# **ACM程序设计**

## 东北林业大学 陈宇

## Lg\_chenyu@yahoo.com.cn

# **第一讲**

## **算法原理和ACM入门**

## (Introduction to ACM)

# 我校的ACM在线评测系统

## acm.nefu.edu.cn

## 课件下载地址：

## acm.nefu.edu.cn/kj/suanfa01.ppt

# **预期赛事（今后每年）**

## 3~4月，举行校内大赛（暨选拔赛）

## 4月， ACM全国邀请赛

## 5月，参加黑龙江省大学生程序设计大赛

## 6月，参加东北4省大学生程序设计大赛

## 10~11月，参加ACM/ICPC亚洲区比赛（至少参加4～5个赛区的比赛）

## 另外，每学期至少有三次月赛以及适当的练习赛

# **第一部分 算法概述**

## 算法分析（Algorithm Analysis）：对算法所需要的两种计算机资源——时间和空间进行估算；

# **时间复杂性（Time Complexity）**

# **空间复杂性（Space Complexity）**

# **算法分析的目的：**

# **设计算法——设计出复杂性尽可能低的算法**

# **选择算法——在多种算法中选择其中复杂性最低者**

# **算法的描述语言：**

# ⑴ 自然语言

# 优点：容易理解

# 缺点：冗长、二义性

# 使用方法：粗线条描述算法思想

# 注意事项：避免写成自然段

# （2）流程图

# 优点：流程直观

# 缺点：缺少严密性、灵活性

# 使用方法：描述简单算法

# 注意事项：注意抽象层次

# ⑶ 程序设计语言

# 优点：能由计算机执行

# 缺点：抽象性差，对语言要求高

# 使用方法：算法需要验证

# 注意事项：将算法写成子函数

# （4）伪代码——算法语言

# 伪代码（Pseudocode）：介于自然语言和程序设计语言之间的方法，它采用某一程序设计语言的基本语法，操作指令可以结合自然语言来设计。

# 优点：表达能力强，抽象性强，容易理解

# 评价算法

#### 评价算法的三条主要**标准**是：

#### (1) 算法实现所耗费的**时间**；

#### (2) 算法实现所所耗费的存储**空间**，其中

#### 主要考虑辅助存储空间；

#### (3) 算法应易于**理解**，易于编码，易于调

#### 试等等。

# **和算法执行时间相关的因素**：

## 1）问题中数据存储的数据结构

#### 2）算法采用的数学模型

#### 3）算法设计的策略 4）问题的规模 5）实现算法的程序设计语言 6）编译算法产生的机器代码的质量 7）计算机执行指令的速度

# **算法效率的衡量方法**

## 通常有两种衡量算法效率的方法:

#### 1）事后统计法（有缺点，较少使用）

#### 2）事前分析估算法

## 算法的时间效率是问题规模的函数。假如,随着问题规模n的增长,算法执行时间的增长率和f(n)的增长率相同,则可记作:**T(n)=Ο(f(n))**,称T(n)为算法的**渐近时间复杂度**(Asymptotic Time Complexity),简称**时间复杂度**。Ο是数量级的符号。

#### 一个算法中**所有语句的频度之和**构成了该算法的运行时间。

#### 例如： for(j=1;j<=n;++j)

##### for(k=1;k<=n;++k)

##### ++x；

##### 

#### 语句“++x、k<=n、++k”的频度是n2，

#### 语句“ j=1、k=1”的频度是1，

#### 语句“j<=n;++j”的频度是n。

#### 算法运行时间为：3\*n2+2n+2。

# 再看看这个代码：

## 当一个算法的算法运行时间为n2+n+1，由于n2+n+1与n2的数量级相等(该表达式当n足够大时约等于n2), 我们说这个**算法的渐进时间复杂度**(简称**算法的时间复杂度**)为：T(n)=O(n2)。

# 算法(**渐进**)时间复杂度,一般均表示为以下几种数量级的形式(n为问题的规模,c为一常量)：

#### Ο(1)称为常数级

#### Ο(logn)称为对数级

#### Ο(n)称为线性级

#### Ο(n**c**)称为多项式级

#### Ο(c**n**)称为指数级

#### Ο(n!)称为阶乘级

## Temp=i；i=j；j=temp；

## 以上三条单个语句的频度均为1，该算法段的执行时间是一个与问题规模n无关的常数。算法的时间复杂度为常数阶，记作T(n)=Ο(1)。

## 如果算法的执行时间不随着问题规模n的增加而增长，即使算法中有上千条语句,其执行时间也不过是一个较大的常数。此类算法的时间复杂度是Ο(1)。

# **【例2】变量计数之一。**

## (1) x=0；=0；

## (2) for(k-1；<=n；++)

## (3) x++；

## (4) for(i=1；<=n；++)

## (5) for(j=1；j<=n；++)

## (6) y++；

## 该算法段的时间复杂度为T(n)=Ο(n**2**)。

## 当有若干个循环语句时，算法的时间复杂度是由嵌套层数最多的循环语句中最内层语句的频度f(n)决定的。

# **【例3】变量计数之二**

## (1) x=1；

## (2) for(i=1；i<=n；i++)

## (3) for(j=1；j<=i；j++)

## (4) for(k=1；k<=j；k++)

## (5) x++；

## 该算法段中频度最大的语句是(5)，从内层循环向外层分析语句(5)的执行次数：

## 复杂度：O(n3)

# **第二部分**

# 先看这道题

## The Hardest Problem Ever hdu1048

## Acm.nefu.edu.cm 第60题

# 【问题描述】

## Julius Caesar生活在一个危险而又充斥着阴谋的时代。Caesar面对的最难的情况关系着他的存亡。为了让自己生存，他决心去创造第一种加密方法之一。这个加密方法听起来是这样的令人难以置信，没有一个人可以指出它（的原文）除非知道它怎样工作。 你是Caesar军队的一个分队长。你的工作是破译Caesar送来的信息并汇报给你的上级。 密码很简单，每一个字母对应着一个明文，你将明文向右五步来得到安全的信息。（比如，假如那个字母是‘A’，密文就是‘F’）

## 加密文本 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 明文文本 V W X Y Z A B C D E F G H I J K L M N O P Q R S T U 密文中只有字母被切换了，非字母的字符应该保持不变，所有的字母都是大写的。

# 【输入】

## 这个问题的输入包括一系列（非空）最多100个数据。每一个数据的格式会按照以下格式，并且在不同组数据间不会有空行分隔。所有的字符都是大写的。 一个单独的测试数据包括三个部分： 1. 开始行：单独的一行“START” 。 2. 加密的信息：单独的一行，由1~200个字符组成来自Caesar的一行信息。 3. 结束行：单独的一行“END” 。 最后一组测试数据结束会跟着单独的一行“ENDOFINPUT”。

# 【输出】

## 对每一个测试数据只会有一行输出。它是Caesar的原文。

# 【样例输入】

## START NS BFW, JAJSYX TK NRUTYFSHJ FWJ YMJ WJXZQT TK YWNANFQ HFZXJX END START N BTZQI WFYMJW GJ KNWXY NS F QNYYQJ NGJWNFS ANQQFLJ YMFS XJHTSI NS WTRJ END START IFSLJW PSTBX KZQQ BJQQ YMFY HFJXFW NX RTWJ IFSLJWTZX YMFS MJ END ENDOFINPUT

# 【样例输出】

## IN WAR, EVENTS OF IMPORTANCE ARE THE RESULT OF TRIVIAL CAUSES I WOULD RATHER BE FIRST IN A LITTLE IBERIAN VILLAGE THAN SECOND IN ROME DANGER KNOWS FULL WELL THAT CAESAR IS MORE DANGEROUS THAN HE

# 分析：

## ‘A’ 的值是65，则‘Z’的值是91

## #include<iostream>

## #include<stdio.h>

## #include<string>

## using namespace std;

## int main(void)

## {

## char a[1000],b[1000];

## int i;

## while(gets(a))

## {

## if(strcmp(a,"START")==0) memset(b,0,sizeof(b));

## else if(strcmp(a,"END")==0) printf("%s\n",b);

## else if(strcmp(a,"ENDOFINPUT")==0) break;

## else

## {

## for(i=0;a[i]!='\0';i++)

## {

## if(a[i]>='A'&&a[i]<='Z')

## {

## if(a[i]+21>90)

## b[i]=(a[i]-5);

## else

## b[i]=(a[i]+21);

## }

## else

## b[i]=a[i];

## }

## }

## }

## 

# memset()是干什么的？

## Memset(a,0,sizeof(a))的最用是把数组a清0，在string.h中定义，很方便！

## 代码很规范！

# **对于字符串输入的处理：**

## C语法：

## char buf[20];  gets(buf);

## C++语法：

## 如果用string buf;来保存：

## getline( cin , buf );

## 如果用char buf[ 255 ]; 来保存： cin.getline( buf, 255 );

# **说明：**

## scanf(“ %s%s”,str1,str2)，在多个字符串之间用一个或多个空格分隔；

## 若使用gets函数，应为gets(str1); gets(str2); 字符串之间用回车符作分隔。

## 通常情况下，接受短字符用scanf函数，接受长字符用gets函数。

## 而getchar函数每次只接受一个字符，经常c=getchar()这样来使用。

# **说明：cin.getline的用法：**

## getline 是一个函数，它可以接受用户的输入的字符，直到已达指定个数，或者用户输入了特定的字符。它的函数声明形式（函数原型）如下：

## istream& getline(char line[], int size, char endchar = '\n');

## 不用管它的返回类型，来关心它的三个参数：

## char line[]： 就是一个字符数组，用户输入的内容将存入在该数组内。

## int size : 最多接受几个字符？用户超过size的输入都将不被接受。

## char endchar :当用户输入endchar指定的字符时，自动结束。默认是回车符。

# **说明**

## 结合后两个参数，getline可以方便地实现： 用户最多输入指定个数的字符，如果超过，则仅指定个数的前面字符有效，如果没有超过，则用户可以通过回车来结束输入。

## char name[4];

## cin.getline(name,4,'\n');

## 由于 endchar 默认已经是 '\n'，所以后面那行也可以写成：

## cin.getline(name,4);

# **OJ评测原理**

# **Rightmost Digit（hdu1061）**

## **Problem Description**

## Given a positive integer N, you should output the most right digit of N^N.

## **Input**

## The input contains several test cases. The first line of the input is a single integer T which is the number of test cases. T test cases follow. Each test case contains a single positive integer N(1<=N<=1,000,000,000).

