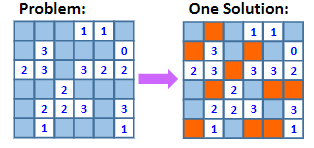
人工智慧概論

0613413 蔡怡君

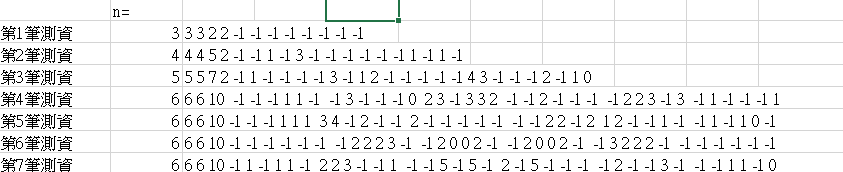
－實驗目的：踩地雷，Minesweeper，給定一個Mine Map，去找到結果，例如：

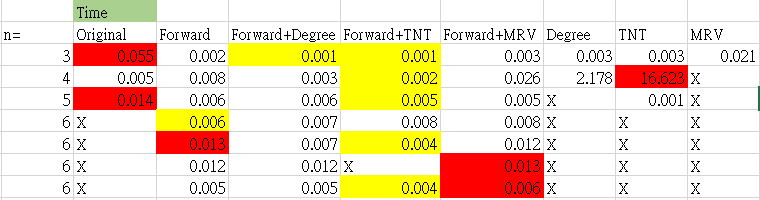


－實驗方法：(depth-first search + forward checking + Heuristic)  
－使用的概念：在explored set和frontier都沒有相同state的node ( 直接使用table 去記錄這個點有沒有被visit過 )：將其加入frontier !

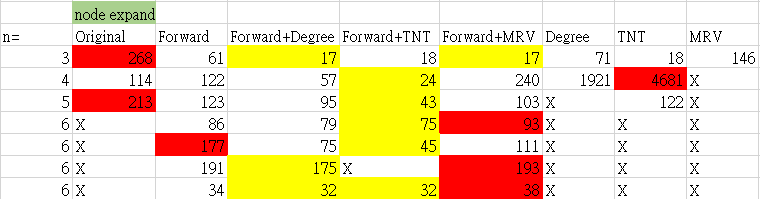
1. **最原始Depth First search：**只有在push進去前去檢查，是否符合local constraint的需求，例如是否hint周圍的mine數量小於或是等於hint mine。
2. **Depth First Search + MRV Heuristic：**看Domain的關係， Domain= 1的點比較早被Assigned。
3. **Depth First Search + Degree Heuristic：**看這個每一格的Variable跟幾個constraint有關，Constraint越多的點越早被Assigned。
4. **Depth First Search + 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： **時間分析(Time ) :  
紅色為最慢的，黃色為表現最佳的。**



**Node expansion:**



**表現總結**

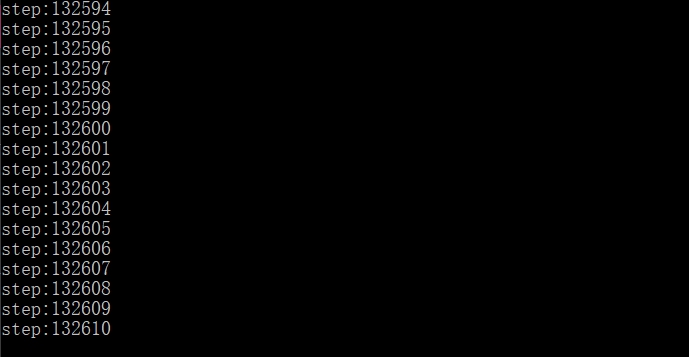
|  |  |
| --- | --- |
| Original | 效率非常差，因為需要去展開所有的node，展開到child node後才能去檢查。 |
| Degree | 表現普通。 |
| TNT | 在n=4時花費的時間較長，但是n=5的時候表現為最佳的。 |
| MRV | 只有在n=3的時候可以展開。 |
| Forward | 表現普通。 |
| Forward + Degree | 表現佳，但通常不是最快速的。 |
| Forward + TNT | 在大多的表現內是最好的，但是在n=6的第三個測資的時候，因為時間過於長，所以無法記錄下來。 |
| Forward + MRV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

