程序设计实习

第二十三讲 习题与答疑

http://ai.pku.edu.cn/cpp2009
http://www.idm.pku.edu.cn/jiaoxue-CPP/2009/index.htm

主要内容

- 期末考试介绍
- B卷样例
- 中期考试试题分析
- 魔兽世界参考
- 答疑

课程考试成绩评定

- 期中考试　　15%　上机考试
- 期末考试　　50%　有B卷，考试内容为作业
- 作业　　25%　平时成绩与B卷成绩加权
  - 作业包括课堂上留的，每周两个小时的到机房上机（期中考试前），以及期中考试后的Blocks 程序对战（不必到机房，按比赛成绩给分）
- 平时成绩　　10%

期末考试

- A卷
  - C++部分（第11-20讲）的内容
  - 题型：单项选择题、写出运行结果、程序填空、编程题（补充程序）
- B卷
  - 20道题，每题5分
  - 其中18个题是来自平时作业的问题
  - 19/20题：课程反馈题

考试

- 6月19日
- 注意事项
  - 1、带好学生证，以及必要文具
  - 2、会统一分发答题纸和草稿纸
  - 3、遵守考试纪律，绝对不要作弊

B卷样例(1)

- 计算R的N次幂问题，其中0.0< R <99.999、0<N≤25。
  请问在你的程序中，使用何种数据结构保存计算结果？

- 数组

B卷样例(2)

- 在神奇的口袋问题中，要从小于40的n个整数a_1、a_2、a_3、…、a_n中凑出一组数，它们的和恰好是40。可以采用递归的方法计算出总共有多少凑法，下面的公式表示了递归求解的思路，请将公式补充完整。其中$f(a_1,a_2,\ldots,a_n,40)$表示从a_1、a_2、a_3、…、a_n中凑出和为40的方法总数。

$f(a_1,a_2,\ldots,a_n,40)= f(a_2,\ldots,a_n,\_\_\_)+ f(a_2,\ldots,a_n,\_\_\_)$

- $f(a_1,a_2,\ldots,a_n,40)= f(a_2,\ldots,a_n,40-a_1)+ f(a_2,\ldots,a_n,40)$

B卷样例(3)

- 下面的函数将一个用字符串表示的skew数转换成十进制整数，请完成函数的代码。

```
int skew2ten(char skew[]) {
    int base=1, result=0;
    for (int i=strlen(skew)-1; i>=0; i--) {
        base = base + base;
        if (skew[i]=='0') continue;
        result += _____________;
    }
    return result;
}
```

- skew[i]-'0') * (base-1);

B卷样例(4)

- 放苹果问题是一个典型的递归问题，计算将m个苹果放到n个盘子里共有多少种方法。请完成下面的递归函数代码。

```
int f(int m, int n){
   if(____||____) return 1;
   if(n>m) return ____;
   return  ____ + ____ ;
}
```

-

```
int f(int m, int n){
   if( n==1||m==0 ) return 1;
   if(n>m) return   f(m,m)   ;
   return   f(m,n-1)   +   f(m-n,n)   ;
}
```

B卷样例(5)

- 在运算符重载一讲，给了一个大整数练习，要求实现一个大整数类CHugeInteger，使得所给的程序能够正确运行。该程序中有下列一段代码：

```
cout<<"multiplying a normal integer and a huge integer:"<<251*a<<endl;
cout<<"multiplying a huge integer and a normal integer:"<<b*436<<endl;
```

其中a、b分别是一个大整数对象。请问这段代码需要哪几个运算符重载函数的支持？给出相应函数的名称、返回值类型、参数类型、是成员函数还是友员函数。

- <<：友元

  *：成员函数和友元都有要用，或者两个友元

B卷样例(6)

- 在循环数问题中，为了判断一个N(2≤N≤60)位的整数X是否是循环数，需要将X与从1到N的每个整数分别相乘。将X的数字首尾相接可以得到一个数字环；对每次相乘的结果Y，将Y的数字首尾相接也可以得到一个数字环。如果得到的全部数字环都是相同的，则X是一个循环数。请简述在程序中判断Y的数字环与X的数字环是否相同的方法。

- 判断YY是否是XXX的子串

B卷样例(7)

- 在画家问题中，有一个由N*N块砖组成的正方型，其中一些砖是黄色的、一些砖是白色的。Bob想将整个墙涂成黄色。你写了一个程序，用枚举的办法帮助Bob计算最少要画多少块砖。在N=5时需要枚举多少种情况？

- 2^5

中期考试试题分析

- Problem A: 和数
- Problem B: 因子问题
- Problem C: 围棋
- Problem D: unix纪元
- Problem E: 集合问题
- Problem F: 仙岛求药
- Problem G: 摘花生
- Problem H: Blah数集

和数(1)

- 给定一个整数序列，判断其中有多少个数，等于数列中其他两个数的和。
  - 数列1 2 3 4, 因为3 = 2 + 1, 4 = 1 + 3，所以答案为2
  - 1 1 2,答案是1/2?
- 输入数据：
  - 第一行是一个整数T，表示一共有多少组数据。 1<= T <= 100
  - 接下来的每组数据共两行，第一行是数列中数的个数n (1 <= n <= 100)，第二行是由n个整数组成的数列。
- 输出数据：
  - 对于每组数据，输出一个整数（占一行），就是数列中等于其他两个数之和的数的个数。

