

CPL Midterm 2

TAs

簡易 Debugger

Problem

Description

在過去半學期中，石上會計 (人名) 發現同學在 debug 時常常看不懂自己寫的程式是如何出錯的，像程式的執行順序，變數的變化等等。

在 cout 大法的幫助下，大部分同學逐漸掌握 debug 技巧，但開始學習 function 之後，石上會計擔心同學沒辦法想像 function 的呼叫順序，因此帮大家設計了一種方便的 debug 技巧，請同學幫忙實作出來。

在有 function 的題目中，可能有多個 function 以特定順序被呼叫，且同一個 function 可能會被重複呼叫。請同學印出每個 function 的呼叫順序，並記下同一個

function 被呼叫的次數。

```
int main(){
    int i;

    while(cin >> i){
        i %= 3;

        if(i == 0)
            func0();
        else if(i == 1)
            func1();
        else
            func2();
    }

    return 0;
}
```

例如上述程式，不斷輸入任意整數 `i`

- 如果 `i` 被三整除，呼叫 `func0`
- 如果 `i` 被三除餘一，呼叫 `func1`
- 如果 `i` 被三除餘二，呼叫 `func2`

請同學印出呼叫順序，並印出每次呼叫時的累計次數，如範例輸出所示。

Solution

- Global variable
- Static variable

```
int count = 1;
void func0(){
    static int count_0 = 0;
    switch(count_0){
        case(0):
            cout << "Line " << count << ": func0() is called the first time";
            break;
        case(1):
            cout << "Line " << count << ": func0() is called the second time";
            break;
        case(2):
            cout << "Line " << count << ": func0() is called the third time";
            break;
        case(3):
            cout << "Line " << count << ": func0() is called the fourth time";
            break;
        case(4):
            cout << "Line " << count << ": func0() is called the fifth time";
            break;
        case(5):
            cout << "Line " << count << ": func0() is called the sixth time";
            break;
        case(6):
            cout << "Line " << count << ": func0() is called the seventh time";
            break;
        case(7):
            cout << "Line " << count << ": func0() is called the eighth time";
            break;
        case(8):
            cout << "Line " << count << ": func0() is called the ninth time";
            break;
        case(9):
            cout << "Line " << count << ": func0() is called the tenth time";
            break;
    }
    cout << endl;
    count_0++;
    count++;
}
```

Weather Forecast Quality

Problem

In a weather forecast, given the actual and forecasted temperatures for each day, find the sum of the weather forecast inaccuracies across all days.

The weather forecast inaccuracy on any day is the absolute difference of the actual temperature and the forecasted temperature. The absolute difference for x and y is $|x - y|$.

Please implement a function that takes in two arrays, denoting actual temperatures and denoting forecasted temperatures across days, and returns the sum of the weather forecast inaccuracies for all days.

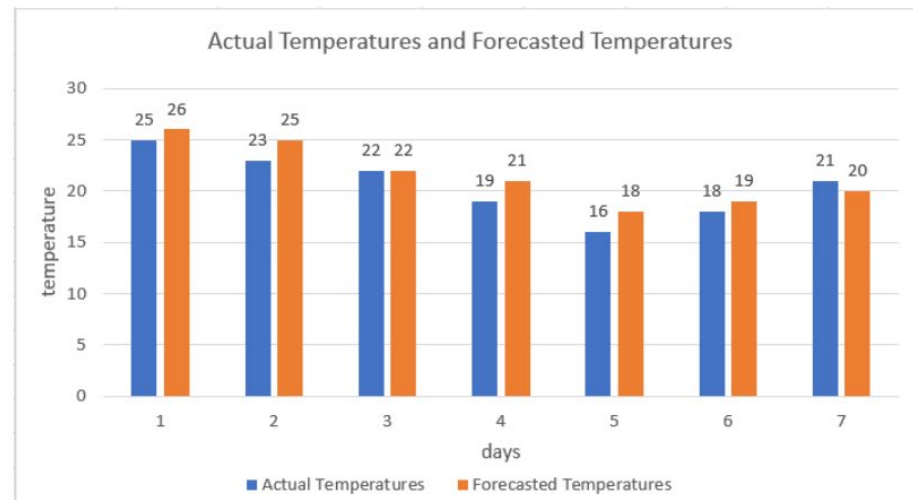
```
25 23 22 19 16 18 21
```

and input array for forecasted temperatures is

```
26 25 22 21 18 19 20
```

You should get output 9 since

$$|25 - 26| + |23 - 25| + |22 - 22| + |19 - 21| + |16 - 18| + |18 - 19| + |21 - 20| = 9$$



Solution

1. Deal with different parameter types !!
2. Use call by reference to change the value of *sum* in function

```
int sum = 0;  
totalForecastInaccuracy(x_int, y_int, sum, N);  
totalForecastInaccuracy(x_float, y_float, sum, N);  
totalForecastInaccuracy(x_double, y_double, sum, N);  
totalForecastInaccuracy(x_long_double, y_long_double, sum, N);  
  
cout << sum << endl;
```

Method 1 – Function Overloading

Mind the return type of function.

