.
695. // 檢查是不是答案
696. if( current.TNT == num )
697. {
698.     if(check_solution(current)){
699.         print_solution(current);
700.         cout<<"node Expand:"<<step<<endl;
701.         return;
702.     }
703.     continue;
704. }
705.
706. // to assign value !
707. for(int i = 0 ; i < r ; i ++){
708.     for(int j = 0 ; j < c ; j++){
709.         if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned variable
710.         {
711.             //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
712.             // 去給這 value 是 1(mine) 還是 0
713.             for(int option = 1; option >= 0 ; option --){
714.                 node now = copy_node(current);
715.                 now.domain[i][j] = 1;
716.                 now.value[i][j] = option;
717.                 if(option == 1)
718.                     now.TNT++;
719.                 if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
720.                 {
721.                     // 沒有被 explored 過的才能加進去 !
722.                     if(isExplored(explored,front,now))
723.                     {
724.                         // 如果 TNT 數量 == 炸彈的數量 檢查 solution
725.                         if( now.TNT == num ){
726.                             if(check_solution(now)){
727.                                 print_solution(now);
728.                                 cout<<"node Expand:"<<step<<endl;

```

```

729.                                     return;
730.                                     }
731.                                     else
732.                                         continue;
733.                                     }
734.                                     front.push(now);
735.                                     step++;
736.                                     }
737.                                 }
738.                            }
739.                    }
740.            }
741.    }
742. }
743.
744. }
745.
746. // forward function with MRV Heuristic
747. void find_forward_solution_MRV(int r, int c, int num)
748. {
749.     stack<node> front;
750.     stack<node> explored;
751.     front.push(first);
752.     int step = 1;
753.     while(!front.empty())
754.     {
755.         node current = front.top();
756.         explored.push(current);
757.         front.pop();
758.
759.         // --- priority_queue using for degree heuristic
760.         priority_queue<point, vector<point>, greater<point> > de;
761.         for(int i = 0 ; i < r ; i ++){
762.             for(int j = 0 ; j < c ; j ++){
763.                 point p = { i , j , current.domain[i][j] };
764.                 de.push(p);
765.             }
766.         }
767.
768.         // --- forward checking
769.         bool flag = true;
770.         for(int i = 0 ; i < hint.size() ; i ++){
771.             int lowerbound = 0;
772.             int upperbound = 0;
773.             point no = hint[i];
774.             // direction
775.             //cout<<"forward x,y : "<< no.x<<" "<<no.y<<endl;
776.             for(int j = 0 ; j < 8 ; j ++){
777.                 int x = no.x + dir[j].x;
778.                 int y = no.y + dir[j].y;
779.                 if(!isVaild(x,y))
780.                     continue;
781.                 if ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
782.                     lowerbound ++;
783.                 else if(current.domain[x][y] == 2) // unassigned
784.                     upperbound ++;
785.             }
786.             upperbound += lowerbound;
787.             // lower bound & upper bound
788.             if( lowerbound > mine[no.x][no.y] )
789.             {
790.                 flag = false;

```

```

791.         //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
792.         break;
793.     }
794.     else if (lowerbound == mine [no.x][no.y])
795.     {
796.         // 代表其他 unassigned 的值 都要成為 0
797.         for(int j = 0 ; j < 8 ; j ++){
798.             int x = no.x + dir[j].x;
799.             int y = no.y + dir[j].y;
800.             if(!isVaild(x,y))
801.                 continue;
802.             if(current.domain[x][y] == 2){
803.                 current.domain[x][y] = 1 ;
804.                 current.value[x][y] = 0 ;
805.             }
806.         }
807.     }
808.     if( upperbound < mine[no.x][no.y])
809.     {
810.         flag = false;
811.         //cout<<"upperbound < mine[no.x][no.y]"<<endl;
812.         break;
813.     }
814.     else if( upperbound == mine[no.x][no.y])
815.     {
816.         // 代表其他 unassigned 的值 都要變成 1
817.         for(int j = 0 ; j < 8 ; j ++){
818.             int x = no.x + dir[j].x;
819.             int y = no.y + dir[j].y;
820.             if(!isVaild(x,y))
821.                 continue;
822.             if(current.domain[x][y] == 2){
823.                 current.domain[x][y] = 1 ;
824.                 current.value[x][y] = 1 ;
825.                 current.TNT++;
826.             }
827.         }
828.     }
829. }
830. // 代表這個 state 不滿足 state 繼續 pop
831. if(!flag) continue;
832.
833. // 檢查是不是答案
834. if( current.TNT == num )
835. {
836.     if(check_solution(current)){
837.         print_solution(current);
838.         cout<<"node Expand:"<<step<<endl;
839.         return;
840.     }
841.     continue;
842. }
843.
844. // to assign value !
845. while(!de.empty()){
846.     point n = de.top();
847.     de.pop();
848.     int i = n.x, j = n.y;
849.
850.     if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned variable
851.     {

```