和数(2)

- 输入样例：

2
4
1 2 3 4
5
3 5 7 9 10

- 输出样例：

2
1

和数(3)

- 关键是要注意：一个数可能以不止一种方式被表示为另外两个数的和，以及一个数和0相加还等于自己
- 简单枚举思想
  - 三个游标i, j, k, 遍历$1 \sim N$
  - $a_i+a_j=a_k$ && $i != j != k != i$

和数(4)

```
// 参考源码的主要部分
count=0;
for (i = 0;i < n;i++)
    for (j = 0;j < n;j++)
        for (k = 0;k < n;k++)
            if ((a[i] == a[j] + a[k])
                && (j != k)
                && (i != j)
                && (i != k))
                {
                    count++;
                    break;
                }
```

因子问题

- 任给两个正整数*N*、*M*，求一个最小的正整数a，使得a和(M-a)都是N的因子。
- 简单枚举思想
  - a:1 ～ *M-1*
  - *N % a = 0 && N % (M-a) = 0*

```
// 参考源码的主要部分
for ( a = 1; a < m; a ++)
    if (n % a == 0 && n % (m - a) == 0)
    {
            flag = true;        break;
    }
```

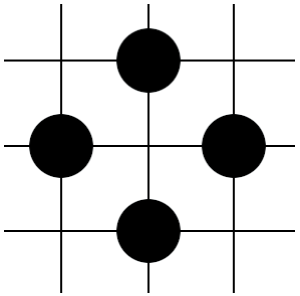
围棋 (1)

描述：

围棋的棋盘上有19*19条线交织成的361个交点，黑棋和白棋可以下在交点上。我们称这些交点为“目”。

一个目的上下左右四个方向，称之为“气”，如果一个目的四个方向都被某一种颜色的棋子占据，那么即使这个目上并没有棋子，仍然认为这个目被该颜色棋子占据。

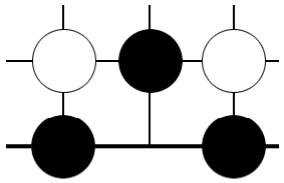
如下图中，四个黑棋中心的交点，由于被黑棋包围，因此我们认为这个目属于黑棋，



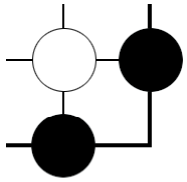
黑棋拥有4+1=5目

围棋 (2)

在棋盘的边框地区，只要占据目的三个方向，就可以拥有这个目。

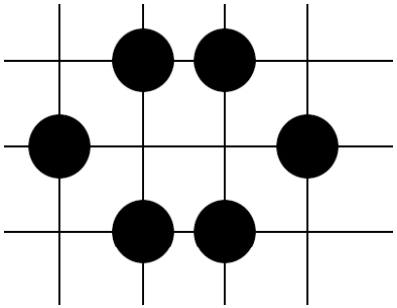


黑棋拥有3+1=4目
同理在棋盘的四个角上，只要占据目的两个气即可。



黑棋拥有2+1=3目
推而广之，当有多个目互相连通的时候，如果能用一种颜色把所有交点的气都包裹住，那么就拥有所有目。

围棋 (3)



黑棋拥有6+2 = 8目

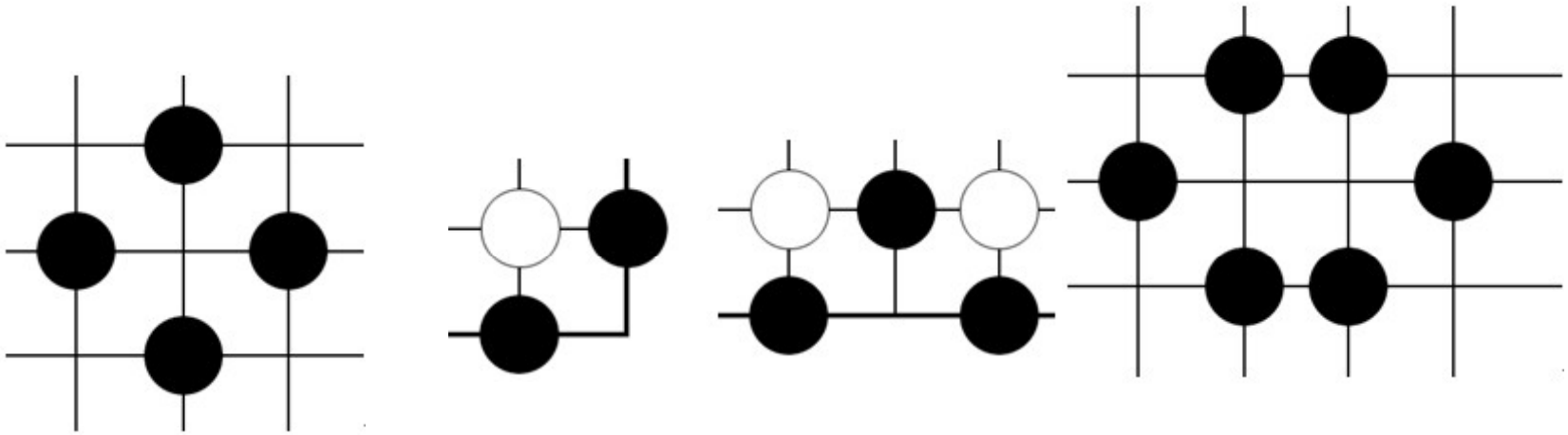
请编写一个程序，计算棋盘上黑棋和白棋的目数。
输入数据中保证所有的目，不是被黑棋包裹，就是被白棋包裹。不用考虑某些棋子按照围棋规则实际上是死的，以及互相吃（打劫），双活等情况。

