



中國人民大學
RENMIN UNIVERSITY OF CHINA

递归练习

余力

buaayuli@ruc.edu.cn

#672. 快速排序函数

```
void quicksort( int ary[ ], int left, int right )  
{  
  
}
```

```
    if(left >= right) return;
```

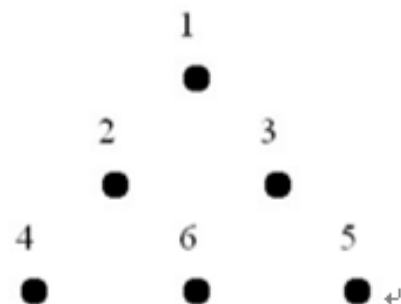
```
    int m = partition(ary, left, right);
```

```
    quicksort(ary, left, m-1);
```

```
    quicksort(ary, m+1, right);
```

#117 摘桃子

一只小猴子正在到处觅食，突然他发现一颗长满桃子的树，小猴子特别高兴就决定爬上去摘桃子吃。这颗长满桃子的树很大，共有 n 层（最高层为第 1 层），第 i 层有 i 条树枝，树的形状呈一个三角形（如图）：



图中的点表示树枝，每个点上方的数字当前这条树枝最多能摘到的桃子数（小于 100），在摘得某枝条的桃子之后，小猴子只能选择往左上方爬或者是往右上方爬（也就是说在摘了有 6 个桃子的树枝之后只能摘有 2 个桃子的树枝或是有 3 个桃子的树枝），然后继续摘桃子。小猴子现在想要从最低层开始一直爬到树顶（也就是最上面的那个枝条），摘尽可能多的桃子。请你编程帮他解决这个问题。

【输入格式】

□□ 包含 $n+1$ 行，第一行是一个整数 n ($1 \leq n \leq 100$)，表示这颗树一共有 n 层。

□□ 第 2~ $n+1$ 行中，第 i 行有 i 个用空格隔开的整数，表示树上第 i 层中每条树枝上的桃子数。

【输出格式】

□□ 共有一行，包括一个整数，表示小猴子能摘到的最多的桃子数。

int zai(int x, int y)

```
{ int topleft, topright;
```

```
  if (x==1) return a[x][y];
```

```
  else
```

```
    if(y==1)
```

```
    {  
      return a[x][y]+zai(x-1,y)  
    }
```

```
  else
```

```
  {  
    左上: a[x][y]+zai(x-1,y-1);  
    右上: a[x][y]+zai(x-1,y)  
    return max(左上、右上)  
  }
```

```
}
```

```
for(i=2;i<=tier;i++){  
  for(j=1;j<=i;j++){  
    if(a[i-1][j-1]>a[i-1][j]) num=a[i-1][j-1];  
    else num=a[i-1][j];  
    a[i][j]+=num;  
  }  
}
```

纯数组

纯递归

计算超时！！

```
int zai(int x, int y) {
```

```
int taozi[101][101];    -1
```

```
    int topleft, topright;
```

```
    if (taozi[x][y] == -1) {
```



```
    }
```

```
    return taozi[x][y];
```

```
}
```