Non-void function may lead to TLE

/ Run Time Error on OJ while fine
in your IDE

Call by reference !!

```
void totalForecastInaccuracy(int *x, int *y, int &sum, int N){
    int temp_x = 0, temp_y = 0;
    for(int i=0; i<N; i++){
        temp_x = round(x[i]);
        temp_y = round(y[i]);
        if((temp_x - temp_y) >= 0) sum += temp_x - temp_y;
        else sum += temp_y - temp_x;
    }
}

void totalForecastInaccuracy(float *x, float *y, int &sum, int N){
    int temp_x = 0, temp_y = 0;
    for(int i=0; i<N; i++){
        temp_x = round(x[i]);
        temp_y = round(y[i]);
        if((temp_x - temp_y) >= 0) sum += temp_x - temp_y;
        else sum += temp_y - temp_x;
    }
}
```

```
void totalForecastInaccuracy(double *x, double *y, int &sum, int N){
    int temp_x = 0, temp_y = 0;
    for(int i=0; i<N; i++){
        temp_x = round(x[i]);
        temp_y = round(y[i]);
        if((temp_x - temp_y) >= 0) sum += temp_x - temp_y;
        else sum += temp_y - temp_x;
    }
}

void totalForecastInaccuracy(long double *x, long double *y, int &sum, int N){
    int temp_x = 0, temp_y = 0;
    for(int i=0; i<N; i++){
        temp_x = round(x[i]);
        temp_y = round(y[i]);
        if((temp_x - temp_y) >= 0) sum += temp_x - temp_y;
        else sum += temp_y - temp_x;
    }
}
```


Method 2 – Template

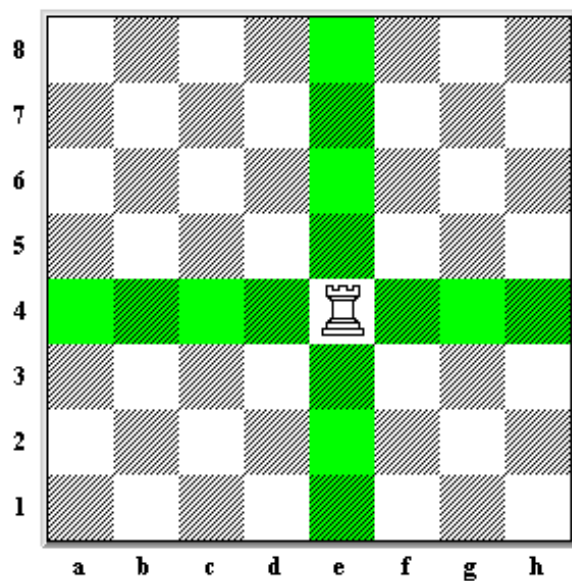
```
template <class T>
void totalForecastInaccuracy(T *x, T *y, int &sum, int N){
    int temp_x = 0, temp_y = 0;
    for(int i=0; i<N; i++){
        temp_x = round(x[i]);
        temp_y = round(y[i]);
        if((temp_x - temp_y) >= 0) sum += temp_x - temp_y;
        else sum += temp_y - temp_x;
    }
}
```

Rooks

Problem

Description

The picture below shows how a rook moves in chess, same as "車" in Chinese chess.

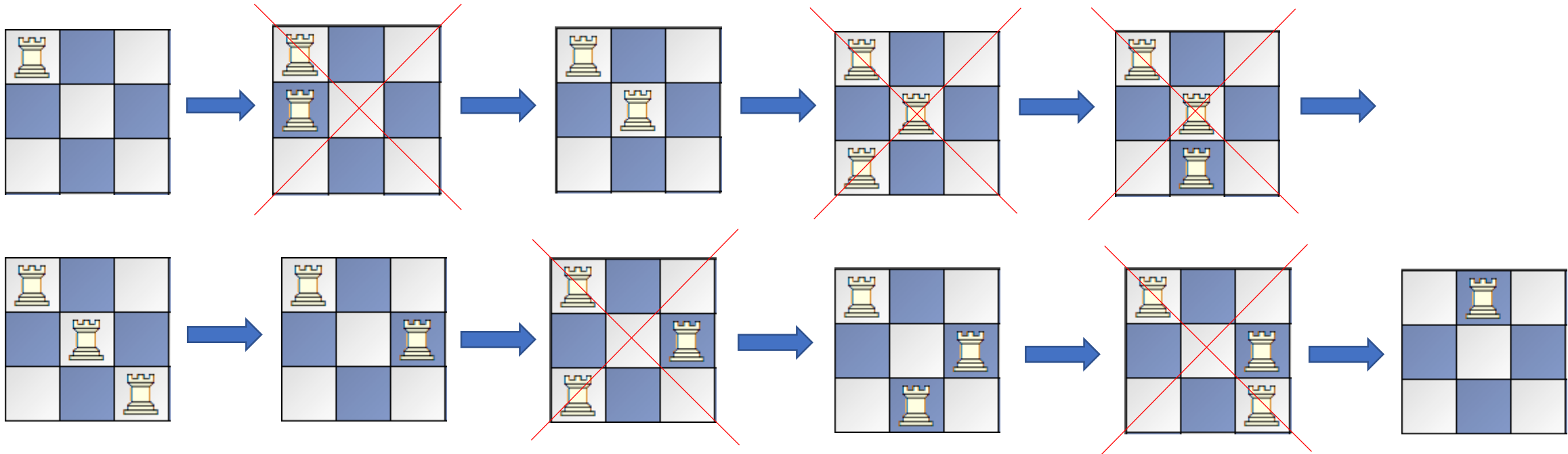


The N rooks puzzle is the problem of placing N chess rooks on an $N \times N$ chess board so that no two rooks threaten each other.

Thus, a solution requires that no two rooks share the same row and column.

Solution

- Start from top-left to bottom-right
- Recursive



```

int N;
int ans = 0;

int main()
{
    cin >> N;

    int **mat = new int*[N];
    for (int i = 0; i < N; i++)
        mat[i] = new int[N]();

    nRook(mat, 0);

    cout << "There are " << ans << " solutions." << endl;

    return 0;
}

```

How to initialize to zero

```

int **mat = new int*[N];
for (int i = 0; i < N; i++)
    mat[i] = new int[N]();

```

```

void nRook(int **mat, int r)
{
    // if `N` rooks are placed successfully, print the solution
    if (r == N)
    {
        printSolution(mat);
        ans++;
        return;
    }

    // place rook at every square in the current row `r`
    // and recur for each valid movement
    for (int i = 0; i < N; i++)
    {
        // if no two rooks threaten each other
        if (isSafe(mat, r, i))
        {
            // place rook on the current square
            mat[r][i] = 1;

            // recur for the next row
            nRook(mat, r + 1);

            // backtrack and remove the rook from the current square
            mat[r][i] = 0;
        }
    }
}