输入：
第一行，只有一个整数N(1<=N<=100)，代表棋盘的尺寸是N * N
第2~n+1行，每行n个字符，代表棋盘上的棋子颜色。
“.”代表一个没有棋子的目
“B”代表黑棋
“W”代表白棋
输出：
只有一行，包含用空格分隔的两个数字，第一个数是黑棋的目数，第二个数是白棋的目数。

样例输入：
4
..BW
...B
....
....
样例输出：
15 1

围棋 (4)

- 递归游历
  - 不用考虑某些棋子按照围棋规则实际上是死的,以及互相吃，双活等情况。
  - 边界处理
    - 开设 $N+1 \times N+1$ 矩阵
    - 边界初始化为非棋子元素



从每一个黑棋出发，进行填色，最后算一遍有多少个黑棋
(是否会偏向黑？若属于双活的情况该怎么处理？)

```
#include <iostream>
using namespace std;
int n;
char Board[200][200];

void FloodFill(int i,int j)
{
        if( Board[i][j] == 'B' || Board[i][j] == '.' ) {
                Board[i][j] = 'Y'; //黑棋及其地盘填为 'Y'
                FloodFill( i +1,j);
                FloodFill( i -1,j);
                FloodFill( i ,j + 1);
                FloodFill( i ,j - 1);
        }
}
```

```
int main()
{
        int i,j,k;
        cin >> n;
        memset( Board,'W',sizeof(Board));
        for( i = 1;i <= n;i ++ )
                for( j = 1; j <= n; j ++ )
                        cin >> Board[i][j];
        for( i = 1;i <= n;i ++ ) //填色
                for( j = 1; j <= n; j ++ )
                        if( Board[i][j] == 'B' )
                                FloodFill(i,j);
        int nTotalB = 0;
        for( i = 1;i <= n;i ++ ) //统计
                for( j = 1; j <= n; j ++ )
                        if( Board[i][j] == 'Y' )
                                nTotalB ++;

        cout << nTotalB << " " << n * n - nTotalB;
        return 0;
}
```

Unix纪元

- 将1970年1月1日0点作为“unix纪元”的原点，从1970年1月1日开始经过的秒数存储为一个32位整数
- 给定一个unix时间辍，转换成形如“YYYY-mm-dd HH:ii:ss”的格式
- 日期问题

集合问题(1)

- 负荷分配
  - 集合A中当前各元素之和记为SUM(A)，称为A的负荷
  - SUM(A)与M之差的绝对值称为A的负荷与理想负荷的偏差，简称为A的偏差
  - 按照从大到小的顺序，依次为每个整数分别选择一个集合；确定一个整数所属集合时，先计算各集合的负荷，将该整数分配给负荷最小的那个集合
  - 求使得各集合的偏差之和最小的划分方案中，集合数最大的那种方案的集合数N
- 模拟题
  - 依据分配方案，逐一模拟，得到最优解

集合问题(2)

```
for n : K ~ 1  //K个数最多划分为K个集合，遍历所有可能
   for a1, a2, …, ak //对这K个数逐一分配
      for set1, set2, …, setn //每次选择最小负荷的集合分配
            choose min load setj
            send ai to setj
   calculate total deviation //计算当前分配方案的总偏差
for n : K ~ 1
    choose min total deviation //找到最小总偏差的分配方案
```

集合问题(3)