## **Output**

## For each test case, you should output the rightmost digit of N^N.

## **Sample Input**

## 2

## 3

## 4

## **Sample Output**

## 7

## 6

## ***Hint***

## In the first case, 3 \* 3 \* 3 = 27, so the rightmost digit is 7. In the second case, 4 \* 4 \* 4 \* 4 = 256, so the rightmost digit is 6.

# 直接做

## int i,n,tmp,sum;

## cin>>n;

## while(n--)

## {

## sum=1;

## cin>>i;

## tmp=i%10;

## for(int j=1;j<=i;j++)

## { sum=sum\*tmp;

## sum=sum%10;

## }

## sum=sum%10;

## cout<<sum<<endl;

## }

# Time Limit Exceeded

## 时间超时了！

## 就不是直接做的题！

# 先找规律

## 0 1 5 6 这4个数好啊 周期=1

## 4：4 6 周期=2

## 9：9 1

## 2：2 4 8 6 周期=4

## 3: 3 9 7 1

## 7: 7 9 3 1

## 8: 8 4 2 6

## int n;

## int a[10][4] = {

## {0},{1},{6,2,4,8},{1,3,9,7},{6,4},{5},{6},{1,7,9,3},{6,8,4,2},{1,9}

## },d,num;

## scanf("%d",&num);

## while(num--)

## {

## scanf("%ld",&n);

## d = n % 10;

## if(d == 0||d == 1||d == 5||d == 6)

## printf("%d\n",d);

## else if(d == 4||d == 9)

## printf("%d\n",a[d][n % 2]);

## else if(d == 2||d == 3||d == 7||d == 8)

## printf("%d\n",a[d][n % 4]);

## }

# 数的长度 (nefu\_oj 65)

## N! (N的阶乘) 是非常大的数，计算公式为：N! = N \* (N - 1) \* (N - 2) \* ... \* 2 \* 1)。现在需要知道N!有多少（十进制）位。

# 如何计算？

## 所谓N!的（十进制）位数，就是Lg(N!)+1，根据数学公式，有

## N!=1\*2\*……\*N

## Lg(N!)=lg(2)+……+lg(N)

# 核心代码：

## int main(){

## while(scanf("%ld",&n)!=EOF)

## {sum=0.0;

## for(i=2;i<=n;i++) sum+=log10(i);

## printf("%ld\n",(int)sum+1); }

## }

# Leftmost Digit (hdu 1060,nefu\_oj 66)

## **Problem Description**

## Given a positive integer N, you should output the leftmost digit of N^N.  **Input**

## The input contains several test cases. The first line of the input is a single integer T which is the number of test cases. T test cases follow. Each test case contains a single positive integer N(1<=N<=1,000,000,000).

# **Output** For each test case, you should output the leftmost digit of N^N

## **Sample Input**

## 2

## 3

## 4

## **Sample Output**

## 2

## 2

## ***Hint***

## In the first case, 3 \* 3 \* 3 = 27, so the leftmost digit is 2. In the second case, 4 \* 4 \* 4 \* 4 = 256, so the leftmost digit is 2.

# 分析问题：

## 本题目主要是给定一个正整数n，计算nn的结果的最高数位上的数字的值。

## 如果通过求nn的最后结果再求最高数位的数字，方法简单，但是存在问题，就是n较大时候，nn太大而无法用程序存储，需要设计大数的存储方案，且带来了计算过程中的难度而且消耗大量内存空间和时间。在本题中不可取。

# 求解：数学知识

## 另外一种方法，设nn = d.xxx \* 10 (k-1) ,其中k表示nn的位数。

## 那么d.xxx = 10(log10(n^n)-(k-1)) ,再对d.xxx取整即可获得最终结果。那么k是多少呢？

## k = log10(nn)的整数部分+1 = (int)log10(nn)+1;至此，可以获得d的计算公式为

## d = (int)(10^(log10(n^n)-(int)log10(n^n));

## #include <stdio.h>

## #include <math.h>

## int main()

## {

## int t,n;

## double x = 0.0;

## scanf("%d",&t);

## while ( t-- )

## {

## scanf("%d",&n);

## x = n\*log10( (double)n );

## x -= (long long)x;

## x = (int)pow(10,x);

## printf("%.0lf\n",x);

## }

## return 0;

## }

# 64位的类型

## 在VC 里有\_\_int64 这个类型，输出时用%I64d来输出，我们使用 long long类型，也是64位的，输出时用%lld就可以了！

## 用cout<<可以避免%的问题！

# 排序算法

## 1、 sort

## 2、 qsort

# sort

## 头文件： algorithm.h

## 用法：很简单

## 复杂度：n\*log(n)

## 语句：sort(a,a+10);

## 默认：升序

# Sort 的例子

## #include <cstdlib>

## #include <iostream>

## #include <algorithm>

## using namespace std;

## int main(int argc, char \*argv[])

## {

## 

## int data[10];

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

## cin>>data[i];

## sort(data,data+5);

## for(int j=0;j<=4;j++)

## cout<<data[j]<<endl;

## system("PAUSE");

## return EXIT\_SUCCESS;

## }

# Sort 降序该怎么办？

## STL 已经为我们准备好了！

## 升序：sort(begin,end,less<data-type>()); 降序：sort(begin,end,greater<data-type>()).

## 升序：

## Sort(a,a+20,less<int>());

## 降序：

## Sort(a,a+20,greater<int>());

## 头文件没啥变化，就是 algorithm.h

# qsort 方法

## 格式：

## qsort ( 数组名 ，元素个数，元素占用的空间(sizeof)，比较函数)

## 说明：要写比较函数

## 头文件：在 iostream.h

# 比较函数的写法

## int compare(const void \*a,const void \*b) {      return \*(int\*)b-\*(int\*)a;   } //降序

## int compare(const void \*a,const void \*b) {      return \*(int\*)a-\*(int\*)b;   }//升序

# qsort的例子

## int compare(const void \*a,const void \*b) {      return \*(int\*)b-\*(int\*)a;   }

## int main() {      int a[20]={2,4,1,23,5,76,0,43,24,65},i;      for(i=0;i<20;i++)         cout<<a[i]<<endl;      qsort(a,20,sizeof(int),compare);      for(i=0;i<20;i++)         cout<<a[i]<<endl;      return 0; }

# 简单的练习题目：

## 林大OJ 30、32题

## 杭电OJ 2014，2019，2010，2015等

## 排序是基本知识，必会！

# 【循环节】 问题

## Number Sequence hdu1005,nefu\_oj 67

## A number sequence is defined as follows: f(1) = 1, f(2) = 1, f(n) = (A \* f(n - 1) + B \* f(n - 2)) mod 7. Given A, B, and n, you are to calculate the value of f(n).

## **Input**

## The input consists of multiple test cases. Each test case contains 3 integers A, B and n on a single line (1 <= A, B <= 1000, 1 <= n <= 100,000,000). Three zeros signal the end of input and this test case is not to be processed.

