Contents

I. Thuật toán DFS-BFS 2

1. PTIT124E Họp mặt 2

2. PTIT125D Di chuyển ăn cỏ 3

3. PTIT126H Đọng nước 5

4. P133SUMB Bữa tiệc sinh nhật 7

5. BCISLAND Nước biển 9

6. BCGRASS Bãi cỏ ngon nhất 11

7. BCMUNCH Gặm cỏ 12

8. BCLKCOUN Đếm số ao 15

II. Quy hoạch động 17

1. PTIT136D Dãy số đặc biệt 17

2. PTIT138G Làm bánh rán 18

3. P132SUMG Con ếch 20

4. P134SUMA Trò chơi với xúc sắc 23

5. P136SUMA Xếp tháp 27

6. P152PROE Đếm số sách 29

7. P155PROE Điện tử số 30

I. Thuật toán DFS-BFS

1. PTIT124E Họp mặt

Có K người ($1 \le K \le 100$) đứng tại vị trí nào đó trong N địa điểm cho trước ($1 \le N \le 1,000$) được đánh số từ 1..N. Các điểm được nối với nhau bởi M đoạn đường một chiều ($1 \le M \le 10,000$) (không có đoạn đường nào nối một điểm với chính nó).

Mọi người muốn cùng tụ họp tại một địa điểm nào đó. Tuy nhiên, với các đường đi cho trước, chỉ có một số địa điểm nào đó có thể được chọn là điểm họp mặt. Cho trước K, N, M và vị trí ban đầu của K người cùng với M đường đi một chiều, hãy xác định xem có bao nhiêu điểm có thể được chọn làm điểm họp mặt.

Input

Dòng 1: Ghi 3 số: K, N, và M

Dòng 2 đến K+1: dòng i+1 chứa một số nguyên trong khoảng (1..N) cho biết địa điểm mà người thứ i đang đứng.

Dòng K+2 đến M+K+1: Mỗi dòng ghi một cặp số A và B mô tả một đoạn đường đi một chiều từ A đến B (cả hai trong khoảng 1..N và A != B).

Output

Số địa điểm có thể được chọn là điểm họp mặt.

Example

Input:
2 4 4
2
3
1 2
1 4
2 3
3 4
Output:
2
Giải thích test ví dụ: có thể họp mặt tại điểm 3 và điểm 4.

Bài làm:

```
public class PTIT124E {
    static long chuaxet[] = new long[1001];
    static long a[][] = new long[1001][1001];
    static long dem[] = new long[1001];
    static long b[] = new long[1001];
    static long k, m, n;
    static void nhap() {
        Scanner sc = new Scanner(System.in);
        k = sc.nextLong();
        n = sc.nextLong();
        m = sc.nextLong();
        for (int i = 1; i <= k; i++) {
            b[i] = sc.nextLong(); }
        for (int i = 1; i <= n; i++) {
```

```
            for (int j = 1; j <= n; j++) {
                a[i][j] = 0; } }
        for (int i = 1; i <= m; i++) {
            int ha, hi;
            ha = sc.nextInt();
            hi = sc.nextInt();
            a[ha][hi] = 1; }
        for (int i = 1; i <= n; i++) {
            dem[i] = k; } }
    static void reset() {
        for (int i = 1; i <= n; i++) {
            chuaxet[i] = 1; } }
    static void DFS(int u) {
        chuaxet[u] = 0;
        dem[u]--;
        for (int i = 1; i <= n; i++) {
            if (chuaxet[i]!=0 && a[u][i]!=0) {
                DFS(i); } } }
    public static void main(String[] args) {
        nhap();
        for (int i = 1; i <= k; i++) {
            reset();
            DFS((int) b[i]); }
        long t = 0;
        for (int i = 1; i <= n; i++) {
            if (dem[i] == 0) {
                t++; } }
        System.out.println(t); } }
```

2. PTIT125D Di chuyển ăn cỏ

Trang trại của FJ là một hình gồm 5×5 ô vuông nhỏ, ô trên cùng bên trái là ô (1,1), và dưới cùng bên phải là ô (5,5).

(1,1) (1,2) (1,3) (1,4) (1,5)
(2,1) (2,2) (2,3) (2,4) (2,5)
(3,1) (3,2) (3,3) (3,4) (3,5)
(4,1) (4,2) (4,3) (4,4) (4,5)
(5,1) (5,2) (5,3) (5,4) (5,5)

Mỗi ô đều có chứa cỏ, ngoại trừ K ô không có (0<=K<=22, K chẵn). Bò A bắt đầu ăn cỏ từ ô (1,1), bò B bắt đầu từ ô (5,5) (2 ô này luôn chứa cỏ).

Sau mỗi nửa giờ, bò A và bò B ăn hết cỏ của ô đang đứng, và di chuyển sang một ô còn có cỏ kề cạnh. Chúng muốn ăn hết tất cả cỏ và kết thúc ở cùng một ô cuối cùng. Bạn hãy tính xem có bao nhiêu cách di chuyển khác nhau thỏa mãn điều kiện trên. Biết bò A và bò B luôn luôn ăn cỏ ở trên các ô khác nhau, trừ ô kết thúc.

Input

- Dòng 1: số K
- Dòng 2..1+K: Mỗi dòng chứa vị trí của ô không có cỏ (i,j), 2 số cách nhau bởi dấu cách

Output

Số cách di chuyển thỏa mãn đề bài.

Example

Input:

4
3 2
3 3
3 4
3 1

Output:

1

Giải thích:

- Ô 'x": Các ô không chứa cỏ.
- Ô 'a','b': đường di chuyển của bò A và bò B tương ứng

```
a  a--a  a--a
|  |  |  |  |
a--a  a--a  a
         |
x  x  x  x a/b
         |
b--b--b--b--b
|
b--b--b--b--b
```

Bài làm:

```
public class PTIT125D {
    static int Map[][] = new int[7][7];
    static int cx[][] = new int[7][7];
    static int dco = 0;
    static int dem = 1;
    static int res = 0;
    static int X_[] = {-1, 0, 1, 0};
    static int Y_[] = {0, 1, 0, -1};
    static int n = 5;
    static void Try(int u, int v) {
        int x, y;
        for (int i = 0; i < 4; i++) {
            x = u + X_[i];
            y = v + Y_[i];
            if (Map[x][y] + cx[x][y] == 0) {
                cx[x][y] = 1;
                dem++;
                if (x + y == 10) {
```

```
                if (dem == dco) {
                    res++;  }
                } else {
                    Try(x, y);  }
                cx[x][y] = 0;
                dem--; } } }
        static void init() {
            int k, x, y;
            Scanner sc = new Scanner(System.in);
            k = sc.nextInt();
            for (int i = 0; i < k; i++) {
                x = sc.nextInt();
                y = sc.nextInt();
                Map[x][y] = 1;   }
            for (int i = 0; i <= n + 1; i++) {
                Map[i][0] = Map[0][i] = Map[i][n + 1] = Map[n + 1][i] = 1;   }
            dco = 25 - k;  }
        static void proce() {
            cx[1][1] = 1;
            Try(1, 1);
            System.out.println(res);  }
        public static void main(String[] args) {
            init();
 proce(); } }
```