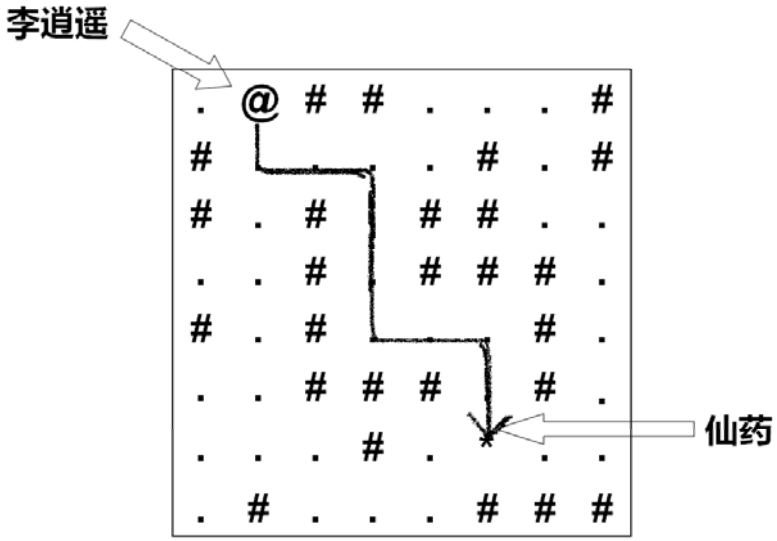
```
qsort(a,k,sizeof(int),MyCompare);
m = a[0];
int nMinSumOfDist = -1;
int nMaxN = k;
for( n = k; n >= 1; n -- )
{
    memset( aSum,0,sizeof(aSum));
    for( i = 0;i < k;i ++ )
    {
        int nMinIdx = 0;
        int nMinSum = aSum[0];
        for( j = 0; j < n; j ++ )
        {
            if( aSum[j] < nMinSum )
            {
                nMinIdx = j;
                nMinSum = aSum[j];
            }
        }
        aSum[nMinIdx] += a[i];
    }
    nSumOfDist = 0;
    for( i = 0; i < n; i ++ )
        nSumOfDist += abs(aSum[i] - m);
    if( nMinSumOfDist == -1 )
    {
        nMinSumOfDist = nSumOfDist;
        nMaxN = n;
    }
    else if( nMinSumOfDist > nSumOfDist )
    {
        nMinSumOfDist = nSumOfDist;
        nMaxN = n;
    }
```

仙岛求药

Description：

少年李逍遥的婶婶病了，王小虎介绍他去一趟仙灵岛，向仙女姐姐要仙丹救婶婶。叛逆但孝顺的李逍遥闯进了仙灵岛，克服了千险万难来到岛的中心，发现仙药摆在了迷阵的深处。迷阵由M×N个方格组成，有的方格内有可以瞬秒李逍遥的怪物，而有的方格内则是安全。现在李逍遥想尽快找到仙药，显然他应避开有怪物的方格，并经过最少的方格，而且那里会有神秘人物等待着他。现在要求你来帮助他实现这个目标。

图-1 显示了一个迷阵的样例及李逍遥找到仙药的路线.



Input：

输入有多组测试数据. 每组测试数据以两个非零整数 M 和 N 开始，两者均不大于20。M 表示迷阵行数, N 表示迷阵列数。接下来有 M 行, 每行包含N个字符,不同字符分别代表不同含义:

- ‘@’：少年李逍遥所在的位置；
- ‘.’：可以安全通行的方格；
- ‘#’：有怪物的方格；
- ‘*’：仙药所在位置。

当在一行中读入的是两个零时，表示输入结束。

Output：

对于每组测试数据，分别输出一行，该行包含李逍遥找到仙药需要穿过的最少的方格数目(计数包括初始位置的方块)。如果他不可能找到仙药, 则输出 -1 。

Sample Input:

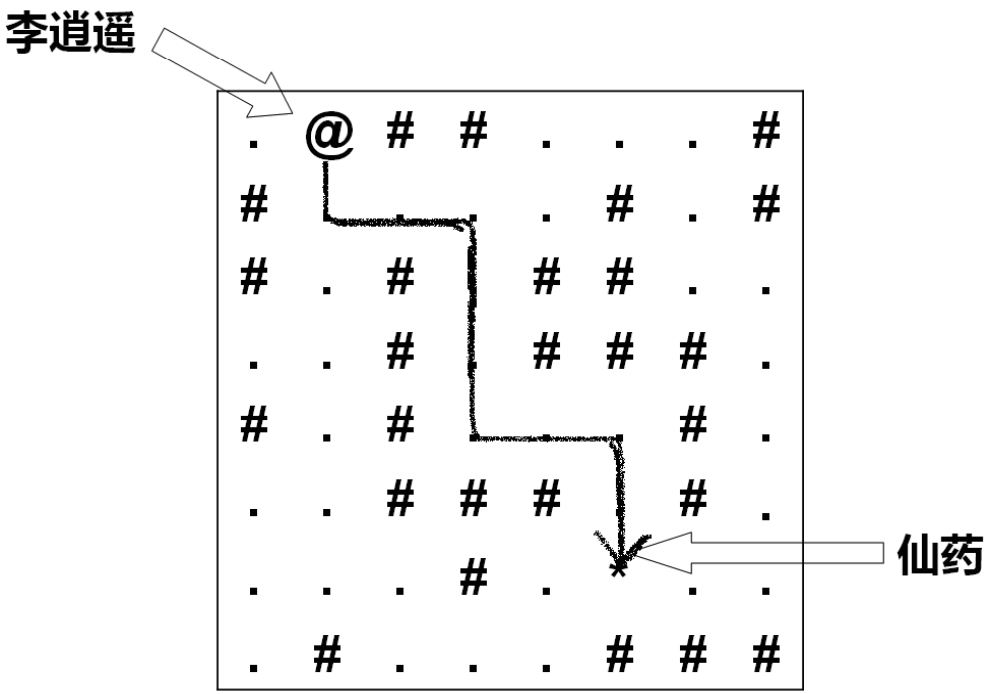
```
8 8
.@##...#
#....#.#
#.#.##..
..#.###.
#.#...#.
..###.#.
...#.*..
.#...###
6 5
.*.#.
.#...
..##.
.....
.#...
....@
9 6
.#..#.
.#.*.#
.####.
..#...
..#...
..#...
..#...
#.@.##
.#..#.
0 0
```

