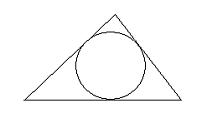
三角形内切圆的面积

题目描述:

给出三角形三边的边长，求此三角形内切圆（如下图所示，三角形的内切圆是和三角形三边都相切的圆）的面积。



输入:

三个正实数a、b、c（满足a+b>c，b+c>a，c+a>b）, 表示三角形三边的边长。

输出:

三角形内切圆的面积，结果四舍五入到小数点后面2位。

输入样例:

3 4 5

输出样例:

3.14

程序：

#include <stdio.h>

#include <math.h>

int main() {

float a,b,c,r,s,t;

scanf("%f %f %f",&a,&b,&c);

s = (a + b + c) / 2;

t = sqrt(s \* (s - a) \* (s - b) \* (s - c));

r = t / s;

printf("%0.2f", 3.1415927 \* r \* r);

return 0;

}

问题描述:工厂在每天的生产中,需要一定数量的零件,同时也可以知道每天生产一个零件的生产单价。在N天的生产中,当天生产的零件可以满足当天的需要,若当天用不完,可以放到下一天去使用,但要收取每个零件的保管费,不同的天收取的费用也不相同。

问题求解:求得一个N天的生产计划(即N天中每天应生产零件个数),使总的费用最少。

输入:N(天数N<=29)

每天的需求量(N个整数)

每天生产零件的单价(N个整数)

每天保管零件的单价(N个整数)

输出:每天的生产零件个数(N个整数)

例如:当N=3时,其需要量与费用如下:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第一天 | 第二天 | 第三天 |
| 需要量 | 25 | 15 | 30 |
| 生产单价 | 20 | 30 | 32 |
| 保管单价 | 5 | l0 | 0 |

生产计划的安排可以有许多方案,如下面的三种:

|  |  |  |  |
| --- | --- | --- | --- |
| 第一天 | 第二天 | 第三天 | 总的费用 |
| 25 | 15 | 30 | 25\*2O+15\*30+30\*32=1910 |
| 40 | 0 | 30 | 40\*20+15\*5+30\*32=1835 |
| 70 | 0 | 0 | 70\*20+45\*5+30\*10=1925 |

程序说明: （应该特别注意）

b[n]:存放每天的需求量

c[n]:每天生产零件的单价

d[n]:每天保管零件的单价

e[n]:生产计划

程序:

#include <stdio.h>

int main() {

int i,j,n,yu,j0,j1,s;

int b[31],c[31],d[31],e[31];

scanf("%d",&n);

for (i = 1;i <= n;i ++){

scanf("%d %d %d",b + i, c + i,d + i);

}

for (i = 1;i <= n;i ++){

e[i] = 0;

}

c[n + 1] = 10000;

c[n + 2] = 0;

b[n + 1] = 0;

j0 = 1;

while(j0 <= n) {

yu = c[j0];

j1 = j0;

s = b[j0];

while(yu + d[j1] < c[j1 + 1]) {

yu = yu + d[j1];

j1 = j1 + 1;

s = s + b[j1];

}

e[j0] = s;

j0 = j1 + 1;

}

for (i = 1;i <= n;i ++){

printf("%4d",e[i]);

}

return 0;

}

题目描述：

木材厂有一些原木，现在想把这些木头切割成一些长度相同的小段木头（木头有可能有

剩余），需要得到的小段的数目是给定的。当然，我们希望得到的小段越长越好，你的任务

是计算能够得到的小段木头的最大长度。木头长度的单位是cm。原木的长度都是正整数，

我们要求切割得到的小段木头的长度也是正整数。

输入:

第一行是两个正整数N和K(1 ≤ N ≤ 10000，1 ≤ K ≤ 10000)，N是原木的数目，

K是需要得到的小段的数目。

接下来的N行，每行有一个1到10000之间的正整数，表示一根原木的长度。

输出:

输出能够切割得到的小段的最大长度。如果连1cm长的小段都切不出来，输出”0”。

输入样例:

3 7

232

124

456

输出样例:

114

程序：

#include <stdio.h>

#include <stdbool.h>

int n,k;

int len[10001];

int i,left,right,mid;

bool isok(int t){

int num,i;

num = 0;

for(i = 1;i <= n;i ++){

if(num >= k){

break;

}

num = num + len[i] / t;

