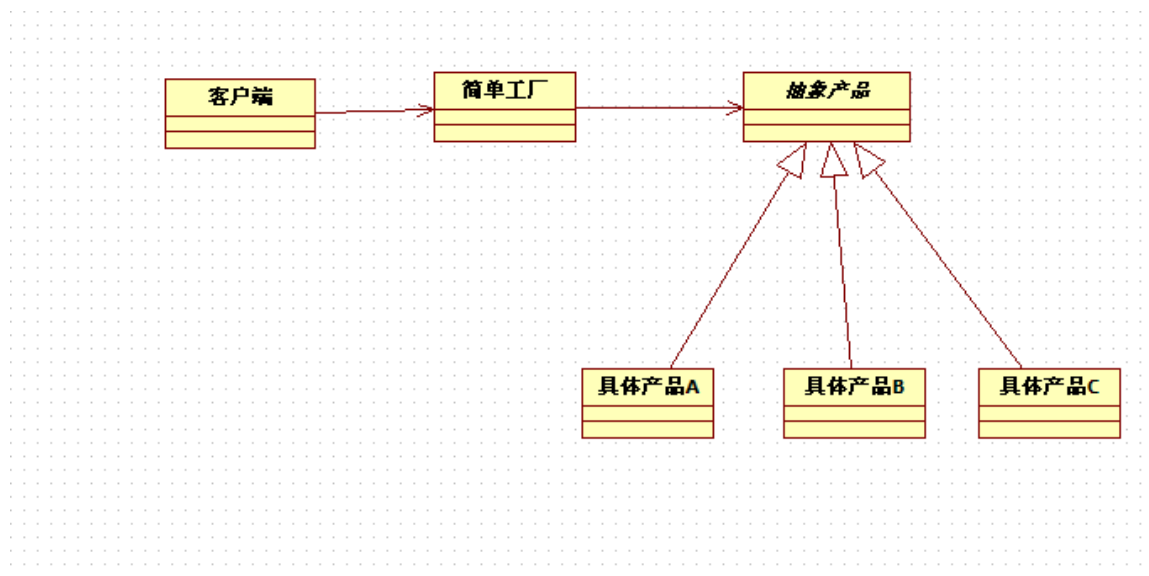


设计模式之简单工厂模式

2016年5月8日 22:37

简单工厂模式是属于创建型模式，又叫做静态工厂方法（Static Factory Method）模式，但不属于23种GOF设计模式之一。简单工厂模式是由一个工厂对象决定创建出哪一种产品类的实例。简单工厂模式是工厂模式家族中最简单实用的模式，可以理解为是不同工厂模式的一个特殊实现。

简单工厂模式的UML图：



简单工厂模式深入分析：

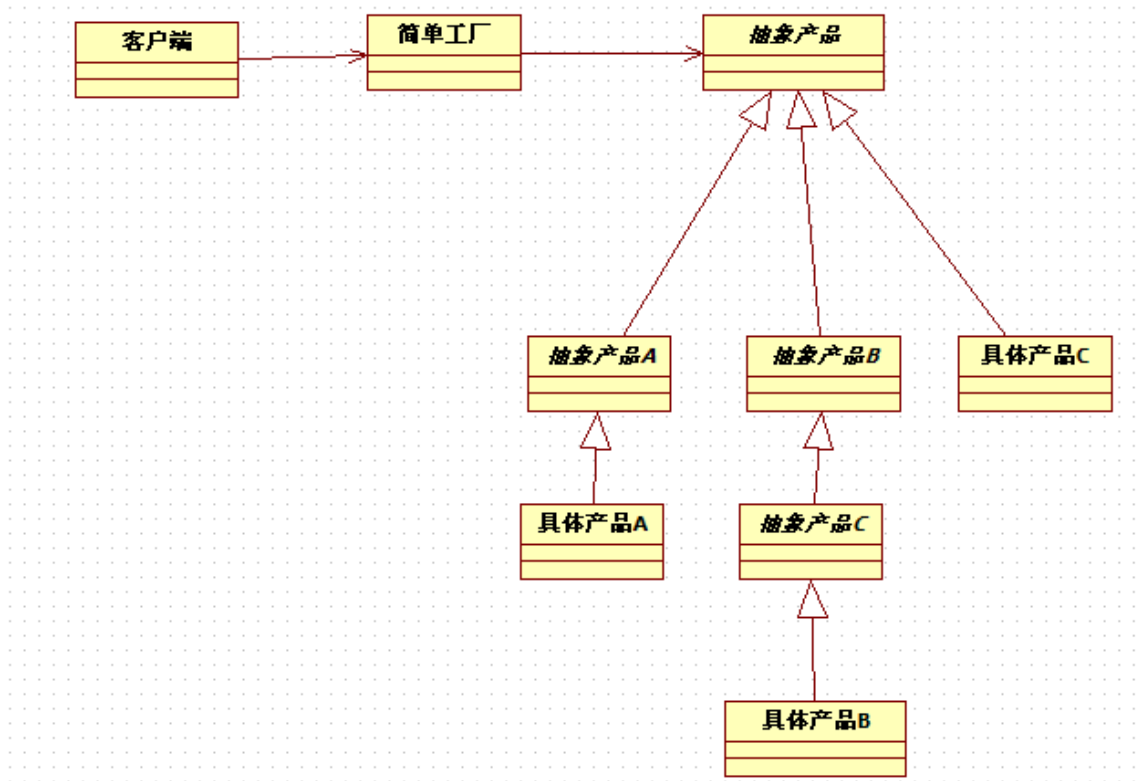
简单工厂模式解决的问题是如何去实例化一个合适的对象。

简单工厂模式的核心思想就是：有一个专门的类来负责创建实例的过程。

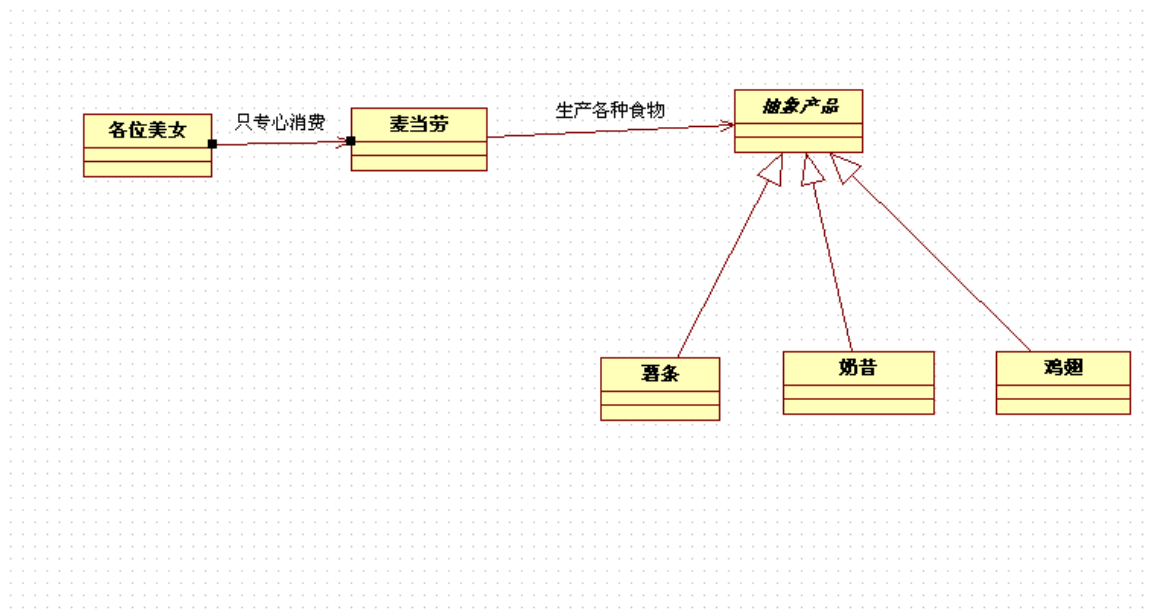
具体来说，把产品看着是一系列的类的集合，这些类是由某个抽象类或者接口派生出来的一个对象树。而工厂类用来产生一个合适的对象来满足客户的要求。

如果简单工厂模式所涉及到的具体产品之间没有共同的逻辑，那么我们就可以使用接口来扮演抽象产品的角色；如果具体产品之间有功能的逻辑或，我们就必须把这些共同的东西提取出来，放在一个抽象类中，然后让具体产品继承抽象类。为实现更好复用的目的，共同的东西总是应该抽象出来的。

在实际的使用中，抽象产品和具体产品之间往往是多层次的产品结构，如下图所示：



来一个具体的例子：



优点：

工厂类是整个模式的关键,包含了必要的逻辑判断,根据外界给定的信息,决定究竟应该创建哪个具体类的对象。

通过使用工厂类,外界可以从直接创建具体产品对象的尴尬局面摆脱出来,仅仅需要负责“消费”对象就可以了。而不必管这些对象究竟如何创建及如何组织的。

明确了各自的职责和权利，有利于整个软件体系结构的优化。

缺点：

由于工厂类集中了所有实例的创建逻辑，违反了高内聚责任分配原则，将全部创建逻辑集中到了一个工厂类中；它所能创建的类只能是事先考虑到的，如果需要添加新的类，则就需要改变工厂类了；

当系统中的具体产品类不断增多时候，可能会出现要求工厂类根据不同条件创建不同实例的需求。这种对条件的判断和对具体产品类型的判断交错在一起，很难避免模块功能的蔓延，对系统的维护和扩展非常不利。

使用场景

工厂类负责创建的对象比较少；

客户只知道传入工厂类的参数，对于如何创建对象（逻辑）不关心；

由于简单工厂很容易违反高内聚责任分配原则，因此一般只在很简单的情况下应用。

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=5465369>>

DP大作战——多重背包

2016年5月8日 22:39

题目描述

在之前的上机中，零崎已经出过了01背包和完全背包，也介绍了使用-1初始化容量限定背包必须装满这种小技巧，接下来的背包问题相对有些难度，可以说是01背包和完全背包的进阶问题。