## 

## **Output**

## For each test case, print the value of f(n) on a single line.

## **Sample Input**

## 1 1 3

## 1 2 1

## 0 0 0 0

## 

## **Sample Output**

## 2

## 5

# 找出循环节

## 设A=3,B=4时，手工打印出结果，LOOK一下：

## 0 4 5 3 1 1；

## 0 4 5 3 1 1

## 取余后结果是循环的啊！

## 发现新大陆了~~

# 自动找出循环节？

## **for(**i**=**3**;**i**<=**49**;**i**++)**

## **{**

## data**[**i**]=(**a**\***data**[**i**-**1**]+**b**\***data**[**i**-**2**])%**7**;**

## **if (**data**[**i**]==**1**&&**data**[**i**-**1**]==**1**) break;**

## **}**

## int data[101],k=1;

## data[1]=1;

## data[2]=1;

## int a,b,n,t,i;

## while(cin>>a>>b>>n)

## {

## if (a==0&&b==0&&n==0) break;

## if (n<3) cout<<"1"<<endl;

## else

## {

## for(i=3;i<=49;i++)

## {

## data[i]=(a\*data[i-1]+b\*data[i-2])%7;

## //cout<<data[i]<<endl;

## if (data[i]==1&&data[i-1]==1)

## break;

## }

## n=n%(i-2);//我们从3开始跑的

## if (n==0)

## cout<<data[i-2]<<endl;

## else

## cout<<data[n]<<endl;

## }

## }

# **学习方式**

## 练习->总结->练习->总结->……

## [http://acm.hdu.edu.cn](http://acm.hdu.edu.cn/)

## http://acm.nefu.edu.cn

## 杭电ACM论坛

## google、baidu

# **ACM程序设计**

## 东北林业大学 陈宇

## Lg\_chenyu@yahoo.com.cn

# **第二讲**

## **递归**

# 我校的ACM在线评测系统

## acm.nefu.edu.cn

## 课件下载地址：

## acm.nefu.edu.cn/kj/suanfa02.ppt

# **递归（Recursion）就是子程序（或函数）直接调用自己或通过一系列调用语句间接调用自己，是一种描述问题和解决问题的基本方法。**

# **递归有两个基本要素：**

# **⑴ 边界条件：确定递归到何时终止；**

# **⑵ 递归模式：大问题是如何分解为小问题的。**

# 

# 计算阶乘N！

## f(n)=n!可以定义为：

# 代码：

## #include<stdio.h>

## int f(int n){

## return n == 0 ? 1 : f(n-1)\*n;

## }

## int main(){

## printf("%d\n", f(3));

## return 0;

## }

# 递归要调用栈来进行！

## 皇帝（拥有main函数的栈）：大臣，你给我算一 下f(3).

## 大臣（拥有f(3)的栈):知府，你给我算一下f(2).

## 知府（拥有f(2)的栈）：县令，你给我算一下f(1).

## 县令（拥有f(1)的栈）：师爷，你帮我算一下f(0).

## 师爷（拥有f(0)的栈）：回老爷，f(0)=1.

## 县令：（心算）回知府大人，f(1)=1.

## 知府 ： （心算）回大人，f(2)=2.

## 大臣： （心算） 3\*f(2)=6 ，回皇上， f(3)=6

# 运行

## 计算f(3)=6;

## 计算f(100000000),没有输出，溢出也应该有数啊！

## 是段错误！

## 段：是指二进制文件内的区域，某种特定类型的信息被保存在里面。

# 追忆2008年亚洲哈尔滨赛区

## 杨成虎同学的深搜算法就是递归写的，就是不过，因为该算法在递归调用5000次就段错误了，后来改成广搜算法（非递归）的就AC 了，时间多了1个小时，离银牌只差2名，血的教训！

## 我们要牢记！

# 蟠桃记

## Problem Description

## 喜欢西游记的同学肯定都知道悟空偷吃蟠桃的故事，你们一定都觉得这猴子太闹腾了，其实你们是有所不知：悟空是在研究一个数学问题！ 什么问题？他研究的问题是蟠桃一共有多少个！ 不过，到最后，他还是没能解决这个难题，呵呵^-^ 当时的情况是这样的： 第一天悟空吃掉桃子总数一半多一个，第二天又将剩下的桃子吃掉一半多一个，以后每天吃掉前一天剩下的一半多一个，到第n天准备吃的时候只剩下一个桃子。聪明的你，请帮悟空算一下，他第一天开始吃的时候桃子一共有多少个呢？

## **Input**

## 输入数据有多组，每组占一行，包含一个正整数n（1<n<30），表示只剩下一个桃子的时候是在第n天发生的。

## 

## **Output**

## 对于每组输入数据，输出第一天开始吃的时候桃子的总数，每个测试实例占一行。

## 

## **Sample Input**

## 2

## 4

## **Sample Output**

## 4

## 22

# 分析：

## 设A0代表第1天的桃子总数，Ai代表吃完后剩下的桃子数，则：

## A0=2\*（A1+1）

## A1=2\*(A2+1)

## …………….

## An-1=2\*(An+1)

## An=1

# 代码：倒推

## #include <cstdlib>

## #include <iostream>

## using namespace std;

## long long f(int n)

## {

## if (n==1) return 1;

## else

## return 2\*(f(n-1)+1);

## }

# Fibonacci数列

## **description**

## 计算Fibonacci数列的值！该数列为1 1 2 3 5 8 13 .........**input**

## 有多组数据，输入N(3<=N<=50),N代表该数列的第N项的值。

## **output**

## 输出该数列的第N 项的值。

## **sample\_input**

## 4

## 5

## **sample\_output**

## 3

## 5

# 代码：

## long long f(int n)

## {

## if (n==1||n==2) return 1;

## else

## return f(n-1)+f(n-2);

## }

# 实际运行：

## 35

## 9227465

## 36

## 14930352

## 37

## 24157817

## 38

## 39088169

## 39

## 63245986

## 40

## 102334155

## 41

## 165580141

# 献给不懂得优化的同学！

## long long data[100];

## data[1]=1;

## data[2]=1;

## for(int i=3;i<=50;i++)

## data[i]=data[i-1]+data[i-2];

## while(cin>>n)

## cout<<data[n]<<endl;

# Fibonacci Again hdu1021

## **Problem Description**

## There are another kind of Fibonacci numbers: F(0) = 7, F(1) = 11, F(n) = F(n-1) + F(n-2) (n>=2). 林大OJ 70题 **Input**

## **Input consists of a sequence of lines, each containing an integer n. (n < 1,000,000). Output**

## **Print the word "yes" if 3 divide evenly into F(n). Print the word "no" if not.**

## **Sample Input**

## 0

## 1

## 2

## 3

## 4

## 5

# 思路：先找循环节

## 先计算前30项，看看

## 1 2 0 2 2 1 0 1 1 2 0 2 2 1 0 1

## 循环节=8

# 用递归计算

## int f(int n)

## {

## if (n==0) return 1;

## if (n==1) return 2;

## return (f(n-1)+f(n-2))%3;

## }

# main函数

## while(cin>>n)

## {

## t=n%8;

## if (f(t)==0)

## cout<<"yes"<<endl;

## else

## cout<<"no"<<endl;

## 

## }

# 红与黑 hdu 1312

## 有一间长方形的房子，地上铺了红色、黑色两种颜色的正方形瓷砖。你站在其中一块黑色的瓷砖上，只能向相邻的黑色瓷砖移动。请写一个程序，计算你总共能够到达多少块黑色的瓷砖。

# 输入数据

## 包括多个数据集合。每个数据集合的第一行是两个整数W 和H，分别表示x 方向

## 和y 方向瓷砖的数量。W 和H 都不超过20。在接下来的H 行中，每行包括W 个字符。

## 每个字符表示一块瓷砖的颜色，规则如下：

## 1）‘.’：黑色的瓷砖；

## 2）‘#’：白色的瓷砖；

## 3）‘@’：黑色的瓷砖，并且你站在这块瓷砖上。该字符在每个数据集合中唯一出现一次。

## 当在一行中读入的是两个零时，表示输入结束。

# 输出要求

## 对每个数据集合，分别输出一行，显示你从初始位置出发能到达的瓷砖数(记数时包括初始位置的瓷砖)。

# 输入样例

## 6 9

## ....#.

## .....#

## ......

## ......

## ......

## ......

## ......

## #@...#

## .#..#.

## 0 0

# 解题思路

## 这个题目可以描述成给定一点，计算它所在的连通区域的面积。需要考虑的问题包括矩阵的大小，以及从某一点出发向上下左右行走时，可能遇到的三种情况：出了矩阵边界、遇到’.’、遇到’#’。

## 设f(x, y)为从点(x,y)出发能够走过的黑瓷砖总数，则

## f(x, y) = 1 + f(x - 1, y) + f(x + 1, y) + f(x, y - 1) + f(x, y + 1)

## 这里需要注意，凡是走过的瓷砖不能够被重复走过。可以通过每走过一块瓷砖就将它作标记的方法保证不重复计算任何瓷砖。

# Main()函数

## **int** w**,**h**; //定义公有变量 长和宽**

## **char** z**[**21**][**21**];**

## 

## int main(int argc, char \*argv[])

## {

## while(cin>>w>>h)

## {

## if (w==0&&h==0) break;

## 

## for(int i=1;i<=h;i++)

## for(int j=1;j<=w;j++)

## cin>>z[i][j];

## for(int i=1;i<=h;i++)

## for(int j=1;j<=w;j++)

## if (z[i][j]=='@')

## cout<<f(i,j)<<endl;

## }

## //system("PAUSE");

## return EXIT\_SUCCESS;

## }

# 再看递归部分：

## int f(int i,int j)

## {

## if (i<1||i>h||j<1||j>w) //处理边界

## return 0;

## if (z[i][j]!='#')

## {

## z[i][j]=‘#’; //这句话是啥意思？

## return 1+f(i,j-1)+f(i,j+1)+f(i-1,j)+f(i+1,j);

## }

## else

## return 0;

## }

# **ACM程序设计**

## 东北林业大学 陈宇

## Lg\_chenyu@yahoo.com.cn

# **第二讲**

## **高精度**

# 我校的ACM在线评测系统

## acm.nefu.edu.cn

## 课件下载地址：

## acm.nefu.edu.cn/kj/acm03.ppt

# 还看N!的位数这题

## Hdu 1018 Big Number

## Pku 1423 Big Number

## 用1+lg(1)+lg(2)+..+lg(n)的方法，在PKU上TLE 了，超时了

# 原因！

## 1 <= m <= 10^7 规模很大

## 这道题目给出的限时是1000ms。假如对于给出的每一个数据，我们都慢慢地将它从log10(1) 慢慢加到log10(N)，绝对会超时。因为里面有大量重复的运算，例如log10(1)，如果有100组数据，那么它的值就会被计算100次。于是，我们想到，能否把所有计算过的log10值保存起来，然后若遇到重复的，就从这个结果数组里面提取就可以了，不用再计算。这个方法很好，但是题目给出的数据规模比较大，开一个10^7大的double数组，绝对会Memory Excceed Limit。那么，既然如此，我们应该用什么方法来计算呢？