3. PTIT126H Đọng nước

Nền phẳng của một công trường xây dựng đã được chia thành lưới ô vuông đơn vị kích thước mxn ô. Trên mỗi ô (i, j) của lưới, người ta dựng một cột bê tông hình hộp có đáy là ô (i, j) và chiều cao là Hij đơn vị. Sau khi dựng xong, thì trời đổ mưa to và đủ lâu. Giả thiết rằng nước không thẩm thấu qua các cột bê tông cũng như không rò rỉ qua các đường ghép giữa chúng.

Yêu cầu: Xác định lượng nước đọng giữa các cột

Chú ý: m, n, H_{ij} là các số nguyên dương. 1 <= m, n <= 100. 1 <= H_{ij} <= 1000

Input

Dòng 1:	m n
Dòng 2:	H_{11} H_{12} ... H_{1n}
Dòng 3:	H_{21} H_{22} ... H_{2n}
...	...
Dòng m + 1:	H_{m1} H_{m2} ... H_{mn}

Các số trên 1 dòng các nhau ít nhất 1 dấu cách

Output

Số đơn vị khối nước đọng

Example

Input:
5 7
3 3 3 3 3 3 3
3 1 1 1 1 1 3
3 1 2 2 2 1 3
3 1 1 1 1 1 3
3 3 3 3 3 3 3
Output:
27
Input:
5 5
9 9 9 9 9
9 2 2 2 9
9 2 1 2 9
9 2 2 2 9
9 9 9 9 9
Output:
64

Bài làm:

```
public class PTIT126H {
    static int n, m;
    static int h[][] = new int[125][125];
    static int mark[][] = new int[102][102];
    static ArrayList<Integer> tall = new ArrayList<>();
    static int _X[] = {-1, 0, 1, 0};
    static int _Y[] = {0, 1, 0, -1};
    static int add;
    static int MAX = 1001;
    static void init() {
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        m = sc.nextInt();
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= m; j++) {
                h[i][j] = sc.nextInt();
                tall.add(h[i][j]); } }
        tall.remove(0);
        for (int i = 0; i <= n + 1; i++) {
            h[i][0] = h[i][m + 1] = MAX;   }
        for (int j = 0; j <= m + 1; j++) {
            h[0][j] = h[n + 1][j] = MAX; } }
    static void DFS(int x, int y, int t) {
        mark[x][y] = 1;
        add += t - h[x][y];
```

```
    h[x][y] = t;
    int u, v;
    for (int i = 0; i < 4; i++) {
      u = x + _X[i];
      v = y + _Y[i];
      if (mark[u][v] == 0 && h[u][v] < t) {
        DFS(u, v, t); } } }
  static void perform() {
    int sum = 0;
    for (int k = tall.get(0); k != tall.get(tall.size() - 1); k++) {
      Arrays.fill(mark, 0);
      for (int i = 2; i < n; i++) {
        for (int j = 2; j < m; j++) {
          if (mark[i][j] != 0 && h[i][j] < k) {
            add = 0;
            DFS(i, j, k);
            for (int l = 0; l <= n; l++) {
              if (mark[l][1] + mark[l][m] != 0) {
                add = mark[i][1] = mark[i][m] = 0; } }
            for (int l = 0; l < 10; l++) {
              if (mark[1][l] + mark[n][l]!=0) {
                add = mark[1][j] = mark[n][j] = 0; } }
            sum += add; } } } }
    System.out.println(sum);    }
  public static void main(String[] args) {
    init();
    perform(); } }
```

4. P133SUMB Bữa tiệc sinh nhật

Sắp đến ngày sinh nhật của mình, Tèo định tổ chức một bữa tiệc sinh nhật thật đình đám, và mời tất cả các bạn bè của mình. Không may thay, trong lớp của Tèo vừa xảy ra một số chuyện không vui, làm các thành viên trong lớp có một số xích mích với nhau. Có hai tình trạng phổ biến như sau: bạn A rằng sẽ tham dự nếu như có bạn B tham dự cùng. Ngược lại, một số lại trong tình trạng thù giận, bạn C sẽ tham dự nếu như bạn D không tham dự.

Tèo thấy các bạn mình đang trong tình trạng căng thẳng quá, tự hỏi rằng liệu sinh nhật của mình sẽ có ai đến tham dự hay không?

Trường hợp có bạn nào đó có đi hay không mà không phụ thuộc vào ai cả, bạn đó sẽ có tham gia.

Input

Dòng đầu tiên là số lượng bộ test T (T <= 20).

Mỗi test bắt đầu bởi 2 số N và M, trong đó N là số người bạn (N <= 100) và M là số mối quan hệ.

M dòng tiếp theo, mỗi dòng chứa 3 số i, j, c.

c = 1 tức là bạn i chỉ đi nếu bạn j đi. Ngược lại, c = -1, bạn i chỉ đi khi bạn j không đi.

Output

Với mỗi test, in ra “YES” hoặc “NO” cho câu hỏi ở trên.