```

```
int isSafe(int **mat, int r, int c)
{
    // return 0 if two rooks share the same column
    for (int i = 0; i < r; i++)
    {
        if (mat[i][c] == 1) {
            return 0;
        }
    }

    return 1;
}
```

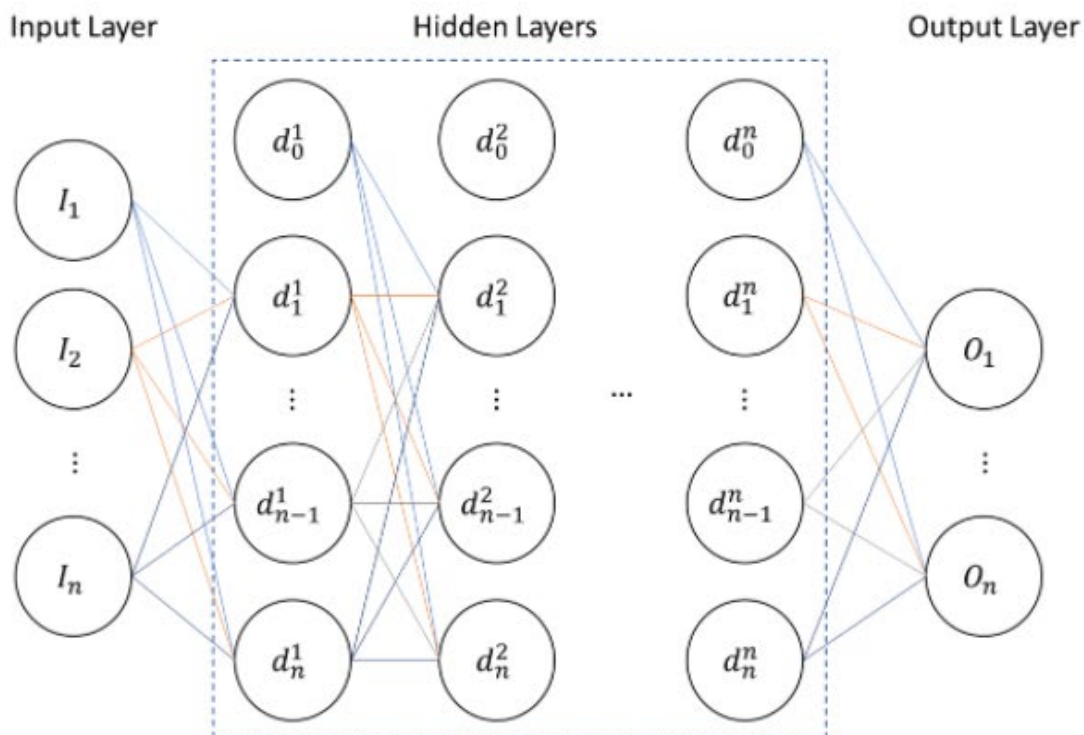
```
void printSolution(int **mat)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (mat[i][j] == 1) cout << "R";
            else cout << "#";
        }
        cout << endl;
    }
    cout << endl;
}
```

Neural Network

Problem

Description

A neural network contains three components: one input layer, one output layer, and several hidden layers. We can see the figure below as an example.



Now, you are given the number of units in the input layer, the number of units in the output layer, the total number of units in hidden layers, and the maximum number of units in each hidden layer.

Please determine the maximum number of links in the neural network.

(給定input的unit個數、所有hidden layer unit的個數、output的unit個數、每層hidden layer所能容納最多的unit數，請算出最大可能的link數量。)

Solution

- Iterate all possible solutions to determine the largest one.
- How to do that?
 - Use recursive call
 - In every hidden layer, there are two possible scenarios.
 - Suppose there are “n” units remained, “pre” units in the previous layer, “o” units in the output layer, and the maximum units number is “m” in each hidden layer.
 - The current hidden layer is the last hidden layer.
 - The number of links from the current layer to the output layer is
$$pre \times (n - 1) + n \times o$$
 - The current hidden layer is not the last hidden layer.
 - Iterate all the possible number i of units in the current layer to find the answer
 - From 2(because of the bias unit) to $\min(n - 2, m)$
 - The number of links from the current layer to the output layer is
$$pre \times (i - 1) + recursive(n - i, i, o, m)$$
 - Compare the results above and return the largest one

Solution

- Base case
 - When $n = 2$ or 3 , the current layer must be the last hidden layer because of the bias units.
 - The number of links from the current layer to the output layer is
$$pre \times (n - 1) + n \times o$$

```
5 int calculate(int n,int pre, int o, int m){
6     if( n == 2 || n == 3 ){
7         return pre * (n - 1) + n * o;
8     }
9     else{
10        int max = 0;
11        if( n <= m )
12            max = pre * (n - 1) + n * o;
13        int n_max = std::min(n-2, m);
14        for( int i = 2; i <= n_max; i++ ){
15            int temp = pre * (i - 1) + calculate(n - i, i, o, m);
16            if( temp >= max ){
17                max = temp;
18            }
19        }
20        return max;
21    }
22 }
```

```
23 int main(){
24     int n, d, o, m;
25     cin >> d >> n >> o >> m;
26     cout << calculate(n,d,o,m) << endl;
27 }
```

Solution

- Oops, TLE 😞

ID	Status	Memory	Time	Score	Real Time	Signal
1	Accepted	3MB	0ms	2	13ms	0
2	Accepted	3MB	0ms	2	10ms	0
3	Accepted	3MB	0ms	2	22ms	0
4	Accepted	3MB	1ms	2	16ms	0
5	Accepted	3MB	0ms	2	11ms	0
6	Accepted	3MB	1ms	1	23ms	0
7	Accepted	4MB	0ms	1	18ms	0
8	Accepted	3MB	190ms	1	218ms	0
9	Time Limit Exceeded	3MB	2000ms	0	2035ms	9
10	Time Limit Exceeded	3MB	1994ms	0	2029ms	9