## 假设问题给出的C组数据，是从小往大排列的，例如，给出三个数据，10，20，30，那么我们可以想到，计算log10(20!)的时候，我们是可以利用long10(10!)的结果。因为：

## log10(10!) = log10(1) + log10(2) + log10(3) + ... + log10(9) +log10(10)

## log10(20!) = log10(1) + log10(2) + log10(3) + ... + log10(9) +log10(10) + log10(11) + log10(12) +... +log10(19) + log10(20)

## 容易看出：

## log10(20!) = log10(10!) + log10(11) + log10(12) +... +log10(19) + log10(20) ;

## 同理：

## log10(30!) = log10(10!) + log10(21) + log10(22) +... +log10(29) + log10(30) ;

## 也就是说，我们只需要保存之前的那个数的运算结果。这种方法，有人就说是DP，但是我觉得这算是记忆化搜索吧。

## 题目没有说给出的数据是有序的，但是我们可以通过排序使之有序。至于排序，那么当然是系统的sort()函数了。我解题时定义了一个结构体：

## typedef struct

## {

## int num;

## int id;

## double result;

## } data;

# 【Stirling公式】

## 利用斯特林(Stirling)公式求解n!的位数:

## 易知整数n的位数为[lg10(n)]+1. 用Stirling公式计算n!结果的位数时，可以两边取对数，得： log10(n!) = log10(2\*PI\*n)/2+n\*log10(n/E); 故n!的位数为

## res=log10(2\*PI\*n)/2+n\*log10(n/E)+1

## #include <iostream>

## #include <math.h>

## using namespace std; //注意 e和pi的值要精确

## const double 2.7182818284590452354, pi = 3.141592653589793239;

## double str\_ling(int n)

## {

## return 0.5\*log10(2\*pi\*n)+n\*log10(n/e);

## }

## int main()

## {

## int t,m;

## cin>>t;

## while(t--)

## {

## cin>> m;

## cout<<(int)str\_ling(m)+1<<endl;

## }

## system("pause");

## return 1;

## }

## 在32位机器里，有符号整数(int)的取值范围是**-2147483648 ~ +2147483647**，无符号整数(unsigned int) 的取值范围是**0 ~ 4294967295**。超过这个范围的数据可以用浮点型(double)来表示，如50!。但用浮点数来表示整数通常不便于整数的运算，比如整数除法跟浮点数除法含义不一样，浮点数也无法实现取余运算等等。

## 另外，**超过浮点数取值范围的数据**，比如一个1000位的整数，**无法用常规方法来处理**。

## 这些精度很高的数据通常称为**高精度数**，或称为**大数**。

## 高精度数的运算只能用本章介绍的高精度数计算方法来处理。

# **Primary Arithmetic(初等算术） pku2562， nefu也有**

# 分析

## Int 型的上限是2000000000，可以保存9位整数，因此本题可以用整数来保存输入，每次把a和b分别模10，就能获取他们的位数。

## long long a,b,c,sum;

## while(cin>>a>>b)

## {

## if (a==0&&b==0) break;

## c=0;sum=0;

## for(int i=0;i<=9;i++)

## {

## if ((a%10+b%10+c)>=10)

## {sum++;

## c=1;

## }

## else

## c=0;

## a=a/10;b=b/10;

## 

## }

# 计算N!

## 输入不超过10000的正整数n,计算n!的具体值？

## 输入：30

## 输出：265252859812191058636308480000000

# 分析：

## 10000！的有多大？ 用计算器算=5\*1035658，可以用50000个元素的数组f来保存，为了防止进位溢出，我们让f[0]保存结果的个位，f[1]表示十位，等等。输出的时候，数组高位上的0要忽略。

## 例如：11！=39916800

## 则，12！=11！\*12，也就是39916800的每位都乘12，再进位，从低到高，所以数组要用f[0]表示低位，输出时再逆序输出！

# 先定义变量

## const int maxn=50000;

## int n,c,k;

## int f[maxn+1];

## 如果不知道10000！的阶乘有多少位，那肯定WA!

## while(cin>>n)

## {

## 

## memset(f,0,sizeof(f));

## f[0]=1;

## for (int i=1;i<=n;i++)

## {

## c=0;//代表进位

## for(int j=0;j<=maxn;j++)

## {

## int s=f[j]\*i+c;

## f[j]=s%10;

## c=s/10;

## }

## }

# 看输出部分

## **for(**k**=**maxn**;**k**>=**0**;**k**--)**

## **if (**f**[**k**]!=**0**) break;**

## **for(int** j**=**k**;**j**>=**0**;**j**--)**

## cout**<<**f**[**j**];**

## cout**<<**endl**;**

## **}**

# 能不能再快点？ 思考

## 发现f[i]里面的值只有1个数字，但f[i]是INT型啊，能存9位数啊，利用一下。

## int s=f[j]\*i+c;

## f[j]=s%100000;//好好看看

## c=s/100000;//每5位数字再进位

## 这回数组f开20000就够了，时间也快了~

## 运行时间是1778ms！

# 如何打印输出？

## 12！=479001600，后5位01600，输出是1600，要处理好这个问题就行了！

## for(k=maxn;k>=0;k--)

## if (f[k]!=0) break;

## printf("%d",f[k]);

## for(int j=k-1;j>=0;j--)

## printf(“%05d”,f[j]);//5位的数字，不足前面添0

## printf("\n");

# 当输入的数很大时，则必须用字符串来处理

## 再来看“初等算术”这道题~~

# 高精度计算的基本思路

## 高精度计算的基本思路是：**用数组存储参与运算的数的每一位，在运算时以数组元素所表示的位为单位进行运算**。可以采用**字符数组**，也可以采用**整数数组**，到底采用字符数组还是整数数组更方便，应试具体题目而定。

# 高精度数的基本运算

## 本节以几道竞赛题目为例讲解高精度数的**加法**、**乘法**和**除法**运算的实现方法。

# 高精度数的加法

**题目描述：**

十进制大数的加法运算。

**输入描述：**

输入文件的第1行为一个整数N，表示输入文件中接下来有N组数据。每组数据最多包含100行。每一行由一个非常长的十进制整数组成，**这个整数的长度不会超过100个字符而且只包含数字**，每组数据的最后一行为0，表示这组数据结束。

每组数据之间有一个空行。

**输出描述：**

对**输入文件中的每组数据**，**输出它们的和**。每两组数据的输出之间有一个空行。

**样例输入：**

1

99999278961257987

126792340765189

998954329065419876

432906541

23

0

**样例输出：**

1099080400800349616

**分析：**

首先，题目中提到，**整数的长度不会超过100位**，**所以这些整数只能采用字符数组读入**。但在**对每位进行求和**时，**可以采用字符形式**，**也可以采用整数形式**。

本题**用整数形式处理更方便**。

在本题中，求和时要注意以下两点：

1)计算每位和时，**得到的进位可能大于1**，如图7.4所示。

2)累加各大数得到的和，**其位数可能会比参与运算的大数的位数还要多**。稍加分析即可得出结论，如果参与求和运算的大数**最大长度为maxlen**，因为参加求和运算的大数个数不超过100个，所以**求和结果长度不超过maxlen+2**。因此求和时可以一直求和到maxlen+2位，然后去掉后面的0，再以相反的顺序输出各位整数即可。

#include <stdio.h>

#include <string.h>

int main( )

{

char buffer[200]; //存储(以字符形式)读入的每个整数

int array[200][200]; //逆序后的大数(每位是整数形式)

int answer[200]; //及求得的和

int i, j, k; //循环变量

int num\_integers; //读入整数的个数

int len, maxlen; //每个整数的长度，及这些整数的最大长度

int sum, carry, digit;//每位求和运算后得到的总和, 进位, 及该位的结果

int N; //测试数据的个数

scanf( "%d", &N );

for( k=1; k<=N; k++ )

{

maxlen = -1;

memset( array, 0, sizeof(array) );

memset( answer, 0, sizeof(answer) );

for( num\_integers = 0; num\_integers < 200; num\_integers++ )

{

gets( buffer );

if( strcmp(buffer, "0") == 0 ) break;

len = strlen(buffer);

if( len>maxlen ) maxlen = len;

for( i = 0; i < len; i++ )//逆序存放大数的每位(整数形式)

array[num\_integers][i] = buffer[len - 1 - i] - '0';

}

carry = 0;

for( i = 0; i < maxlen+2; i++ ) //对这些整数的每位进行求和

{

sum = carry;

for( j = 0; j < num\_integers; j++ )

sum += array[j][i];

digit = sum % 10;

carry = sum / 10;

answer[i] = digit;

}

for( i = maxlen+2; i >= 0; i-- ) //统计求和结果的位数

{

if( answer[i] != 0 ) break;

}

while( i >= 0 ) printf( "%d", answer[i--] ); //逆序输出求和结果

printf( "\n" );

if( k<N ) printf( "\n" ); //两个输出块之间有一个空行

}

return 0;