```

852.         // 去給這 value 是 1(mine) 還是 0
853.         for(int option = 1; option >= 0 ; option --){
854.             node now = copy_node(current);
855.             now.domain[i][j] = 1;
856.             now.value[i][j] = option;
857.             if(option == 1)
858.                 now.TNT++;
859.             // // 沒有被 explored 過的才能加進去！ 且 if 這個點有符合 17 constraints
860.             if(checkVaild(now,num,hint)&&isExplored(explored,front,now))
861.             {
862.                 // 如果 TNT 數量 == 炸彈的數量 檢查 solution
863.                 if( now.TNT == num ){
864.                     if(check_solution(now)){
865.                         print_solution(now);
866.                         cout<<"node Expand:"<<step<<endl;
867.                         return;
868.                     }
869.                     else
870.                         continue;
871.                 }
872.                 front.push(now);
873.                 step ++;
874.             }
875.         }
876.     }
877. }
878. }
879. }
880.
881.
882. int main()
883. {
884.     int num_mines;
885.     time_t start,end;
886.     double t;
887.
888.     cin >> row >> column >> num_mines;
889.
890.     for( int i = 0 ; i < row ; i ++ )
891.         for ( int j = 0 ; j < column ; j ++ )
892.             cin>>mine[i][j];
893.
894.     initial();
895.
896.
897.     //---- original without forward & heuristic
898.     start = clock();
899.     find_solution(row,column,num_mines);
900.     end = clock();
901.     t = ((double)(end-start))/CLOCKS_PER_SEC;
902.     printf("Time : %fs\n",t);
903.
904.     //---- forward checking without heuristic
905.     cout<<"forward checking without heuristic\n";
906.     start = clock();
907.     find_forward_solution(row,column,num_mines);
908.     end = clock();
909.     t = ((double)(end-start))/CLOCKS_PER_SEC;
910.     printf("Time : %fs\n",t);
911.
912.     //---- forward checking with Degree heuristic

```

```

913.     cout<<"forward checking with Degree heuristic\n";
914.     start = clock();
915.     find_forward_solution_Degree(row,column,num_mines);
916.     end = clock();
917.     t = ((double)(end-start))/CLOCKS_PER_SEC;
918.     printf("Time : %fs\n",t);
919.
920.
921.
922.     //---- forward checking with user-defined heuristic
923.     cout<<"forward checking with TNT heuristic\n";
924.     start = clock();
925.     find_forward_solution_TNT(row,column,num_mines);
926.     end = clock();
927.     t = ((double)(end-start))/CLOCKS_PER_SEC;
928.     printf("Time : %fs\n",t);
929.
930.     //---- forward checking with user-defined heuristic
931.     cout<<"forward checking with MRV heuristic\n";
932.     start = clock();
933.     find_forward_solution_MRV(row,column,num_mines);
934.     end = clock();
935.     t = ((double)(end-start))/CLOCKS_PER_SEC;
936.     printf("Time : %fs\n",t);
937.
938.     return 0;
939. }

```

1. 單純使用 Heuristic function 的 code :

```

1. #include<iostream>
2. #include<stdio.h>
3. #include<stdlib.h>
4. #include<algorithm>
5. #include<vector>
6. #include<stack>
7. #include<queue>
8. #include<time.h>
9. #include<iomanip>
10. #define N 6
11. using namespace std;
12.
13. /*
14.     author : 蔡怡君
15.     content : Minesweeper using Backtrack Search
16. */
17.
18. // mine map
19. int mine[N][N],row,column;
20.
21.
22. struct node
23. {
24.     int assigned_node;
25.     int TNT;
26.     int domain[N][N]; // domain's number {0,1} = 2
27.     int value[N][N]; // value : 0 or 1 (have mines)
28. }first;
29.
30.

```