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;
13. /\*
14. author : 蔡怡君
15. content : Minesweeper using Backtrack Search
16. \*/
18. // mine map
19. **int** mine[N][N],row,column;

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;

31. **struct** point
32. {
33. **int** x,y;
34. **int** degree;
35. }dir[8];
37. vector<point>hint;
39. //point \*\*goal;
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};
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. }
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. }
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. }
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. }
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. }
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. \*/
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. }
151. }
152. **if**(TNT > mine[hhint[i].x][hhint[i].y])
153. **return** **false**;
154. // 八個方位都 assigned了 卻不符合TNT -> false
155. **if**(all\_assigned == **true** && TNT != mine[hhint[i].x][hhint[i].y] )
156. **return** **false**;
157. }
158. **return** **true**;
159. }
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. }
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. }
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. }

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. }
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. }
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. }
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 varaible
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. }
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();
345. // --- forward checking
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unassinged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**;
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 varaible
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. }
459. }
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;}
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();
478. // --- priorty\_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. }
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unassinged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**;
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. }
568. // to assign value！
569. **while**(!de.empty()){
570. point n = de.top();
571. de.pop();
572. **int** i = n.x, j = n.y;
574. **if**(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
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. }
604. }

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;}
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. }
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unassinged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**;
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. }
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 varaible
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. }
744. }
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();
759. // --- priorty\_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. }
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unassinged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**;
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. }
844. // to assign value！
845. **while**(!de.empty()){
846. point n = de.top();
847. de.pop();
848. **int** i = n.x, j = n.y;
850. **if**(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
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. }

882. **int** main()
883. {
884. **int** num\_mines;
885. **time\_t** start,end;
886. **double** t;
888. cin >> row >> column >> num\_mines;
890. **for**( **int** i = 0 ; i < row ; i ++ )
891. **for** ( **int** j = 0 ; j < column ; j ++ )
892. cin>>mine[i][j];
894. initial();

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);
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);
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);


922. //----  forward checking with uesr-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);
930. //----  forward checking with uesr-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);
938. **return** 0;
939. }
940. **單純使用Heuristic function的code：**  
     #include<iostream>
941. #include<stdio.h>
942. #include<stdlib.h>
943. #include<algorithm>
944. #include<vector>
945. #include<stack>
946. #include<queue>
947. #include<time.h>
948. #include<iomanip>
949. #define N 6
950. **using** **namespace** std;
952. /\*
953. author : 蔡怡君
954. content : Minesweeper using Backtrack Search
955. \*/
957. // mine map
958. **int** mine[N][N],row,column;

961. **struct** node
962. {
963. **int** assigned\_node;
964. **int** TNT;
965. **int** domain[N][N]; // domain's number {0,1} = 2
966. **int** value[N][N]; // value : 0 or 1 (have mines)
967. }first;