Example

Input:

```
2
3 3
1 2 1
2 3 1
3 1 1
3 4
1 2 1
1 3 -1
2 3 1
3 1 1
```

Output:

```
YES
NO
```

Bài làm:

```
public class P133SUMB {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int test;
        test = sc.nextInt();
        for (int u = 0; u < test; u++) {
            int N, M;
            N = sc.nextInt();
            M = sc.nextInt();
            int matrix[][] = new int[N + 1][N + 1];
            for (int i = 0; i < M; i++) {
                int a, b, c;
                a = sc.nextInt();
                b = sc.nextInt();
                c = sc.nextInt();
                matrix[a][b] = c; }
            int checknguoidi = 0;
            for (int i = 1; i <= N; i++) {
                int checknguoidicuai = 0;
                Queue<Integer> queue = new LinkedList<Integer>();
                queue.add(i);
                int check[] = new int[N + 1];
                check[i] = 1;
                while (!queue.isEmpty()) {
                    int k = queue.poll();
```

```
            for (int j = 1; j <= N; j++) {
                if (matrix[k][j] != 0) {
                    int h = matrix[k][j];
                    if (check[j] == 0) {
                        check[j] = h;
                        if (h == 1) {
                            queue.add(j);    }
                    } else {
                        if (check[j] != h) {
                            checknguoidicuai = 1;
                            break; } } } }
                if (checknguoidicuai == 1) {
                    break; } }
            if (checknguoidicuai == 0) {
                checknguoidi = 1;
                System.out.println("YES");
                break; } }
        if (checknguoidi == 0) {
            System.out.println("NO"); } } } }
```

5. BCISLAND Nước biển

Trái đất nóng lên kéo theo mực nước biển dâng. Hòn đảo nhỏ Gonnasinka thuê bạn để dự báo trước hiểm họa này. Cho trước 1 lưới tọa độ thể hiện cao độ của đảo, hãy giúp họ tính toán xem nước biển dâng cao bao nhiêu thì hòn đảo sẽ bị chia cắt.

Input

Input gồm nhiều bộ test, mỗi bộ test bao gồm:

- Dòng đầu ghi 2 số n, m là chiều dài và chiều rộng.
- Sau đó là n dòng, mỗi dòng gồm m số, mỗi số cho biết độ cao của ô đó, giá trị 0 chỉ mực nước biển. Những ô giá trị 0 dọc theo đường viền và những ô số 0 liền kề những ô này chỉ mặt biển. Những ô có giá trị 0 còn lại (được bao bọc bởi các số > 0) là đất liền bên trong đảo nhưng có độ cao ngang mặt nước biển. Hòn đảo lúc đầu chưa bị chia cắt. Số n và m không lớn hơn 100 và độ cao không lớn hơn 1000.
- Dòng cuối cùng của input chứa 2 số 0

Output

Với mỗi bộ test, in ra:

Case n: Island splits when ocean raises f feet. (Đảo bị chia khi nước biển dâng cao f feet)

Hoặc

Case n: Island never splits. (Đảo không bao giờ bị chia cắt)

Example

```
Input:
5 5
3 4 3 0 0
3 5 5 4 3
```

```
2 5 4 4 3
1 3 0 0 0
1 2 1 0 0
5 5
5 5 5 5 7
4 1 1 1 4
4 1 2 1 3
7 1 0 0 4
7 3 4 4 4
0 0
```

Output:

```
Case 1: Island never splits.
Case 2: Island splits when ocean rises 3 feet.
```

Bài làm:

```
public class BCISLAND {
    static boolean chuaxet[][];
   static final int dx[] = {-1, 1, 0, 0};
   static final int dy[] = {0, 0, -1, 1};
   static boolean test(int x, int y, int n, int m) {
      return !(x < 0 || x > n + 1 || y < 0 || y > m + 1); }
   static void DFS_L(int x, int y, int f, int a[][], int n, int m) {
      chuaxet[x][y] = true;
      for (int i = 0; i < 4; i++) {
         int xx = x + dx[i];
         int yy = y + dy[i];
         if (test(xx, yy, n, m) && !chuaxet[xx][yy] && a[xx][yy] <= f) {
            chuaxet[xx][yy] = true;
            DFS_L(xx, yy, f, a, n, m); } } }
   static void DFS(int x, int y, int a[][], int n, int m) {
      chuaxet[x][y] = true;
      for (int i = 0; i < 4; i++) {
         int xx = x + dx[i];
         int yy = y + dy[i];
         if (test(xx, yy, n, m) && !chuaxet[xx][yy]) {
            chuaxet[xx][yy] = true;
            DFS(xx, yy, a, n, m); } } }
   static boolean Check(int f, int a[][], int n, int m) {
      chuaxet = new boolean[n + 2][m + 2];
      int solt = 0;
      DFS_L(0, 0, f, a, n, m);
      for (int i = 1; i <= n; i++) {
         for (int j = 1; j <= m; j++) {
            if (!chuaxet[i][j]) {
               solt++;
```

```
                if (solt > 1) {
                    return true;    }
                DFS(i, j, a, n, m); } } }
        return false;    }
    public static void main(String[] args) {
        int n, m, a[][], gmax, gmin,count=1;
        boolean kt = false;
        Scanner in = new Scanner(System.in);
        n = in.nextInt();
        m = in.nextInt();
        gmax = 0;
        gmin = Integer.MAX_VALUE;
        while (n != 0 || m != 0) {
            a = new int[n + 2][m + 2];
            for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= m; j++) {
                    a[i][j] = in.nextInt();
                    if (a[i][j] > gmax) {
                        gmax = a[i][j]; }
                    if (a[i][j] < gmin) {
                        gmin = a[i][j]; } } }
            for (int k = gmin; k <= gmax; k++) {
                if (Check(k, a, n, m)) {
                    System.out.println("Case "+count+": Island splits when ocean raises "+k+"
feet.");
                    kt = true;
                    break; } }
            if (!kt) {
                System.out.println("Case "+count+": Island never splits.");  }
            n = in.nextInt();
            m = in.nextInt();
count++; } } }
```

6. BCGRASS Bãi cỏ ngon nhất

Bessie dự định cả ngày sẽ nhai cỏ xuân và ngắm nhìn cảnh xuân trên cánh đồng của nông dân John, cánh đồng này được chia thành các ô vuông nhỏ với R (1 <= R <= 100) hàng và C (1 <= C <= 100) cột.

Bessie ước gì có thể đếm được số khóm cỏ trên cánh đồng.

Mỗi khóm cỏ trên bản đồ được đánh dấu bằng một ký tự '#' hoặc là 2 ký tự '#' nằm kề nhau (trên đường chéo thì không phải).

Cho bản đồ của cánh đồng, hãy nói cho Bessie biết có bao nhiêu khóm cỏ trên cánh đồng.

Ví dụ như cánh đồng dưới dây với R=5 và C=6:

.#....

..#...
..#..#
...##.
.#....
Cánh đồng này có 5 khóm cỏ: một khóm ở hàng đầu tiên, một khóm tạo bởi hàng thứ 2 và thứ 3 ở cột thứ 2, một khóm là 1 ký tự nằm riêng rẽ ở hàng 3, một khóm tạo bởi cột thứ 4 và thứ 5 ở hàng 4, và một khóm cuối cùng ở hàng 5.

Dữ liệu

Dòng 1: 2 số nguyên cách nhau bởi dấu cách: R và C

Dòng 2…R+1: Dòng i+1 mô tả hàng i của cánh đồng với C ký tự, các ký tự là '#' hoặc '.' .