Sample Output:

```
10
8
-1
```

解法1

- 广度优先递归游历
  - 从出发点开始记录第一步可以到达的方格，然后第二步可以达到的方格，……，第N步可以到达的方格
  - 上下左右四个方向递归游历
  - 计算到达仙药所在方格的步数



解法1示例

```
for(i=1;i<=M;i++)
    for(j=1;j<=N;j++)
    {
        cin>>map[i][j];
        depth[i][j]=MAX_DEPTH;
        if( map[i][j] == '@')
        { x=i; y=j; map[i][j]='.'; }
        else if(map[i][j] == '*')
        { u=i; v=j; map[i][j]='.'; }
    }
depth[x][y]=0;//到达原点步数为0
search(x+1, y, 1); search(x, y+1, 1);
search(x-1, y, 1); search(x, y-1, 1);
// 输出到达目标方格所需的步数
if(depth[u][v] != MAX_DEPTH)
     cout<<depth[u][v]<<endl;
else
     cout<<-1<<endl;
```

```
int search(int i, int j, int current_depth)
{
    if( i<1 || j<1 || i>M || j>N ||
       map[i][j] == '#' )
    return 0;
    else if( current_depth < depth[i][j])
    //如果新的步数比原有步数少
    {
          depth[i][j]=current_depth;
          search(i+1, j, current_depth+1);
          search(i-1, j, current_depth+1);
          search(i, j+1, current_depth+1);
          search(i, j-1, current_depth+1);
          return 0;
     }
     else
          return 0;
}
```

```
//解法2：同样用广度优先搜索解决此题，不用递归，而用队列来实现。
//每次从队头取一个节点，看是否是终点，如果不是，就将队头节点周围的可达
//的点都放入队列。要记住每个点的上一个点是什么

#include <iostream>
using namespace std;
int M,N;
char Maze[30][30];
int nStartR,nStartC;
int nDestR,nDestC;
int nHead;
int nTail;
struct Position
{
        int r;
        int c;
        int nFather;
} Queue[4000];
```

```
int Bfs( ) {
        nHead = 0;
        nTail = 1;
        Queue[nHead].r =  nStartR;
        Queue[nHead].c =  nStartC;
        Queue[nHead].nFather =  -1;
        bool bFound = false;
        while( nHead != nTail) {
                if( Queue[nHead].r == nDestR && Queue[nHead].c == nDestC ) {
                        bFound = true;
                        break;
                }
                int nCurR = Queue[nHead].r;
                int nCurC = Queue[nHead].c;
                if( Maze[nCurR][nCurC+1] == '.' ) {
                        Queue[nTail].r = nCurR;
                        Queue[nTail].c = nCurC + 1;
                        Queue[nTail].nFather = nHead;
                        Maze[nCurR][nCurC+1] = '#';
                        nTail ++;
                }
```

```
            if( Maze[nCurR][nCurC-1] == '.' ) {
                        Queue[nTail].r = nCurR;
                        Queue[nTail].c = nCurC - 1;
                        Queue[nTail].nFather = nHead;
                        Maze[nCurR][nCurC-1] = '#';
                        nTail ++;
            }
            if( Maze[nCurR+1][nCurC] == '.' ) {
                        Queue[nTail].r = nCurR+1;
                        Queue[nTail].c = nCurC;
                        Queue[nTail].nFather = nHead;
                        Maze[nCurR+1][nCurC] = '#';
                        nTail ++;
            }
            if( Maze[nCurR-1][nCurC] == '.' ) {
                        Queue[nTail].r = nCurR-1;
                        Queue[nTail].c = nCurC ;
                        Queue[nTail].nFather = nHead;
                        Maze[nCurR-1][nCurC] = '#';
                        nTail ++;
            }
            nHead ++;
}
```