970. **struct** point
971. {
972. **int** x,y;
973. **int** degree;
974. }dir[8];
976. vector<point>hint;
978. //point \*\*goal;
980. **void** initial()
981. {
982. dir[0]={-1,-1};
983. dir[1]={-1,0};
984. dir[2]={-1,1};
985. dir[3]={0,-1};
986. dir[4]={0,1};
987. dir[5]={1,-1};
988. dir[6]={1,0};
989. dir[7]={1,1};
991. first.assigned\_node = 0;
992. first.TNT = 0;
993. **for**(**int** i = 0 ; i < row ; i ++){
994. **for**(**int** j = 0 ; j < column ; j ++){
995. **if**(mine[i][j] == -1){
996. first.domain[i][j] = 2;
997. first.value[i][j] = -1;
998. }
1000. **else**
1001. {   // means hint
1002. point h = {i,j};
1003. hint.push\_back(h);
1004. first.domain[i][j] = 1;
1005. first.value[i][j] = mine[i][j];
1006. }
1007. }
1008. }
1009. }
1011. **bool** operator == (**const** node &p1,**const** node &p2)
1012. {
1013. **if**(p1.TNT != p2.TNT )
1014. **return** **false**;
1015. **for**(**int** i = 0 ; i < row ; i ++){
1016. **for**(**int** j = 0 ; j < column ; j ++){
1017. **if**(p1.domain[i][j] != p2.domain[i][j])
1018. **return** **false**;
1019. **if**(p1.value[i][j] != p2.value[i][j])
1020. **return** **false**;
1021. }
1022. }
1023. **return** **true**;
1024. }
1026. **bool** isVaild(**int** i,**int** j)
1027. {
1028. **if**(mine[i][j] != -1) **return** **false**;
1029. **return** ( (i >= 0 && i < row ) && (j >= 0 && j < column) );
1030. }
1032. node copy\_node(node A)
1033. {
1034. node tmp;
1035. **for**(**int** i = 0 ; i < row ;i ++){
1036. **for**(**int** j = 0 ; j < column ; j ++){
1037. tmp.domain[i][j] = A.domain[i][j];
1038. tmp.value[i][j] = A.value[i][j];
1039. }
1040. }
1041. tmp.assigned\_node = A.assigned\_node;
1042. tmp.TNT = A.TNT;
1043. **return** tmp;
1044. }
1046. **bool** checkVaild(node now, **int** num\_TNT, vector<point> hhint)
1047. {
1048. **bool** all\_assigned = **true**;
1049. **if**(now.TNT > num\_TNT)    **return** **false**;
1050. **for**(**int** i = 0 ; i < hhint.size() ; i ++)
1051. {
1052. // check hint 的八維
1053. **int** TNT = 0;
1054. **for**(**int** j = 0 ; j < 8 ; j ++){
1055. **int** px = hhint[i].x + dir[j].x;
1056. **int** py = hhint[i].y + dir[j].y;
1057. **if**(isVaild(px , py))
1058. {
1059. **if**(now.value[px][py] == 1) // 放炸彈的地方
1060. TNT ++ ;
1061. **if**(now.domain[px][py] != 1) // 如果八個方位都assigned完
1062. all\_assigned = **false**;
1063. }
1065. }
1066. **if**(TNT > mine[hhint[i].x][hhint[i].y])
1067. **return** **false**;
1068. // 八個方位都 assigned了 卻不符合TNT -> false
1069. **if**(all\_assigned == **true** && TNT != mine[hhint[i].x][hhint[i].y] )
1070. **return** **false**;
1071. }
1072. **return** **true**;
1073. }
1075. **void** print\_solution(node result)
1076. {
1077. **for**(**int** i = 0 ; i < row ;i ++){
1078. **for**(**int** j = 0 ; j < column ; j ++){
1079. **if**(mine[i][j] != -1)
1080. cout<<mine[i][j]<<" ";
1081. **else** **if**(result.value[i][j] == 1)
1082. cout<<"\*"<<" ";
1083. **else**
1084. cout<<" "<<" ";
1085. }
1086. cout<<endl;
1087. }
1088. }
1090. // using for checking
1091. **void** print\_do(node result)
1092. {
1093. cout<<"domain: \n";
1094. **for**(**int** i = 0 ; i < row ;i ++){
1095. **for**(**int** j = 0 ; j < column ; j ++){
1096. cout<<result.domain[i][j]<<" ";
1097. }
1098. cout<<endl;
1099. }
1100. }
1102. // using for checking
1103. **void** print\_value(node result)
1104. {
1105. cout<<"value: \n";
1106. **for**(**int** i = 0 ; i < row ;i ++){
1107. **for**(**int** j = 0 ; j < column ; j ++){
1108. **if**(mine[i][j] == -1 && result.value[i][j] == 1)
1109. cout<<" \*"<<" ";
1110. **else**
1111. cout<<setw(2)<<result.value[i][j]<<" ";
1112. }
1113. cout<<endl;
1114. }
1115. }