Kết quả

Dòng 1: Một số nguyên cho biết số lượng khóm cỏ trên cánh đồng.

Ví dụ

Dữ liệu

5 6
.#....
..#...
..#..#
...##.
.#....

Kết quả

5

Bài làm:

```
public class BCGRASS {
   public static void main(String[] args) {
      int m, n;
      String s[] = new String[100];
      Scanner sc = new Scanner(System.in);
      m = sc.nextInt();
      n = sc.nextInt();
      for (int i = 0; i < m; i++) {
         s[i] = sc.next(); }
      int dem = 0;
      for (int i = 0; i < m; i++) {
         for (int j = 0; j < n; j++) {
            if (s[i].charAt(j)=='#' &&(i==0||s[i-1].charAt(j)!='#') && (j==0||s[i].charAt(j-
1)!='#')) {
                  dem++; } } }
      System.out.println(dem); } }
```

7. BCMUNCH Gặm cỏ

Bessie rất yêu bãi cỏ của mình và thích thú chạy về chuồng bò vào giờ vắt sữa buổi tối.

Bessie đã chia đồng cỏ của mình là 1 vùng hình chữ nhật thành các ô vuông nhỏ với R (1 <= R <= 100) hàng và C (1 <= C <= 100) cột, đồng thời đánh dấu chỗ nào là cỏ và chỗ nào là đá. Bessie đứng ở vị trí R_b,C_b và muốn ăn cỏ theo cách của mình, từng ô vuông một và trở về chuồng ở ô 1,1 ; bên cạnh đó đường đi này phải là ngắn nhất.
Bessie có thể đi từ 1 ô vuông sang 4 ô vuông khác kề cạnh.
Dưới đây là một bản đồ ví dụ [với đá ('*'), cỏ ('.'), chuồng bò ('B'), và Bessie ('C') ở hàng 5, cột 6] và một bản đồ cho biết hành trình tối ưu của Bessie, đường đi được dánh dấu bằng chữ 'm'.

Bản đồ	Đường đi tối ưu
1 2 3 4 5 6 <-cột	1 2 3 4 5 6 <-cột
1 B . . . * .	1 B m m m * .
2 . . * . . .	2 . . * m m m
3 . * * . * .	3 . * * . * m
4 . . * * * .	4 . . * * * m
5 * . . * . C	5 * . . * . m

Bessie ăn được 9 ô cỏ.
Cho bản đồ, hãy tính xem có bao nhiêu ô cỏ mà Bessie sẽ ăn được trên con đường ngắn nhất trở về chuồng (tất nhiên trong chuồng không có cỏ đâu nên đừng có tính nhé)

Dữ liệu

- Dòng 1: 2 số nguyên cách nhau bởi dấu cách: R và C
- Dòng 2..R+1: Dòng i+1 mô tả dòng i với C ký tự (và không có dấu cách) như đã nói ở trên

Kết quả

- Dòng 1: Một số nguyên là số ô cỏ mà Bessie ăn được trên hành trình ngắn nhất trở về chuồng.

Ví dụ

Dữ liệu

```
5 6
B...*.
..*...
.**.*.
..***.
*..*.C
```

Kết quả

```
9
```

Bài làm:

```
public class BCMUNCH {
   static int maxn = 101;
   static int R, C;
   static boolean a[][] = new boolean[maxn][maxn];
   static int xb, yb, xc, yc;
   static String ch;
   static class Node {
```

```
        public int x, y, d;
        public Node(int x, int y, int d) {
            this.x = x;
            this.y = y;
            this.d = d; } }
    static void Init() {
        Scanner sc = new Scanner(System.in);
        R = Integer.parseInt(sc.next());
        C = Integer.parseInt(sc.next());
        for (int i = 0; i < R; i++) {
            ch = sc.next();
            for (int j = 0; j < C; j++) {
                if (ch.charAt(j) == 'B') {
                    xb = i;
                    yb = j;
                    a[i][j] = true;  }
                if (ch.charAt(j) == 'C') {
                    xc = i;
                    yc = j;
                    a[i][j] = false;  }
                if (ch.charAt(j) == '.') {
                    a[i][j] = true;  }
                if (ch.charAt(j) == '*') {
                    a[i][j] = false; } } } }
    static void Solve() {
        int res = 100000000;
        int dx[] = {1, -1, 0, 0};
        int dy[] = {0, 0, 1, -1};
        ArrayList<Node> list = new ArrayList<>();
        list.add(new Node(xc, yc, 1));
        for (int i = 0; i < list.size(); i++) {
            Node t = list.get(i);
//            list.remove(i);
            if (t.x == xb && t.y == yb) {
                res = Math.min(res, t.d);
            } else {
                for (int j = 0; j < 4; j++) {
                    int x = t.x + dx[j];
                    int y = t.y + dy[j];
                    if (x!=-1 && y!=-1 && a[x][y] && x>=0 && x< R && y >= 0 && y < C) {
                        list.add(new Node(x, y, t.d + 1));
                        a[x][y] = false; } } } }
        System.out.println(res - 1); }
```

```
public static void main(String[] args) {
   Init();
   Solve(); } }
```

8. BCLKCOUN Đếm số ao

Sau khi kết thúc OLP Tin Học SV, một số OLP-er quyết định đầu tư thuê đất để trồng rau. Mảnh đất thuê là một hình chữ nhật N x M (1<=N<=100; 1<=M<=100) ô đất hình vuông. Nhưng chỉ sau đó vài ngày, trận lụt khủng khiếp đã diễn ra làm một số ô đất bị ngập :((Mảnh đất biến thành một số các ao. Các OLP-er quyết định chuyển sang nuôi cá :D Vấn đề lại
nảy sinh, các OLP-er muốn biết mảnh đất chia thành bao nhiêu cái ao để có thể tính toán nuôi trồng hợp lý. Bạn hãy giúp các bạn ý nhé.
Chú ý: Ao là gồm một số ô đất bị ngập có chung đỉnh. Dễ nhận thấy là một ô đất có thể có tối đa 8 ô chung đỉnh.

INPUT:

* Dòng1: 2 số nguyên cách nhau bởi dấu cách: N và M
* Dòng 2..N+1: M kí tự liên tiếp nhau mỗi dòng đại diện cho 1 hàng các ô đất.

Mỗi kí tự là 'W' hoặc '.' tương ứng với ô đất đã bị ngập và ô đất vẫn còn nguyên.

OUTPUT:

* Dòng 1: 1 dòng chứa 1 số nguyên duy nhất là số ao tạo thành.

