2009 年山东大学计算机考研上机试题

【1】输入一个整数,它可以由 n(n>=2)个连续整数相加得到,输出所有可能的连续整数序列,每个序列占一行,数字之间用空格分开,数据从小到大,每列按最小元素递增顺序排列,如果找不到,输出 none

例:21=1+2+3+4+5+6

21=6+7+8

则输出 1 2 3 4 5 6

6 7 8

【2】某国设计了一种导弹防御系统,但有缺陷,导弹来袭时,第一枚炮弹可以达到任意高度,但以后的任意一炮均不能超过前一发炮弹高度。现在仅有一套这样的系统

输入:来袭的导弹数目(不超过 100 枚)

输出:1:能够拦截的导弹数目 30 分

2:若要拦截所有导弹,需要几套这样的系统 20 分

其实第一问就是找一个递减序列的长度,第二问就是找一下有几个这样的递减序列。

第一题:

```
#include<stdlib.h>
#include<iostream>
using namespace std;
//使用算法，搜索
int n;   //输入
int copyN; //保存 n 的一个副本
int length; //纪录从 start 开始的长度，例如 1,2,3,4,5，length 则为 5
int yes;      //标记
void initialize() {
      yes = 0;
      length = 0;
}
void DFS( int start ) { //从 start 开始，start+1, start+2,...
      if( n == 0 ) {    //说明 n 成功分解
            yes = 1;
      }
      else {
            if( n >= start ) {    //说明还可以继续
                        n -= start;
                        length++;
                        DFS( start + 1 );
            }
            else {     //说明 n 分解失败
                        return;
            }
      }
```

```
}
void print( int i ) {//输出 i，i+1, i+2,...
	int j;
	for( j = 0; j < length; j++ ) {
		cout << i + j << ' ';
	}
	cout << endl;
}
int main() {
	cin >> n;
	while( n > 0 ) {
		int i;
		for( i = 1; i <= n / 2; i++ ) { //对第一个数进行试探
			initialize();

			copyN = n;
			DFS( i ); //从 i 开始一直往下数
			if( yes == 1 ) {
				print( i );
			}
			n = copyN;
		}

		cin >> n;
	}
	return 0;
}
```

第二题：

```
#include<iostream>
#include<stdlib.h>
using namespace std;
//使用算法，动态规划，典型算法是求最长递增子序列，此题是求最长递减子序列，
//算法如下: m 为以 a 结尾的最长递减子序列的长度，例如数组 a 为 9 10 5 6 4 1 3
// 那么 m[5]为 4，最长递减子序列为 9 5 4 1，最长递减子序列不一定是唯一的
//m = max{ m[k](若 a > a[k])，m[k] + 1(若 a <= a[k]) } 0 <= k < i
int a[100]; //  输入的导弹的高度
int n;          //  导弹的个数
int m[100];
void initialize() {
	//初始化 m，最长递减子序列的长度至少是 1
	int i;
	for( i = 0; i < n; i++ ) {
		m = 1;
```

```
        }
    }
}
void print() {
    int i;
    for( i = 0; i < n; i++ ) {
        cout << m << ' ';
    }
    cout << endl;
}
int main() {
    cin >> n;
    int i, k;
    for( i = 0; i < n; i++ ) {
        cin >> a;
    }

    initialize();

    //与上面的动态规划算法一致
    for( i = 1; i < n; i++ ) {
        for( k = 0; k < i; k++ ) {
            if( a > a[k] ) {
                if( m[k] > m ) {
                    m = m[k];
                }
            }
            else {
                if( m[k] + 1 > m ) {
                    m = m[k] + 1;
                }
            }
        }
    }

    //解第一问，能够最多拦截的导弹数目
    int largest = m[0];
    for( i = 1; i < n; i++ ) {
        if( m > largest ) {
            largest = m;
        }
    }
    cout << largest << endl;

    //第二问没有那么简单，不能直接用上面的结果，考虑下面这个例子
```

```
    //a 为 9 5 10 6 4 1， 最长递减子序列的长度为 4，
    //方案一， 第一次去掉 9 5 4 1，第二次去掉 10 6
    //方案二， 第一次去掉 9 6 4 1，第二次去掉 5， 第三次去掉 10
    //可以看到，需要系统的个数与最长子序列的选择有关，必须选出一个好的
    //最长子序列，以达到最优，谁能做出来？

    system( "pause" );
}
```

第二题的第二小题用动态规划,这样,a[105],a[n]表示前 n 个点最少需要几个防御系统,a[1]=1,从 i=2tom(m 为导弹数目),
假如 m 个导弹的高度存在 b[105](b 表第 i 个导弹的高度),同时设个数组 c[105]表,c==-1 的话表这是在这导弹系统中高度最低的.然后从第二个开始,i=2to n,找它前面的 b<=a 且 c==-1 且最小的,如有满足的,(比如为 minj)则将最小的 c[minj]改为 0,c 改为-1.否则,c=-1,a=a[i-1]+1;
伪代码如下.

```
memset(a,0,sizeof(a));
memset(c,0,sizeof(c));
a[1]=1;
c[1]=-1;
int min1,minj;
for(i=2;i<=m;i++)
{
min1=b;
minj=-1;
for(j=1;j<i;j++)
{
if(b[j]<=min1&&c[j]==-1)
{
min1=b[j];
minj=j;
}
}
if(minj==-1)
{
c=-1;
a=a[i-1]+1;
}
else
{
c=-1;
a=a[i-1];
c[minj]=0;
}
}
```

最后 a[m]就是结果.
没法证明,但感觉应该没错.

以下是我修正了 LS 的程序。

```
#include<stdio.h>
void main()
{
        int x,y=0,n,k,j,i,u,sum=0;
        int found=0;
        scanf("%d",&x);
        u=x;
        x*=2;
        for(n=x/2;n>=2;n--)
        {
                if(x%n==0)
                        y=x/n-n+1;
                if((y>=2)&&(y%2==0))
                {
                        k=y/2;
                        sum=k;
                        for(i=1;i<=n-1;i++)
                                sum+=k+i;
                        if(sum==u)
                        {
                                for(j=0;j<=n-1;j++)
                                        printf("%5d",k+j);
                        //printf("%d\n",k+n-1);
                                printf("\n";
                                found=1;
                        }
                        sum=0;
                }
        }
        if(found==0)
                printf("none\n");

}
```