1118. **bool** check\_solution(node result)
1119. {
1120. **for**(**int** i = 0; i < row ; i ++){
1121. **for**(**int** j = 0 ; j < column ; j ++){
1122. **if**(mine[i][j] != -1)
1123. {   // local check 8 directions 周圍的TNT
1124. **int** TNT=0;
1125. **for**(**int** k = 0; k < 8 ; k ++){
1126. **if**(isVaild(i + dir[k].x,j +dir[k].y))
1127. **if**(result.value[i + dir[k].x][j +dir[k].y ] == 1)
1128. TNT ++;
1129. }
1130. **if**(TNT != mine[i][j])
1131. **return** **false**;
1132. }
1133. }
1134. }
1135. **return** **true**;
1136. }
1138. **bool** isExplored(stack<node> explored , stack<node>front, node now)
1139. {
1140. // 在 Explored set裡面
1141. **while**(!explored.empty())
1142. {
1143. node current=explored.top();
1144. explored.pop();
1145. // 如果current == now 一模一樣
1146. **if**(current == now)
1147. {
1148. //cout<<"the same"<<endl;
1149. **return** **false**;
1150. }
1151. }
1152. // 在 Frontier 裏頭
1153. **while**(!front.empty())
1154. {
1155. node current = front.top();
1156. front.pop();
1157. // 如果current == now 一模一樣
1158. **if**(current == now)
1159. {
1160. //cout<<"the same"<<endl;
1161. **return** **false**;
1162. }
1163. }
1164. **return** **true**;
1165. }
1167. **bool** isExplored(stack<node> explored , priority\_queue<node>front, node now)
1168. {
1169. // 在 Explored set裡面
1170. **while**(!explored.empty())
1171. {
1172. node current=explored.top();
1173. explored.pop();
1174. // 如果current == now 一模一樣
1175. **if**(current == now)
1176. {
1177. //cout<<"the same"<<endl;
1178. **return** **false**;
1179. }
1180. }
1181. // 在 Frontier 裏頭
1182. **while**(!front.empty())
1183. {
1184. node current = front.top();
1185. front.pop();
1186. // 如果current == now 一模一樣
1187. **if**(current == now)
1188. {
1189. //cout<<"the same"<<endl;
1190. **return** **false**;
1191. }
1192. }
1193. **return** **true**;
1194. }
1196. // original without forward checking / heuristic
1197. **void** find\_solution(**int** r , **int** c , **int** num)
1198. {
1199. stack<node> front;
1200. stack<node> explored;
1201. **int** step = 1;
1202. front.push(first);
1203. //bool flag = true;
1204. **while**(!front.empty())
1205. {
1206. node current = front.top();
1207. explored.push(current);
1208. front.pop();
1209. //cout<<"layer9 :"<<step<<endl;
1210. **for**(**int** i = 0 ; i < r ; i ++){
1211. **for**(**int** j = 0 ; j < c ; j++){
1212. **if**(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
1213. {
1214. //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
1215. **for**(**int** option = 0; option < 2 ; option ++){
1216. node now = copy\_node(current);
1217. now.domain[i][j] = 1;
1218. now.value[i][j] = option;
1219. **if**(option == 1)
1220. now.TNT++;
1221. **if**(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
1222. {
1223. // 沒有被explored過的才能加進去！
1224. **if**(isExplored(explored,front,now))
1225. {
1226. **if**(now.TNT == num){
1227. **if**(check\_solution(now)){
1228. print\_solution(now);
1229. cout<<"node Expand:"<<step<<endl;
1230. **return**;
1231. }
1232. **else**
1233. **continue**;
1234. }
1235. front.push(now);
1236. step++;
1237. }
1238. }
1239. }
1240. }
1241. }
1242. }
1243. }
1244. cout<<"No Solution!"<<endl;
1245. }
1247. **void** find\_forward\_solution(**int** r, **int** c, **int** num)
1248. {
1249. stack<node> front;
1250. stack<node> explored;
1251. front.push(first);
1252. **int** step = 1;
1253. **while**(!front.empty())
1254. {
1255. node current = front.top();
1256. explored.push(current);
1257. front.pop();
1259. // --- forward checking
1261. **bool** flag = **true**;
1262. **for**(**int** i = 0 ; i < hint.size() ; i ++){
1263. **int** lowerbound = 0;
1264. **int** upperbound = 0;
1265. point no = hint[i];
1266. // direction
1267. //cout<<"forward x,y :"<< no.x<<","<<no.y<<endl;
1268. **for**(**int** j = 0 ; j < 8 ; j ++){
1269. **int** x = no.x + dir[j].x;
1270. **int** y = no.y + dir[j].y;
1271. **if**(!isVaild(x,y))
1272. **continue**;
1273. **if** ( current.domain[x][y] == 1 && current.value[x][y] == 1 )
1274. lowerbound ++;
1275. **else** **if**(current.domain[x][y] == 2) // unassigned
1276. upperbound ++;
1277. }
1278. upperbound += lowerbound;
1279. // lower bound & upper bound
1280. **if**( lowerbound > mine[no.x][no.y] )
1281. {
1282. flag = **false**;
1283. //cout<<"lowerbound > mine[no.x][no.y]"<<endl;
1284. **break**;
1285. }