递归+数组

#202 分书问题

有若干个人，比如 3 个人，分别叫 P1、P2、P3。↵

有若干本书，比如 3 本书，分别是 B1、B2、B3。↵

现在每个人，都有意向想看的若干本书，而有些书不想看。↵

请给出分书方案，使得每个人都满意。也就是，每个人拿到的，都是自己想看的书。↵

- ↵

【输入格式】↵

第一行，表示有几个人（意味着有几本书）。↵

其后，若干行用 0 和 1 表示，某个人对某本书是不喜好，还是喜好。有几个人就有几行。

- ↵

【输出格式】↵

输出分配方案总数。↵

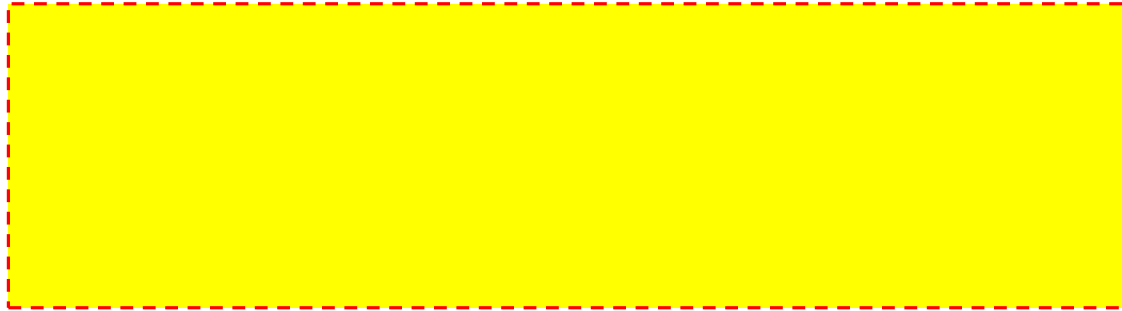
按照顺序给出所有的分配方案。↵

如果没有分配方案，输出 NO。↵

```
int Like[10][10], Fangan[10000][10] = {0}, Book[10] = {0}, Take[10];
```

```
void Try(int i) {
```

```
    if (i == n) {
```



```
    } else
```

```
        for (int j = 0; j < BNUM; j++) { // for each book  
            if (book[j] == 1) continue; // already taken  
            if (like[i][j] == 0) continue; // not like
```

```
                take[i] = j; // take the book  
                book[j] = 1; // update the flag
```

```
                Try(i+1);
```

```
                take[i] = -1; // return the book  
                book[j] = 0; // update the flag
```

```
        }
```

```
}
```

```

void input() {
    int like_value;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &like_value);
        for (int j = 0; j < n; j++) {
            Like[i][n - j - 1] = like_value % 10;
            like_value /= 10; }
        }
    }
}

```

```

void output() {
    if (count == 0) printf("NO");
    else { printf("%d\n", count);
        for (int i = 0; i < count; i++) {
            for (int j = 0; j < n; j++)
                printf("B%d", Fangan[i][j] + 1);
            printf( "\n" ); }
        }
    }
}

```

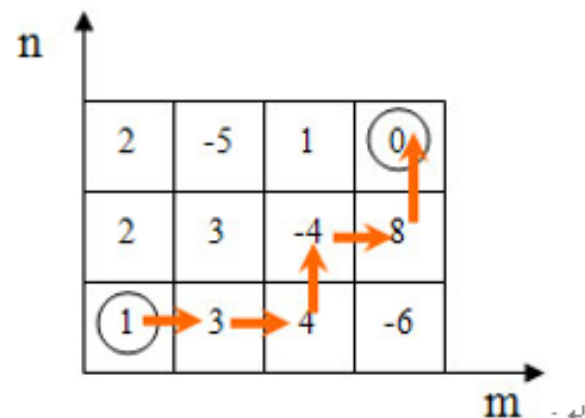
```

int main() {
    input();
    Try(0);
    output();
    return 0;
}

```


#126 Travel

有一个 $n \times m$ 的棋盘，如图所示，骑士 X 最开始站在方格(1,1)中，目的地是方格(n,m)。他的每次都只能移动到上、左、右相邻的任意一个方格。每个方格中都有一定数量的宝物 k (可能为负)，对于任意方格，骑士 X 能且只能经过最多 1 次 (因此从(1,1)点出发后就不能再回到该点了)。



你的任务是，帮助骑士 X 从(1,1)点移动到(n,m)点，且使得他获得的宝物数最多。

输入格式

□□ 输入共有 $n+1$ 行。

□□ 第一行为两个整数 n 和 m ($1 \leq n, m \leq 8$)。

□□ 接下来 n 行，每行有 m 个整数，每 2 个整数之间由一个空格分隔。第 $i+1$ 行第 j 个整数表示方格(i, j)中的宝物数目 k ($-100 \leq k \leq 100$)。

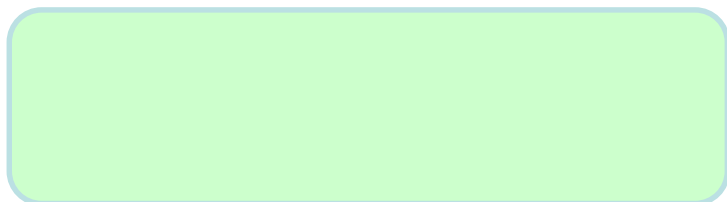
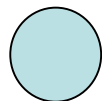
输出格式

□□ 输出数据仅一个整数，即为骑士获得的宝物数。

```
void Try(int x, int y) {
```

0、排队越界+不可用

○ if ($x < 1 \parallel x > n \parallel y < 1 \parallel y > m \parallel \text{used}[x][y]$) return;



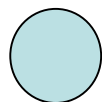
1、先用

Try(1, 1) ?

if ($x == n \ \&\& \ y == m$) // 当前探索完毕

Try(0, 0) ?

