目录

A、 求阴影面积.....2
题目描述.....2
题目分析.....2
参考代码.....2
B、 请叫我简单题.....3
题目描述.....3
题目分析.....3
参考代码.....3
C、 第二长不下降子序列.....4
题目描述.....4
题目分析.....4
参考代码.....4
D、 危险的 td 5
题目描述.....5
题目分析.....5
参考代码.....6
E、 来来来来数钱.....6
题目描述.....6
题目分析.....6
参考代码.....7
F、 N-way Assemble Line.....7
题目描述.....8
题目分析.....8
参考代码.....8
G、 金字塔 9
题目描述.....9
题目分析.....9
参考代码.....9
H、 "Snake"的双 11 10
题目描述.....10
题目分析.....11
参考代码.....11

A、求阴影面积

题目描述

/*注释略*/
a*x^2+b*y^2+c*x*y+d*x+e*y<=f 为阴影方程
求在(-1<=x<=1,-1<=y<=1)区域内阴影部分的面积所占总区域的比例？

题目分析

一道求解几何概型的题目。对于此类问题，如果要求的精度高，可以从曲线方程表示的图形入手，但是这种做法对数学要求比较高，尤其是方程不是某些图形的标准方程时求解更加麻烦。所以这个题直接暴力枚举或者直接枚举，判断某点是不是在图形上就可以了……

参考代码

```
#include <cstdio>
#include <ctime>
#include <cstdlib>
int main()
{
	int a, b, c, d, e, f;
	double x, y, cnt, r;
	const int n = 8e5;
	while( ~scanf("%d%d%d%d%d%d",&a ,&b, &c, &d, &e, &f) )
	{
		cnt = 0;
		srand( time(NULL) );
		for( int i = n; i; i-- )
		{
			x = rand()%5000/2500.0-1;	 / /不能枚举到边界，但是不影响结果
			y = rand()%5000/2500.0-1;
			r = a*x*x+b*y*y+c*x*y+d*x+e*y-f;
			if(r<=0)
				cnt++;
		}
		printf( "%.2f\n", cnt/n );		/ /注意控制输出格式
	}
}
```

B、请叫我简单题

题目描述

给定一个只含有字母的字符串，请按照下面定义的序对字符串中的字符进行升序排序，生成一个新的字符串：a<b<c<...<z<Z<Y<X....<B<A

题目分析

这是一道比较简单的字符串排序题，对于排序，大家都很熟悉，所以重点就在于选择哪种排序算法来完成这个题。对于这个多组测试数据，每组上限 1e8 的输入来说，O(n)的算法恐怕都会 TLE。但是正常情况下，排序算法的理论极值为 O(nlogn)，所以一般的排序算法似乎解决不了这个问题。不过我们还有”特殊的排序技巧”(BOJ 968)，也就是虽说是排序其实是计数的计数排序……对于数据量非常大但是数据范围相对较小的情况，计数排序是很强大的。

参考代码

```
#include <cstdio>
int main()
{
	int num[60];
	int n,i;
	char x;
	while( ~scanf("%d", &n) )
	{
		i=53;
		while (i--)
			num[i]=0;
		getchar();  / /注意读入回车
		while(n--)
		{
			x=getchar();
			if(x<'a')
				num[x-'A'+26]++;
			else
				num[x-'a']++;
		}
		x = 'a';
		for(i=0;i<26;i++)
```

```
        {
            while(num[i]--)
                printf("%c",x);
            x++;
        }
        i=26;
        x='Z';
        while(i--)
        {
            while(num[i+26]--)
                printf("%c",x);
            x--;
        }          / /注意，大写字母是倒字典序……
        printf("\n");
    }
}
```

C、第二长不下降子序列

题目描述

对于一个含有 n 个整数的输入序列 A[1..n]，A[i1],A[i2],...,A[ik]被称为一个长度为 k 的不下降子序列，当且仅当 0<i1<i2<..<ik<=n 且 A[i1]<=A[i2]<=...<=A[ik].请求出给定序列中，第二长的不下降子序列的长度.（若有多个序列的长度均为最大长度，即并列最长的情况，可定义其中任何一个为第二长，因而应输出该最大长度，eg.第 2 个样例）