}

# 高精度数的乘法

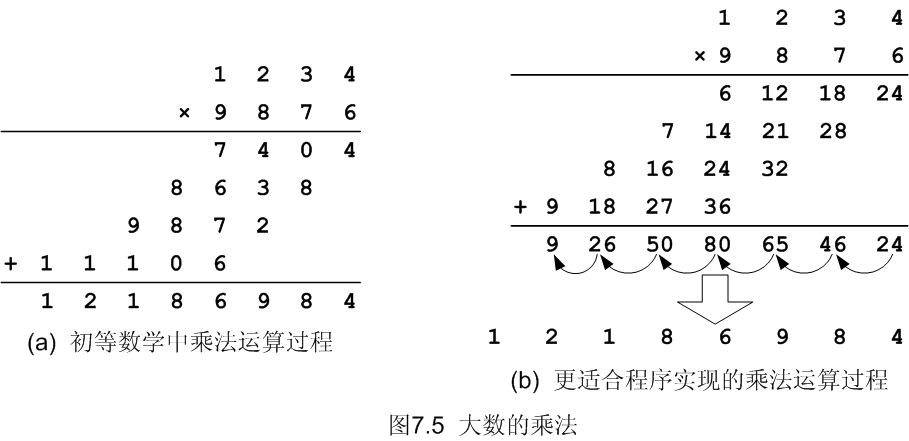
## 初等数学里乘法的运算过程，如图所示。该运算过程有如下特点：

### **多位数的乘法是转换成1位数的乘法及加法来实现的**，即把第二个乘数的每位数乘以第一个乘数，把得到的中间结果累加起来。

### 第二个乘数的每位数进行乘法运算的**中间结果**，**是与第二个乘数参与运算的位右对齐的**。如图(a)所示，第二个程序的第2位为7，参与乘法运算得到的中间结果“8638”是和7对齐的。

## 在初等数学里，**乘法运算得到的每个中间结果都是处理了进位的**：在中间结果里，一出现进位马上累加到高一位，如图(a)中的中间结果“11106”是已经处理了进位的结果。

## 但是，为方便程序实现，**对中间结果的进位处理更方便的做法是等全部中间结果运算完后再统一处理**。如图(b)所示，每个中间结果，“6 12 18 24”、“7 14 18 24”、“8 16 24 32”、“9 18 27 36”都是没有处理进位的，都是第二个乘数的每位乘以第一个乘数每位的原始乘积。这些中间结果累加后，再一位一位的处理进位。



**题目描述：**

给定两个位数不超过100位的正整数，求它们的乘积。

**输入描述：**

输入文件中包含多个测试数据。每个测试数据占两行，分别为一个正整数，每个正整数的位数不超过100位。输入数据一直到文件尾。

**输出描述：**

对输入文件中的每个测试数据，输出其中两个正整数的乘积。

**样例输入：**

981567

32976201

123456789

987654321

123456789987654321

987654321123456789

**样例输出：**

32368350686967

121932631112635269

121932632103337905662094193112635269

分析：

1. 对读入的字符形式的大整数，把其各位上的数值**以整数形式取出来**，**以相反的顺序**存放到一个整型数组里。如下图所示。
2. 把**第二个乘数中的每个数乘以第一个乘数**，把**得到的中间结果累加起来**，注意对齐方式，以及累加每位的中间结果时不进位。
3. 把累加的**中间结果**，**由低位向高位进位**。再把得到的最终结果**按相反的顺序转换成字符输出**。

代码：

#include <stdio.h>

#include <string.h>

char a[101], b[101]; //输入的两个正整数(字符形式)

int len\_a, len\_b; //输入的正整数长度

int ai[101], bi[101]; //输入的两个正整数(以整数形式存储每一位)

int temp[202]; //每一位乘法的中间结果

char product[201]; //乘积

void reverse( char s[ ], int si[] ) //以逆序顺序存放大数中的各位数(整数形式)

{

int len = strlen(s);

for( int i=0; i<len; i++ )

si[len-1-i] = s[i]-'0';

}

int main( )

{

int i, j;

while( scanf( "%s", a ) != EOF )

{

scanf( "%s", b );

len\_a = strlen(a); len\_b = strlen(b);

reverse(a, ai); reverse(b, bi);

memset( temp, 0, sizeof(temp) );

memset( product, 0, sizeof(product) );

for( i=0; i<len\_b; i++ ) //用大整数b的每位去乘大整数a

{

int start = i;//得到的中间结果跟大整数b中的位对齐

for( j=0; j<len\_a; j++ )

{

temp[start++] += ai[j]\*bi[i];

}

}

for( i=0; i<202; i++ ) //低位向高位进位

{

if( temp[i]>9 )

{

temp[i+1] += temp[i]/10;

temp[i] = temp[i]%10;

}

}

for( i=201; i>=0; i-- ) //求乘积的长度

{ if( temp[i] ) break; }

int lenp = i+1; //乘积的长度

for( i=0; i<lenp; i++ ) //将乘积各位转换成字符形式

product[lenp-1-i] = temp[i]+'0';

product[lenp] = 0; //串结束符标志

printf( "%s\n", product );

}

return 0;

}

# **ACM程序设计**

## 东北林业大学 陈宇

## Lg\_chenyu@yahoo.com.cn

# **第三讲**

## **贪心**

# 我校的ACM在线评测系统

## acm.nefu.edu.cn

## 课件下载地址：

## acm.nefu.edu.cn/kj/acm04.ppt

**贪心法在解决问题的策略上目光短浅，只根据当前已有的信息就做出选择，而且一旦做出了选择，不管将来有什么结果，这个选择都不会改变。换言之，贪心法并不是从整体最优考虑，它所做出的选择只是在某种意义上的局部最优。**

**这种局部最优选择并不总能获得整体最优解（Optimal Solution），但通常能获得近似最优解（Near-Optimal Solution）。**

**例：用贪心法求解付款问题。**

**假设有面值为5元、2元、1元、5角、2角、1角的货币，需要找给顾客4元6角现金，为使付出的货币的数量最少，首先选出1张面值不超过4元6角的最大面值的货币，即2元，再选出1张面值不超过2元6角的最大面值的货币，即2元，再选出1张面值不超过6角的最大面值的货币，即5角，再选出1张面值不超过1角的最大面值的货币，即1角，总共付出4张货币。**

## **在付款问题每一步的贪心选择中，在不超过应付款金额的条件下，只选择面值最大的货币，而不去考虑在后面看来这种选择是否合理，而且它还不会改变决定：一旦选出了一张货币，就永远选定。付款问题的贪心选择策略是尽可能使付出的货币最快地满足支付要求，其目的是使付出的货币张数最慢地增加，这正体现了贪心法的设计思想。**

**贪心法求解的问题的特征：**

（1）最优子结构性质

当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质，也称此问题满足最优性原理。

（2）贪心选择性质

所谓贪心选择性质是指问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来得到。

## **用贪心法求解问题应该考虑如下几个方面**：

## （1）候选集合C：为了构造问题的解决方案，有一个候选集合C作为问题的可能解，即问题的最终解均取自于候选集合C。例如，在付款问题中，各种面值的货币构成候选集合。

## （2）解集合S：随着贪心选择的进行，解集合S不断扩展，直到构成一个满足问题的完整解。例如，在付款问题中，已付出的货币构成解集合。

## （3）解决函数solution：检查解集合S是否构成问题的完整解。例如，在付款问题中，解决函数是已付出的货币金额恰好等于应付款。

## （4）选择函数select：即贪心策略，这是贪心法的关键，它指出哪个候选对象最有希望构成问题的解，选择函数通常和目标函数有关。例如，在付款问题中，贪心策略就是在候选集合中选择面值最大的货币。

## （5）可行函数feasible：检查解集合中加入一个候选对象是否可行，即解集合扩展后是否满足约束条件。例如，在付款问题中，可行函数是每一步选择的货币和已付出的货币相加不超过应付款。

## **贪心算法的基本步骤：**

## 1、从问题的某个初始解出发。

## 2、采用循环语句，当可以向求解目标前进一步时，就根据局部最优策略，得到一个部分解，缩小问题的范围或规模。

## 3、将所有部分解综合起来，得到问题的最终解。

# **导引问题：FatMouse' Trade**

## 一只老鼠有M磅猫食，然后在N个房间里面用猫食换JavaBean，房间i中能用F[i]磅的猫食来换J[i]磅的JavaBean，而且老鼠可以在一个房间里根据一定比例a%来换取JavaBean. 现在他是这任务分配给你：告诉他，他的JavaBeans的获取能最多。

# input

## The input consists of multiple test cases. Each test case begins with a line containing two non-negative integers M and N. Then N lines follow, each contains two non-negative integers J[i] and F[i] respectively. The last test case is followed by two -1's. All integers are not greater than 1000.

# out

## For each test case, print in a single line a real number accurate up to 3 decimal places, which is the maximum amount of JavaBeans that FatMouse can obtain.

## 

# 怎么算出来？

## 输入

## 5 3

## 7 2

## 4 3

## 5 2

# 如何对结构体变量进行排序？sort

## struct sa {

## int j; // javabeen 的数量

## int f; // 猫食的数量

## double awk; // f[i]/j[i]的比值

## }data[2002];

## int cmp( const sa &a,const sa &b)

## {

## return (a.awk) > (b.awk);

## }

# <algorithm.h>

## sort(data,data+n,cmp);

## 说明：

# 分析：先排序，根据J[i]/F[i]

## 可以考虑用冒泡法，但不推荐，已经会sort了，qsort也可以的！

## 定义结构体：

## struct sa {

## int j; // 放J[i]

## int f; // 放F[i]

## double awk; // 放 J[i]/F[i]

## }data[2002]; //其实定义1001就够！

## while(cin>>m>>n)

## {

## sum=0.0;k=0;

## if (m==-1&&n==-1) break;

## for(int i=0;i<n;i++)

## {

## cin>>data[i].j>>data[i].f;

## data[i].awk=(double)data[i].j/(double)data[i].f ;

## }

## sort(data,data+n,cmp);

## for(k=0;k<n;k++)

## {

## if (m>=data[k].f)//大于 F[i]

## {

## sum=sum+data[k].j; //都换了

## 

## m=m-data[k].f;

## }

## else

## {

## sum=sum+(double)m\*data[k].awk;

## break;

## }

## }

## printf("%.3f\n",sum);

## }

# **所谓“贪心算法”是指：**

## 在对问题求解时，总是作出在**当前看来是最好**的选择。也就是说，不从整体上加以考虑，它所作出的仅仅是在某种意义上的局部最优解（是否是全局最优，需要证明）。

# **特别说明：**

## 若要用贪心算法求解某问题的整体最优解，必须首先证明贪心思想在该问题的应用结果就是最优解！！

# **用事实说话¡ª¡ª**

# **算法分析：**

## 不妨用Begin[i]和End[i]表示事件i的开始时刻和结束时刻。则原题的要求就是找一个最长的序列a1<a2<…<an，满足：

## Begin[a1]<End[a1]<=…<= Begin[an]<End[an]

# **思考：**

## **请谈谈自己的解题思路**

# **思考题**

## **hdu2037 今年暑假不AC**

## “今年暑假不AC？” “是的。” “那你干什么呢？” “看世界杯呀，笨蛋！” “@#$%^&\*%...” 确实如此，世界杯来了，球迷的节日也来了，估计很多ACMer也会抛开电脑，奔向电视了。 作为球迷，一定想看尽量多的完整的比赛，当然，作为新时代的好青年，你一定还会看一些其它的节目，比如新闻联播（永远不要忘记关心国家大事）、非常6+7、超级女生，以及王小丫的《开心辞典》等等，假设你已经知道了所有你喜欢看的电视节目的转播时间表，你会合理安排吗？（目标是能看尽量多的完整节目）