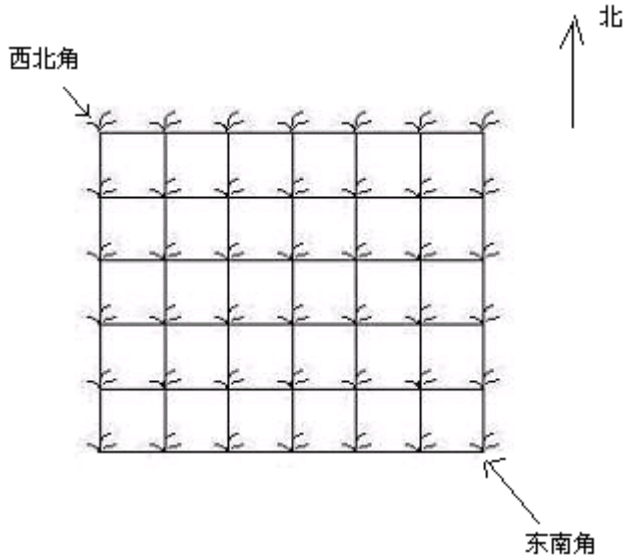
```
        if( bFound ) {
                int nTotal = 0;
                int i = nHead;
                while( i != -1 ) {
                        nTotal ++;
                        i = Queue[i].nFather;
                }
                return nTotal - 1;
        }
        else
                return -1;
}
```

```
int main()
{
        int i,j,k;
        while(1) {
                cin >> M >> N;
                if( M == 0 && N == 0 )
                        break;
                memset( Maze,'#',sizeof(Maze));
                for( i = 1;i <= M; i ++ )
                        for( j = 1; j <= N; j ++ ) {
                                cin >> Maze[i][j];
                                if( Maze[i][j] == '@' ) {
                                        nStartR = i;
                                        nStartC = j;
                                        Maze[i][j] = '.';
                                }
                                else if( Maze[i][j] == '*' ) {
                                        nDestR = i;
                                        nDestC = j;
                                        Maze[i][j] = '.';
                                }
                        }
                cout << Bfs() << endl;
        }
}
```

摘花生

Description:

Hello Kitty 想摘点花生送给她喜欢的米老鼠。她来到一片有网格状道路的矩形花生地(如下图)，从西北角进去，东南角出来。地里每个道路的交叉点上都有种着一株花生苗，上面有若干颗花生，经过一株花生苗就能摘走该它上面所有的花生。Hello Kitty只能向东或向南走，不能向西或向北走。问Hello Kitty 最多能够摘到多少颗花生。



Input:

第一行是一个整数T，代表一共有多少组数据。1<=T <= 100

接下来是T组数据。

每组数据的第一行是两个整数，分别代表花生苗的行数R和列数 C (1<= R,C <=100)

每组数据的接下来R行数据，从北向南依次描述每行花生苗的情况。每行数据有 C 个整数，按从西向东的顺序描述了该行每株花生苗上的花生数目 M (0<= M <= 1000)。

Input:

对每组输入数据，输出一行，内容为Hello Kitty能摘到得最多的花生颗数。

Sample Input:

```
2
2 2
1 1
3 4
2 3
2 3 4
1 6 5
```

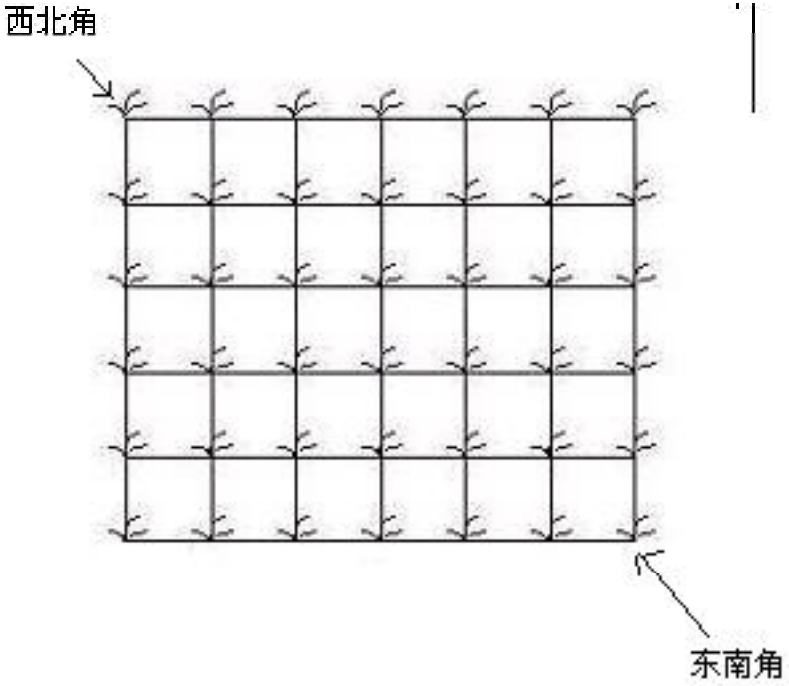
Sample Output:

```
8
16
```

摘花生(1)

- 动态规划
  - 储存子问题结果

$$answer_{i,j} = \max(answer_{i,j-1},\ answer_{i-1,j}) + value_{i,j}$$



解题思想

```
for(int i = 0; i < n; ++ i)
for(int j = 0; j < m; ++ j)
{
   if(i == 0 && j == 0)
         answer[i][j] = data[i][j];
   else if(i == 0)
         answer[i][j] = data[i][j] + answer[i][j-1];
   else if(j == 0)
         answer[i][j] = data[i][j] + answer[i-1][j];
   else
          answer[i][j] = MAX(answer[i][j-1], answer[i-1][j])
                        + data[i][j];
}
cout << answer[n-1][m-1] << endl;
```