SAMPLE INPUT :

```
10 12
W........WW.
.WWW.....WWW
....WW...WW.
.........WW.
.........W..
..W......W..
.W.W.....WW.
W.W.W.....W.
.W.W......W.
..W.......W.
```

SAMPLE OUTPUT :

```
3
```

Due to recent rains, water has pooled in various places in Farmer
John's field, which is represented by a rectangle of N x M (1 <= N
<= 100; 1 <= M <= 100) squares. Each square contains either water ('W') or dry land ('.'). Farmer John would like to figure out how many ponds have formed in his field. A pond is a connected set of squares with water in them, where a square is considered adjacent to all eight of its neighbors. Given a diagram of Farmer John's field, determine how many ponds he has.

PROBLEM NAME: lkcount

INPUT FORMAT:
* Line 1: Two space-separated integers: N and M
* Lines 2..N+1: M characters per line representing one row of Farmer John's field. Each character is either 'W' or '.'. The characters do not have spaces between them.
SAMPLE INPUT (file lkcount.in):
10 12
W........WW.
.WWW.....WWW
....WW...WW.
.........WW.
.........W..
..W......W..
.W.W.....WW.
W.W.W.....W.
.W.W......W.
..W.......W.
OUTPUT FORMAT:
* Line 1: The number of ponds in Farmer John's field.
SAMPLE OUTPUT (file lkcount.out):
3
OUTPUT DETAILS:
There are three ponds: one in the upper left, one in the lower left, and one along the right side.

Bài làm:

```
public class BCLKCOUN {
    static int x[] = {-1, -1, -1, 0, 1, 1, 1, 0};
    static int y[] = {-1, 0, 1, 1, 1, 0, -1, -1};
    static String s[] = new String[100];
    static int n, m;
    static void lan(int i, int j) {
        StringBuilder sb = new StringBuilder(s[i]);
        sb.setCharAt(j, '.');
        s[i] = sb.toString();
        for (int k = 0; k < 8; k++) {
            if (s[i + x[k]].charAt(j + y[k]) == 'W') {
                lan(i + x[k], j + y[k]); } } }
    static void init() throws FileNotFoundException {
        Scanner sc = new Scanner(new File("demao.txt"));
        String word[] = sc.nextLine().split(" ");
        n = Integer.parseInt(word[0]);
        m = Integer.parseInt(word[1]);
        s[0] = "";
        s[n + 1] = "";
```

```
            for (int i = 0; i < m + 2; i++) {
                s[0] += ".";
                s[n + 1] += "."; }
            for (int i = 1; i <= n; i++) {
                s[i] = sc.nextLine();
                s[i] = "." + s[i] + "."; } }
        static int dem() {
            int d = 0;
            for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= m; j++) {
                    if (s[i].charAt(j) == 'W') {
                        d++;
                        lan(i, j); } } }
            return d; }
        public static void main(String[] args) throws FileNotFoundException {
            init();
    System.out.println(dem()); } }
```

II. Quy hoạch động

1. PTIT136D Dãy số đặc biệt

Cho trước một dãy số có N phần tử. Phần tử thứ i được gọi là phần tử đặc biệt nếu như nó bằng tổng của 3 phần tử có thứ tự nhỏ hơn nó (một phần tử có thể được sử dụng nhiều lần trong phép tính tổng này).

Nhiệm vụ của bạn là hãy tính toán xem có bao nhiêu phần tử đặc biệt trong dãy số?

Input

Dòng đầu tiên là số lượng phần tử của dãy số N <= 5000.

Dòng tiếp theo chứa N phần tử của dãy, (-100 000 <= A_i <= 100 000).

Output

In ra số lượng phần tử đặc biệt có trong dãy số đã cho.

Example

Input1:

2

1 3

Output1:

1

Input2:

6

1 2 3 5 7 10

Output2:

4

Input3:

3

-1 2 0

Output3:
1

Bài làm:

```
public class PTIT136D {
    static int n;
    static int a[] = new int[5005];
    static int md[] = new int[300005];
    static int ma[] = new int[300005];
    static void init() {
        int s;
        for (int i = 0; i <= 300000; i++) {
            ma[i] = md[i] = 5005;   }
        for (int i = 1; i < n; i++) {
            for (int j = 1; j <= i; j++) {
                s = a[i] + a[j];
                if (s < 0) {
                    ma[-s] = Math.min(ma[-s], i);
                } else {
                    md[s] = Math.min(md[s], i); } } } }
    static void process() {
        int res = 0, s;
        boolean flag;
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        for (int i = 1; i <= n; i++) {
          a[i] = sc.nextInt();   }
        init();
        for (int i = 1; i <= n; i++) {
            flag = false;
            for (int j = 1; j < i; j++) {
                s = a[i] - a[j];
                if (s < 0) {
                    if (ma[-s] < i) {
                        flag = true;
                        break;   }
                } else {
                    if (md[s] < i) {
                        flag = true;
                        break; } } }
            if (flag) {
                res++; } }
        System.out.println(res); }
    public static void main(String[] args) {
```

process(); } }

2. PTIT138G Làm bánh rán

Để chuẩn bị cho ngày Tết, Nam phải làm một số bánh rán theo yêu cầu, càng nhanh càng tốt. Có nhiều loại bánh rán, mỗi bánh cần một khoảng thời gian khác nhau để hoàn thành. Nam có ba cái chảo để có thể rán bánh cùng một lúc, mỗi cái chảo chỉ rán được một cái bánh tại một thời điểm. Giả sử ta bỏ qua thời gian để bỏ một cái bánh ra khỏi chảo và đưa một cái bánh khác vào rán, hãy giúp Nam tính xem khoảng thời gian ngắn nhất có thể để hoàn thành tất cả số lượng bánh được yêu cầu.

Input

- Gồm nhiều bộ test, mỗi bộ test viết trên một dòng, gồm một số nguyên n ($1<=n<=40$) là số lượng bánh cần làm. Tiếp theo là n số nguyên t_1 … t_n($1<=t_i<=30$) cho biết thời gian, tính theo phút, cần để hoàn thành mỗi chiếc bánh theo thứ tự từ 1 đến n.
- Đầu vào kết thúc với một dòng chứa duy nhất một số 0.

Output

- Với mỗi bộ test, in ra màn hình, trên một dòng, thời gian cần thiết ít nhất để hoàn thành tất cả các chiếc bánh, tính theo phút.