1286. **else** **if** (lowerbound == mine [no.x][no.y])
1287. {
1288. // 代表其他unassinged的值 都要成為 0
1289. **for**(**int** j = 0 ; j < 8 ; j ++){
1290. **int** x = no.x + dir[j].x;
1291. **int** y = no.y + dir[j].y;
1292. **if**(!isVaild(x,y))
1293. **continue**;
1294. **if**(current.domain[x][y] == 2){
1295. current.domain[x][y] = 1 ;
1296. current.value[x][y] = 0 ;
1297. }
1298. }
1299. }
1300. **if**( upperbound < mine[no.x][no.y])
1301. {
1302. flag = **false**;
1303. //cout<<"upperbound < mine[no.x][no.y]"<<endl;
1304. **break**;
1305. }
1306. **else** **if**( upperbound == mine[no.x][no.y])
1307. {
1308. // 代表其他 unassigned的值 都要變成 1
1309. **for**(**int** j = 0 ; j < 8 ; j ++){
1310. **int** x = no.x + dir[j].x;
1311. **int** y = no.y + dir[j].y;
1312. **if**(!isVaild(x,y))
1313. **continue**;
1314. **if**(current.domain[x][y] == 2){
1315. current.domain[x][y] = 1 ;
1316. current.value[x][y] = 1 ;
1317. current.TNT++;
1318. }
1319. }
1320. }
1321. //cout<<"TNT:"<<current.TNT<<endl;
1322. //print\_value(current);
1323. }
1324. // 代表這個state不滿足 state 繼續pop
1325. **if**(!flag) **continue**;
1327. // 檢查是不是答案
1328. **if**( current.TNT == num )
1329. {
1330. **if**(check\_solution(current)){
1331. print\_solution(current);
1332. cout<<"node Expand:"<<step<<endl;
1333. **return**;
1334. }
1335. **continue**;
1336. }
1337. // to assign value！
1338. **for**(**int** i = 0 ; i < r ; i ++){
1339. **for**(**int** j = 0 ; j < c ; j++){
1340. **if**(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
1341. {
1342. //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
1343. // 去給這 value 是 1(mine) 還是 0
1344. **for**(**int** option = 1; option >= 0  ; option --){
1345. node now = copy\_node(current);
1346. now.domain[i][j] = 1;
1347. now.value[i][j] = option;
1348. **if**(option == 1)
1349. now.TNT++;
1350. **if**(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
1351. {
1352. // 沒有被explored過的才能加進去！
1353. **if**(isExplored(explored,front,now))
1354. {
1355. // 如果TNT數量 == 炸彈的數量 檢查solution
1356. **if**( now.TNT == num ){
1357. **if**(check\_solution(now)){
1358. print\_solution(now);
1359. cout<<"node Expand:"<<step<<endl;
1360. **return**;
1361. }
1362. **else**
1363. //cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
1364. **continue**;
1365. }
1366. front.push(now);
1367. step++;
1368. }
1369. }
1370. }
1371. }
1372. }
1373. }
1374. }
1376. }
1378. // using for priority queue
1379. **bool** operator < (**const** point &p1,**const** point &p2){ **return** p1.degree < p2.degree;}
1380. **bool** operator > (**const** point &p1,**const** point &p2){ **return** p1.degree > p2.degree;}
1382. // forward function with Degree Heuristic
1383. **void** find\_forward\_solution\_Degree(**int** r, **int** c, **int** num)
1384. {
1385. stack<node> front;
1386. stack<node> explored;
1387. front.push(first);
1388. **int** step = 1;
1389. **while**(!front.empty())
1390. {
1391. node current = front.top();
1392. explored.push(current);
1393. front.pop();
1395. // --- priorty\_queue using for degree heuristic
1396. priority\_queue<point,vector<point>,greater<point> > de;
1397. **for**(**int** i = 0 ; i < r ; i ++){
1398. **for**(**int** j = 0 ; j < c ; j ++){
1399. **if**(mine[i][j] != -1) **continue**;
1400. point p = { i , j , 0};
1401. **for**(**int** k = 0 ; k < 8 ; k ++)
1402. **if**(isVaild( i + dir[k].x , j + dir[k].y ))
1403. **if**(mine[i + dir[k].x][j + dir[k].y] != -1)
1404. p.degree ++;
1405. de.push(p);
1406. }
1407. }
1408. // to assign value！
1409. **while**(!de.empty()){
1410. point n = de.top();
1411. de.pop();
1412. **int** i = n.x, j = n.y;
1414. **if**(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
1415. {
1416. // 去給這 value 是 1(mine) 還是 0
1417. **for**(**int** option = 1; option >= 0  ; option --){
1418. node now = copy\_node(current);
1419. now.domain[i][j] = 1;
1420. now.value[i][j] = option;
1421. **if**(option == 1)
1422. now.TNT++;
1423. // // 沒有被explored過的才能加進去！ 且 if 這個點有符合 17 constraints
1424. **if**(checkVaild(now,num,hint)&&isExplored(explored,front,now))
1425. {
1426. // 如果TNT數量 == 炸彈的數量 檢查solution
1427. **if**( now.TNT == num ){
1428. **if**(check\_solution(now)){
1429. print\_solution(now);
1430. cout<<"node Expand:"<<step<<endl;
1431. **return**;
1432. }
1433. **else**
1434. **continue**;
1435. }
1436. front.push(now);
1437. step++;
1438. }
1439. }
1440. }
1441. }
1442. }
1444. }