```

31. struct point
32. {
33.     int x,y;
34.     int degree;
35. }dir[8];
36.
37. vector<point>hint;
38.
39. //point **goal;
40.
41. void initial()
42. {
43.     dir[0]={-1,-1};
44.     dir[1]={-1,0};
45.     dir[2]={-1,1};
46.     dir[3]={0,-1};
47.     dir[4]={0,1};
48.     dir[5]={1,-1};
49.     dir[6]={1,0};
50.     dir[7]={1,1};
51.
52.     first.assigned_node = 0;
53.     first.TNT = 0;
54.     for(int i = 0 ; i < row ; i++){
55.         for(int j = 0 ; j < column ; j++){
56.             if(mine[i][j] == -1){
57.                 first.domain[i][j] = 2;
58.                 first.value[i][j] = -1;
59.             }
60.
61.             else
62.             { // means hint
63.                 point h = {i,j};
64.                 hint.push_back(h);
65.                 first.domain[i][j] = 1;
66.                 first.value[i][j] = mine[i][j];
67.             }
68.         }
69.     }
70. }
71.
72. bool operator == (const node &p1,const node &p2)
73. {
74.     if(p1.TNT != p2.TNT )
75.         return false;
76.     for(int i = 0 ; i < row ; i++){
77.         for(int j = 0 ; j < column ; j++){
78.             if(p1.domain[i][j] != p2.domain[i][j])
79.                 return false;
80.             if(p1.value[i][j] != p2.value[i][j])
81.                 return false;
82.         }
83.     }
84.     return true;
85. }
86.
87. bool isVaild(int i,int j)
88. {
89.     if(mine[i][j] != -1) return false;
90.     return ( (i >= 0 && i < row ) && (j >= 0 && j < column) );
91. }
92.

```

```

93. node copy_node(node A)
94. {
95.     node tmp;
96.     for(int i = 0 ; i < row ; i++){
97.         for(int j = 0 ; j < column ; j++){
98.             tmp.domain[i][j] = A.domain[i][j];
99.             tmp.value[i][j] = A.value[i][j];
100.        }
101.    }
102.    tmp.assigned_node = A.assigned_node;
103.    tmp.TNT = A.TNT;
104.    return tmp;
105.}
106.
107. bool checkVaild(node now, int num_TNT, vector<point> hhint)
108.{
109.    bool all_assigned = true;
110.    if(now.TNT > num_TNT) return false;
111.    for(int i = 0 ; i < hhint.size() ; i++)
112.    {
113.        // check hint 的八維
114.        int TNT = 0;
115.        for(int j = 0 ; j < 8 ; j++){
116.            int px = hhint[i].x + dir[j].x;
117.            int py = hhint[i].y + dir[j].y;
118.            if(isVaild(px , py))
119.            {
120.                if(now.value[px][py] == 1) // 放炸彈的地方
121.                    TNT ++ ;
122.                if(now.domain[px][py] != 1) // 如果八個方位都 assigned 完
123.                    all_assigned = false;
124.            }
125.        }
126.        if(TNT > mine[hhint[i].x][hhint[i].y])
127.            return false;
128.        // 八個方位都 assigned 了 卻不符合 TNT -> false
129.        if(all_assigned == true && TNT != mine[hhint[i].x][hhint[i].y] )
130.            return false;
131.    }
132.    return true;
133.}
134.}
135.
136. void print_solution(node result)
137.{
138.    for(int i = 0 ; i < row ; i++){
139.        for(int j = 0 ; j < column ; j++){
140.            if(mine[i][j] != -1)
141.                cout<<mine[i][j]<<" ";
142.            else if(result.value[i][j] == 1)
143.                cout<<"*"<<" ";
144.            else
145.                cout<<" "<<" ";
146.        }
147.        cout<<endl;
148.    }
149.}
150.
151. // using for checking
152. void print_do(node result)
153.{

```