Solution

- Think about the recursive call `calculate(4, 3, 2, 5)`.
 - How many times it would occur during the whole recursive call?
 - Can be more than 1
 - Some case can appear thousands of times
 - How about record the value of recursive call?
 - Use a table to record
 - Before calling a recursive call, check the table if the value has been calculated before.

```
5  int** table;
6  int calculate(int n,int pre, int o, int m){
7      if( n == 2 || n == 3 ){
8          return pre * (n - 1) + n * o;
9      }
10     else{
11         int max = 0;
12         if( n <= m )
13             max = pre * (n - 1) + n * o;
14         int n_max = std::min(n-2, m);
15         for( int i = 2; i <= n_max; i++ ){
16             if( table[n-i][i] == 0 )
17                 table[n-i][i] = calculate(n - i, i, o, m);
18             int temp = pre * (i - 1) + table[n-i][i];
19             if( temp >= max ){
20                 max = temp;
21             }
22         }
23         return max;
24     }
25 }
26 int main(){
27     int n, d, o, m;
28     cin >> d >> n >> o >> m;
29     table = new int*[n+1];
30     for( int i = 0; i < n + 1; i++ )
31         table[i] = new int[n+1];
32     for( int i = 0; i < n + 1; i++)
33         for( int j = 0; j < n + 1; j++)
34             table[i][j] = 0;
35     table[n][d] = calculate(n,d,o, m);
36     cout << table[n][d] << endl;
37 }
```

Solution

- AC 😊

ID	Status	Memory	Time	Score	Real Time	Signal
1	Accepted	3MB	1ms	2	14ms	0
2	Accepted	3MB	0ms	2	12ms	0
3	Accepted	3MB	1ms	2	17ms	0
4	Accepted	4MB	1ms	2	9ms	0
5	Accepted	3MB	1ms	2	27ms	0
6	Accepted	3MB	0ms	1	16ms	0
7	Accepted	3MB	1ms	1	20ms	0
8	Accepted	3MB	0ms	1	29ms	0
9	Accepted	4MB	1ms	1	21ms	0
10	Accepted	3MB	0ms	1	24ms	0

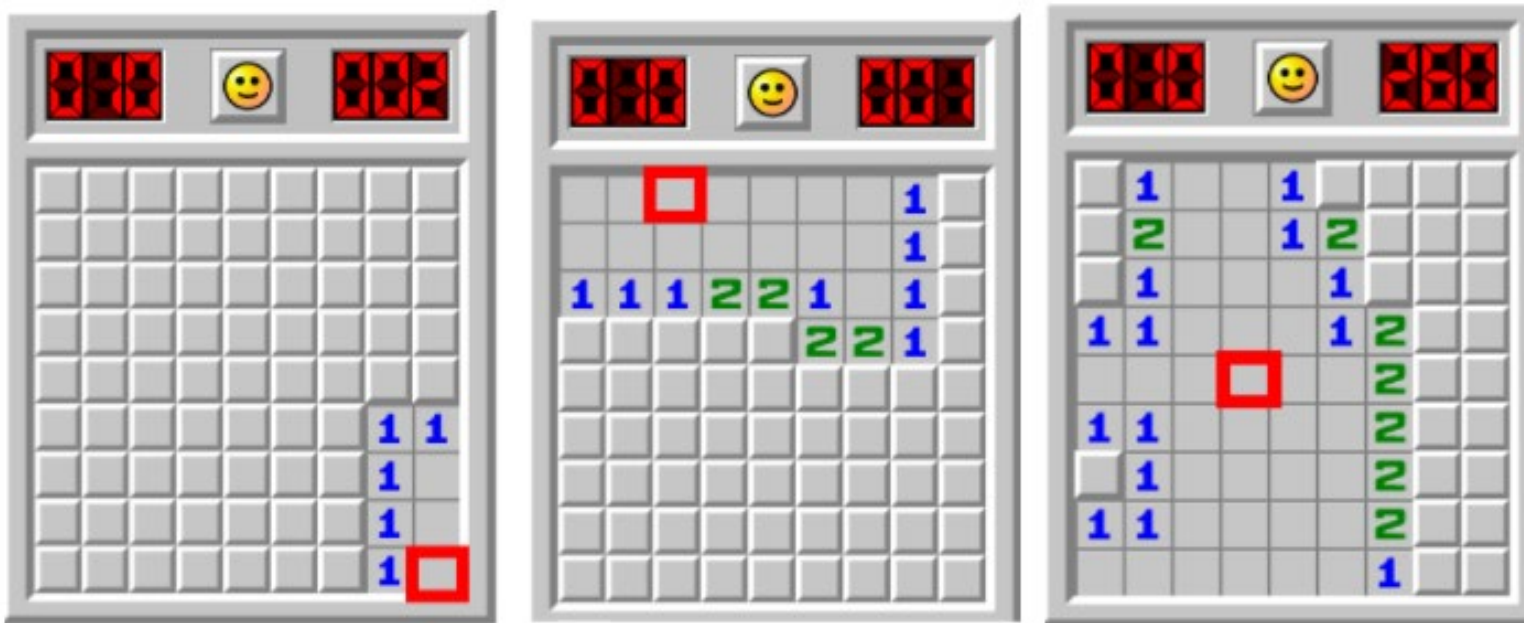
Extension

- The method of using table is called Dynamic Programming(DP)
- Check yourself if you are interested in it
- <https://zh.wikipedia.org/wiki/%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%92>

Minesweeper (v2.0)

Problem

- In a minesweeper game, when a player uncovers a mined cell, the game ends. Otherwise, the uncovered cells displays either a non-zero number, indicating the number of mines diagonally and/or adjacent to it, or a blank tile (or "0"), and **all adjacent non-mined cells will automatically be uncovered**.



Method (1)

- Make digit map as you did in the midterm-1

.
.	.	.	*	*
.	.	*	.	.
*	.	*	.	.
.	*	.	.	.