else {

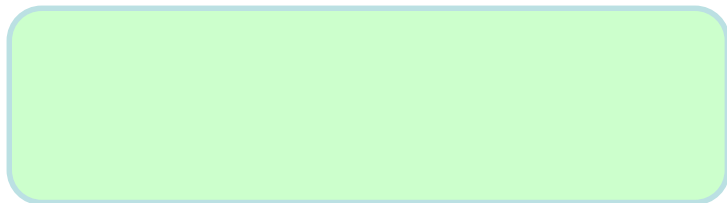
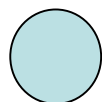


2.1、终点到站



2.2、直接探索

}



3、回溯

```
}
```

```
void Try(int x, int y) {
```

Try(1, 1)

1、先用

```
if ( x == n && y == m )
```



2.1、终点到站

```
else {
```

2.2、判断探索

3、回溯

```
}
```

```
void Try(int x, int y) {  
    // 第1步 探索  
    mine += a[x][y]; used[x][y] = 1;  
    value[step] = a[x][y]; step++;  
  
    // 第2步 判断+递归  
    if ( x == n && y == m ) {  
          
    } else {  
          
    }  
  
    // 第3步 回溯  
    mine -= a[x][y]; used[x][y] = 0; step--;  
}
```

输出最优值

#122 迷宫

给定一个由 0（表示墙壁）和 1（表示道路）的迷宫，请你判断进入迷宫后，仅通过横向和纵向的行走是否能从迷宫中走出来，即能否从坐标 (1, 1) 走到 (n, m)。

输入格式

□□共 n+1 行。

□□第一行为两个数 n, m ($1 \leq n, m \leq 9$)，表示迷宫的长和宽。

□□第 2 行到第 n+1 行，每行 m 个数，为一个由 0 和 1 组成的 n*m 的矩阵，表示迷宫，其中 0 表示墙壁（不通），1 表示道路（坐标 (1, 1) 和坐标 (n, m) 处都为 1）。

输出格式

□□仅一行。

□□若该迷宫存在从坐标 (1, 1) 到坐标 (n, m) 的由数字 1 组成的通路，则输出 YES，否则输出 NO。

输入样例

5 5

1 1 1 0 1

0 0 1 0 1

1 1 1 1 1

1 0 1 0 0

1 0 1 1 1

输出样例

YES

```
void Try(int x, int y) {
```

```
    if (x == n - 1 && y == m - 1)
```

```
        { find = 1; //判断条件
```

```
          return;    }
```

```
for (i = 0; i < R; i++)  
for (j = 0; j < C; j++)  
    search(i, j, 0);
```

不回朔

```
used[x][y] = 0; //使用 这一点标记搜索过
```

分别探索四个方向

条件：不会出界、是道路

```
if (y != 0 && used[x][y - 1] == 1)    search(x, y - 1);
```

```
if (y != m - 1 && used[x][y + 1] == 1)    search(x, y + 1);
```

```
if (x != 0 && used[x - 1][y] == 1)    search(x - 1, y);
```

```
if (x != n - 1 && used[x + 1][y] == 1)    search(x + 1, y);
```

```
}
```

#124 滑雪

小袁非常喜欢滑雪，因为滑雪很刺激。为了获得速度，滑的区域必须向下倾斜，而且当你滑到坡底，你不得不再次走上坡或者等待升降机来载你。小袁想知道在某个区域中最长的一个滑坡。区域由一个二维数组给出。数组的每个数字代表点的高度。如下：

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

一个人可以从某个点滑向上下左右相邻四个点之一，当且仅当高度减小。在上面的例子中，一条可滑行的滑坡为 24-17-16-1。当然 25-24-23-...-3-2-1 更长。事实上，这是最长的一条。你的任务就是找到最长的一条滑坡，并且将滑坡的长度输出。滑坡的长度定义为经过点的个数，例如滑坡 24-17-16-1 的长度是 4。

【输入格式】

输入的第一行表示区域的行数 R 和列数 C ($1 \leq R, C \leq 50$)。下面是 R 行，每行有 C 个整数，依次是每个点的高度 h ($0 \leq h \leq 10000$)。

【输出格式】

只有一行，为一个整数，即最长区域的长度。

```
void Try( int x, int y, int len ) {
```

```
Try(i, j, 0)
```

```
if
```

```
return;
```

```
else {
```

```
if ( a[x][y] > a[x - 1][y] )
```

```
if ( a[x][y] > a[x + 1][y] )
```

```
if ( a[x][y] > a[x][y - 1] )
```

```
if ( a[x][y] > a[x][y + 1] )
```

```
}
```

```
}
```



```
void Try(int x, int y, int len) {
```

```
Try(i, j, 0)
```

```
    if (a[x - 1][y] < a[x][y] && x - 1 > 0 )
```

```
    if (a[x + 1][y] < a[x][y] && x + 1 <= c)
```

```
    if (a[x][y - 1] < a[x][y] && y - 1 > 0 )
```

```
    if (a[x][y + 1] < a[x][y] && y + 1 <= r)
```

```
}
```