1447. **bool** operator < (**const** node &p1,**const** node &p2){ **return** p1.TNT < p2.TNT;}
1448. **bool** operator > (**const** node &p1,**const** node &p2){ **return** p1.TNT > p2.TNT;}
1450. // forward function with TNT Heuristic
1451. **void** find\_forward\_solution\_TNT  (**int** r, **int** c, **int** num)
1452. {
1453. priority\_queue<node> front;
1454. stack<node> explored;
1455. front.push(first);
1456. **int** step =1;
1457. **while**(!front.empty())
1458. {
1459. node current = front.top();
1460. explored.push(current);
1461. front.pop();
1462. **if**(current.TNT == num)
1463. **for**(**int** i = 0 ; i < r; i ++)
1464. **for**(**int** j = 0 ; j <c ; j ++)
1465. **if**(current.domain[i][j] == 2){
1466. current.domain[i][j] = 1;
1467. current.value[i][j] = 0;
1468. }
1470. // to assign value！
1471. **for**(**int** i = 0 ; i < r ; i ++){
1472. **for**(**int** j = 0 ; j < c ; j++){
1473. **if**(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
1474. {
1475. //cout<<"fill in (x,y)"<<j<<","<<i<<endl;
1476. // 去給這 value 是 1(mine) 還是 0
1477. **for**(**int** option = 1; option >= 0  ; option --){
1478. node now = copy\_node(current);
1479. now.domain[i][j] = 1;
1480. now.value[i][j] = option;
1481. **if**(option == 1)
1482. now.TNT++;
1483. **if**(checkVaild(now,num,hint)) // if 這個點有符合 17 constraints
1484. {
1485. // 沒有被explored過的才能加進去！
1486. **if**(isExplored(explored,front,now))
1487. {
1488. // 如果TNT數量 == 炸彈的數量 檢查solution
1489. **if**( now.TNT == num ){
1490. **if**(check\_solution(now)){
1491. print\_solution(now);
1492. cout<<"node Expand:"<<step<<endl;
1493. **return**;
1494. }
1495. **else**
1496. **continue**;
1497. }
1498. front.push(now);
1499. step++;
1500. }
1501. }
1502. }
1503. }
1504. }
1505. }
1506. }
1508. }
1510. // forward function with MRV Heuristic
1511. **void** find\_forward\_solution\_MRV(**int** r, **int** c, **int** num)
1512. {
1513. stack<node> front;
1514. stack<node> explored;
1515. front.push(first);
1516. **int** step = 1;
1517. **while**(!front.empty())
1518. {
1519. node current = front.top();
1520. explored.push(current);
1521. front.pop();
1523. // --- priorty\_queue using for degree heuristic
1524. priority\_queue<point,vector<point>,greater<point> > de;
1525. **for**(**int** i = 0 ; i < r ; i ++){
1526. **for**(**int** j = 0 ; j < c ; j ++){
1527. point p = { i , j , current.domain[i][j] };
1528. de.push(p);
1529. }
1530. }