题目分析

求解第二长子序列的策略和最长的是相同的。当我找到一个新的最长序列之后，之前找到的最长子序列就变成了第二长，所以只需要保留下每次更新 max 记录前的 max 值就好。

而求解最长不下降子序列问题，则是一道经典的 dp 问题。假设已知 A[i1],A[i2],…A[ik]序列为 A[1…n]中的一个最长不下降子序列，那么求解 A[1…n+1]中的最长不下降子序列时只需要考虑 A[n+1]与 A[ik]的大小关系就可以了。由于 A[1…n]中的最长不下降子序列不一定唯一，所以对于 A[n+1]，需要和之前的数比较多次。最一般的算法是 O(n^2)，也就是说和 A[1]到 A[n]每个都比较。而更好的算法为 O(nlogn)，利用了二分查找。

参考代码

```
#include <cstdio>
int a[10001],len[10001];
int main()
```

```
{
    int n,i,j,m,s;
    while(~scanf("%d", &n))
    {
        for(i=0; i<n; i++)
            scanf("%d", &a[i]);
        m = 0,s = 0;
        for(i=0; i<n; i++)
        {
            len[i]=1;
            for(j=0; j<i; j++)
                if(a[j]<=a[i]&&len[j]+1>len[i])
                    len[i] = len[j]+1;
            if(m<=len[i])
                s = m, m = len[i];
        }
        printf("%d\n", s);
    }
}
```

D、危险的 td

题目描述

假设 td 线一共有 n 个障碍！(1<n<=100000)
沛沛在每个障碍处有两种选择，一个是好好翻越，另外一个是从旁边溜
为了生命安全，沛沛任意两个连续的障碍都不会全好好翻越过去
这也意味着每两个连续的障碍必然至少有一个障碍是从旁边溜过去的
那么请问，沛沛跑到 td 的最后能有多少种不同的走法？
答案对 1e9+7 取模

题目分析

还是 dp，而且其实是很熟悉的 dp……
换个说法好了，小和尚爬楼梯每次可以上 1 个台阶或者两个台阶……
那么问题来了，“任意两个连续的障碍都不会好好翻越过去”跟上台阶有什么关系……
我们把问题抽象一下，对于上楼梯，假设已知 F（n-2）和 F（n-1），求 F（n），F（n-2）有两种办法可以到达 n 状态，走一步再走一步（这种其实会包含在 F（n-1）中）或者直接走两步，F（n-1）只有走一步一种。
而对于翻 td，已知沛沛姐 n-1（溜），则沛沛姐有两种办法到达 n 状态，翻和溜，而 n-1（翻）则只有溜一种。
现在看起来一样了吧……所以问题解决了……这道题就是斐波那契数……

参考代码

```
#include <iostream>
#include <cstdio>
using namespace std;
const long long MOD=1e9+7;
long long sum[100010];
int main()
{
    int n;
    sum[0] = 1;
    sum[1] = 2;
    for(n=2;n<100001;n++)
        sum[n]=(sum[n-2]+sum[n-1])%MOD; / /数据量略大，先打表后查询
    while(~scanf("%d",&n))
        printf("%lld\n",sum[n]);
}
```

E、来来来来数钱

题目描述

有 1 元的、2 元、5 元、10 元……的常见纸币多张
还有 3 元、7 元、11 元……的其他纸币多张
数一数一共可以组成多少种不同的总钱数

题目分析

//这个题应该算是本次上机的 BOSS 了……看了一天的背包问题，终于看懂了这题要怎么解……以及如果谁有非背包做法请务必告诉我……
因为只有半吊子水准如果想彻底把多重背包讲清楚是在无能为力，所以只能尽量说清楚下面的代码是在干吗……
W[],物品价值 d[]物品数量，cnt[j]，使用前 i-1 种物品组成价值 j 时，物品 i 的剩余量（-1 表示不能组成价值 j）。

既然是背包问题，肯定要对 i 种物品枚举。对于价值为 w[i]的第 i 种物品来说，当组成价值 j 时，可以使用 0——d[i]个，当使用 k 个时，只需要前 i-1 中物品能够组成价值 j-k*w[i]就满足了要求。所以利用这种思路可以使用二维 bool 数组进行 dp.