Example

Input:

```
1 30
3 15 10 20
5 6 7 8 9 10
0
```

Output:

```
30
20
15
```

Bài làm:

```
public class PTIT138G {
   static int n, s;
   static int a[] = new int[42];
   static int d[][][] = new int[42][1205][1205];
   static Scanner sc = new Scanner(System.in);
   static int init() {
      for (int i = 1; i <= n; i++) {
         for (int j = 0; j <= s; j++) {
            for (int k = 0; k <= s; k++) {
               d[i][j][k] = 0; } } }
      d[1][0][0] = d[1][a[1]][0] = d[1][0][a[1]] = 1;
      for (int i = 2; i <= n; i++) {
         for (int j = 0; j <= s; j++) {
            for (int k = 0; k <= s; k++) {
               if (d[i - 1][j][k] != 0) {
```

```
                d[i][j][k] = 1;
                if (j + a[i] <= s) {
                   d[i][j + a[i]][k] = 1;  }
                if (k + a[i] <= s) {
                   d[i][j][k + a[i]] = 1; } } } } }
   for (int i = 0; i <= s; i++) {
      for (int j = 0; j <= i; j++) {
         if (d[n][i][j] != 0 && (s - i - j) <= j) {
            return i; } } }
   return 0; }
static void process() {
   s = 0;
   for (int i = 1; i <= n; i++) {
      a[i] = sc.nextInt();
      s += a[i];  }
   System.out.println(init()); }
public static void main(String[] args) {
   while (true) {
      n = sc.nextInt();
      if (n == 0) {
         break;  }
      process(); } } }
```

3. P132SUMG Con ếch

Bep là một con ếch không bình thường. Nó sống ở một cái ao có N lá sen nổi trên mặt nước. Các lá sen được đánh dấu từ 1 tới N. Các lá sen được đánh dấu tọa độ theo hệ trục tọa độ Oxy. Điều không bình thường ở con ếch Bep là nó không thể nhảy chéo hay nhảy ngược lại, nó chỉ có thể nhảy tiến lên theo các đoạn thẳng song song với trục tọa độ. Nghĩa là con Bep chỉ có thể nhảy từ lá sen có tọa độ (x1,y1) tới lá sen có tọa độ (x2,y2) nếu:

* y2 = y1 và x2 > x1, hoặc
* x2 = x1 và y2 > y1.

Khi ở trên mỗi lá sen, Bep sẽ dùng lưỡi của mình để bắt ăn tất cả các côn trùng đang ở trên chiếc lá sen ấy.

Con Bep sẽ hấp thụ được một đơn vị năng lượng cho mỗi con côn trùng mà nó ăn được, và sử dụng K đơn vị năng lượng cho mỗi bước nhảy. Nó sẽ không thể thực hiện được bước nhảy nếu như không có đủ năng lượng.

Con ếch Bep muốn nhảy từ lá sen 1 tới lá sen N và có năng lượng lớn nhất có thể sau khi thực hiện bước nhảy cuối cùng. Ban đầu nó không có năng lượng và phải thu thập năng lượng bằng cách ăn hết số côn trùng trên lá sen số 1.

Hãy tìm cách nhảy cho con ếch Bep để nó đạt được năng lượng lớn nhất.

Input

Dòng đầu tiên chứa hai số nguyên N và K ($2 \le N \le 300\,000$, $1 \le K \le 1000$).

N dòng tiếp theo, dòng thứ i chứa 3 số nguyên X, Y, F ($0 \le X, Y \le 100\,000$, $0 \le F \le 1000$), biểu diễn lá sen thứ i có tọa độ (X,Y) và có chứa F con côn trùng. Sẽ không có hai lá sen nào trùng tọa độ.
Input luôn được đảm bảo để có đáp số.

Output

Một dòng duy nhất là năng lượng lớn nhất con ếch thu được.

Example

Test 1:
Input:
6 5
1 1 5
2 1 5
1 2 4
2 3 5
3 2 30
3 3 5
Output:
5

Test 2:
Input:
8 10
1 1 15
2 2 30
1 2 8
2 1 7
3 2 8
2 3 7
4 2 100
3 3 15
Output:
36

Test 3:
Input:
9 5
5 5 10
6 5 2
7 5 1
5 6 2
6 6 6
7 6 2
5 7 1

6 7 2
7 7 1
Output:
2

Bài làm:

```
public class P132SUMG {
   static int gtdx, gtcx, gtdy, gtcy, k;
   static int csd, csc;
   static int n;
   static int t[] = new int[300005];
   static int x[] = new int[300005];
   static int y[] = new int[300005];
   static int best[] = new int[300005];
   static int bestx[] = new int[100005];
   static int besty[] = new int[100005];
   static void nhap() {
      Scanner sc = new Scanner(System.in);
      n = sc.nextInt();
      k = sc.nextInt();
      for (int i = 1; i <= n; i++) {
         x[i] = sc.nextInt();
         y[i] = sc.nextInt();
         t[i] = sc.nextInt();  }
      gtdx = x[1];
      gtdy = y[1];
      gtcx = x[n];
      gtcy = y[n];  }
   static void doicho(int a, int b) {
      int tg;
      tg = a;
      a = b;
      b = tg;  }
   static void quickshor(int L, int H) {
      int i, j;
      int mid;
      i = L;
      j = H;
      mid = L + new Random().nextInt() % (H - L + 1);
      do {
         while ((x[i] < x[mid]) || ((x[i] == x[mid]) && (y[i] < y[mid]))) {
            i++;   }
         while ((x[j] > x[mid]) || ((x[j] == x[mid]) && (y[j] > y[mid]))) {
            j--;  }
```

```
            if (i <= j) {
               doicho(x[i], x[j]);
               doicho(y[i], y[j]);
               doicho(t[i], t[j]);
               i++;
               j--;  }
         } while (i <= j);
         if (i < H) {
            quickshor(i, H); }
         if (j > L) {
            quickshor(L, j); } }
      static int max(int a, int b) {
         if (a > b) {
            return a;  }
         return b;  }
      static void xu_li() {
         int i, c;
         quickshor(1, n);
         for (i = 1; i <= n; i++) {
            if ((x[i] == gtdx) && (y[i] == gtdy)) {
               csd = i; }
            if ((x[i] == gtcx) && (y[i] == gtcy)) {
               csc = i; } }
         for (i = 0; i <= 100000; i++) {
            bestx[i] = -1;
            besty[i] = -1; }
         bestx[x[csd]] = t[csd];
         besty[y[csd]] = t[csd];
         best[csd] = t[csd];
         for (i = csd + 1; i <= csc; i++) {
            best[i] = -1;
            if (bestx[x[i]] >= k) {
               best[i] = bestx[x[i]] - k + t[i]; }
            if (besty[y[i]] >= k) {
               best[i] = max(best[i], besty[y[i]] - k + t[i]); }
            bestx[x[i]] = max(bestx[x[i]], best[i]);
            besty[y[i]] = max(besty[y[i]], best[i]); }
         System.out.println(best[csc]); }
      public static void main(String[] args) {
         nhap();
xu_li(); } }
```