1533. // to assign value！
1534. **while**(!de.empty()){
1535. point n = de.top();
1536. de.pop();
1537. **int** i = n.x, j = n.y;
1539. **if**(mine[i][j] == -1 && current.domain[i][j] == 2) // unassigned varaible
1540. {
1541. // 去給這 value 是 1(mine) 還是 0
1542. **for**(**int** option = 1; option >= 0  ; option --){
1543. node now = copy\_node(current);
1544. now.domain[i][j] = 1;
1545. now.value[i][j] = option;
1546. **if**(option == 1)
1547. now.TNT++;
1548. // // 沒有被explored過的才能加進去！ 且 if 這個點有符合 17 constraints
1549. **if**(checkVaild(now,num,hint)&&isExplored(explored,front,now))
1550. {
1551. // 如果TNT數量 == 炸彈的數量 檢查solution
1552. **if**( now.TNT == num ){
1553. **if**(check\_solution(now)){
1554. print\_solution(now);
1555. cout<<"node Expand:"<<step<<endl;
1556. **return**;
1557. }
1558. **else**
1559. **continue**;
1560. }
1561. front.push(now);
1562. step ++;
1563. }
1564. }
1565. }
1566. }
1567. }
1568. }

1571. **int** main()
1572. {
1573. **int** num\_mines;
1574. **time\_t** start,end;
1575. **double** t;
1577. cin >> row >> column >> num\_mines;
1579. **for**( **int** i = 0 ; i < row ; i ++ )
1580. **for** ( **int** j = 0 ; j < column ; j ++ )
1581. cin>>mine[i][j];
1583. initial();
1585. /\*
1586. //---- original without forward & heuristic
1587. start = clock();
1588. find\_solution(row,column,num\_mines);
1589. end = clock();
1590. t = ((double)(end-start))/CLOCKS\_PER\_SEC;
1591. printf("Time : %fs\n",t);   \*/
1593. //----  forward checking without heuristic
1594. cout<<"forward checking without heuristic\n";
1595. start = clock();
1596. find\_forward\_solution(row,column,num\_mines);
1597. end = clock();
1598. t = ((**double**)(end-start))/CLOCKS\_PER\_SEC;
1599. printf("Time : %fs\n",t);

1602. //----   with Degree heuristic
1603. cout<<"with Degree heuristic\n";
1604. start = clock();
1605. find\_forward\_solution\_Degree(row,column,num\_mines);
1606. end = clock();
1607. t = ((**double**)(end-start))/CLOCKS\_PER\_SEC;
1608. printf("Time : %fs\n",t);


1612. //----  with uesr-defined heuristic
1613. cout<<"with TNT heuristic\n";
1614. start = clock();
1615. find\_forward\_solution\_TNT(row,column,num\_mines);
1616. end = clock();
1617. t = ((**double**)(end-start))/CLOCKS\_PER\_SEC;
1618. printf("Time : %fs\n",t);
1620. //----   with MRV heuristic
1621. cout<<"with MRV heuristic\n";
1622. start = clock();
1623. find\_forward\_solution\_MRV(row,column,num\_mines);
1624. end = clock();
1625. t = ((**double**)(end-start))/CLOCKS\_PER\_SEC;
1626. printf("Time : %fs\n",t);
1628. **return** 0;
1629. }