}

return num >= k;

}

int main(){

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

right = 0;

for(i = 1;i <= n;i ++){

scanf("%d",len + i);

if(right < len[i]){

right = len[i];

}

}

right ++;

left = 0;

while(left + 1 < right) {

mid = (left + right) / 2;

if (!isok(mid)) {

right = mid;

} else {

left = mid;

}

}

printf("%d",left);

return 0;

}

问题描述:有n种基本物质(n≤10),分别记为P1,P2,……,Pn,用n种基本物质构造一种物品,物品使用在k个不同地区(k≤20),每个地区对物品提出自己的要求,这些要求用一个n位的数表示:α1α2……αn,其中:

αi =1表示必须有第i种基本物质

=-1表示必须不能有第i种基本物质

=0无所谓

问题求解:当k个不同地区要求给出之后,给出一种方案,指出哪些物质被使用,哪些物质不被使用。

程序说明:数组b[1],b[2],...,b[n]表示某种物质是否需要

a[1..k,1..n]记录k个地区对物质的要求,其中:

a[I,j]=1表示第i个地区对第j种物质是需要的

a[i,j]=0表示第i个地区对第j种物质是无所谓的

a[i,j]=-1表示第i个地区对第j种物质是不需要的

程序:

#include <stdio.h>

#include <stdbool.h>

int main() {

int i,j,k,n;

bool p;

int a[21][11],b[21];

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

for(i = 1;i <= k;i ++){

for(j = 1;j <= n;j ++){

scanf("%d",&a[i][j]);

}

}

for(i = 0;i <= n;i ++){

b[i] = 0;

}

p = true;

while(p && b[0] == 0){

j = n;

while(b[j] == 1){

j --;

}

b[j] = 1;

p = false;

for(i = j + 1;i <= n;i ++) {

b[i] = 0;

}

for(i = 1;i <= k;i ++){

if((a[i][j] == 1 && b[j] == 0) || (a[i][j] == -1 && b[j] == 1)){

p = true;

}

}

}

if(p){

printf("找不到!");

} else {

for (i = 1;i <= n;i ++){

if(b[i] == 1){

printf("物质 %d 需要",i);

} else {

printf("物质 %d 不需要",i);

}

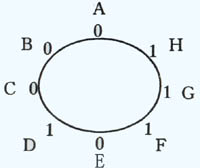
}

}

}

将2n个0和2n个1，排成一个圈。从任一个位置开始，每次按逆时针的方向以长度为n+1的单位进行数二进制数。要求给出一种排法，用上面的方法产生出来的2n+1个二进制数都不相同。

例如，当n=2时，即22个0和22个1排成如下一圈：



比如，从A位置开始，逆时针方向取三个数000，然后再从B位置上开始取三个数001，接着从C开始取三个数010，…可以得到000，001，010，101，011，111，110，100共8个二进制数且都不相同。

程序说明{重要，可以先自己设计算法}

以N=4为例，即有16个0，16个1，数组A用以记录32个0，1的排法，数组B统计二进制数是否已出现过。

程序清单

#include <stdio.h>

int main() {

int a[37],b[32];

int i,j,k,s,p;

for(i = 1;i <= 36;i ++){

a[i] = 0;

}

for(i = 28;i <= 32;i ++){

a[i] = 1;

}

p = 1;

a[6] = 1;

while(p == 1){

j = 27;

while(a[j] == 1){

j --;

}

a[j] = 1;

for(i = j + 1;i <= 27;i ++){

a[i] = 0;

}

for(i = 0;i <= 31;i ++){

b[1] = 0;

}

for(i = 1;i <= 32;i ++){

s = 0;

for(k = i;k <= i + 4;k ++){

s = s\*2 + a[k];

}

b[s] = 1;

}

s = 0;

for(i = 0;i <= 31;i ++){

s = s + b[i];

}

if (s == 32){

p = 0;

}

}

for(i = 1;i <= 32;i ++){

for(j = i;j <= i + 4;j ++){

printf("%d",a[j]);

}

printf("\n");

}

return 0;

}

问题描述:将n个整数分成k组(k≤n,要求每组不能为空),显然这k个部分均可得到一个各自的和s1,s2,……sk,定义整数P为:

P=(S1-S2)2+(S1一S3)2+……+(S1-Sk)2+(s2-s3)2+……+(Sk-1-Sk)2

问题求解:求出一种分法,使P为最小(若有多种方案仅记一种〉

程序说明:

数组:a[1],a[2],...A[N]存放原数

s[1],s[2],...,s[K]存放每个部分的和

b[1],b[2],...,b[N]穷举用临时空间

d[1],d[2],...,d[N]存放最佳方案

程序:

#include <stdio.h>

int main() {

int cmin,sum,i,j,n,k,a[101],b[101],d[101],s[31];

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

for(i = 1;i <= n;i ++){

scanf("%d",a+i);

}

for(i = 0;i <= n;i ++){

b[i] = 1;

}

cmin = 1000000;

while(b[0] == 1){

for(i = 1;i <= k;i ++){

s[k] = 0;

}

for(i = 1;i <= n;i ++){

s[b[i]] = s[b[i]] + a[i];

}

sum = 0;

for(i = 1;i <= k - 1;i ++){

for(j = i + 1;j <= k;j ++){

sum = sum + (s[i] - s[j]) \* (s[i] - s[j]);

}

}

if(sum < cmin){

cmin = sum;

for(i = 1;i <= n;i ++){

d[i] = b[i];

}

}

j = n;

while(b[j] == k){

j --;

}

b[j] ++;

for(i = j + 1;i <= n;i ++){

b[i] = 1;

}

}

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

for(i = 1;i <= n;i ++){

printf("%40d",d[i]);

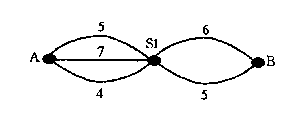
}

}

在A，B两个城市之间设有N个路站(如下图中的S1，且N<100)，城市与路站之间、路站和路站之间各有若干条路段(各路段数≤20，且每条路段上的距离均为一个整数)。

  　A，B的一条通路是指：从A出发，可经过任一路段到达S1，再从S1出发经过任一路段，…最后到达B。通路上路段距离之和称为通路距离(最大距离≤1000)。当所有的路段距离给出之后，求出所有不同距离的通路个数(相同距离仅记一次)。

　例如：下图所示是当N=1时的情况：



　从A到B的通路条数为6，但因其中通路5+5=4+6，所以满足条件的不同距离的通路条数为5。

　算法说明：本题采用穷举算法。

　数据结构：N：记录A，B间路站的个数  
　　　　　　数组D[I，0]记录第I-1到第I路站间路段的个数  
　　　　　　　　D[I，1]，D[I，2]，…记录每个路段距离  
　　　　　　数组G记录可取到的距离

程序清单：

#include <stdio.h>

int main() {

int i,j,n,s;

int b[101],d[101][21];

int g[1001];

scanf("%d",&n);

for(i = 1;i <= n + 1;i ++){

scanf("%d",&d[i][0]);

for(j = 1;j <= d[i][0];j ++){

scanf("%d",&d[i][j]);

}

}

d[0][0] = 1;

for(i = 1;i <= n + 1;i ++){

b[i] = 1;

}

b[0] = 0;

for(i = 0;i <= 1000;i ++){

g[i] = 0;

}

while(b[0] == 0){

s = 0;

for(i = 1;i <= n + 1;i ++){

s = s + d[i][b[i]];

}

g[s] = 1;

j = n + 1;

while(b[j] == d[j][0]){

j --;

}

b[j] = b[j] + 1;

for(i = j + 1;i <= n + 1;i ++){

b[i] = 1;

}

}

s = 0;

for(i = 1;i <= 1000;i ++){

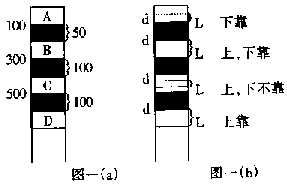
s = s + g[i];

}

printf("%d",s);

}

存储空间的回收算法。设在内存中已经存放了若干个作业A，B，C，D。其余的空间为可用的(如图一中(a))。



　　此时，可用空间可用一个二维数组dk[1..100，1..2 ]表示，(如下表一中(a))，其中：dk[i，1]对应第i个可用空间首址，dk[i，2]对应第i个可用空间长度如上图中，dk：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 100 | 50 | | 300 | 100 | | 500 | 100 | | |  |  | | --- | --- | | 0 | 0 | | 100 | 50 | | 300 | 100 | | 500 | 100 | | 10000 | 0 | |
| 表一(a) | 表一(b) |