状态转移方程为 dp[i][j] |= dp[i-1][j-k*w[i]] ，k← 0 to d[i] （只要有一种成立（值为 1），dp[i][j]就成立（为 1））

背包问题都可以由二维压缩为 1 维，因为计算 i 时只需要 i-1，而使用 1 维数组时，每次外层循环结束，内层循环开始之前，dp 数组中的值都相当于是 i-1 时的值。另外 dp 数组中

可以存放物品 i 的剩余量而不是简单的 0 或 1，这样可以将空间重复利用而且时间复杂度也会降低（只需要对 j 枚举而不需要枚举 k）。

参考代码

```
#include <cstdio>
#include <cstring>
int w[51],d[51],cnt[25010];
int main()
{
	int n, m, i, j;
	int r; / /结果（价值种类数）
	while(~scanf("%d", &n))
	{
		r = m = 0;
		for(i=1;i<=n;i++)
		{
			scanf("%d%d", &w[i], &d[i]);
			m += w[i]*d[i];	/ /记录能组合出的最大价值（枚举上界）
		}
		memset(cnt,-1,sizeof(cnt)); / /假定 j 不能获得
		cnt[0] = 0;
		for(i=1; i<=n; i++)
		{
			for(j=0; j<=m; j++)
			{
				if(cnt[j] >= 0)
					cnt[j] = d[i];
				else if(j<w[i] || cnt[j-w[i]]<=0)
					cnt[j] = -1;
				else
					cnt[j] = cnt[j-w[i]]-1;
			}
		for(i=1; i<=m; i++)
			if(cnt[i]>=0)
				r++;
		printf("%d\n",r);
	}
}
```

F、 N-way Assemble Line

题目描述

假设这是一个 n 条工作线的工厂，每条工作线有 m 个装配站，编号都为 1-m，每条工作线的第 i 个装配站都执行相同的功能。元件一个元件要经过 m 个装配站才能加工完成，经过第 i 条工作线的第 j 个装配站要花费 p[i][j]的时间，从第 i 个工作线移动到第 j 个工作线要花费 t[i][j]的时间（不管当前元件是处于第几次装配），请问一个元件要加工完成需要的最少时间是多少？

题目分析

这个问题和 2 条工作线的问题很像，如果还记得上个学期的数据结构学的图算法的话……其实流水线问题就是弗洛伊德算法的一个应用，只不过最后的结果只需要求 A[][1]到点 A[][n]之间的最短距离。

参考代码

```
#include <cstdio>
#include <iostream>
using namespace std;
int p[101][101];
int t[101][101];
int Floyd(int n,int m)
{
	int dist[101][101];
	int i,j,k,r=1e9;
	for(i=1;i<=n;i++)
		for(j=1;j<=m;j++)
		dist[i][j] = 1e9;
	for(i=1;i<=n;i++)
		dist[i][1] = p[i][1];
	for(i=2;i<=m;i++)
		for(j=1;j<=n;j++)
			for(k=1;k<=n;k++)
				dist[j][i] = min(dist[j][i],dist[k][i-1]+p[j][i]+t[k][j]);
	for(i=1;i<=n;i++)
		r = min(r,dist[i][m]);
	return r;
}
int main()
{
	int T,n,m,i,j;
	scanf("%d", &T);
```

```
    while(T--)
    {
        scanf("%d%d", &n, &m);
        for(i=1;i<=n;i++)
            for(j=1;j<=m;j++)
                scanf("%d", &p[i][j]);
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                scanf("%d", &t[i][j]);
        printf("%d\n",Floyd(n,m));
    }
}
```