不能<=

```
void Try(int x, int y, int len) {
```

```
    int move[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
```

```
    len++;
```

```
    for (int i = 0; i < 4; i++)
```

```
    { used[x][y]++;
```

探索每一个方向

条件：不会出界、没探过、数值小

```
        nx = x + move[i][0]; ny = y + move[i][1];
```

```
        if ( nx >= 0 && nx < R && ny >= 0 && ny < C )
```

```
            if ( !used[nx][ny] && data[nx][ny] <= data[x][y] )
```

```
                search(nx, ny, len);
```

```
        if (len > max_len) max_len = len;
```

```
        used[x][y]--;
```

```
    }
```

```
    return;
```

```
}
```

可以<=

#437 寻找一卡通

时间飞逝，岁月如梭，FightAlone 慢慢长大了，可是他依旧很粗心。这天，他的一卡通又不见了。通过全球定位系统，他发现原来一卡通在 RUC 内的某个角落。但是马上就要上数分课了，他必须尽快取回一卡通，并且赶往上课地点。FightAlone 找到了擅长编程的你，希望你能够帮他设计一个最佳路线，能够让他取回一卡通，并且尽可能快地赶到上课地点(也就是走最短的路)。

输入格式

- 第一行一个数 n ，表示地图为 $n*n$ 的方阵 ($n \leq 10$)
- 以下为 n 行每行有 n 个数 A_{ij} ($0 \leq A_{ij} \leq 4$)，表示地图
- 若该数为 0，则表示为空地，可以通过。
- 若为 1，则表示为障碍物，不能通过。
- 若为 2，则表示 FightAlone 出发的东风六寝室楼。其本身为可以通过的地点。
- 若为 3，则表示一卡通遗失的地点。其本身为可以通过的地点。
- 若为 4，则表示上数分课的地点。其本身为可以通过的地点。
- FightAlone 每走一个单位长度需要一个单位时间。

输出格式

- 一个整数 t ，表示找到一卡通并赶到上课地点所需要的时间。

```
int main() {
    scanf("%d", &n);
    int i = 0, j = 0, x1, y1, x2, y2;
```

```
    while (i < n) {
        j = 0;
        while (j < n) {
            scanf("%d", &a[i][j]);
            if (a[i][j] == 2) { x1 = i; y1 = j; }
            if (a[i][j] == 3) { x2 = i; y2 = j; }
            j++;
        }
        i++;
    }
```

```
int a[11][11] = {0};
int used[11][11] = {0};
int step = 0;
int min_distance;
```

找出发点

```
    min_distance = 100;
    walk(x1, y1, 3); //到分值为3的点
    int dis1 = min_distance;
    min_distance = 100;
    walk(x2, y2, 4); //到分值为4的点
    int dis2 = min_distance;
    printf("%d", dis2 + dis1);
    return 0;
```

进行搜索

```
void walk(int x, int y, int goal) {
```

```
    if (a[x][y] == goal)
```

```
        if (step < min_distance) min_distance = step;
```

```
    else if (used[x][y] == 0) {
```

先用

探索
四个
方向

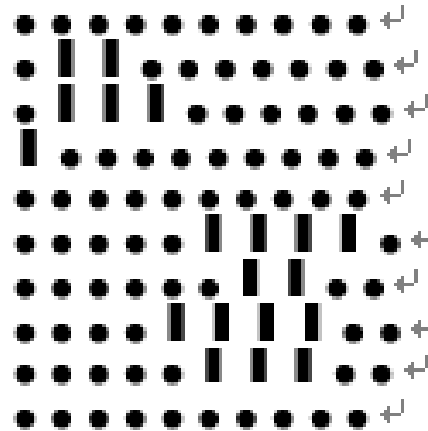
回溯

```
    }
```

```
}
```

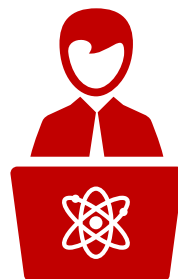
练习

- #163 最大岛屿
- #320 物体称重





中國人民大學
RENMIN UNIVERSITY OF CHINA



谢谢大家!