## **Input**

## 输入数据包含多个测试实例，每个测试实例的第一行只有一个整数n(n<=100)，表示你喜欢看的节目的总数，然后是n行数据，每行包括两个数据Ti\_s,Ti\_e (1<=i<=n)，分别表示第i个节目的开始和结束时间，为了简化问题，每个时间都用一个正整数表示。n=0表示输入结束，不做处理。

## **Output**

## 对于每个测试实例，输出能完整看到的电视节目的个数，每个测试实例的输出占一行。

# 

## 输入：

## 12

## 1 3

## 3 4

## 0 7

## 3 8

## 15 19

## 15 20

## 10 15

## 8 18

## 6 12

# 先按结束时间排下序（从小到大）

## 1 3 0 3 2 5 6 4 10 8 15 15

## 3 4 7 8 9 10 12 14 15 18 19 20

## 竖着看！

## 用心一算：

## 1 3 5 10 15

## 3 4 10 15 19

# 代码： 固定的套路

## struct sa {

## int x;

## int y;

## }data[10000];//又开大了，开101就行

## int cmp(const sa &a,const sa &b)

## {

## return a.y < b.y;

## }

## sa tmp;

## while(cin>>n&&n!=0)

## {

## sum=0;

## for(int i=0;i<n;i++)

## cin>>data[i].x>>data[i].y;

## sort(data,data+n,cmp);

## sum=1;

## tmp=data[0];

## for(int j=1;j<n;j++)

## {

## if (tmp.y<=data[j].x)

## {

## sum++;

## tmp=data[j];

## }

## 

## }

## cout<<sum<<endl;

# **算法分析：**

## 如果N>=M，那么显然用M条长度为1的线段可以覆盖住所有的区间，所求的线段总长为M。

## 如果N=1，那么显然所需线段总长为：…

## 如果N=2，相当于N=1的情况下从某处断开（从哪儿断开呢？）。

## 如果N=k呢？从间隔最大的2个部分断开！

# Wooden Sticks (hdu1051)

## There is a pile of n wooden sticks. The length and weight of each stick are known in advance. The sticks are to be processed by a woodworking machine in one by one fashion. It needs some time, called setup time, for the machine to prepare processing a stick. The setup times are associated with cleaning operations and changing tools and shapes in the machine. The setup times of the woodworking machine are given as follows:

## (a) The setup time for the first wooden stick is 1 minute.

## (b) Right after processing a stick of length l and weight w , the machine will need no setup time for a stick of length l' and weight w' if l<=l' and w<=w'. Otherwise, it will need 1 minute for setup. You are to find the minimum setup time to process a given pile of n wooden sticks. For example, if you have five sticks whose pairs of length and weight are (4,9), (5,2), (2,1), (3,5), and (1,4), then the minimum setup time should be 2 minutes since there is a sequence of pairs (1,4), (3,5), (4,9), (2,1), (5,2).

# input

## The input consists of T test cases. The number of test cases (T) is given in the first line of the input file. Each test case consists of two lines: The first line has an integer n , 1<=n<=5000, that represents the number of wooden sticks in the test case, and the second line contains 2n positive integers l1, w1, l2, w2, ..., in, wn, each of magnitude at most 10000 , where li and wi are the length and weight of the i th wooden stick, respectively. The 2n integers are delimited by one or more spaces

# out

## The output should contain the minimum setup time in minutes, one per line.

# Sample Input

## 3

## 5

## 4 9 5 2 2 1 3 5 1 4

## 3

## 2 2 1 1 2 2

## 3

## 1 3 2 2 3 1

## (4,9) (5,2) (2,1) (3,5) (1,4)

## 先根据 L 进行排序，注意：当L 值相同时？怎么处理？

## 比如：

## (4,9) (5,2) (2,1) (3,5) (1,4) (1,3)

## 结果相同吗？找一名同学提问？

# 分析代码：

## int tmp,m,n,s,e,

## int l[5001],w[5001]；

## int data[5001];//做标记，用过否？

## cin>>m; //数据的组数

## **for**(int k=1;k<=m;k++)

## {

## **memset**(data,-1,**sizeof**(data));

## cin>>n;

## **for**(int i=1;i<=n;i++)

## cin>>l[i]>>w[i];

# 用了冒泡排序了

## **for**(int i=1;i<n;i++)

## {

## tmp=i;

## **for**(int j=i+1;j<=n;j++)

## {

## **if** (l[j]<l[tmp]) tmp=j;

## **if** (l[j]==l[tmp]&&w[j]<w[tmp]) tmp=j;

## }

## s=l[i];l[i]=l[tmp];l[tmp]=s; e=w[i];w[i]=w[tmp];w[tmp]=e;

## }

## int count=0,k=1;

## **for**(int i=1;i<=n;i++)

## { k=i;

## **if** (data[i]==-1) //干啥的？

## { count++;data[i]=1;

## **for**(int j=i+1;j<=n;j++)

## { **if** (data[j]==-1)

## {

## **if** (l[j]>=l[k]&&w[j]>=w[k])

## { data[j]=1; k=j;}

## } } } }

## cout<<count<<endl;

## **return** EXIT\_SUCCESS; }

# 我们进一步提高编码的水平

## 改进排序的方法：

## 用sort

## int cmp( const sa &a,const sa &b)

## {

## if (a.x!=b.x)

## return (a.x) < (b.x);

## else

## return (a.y)<(b.y);

## }

# 定义结构体

## struct sa{

## int x;//length 的值

## int y;//weight的值

## int flag; //是否被用过

## }data[5001];

# 主函数

## int main(int argc, char \*argv[])

## {

## 

## int t,n,count,tmp;

## cin>>t;

## while(t--)

## {

# 输入数据

## cin>>n;

## for(int i=0;i<n;i++)

## {

## cin>>data[i].x;

## cin>>data[i].y;

## data[i].flag=0;//没被用过

## }

## sort(data,data+n,cmp);

## count=0;

## for(int j=0;j<n;j++)

## {

## if (data[j].flag==1) continue;

## count++; tmp=j;

## for(int k=j+1;k<n;k++)

## {

## if (!data[k].flag&&data[k].x>=data[tmp].x && data[k].y>=data[tmp].y)

## {data[k].flag=1;tmp=k;}

## }

## 

## }

# 收尾工作

## cout<<count<<endl;

## }

## system("PAUSE");

## return EXIT\_SUCCESS;

## }

# Hdu 1050 **Moving Tables**

## The floor has 200 rooms each on the north side and south side along the corridor. Recently the Company made a plan to reform its system. The reform includes moving a lot of tables between rooms. Because the corridor is narrow and all the tables are big, only one table can pass through the corridor. Some plan is needed to make the moving efficient. The manager figured out the following plan: Moving a table from a room to another room can be done within 10 minutes. When moving a table from room i to room j, the part of the corridor between the front of room i and the front of room j is used. So, during each 10 minutes, several moving between two rooms not sharing the same part of the corridor will be done simultaneously. To make it clear the manager illustrated the possible cases and impossible cases of simultaneous moving.

## For each room, at most one table will be either moved in or moved out. Now, the manager seeks out a method to minimize the time to move all the tables. Your job is to write a program to solve the manager's problem

# Input

## The input consists of T test cases. The number of test cases ) (T is given in the first line of the input file. Each test case begins with a line containing an integer N , 1 <= N <= 200, that represents the number of tables to move. Each of the following N lines contains two positive integers s and t, representing that a table is to move from room number s to room number t each room number appears at most once in the N lines). From the 3 + N -rd line, the remaining test cases are listed in the same manner as above.