　　现某个作业释放一个区域，其首址为d，长度为L，此时将释放区域加入到可用空间表中。要求在加入时，若可用空间相邻时，则必须进行合并。因此出现下面的4种情况：

　(1)下靠，即回收区域和下面可用空间相邻，例如，d=80，L=20，此时成为表二中的(a)。

　(2)上靠，例如，d=600，L=50，此时表成为表二中的(b)。

　(3)上、下靠，例如，d=150，L=150，此时表成为表二中的(c)。

　(4)上、下不靠，例如，d=430，L=20，此时表成为表二中的(d)。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 80 | 70 | | 300 | 100 | | 50 | 100 | | |  |  | | --- | --- | | 100 | 50 | | 300 | 100 | | 500 | 150 | | |  |  | | --- | --- | | 100 | 300 | | 500 | 100 | | |  |  | | --- | --- | | 100 | 50 | | 300 | 100 | | 430 | 20 | | 500 | 100 | |
| 表二(a)(下靠) | 表二(b)(上靠) | 表二(c)(上，下靠) | 表二(d)(上，下不靠) |

　程序说明：对数组dk预置2个标志，即头和尾标志，成为表一中(b)，这样可使算法简单，sp为dk表末地址。

程序清单：

#include <stdio.h>

int main(){

int i,j,k,sp,d,l;

int dk[101][3];

scanf("%d",&sp);

for(i = 1;i <= sp;i ++){

scanf("%d %d",&dk[i][1],&dk[i][2]);

}

dk[0][1] = dk[0][2] = 0;

sp ++;

dk[sp][1] = 10000;

dk[sp][2] = 0;

scanf("%d %d",&d,&l);

i = 1;

while(dk[i][1] < d){

i = i + 1;

}

i --;

if(dk[i][1] + dk[i][2] == d) {

if(d + l == dk[i + 1][1]){

dk[i][2] = dk[i][2] + l + dk[i + 1][2];

for(j = i + 1;j <= sp - 1;j ++){

for(k = 0;k <= 2;k ++){

dk[j][k] += dk[j + 1][k];

}

}

sp --;

} else {

dk[i][2] = dk[i][2] + l;

}

} else if(d + l == dk[i + 1][1]) {

dk[i + 1][1] = d;

dk[i + 1][2] = dk[i + 1][2] + l;

} else {

for(j = sp;j >= i + 1;j --){

for(k = 0;k <= 2;k ++){

dk[j + 1][k] = dk[j][k];

}

}

dk[i + 1][1] = d;

dk[i + 1][2] = l;

sp ++;

}

for(i = 1;i <= sp - 1;i ++){

printf("%4d %4d",dk[i][1],dk[i][2]);

}

}

题目描述:

原始的Joseph问题的描述如下：有n个人围坐在一个圆桌周围，把这n个人依次编号为1，…，n。从编号是1的人开始报数，数到第m个人出列，然后从出列的下一个人重新开始报数，数到第m个人又出列，…，如此反复直到所有的人全部出列为止。比如当n=6，m=5的时候，出列的顺序依次是5，4，6，2，3，1。

现在的问题是：假设有k个好人和k个坏人。好人的编号的1到k，坏人的编号是k+1到2k。我们希望求出m的最小值，使得最先出列的k个人都是坏人。

输入:

仅有的一个数字是k（0 < k <14）。

输出:

使得最先出列的k个人都是坏人的m的最小值。

输入样例:

4

输出样例:

30

程序：

#include <stdio.h>

#include <stdbool.h>

long i,k,m,start;

bool find;

bool check(int remain){

int result;

result = (start + m - 1) % remain;

if(result >= k){

start = result;

return true;

} else {

return false;

}

}

int main() {

find = false;

scanf("%ld",&k);

m = k;

while(!find) {

find = true;

start = 0;

for(i = 0;i <= k - 1;i ++){

if(!check(2 \* k - i)){

find = false;

break;

}

}

m ++;

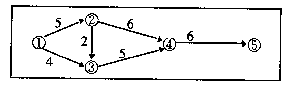
}

printf("%ld",m - 1);