多重背包：物品可以有0-n件。

对于第*i*种物品，我们有取0件，1件…*n*[*i*]件共*n*[*i*]+1种策略，状态转移方程为 $f[i][v] = \max \{ f[i-1][v-k \times c[i]] + k \times w[i] \mid 0 \leq k \leq n[i] \}$ 。在这里，很自然的有一种策略可以将其转化为01背包，即将物品换为*n*[*i*]件01背包中的物品，但是复杂度为 $O(V \sum n_i)$ ，时间复杂度没有降低。实际上，对于所有类似情况，我们都可以利用二进制求和来降低时间复杂度。即将物品替换为价值和费用 * 系数=1,2,2^2,...,2^k, *n*[*i*]-2^k+1的物品。系数之和为*n*[*i*]，表明不能取到多于*n*[*i*]件物品，但可以取到0…*n*[*i*]中任意一个整数件。利用这一优化，算法事件复杂度可以降到 $O(V \sum \log n_i)$ 。

实际上 $F[i][j]$ 只依赖于 $F[i-1][j-k \times w[i]]$ ，这里依赖项之间构成了一个 $\{j \bmod w[i]\}$ 剩余类，不同剩余类之间无关，注意到这点利用单调队列，每个状态均摊 $O(1)$ 的时间，可以进一步将算法时间复杂度优化至 $O(VN)$ 级别的，不过在此不再详细阐述。（其实也就是NOIP程度，放在大学应该可以接受，但是这个优化个人感觉已经脱离dp）

DD大牛给出的伪代码。

```
def MultiplePack(F,C,W,M)
    if C * M >= V
        CompletePack(F,C,W)
        return //考虑这里为什么可以直接用完全背包
    k := 1
    while k < M
        ZeroOnePack(kC,kW)
        M := M - k
        k := 2k
    ZeroOnePack(C-M,W-M)
```

输入

第一个数为数据组数 $1 \leq n \leq 10$

接下来*n*组测试数据，每组测试数据由2部分组成。

第一行为背包容量*V*，物品种类数*N*。 $1 \leq V \leq 30000, 1 \leq N \leq 200$

接下来*N*行每行三个数为物品价值*v*，物品重量*w*，物品件数*M*。

$1 \leq v, w \leq 200, 1 \leq M \leq 25$

输出

对于每组数据，输出一行，背包能容纳的最大物品价值

输入样例

```
1
10 2
1 2 3
2 3 2
```

输出样例

```
6
```

题目来源：

<http://biancheng.love/contest/10/problem/>

E/index

解题思路:

问题属于背包问题, 同时包括了0-1和完全背包, 因此为多重背包问题。

按照之前的想法, 只要判断每件物品的件数, 可以确定对于该物品是使用0-1背包还是完全背包。

0-1背包的代码:

```
1 void Zeronepack(int w,int v)
2 {
3     for(int i=V; i>=w; i--)
4         if(dp[i]<dp[i-w]+v)
5             dp[i]=dp[i-w]+v;
6 }
```

完全背包的代码:

```
1 void Compack(int w,int v)
2 {
3     for(int i=w; i<=V; i++)
4         if(dp[i]<dp[i-w]+v)
5             dp[i]=dp[i-w]+v;
6 }
```

本题需要利用0-1背包以及完全背包来解决多重背包问题

代码:

```
1 #include <bits/stdc++.h>
2 #include<stdio.h>
3 #include<string.h>
4 int dp[30005];
5 int V,N;
6 void Compack(int w,int v)
7 {
8     for(int i=w; i<=V; i++)
9         if(dp[i]<dp[i-w]+v)
10             dp[i]=dp[i-w]+v;
11 }
12
13 void Zeronepack(int w,int v)
14 {
15     for(int i=V; i>=w; i--)
16         if(dp[i]<dp[i-w]+v)
17             dp[i]=dp[i-w]+v;
18 }
19
20 int main()
21 {
22     int kase,v,w,m;
23     scanf("%d",&kase);
24     while(kase--)
25     {
26         memset(dp,0,sizeof(dp));
27         scanf("%d%d",&V,&N);
28         for(int i=1; i<=N; i++)
29         {
30             scanf("%d%d%d",&v,&w,&m);
31             if(w*m>=V)
32                 Compack(v,w);
33             else
34             {
35                 for(int j=1; j<m; j<<1)
36                 {
37                     Zeronepack(j*w,j*v);
38                     m-=j;
39                 }
40                 Zeronepack(m*w,m*v);
41             }
42         }
43     }
```

```
43         printf("%d\n", dp[V]);  
44     }  
45     return 0;  
46 }
```

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4991895>>

单链表的使用——计算多项式加法

2016年5月8日 22:40

Problem Description

jhljx是一名学习特别差的学生，尤其在数学方面非常不擅长。。像计算类的问题从来算不对。。这天，上初中的jhljx走在回家的路上.....

郁闷ing.....



他突然想起老师给他布置的一个作业，让他计算多项式的加法。

居然还没做。。



给出的两个多项式的项数分别为 n 和 m 。两个多项式为 $P_1(x)=A_0+A_1*x+A_2*x^2+\cdots+A_{n-1}*x^{(n-1)}+A_n*x^n$, $P_2(x)=B_0+B_1*x+B_2*x^2+\cdots+B_{n-1}*x^{(n-1)}+B_n*x^n$ 。

请你计算两个多项式相加的结果。

Input

输入多组数据。

对于每组数据，第一行输入两个数 $n(0 \leq n \leq 500)$ 和 $m(0 \leq m \leq 500)$ ，分别表示两个多项式的最高次数。

第二行为 $n+1$ 个整数(可以为负数)，分别表示第一个多项式的系数。

第三行为 $m+1$ 个整数(可以为负数)，分别表示第二个多项式的系数。

保证系数从0次项，1次项，2次项…… n 次项(m 次项)的顺序给出。

Output

输出最终的多项式。详细输出格式见样例。

Sample Input

```
14 18
1 0 0 0 0 0 -10 0 2 0 0 0 0 0 7
0 0 0 0 -1 0 10 0 0 0 -3 0 0 0 8 0 0 0 4
```

Sample Output

```
1-x^4+2x^8-3x^10+15x^14+4x^18
```

Hint

jhljx:不会敲?



***: 额。。还不是很熟嘛。。



jhljx: 见书上P66页

***: 谢啦。。

jhljx: 书上代码有错。。请谨慎使用。。

***:



请用链表实现，不要偷工减料。

题目来源:

<http://acm.buaa.edu.cn/problem/1201/>

下面给出链表操作代码：（关键在于输入阶段的判断，建议自己写代码并提交）

```

1  #include<iostream>
2  #include<cstdio>
3  #include<cstdlib>
4  #include<cmath>
5  #include<memory.h>
6  #define maxsize 510
7  int c[maxsize],d[maxsize],e[maxsize];
8  using namespace std;
9  typedef struct Term
10 {
11     float coef;
12     int exp;
13     struct Term *link;
14 } List,*Polynomial;
15 void initTerm(Polynomial& first)
16 {
17     first=new List;
18     if(!first)
19     {
20         cerr<<"存储分配失败\n";
21     }
22     first->link=NULL;
23 }
24 int main()
25 {
26     int m,n,mnum;
27     while(cin>>m>>n)
28     {
29         int counter=0;
30         memset(c,0,sizeof(c));
31         memset(d,0,sizeof(d));
32         memset(e,0,sizeof(e));
33         if(n>m)mnum=n;
34         else mnum=m;
35         Polynomial firsts,firstp,s,p,prs,prp;
36         initTerm(firsts);
37         initTerm(firstp);
38         prs=firsts;
39         prp=firstp;
40         for(int i=0; i<=m; i++)
41         {
42             s=new List;
43             s->link=prs->link;
44             prs->link=s;
45             prs=s;
46             s->exp=i;
47             cin>>s->coef;
48         }
49         for(int i=0; i<=n; i++)
50         {
51             p=new List;
52             p->link=prp->link;
53             prp->link=p;

```

```

54         prp=p;
55         p->exp=i;
56         cin>>p->coef;
57     }
58     float a[510];
59     if(m>=n)
60     {
61         for(int i=0; i<=n; i++)
62         {
63             firstp=firstp->link;
64             firsts=firsts->link;
65             c[i]=firstp->coef+firsts->coef;
66         }
67
68         for(int i=n+1; i<=m; i++)
69         {
70             firsts=firsts->link;
71             c[i]=firsts->coef;
72         }
73     }
74     if(m<n)
75     {
76         for(int i=0; i<=m; i++)
77         {
78             firstp=firstp->link;
79             firsts=firsts->link;
80             c[i]=firstp->coef+firsts->coef;
81         }
82         for(int i=m+1; i<=n; i++)
83         {
84             firstp=firstp->link;
85             c[i]=firstp->coef;
86         }
87     }
88     for(int i=0; i<=mnum; i++)
89     {
90         if(c[i]!=0)
91         {
92             d[counter]=c[i];
93             e[counter]=i;
94             counter++;
95         }
96     }
97     // for(int i=0;i<counter;i++)
98     //     cout<<d[i]<<" ";
99     // cout<<endl;
100    // for(int i=0;i<counter;i++)
101    //     cout<<e[i]<<" ";
102    // cout<<endl;
103    if(e[0]==0)
104    {
105        cout<<d[0];
106    }
107    else if(e[0]==1)
108    {
109        if(d[0]>0)
110        {
111            if(d[0]==1)
112                cout<<"x";
113            else
114                cout<<d[0]<<"x";
115        }
116        if(d[0]<0)
117        {
118            if(d[0]==-1)
119                cout<<"-x";
120            else
121                cout<<d[0]<<"x";
122        }
123    }
124    else
125    {
126        if(d[0]>0)
127        {
128            if(d[0]==1)