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int Max[200][200];
int Field[200][200];
int R,C;
int FindMax(int r,int c )    {
        if( r == R -1 && c == C -1 )
                Max[r][c] = Field[r][c];
        else if( r == R -1 )
                Max[r][c] = Field[r][c] + FindMax(r,c+1);
        else if( c == C - 1 )
                Max[r][c] = Field[r][c] + FindMax(r+1,c);
        else {
                int a1 = FindMax(r,c+1);
                int a2 = FindMax(r+1,c);
                if( a1 > a2 )
                        Max[r][c] = Field[r][c] + a1;
                else
                        Max[r][c] = Field[r][c] + a2;
        }
        return Max[r][c];
}
```

```
int main()
{

        int i,j,k,t;
        cin >> t;
        while( t -- ) {
                memset( Max,-1,sizeof(Max));
                cin >> R >> C;
                for( i = 0; i < R; i ++ )
                        for ( j = 0; j < C ; j ++ ) {
                                cin >> Field[i][j];
                        }
                cout << FindMax(0,0) << endl;
        }
        return 0;
}
```

Blah数集

Description:

大数学家高斯小时候偶然间发现一种有趣的自然数集合Blah，对于以a为基的集合Ba定义如下：

(1) a是集合Ba的基，且a是Ba的第一个元素；

(2)如果x在集合Ba中，则$2x+1$和$3x+1$也都在集合Ba中；

(3)没有其他元素在集合Ba中了。

现在小高斯想知道如果将集合Ba中元素按照升序排列，第N个元素会是多少？

Input:

输入包括很多行，每行输入包括两个数字，集合的基a(1<=a<=50))以及所求元素序号n(1<=n<=1000000)

Output:

对于每个输入，输出集合Ba的第n个元素值

Sample Input	Sample Output
1 10028 5437	418900585

解题思想

- 自然数集合Blah
  - a是集合Ba的基，且a是Ba的第一个元素；
  - 如果x在集合Ba中，则2x+1和3x+1也都在集合Ba中
  - 集合Ba中元素按照升序排列求第N个元素
- 观察规律
  - a<b时，2a+1、3a+1 、2b+1与 3b+1 关系未知，需作下判断
  - 设置三个游标*k, i, j*
    - *k* 保存升序顺序
    - *i, j* 分别游历Blah数集

解法1

```
#include <stdio.h>
int data[1000001],top;

int main()
{
	int a, n, x, y;

	while(scanf("%d %d",&a, &n)!=2)
	{
		data[1]=a;
		x=y=1;
		top=2;
```

```
        while(n>=top)      {
                if(2*data[x]<3*data[y])      {
                        data[top]=2*data[x]+1;
                        x++;
                }
                else if(2*data[x]>3*data[y])  {
                        data[top]=3*data[y]+1;
                        y++;
                }
                else       {
                        data[top]=2*data[x]+1;
                        x++;       y++;
                }
                top++;
        }
        printf("%d\n",data[n]);
    }
    return 0;
}
```

解法2

```
#include <iostream> using namespace std;
int Stack[1000010];
int main(){
        int a,n;
        while( scanf("%d%d",&a,&n) != EOF) {
                int p2 = 1 ,p3 = 1;
                int nTop = 1;
                Stack[1] = a;
                while(1) {
                        if( nTop == n ) {
                                printf("%d\n",  Stack[nTop] );
                                break;
                        }

                        if( 2 * Stack[p2] + 1 < 3 * Stack[p3]+1 ) {
                                nTop ++;
                                Stack[nTop] = 2 * Stack[p2] + 1;
                                p2 ++;
                        }
```

```
                else if( 2 * Stack[p2] + 1 > 3 * Stack[p3]+1 ) {
                        nTop ++;
                        Stack[nTop] = 3 * Stack[p3] + 1;
                        p3 ++;
                }
                else {
                        nTop ++;
                        Stack[nTop] = 3 * Stack[p3] + 1;
                        p3 ++;
                        p2 ++;

                }
            }
        }
        return 0;
}
```

上机题目总结

- 题目类型
  - 枚举：考虑全面
  - 递归：递归游历，重复问题
  - 动态规划：分解子问题
  - 模拟：遵循准则
- 做题技巧
  - 先易后难
    - 参考 Ranklist 的排名
  - 解题模板
    - 总结自己的解题模板，快速应用到新题目

魔兽世界参考

- CKingdom 代表整个世界，
- CCity是城市
- CWarrior是武士的基类
  - CWarriror有成员变量 CHeadquarter * pheadquarter; 指向其所属的司令部
- CHeadquarter 有成员变量：
  - CKingdom * pKingdom; //魔兽世界指针
  - CHeadquarter * pEnemyheadquarter; //敌人司令部指针
  - 通过这些指针实现不同类对象之间的相互作用

```
class CWarrior
{
protected:
	int nStrength;
	int nForce;
	int nCityNo;
	int nId;
	CHeadquarter * pheadquarter;
public:
	virtual int Attack( CWarrior * pEnemy);
	virtual int  Hurted(CWarrior * pEnemy);
	virtual void Killed();
	void March();
};
```