4. P134SUMA Trò chơi với xúc sắc

Tí ở nhà phải trông em bé. Tí và nó cùng chơi cá ngựa. Nhưng thằng bé còn quá nhỏ nên nó không hiểu được trò chơi, nó cứ di chuyển tiến lùi con xúc sắc trên đường đi. Vì vậy, Tí lấy một dải ô vuông thẳng hàng có độ dài là n cho thằng bé thích nghịch thế nào thì tùy, không lại hỏng mất bàn cá ngựa của mình. Mỗi bước, thằng bé chỉ di chuyển con xúc sắc đi đúng một ô, và không đi ra ngoài giới hạn. Biết ban đầu Tí để con xúc sắc ở ô ngoài cùng bên trái, hỏi có bao nhiêu cách mà sau k bước, con xúc sắc sẽ ở vị trí ngoài cùng bên phải?

Input

Dòng đầu tiên gồm 2 số n, m là độ dài của dải ô vuông và số trường hợp yêu cầu.
(2 <= n <= 10, m <= 50).
m dòng tiếp theo, mỗi dòng chứa một số nguyên k_i (k_i <= 10^9).

Output

Với mỗi trường hợp, hãy in ra số cách để sau k_i bước, con xúc sắc ở vị trí cuối cùng bên phải.
Vì đáp số có thể rất lớn nên lấy dư theo mod 1 000 000 007.

Example

Test 1:
Input:
4 4
3
4
5
15
Output:
1
0
3
377

Test 2:
Input:
2 4
1
2
3
4
Output:
1
0
1
0

Bài làm:

```
public class P134SUMA {
    static int mod = 1000000007;
```

```
static class matrix {
   int a[][] = new int[7][7];    }
static matrix[] f = new matrix[15];
static int a[][] = new int[15][10];
static void init() {
   a[2][1] = a[2][2] = a[2][3] = a[2][4] = a[2][5] = 1;
   a[3][1] = 1;
   a[3][2] = 2;
   a[3][3] = 4;
   a[3][4] = 8;
   a[3][5] = 16;
   a[4][1] = 0;
   a[4][2] = 1;
   a[4][3] = 3;
   a[4][4] = 8;
   a[4][5] = 21;
   a[5][1] = 0;
   a[5][2] = 1;
   a[5][3] = 4;
   a[5][4] = 13;
   a[5][5] = 40;
   a[6][1] = 0;
   a[6][2] = 0;
   a[6][3] = 1;
   a[6][4] = 5;
   a[6][5] = 19;
   a[7][1] = 0;
   a[7][2] = 0;
   a[7][3] = 1;
   a[7][4] = 6;
   a[7][5] = 26;
   a[8][1] = 0;
   a[8][2] = 0;
   a[8][3] = 0;
   a[8][4] = 1;
   a[8][5] = 7;
   a[9][1] = 0;
   a[9][2] = 0;
   a[9][3] = 0;
   a[9][4] = 1;
   a[9][5] = 8;
   a[10][1] = 0;
   a[10][2] = 0;
```

```
    a[10][3] = 0;
    a[10][4] = 0;
    a[10][5] = 1;
    /*for(int i = 2; i <= 10; i++){
     for(int j = 1; j <= 5; j++)printf("%4d",a[i][j]);
     printf("\n");
     }*/
    for (int i = 2; i <= 10; i++) {
        for (int j = 1; j <= 5; j++) {
            for (int k = 1; k <= 5; k++) {
                f[i].a[j][k] = 0; } }

        f[i].a[2][1] = f[i].a[3][2] = f[i].a[4][3] = f[i].a[5][4] = 1;
        f[i].a[5][5] = i - 1;  }
    f[4].a[4][5] = -1;
    f[5].a[4][5] = -3;
    f[6].a[3][5] = 1;
    f[6].a[4][5] = -6;
    f[7].a[3][5] = 4;
    f[7].a[4][5] = -10;
    f[8].a[2][5] = -1;
    f[8].a[3][5] = 10;
    f[8].a[4][5] = -15;
    f[9].a[2][5] = -5;
    f[9].a[3][5] = 20;
    f[9].a[4][5] = -21;
    f[10].a[1][5] = 1;
    f[10].a[2][5] = -15;
    f[10].a[3][5] = 35;
    f[10].a[4][5] = -28;   }
static matrix multi(matrix a, matrix b) {
    matrix p = new matrix();
    for (int i = 1; i <= 5; i++) {
        for (int j = 1; j <= 5; j++) {
            p.a[i][j] = 0;
            for (int k = 1; k <= 5; k++) {
                p.a[i][j] = (p.a[i][j] + (a.a[i][k] * b.a[k][j]) % mod) % mod; } } }
    return p;   }
static matrix mpow(matrix a, long n) {
    if (n == 1) {
        return a;   }
    matrix t = mpow(a, n / 2);
    if (n != 1) {
```

```
        return multi(multi(t, t), a);   }
    return multi(t, t);  }
public static void main(String[] args) {
    init();
    int n, q;
    matrix fo;
    int p, k, res;
    Scanner sc = new Scanner(System.in);
    n = sc.nextInt();
    q = sc.nextInt();
    while (q > 0) {
        k = sc.nextInt();
        if ((k + n) % 2 == 0) {
            System.out.println("0");
        } else {
            p = k / 2 + k % 2;
            if (p < 6) {
                System.out.println(a[n][p]);
            } else {
                fo = mpow(f[n], p - 5);
                res = fo.a[5][5] * a[n][5] + fo.a[4][5] * a[n][4]
                        + fo.a[3][5] * a[n][3] + fo.a[2][5] * a[n][2] + fo.a[1][5] * a[n][1];
                res %= mod;
                res = (res + mod * mod) % mod;
                System.out.println(res); } } } } }
```

5. P136SUMA Xếp tháp

Tèo có n viên gạch hình thang cân đánh số từ 1 tới n. Viên gạch thứ i có đáy nhỏ độ dài a_i, đáy lớn độ dài b_i và chiều cao h_i (a_i < b_i). Tèo muốn xếp chồng một số viên gạch lên nhau để tạo ra một hình tháp. Ngoại trừ đúng 1 viên gạch ở trên cùng, mỗi viên gạch khác trong tháp có đáy nhỏ chứa trọn vẹn đáy lớn của viên gạch duy nhất nằm trên (đáy lớn của viên gạch dưới cùng được đặt trên mặt đất).