```

```

129         cout<<"x^"<<e[0];
130     else
131         cout<<d[0]<<"x^"<<e[0];
132     }
133     if(d[0]<0)
134     {
135         if(d[0]==-1)
136             cout<<"-x^"<<e[0];
137         else
138             cout<<d[0]<<"x^"<<e[0];
139     }
140 }
141 for(int i=1; i<counter; i++)
142 {
143     if(e[i]==1)
144     {
145         if(d[i]>0)
146         {
147             if(d[i]==1)
148                 cout<<"x";
149             else
150                 cout<<"+"<<d[i]<<"x";
151         }
152         if(d[i]<0)
153         {
154             if(d[i]==-1)
155                 cout<<"-x";
156             else
157                 cout<<d[i]<<"x";
158         }
159     }
160     else
161     {
162         if(d[i]>0)
163         {
164             if(d[i]==1)
165                 cout<<"+"<<"x^"<<e[i];
166             else
167                 cout<<"+"<<d[i]<<"x^"<<e[i];
168         }
169         if(d[i]<0)
170         {
171             if(d[i]==-1)
172                 cout<<"-x^"<<e[i];
173             else
174                 cout<<d[i]<<"x^"<<e[i];
175         }
176     }
177 }
178 cout<<endl;
179 }
180 }

```

下面给出纯数组求解过程:



View Code

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4989313>>

单链表逆置

2016年5月8日 22:41

单链表在数据结构中算是一个十分基础的结构。在单链表上可以有很多的其他衍生比如，循环链表，带头结点的单链表，双向链表等等。同时单链表在实际应用中也很有很重要的意义。举例说明：集合的交集、并集问题，使用循环链表可以解决约瑟夫环问题、使用链表可以解决多项式的加法乘法。

也许在以后的学习中很少自己再完整写过链表，但是在数据结构这门课的学习中，将一些基础的链表操作写一遍是很重要的。

单链表的一些优点：相对于顺序表而言，插入删除算法较简便，但是对于查找以及返回某位置的数据却没有顺序表方便。

下面代码给出单链表的结构定义，以及简单的逆序操作。

代码：

```
1 #include<bits/stdc++.h>
2
3 #define max_size 1000010
4 int a[max_size];
5
6 using namespace std;
7
8 typedef int DataType; //类型定义
9 typedef struct node //单链表
10 {
11     DataType data;
12     struct node* next;
13 } LinkNode, *LinkList;
14
15 /****由数组创建单链表****/
16 LinkList CreateList(DataType a[], int n)
17 {
18     LinkNode* ListHead=new LinkNode();
19     ListHead->data=a[0];
20     ListHead->next=NULL;
21     for(int i=n-1; i>=1; i--)
22     {
23         LinkNode* p=new LinkNode();
24         p->data=a[i];
25         p->next=ListHead->next;
26         ListHead->next=p;
27     }
28     return ListHead;
29 }
30
31 void PrintList(LinkList ListHead)
32 {
33     if(NULL==ListHead)cout<<"The List is empty!"<<endl;
34     else
35     {
36         LinkNode* p=ListHead;
37         while(p!=NULL)
38         {
39             cout<<p->data<<" ";
40             p=p->next;
41         }
42         cout<<endl;
43     }
44 }
45
46 void ReverseList(LinkNode* pCur, LinkList& ListHead) //递归算法
47 {
48     if( (NULL==pCur) || (NULL==pCur->next) )
49     {
```

```

50     ListHead=pCur;
51 }
52 else
53 {
54     ListNode* pNext=pCur->next;
55     ReverseList(pNext,ListHead); //递归逆置后继结点
56     pNext->next=pCur;           //将后继结点指向当前结点。
57     pCur->next=NULL;
58 }
59 }
60
61 int main()
62 {
63
64     while(true)
65     {
66         int n;
67         cout<<"请输入创建单链表的长度:"<<endl;
68         cin>>n;
69         for(int i=0; i<n; i++)
70         {
71             cout<<"请输入第"<<i+1<<"个数据:";
72             cin>>a[i];
73         }
74         ListNode* list=CreateList(a,n);
75         cout<<"逆置前:"<<endl;
76         PrintList(list);
77         ListNode*pTemp=list;
78         ReverseList(pTemp,list);
79         cout<<"逆置后:"<<endl;
80         PrintList(list);
81     }
82     return 0;
83 }

```

下面给出其他操作的代码:

单链表结构定义:

```

1 //程序2-15 (单链表的结构定义)
2
3 typedef int DataType;
4 typedef struct node//链表结点
5 {
6     DataType data;//数据域
7     struct node *link;//链接指针域
8 }ListNode, *LinkList;
9
10 //单链表适用于插入或删除频繁、存储空间需求不定的情形
11
12 /* (1) 单链表中数据的逻辑结构与其物理结构可能不一致,
13     通过指针将各个元素按照线性表的逻辑顺序连接起来
14     (2) 单链表的长度可以扩充
15     (3) 对单链表的便利和查找只能从头指针指示的首元节点开始,
16         不能像顺序表那样直接访问某个指定的节点
17     (4) 当进行插入或删除运算时, 只需修改相关结点的指针域即可, 既方便又快捷
18     (5) 由于单链表的每个结点带有指针域, 因而比顺序表需要的存储空间多
19 */

```

单链表的插入算法:



[View Code](#)

单链表的删除算法:



[View Code](#)

单链表的一些其他操作 (初始化、表空、清空、表长计算等):



View Code

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4989302>>

钢条切割问题

2016年5月8日 22:41

突然发现，钢条切割竟然没有写。

假设公司出售一段长度为*i*英寸的钢条的价格为 P_i ($i = 1, 2, \dots$ 单位：美元)，下面给出了价格表样例：

长度 <i>i</i>	1	2	3	4	5	6	7	8	9	10
价格 P_i	1	5	8	9	10	17	17	20	24	30

切割钢条的问题是这样的：给定一段长度为*n*英寸的钢条和一个价格表 P_i ，求切割方案，使得销售收益 R_n 最大。

当然，如果长度为*n*英寸的钢条价格 P_n 足够大，最优解可能就是完全不需要切割。

对于上述价格表样例，我们可以观察所有最优收益值 R_i 及对应的最优解方案：

$R_1 = 1$ ，切割方案 $1 = 1$ （无切割）
 $R_2 = 5$ ，切割方案 $2 = 2$ （无切割）
 $R_3 = 8$ ，切割方案 $3 = 3$ （无切割）
 $R_4 = 10$ ，切割方案 $4 = 2 + 2$
 $R_5 = 13$ ，切割方案 $5 = 2 + 3$
 $R_6 = 17$ ，切割方案 $6 = 6$ （无切割）
 $R_7 = 18$ ，切割方案 $7 = 1 + 6$ 或 $7 = 2 + 2 + 3$
 $R_8 = 22$ ，切割方案 $8 = 2 + 6$
 $R_9 = 25$ ，切割方案 $9 = 3 + 6$
 $R_{10} = 30$ ，切割方案 $10 = 10$ （无切割）

更一般地，对于 R_n ($n \geq 1$)，我们可以用更短的钢条的最优切割收益来描述它：

$$R_n = \max(P_n, R_1 + R_{n-1}, R_2 + R_{n-2}, \dots, R_{n-1} + R_1)$$

首先将钢条切割为长度为*i*和*n - i*两段，接着求解这两段的最优切割收益 R_i 和 R_{n-i}

（每种方案的最优收益为两段的最优收益之和），由于无法预知哪种方案会获得最优收益，我们必须考察所有可能的*i*，选取其中收益最大者。如果直接出售原钢条会获得最大收益，我们当然可以选择不做任何切割。

分析到这里，假设现在出售8英寸的钢条，应该怎么切割呢？

下面给出三种方法：

- 1、自顶向下；
- 2、带备忘的自顶向下
- 3、由底向上

其中由底向上最简单也最高效。大家可以尝试将钢管长度设大些，比较不同方法之间的运行速度差异。通过对比，可以加深对动态规划的思想的了解与记忆。

给出例子：对于自顶向下钢条收益按照上述要求，那么可以计算钢条长度在30以下。（效率很低）

对于带备忘的自顶向下，可以计算钢条长度在10000以内。

对于自底向上，可以计算钢条长度在30000以内。（上述范围仅供参考）

代码：

```
1 #include <bits/stdc++.h>
2 #define min_num -100010
3 #define max_size 100010
4 int p[max_size];
5 int r[max_size];
6 using namespace std;
7 //*****自顶向下递归实现*****
8 double cut_rod(int p[],int n){
9     if(n==0) return 0;
10    double q=min_num;
11    for(int i=1;i<=n;i++){
12        q=max(q,p[i]+cut_rod(p,n-i));
13    }
14    return q;
15 }
16 //*****带备忘的自顶向下*****//
17 int memoized_cut_rod_aux(int p[],int n,int r[]){
18     int q=0;
19     if(r[n]>=0)
20         return r[n];
21     if(n==0)
22         q=0;
23     else q=min_num;
24     for(int i=1;i<=n;i++){
25         q=max(q,p[i]+memoized_cut_rod_aux(p,n-i,r));
26     }
27     r[n]=q;
28     return q;
29 }
30
31 int memoized_cut_rod(int p[],int n){
32     for(int i=0;i<=n;i++){
33         r[i]=min_num;
34     }
35     return memoized_cut_rod_aux(p,n,r);
36 }
37
38 //*****自底向上的算法*****//
39 int bottom_up_cut_rod(int p[],int n){
40     for(int i=0;i<=n;i++){
41         r[i]=0;
42     }
43     int q;
44     for(int j=1;j<=n;j++){
45         q=min_num;
46         for(int i=1;i<=j;i++){
47             q=max(q,p[i]+r[j-i]);
48         }
49         r[j]=q;
50     }
51     return r[n];
52 }
53 //*****测试函数*****//
54 int main(){
55     int num;
56     cout<<"请输入钢条收益num:"<<endl;
57     cin>>num;
58     for(int i=1;i<=num;i++){
59         scanf("%d",&p[i]);
60     }
61     int n;
62     cout<<"请输入钢条长度:\n";
63     while(~scanf("%d",&n)){
64         //cout<<"最大收益是:"<<cut_rod(p,n)<<endl; //自顶向下的递归算法
```



```
65         //cout<<"最大收益是:"<<memoized_cut_rod(p,n)<<endl;//带备忘的自顶向下
66         cout<<"最大收益是:"<<bottom_up_cut_rod(p,n)<<endl;//自底向上的算法
67         cout<<"请输入钢条长度:\n";
68     }
69 }
```