0	0	1	2	2
0	1	2	*	*
1	3	*	4	2
*	4	*	2	0
2	*	2	1	0

Solution

```
void initialize_game(Game &g, int N){
    // Initialize map and digit_map in Game with dynamic memory allocation
    // and initialize size in Game to N
    g.map = new char*[N];
    for(int i=0; i<N; i++) g.map[i] = new char[N];
    g.digit_map = new char*[N];
    for(int i=0; i<N; i++) g.digit_map[i] = new char[N];
    g.size = N;
}

void make_digit_map(Game &g){
    // Make digit map, where each digit indicates the number of mines diagonally and/or adjacent to it
    // Just as what you did in Midterm :)
    int count = 0;
    for(int i=0; i<g.size; i++){
        for(int j=0; j<g.size; j++){
            if(g.map[i][j] != '*'){
                count = 0;
                // count the number of mines adjacent to the cell
                for(int a=-1; a<=1; a++){
                    for(int b=-1; b<=1; b++){
                        if(i+a>=0 && i+a <g.size && j+b >=0 && j+b < g.size){
                            if(g.map[i+a][j+b] == '*') count++;
                        }
                    }
                }
                g.digit_map[i][j] = count + '0';
            }
            else g.digit_map[i][j] = '*';
        }
    }
}
```

Method (2)

- How to uncover adjacent blank tiles automatically ?
- **Recursive !!! Uncover adjacent blank tiles (8 directions) and do it recursively.**

0	0	1	2	2
0	1	2	*	*
1	3	*	4	2
*	4	*	2	0
2	*	2	1	0



0	0	1	.	.
0	1	2	.	.
1	3	.	.	.
.
.

Method (3)

- Recursive !!!

Base case : when the cell is not a blank tile

Uncover adjacent blank tiles (8 directions) and do it recursively.

0
.
.
.
.

0	→ 0	.	.	.
.
.
.
.

0	0	.	.	.
0	1	.	.	.
.
.
.

Method (4)

- Recursive !!!

Base case : when the cell is not a blank tile

Uncover adjacent blank tiles (8 directions) and do it recursively.

0	0	.	.	.
0
.
.
.

0	0	.	.	.
0	1	.	.	.
.
.
.

0	0	.	.	.
0	1	.	.	.
.
.
.

Method (5)

- Recursive !!!

Base case : when the cell is not a blank tile

Uncover adjacent blank tiles (8 directions) and do it recursively.

0
.
.
.
.

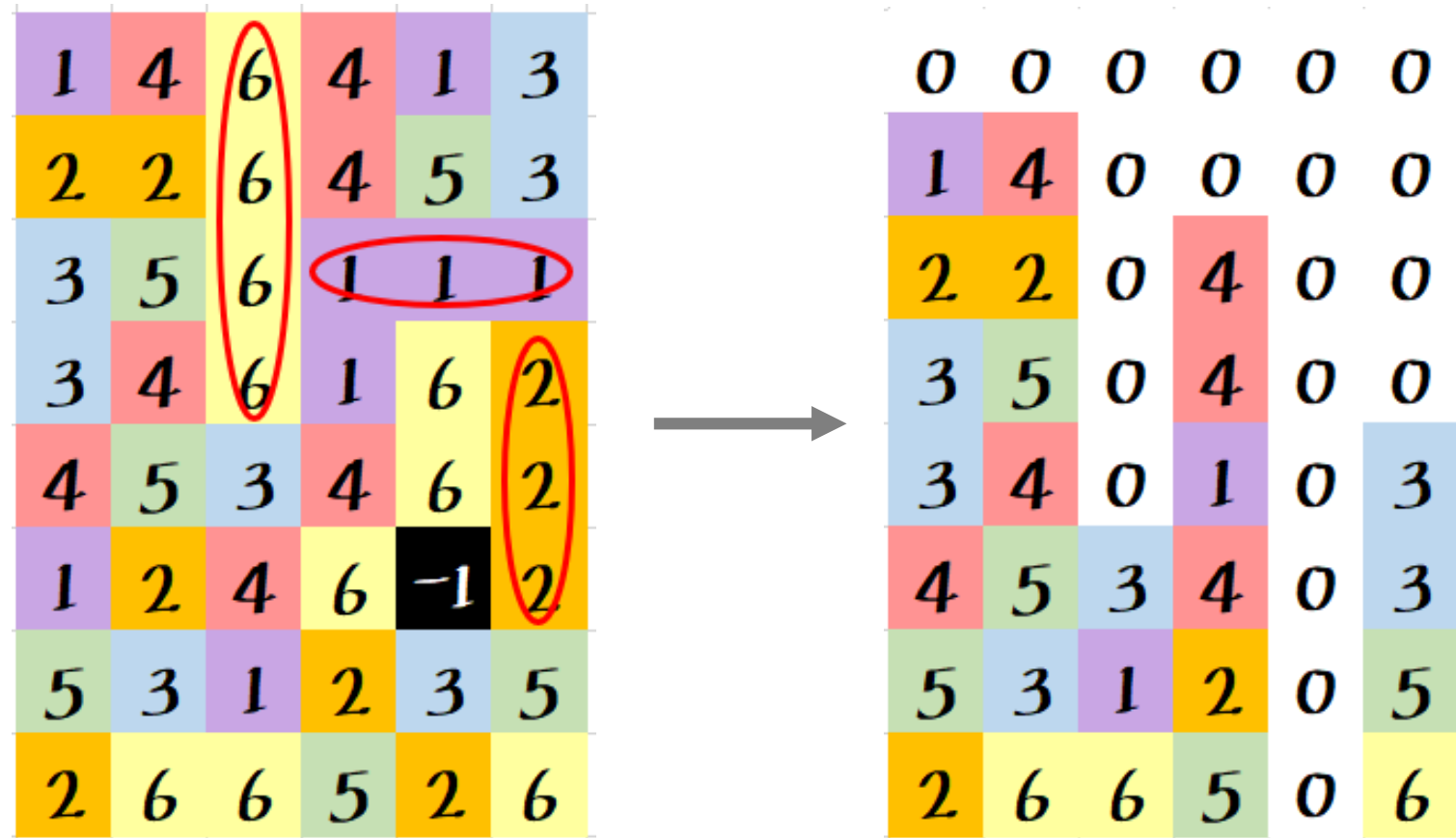
Solution

```
void update_game(Game &g, int index_x, int index_y){  
    // If the target cell is not a blank tile (i.e. digit in digit map isn't 0), just update the target cell with corresponding digit  
    if(g.digit_map[index_x][index_y] != '0') g.map[index_x][index_y] = g.digit_map[index_x][index_y]; Base case  
    // If the target cell is a blank tile, search adjacent blank tiles in 8 directions  
    else{  
        g.map[index_x][index_y] = g.digit_map[index_x][index_y];  
        if(index_x != 0){  
            // up  
            if(g.map[index_x-1][index_y] == '.') update_game(g, index_x - 1, index_y);  
        }  
        if(index_x != g.size-1){  
            // down  
            if(g.map[index_x+1][index_y] == '.') update_game(g, index_x + 1, index_y);  
        }  
        if(index_y != 0){  
            // left  
            if(g.map[index_x][index_y-1] == '.') update_game(g, index_x, index_y - 1);  
        }  
        if(index_y != g.size-1){  
            // right  
            if(g.map[index_x][index_y+1] == '.') update_game(g, index_x, index_y + 1);  
        }  
        if(index_x != 0 && index_y != 0){  
            // up-left  
            if(g.map[index_x-1][index_y-1] == '.') update_game(g, index_x - 1, index_y - 1);  
        }  
        if(index_x != g.size-1 && index_y != 0){  
            // lower-left  
            if(g.map[index_x+1][index_y-1] == '.') update_game(g, index_x + 1, index_y - 1);  
        }  
        if(index_x != 0 && index_y != g.size-1){  
            // up-right  
            if(g.map[index_x-1][index_y+1] == '.') update_game(g, index_x - 1, index_y + 1);  
        }  
        if(index_x != g.size-1 && index_y != g.size-1){  
            // lower-right  
            if(g.map[index_x+1][index_y+1] == '.') update_game(g, index_x + 1, index_y + 1);  
        }  
    }  
}
```