```
class CIceman:public Cwarrior{
private:  int nSteps;
public:    virtual void March() ;
};
class CDragon:public CWarrior {
public:    virtual int Attack( CWarrior * p) ;
                virtual string GetName();
};
class CLion:public CWarrior{
public:    virtual int Hurted(CWarrior * pEnemy);
};
class CWolf:public CWarrior {
private:  int nEnemyKilled;
public:    virtual int Attack( CWarrior * pEnemy);
};
class CRenZhe:public CWarrior{
   virtual int  Hurted(CWarrior * pEnemy);
};
```

```
class CHeadquarter {
private:
	int nMoney;
	int nEnemys;
	int nWarriorNo;
	list<CWarrior * > lstWarrior; //武士的列表
	int nColor;
	CKingdom * pKingdom; //魔兽世界指针
	CHeadquarter * pEnemyheadquarter; //敌人司令部指针
public:
	void WarriorBorn();
	void HWarriorWin( CWarrior * pWarrior);
	void WarriorsMarch(int nEnemyHeadquterCityNo );
	void WarriorsAttack( );
	void EnemyReach();
	void WarriorsGetMoney(int);
	void WarriorKilled(CWarrior * p);
	void EarnMoney( CWarrior * p, int nCityNo);
};
```

```
class CCity {
private:
	int nFlagColor;
	int nNo;
	int nLastWinWarriorColor;
	int nMoney;
public:
	bool CWarriorWin( int nColor);  //返回是否要升旗子
	void SetFlagColor( int nColor);

};
```

```
class CEvent{
private:
	EEventType eEventType;
	int nTime;
	int nCityNo;
	int nColor;
	string sDescribe;
};
```

```
class CKingdom {
private:
	CHeadquarter Red, Blue;
	int nTimeInMinutes;
	vector<CEvent> vEvent;
	vector<CCity> vCity;
	int nEndTime;
	int nCityNum;
public:
	void KWarriorWin( CWarrior * pWarrior)
	int TimePass(int nMinutes) {
		int i;
		nTimeInMinutes = nMinutes;
		if( nTimeInMinutes > nEndTime )   return 0;
		int nRemain = nTimeInMinutes % 60;
```

```
 switch( nRemain) {
        case 0: //生产怪物
           Red.WarriorBorn();
           Blue.WarriorBorn();
           break;
        case 10: //前进
           Red.WarriorsMarch(nCityNum + 1);
           Blue.WarriorsMarch(0);
           break;
        case 20: //城市出产生命元
           for( i = 0;i < vCity.size();i ++ )
                    vCity[i].AddMoney();
           break;
        case 30: //挣钱
           Red.WarriorsGetMoney(nCityNum+1);
           Blue.WarriorsGetMoney(nCityNum+1);
           break;
        case 40://发生战斗
           Red.WarriorsAttack();
           Blue.WarriorsAttack();
           break;
        case 50:
           break;
       }
       return 1;
     } //end function TimePass
}; //end class CKingdom
```

```
int main()
{
	int N,T;
	cin >> INITMONEY >> N >> T ; //N is city Num
	cin >> STRENGTH_DRAGON
	    >> STRENGTH_RENZHE
	    >> STRENGTH_ICEMAN
	    >> STRENGTH_LION
	    >> STRENGTH_WOLF;
	cin >> FORCE_DRAGON
	    >> FORCE_RENZHE
	    >> FORCE_ICEMAN
	    >> FORCE_LION
	    >> FORCE_WOLF;

	CKingdom MyKingdom(N);
	for( int t = 0; t < T; t += 10 ) {
	    if( MyKingdom.TimePass(t) == 0)
	        break;
	}
	MyKingdom.OutputResult();
	return 0;
}
```

样例输入:war1.in

150 4 1000
90 20 30 10 20
20 50 20 40 30

详细的输入输出样例请参见课程网站

样例输入:war1.out

000:00 red iceman 1 born
000:00 blue lion 1 born
000:10 red iceman 1 marched to city 1
000:10 blue lion 1 marched to city 4
000:30 red iceman 1 earned 10 elements for his headquarter
000:30 blue lion 1 earned 10 elements for his headquarter
001:00 red lion 2 born
001:00 blue dragon 2 born
001:10 red lion 2 marched to city 1
001:10 red iceman 1 marched to city 2
001:10 blue lion 1 marched to city 3
001:10 blue dragon 2 marched to city 4
001:30 red lion 2 earned 10 elements for his headquarter
001:30 red iceman 1 earned 20 elements for his headquarter
001:30 blue lion 1 earned 20 elements for his headquarter
001:30 blue dragon 2 earned 10 elements for his headquarter
002:00 red wolf 3 born
002:00 blue renzhe 3 born
002:10 red wolf 3 marched to city 1
002:10 red lion 2 marched to city 2
002:10 blue lion 1 marched to city 2
002:10 red iceman 1 marched to city 3

National Engineering Laboratory for Video Technology

欢迎到数字媒体所(视频编解码技术国家工程实验室)参与本科生科研项目

Q&A

yhtian@pku.edu.cn
理科2号楼2641
62754541