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4986169>>

哈夫曼树及解码

2016年5月8日 22:41

添加上解码。

解码要求：

根据输入的01字符串输出相对应的字符。

解码过程：

(1) `node *p`, `p`作为移动指针，在已经构造好的哈夫曼树中进行移动。移动规则，遇到0向左子树移动，遇到1向右子树移动。

(2) 输入01字符串`s`（可以用`string`也可以用`char`数组，在此使用`string`），求出串的长度`s.size()`。

(3) 进入循环,进行相应判断以及输出。关键代码：

```
for(int i=0;i<lens;i++)
{
    if(s[i]=='0')
    {
        if(p->lchild!=NULL)
        {
            p=p->lchild;
        }
    }
    else if(s[i]=='1')
    {
        if(p->rchild!=NULL)
        {
            p=p->rchild;
        }
    }
    if(p->lchild==NULL&& p->rchild==NULL)
    {
        printf("%c",p->ch);
        p=tree;
    }
}
```

给出全部代码以及运行实例：



View Code

```
7
e 1
i 1
l 1
o 1
u 1
v 1
y 1
Huffman code:
u:00 e:010 l:011 v:100 y:101 i:110 o:111
11001111110001010111100
iloveyou
```

输入:

首先输入要构造的哈夫曼树有多少的元素。

然后输入每个元素以及其出现的频率（上面全部设为1）

然后输入01串，对其按照上面哈夫曼树的规则进行解码。

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4986116>>

双“11”的抉择

2016年5月8日 22:41

题目描述

把钱花完了，所以单身了，单身了所以过双“11”，过双“11”所以把钱花完了。

今年Nova君(三号)照旧过着他暗无天日的“买买买”的双“11”，然而因为囊中羞涩，并不能够太任性。他的购物车中，列满了数不清的商品，共有 N 件，好多商品居然还不止一件
__(:3」∠)_ 现在Nova君要做出一个艰难的抉择， he 要从所有商品中挑出 m 件拼成一个订单，请问有多少种凑单的方法呢？求方法数对 M 的余数。

PS:同一种商品不作区分。

输入

多组测试数据（不超过100组）

每组数据两行，第一行为三个正整数 N, m, M ，具体意义详见描述，第二行为 N 个正整数 a_1, a_2, \dots, a_n ，代表第 i 个商品的个数

($1 \leq N, a_i, m \leq 1000, 2 \leq M \leq 10000$)

输出

对于每组数据，输出一行，表示方法总数

输入样例

```
3 3 10000
1 2 3
```

输出样例

```
6
```

解题思路：

关键词：多重集组合数

1、为了拉不重复计算，最好是同一种物品一次性解决掉，所以定义如下：

$dp[i+1][j]$ ：从前 $i+1$ 种物品中取出 j 个的方案总数

2、状态转移方程：

前 $i+1$ 种物品取出 j 个 = 前 $i+1$ 种物品取出 $j-1$ 个 + 前 i 种物品取出 j 个 - 前 i 种物品中取出 $j-1-a_i$ 个。

因为取出 $j-1-a_i$ 个，最后需要 $j-1$ 个，则 a_i 需要全部取出，前两个相重复，则必然全部取出。

递推公式： $dp[i+1][j] = dp[i+1][j-1] + dp[i][j] - dp[i][j-1-a_i]$ 时间复杂度 $O(nm)$ 。

呈上代码：

```
#include <bits/stdc++.h>
using namespace std;
const int N=1010;
int a[N];
int b[N][N];
int main()
{
    int n,m,M;
    while(scanf("%d%d%d",&n,&m,&M)==3)
    {
        for(int i = 0; i < n; i++)
```

```

        scanf("%d",&a[i]);
for(int i = 0; i <= n; i++)
    b[i][0] = 1;
for(int i = 0; i < n; i++)
    for(int j = 1; j <= m; j++)
    {
        if(j - 1 - a[i] >= 0)
            b[i+1][j] = (b[i][j] + b[i+1][j-1] - b[i][j-1-a[i]] + M) % M;
        else
            b[i+1][j] = (b[i][j] + b[i+1][j-1]) % M;
    }
    printf("%d\n",b[n][m]);
}
}

```

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4978945>>

矩阵链相乘助教版标准代码

2016年5月8日 22:42

题目描述

零崎有很多朋友，其中有一个叫jhljx。

jhljx大家很熟悉了，他数学不好也是出了名的，大家都懂。

现在jhljx遇到了矩阵乘法，他当时就懵了。数都数不清的他，矩阵乘法怎么可能会算的清楚呢？虽然零崎觉得还不如让你们来算，不过好歹也要给jhljx个面子，给她留下一个证明自己数学实力的机会。为了减小jhljx的计算量，让他赶快算出不正确答案来（估计他算上几遍都不一定能出一个正确答案），零崎请你们帮助jhljx。

输入

多组输入数据。

每组数据以N开始，表示矩阵链的长度。接下来一行N+1个数表示矩阵的行/列数。

$1 \leq N \leq 300$

输出

对于每组样例，输出一行最少运算次数的方案，每两个矩阵相乘都用“()”括起来，详见样例

如果存在多种解决方案,最终输出结果选取先计算左边的矩阵，详见Hint

输入样例

```
3
10 30 5 60
3
10 20 5 4
```

输出样例

```
((A1A2)A3)
((A1A2)A3)
```

Hint

对于输入的第二组数据，

如果计算顺序为((A1A2)A3)，结果为 $10 \times 20 \times 5 + 10 \times 5 \times 4 = 1200$ ，

如果计算顺序为A1(A2A3)，结果为 $20 \times 5 \times 4 + 10 \times 20 \times 4 = 1200$

那么输出结果选取第一个

解题思路：

1、课堂上讲过的经典动态规划题：矩阵链乘问题（MCM）

给定n个矩阵 $\{A_1, A_2, \dots, A_n\}$ ，其中 A_i 与 A_{i+1} 是可乘的， $i=1, 2, \dots, n-1$ 。如何确定计算矩阵连乘积的计算次序，使得依此次序计算矩阵连乘积需要的数乘次数最少。

解答:我们按照动态规划的几个步骤来分析：

（1）找出最优解的性质，刻画其特征结构

对于矩阵连乘问题，最优解就是找到一种计算顺序，使得计算次数最少。

令 $m[i][j]$ 表示第i个矩阵至第j个矩阵这段的最优解。

将矩阵连乘积 简记为 $A[i:j]$ ，这里 $i \leq j$ 。假设这个最优解在第 k 处断开， $i \leq k < j$ ，则 $A[i:j]$ 是最优的，那么 $A[i,k]$ 和 $A[k+1:j]$ 也是相应矩阵连乘的最优解。可以用反证法证明之。这就是最优子结构，也是用动态规划法解题的重要特征之一。

(2) 建立递归关系

假设计算 $A[i:j]$ ， $1 \leq i \leq j \leq n$ ，所需要的最少乘次数 $m[i,j]$ ，则原问题的最优值为 $m[1,n]$ 。

当 $i=j$ 时， $A[i,j]=A_i$ ， $m[i,j]=0$ ；(表示仅一个矩阵，如 A_1 ，没有和其他矩阵相乘，故乘的次数为0)

当 $i < j$ 时， $m[i,j] = \min\{m[i,k] + m[k+1,j] + p_{i-1} * p_k * p_j\}$ ，其中 $i \leq k < j$

(相当于对 $i \sim j$ 这段，把它分成2段，看哪种分法乘的次数最少，如 A_1, A_2, A_3, A_4 ，则有3种分法： $\{A_1\} \{A_2 A_3 A_4\}$ 、 $\{A_1 A_2\} \{A_3 A_4\}$ 、 $\{A_1 A_2 A_3\} \{A_4\}$ ，其中 $\{\}$ 表示其内部是最优解，如 $\{A_1 A_2 A_3\}$ 表示是 $A_1 A_2 A_3$ 的最优解)，

也即（ k 为分隔点）：

(3) 计算最优值

对于 $1 \leq i \leq j \leq n$ 不同的有序对 (i,j) 对于不同的子问题，因此不同子问题的个数最多只有 $O(n^2)$

但是若采用递归求解的话，许多子问题将被重复求解，所以子问题被重复求解，这也是适合用动态规划法解题的主要特征之一，这也是为什么很多人RE的原因，递归过多导致爆函数栈。

用动态规划算法解此问题，可依据其递归式以自底向上的方式进行计算。在计算过程中，保存已解决的子问题答案。每个子问题只计算一次，而在后面需要时只要简单查一下，从而避免大量的重复计算，最终得到多项式时间的算法。