```

154.     cout<<"domain: \n";
155.     for(int i = 0 ; i < row ; i ++){
156.         for(int j = 0 ; j < column ; j ++){
157.             cout<<result.domain[i][j]<<" ";
158.         }
159.         cout<<endl;
160.     }
161. }
162.
163. // using for checking
164. void print_value(node result)
165. {
166.     cout<<"value: \n";
167.     for(int i = 0 ; i < row ; i ++){
168.         for(int j = 0 ; j < column ; j ++){
169.             if(mine[i][j] == -1 && result.value[i][j] == 1)
170.                 cout<<"*"<<" ";
171.             else
172.                 cout<<setw(2)<<result.value[i][j]<<" ";
173.         }
174.         cout<<endl;
175.     }
176. }
177.
178.
179. bool check_solution(node result)
180. {
181.     for(int i = 0; i < row ; i ++){
182.         for(int j = 0 ; j < column ; j ++){
183.             if(mine[i][j] != -1)
184.             { // local check 8 directions 周圍的 TNT
185.                 int TNT=0;
186.                 for(int k = 0; k < 8 ; k ++){
187.                     if(isVaild(i + dir[k].x,j +dir[k].y))
188.                         if(result.value[i + dir[k].x][j +dir[k].y ] == 1)
189.                             TNT ++;
190.                 }
191.                 if(TNT != mine[i][j])
192.                     return false;
193.             }
194.         }
195.     }
196.     return true;
197. }
198.
199. bool isExplored(stack<node> explored , stack<node>front, node now)
200. {
201.     // 在 Explored set 裡面
202.     while(!explored.empty())
203.     {
204.         node current=explored.top();
205.         explored.pop();
206.         // 如果 current == now 一模一樣
207.         if(current == now)
208.         {
209.             //cout<<"the same"<<endl;
210.             return false;
211.         }
212.     }
213.     // 在 Frontier 裏頭
214.     while(!front.empty())

```



```

215.     {
216.         node current = front.top();
217.         front.pop();
218.         // 如果 current == now 一模一樣
219.         if(current == now)
220.         {
221.             //cout<<"the same"<<endl;
222.             return false;
223.         }
224.     }
225.     return true;
226. }
227.
228. bool isExplored(stack<node> explored , priority_queue<node>front, node now)
229. {
230.     // 在 Explored set 裡面
231.     while(!explored.empty())
232.     {
233.         node current=explored.top();
234.         explored.pop();
235.         // 如果 current == now 一模一樣
236.         if(current == now)
237.         {
238.             //cout<<"the same"<<endl;
239.             return false;
240.         }
241.     }
242.     // 在 Frontier 裏頭
243.     while(!front.empty())
244.     {
245.         node current = front.top();
246.         front.pop();
247.         // 如果 current == now 一模一樣
248.         if(current == now)
249.         {
250.             //cout<<"the same"<<endl;
251.             return false;
252.         }
253.     }
254.     return true;
255. }
256.
257. // original without forward checking / heuristic
258. void find_solution(int r , int c , int num)
259. {
260.     stack<node> front;
261.     stack<node> explored;
262.     int step = 1;
263.     front.push(first);
264.     //bool flag = true;
265.     while(!front.empty())
266.     {
267.         node current = front.top();
268.         explored.push(current);
269.         front.pop();
270.         //cout<<"layer9 : "<<step<<endl;
271.         for(int i = 0 ; i < r ; i++){
272.             for(int j = 0 ; j < c ; j++){
273.                 if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
274.                 {
275.                     //cout<<"fill in (x,y)"<<j<<","<<i<<endl;

```