G、 金字塔

题目描述

这时，一座金字塔耸立在你面前，不过它是数字的！
三角形从上往下每层数目递增 1。我们假设你从顶层往下走，要走到底层，而且每个点都只能往左下或右下走。

```
          7
        3   8
      8   1   0
    2   7   7   4
  4   5   2   6   5
```

如上所示，我们可以认为 7->3->1->7->5 是一条合法的路径，请问对于所有的合法路径，权值和 最大是多少？

题目分析

题目比较简单，递归的算法非常好想，只需要求解下面两个点的权值和，取两者中大的加上上面点的权值，就可以获得上面点的最大权值和。递归算法需要先递归调用到最底层，然后逐步向上返回最大值，直接修改成迭代算法，就可以获得自底层向上求解的状态转移方程 W[i][j] += MAX(W[i+1][j],W[i+1][j+1])。
虽然从顶向下写 dp 也可以，但是还需要 o(N)的操作寻找最大值。

参考代码

```
#include <cstdio>
#include <iostream>
using namespace std;
int w[101][101];
```

```
void dp(int n)
{
	while(--n)
	{
		int i = n;
		while(i--)
			w[n-1][i] += max(w[n][i],w[n][i+1]);
	}
}
int main()
{
	int t,n;
	scanf("%d",&t);
	while(t--)
	{
		scanf("%d",&n);
		for(int i=0;i<n;i++)
			for(int j=0;j<=i;j++)
				scanf("%d",&w[i][j]);
		dp(n);
		printf("%d\n",w[0][0]);
	}
}
```

H、"Snake"的双 11

题目描述

Snake 看到 11.11 对于广大单身青年这么喜庆的日子被一对对情侣抢走后，心中无比郁闷，他决定从电影院下手来”报复“一下。

Snake 侵入了某电影院的售票系统，篡改数据库，这样导致的结果就是前台 MM 售出的票就有了重号的票了，即在电影院的一个位置可能同时出售给两个或者更多的人。

那到电影放映的时候会是什么样的呢？观众如果发现手中电影票对应的座位没有人，那他就会坐在这个位置；如果这个位置已经有人了，那他就会与这个人发生争吵，
这时候电影院的经理就需要出来协调，协调的结果就是让后到的这个人从手中票对应的座位开始依次向后走，直到找到一个没人的位置坐下。如果已经到了座位的末尾，
那么就从 1 号继续开始找起。

不断争吵的情景就是 Snake 最喜闻乐见的，Snake”报复“的目的就达到了。

现在给定观众每个人手中票所对应的座位号，请你给出最后各个位置的观众情况。

题目分析

这个题数据量是在太水，所以直接从票号开始一个一个找就能过……
但是我实在是没想到数据会这么水，所以用了一个稍微有点麻烦的算法……
position[i]记录最后一个票号为 i 的人的座位号，seat[i]则记录座位 i 上的人的票号。
初始状态数组都为 0，之后在找空座的时候，如果座位不空，则根据这个人的票号 k，直接找到最后一个 k 后面的座位。由于座位相当于是环形的，所以找的过程直接对 n 取模了。

参考代码

```
#include <cstdio>
int p[5001],s[5001];
int main()
{
	int n,m,i,x,y;
	while(~scanf("%d%d",&n, &m))
	{
		if (!(n | m))
			break;
		for(i=0;i<=n;i++)
			p[i]=s[i]=0;
		for(i=0;i<m;i++)
		{
			scanf("%d", &x);
			if(s[x]==0)
			{
				p[x]=x;
				s[x]=x;
			}
			else
			{
				y = (p[s[x]]+1)%n;
				if(y==0)
					y = n;
				while(s[y]!=0)
				{
					y = (p[s[y]]+1)%n;
					if(y==0)
						y=n;
				}
				p[x]=y;
				s[y]=x;
			}
```

```
            }
            for(i=1;i<=n;i++)
                  printf("%d ",s[i]);
            printf("\n");
      }
}
```