呈上代码：

```
1 #include <stdio>
2 #include <string>
3 long long m[311][311];
4 long long p[311];
5 int s[311][311];
6 #define INF 999999999
7 void print(int i,int j)
8 {
9     if(i==j)
10         printf("A%d",i);
11     else{
12         printf("(");
13         print(i,s[i][j]);
14         print(s[i][j]+1,j);
15         printf(")");
16     }
17 }
18 int main() {
19     long long n,t;
20     int j;
21     while(~scanf("%lld",&n)){
22         for(int i=1;i<=n;i++){
23             m[i][i]=0;
24             for(int i=0;i<=n;i++){
25                 scanf("%lld",&p[i]);
26             }
27             for(int l=2;l<=n;l++){
28                 for(int i=1;i<=n-l+1;i++){
```

```
28         j = i+1-1;
29         m[i][j]=INF;
30         for(int k=i;k<=j-1;k++){
31             t = m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
32             if(t<=m[i][j])
33             {
34                 m[i][j]=t;
35                 s[i][j]=k;
36             }
37         }
38     }
39     print(1,n);
40     printf("\n");
41 }
42 }
```

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4978932>>

abs()函数的返回值问题

2016年5月8日 22:42

转载原文地址：

<http://www.cnblogs.com/webary/p/4967868.htm>

！

在牛客网看到一道关于abs()函数返回值的题目，见下图，当时还没反应过来，第一反应是：自从我开始学C语言，就知道它是用来求int数的绝对值的，返回值当然是0或者正数啊，一看答案就是A。

The screenshot shows a question on the NowCoder platform. The question is titled "math.h的abs返回值()" and is a multiple-choice question. The options are: A. 不可能是负数 (Not possible to be negative), B. 不可能是正数 (Not possible to be positive), C. 都有可能 (Both possible), and D. 不可能是0 (Not possible to be 0). The question is from a C/C++ category, has a difficulty of 2 stars, and has 4 answers, 2 collections, and 604 views. The question is part of a set of questions about the abs() function. The question is also part of a set of questions about the abs() function. The question is also part of a set of questions about the abs() function.

后来想来想去，质问自己 难道这道题就这么简单？于是果断先查函数库，得到：

```
#include <stdlib.h> //或math.h
int abs( int num );
```

发现库函数的返回值形式都写的是int，为什么是int？经过深入的思考、验证和查阅资料，最后得出：

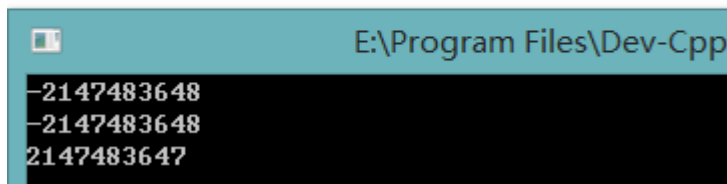
当num为0或正数时，函数返回num值；

当num为负数且不是最小的负数时（即0x80000000），函数返回num的对应绝对值数，即将内存中该二进制位的符号位取反，并把后面数值位取反加一；

当num为最小的负数时，由于正数里int类型32位表示不了这个数的绝对值，所以依然返回该负数。

贴上代码运行结果：

```
int a = 0x80000000, b = 0x7fffffff;
cout<<a<<endl;
cout<<abs(a)<<endl<<abs(b)<<endl;
```



```
E:\Program Files\Dev-Cpp
-2147483648
-2147483648
2147483647
```

也就是说，对于普通程序员来说，直接用`abs`来求一个数的绝对值并没有想象中那么安全，如果恰好给它传递了一个最小的负数作为参数，得到的依然是这个负数，而不是它的绝对值，也许后面将会发生各种意想不到的情况。因此，每一位程序员都应该明白：**调用函数后检查函数返回值是一个优秀程序员应该具备的基本素养。**

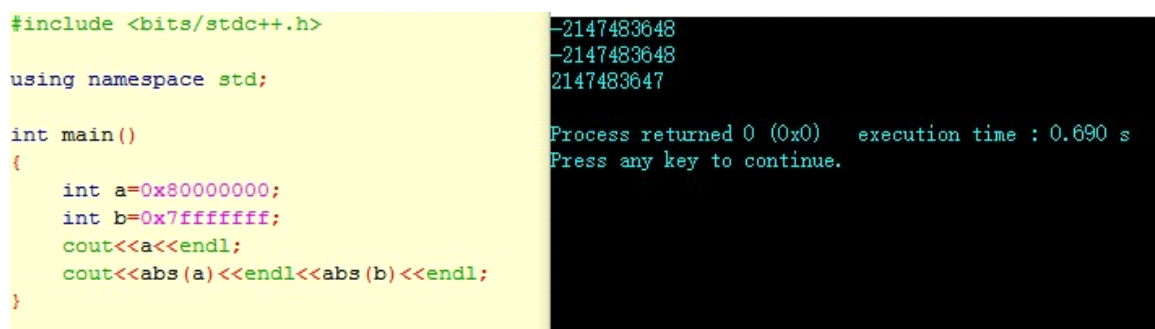
“经常反问：这个变量或表达式会上溢或下溢吗？”（《编程精粹—Microsoft编写优质无错C程序秘诀》P80, Steve Maguire 著）

引用一段来自一个博客的话作为结束：

ANSI C标准规定了每种整数类型的最小值域(但没有规定它们必须采用哪种编码方案)，并要求所有的C语言实现都要在`limits.h`头文件中通过诸如`INT_MIN`、`INT_MAX`这样的宏来指定该实现中整型数据的实际值域，而且这些实际的值域一定不能比标准规定的最小值域还要小(也即要求每种实现在`limits.h`中定义的宏的绝对值不小于C标准规定的同名宏的绝对值，并且正负号要保持一致)。

标准定义的`INT_MIN`是 $-(2^{15} - 1)$ ，`INT_MAX`是 $(2^{15} - 1)$ ，换句话说，标准保证了`int`型数据至少能表示 $-(2^{15} - 1)$ 到 $(2^{15} - 1)$ 这样一个对称区间内的所有整数，因此程序员可以放心大胆地用`int`变量存储以上范围内的任何整数。但与此同时请特别注意，用`int`变量表达超出上述范围的数将是一件没有法律(标准即是程序员的法律)保障的事情，所以不应该想当然地认为 -32768 (即 -2^{15})一定可以用`int`型来表达，尽管它确实位于用二次补码记法(`twos-complement notation`)进行编码的16位整数的值域内。熟悉以上背景后再回顾`abs`函数的问题就会发现，实际上该函数对于标准规定范最小值域内的所有整数都能正常工作，而上面提到的引起错误的输入数据已经不在此范围内了，所以此错误不应由`abs`而应由函数调用者负责。由此可见，为了安全以及可移植性，将表达式欲表示的值严格限制在标准指定的最小值域内是一个良好的编程习惯。

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main()
6 {
7     int a=0x80000000;
8     int b=0x7FFFFFFF;
9     cout<<a<<endl;
10    cout<<abs(a)<<endl<<abs(b)<<endl;
11 }
```



```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int a=0x80000000;
    int b=0x7FFFFFFF;
    cout<<a<<endl;
    cout<<abs(a)<<endl<<abs(b)<<endl;
}

-2147483648
-2147483648
2147483647

Process returned 0 (0x0)   execution time : 0.690 s
Press any key to continue.
```

来自 <<http://i.cnblogs.com/EditArticles.aspx?postid=4970547>>

值得用一首歌时间来阅读的文章

2016年5月8日 22:42

One Song,One Article!

We have already learned how to use the algorithm of quick_sort ,however ,when we want to complete something else,we need and have to turn back and rewrite the code again. Therefore, be familiar with the quick_sort's main thought and be skilled with coding are vital to everyone especially you who are programmer. So, I just want to give the code of quick_sort for you and me.

Do you want some changes?

Or just use the sort(A),sort(B),sort(blablaba);

Here I give the code which comes from the famous book---Introduction to algorithm.

Enjoy it with the background music!

```
1 #include <bits/stdc++.h>
2 #define max_size 100010
3 int A[max_size];
4
5 using namespace std;
6 int PARTITION(int A[],int p,int r)
7 {
8     int x=A[r];
9     int i=p-1;
10    for(int j=p; j<=r-1; j++)
11    {
12        if(A[j]<=x)
13        {
14            i++;
15            swap(A[i],A[j]);
16        }
17    }
18    swap(A[i+1],A[r]);
19    return i+1;
20 }
21
22 int RANDOMIZED_PARTITION(int A[],int p,int r)
23 {
24     int i=rand()%(r-p+1)+p;
25     swap(A[r],A[i]);
26     return PARTITION(A,p,r);
27 }
28 void RANDOMIZED_QUICKSORT(int A[],int p,int r)
29 {
30     if(p<r)
31     {
32         int q=RANDOMIZED_PARTITION(A,p,r);
33         RANDOMIZED_QUICKSORT(A,p,q-1);
34         RANDOMIZED_QUICKSORT(A,q+1,r);
35     }
36 }
37
38 int RANDOMIZED_SELECT(int A[],int p,int r,int i)
39 {
40     if(p==r)
```