Recursive through 8 directions

Candy Crush

Review

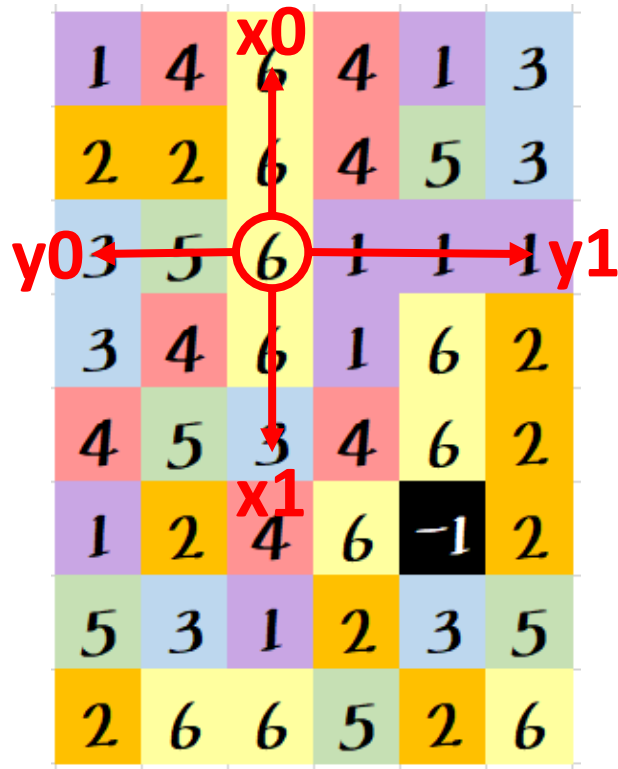


Initial

- ***del*** is used to check if there are any stones to be eliminated.
- ***boom*** is used to store the bomb position.
- ***b*** records the number of bombs
- ***clear*** records which position of the stone should be eliminated

```
17     bool del = false;
18     int boom[3][2] = {{-1, -1}, {-1, -1}, {-1, -1}}; int b=0;
19     int clear[x][y];
20     for(int i=0; i<x; i++){
21         for(int j=0; j<y; j++){
22             clear[i][j]=0;
23         }
24     }
```