}

  　设有一个工程网络如下图表示(无环路的有向图)：

  　其中，顶点表示活动，①表示工程开始，⑤表示工程结束(可变，用N表示)，边上的数字表示活动延续的时间。



  如上图中，活动①开始5天后活动②才能开始工作，而活动③则要等①、②完成之后才能开始，即最早也要7天后才能工作。

  　在工程网络中，延续时间最长的路径称为关键路径。上图中的关键路径为：①—②—③—④—⑤共18天完成。

　关键路径的算法如下：

1.数据结构：{重要定义}

　R[1..N，1..N]OF INTEGER；　　　表示活动的延续时间，若无连线，则用-1表示；

  　EET[1..N]　　　　　　　　　　　表示活动最早可以开始的时间

  　ET[1..N]　　　　　　　　　　 　表示活动最迟应该开始的时间

     关键路径通过点J，具有如下的性质：EET[J]=ET[J]

2.约定：

  　结点的排列已经过拓扑排序，即序号前面的结点会影响序号后面结点的活动。

程序清单：

#include <stdio.h>

int i,j,n,max,min,w,x,y;

int r[21][21];

int eet[21],et[21];

int main(){

scanf("%d",&n);

for(i = 1;i <= n;i ++){

for(j = 1;j <= n;j ++){

r[i][j] = -1;

}

}

scanf("%d %d %d",&x,&y,&w);

while(x != 0){

r[x][y] = w;

scanf("%d %d %d",&x,&y,&w);

}

eet[1] = 0;

for(i = 2;i <= n;i ++){

max = 0;

for(j = 1;j <= n;j ++){

if(r[j][i] != -1){

if(r[j][i] + eet[j] > max){

max = r[j][i] + eet[j];

}

}

}

}

et[n] = eet[n];

for(i = n - 1;i >= 1;i --){

min = 10000;

for(j = 1;j <= n;j ++){

if(r[i][j] != -1){

if(et[j] - r[i][j] < min){

min = et[j] - r[i][j];

}

}

}

}

printf("%d\n",eet[n]);

for(i = 1;i <= n - 1;i ++){

if(eet[i] == et[i]){

printf("%d ->",i);

}

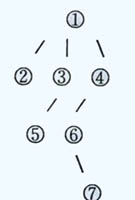
}

printf("%d",n);

}

求出一棵树的深度和宽度。例如有如下的一棵树：

其树的深度为从根结点开始到叶结点结束的最大深度，树的宽度为同一层上结点数的最大值。在上图中树的深度为4，宽度为3。



用邻接表来表示树，上图中的树的邻接表示如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 5 | 0 | 0 | 0 | 0 |
| 4 | 6 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 |
| 6 | 7 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 |

程序清单

#include <stdio.h>

int i,j,sp1,sp2,l,max;

int tree[21][7];

int q[101][7];

int d[21];

int main(){

for(i = 1;i <= 14;i ++){

for(j = 1;j <= 6;j ++){

tree[i][j] = 0;

}

}

for(j = 1;j <= 14;j ++){

tree[j][i] = j;

}

tree[1][2] = 2;

tree[1][3] = 3;

tree[1][4] = 4;

tree[2][2] = 5;

tree[2][3] = 6;

tree[3][2] = 7;

tree[3][3] = 8;

tree[4][2] = 9;

tree[4][3] = 10;

tree[4][4] = 11;

tree[7][2] = 12;

tree[7][3] = 13;

tree[13][2] = 14;

sp1 = 1;

sp2 = 1;

for(i = 1;i <= 6;i ++){

q[1][i] = tree[1][i];

}

q[1][0] = 1;

while(sp1 <= sp2){

l = q[sp1][0] + 1;

j = 2;

while(q[sp1][j] != 0){

sp2 ++;

q[sp2][0] = l;

q[sp2][1] = q[sp1][j];

for(i = 2;i <= 6;i ++){

q[sp2][i] = tree[q[sp1][j]][i];

}

j ++;

}

sp1 ++;

}

printf("%d\n",q[sp2][0]);

for(i = 0;i <= 20;i ++){

d[i] = 0;

}

for(i = 1;i <= sp2;i ++){

d[q[i][0]] = d[q[i][0]] + 1;

}

max = d[1];

for(i = 2;i <= 20;i ++){

if(d[i] > max){

max = d[i];