# Output

## The output should contain the minimum time in minutes to complete the moving, one per line.

# Sample Input

## 3

## 4

## 10 20

## 30 40

## 50 60

## 70 80

## 2

## 1 3

## 2 200

## 3

## 10 100

## 20 80

## 30 50

# 先分析问题：

## 是贪心，而且是简单的贪心。

## 怎么排序？

## 感觉好像是：看电视节目的那个题啊，1次能搬多少就搬多少，别客气！

# 思路：

## 10 20 start[i] end[i]

## 30 40

## 50 60

## 70 80

## 排序后，start[i+1]>end[i]，就可以1次搬了，不满足条件的，剩下接着搬，反正搬1次10分钟啊

# Int main()

## int data[401],start[401],end[401];

## int k,s,e,m,n,tmp,top,count=0;

## cin>>m;

## **for**(int p=1;p<=m;p++)

## {

## 循环内部见后面。。。。

## **memset**(data,0,**sizeof**(data));

## cin>>n; //多少对？

## **for**(int i=1;i<=n;i++) {cin>>start[i]>>end[i];

## **if** (start[i]>end[i]) //防止数据逆序的

## {

## tmp=start[i];start[i]=end[i];end[i]=tmp;}

## }

## **for**(int i=1;i<=n-1;i++)

## {

## top=i;

## **for**(int j=i+1;j<=n;j++)

## {

## **if** (start[j]<start[top]) top=j;

## **if** (start[j]==start[top]&& end[j]<end[top]) top=j;

## }

## s=start[i];start[i]=start[top];start[top]=s; e=end[i];end[i]=end[top];end[top]=e;

## } count=0; k=1;

## **for**(int i=1;i<=n;i++)

## { k=i;

## **if** (data[i]==0) { data[i]=1;count=count+1;

## **for**(int j=i+1;j<=n;j++)

## { **if** (data[j]==0)

## { **if**(start[j]>end[k]&&!(start[j]%2==0&&start[j]-1==end[k]))//是为啥？ {data[j]=1;k=j;}

## }

## } } }

# 收尾工作

## cout<<count\*10<<endl;

## } *//system("PAUSE");*

## **return** EXIT\_SUCCESS;

## }

## //对门的房间号例5，6，是不能同时搬的，如 3-5；6-8，是不行的，因为5和6共享了1个走廊！

# 代码也需要提高：

## struct sa{

## int start;

## int end;

## int flag;

## }data[501];

## int cmp( const sa &a,const sa &b)

## {

## if (a.start!=b.start)

## return (a.start) < (b.start);

## else

## return (a.end)<(b.end);

## }

# int main(int argc, char \*argv[])

## int t,n,count,tmp;

## cin>>t;

## while(t--)

## {

## cin>>n;

## for(int i=0;i<n;i++)

## {

## cin>>data[i].start;

## cin>>data[i].end;

## if (data[i].start>data[i].end)//下面防止输入 30 20这种情况，就 //是大的在前面

## {tmp=data[i].start;data[i].start=data[i].end;data[i].end=tmp; }

## data[i].flag=0;//没被用过

## }

## sort(data,data+n,cmp);

## count=0;

## for(int j=0;j<n;j++)

## {

## if (data[j].flag==1) continue;

## count++; tmp=j;data[j].flag=1;

## for(int k=j+1;k<n;k++)

## {

## if (!data[k].flag && data[k].start>=data[tmp].end && !(data[k].start%2==0&&(data[k].start-1==data[tmp].end)))

## {data[k].flag=1;tmp=k;}

## }

## 

## }

# 收尾：

## cout<<count\*10<<endl;

## }

## system("PAUSE");

## return EXIT\_SUCCESS;

## }

## 本题还可以这样做？

# **HDOJ\_1050 Moving Tables**

## Sample Input

## 3 4 10 20 30 40 50 60 70 80 2 1 3 2 200 3 10 100 20 80 30 50

# **算法分析：**

## 1、如果没有交叉，总时间应该是多少？

## 2、影响搬运时间的因素是什么？

## 3、如果每趟处理都包含最大重叠，处理后的效果是什么？

## 4、得出什么结论？

## 把房间号和走廊对应起来~

## 如 输入 10 - 20，其实对应：5-10

## 就是把走廊编号，然后找重叠的部分。

## 输入基数：加1后，除2；

## 输入偶数：直接除2；

## int main(int argc, char \*argv[])

## {

## int start[401],end[401],data[401];

## int t,n,tmp,aaa;

## cin>>t;

## while(t--)

## {

## cin>>n;

## for(int i=0;i<=400;i++) data[i]=0;

## for(int i=0;i<n;i++)

## {

## cin>>start[i]>>end[i];

## if ( start[i]>end[i])

## {tmp=start[i];start[i]=end[i];end[i]=tmp;}

## if (start[i]%2==1) start[i]=(start[i]+1)/2;

## else start[i]=start[i]/2;

## if (end[i]%2==1) end[i]=(end[i]+1)/2;

## else end[i]=end[i]/2;

## for(int k=start[i];k<=end[i];k++)

## data[k]++;

## }

## aaa=data[1];

## for(int j=2;j<=200;j++)

## if (aaa<data[j]) aaa=data[j];

## cout<<aaa\*10<<endl;

## 

## }

# 收尾

## system("PAUSE");

## return EXIT\_SUCCESS;

## }

# **贪心算法的基本步骤（总结）**

## 1、从问题的某个初始解出发。

## 2、采用循环语句，当可以向求解目标前进一步时，就根据局部最优策略，得到一个部分解，缩小问题的范围或规模。

## 3、将所有部分解综合起来，得到问题的最终解。

# **思考：田忌赛马-经典版 hdu1052**

# **再思考：田忌赛马-升级版**

# **谈谈自己的想法吧～**

# **Case 1:**

## **King: 200 180 160**

## **Tianji: 190 170 150**

# **Case 2:**

## **King: 200 180 160**

## **Tianji: 180 170 150**

# **Case 3:**

## **King: 200 180 160**

## **Tianji: 180 155 150**

# **总体的思路是什么？**

# **提醒：**

## 很多贪心类型的题目都象本题一样，不是最朴素的贪心，而是需要做一些变化，对于我们，关键是找到贪心的本质！

# **最后一个思考题**

# ***Any questions?***

# **相关练习**

## 更多贪心题目请参见HDOJ题目分类——

## http://acm.hdu.edu.cn/typeclass.php

# **Welcome to HDOJ**

# **ACM程序设计**

# 动态规划

**动态规划法将待求解问题分解成若干个相互重叠的子问题，每个子问题对应决策过程的一个阶段，一般来说，子问题的重叠关系表现在对给定问题求解的递推关系（也就是动态规划函数）中，将子问题的解求解一次并填入表中，当需要再次求解此子问题时，可以通过查表获得该子问题的解而不用再次求解，从而避免了大量重复计算。**

