此题可采用DFS＋动态规划的方法解决。

dfs枚举每张邮票的面值了，那么上界在哪呢？可以发现，若当前连续值最大到mxi，则下一张邮票最大可以是mxi+1，再大就会在mxi+1位置上空缺，没有意义了

那么问题就是如何在每一个dfs里找到当前最大连续值－－－－背包DP f[i]表示到达i这个价值的最少邮票数，递推，一旦>n马上break

状态：f[i]为拼i所需的最少数的个数

阶段：1 to 之前取的数的个数

状态转移方程：f[i]:=min(f[i],f[j-a[i]]+1)

设f[dep][value]表示用已确定的第dep种邮票凑出value（邮资总值）最少需要的邮票数，则动态转移方程为：

f[dep][value]＝min{f[dep－1][value－v×i]＋i | i≥0且value－v×i≥0}(v是第dep种邮票的面值)

既要用前dep种邮票凑出value数额，可用的方案为用前dep－1种邮票凑出value－v×i的数额，再加上i张第dep种邮票，其中i≥0且value－v×i≥0。在这么多种方案中取使用邮票数最少的方案即可。

例如现在已经确定两种邮票的面值，即dep＝2，第dep种邮票的面值v＝3，现在求f[2][10]即前两种邮票凑成value＝10所需要的最少邮票数，那么就需要知道f[1][10]，f[1][10－3]，f[1][10－6]，f[1][10－9]的值，即用第一种邮票凑成value＝10，value＝7，value＝4，value＝1分别需要的邮票数。因为在此基础上只要分别再加0张第dep种邮票、再加1张第dep种邮票、再加2张第dep种邮票、再加3张第dep种邮票即可构成value＝10这一种情况。故只需从f[1][10]＋0，f[1][7]＋1，f[1][4]＋2，f[1][1]＋3中选择一个最小值，即为f[2][10]的值。

可以注意到：

f[dep][value－v]＝min{f[dep－1][value－v－v×j]＋j | j≥0且value－v－v×j≥0}

　　＝min{f[dep－1][value－v×(j＋1)]＋(j＋1)－1 }

＝min{f[dep－1][value－v×i]＋i－1 | i≥1且value－v×i≥0}(令i＝j＋1)

将－1提取到括号外得：

f[dep][value－v]＝min{f[dep－1][value－v×i]＋i | i≥1且value－v×i ≥0}－1

即f[dep][value－v]＋1＝min{f[dep－1][value－v×i]＋i | i≥1且value－v×i ≥0}　（１）

又已知f[dep][value]＝min{f[dep－1][value－v×i]＋i | i≥0且value－v×i≥0}

将i＝0的情况分离出来得到：

f[dep][value]＝min{f[dep－1][value－v×0]＋0， min{f[dep－1][value－v×i]＋i | i≥1 且value－v×i≥0}}　　　　　　　　　　　　　　　　　　　　　　　　　　　（２）

发现右边的表达式即刚求得的f[dep][value－v]＋1即（１）式，代入（２）式得：

f[dep][value]＝min{f[dep－1][value]，f[dep][value－v]＋1}

根据方程特点可将二维动态转移方程压缩为一维动态转移方程：

f[value]＝min{f[value]，f[value－v]＋1}

反过来可以有f[value＋v]＝min{f[value＋v] ， f[value]＋1}

具体实现时程序使用DFS深度优先搜索枚举每一种邮票的面值，对每个确定的面值通过递归调用求出用当前邮票种类能够连续凑出的最大数值。

参考代码如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | //邮票面值动规算法  #include <bits/stdc++.h>  using namespace std;  int k,n;  int ans[11],val[11]; //val[i]表示第i种邮票的面额  int f[1000],Max;  void DFS(int dep,int max) // max表示前dep－1种邮票最多能连续凑到max的数额  {  int i,j,v,t[1000];  if(dep>k) //如果已选了k所有种类的邮票都已经确定  {  if(max>=Max) //记录最优解  {  memcpy(ans,val,sizeof(val));  Max=max;  }  return;  }  memcpy(t,f,sizeof(f)); //备份动规数组  for(v=val[dep-1]+1; v<=max+1; v++)//若邮票面值v为max+2，则max+1无法凑出  {  val[dep]=v; //当前邮票面值存入val  for(int value=0; value<1000-v; value++)//动规  f[value+v]=min(f[value]+1,f[value+v]);  for(j=max; f[j]<=n; j++); //找到最小的j使得f[j]>n  //若f[j]>n则表示无法用n张已确定的dep种邮票凑出j  //因此用已确定的dep种邮票最多只能凑出连续j－1种数额  DFS(dep+1, j-1); //dfs确定下一种邮票面额  memcpy(f,t,sizeof(t));//还原动规数组  }  }  int main()  {  scanf("%d%d",&n,&k);  val[1]=1; //一开始只确定了第一种邮票，第一种邮票必然是1  memset(f,127/2,sizeof(f)); //超过n的面值都不能凑出，故设为无穷  for(int i=0; i<=n; i++) //要凑出不超过n张邮票的面值i需要i张第一种邮票  f[i]=i;  //因为第一种邮票面值已经确定，直接从第二种邮票开始枚举  DFS(2,n); //仅用第一种邮票最多能连续凑到面值n，即用n张第一种邮票  for(int i =1; i<=k; i++)  printf("%d ",ans[i]);  printf("%d\n",Max);  return 0;  } |