**算法设计与分析——背包问题**

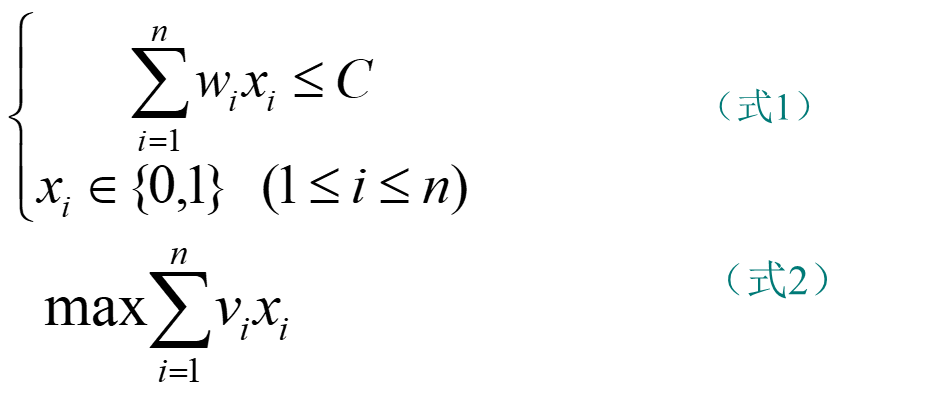


|  |  |
| --- | --- |
| 实验名称： | 算法设计与分析——0-1背包 |
| 姓 名： | 熊宇 |
| 班 级： | 2018211302 |
| 学 号： | 2018210074 |
| 学 院(系)： | 计算机学院 |
| 专 业： | 计算机科学与技术 |

2020年 11月11日

1. 问题描述
2. 编写满足下面要求的0-1背包算法，（必做）

**0-1背包问题：物品*i*或者被装入背包，或者不被装入背包，设*xi*表示物品*i*装入背包的情况，则当*xi*=0时，表示物品*i*没有被装入背包，*xi*=1时，表示物品*i*被装入背包。根据问题的要求，有如下约束条件和目标函数：**



**寻找一个满足约束条件式1，并使目标函数式2达到最大的解向量*X*=(*x*1, *x*2, …, *xn*)。**

（1）证明该问题满足最优子结构；

（2）给出递推式子；

（3）基于动态规划实现算法；

（4）分析算法的复杂度。

1. 屋上架屋实现对上述0-1背包问题的改进。

（1）上述算法要求所给物品的重量必须是整数，而实际处理问题时无法避免物品的重量是小数的情况，试编写一个能够处理重量为小数的情况。

（2）当背包容量c很大时，算法需要计算的时间很大，该算法的时间复杂度在c>2^n时为n\*c; 在算法中，注意到m(i,j)是阶梯状单调不减函数。请试图改进该算法，提高算法复杂度。

1. 问题分析
2. 初步部分
3. 证明该问题满足最优子结构

证明：设有x1，x2，…，xn共n个物品，他们的重量为w1，w2，…，wn，他们的价值为v1，v2，…，vn。背包可容纳最大重量为C。

设（y1,y2,…,yk-1）是x1~xn的一个最优解，我们可以推断：

* 1. 如果yk=1，那么m[k-1][MaxWeight-wk]+vk>m[k-1][MaxWeight]。

我们假设m[k-1][MaxWeight-wk]+vk<=m[k-1][MaxWeight]，那么就有m[k][MaxWeight]= m[k-1][MaxWeight-wk]+vk<=m[k-1][MaxWeight]，就有（y1,y2,…,yk-1）不是最优解，与前提矛盾。

* 1. 如果yk=0，那么m[k-1][MaxWeight-wk]+vk<=m[k-1][MaxWeight]。

证明同上。

1. 给出递推式子
2. 基于动态规划实现算法

我们可以采用一个二维数组去解决：m[i][j]，其中i代表加入背包的是前i件物品，j表示背包的承重，m[i][j]表示当前状态下能放进背包里面的物品的最大总价值。那么，m[n][m]就是我们的最终结果了。采用动态规划，必须要知道初始状态和状态转移方程。初始状态很容易就能知道，那么状态转移方程如何求呢？对于一件物品，我们有放进或者不放进背包两种选择：