}

}

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

}

题目描述：

一摞硬币共有m枚，每一枚都是正面朝上。取下最上面的一枚硬币，将它翻面后放回原处。然后取下最上面的2枚硬币，将他们一起翻面后放回原处。在取3枚，取4枚……直至m枚。然后在从这摞硬币最上面的一枚开始，重复刚才的做法。这样一直做下去，直到这摞硬币中每一枚又是正面朝上为止。例如，m为1时，翻两次即可。

输    入：仅有的一个数字是这摞硬币的枚数m ，0< m <1000。

输    出：为了使这摞硬币中的每一枚都是朝正面朝上所必须翻的次数。

输入样例：30

输出样例：899

#include <stdio.h>

#include <stdbool.h>

int m;

int solve(int m){

int i,t,d;

bool flag;

int ret;

if(m == 1){

ret = 2;

} else {

d = 2 \* m + 1;

t = 2;

i = 1;

flag = false;

do{

if(t == 1){

ret = i \* m;

flag = true;

} else if(t == 2 \* m){

ret = i \* m - 1;

flag = true;

} else {

t = (t \* 2) % d;

}

i ++;

}while(!flag);

}

return ret;

}

int main() {

scanf("%d",&m);

if(m > 0 && m < 1000){

printf("%d",solve(m));

}

}

题目描述：

二维离散世界有一种地形叫OIM(OI Mountain)。这种山的坡度只能上升('/')或下降('\'),而且两边的山脚都与地平线等高,山上所有地方都不低于地平线.例如：

  /\           /\

 /  \/\ 是一座OIM；而 /   \    不是。

                            \/

这个世界的地理学家们为了方便纪录，给OIM所有可能的形状用正整数编好号，而且每个正整数恰好对应一种山形。他们规定，若两座山的宽度不同，则较宽的编号较大；若宽度相同，则比较从左边开始第1个坡度不同的地方，坡度上升的编号较大。以下三座OIM的编号有小到大递增：

 /\    /\     /\  /\

/  \/\  /  \/\/\  /  \/  \。显然/\的编号为1。但是地理学家在整理纪录是发觉，查找编号与山形的对应关系不是很方便。他们希望能快速地从编号得到山的形状。你自告奋勇答应他们写一个程序，输入编号，能马上输出山形。

输    入：一个编号（编号大小不超过600,000,000），

输    出：输入编号所对应的山形，1座山所占行数恰为它的高度，即山顶上不能有多余空行。

输入样例：15

输出样例：   /\  /\

       /  \/  \

程    序：

#include <stdio.h>

int L = 19;

int SZ = 50;

char UP = '/';

char DN = '\\';

int i,nth,x,y,h,e,f;

int m[2][39][20];

char pic[50][50];

void init(){

int k,s,a,b,c;

for(a = 0;a <= 1;a ++){

for(b = 0;b <= 2 \* L;b ++){

for(c = 0;c <= L;c ++){

m[a][b][c] = 0;

}

}

}

m[0][0][0] = 1;

for(k = 0;k <= 2 \* L - 1;k ++){

for(s = 1;s <= L;s ++){

m[0][k + 1][s] = m[0][k][s + 1] + m[1][k][s + 1];

m[1][k + 1][s] = m[0][k][s - 1] + m[1][k][s - 1];

}

m[0][k + 1][0] = m[0][k][1] + m[1][k][1];

}

}

void draw(int k,int s,int nth) {

printf("k = %d s = %d nth = %d\n",k,s,nth);

if(k == 0){

return;

}

if(nth - m[1][k][s] >= 0){

nth = nth - m[1][k][s];

if(y > h){

h = y;

}

printf("y = %d x = %d UP\n",y,x);

pic[y][x] = UP;

y ++;

x ++;

draw(k - 1,s + 1,nth);

} else {

y = y - 1;

printf("y = %d x = %d DOWN\n",y,x);

pic[y][x] = DN;

x ++;

draw(k - 1,s - 1,nth);

}

}

int main(){

init();

scanf("%d",&nth);

for(e = 0;e <= SZ -1;e ++){

for(f = 0;f <= SZ -1;f ++){

pic[e][f] = ' ';

}

}

x = y = h = i = 0;

while(nth - m[0][2 \* i][0] >= 0){

nth = nth - m[0][2 \* i][0];

i ++;

}

draw(2 \* i,0,nth);

printf("h = %d x = %d\n",h,x);

for(i = h;i >= y;i --){

for(e = 0;e <= x - 1;e ++){

printf("%c",pic[i][e]);

}

printf(" \n");

}

}