```

276.         for(int option = 0; option < 2 ; option ++){
277.             node now = copy_node(current);
278.             now.domain[i][j] = 1;
279.             now.value[i][j] = option;
280.             if(option == 1)
281.                 now.TNT++;
282.             if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
283.             {
284.                 // 沒有被 explored 過的才能加進去 !
285.                 if(isExplored(explored,front,now))
286.                 {
287.                     if(now.TNT == num){
288.                         if(check_solution(now)){
289.                             print_solution(now);
290.                             cout<<"node Expand:"<<step<<endl;
291.                             return;
292.                         }
293.                         else
294.                             continue;
295.                     }
296.                     front.push(now);
297.                     step++;
298.                 }
299.             }
300.         }
301.     }
302. }
303. }
304. }
305.     cout<<"No Solution!"<<endl;
306. }
307.
308. void find_forward_solution(int r, int c, int num)
309. {
310.     stack<node> front;
311.     stack<node> explored;
312.     front.push(first);
313.     int step = 1;
314.     while(!front.empty())
315.     {
316.         node current = front.top();
317.         explored.push(current);
318.         front.pop();
319.
320.         // --- forward checking
321.
322.         bool flag = true;
323.         for(int i = 0 ; i < hint.size() ; i ++){
324.             int lowerbound = 0;
325.             int upperbound = 0;
326.             point no = hint[i];
327.             // direction
328.             //cout<<"forward x,y : "<< no.x<<" "<<no.y<<endl;
329.             for(int j = 0 ; j < 8 ; j ++){
330.                 int x = no.x + dir[j].x;
331.                 int y = no.y + dir[j].y;
332.                 if(!isVaild(x,y))
333.                     continue;
334.                 if ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
335.                     lowerbound ++;
336.                 else if(current.domain[x][y] == 2) // unassigned

```

```

337.         upperbound ++;
338.     }
339.     upperbound += lowerbound;
340.     // lower bound & upper bound
341.     if( lowerbound > mine[no.x][no.y] )
342.     {
343.         flag = false;
344.         //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
345.         break;
346.     }
347.     else if (lowerbound == mine [no.x][no.y])
348.     {
349.         // 代表其他 unassigned 的值 都要成為 0
350.         for(int j = 0 ; j < 8 ; j ++){
351.             int x = no.x + dir[j].x;
352.             int y = no.y + dir[j].y;
353.             if(!isVaild(x,y))
354.                 continue;
355.             if(current.domain[x][y] == 2){
356.                 current.domain[x][y] = 1 ;
357.                 current.value[x][y] = 0 ;
358.             }
359.         }
360.     }
361.     if( upperbound < mine[no.x][no.y])
362.     {
363.         flag = false;
364.         //cout<<"upperbound < mine[no.x][no.y]"<<endl;
365.         break;
366.     }
367.     else if( upperbound == mine[no.x][no.y])
368.     {
369.         // 代表其他 unassigned 的值 都要變成 1
370.         for(int j = 0 ; j < 8 ; j ++){
371.             int x = no.x + dir[j].x;
372.             int y = no.y + dir[j].y;
373.             if(!isVaild(x,y))
374.                 continue;
375.             if(current.domain[x][y] == 2){
376.                 current.domain[x][y] = 1 ;
377.                 current.value[x][y] = 1 ;
378.                 current.TNT++;
379.             }
380.         }
381.     }
382.     //cout<<"TNT:"<<current.TNT<<endl;
383.     //print_value(current);
384. }
385. // 代表這個 state 不滿足 state 繼續 pop
386. if(!flag) continue;
387.
388. // 檢查是不是答案
389. if( current.TNT == num )
390. {
391.     if(check_solution(current)){
392.         print_solution(current);
393.         cout<<"node Expand:"<<step<<endl;
394.         return;
395.     }
396.     continue;
397. }

```

```

398. // to assign value !
399. for(int i = 0 ; i < r ; i ++){
400.     for(int j = 0 ; j < c ; j++){
401.         if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
402.         {
403.             //cout<<"fill in (x,y)"<<j<<" "<<i<<endl;
404.             // 去給這 value 是 1(mine) 還是 0
405.             for(int option = 1; option >= 0 ; option --){
406.                 node now = copy_node(current);
407.                 now.domain[i][j] = 1;
408.                 now.value[i][j] = option;
409.                 if(option == 1)
410.                     now.TNT++;
411.                 if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
412.                 {
413.                     // 沒有被 explored 過的才能加進去 !
414.                     if(isExplored(explored,front,now))
415.                     {
416.                         // 如果 TNT 數量 == 炸彈的數量 檢查 solution
417.                         if( now.TNT == num ){
418.                             if(check_solution(now)){
419.                                 print_solution(now);
420.                                 cout<<"node Expand:"<<step<<endl;
421.                                 return;
422.                             }
423.                             else
424.                                 //cout<<"*****"<<endl;
425.                                 continue;
426.                         }
427.                         front.push(now);
428.                         step++;
429.                     }
430.                 }
431.             }
432.         }
433.     }
434. }
435. }
436.
437. }
438.
439. // using for priority queue
440. bool operator < (const point &p1,const point &p2){ return p1.degree < p2.degree;}
441. bool operator > (const point &p1,const point &p2){ return p1.degree > p2.degree;}
442.
443. // forward function with Degree Heuristic
444. void find_forward_solution_Degree(int r, int c, int num)
445. {
446.     stack<node> front;
447.     stack<node> explored;
448.     front.push(first);
449.     int step = 1;
450.     while(!front.empty())
451.     {
452.         node current = front.top();
453.         explored.push(current);
454.         front.pop();
455.
456.         // --- priority_queue using for degree heuristic
457.         priority_queue<point,vector<point>,greater<point> > de;

```