```

41     {
42         return A[p];
43     }
44     int q=RANDOMIZED_PARTITION(A,p,r);
45     int k=q-p+1;
46     if(i==k) return A[q];
47     else if(i<k)
48         return RANDOMIZED_SELECT(A,p,q-1,i);
49     else return RANDOMIZED_SELECT(A,q+1,r,i-k);
50 }
51
52 int main()//快排程序加第i顺序统计量
53 {
54     int n;
55     cout<<"请输入数组长度:"<<endl;
56     while(~scanf("%d",&n))
57     {
58         memset(A,0,sizeof(A));
59         for(int i=1; i<=n; i++)
60         {
61             scanf("%d",&A[i]);
62         }
63         cout<<"快排结果:"<<endl;
64         RANDOMIZED_QUICKSORT(A,1,n);
65         for(int i=1; i<=n; i++)
66         {
67             cout<<A[i]<<" ";
68         }
69         int t;
70         cout<<"\n请输入查询次数"<<endl;
71         cin>>t;
72         while(t-->0)
73         {
74             int key;
75             cout<<"请输入查找第i小的数"<<endl;
76             cin>>key;
77             int result=RANDOMIZED_SELECT(A,1,n,key);
78             cout<<"第"<<key<<"小的数是:"<<result<<endl;
79         }
80     }
81     cout<<"Program over!"<<endl;
82     return 0;
83 }

```

I just learn how to add background music, there are still many bugs which are waiting to debugs! Choose enjoy it or you can just turn off it.

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4970527>>

优先队列的使用

2016年5月8日 22:43

windows live writer

终于安装好了windows live writer.(不知道什么问题一直出问题)。简直完美！

试一下的确很爽，先来一道优先队列压压惊！

题目描述

最近，Nova君遇到了一件非常棘手的问题。他需要整理非常多的解题报告。每份解题报告的题目数量是不定的。Nova君每次需要将两份报告的题目解析合成到一份里。假设两份报告的题解数分别为a和b，那么合成这两份报告消耗Nova君a+b的hp值。现在有n份报告，题解数分别为 $a_0, a_1, a_2, \dots, a_{n-1}$ ，请问Nova最少消耗多少hp？

输入

多组测试数据。每组数据，第一行为一个正整数n（INT范围内），代表报告份数，接下来一行，包含n个正整数（INT范围内） $a_0, a_1, a_2, \dots, a_{n-1}$ ，代表每份报告的题解数。

输出

对于每组数据，输出一行，代表Nova需要付出的最少的hp值

输入样例

```
4
3 5 7 11
```

输出样例

```
49
```

Hint

陈题，然而请用优先队列实现

来源： <<http://biancheng.love/contest/10/problem/A/index>>

代码实现：

```
1 #include <bits/stdc++.h>
2 #define max_size 1000010
3
4 using namespace std;
5 int a[max_size];
6
7 struct cmp{
8     bool operator()(int &a,int &b){
9         return a>b;
10    }
11 };
12
13 int main(){
14     priority_queue<int,vector<int>,cmp>q;
15     int n;
```

```
16     while (~scanf ("%d", &n)) {
17         long long ans=0;
18         int num;
19         for (int i=1; i<=n; i++) {
20             scanf ("%d", &num);
21             q.push (num);
22         }
23         int key, a, b;
24         while (!q.empty ()) {
25             a=q.top ();
26             q.pop ();
27             if (q.empty ()) break;
28             b=q.top ();
29             q.pop ();
30             ans+=a+b;
31             key=a+b;
32             q.push (key);
33         }
34         cout<<ans<<endl;
35     }
36 }
```

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4966857>>

scanf 用法及陷阱（转）

2016年5月8日 22:43

函数名: scanf

功 能: 执行格式化输入

用 法: int scanf(char *format[,argument,...]);

scanf()函数是通用终端格式化输入函数，它从标准输入设备(键盘)读取输入的信息。可以读入任何固有类型的数据并自动把数值变换成适当的机内格式。

其调用格式为: scanf("<格式化字符串>", <地址表>);

scanf()函数返回成功赋值的数据项数，出错时则返回EOF。

其控制串由三类字符构成:

1. 格式化说明符;

2. 空白符;

3. 非空白符 ;

(A)	格式化说明符
格式字符	说明
%a	读入一个浮点值(仅C99有效)
%A	同上
%c	读入一个字符
%d	读入十进制整数
%i	读入十进制，八进制，十六进制整数
%o	读入八进制整数
%x	读入十六进制整数
%X	同上
%C	读入一个字符
%s	读入一个
%f	读入一个浮点数
%F	同上
%e	同上
%E	同上
%g	同上
%G	同上
%p	读入一个指针
%u	读入一个无符号十进制整数
%n	至此已读入值的等价字符数
%[]	扫描字符集合
%%	读%符号
/*	指定类型的数据但不保存

比如:

百分号(%)与格式符之间的星号(*)表示读指定类型的数据但不保存。因此,

scanf("%d %*c %d", &x, &y);

对 10/20 的读入操作中, 10 放入变量 x, 20 放入 y。

附加格式说明字符表

修饰符	说明
L/l 长度修饰符	输入"长"数据
h 长度修饰符	输入"短"数据
W 整型常数	指定输入数据所占宽度
* 星号	空读一个数据
hh,ll 同上h,l但仅对C99有效。	

(B) 空白字符

空白字符会使scanf()函数在读操作中略去输入中的一个或多个空白字符，空白符可以是space,tab,newline等等，直到第一个非空白符出现为止。

(C) 非空白字符

一个非空白字符会使scanf()函数在读入时剔除掉与这个非空白字符相同的字符。但在输入时必须输入这些字符。否则就会出错。

注：scanf()控制串知识就介绍到这里（应该比较齐全了^_^），如有遗漏下次补上。下面将结合实际例程，一一阐述。

三、scanf()函数的控制串的使用

例1.

```
#include "stdio.h"
int main(void)
{
    int a,b,c;

    scanf("%d%d%d",&a,&b,&c);
    printf("%d,%d,%d\n",a,b,c);
    return 0;
}
```

运行时按如下方式输入三个值：

3□4□5 ✓（输入a,b,c的值）

3, 4, 5（printf输出的a, b, c的值）

(1) &a、&b、&c中的&是地址运算符，分别获得这三个变量的内存地址。

(2) "%d%d%d"是按十进值格式输入三个数值。输入时，在两个数据之间可以用一个或多个空格、tab键、回车键分隔。

以下是合法输入方式：

① 3□□4□□□5✓

② 3✓

4□5✓

③ 3（tab键）4✓

5✓

例2.

```
#include "stdio.h"
int main(void)
{
    int a,b,c;
    scanf("%d,%d,%d",&a,&b,&c);
    printf("%d,%d,%d\n",a,b,c);
    return 0;
}
```

运行时按如下方式输入三个值：

3,4,5 ✓（输入a,b,c的值）

或者

3,□4,□5 ✓（输入a,b,c的值）

3,□□□4,□5 ✓（输入a,b,c的值）

.....

都是合法的，但是","一定要跟在数字后面，如：

3□, 4,□5 ✓就非法了，程序出错。（解决方法与原因后面讲）

再如：

1、scanf()中的变量必须使用地址。

```
int a, b;
scanf("%d%d",a,b); //错误
scanf("%d%d",&a,&b);
```

2、scanf()的格式控制串可以使用其它非空白字符，但在输入时必须输入这些字符。

例：

```
scanf("%d,%d",&a,&b);
```

输入： 3, 4 ✓（逗号与"%d,%d"中的逗号对应）

```
scanf("a=%d,b=%d",&a,&b);
```

输入: a=3, b=4 ✓ ("a=", "b=", 逗号与"%d,%d"中的"a=", "b="及逗号对应)

3、在用"%c"输入时, 空格和“转义字符”均作为有效字符。

例:

```
scanf("%c%c%c%c",&c1,&c2,&c3);
```

输入: a□b□c✓

结果: a→c1, □→c2, b→c3 (其余被丢弃)

scanf()函数接收输入数据时, 遇以下情况结束一个数据的输入: (不是结束该scanf函数, scanf函数仅在每一个数据域均有数据, 并按回车后结束)。

- ① 遇空格、“回车”、“跳格”键。
- ② 遇宽度结束。
- ③ 遇非法输入。

问题二: scanf()函数不能正确接受有空格的字符串? 如: I love you!

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    char str[80];
```

```
    scanf("%s",str);
```

```
    printf("%s",str);
```

```
    return 0;
```

```
}
```

输入: I live you!

输出: I

scanf()函数接收输入数据时, 遇以下情况结束一个数据的输入: (不是结束该scanf函数, scanf函数仅在每一个数据域均有数据, 并按回车后结束)。

- ① 遇空格、“回车”、“跳格”键。
- ② 遇宽度结束。
- ③ 遇非法输入。

所以, 上述程序并不能达到预期目的, scanf()扫描到"I"后面的空格就认为对str的赋值结束, 并忽略后面的"love you!".这里要注意是"love you!"还在键盘缓冲区(关于这个问题, 网上我所见的说法都是如此, 但是, 我经过调试发现, 其实这时缓冲区字符串首尾指针已经相等了, 也就是说缓冲区清空了, scanf()函数应该只是扫描stdin流, 这个残存信息是在stdin中)。我们改动一下上面的程序来验证一下:

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    char str[80];
```

```
    char str1[80];
```

```
    char str2[80];
```

```
    scanf("%s",str);/*此处输入:I love you! */
```

```
    printf("%s",str);
```

```
    sleep(5);/*这里等待5秒,告诉你程序运行到什么地方*/
```

```
    scanf("%s",str1);/*这两句无需你再输入,是对键盘缓冲区再扫描 */
```

```
    scanf("%s",str2);/*这两句无需你再输入,是对键盘缓冲区再扫描 */
```

```
    printf("\n%s",str1);
```

```
    printf("\n%s",str2);
```

```
    return 0;
```

```
}
```

输入: I love you!

输出: I

love

you!

好了, 原因知道了, 那么scanf()函数能不能完成这个任务? 回答是: 能! 别忘了scanf()函数还有一个 %[] 格式控制符(如果对%[]不了解的请查看本文的上篇), 请看下面的程序:

```
#include "stdio.h"
```

```
int main()
```

```

{
    char string[50];

    /*scanf("%s",string);不能接收空格符*/
    scanf("%[^\\n]",string);
    printf("%s\\n",string);
    return 0;
}

```

问题三：键盘缓冲区残余信息问题

