【算法分析】

首先将垃圾按出现时间排序。

垃圾吃或不吃，显然是一个0/1背包问题，即把垃圾的高度看成物体大小，维持生命的时间看成价值，井的高度看成包的容积，要求必须把包填满（或爆）能取得的最小价值。

设dp[i][j]表示前i个垃圾，到达高度j时所拥有的最长的生命时间。则：

1.吃下这个垃圾f[i][j]=max{f[i][j],f[i-1][j]-(上一垃圾到这一垃圾的间隔时间)+该垃圾价值}

2.将这个垃圾叠起来f[i][j]=max(f[i][j],f[i-1][j-a[i].h]-( 上一垃圾到这一垃圾的间隔时间)

初始f[i][j]为-INF。

更进一步的优化是设f[height]=left，即高度为height时的生命值。

参见代码：

|  |  |
| --- | --- |
| 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 | #include <bits/stdc++.h>  using namespace std;  struct p  {  int Time,Height,Life;  } c[101];  int depth,g;  int ti[101];  int f[101];//f[i]=j表示高度为i时的生命值为j  bool cmp(p a,p b)  {  return a.Time<b.Time;  }  int main()  {  cin>>depth>>g;  for(int i=1; i<=g; i++)  cin>>c[i].Time>>c[i].Life>>c[i].Height;  sort(c+1,c+1+g,cmp);  f[0]=10;  for(int i=1; i<=g; i++)  for(int j=depth; j>=0; j--)  if(f[j]>=c[i].Time)//如果生命值足够等待该垃圾下落时间  {  if(j+c[i].Height>=depth)//如果当前高度+垃圾高超过井高  {  cout<<c[i].Time;//则该垃圾扔下的时间即为走出陷阱的时间  return 0;  }  f[j+c[i].Height]=max(f[j+c[i].Height],f[j]);//不吃垃圾,堆高  f[j]+=c[i].Life;//吃垃圾，高度不变  }  cout<<f[0];//无法爬出，输出高度为0时的最大存活时间  return 0;  } |