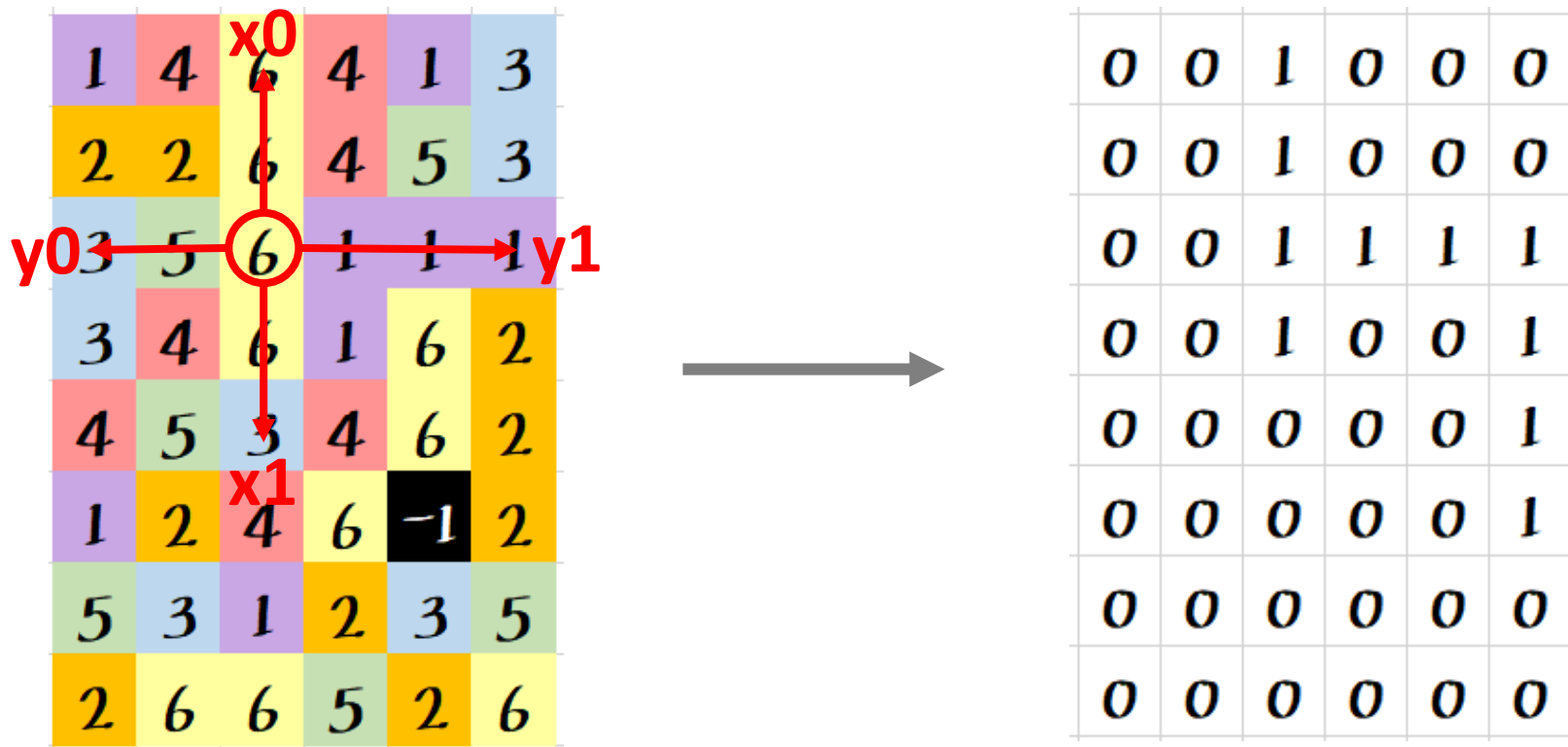
Candy Crush



- Use ***x0***, ***x1***, ***y0***, ***y1*** to check if there are the same stones on the top, bottom, left, and right.
- Assuming that the stone is located at (i, j), the inspection range only needs to be at $(i \pm 2, j \pm 2)$.
- Check every stone in the grid.
- If -1 is found, the coordinates are stored in ***boom***.

Candy Crush

- Use *clear*(x*y) to store where should be eliminated




Candy Crush

- ***del*** is used to check if there are any stones to be eliminated.

```
25     for(int i=0; i<x; i++){
26         for(int j=0; j<y; j++){
27             if(grid[i][j]==-1){
28                 boom[b][0]=i;
29                 boom[b][1]=j;
30                 b++;
31             }
32             int x0 = i, x1 = i, y0 = j, y1 = j;
33             while (x0 >= 0 && x0 > i - 3 && grid[x0][j] == grid[i][j] && grid[i][j]!=0) x0--;
34             while (x1 < x && x1 < i + 3 && grid[x1][j] == grid[i][j] && grid[i][j]!=0) x1++;
35             while (y0 >= 0 && y0 > j - 3 && grid[i][y0] == grid[i][j] && grid[i][j]!=0) y0--;
36             while (y1 < y && y1 < j + 3 && grid[i][y1] == grid[i][j] && grid[i][j]!=0) y1++;
37             if (x1 - x0 > 3 || y1 - y0 > 3) {
38                 del = true;
39                 clear[i][j] = 1;
40             }
41         }
42     }
```

Bomb

1	4	6	4	1	3
2	2	6	4	5	3
3	5	6	1	1	1
3	4	6	1	6	2
4	5	3	4	6	2
1	2	4	6	-1	2
5	3	1	2	3	5
2	6	6	5	2	6



1	4	6	4	0	3
2	2	6	4	0	3
3	5	6	1	0	1
3	4	6	1	0	2
4	5	3	4	0	2
0	0	0	0	0	0
5	3	1	2	0	5
2	6	6	5	0	6

```
7  void bomb(int **grid, int i, int j){
8      for(int a=0; a<x; a++){
9          grid[a][j]=0;
10     }
11     for(int a=0; a<y; a++){
12         grid[i][a]=0;
13     }
14 }
```

Eliminate stones

- Use *empty_x* to record where the stone should fall.

1	4	0	4	0	3
2	2	0	4	0	3
3	5	0	0	0	0
3	4	0	1	0	0
4	5	3	4	0	0
0	0	0	0	0	0
5	3	1	2	0	5
2	6	6	5	0	6