```
#include <stdio.h>
```

```
int main()
```

```

{
    int a;
    char c;
    do
    {
        scanf("%d",&a);
        scanf("%c",&c);
        printf("a=%d    c=%c\\n",a,c);
        /*printf("c=%d\\n",c);*/
    }while(c!='N');
}

```

scanf("%c",&c);这句不能正常接收 字符,什么原因呢? 我们用printf("c=%d\\n",c);将C用int表示出来,启用printf("c=%d\\n",c);这一句,看看 scanf()函数赋给C到底是什么,结果是 c=10 ,ASCII值为10是什么? 换行即\\n.

对了,我们每击打一下"Enter"键,向键盘缓冲区发去一个“回车”(\\r),一个“换行”(\\n),在这里\\r被scanf()函数处理掉了(姑且这么认为吧^_^),而\\n被scant()函数“错误”地赋给了c.

解决办法:可以在两个scanf()函数之后加个fflush(stdin);,

还有加getch(); getchar();也可以,但是要视具体scanf()语句加那个,这里就不分析了,读者自己去摸索吧。

但是加fflush(stdin);不管什么情况都可行。

函数名: fflush

功 能: 清除一个流

用 法: int fflush(FILE *stream);

```
#include <stdio.h>
```

```
int main()
```

```

{
    int a;
    char c;
    do
    {
        scanf("%d",&a);
        fflush(stdin);
        scanf("%c",&c);
        fflush(stdin);
        printf("a=%d    c=%c\\n",a,c);
    }while(c!='N');
}

```

这里再给一个用“空格符”来处理缓冲区残余信息的示例:

运行出错的程序:

```
#include <stdio.h>
```

```
int main()
```

```

{
    int i;
    char j;
    for(i = 0;i < 10;i++)
    {
        scanf("%c",&j);/*这里%前没有空格*/
    }
}

```

```

}
使用了空格控制符后:
#include <stdio.h>
int main()
{
    int i;
    char j;
    for(i = 0; i < 10; i++)
    {
        scanf(" %c",&j);/*注意这里%前有个空格*/
    }
}

```

可以运行看看两个程序有什么不同。

问题四 如何处理scanf()函数误输入造成程序死锁或出错?

```

#include <stdio.h>
int main()
{
    int a,b,c; /*计算a+b*/
    scanf("%d,%d",&a,&b);
    c=a+b;
    printf("%d+%d=%d",a,b,c);
}

```

如上程序，如果正确输入a,b的值，那么没什么问题，但是，你不能保证使用者每一次都能正确输入，一旦输入了错误的类型，你的程序不是死锁，就是得到一个错误的结果，呵呵，这可能所有人都遇到过的问题吧？

解决方法：scanf()函数执行成功时的返回值是成功读取的变量数，也就是说，你这个scanf()函数有几个变量，如果scanf()函数全部正常读取，它就返回几。但这里还要注意另一个问题，如果输入了非法数据，键盘缓冲区就可能还有个有残余信息问题。

正确的例程：

```

#include <stdio.h>
int main()
{
    int a,b,c; /*计算a+b*/
    while(scanf("%d,%d",&a,&b)!=2)fflush(stdin); /*很有趣的代码
    c=a+b;
    printf("%d+%d=%d",a,b,c);
}

```

例如：

```

#include<stdio.h>
int main()
{
    char ch1,ch2;
    printf("Input for ch1:\n");
    scanf("%c",&ch1);
    printf("ch1=%c\n",ch1);
    printf("Input for ch2:\n");
    scanf("%c",&ch2);
    printf("ch2=%c\n",ch2);
}

```

表面上看这段程序是没有错的，也可以运行，但运行过程中到第二个scanf输入值给ch2时，程序并不会停止下来等待你输入，而是直接运行到最后一个printf！

为什么？当时百思不得其解。。。

今天上网查了下才知道，原来scanf是从标准输入缓冲区中读取输入的数据，而%c的字符输入格式会接收回车字符，在输入第一个scanf时输入字符后按回车结束，输入缓冲中保存了这个回车符，遇到第二个scanf时，它自动把这个回车符赋给了ch2。而如果第二个scanf的输入格式不是%c时，由于格式不匹配，这个回车符会被自动忽略，所以只有在连续输入两个%c的格式时才会出现这样的问题！

解决办法：（二办法任选其一）

1、清空输入缓冲区

第一个scanf后加入语句：`fflush(stdin);` //C语言清空输入缓冲区函数

2、格式控制中加入空格

将第二个scanf改为：`scanf(" %c",&ch2);` //在%号前面加一个空格

scanf格式输入时要求输入格式与格式控制符中的完全一样(如：`scanf("abcd%c",&ch);`输入时必须输入abcde,ch得到的值为e)空格可以抵消前面输入的回车符。

总结：可见在scanf函数本身是陷阱重重呀，出现下面的情况的时候要格外小心，i

1, 多个scanf函数依次出现的时候

```
scanf("%c",&ch1);
printf("ch1=%c\n",ch1);
printf("Input for ch2:\n");
scanf("%c",&ch2);
```

2, 当scanf函数输入字符或者输入字符串的时候

```
scanf("%s",str);
```

3, 当scanf函数的输入控制格式中 有多个输入变量的时候

```
scanf("%5d %5d %c %c%f%f*f%f",&a,&b,&c1,&c2,&x,&y,&z);
```

在这个例子中，就要求如下输入才有效：5 9 f e 6.888 7.99*f6.886

当在一个scanf函数中出现多个 %c 即字符输入的时候 最好用分隔符（空格 或 ， ）隔开，以避免把一些不想要的空格或回车读入。

来自 <<http://i.cnblogs.com/EditArticles.aspx?postid=4966537>>

0-1背包再修改版

2016年5月8日 22:44

题目描述

把钱花完了，所以单身了，单身了所以过双“11”，过双“11”所以把钱花完了。

今年Nova君(三号)照旧过着他暗无天日的“买买买”的双“11”，然而因为囊中羞涩，并不能够太任性。他的购物车中，列满了数不清的商品，共有 N 件，好多商品居然还不止一件
__(:3 ∠)_ 现在Nova君要做出一个艰难的抉择，他要从所有商品中挑出 m 件拼成一个订单，请问有多少种凑单的方法呢？求方法数对 M 的余数。

PS:同一种商品不作区分。

输入

多组测试数据（不超过100组）

每组数据两行，第一行为三个正整数 N, m, M ，具体意义详见描述，第二行为 N 个正整数 a_1, a_2, \dots, a_n ，代表第 i 个商品的个数

($1 \leq N, a_i, m \leq 1000, 2 \leq M \leq 10000$)

输出

对于每组数据，输出一行，表示方法总数

输入样例

```
3 3 10000
1 2 3
```

输出样例

```
6
```

题目来源: <http://biancheng.love/contest/17/problem/G/index>

详情见代码以及注释:

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 const int N=1010;
5 int a[N];
6 int b[N][N];
7 int main()
8 {
9     int n,m,M;
10    while(scanf("%d%d%d",&n,&m,&M)==3)
11    {
12        for(int i = 0; i < n; i++)
13            scanf("%d",&a[i]); //每种商品的件数
14
15        for(int i = 0; i <= n; i++)
16            b[i][0] = 1;
17
18        for(int i = 0; i < n; i++) //n种商品
19            for(int j = 1; j <= m; j++) //挑出m件
20            {
21                if(j - 1 - a[i] >= 0) //是否大于第i中商品件数
22                    b[i+1][j] = (b[i][j] + b[i+1][j-1] - b[i][j-1-a[i]] + M) % M;
23                else
24                    b[i+1][j] = (b[i][j] + b[i+1][j-1]) % M;
25            }
26        printf("%d\n",b[n][m]);
27    }
28
29 }
```

另附背包的一些模板函数:

```
1 void ZoreOnePack(int cost , int weight)//0-1背包
2 {
3     for (int i = W ; i >= weight ; -- i)
4         f[i] = max(f[i],f[i-weight]+cost) ;
5 }
6
7 void CompletePack(int cost , int weight)//完全背包
8 {
9     for (int i = weight ; i <= W ; ++ i)
10         f[i] = max(f[i],f[i-weight]+cost) ;
11 }
12
13 void MultiPack(int cost , int weight , int num)//多重背包
14 {
15     if (num*weight>=W)
16         CompletePack(cost,weight) ;
17     else
18     {
19         int k = 1 ;
20         while (k<num)
21         {
22             ZoreOnePack(cost*k,weight*k) ;
23             num -= k ;
24             k += k ;
25         }
26         ZoreOnePack(cost*num,weight*num) ;
27     }
28 }
29
30 void TwoZoreOnePack(int cost , int weight , int many)
31 {
32     for (int i = W ; i >= weight ; -- i)
33         for (int j = M ; j >= many ; -- j)
34             two[i][j] = max(two[i][j],two[i-weight][j-many]+cost) ;
35 }
36
37 void TwoCompletePack(int cost ,int weight ,int many)
38 {
39     for (int i = weight ; i <= W ; ++ i)
40         for (int j = many ; j <= M ; ++ j)
41             two[i][j] = max(two[i][j],two[i-weight][j-many]+cost) ;
42 }
43
44 void TwoMultiPack(int cost ,int weight , int many , int num)
45 {
46     if (num*weight>=W&&num*many>=M)
47         TwoCompletePack(cost,weight,many) ;
48     else
49     {
50         int k = 1 ;
51         while (k<num)
52         {
53             TwoZoreOnePack(cost*k,weight*k,many*k) ;
54             num -= k ;
55             k += k ;
56         }
57         TwoZoreOnePack(cost*num,weight*num,many*num) ;
58     }
59 }
```

LCS修改版（Longest Common Subsequence 最长公共子序列）

2016年5月8日 22:44

题目描述

作为一名情报局特工，Nova君(2号)有着特殊的传达情报的技巧。为了避免被窃取情报，每次传达时，他都会发出两句旁人看来意义不明话，实际上暗号已经暗含其中。解密的方法很简单，分别从两句话里删掉任意多个字母，使得两句话剩余的部分相同，通过一定的删除手法，可以让剩余的部分相同且长度最大，就得到了可能的暗号。暗号可能有多个，还要进行筛选，现在情报局人手不够，希望你能助一臂之力，筛选工作不用你完成，你只需计算出暗号长度以及个数即可。（注意，字母的位置也是暗号的重要信息，位置不同的字母组成的暗号不算同一种，详见样例）

输入

多组测试数据（组数小于20）

每组数据输入两行，分别为两个字符串（只含英文字母，无空格），每个字符串以"."结束

输出

对于每组数据，输出两行，第一行为暗号的长度，第二行为暗号的个数（答案可能很大，对个数100000000求模）

输入样例

AAAA.