```

458.     for(int i = 0 ; i < r ; i ++){
459.         for(int j = 0 ; j < c ; j ++){
460.             if(mine[i][j] != -1) continue;
461.             point p = { i , j , 0};
462.             for(int k = 0 ; k < 8 ; k ++){
463.                 if(isVaild( i + dir[k].x , j + dir[k].y ))
464.                     if(mine[i + dir[k].x][j + dir[k].y] != -1)
465.                         p.degree ++;
466.                 de.push(p);
467.             }
468.         }
469.         // to assign value !
470.         while(!de.empty()){
471.             point n = de.top();
472.             de.pop();
473.             int i = n.x, j = n.y;
474.
475.             if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
476.             {
477.                 // 去給這 value 是 1(mine) 還是 0
478.                 for(int option = 1; option >= 0 ; option --){
479.                     node now = copy_node(current);
480.                     now.domain[i][j] = 1;
481.                     now.value[i][j] = option;
482.                     if(option == 1)
483.                         now.TNT++;
484.                     // // 沒有被 explored 過的才能加進去！ 且 if 這個點有符合 17 constraints
485.                     if(checkVaild(now,num,hint)&&isExplored(explored,front,now))
486.                     {
487.                         // 如果 TNT 數量 == 炸彈的數量 檢查 solution
488.                         if( now.TNT == num ){
489.                             if(check_solution(now)){
490.                                 print_solution(now);
491.                                 cout<<"node Expand:"<<step<<endl;
492.                                 return;
493.                             }
494.                             else
495.                                 continue;
496.                         }
497.                         front.push(now);
498.                         step++;
499.                     }
500.                 }
501.             }
502.         }
503.     }
504.
505. }
506.
507.
508. bool operator < (const node &p1,const node &p2){ return p1.TNT < p2.TNT;}
509. bool operator > (const node &p1,const node &p2){ return p1.TNT > p2.TNT;}
510.
511. // forward function with TNT Heuristic
512. void find_forward_solution_TNT (int r, int c, int num)
513. {
514.     priority_queue<node> front;
515.     stack<node> explored;
516.     front.push(first);
517.     int step =1;
518.     while(!front.empty())

```

```

519.     {
520.         node current = front.top();
521.         explored.push(current);
522.         front.pop();
523.         if(current.TNT == num)
524.             for(int i = 0 ; i < r; i ++){
525.                 for(int j = 0 ; j < c ; j ++){
526.                     if(current.domain[i][j] == 2){
527.                         current.domain[i][j] = 1;
528.                         current.value[i][j] = 0;
529.                     }
530.
531.                     // to assign value !
532.                     for(int i = 0 ; i < r ; i ++){
533.                         for(int j = 0 ; j < c ; j ++){
534.                             if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned variable
535.                             {
536.                                 //cout<<"fill in (x,y)"<<j<<" "<<i<<endl;
537.                                 // 去給這 value 是 1(mine) 還是 0
538.                                 for(int option = 1; option >= 0 ; option --){
539.                                     node now = copy_node(current);
540.                                     now.domain[i][j] = 1;
541.                                     now.value[i][j] = option;
542.                                     if(option == 1)
543.                                         now.TNT++;
544.                                     if(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
545.                                     {
546.                                         // 沒有被 explored 過的才能加進去 !
547.                                         if(isExplored(explored,front,now))
548.                                         {
549.                                             // 如果 TNT 數量 == 炸彈的數量 檢查 solution
550.                                             if( now.TNT == num ){
551.                                                 if(check_solution(now)){
552.                                                     print_solution(now);
553.                                                     cout<<"node Expand:"<<step<<endl;
554.                                                     return;
555.                                                 }
556.                                                 else
557.                                                     continue;
558.                                             }
559.                                             front.push(now);
560.                                             step++;
561.                                         }
562.                                     }
563.                                 }
564.                             }
565.                         }
566.                     }
567.                 }
568.             }
569. }
570.
571. // forward function with MRV Heuristic
572. void find_forward_solution_MRV(int r, int c, int num)
573. {
574.     stack<node> front;
575.     stack<node> explored;
576.     front.push(first);
577.     int step = 1;
578.     while(!front.empty())
579.     {

```