4
4

0

1

4

0

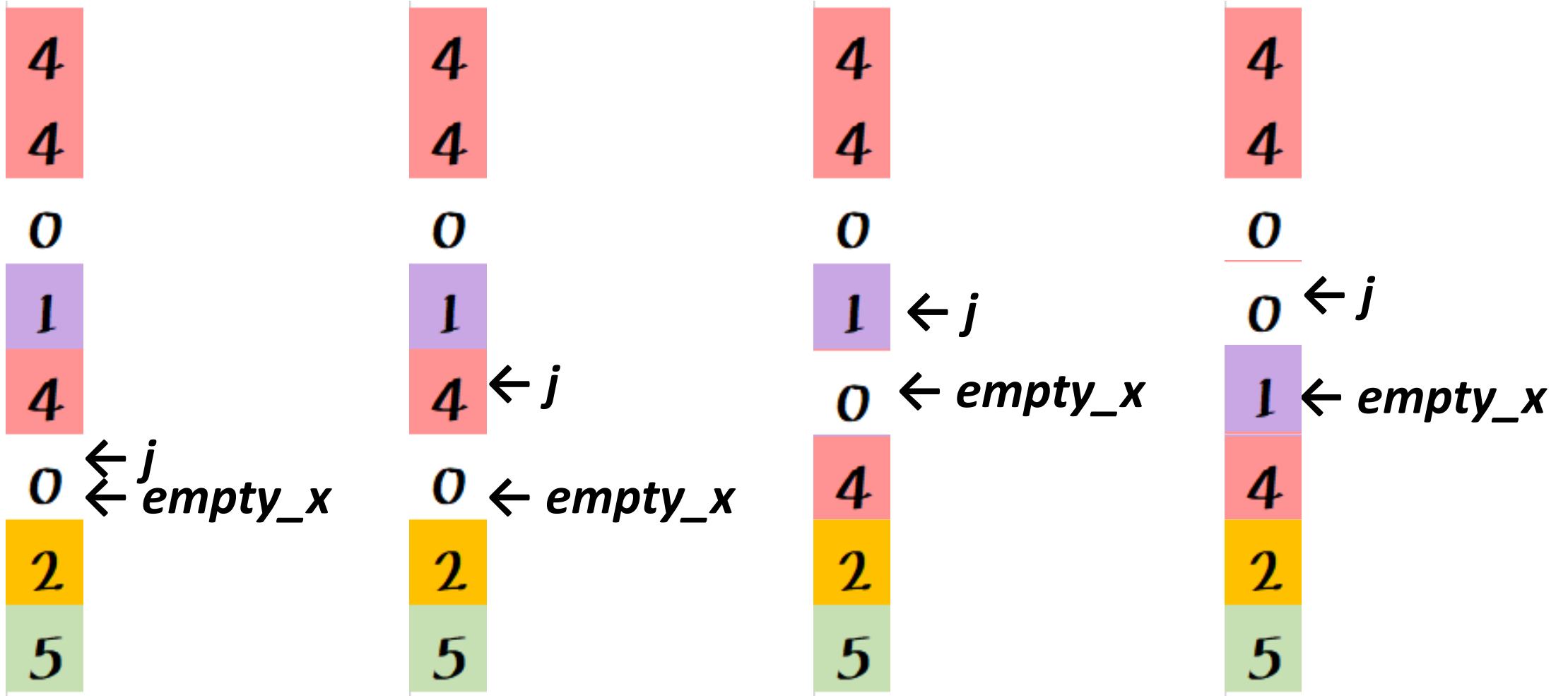
2

5

← *j*
← *empty_x*

- If (*grid[j] > 0*)
 j --
 empty_x --
- Else
 j --

Eliminate stones



Eliminate stones

```
52     for (int i = 0; i < y; ++i) {  
53         int empty_x = x - 1;  
54         for (int j = x - 1; j >= 0; --j) {  
55             if (grid[j][i] > 0) {  
56                 grid[empty_x][i] = grid[j][i];  
57                 empty_x--;  
58             }  
59         }  
60         for (int j = empty_x; j >= 0; --j) {  
61             grid[j][i] = 0;  
62         }  
63     }
```

Print Grid

```
67 void printGrid(int **grid){  
68     for(int i=0; i<x; i++){  
69         for(int j=0; j<y; j++){  
70             cout << setw(2) << grid[i][j];  
71         }  
72         cout<<endl;  
73     }  
74     cout<<endl;  
75 }
```

Complete code

```
7 void bomb(int **grid, int i, int j){
8     for(int a=0; a<x; a++){
9         grid[a][j]=0;
10    }
11    for(int a=0; a<y; a++){
12        grid[i][a]=0;
13    }
14 }
15 void candyCrush(int **grid){
16     while(true){
17         bool del = false;
18         int boom[3][2]= {{-1, -1}, {-1, -1}, {-1, -1}};int b=0;
19         int clear[x][y];
20         for(int i=0; i<x; i++){
21             for(int j=0; j<y; j++){
22                 clear[i][j]=0;
23             }
24         }
25         for(int i=0; i<x; i++){
26             for(int j=0; j<y; j++){
27                 if(grid[i][j]==-1){
28                     boom[b][0]=i;
29                     boom[b][1]=j;
30                     b++;
31                 }
32                 int x0 = i, x1 = i, y0 = j, y1 = j;
33                 while (x0 >= 0 && x0 > i - 3 && grid[x0][j] == grid[i][j] && grid[i][j]!=0) x0--;
34                 while (x1 < x && x1 < i + 3 && grid[x1][j] == grid[i][j] && grid[i][j]!=0) x1++;
35                 while (y0 >= 0 && y0 > j - 3 && grid[i][y0] == grid[i][j] && grid[i][j]!=0) y0--;
36                 while (y1 < y && y1 < j + 3 && grid[i][y1] == grid[i][j] && grid[i][j]!=0) y1++;
37                 if (x1 - x0 > 3 || y1 - y0 > 3) {
38                     del = true;
39                     clear[i][j] = 1;
40                 }
41             }
42         }
43     }
44     for(int i=0; i<x; i++){
45         for(int j=0; j<y; j++){
46             if(clear[i][j]==1) grid[i][j] = 0;
47         }
48     }
49     for(int i=0; i<3; i++){
50         if(boom[i][0]!=-1 && boom[i][1]!=-1) bomb(grid, boom[i][0], boom[i][1]);
51     }
52     for (int i = 0; i < y; ++i) {
53         int empty_x = x - 1;
54         for (int j = x - 1; j >= 0; --j) {
55             if (grid[j][i] > 0) {
56                 grid[empty_x][i] = grid[j][i];
57                 empty_x--;
58             }
59         }
60         for (int j = empty_x; j >= 0; --j) {
61             grid[j][i] = 0;
62         }
63     }
64     if(del==false) break;
65 }
66 }
67 void printGrid(int **grid){
68     for(int i=0; i<x; i++){
69         for(int j=0; j<y; j++){
70             cout << setw(2) << grid[i][j];
71         }
72         cout<<endl;
73     }
74     cout<<endl;
75 }
```