AA.

输出样例

2

6

题目来源: <http://biancheng.love/contest/17/problem/F/index>

最长公共子序列的实现可参考:

<http://www.cnblogs.com/huangxincheng/archive/2012/11/11/2764625.html>

和之前的不同在于需要计算出最长公共子序列一共有多少个！翻看不少博客很少有提到计算最长公共子序列的个数问题

下面给出代码实现:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int m=100000000;//对m求模
5 int f[2][5001]= {0},g[2][5001]= {0};
6
7 int main()
8 {
9     string s1,s2;
10    while(cin>>s1>>s2)
11    {
12        memset(f,0,sizeof(f));
13        memset(g,0,sizeof(g));
14        int len1=s1.size()-1,len2=s2.size()-1;//串s1,s2长度
15
16        for(int i=0; i<=len2; i++)
17            g[0][i]=1;
18        int k;
19        for(int i=1; i<=len1; i++)
20        {
21            k=i & 1;//与运算 当i是奇数时k=1,当i是偶数是k是0
22            memset(g[k],0,sizeof(g[k]));
23            memset(f[k],0,sizeof(f[k]));
24            g[k][0]=1;
```



```

25     g[!k][0]=1;
26     for(int j=1; j<=len2; j++)
27     {
28         if(s1[i-1]==s2[j-1])
29         {
30             f[k][j]=f[!k][j-1]+1;
31             g[k][j]=g[!k][j-1];
32             g[k][j]%=m;
33             if(f[k][j]==f[!k][j])
34             {
35                 g[k][j]+=g[!k][j];
36                 g[k][j]%=m;
37             }
38             if(f[k][j-1]==f[k][j])
39             {
40                 g[k][j]+=g[k][j-1];
41                 g[k][j]%=m;
42             }
43         }
44         else
45         {
46             if(f[!k][j]>f[k][j-1])
47             {
48                 f[k][j]=f[!k][j];
49                 g[k][j]+=g[!k][j];
50                 g[k][j]%=m;
51             }
52             if(f[!k][j]<f[k][j-1])
53             {
54                 f[k][j]=f[k][j-1];
55                 g[k][j]+=g[k][j-1];
56                 g[k][j]%=m;
57             }
58             if(f[!k][j]==f[k][j-1])
59             {
60                 f[k][j]=f[!k][j];
61                 g[k][j]+=g[!k][j]+g[k][j-1];
62                 if(f[!k][j-1]==f[k][j]) g[k][j]-=g[!k][j-1];
63                 g[k][j]=(g[k][j]+3*m)%m;
64             }
65         }
66     }
67 }
68 cout<<f[k][len2]<<endl;
69 cout<<g[k][len2]<<endl;
70 }
71 }

```

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4966370>>

矩阵链相乘实例

2016年5月8日 22:44

题目描述

零崎有很多朋友，其中有一个叫jhljx。

jhljx大家很熟悉了，他数学不好也是出了名的，大家都懂。

现在jhljx遇到了矩阵乘法，他当时就懵了。数都数不清的他，矩阵乘法怎么可能会算的清楚呢？虽然零崎觉得还不如让你们来算，不过好歹也要给jhljx个面子，给她留下一个证明自己数学实力的机会。为了减小jhljx的计算量，让他赶快算出不正确答案来（估计他算上几遍都不一定能出一个正确答案），零崎请你们帮助jhljx。

输入

多组输入数据。

每组数据以N开始，表示矩阵链的长度。接下来一行N+1个数表示矩阵的行/列数。

$1 \leq N \leq 300$

输出

对于每组样例，输出一行最少运算次数的方案，每两个矩阵相乘都用“()”括起来，详见样例

如果存在多种解决方案,最终输出结果选取先计算左边的矩阵，详见Hint

输入样例

```
3
10 30 5 60
3
10 20 5 4
```

输出样例

```
((A1A2)A3)
((A1A2)A3)
```

Hint

对于输入的第二组数据，

如果计算顺序为((A1A2)A3)，结果为 $10 \times 20 \times 5 + 10 \times 5 \times 4 = 1200$ ，

如果计算顺序为A1(A2A3)，结果为 $20 \times 5 \times 4 + 10 \times 20 \times 4 = 1200$

那么输出结果选取第一个

题目来源：<http://biancheng.love/contest/17/problem/D/index>

题目注意事项：如果存在多种解决方案,最终输出结果选取先计算左边的矩阵

下面给出实现代码：

```
1 #include <bits/stdc++.h>
2 #define max_size 400
3 #define INF 1000000000
4 long long s[max_size][max_size]; //保存构造最优解信息
5 long long p[max_size]; //矩阵规模的记录
6 long long m[max_size][max_size]; //记录最优值
7
8 void matrix_chain_order(int n)
9 {
10     for(int i=1; i<=n; i++)
11     {
```

```

12     m[i][i]=0; //初始化最优值(起始于1,结束于n)
13 }
14 for(int l=2;l<=n;l++) //l表示矩阵的长度
15 {
16     for(int i=1;i<=n-l+1;i++)
17     {
18         int j=i+l-1;
19         m[i][j]=INF;
20         s[i][j]=0;
21         for(int k=j-1;k>=i;k--) //解决方案优先选取先左边的矩阵
22         {
23             int q=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
24             if(q<m[i][j])
25             {
26                 m[i][j]=q;
27                 s[i][j]=k;
28             }
29         }
30     }
31 }
32 }
33
34 void print_optimal_parents(int i,int j) //打印最优解的结果
35 {
36     if(i==j)
37         printf("A%d",i);
38     else
39     {
40         printf("(");
41         print_optimal_parents(i,s[i][j]);
42         print_optimal_parents(s[i][j]+1,j);
43         printf(")");
44     }
45 }
46
47 int main()
48 {
49     int n;
50     while(~scanf("%d",&n))
51     {
52         memset(p,0,sizeof(p));
53         for(int i=0;i<=n;i++)
54         {
55             scanf("%lld",&p[i]);
56         }
57         matrix_chain_order(n);
58         print_optimal_parents(1,n);
59         printf("\n");
60     }
61 }

```

对实现过程的详细解答请见

<http://www.cnblogs.com/Anker/archive/2013/03/10/2952475.html>

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4966354>>

0-1背包修改版

2016年5月8日 22:45

题目描述

零崎有很多朋友，其中有一个叫做lfj的接盘侠。

lfj是一个手残，他和零崎一起玩网游的时候不好好打本，天天看拍卖行，没过多久，就成为了一个出色的商人。不过再出色的投机商也有失手成为接盘侠的一天。所谓真正的接盘侠从来不给自己留活路。当lfj接盘成功之时，即分文不剩之日。

作为lfj的友人，零崎实在看不下去，于是他决定帮lfj一把。当然了，零崎肯定不会自己动手，活还得你们来干。

lfj可以提供给你们拍卖行所有能买到物品的价格和利润，还有他的本金。既然是接盘侠，就必须分文不剩。虽然零崎想让你们给出一次接盘中利润最大的购买方案，但是lfj觉得只要知道最大利润就可以了。

输入

每组数据第一行为两个整数P和N，表示本金和拍卖行物品个数。(注意：与B题不同每类物品只有一件)

接下来N行，每行两个数据pi,ci代表第i类物品的利润和购买价格。

$1 \leq P \leq 20000, 1 \leq N \leq 300, 1 \leq c, p \leq 200$

输出

对于每组数据，输出一行，为能获得的最大利润

如果不能成功接盘，则输出jpx

输入样例

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

输出样例

```
jpx
4
```

Hint

使用if直接比较不要调用max（）以防超时

题目来源：<http://biancheng.love/contest/17/problem/C/index>

解题分析：改题和0-1背包问题类似，不过要求实现将背包的整个容量V,也就是题目中的本金P,全部用尽，同时要求得到利润最大。需要对装入的每件商品进行判断，首先是能否满足耗尽本金，另外判断是否比上次的利润大。

先给出关键的代码段配合思考：

```
1     for(int i=0;i<n;i++)
2     {
3         scanf("%d%d",&pi,&ci);
4         for(int j=p; j >= ci; j--)
5             if(V[j-ci]!=-1&&V[j]<V[j-ci]+pi)
6                 V[j]=V[j-ci]+pi;
7     }
```

明白上述讲解之后，下面的输出以及输入只是按照题目要求进行。

代码实现:

```
1 #include <bits/stdc++.h>
2 #define max_size 20010
3 int V[max_size];
4
5 using namespace std;
6
7 int main()
8 {
9     int p,n,pi,ci;
10    while(~scanf("%d%d",&p,&n))
11    {
12        memset(V,-1,sizeof(V));
13        V[0]=0;
14        for(int i=0;i<n;i++)
15        {
16            scanf("%d%d",&pi,&ci);
17            for(int j=p; j >= ci; j--)
18                if(V[j-ci]!=-1&&V[j]<V[j-ci]+pi)
19                    V[j]=V[j-ci]+pi;
20        }
21        if(V[p]==-1)
22            printf("jpx\n");
23        else
24            printf("%d\n",V[p]);
25    }
26 }
```

另附0-1背包的实现代码:

```
1 void ZoreOnePack(int cost , int weight)
2 {
3     for (int i = W ; i >= weight ; -- i)
4         f[i] = max(f[i],f[i-weight]+cost) ;
5 }
```

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=4966349>>