用动态规划法求解的问题具有特征：

* 能够分解为相互重叠的若干子问题；
* 满足最优性原理（也称最优子结构性质）：该问题的最优解中也包含着其子问题的最优解。

**动态规划法设计算法一般分成三个阶段：**

**（1）分段：将原问题分解为若干个相互重叠的子问题；**

**（2）分析：分析问题是否满足最优性原理，找出动态规划函数的递推式；**

**（3）求解：利用递推式自底向上计算，实现动态规划过程。**

**动态规划法利用问题的最优性原理，以自底向上的方式从子问题的最优解逐步构造出整个问题的最优解。**

# 例1斐波那契数列 nefu 85

## 计算斐波那契数列的值！该数列为1 1 2 3 5 8 13 21 .........

## 有多组数据，每组1行，用N表示，1 <= N <= 50。

## 输出Fibonacci(N)的值!

# 代码：

## int n;

## long long feibo[51];

## feibo[1]=1;

## feibo[2]=1;

## for(int i=3;i<=50;i++)

## feibo[i]=feibo[i-1]+feibo[i-2];//打表

## while(cin>>n)

## cout<<feibo[n]<<endl;

# 用递归能做吗？

## long long data[51];

## long long fblq(int n)

## {

## if (n==1||n==2) return 1;

## else

## {

## if (data[n]==0)

## data[n]=fblq(n-1)+fblq(n-2); //看看这个！ 只算1次！

## return data[n];

## 

## }

## 

## 

## }

# 现在同学们在纸上计算下题：

## 计算N!

## **input**

## 有多组数据，每组一行，用N表示，0 < = N <= 20;

## **output**

## 输出N！

# 递归的代码：

## #include <cstdlib>

## #include <iostream>

## using namespace std;

## long long data[21];

## long long jc(int n)

## {

## if (n==0||n==1) return 1;

## if (data[n]==0)

## data[n]=n\*jc(n-1);

## return data[n];

## 

## }

# 递推的代码：

## #include <cstdlib>

## #include <iostream>

## using namespace std;

## int main(int argc, char \*argv[])

## {

## int n; long long data[51];

## memset(data,0,sizeof(data));

## data[0]=1;data[1]=1;

## for(int i=2;i<=20;i++)

## data[i]=i\*data[i-1];

## while(cin>>n)

## cout<<data[n]<<endl;

## system("PAUSE");

## return EXIT\_SUCCESS;

## }

# Nefu20 **穿过街道**

## 一个城市的街道布局如下:从最左下方走到最右上方,每次只能往上或往右走,一共有多少种走法?

## **input**

## 输入很多行行数,每行1个数字代表n的值,当n=0时结束(2<=n<=15)

## **output**

## 输出对应每行n值的走法.

## **sample\_input**

## 1

## 2

## 10

## 5

## 0

## **sample\_output**

## 2

## 6

## 184756

## 252

# 递归的代码：

## int f[100][100];

## int getf(int x,int y)

## {

## if (f[x][y]!=-1) return f[x][y];

## int result;

## if (x==0||y==0)

## result=1;

## else

## result=getf(x-1,y)+getf(x,y-1);

## f[x][y]=result;

## return result;

## }

# Main()

## int main(int argc, char \*argv[])

## {

## int i,j,n;

## memset(f,-1,sizeof(f));

## while(cin>>n)

## {

## if (n==0) break;

## cout<<getf(n,n)<<endl;

## }

## system("PAUSE");

## return EXIT\_SUCCESS;

## }

# 其实这题递推的代码更短：

## int n;

## long long data[21][21];

## for(int i=0;i<=20;i++)

## {

## data[i][0]=1;// 处理边界

## data[0][i]=1;// 处理边界

## }

## for(int j=1;j<=20;j++)

## for(int k=1;k<=20;k++)

## data[j][k]=data[j-1][k]+data[j][k-1];//公式

## while(cin>>n&&n)

## cout<<data[n][n]<<endl;

# 经过上面的思考，回忆一下：

## HDU 2041超级楼梯

## HDU 2044小蜜蜂

## 结论：

## 都可以用DP 来做，其实只要数据不重复，递推和递归就算DP 了！

# 例：**数字三角形 nefu17**

## 7

## 3 8

## 8 1 0

## 2 7 4 4

## 4 5 2 6 5

## 给出了一个数字三角形。从三角形的顶部到底部有很多条不同的路径。对于每条路径，把路径上面的数加起来可以得到一个和，你的任务就是找到最大的和。 注意：路径上的每一步只能从一个数走到下一层上和它最近的左边的那个数或者右边的那个数。

# 分析：

## 7（7）

## 3（10） 8（15）

## 8（18） 1(max(10,15)+1=16) 0(15)

## 2(20) 7(25) 4(20) 4(19)

## 4(24) 5(30) 2(27) 6(26) 5(24)

## 看明白了吧！

## 方程式： 1(max(10,15)+1=16) 有啥感觉？

## inp[i][j]=inp[i-1][j]+inp[i-1][j-1];

# #include <iostream>

# Using namespace std;

## int data[102][102];//输入放数据

## Int inp[102][102];// 放数字和

## Int main()

## {

## int n;

## while(cin>>n)

## {

## memset(inp,0,sizeof(inp));

## memset(data,0,sizeof(data));

## for(int i=1;i<=n;i++)

## for(int j=1;j<=i;j++)

## {

## cin>>data[i][j];

## if (i==1) inp[i][j]=data[i][j];

## else

## inp[i][j]=max(inp[i-1][j],inp[i-1][j-1])+data[i][j];

## 

## 

## }

# //找出最后1行的最大值，并输出之!

## int tmp=0;

## for(int k=1;k<=n;k++)

## if (tmp<=inp[n][k]) tmp=inp[n][k];

## cout<<tmp<<endl;

}

# **ACM程序设计**

# **Function Run Fun nefu16**

## We all love recursion! Don't we? Consider a three-parameter recursive function w(a, b, c):

## if a <= 0 or b <= 0 or c <= 0, then w(a, b, c) returns: 1

## if a > 20 or b > 20 or c > 20, then w(a, b, c)

## returns: w(20, 20, 20)

## if a < b and b < c, then w(a, b, c)

## returns: w(a, b, c-1) + w(a, b-1, c-1) - w(a, b-1, c)

## otherwise it

## returns: w(a-1, b, c) + w(a-1, b-1, c) + w(a-1, b, c-1) - w(a-1, b-1, c-1) This is an easy function to implement. The problem is, if implemented directly, for moderate values of a, b and c (for example, a = 15, b = 15, c = 15), the program takes hours to run because of the massive recursion.

## **input**

## The input for your program will be a series of integer triples, one per line, until the end-of-file flag of -1 -1 -1. Using the above technique, you are to calculate w(a, b, c) efficiently and print the result.

## **output**

## Print the value for w(a,b,c) for each triple.

## **sample\_input**

## 1 1 1

## 2 2 2

## 10 4 6

## 50 50 50

## -1 7 18

## -1 -1 -1

# 思路：递归好

## long long data[21][21][21];

## long long inf(int x,int y,int z)

## { if (x<=0||y<=0||z<=0) return 1;

## if (x>20||y>20||z>20) return inf(20,20,20);

## if (x<y&&y<z)

## {

## if (data[x][y][z]==-1)

## data[x][y][z]=inf(x,y,z-1)+inf(x,y-1,z-1)-inf(x,y-1,z);

## return data[x][y][z];

## 

## }

## else

## {

## if (data[x][y][z]==-1)

## {data[x][y][z]=inf(x-1,y,z)+inf(x-1,y-1,z)+inf(x-1,y,z-1)-inf(x-1,y-1,z-1);

## 

## }

## return data[x][y][z];

## }

## }

# main()函数

## int main(int argc, char \*argv[])

## {

## int x,y,z;

## while(cin>>x>>y>>z)

## {

## if (x==-1&&y==-1&&z==-1) break;

## memset(data,-1,sizeof(data));

## cout<<"w("<<x<<", "<<y<<", "<<z<<")"<<" = "<<inf(x,y,z)<<endl;

## }

## //system("PAUSE");

## return EXIT\_SUCCESS;

## }

# **Recaman Sequence nefu91**

## The Recaman's sequence is defined by a0 = 0 ; for m > 0, am = am−1 − m if the rsulting am is positive and not already in the sequence, otherwise am = am−1 + m. The first few numbers in the Recaman's Sequence is 0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9 ... Given k, your task is to calculate ak.

## **input**

## The input consists of several test cases. Each line of the input contains an integer k where 0 <= k <= 500000. The last line contains an integer −1, which should not be processed.

## **output**

## For each k given in the input, print one line containing ak to the output.

## **sample\_input**

## 7

## 10000

## -1

# 分析：从下往上，打表也行

## #define size 500000

## int data[size+1];

## bool inp[size\*10];

# int main(int argc, char \*argv[]) {

## int n;

## memset(inp,false,sizeof(inp));

## data[0]=0;inp[0]=true;

## for(int i=1;i<=500000;i++)

## {

## if (data[i-1]-i>0&&inp[data[i-1]-i]==false)

## data[i]=data[i-1]-i;

## else

## data[i]=data[i-1]+i;

## inp[data[i]]=true;

## }

## while(cin>>n&&n!=-1)

## {

## cout<<data[n]<<endl;

## }

## system("PAUSE");

## return EXIT\_SUCCESS;

## }

# **滑雪 nefu18**

## **description**

## 每到冬天,信息学院的张健老师总爱到二龙山去滑雪,喜欢滑雪百这并不奇怪， 因为滑雪的确很刺激。可是为了获得速度，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。张老师想知道载一个区域中最长底滑坡。区域由一个二维数组给出。数组的每个数字代表点的高度。

## **input**

## 输入的第一行表示区域的行数R和列数C(1 <= R,C <= 100)。下面是R行，每行有C个整数，代表高度h，0<=h<=10000。

## **output**

## 输出最长区域的长度。

## **sample\_input**

## 5 5

## 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。

## **sample\_output**

## 25

# 分析：递归

## int data[101][101];

## int inp[101][101];

## int m,n;

## int f(int i,int j)

## {

## int max1,max2;

## int max\_right=0,max\_left=0,max\_up=0,max\_down=0;

## if (data[i][j]>0) return data[i][j];

## if (j+1<=n&&inp[i][j]>inp[i][j+1])

## max\_right=f(i,j+1);//右边

## if (j-1>=1&&inp[i][j]>inp[i][j-1])

## max\_left=f(i,j-1);//左边

## if (i+1<=m&&inp[i][j]>inp[i+1][j])

## max\_up=f(i+1,j);//上面

## if (i-1>=1&&inp[i][j]>inp[i-1][j])

## max\_down=f(i-1,j);//下面

## max1=max(max\_right,max\_left);

## max2=max(max\_up,max\_down);

## data[i][j]=max(max1,max2)+1;//4个方向上的最大值

## return data[i][j];

## }

# int main(int argc, char \*argv[]) {

## memset(data,0,sizeof(data));

## int k,max\_all;

## while(cin>>m>>n)

## {

## max\_all=0;

## for(int i=1;i<=m;i++)

## for(int j=1;j<=n;j++)

## cin>>inp[i][j];

## for(int i=1;i<=m;i++)

## for(int j=1;j<=n;j++)

## k=f(i,j);

## for(int i=1;i<=m;i++)

## for(int j=1;j<=n;j++)

## if (max\_all<data[i][j]) max\_all=data[i][j];

## cout<<max\_all<<endl;

## }

## 

## system("PAUSE");

## return EXIT\_SUCCESS;

## }

# 采药 nefu19

## **description**

## 辰辰是个很有潜能、天资聪颖的孩子，他的梦想是称为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。” 如果你是辰辰，你能完成这个任务吗？

## **input**

## 输入的第一行有两个整数T（1 <= T <= 1000）和M（1 <= M <= 100），T代表总共能够用来采药的时间，M代表山洞里的草药的数目。接下来的M行每行包括两个在1到100之间（包括1和100）的的整数，分别表示采摘某株草药的时间和这株草药的价值。

## **output**

## 输出只包括一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

## **sample\_input**

## 70 3

## 71 100

## 69 1

## 1 2

# 代码分析：可递归，也可递推

## int time\_1[1001],value[101];

## int data[1001][101];

## int t,n;

## int dp(int t1,int i)

## { if (data[t1][i]>0) return data[t1][i];

## if (t1==0||i>n) return 0;

## if (t1>=time\_1[i]) data[t1][i]=max(dp(t1-time\_1[i],i+1)+value[i],dp(t1,i+1));

## else data[t1][i]=dp(t1,i+1);

## return data[t1][i]; }

## memset(data,-1,sizeof(data));

## cin>>t>>n;

## for(int i=1;i<=n;i++)

## cin>>time\_1[i]>>value[i];

## cout<<dp(t,1)<<endl;

## return EXIT\_SUCCESS;

## }

# **Welcome to HDOJ**

## **Thank**

## **You ~**