Chiều cao của tháp là tổng chiều cao các viên gạch tạo thành.

Các bạn hãy giúp Tèo chọn các viên gạch để xây được tháp cao nhất có thể.

Input

Dòng đầu tiên là số nguyên dương n (n <= 200 000).

n dòng tiếp theo, mỗi dòng gồm 3 số a_i, b_i, h_i (<= 10^6).

Output

Dòng đầu tiên là chiều cao lớn nhất của tháp có thể xây được.

Dòng thứ hai liệt kê các viên gạch được lựa chọn, in ra theo thứ tự từ đáy tháp đi lên.

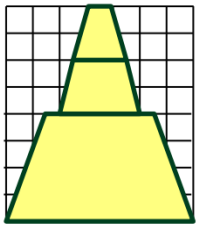
Example

Input:

6
2 3 2
4 7 4
3 5 1
1 2 2
4 5 1
5 6 1

Output:

8
2 1 4



Bài làm:

```
public class P136SUMA {
    static int n;
    static int stt[] = new int[200000];
    static int a[] = new int[200000];
    static int b[] = new int[200000];
    static int h[] = new int[200000];
    static int t[] = new int[200000];
    static int bit[] = new int[200000];
    static int d = 0;
    static int p;
    static int res = 0;
    static void qicksort(int l, int r) {
        int x = b[(l + r) / 2];
        int i = l, j = r;
        while (i <= j) {
            while (b[i] < x) {
                i++;   }
            while (b[j] > x) {
                j--;  }
            if (i <= j) {
                swap(a[i], a[j]);
                swap(b[i], b[j]);
                swap(h[i], h[j]);
                swap(t[i], t[j]);
                swap(stt[i], stt[j]);
                i++;
```

```
            j--; } }
        if (i < r) {
            qicksort(i, r);    }
        if (l < j) {
            qicksort(l, j); } }
    static void swap(int a, int b) {
        int temp = a;
        a = b;
        b = temp;    }
    static void init(int i, int v) {
        while (i <= n) {
            bit[i] = Math.max(bit[i], v);
            i += i & (-i); } }
    static int update(int i) {
        int mx = 0;
        while (i > 0) {
            mx = Math.max(mx, bit[i]);
            i -= i & (-i);    }
        return mx;    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        Stack<Integer> pos = new Stack<>();
        for (int i = 1; i < n; i++) {
            a[i] = sc.nextInt();
            b[i] = sc.nextInt();
            h[i] = sc.nextInt();
            stt[i] = i;
            t[i] = h[i];    }
        qicksort(1, n);
        for (int i = 1; i <= n; i++) {
            h[i] += update(a[i]);
            init(b[i], h[i]);
            res = Math.max(res, h[i]);  }
        for (int i = n; i > 0; i--) {
            if (h[i] == res) {
                pos.push(stt[i]);
                res -= t[i]; } }
        while (!pos.empty()) {
            System.out.print(pos.peek()+" ");
pos.pop(); } } }
```

6. P152PROE Đếm số sách

Cho 1 dãy số gồm n số nguyên a[1], a[2], ..., a[n]. Đếm số cách chia dãy thành 3 phần bằng nhau, hay nói cách khác là đếm số cặp i, j thỏa mãn:

$$\sum_{k=1}^{i-1} a_k = \sum_{k=i}^{j} a_k = \sum_{k=j+1}^{n} a_k$$

Input

Dòng đầu tiên chứa số n (1 ≤ n ≤ 5*10^5).

Dòng thứ 2 gồm n số nguyên a[1], a[2], ..., a[n] (0 <= |a[i]| <= 10^9) là các phần tử của dãy.

Output

In ra kết quả của bài toán.

Example

Test 1:

Input:

5

1 2 3 0 3

Output:

2

Test 2:

Input:

2

4 1

Output:

0

Bài làm:

```
public class P152PROE {
   static long sum[] = new long[500010];
   public static void main(String[] args) {
      Scanner sc= new Scanner(System.in);
      int n = sc.nextInt();
      for (int i = 1; i <=n; i++) {
         int n1 = sc.nextInt();
         sum[i] = sum[i-1]+n1; }
      long tong = 0;
      if(sum[n]%3==0){
         int dem=0;
         for (int i = n-1; i >0; i--) {
            if (sum[i]==sum[n]/3) {
               tong = tong+dem; }
            if(sum[i]==sum[n]/3*2){
               dem++; } } }
 System.out.println(tong); } }
```

7. P155PROE Điện tử số

Trong giờ thí nghiệm môn Điện tử số, 0xb0b0 phải thiết kế mạch để hiển thị 1 số có 2 chữ số trên 2 đèn LED 7 đoạn. Tuy nhiên, do lười nên anh ta không thiết kế được chuẩn cho lắm. Số anh ta thu được thiếu một vài nét so với số ban đầu.
Hãy giúp 0xb0b0 đếm xem có bao nhiêu số có thể khôi phục được nếu thêm 1 vài nét hoặc không thêm vào số hiện tại của anh ta.

Input

Một số n gồm 2 chữ số ($0 \le n \le 99$) là số hiện tại của 0xb0b0 trên bảng mạch. Nếu $n < 10$ thì ta nhập dưới dạng 0n.

Output

In ra một số nguyên duy nhất là đáp số của bài toán.

Example

Test 1:
Input:
89
Output:
2

Test 2:
Input:
00
Output:
4
Giải thích test 2: Các số có thể khôi phục được từ 00 là: 00, 08, 80, 88

Bài làm:

```
public class P155PROE {
   static String s[] = new String[2];
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      String num = sc.nextLine();
      s[0] = String.valueOf(num.charAt(0));
      s[1] = String.valueOf(num.charAt(1));
      Output();  }
   static int getNumber(int x) {
      switch (x) {
         case 0:
            return 2;
         case 1:
            return 7;
         case 2:
            return 2;
         case 3:
```

```
                return 3;
            case 4:
                return 3;
            case 5:
                return 4;
            case 6:
                return 2;
            case 7:
                return 5;
            case 8:
                return 1;
            case 9:
                return 2; }
        return 0;  }
    static void Output() {
        int so1, so2;
        so1 = Integer.parseInt(s[0]);
        so2 = Integer.parseInt(s[1]);
System.out.println(getNumber(so1) * getNumber(so2)); } }
```