```

580.     node current = front.top();
581.     explored.push(current);
582.     front.pop();
583.
584.     // --- priority_queue using for degree heuristic
585.     priority_queue<point,vector<point>,greater<point> > de;
586.     for(int i = 0 ; i < r ; i ++){
587.         for(int j = 0 ; j < c ; j ++){
588.             point p = { i , j , current.domain[i][j] };
589.             de.push(p);
590.         }
591.     }
592.
593.
594.     // to assign value !
595.     while(!de.empty()){
596.         point n = de.top();
597.         de.pop();
598.         int i = n.x, j = n.y;
599.
600.         if(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
601.         {
602.             // 去給這 value 是 1(mine) 還是 0
603.             for(int option = 1; option >= 0 ; option --){
604.                 node now = copy_node(current);
605.                 now.domain[i][j] = 1;
606.                 now.value[i][j] = option;
607.                 if(option == 1)
608.                     now.TNT++;
609.                 // // 沒有被 explored 過的才能加進去！ 且 if 這個點有符合 17 constraints
610.                 if(checkVaild(now,num,hint)&&isExplored(explored,front,now))
611.                 {
612.                     // 如果 TNT 數量 == 炸彈的數量 檢查 solution
613.                     if( now.TNT == num ){
614.                         if(check_solution(now)){
615.                             print_solution(now);
616.                             cout<<"node Expand:"<<step<<endl;
617.                             return;
618.                         }
619.                         else
620.                             continue;
621.                     }
622.                     front.push(now);
623.                     step ++;
624.                 }
625.             }
626.         }
627.     }
628. }
629. }
630.
631.
632. int main()
633. {
634.     int num_mines;
635.     time_t start,end;
636.     double t;
637.
638.     cin >> row >> column >> num_mines;
639.
640.     for( int i = 0 ; i < row ; i ++ )

```

```

641.         for ( int j = 0 ; j < column ; j ++ )
642.             cin>>mine[i][j];
643.
644.     initial();
645.
646.     /*
647.     //---- original without forward & heuristic
648.     start = clock();
649.     find_solution(row,column,num_mines);
650.     end = clock();
651.     t = ((double)(end-start))/CLOCKS_PER_SEC;
652.     printf("Time : %fs\n",t);    */
653.
654.     //---- forward checking without heuristic
655.     cout<<"forward checking without heuristic\n";
656.     start = clock();
657.     find_forward_solution(row,column,num_mines);
658.     end = clock();
659.     t = ((double)(end-start))/CLOCKS_PER_SEC;
660.     printf("Time : %fs\n",t);
661.
662.
663.     //---- with Degree heuristic
664.     cout<<"with Degree heuristic\n";
665.     start = clock();
666.     find_forward_solution_Degree(row,column,num_mines);
667.     end = clock();
668.     t = ((double)(end-start))/CLOCKS_PER_SEC;
669.     printf("Time : %fs\n",t);
670.
671.
672.
673.     //---- with user-defined heuristic
674.     cout<<"with TNT heuristic\n";
675.     start = clock();
676.     find_forward_solution_TNT(row,column,num_mines);
677.     end = clock();
678.     t = ((double)(end-start))/CLOCKS_PER_SEC;
679.     printf("Time : %fs\n",t);
680.
681.     //---- with MRV heuristic
682.     cout<<"with MRV heuristic\n";
683.     start = clock();
684.     find_forward_solution_MRV(row,column,num_mines);
685.     end = clock();
686.     t = ((double)(end-start))/CLOCKS_PER_SEC;
687.     printf("Time : %fs\n",t);
688.
689.     return 0;
690. }

```