* 1. 假如我们放进背包，m[i][j] = m[i - 1][j - weight[i]] + value[i]，这里的m[i - 1][j - weight[i]] + value[i]应该这么理解：在没放这件物品之前的状态值加上要放进去这件物品的价值。而对于m[i - 1][j - weight[i]]这部分，i - 1很容易理解，关键是 j - weight[i]这里，我们要明白：要把这件物品放进背包，就得在背包里面预留这一部分空间。
  2. 假如我们不放进背包，m[i][j] = m[i - 1][j]，这个很容易理解。

因此，我们的状态转移方程就是：m[i][j] = max(m[i][j] = m[i - 1][j] , m[i - 1][j - weight[i]] + value[i])

* 1. 当然，还有一种特殊的情况，就是背包放不下当前这一件物品，这种情况下m[i][j] = m[i - 1][j]。

1. 分析算法的复杂度

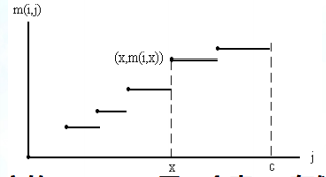
从m[i,j]的递归式容易看出，算法需要O(nC)计算时间。当背包容量C很大时，算法需要的计算时间较多。例如，当C>2^n时，算法需要Ω(n2^n)计算时间。

1. 选做部分
2. 上述算法要求所给物品的重量必须是整数，而实际处理问题时无法避免物品的重量是小数的情况，试编写一个能够处理重量为小数的情况。

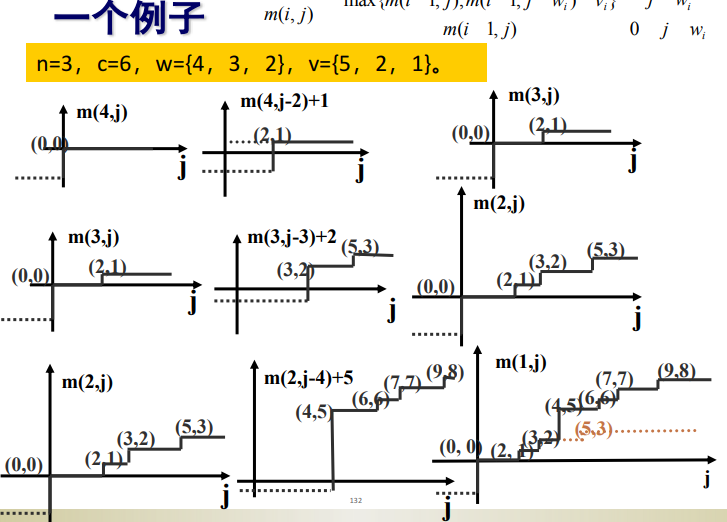
将为小数的重量通过整体乘法变成整数，然后回归到最普通的0-1背包即可。（以二位小数举例实现）

1. 当背包容量c很大时，算法需要计算的时间很大，该算法的时间复杂度在c>2^n时为n\*c; 在算法中，注意到m(i,j)是阶梯状单调不减函数。请试图改进该算法，提高算法复杂度。

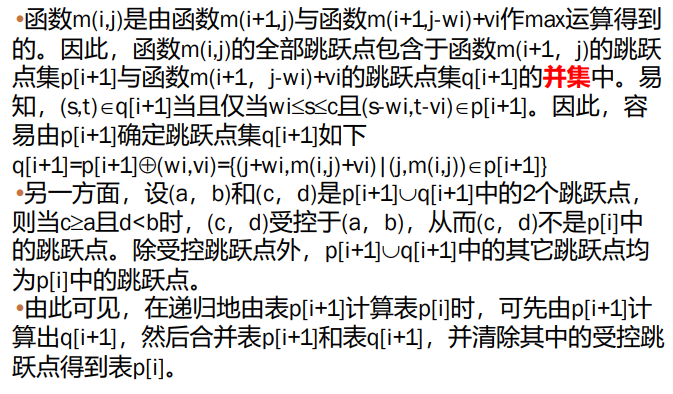
由m(i,j)的递归式容易证明，在一般情况下，对每一个确定的i(1≤i≤n)，函数m(i,j)是关于变量j的阶梯状单调不减函数。跳跃点是这一类函数的描述特征。在一般情况下，函数m(i,j)由其全部跳跃点唯一确定。如图所示。



对每一个确定的i(1≤i≤n)，用一个表p[i]存储函数m(i，j)的全部跳跃点。表p[i]可依计算m(i，j)的递归式递归地由表p[i+1]计算，初始时p[n+1]={(0，0)}。

举个例子：

算法改进想法：



再举个例子：

n=5，c=10，w={2，2，6，5，4}，v={6，3，5，4，6}。

矩阵如下：0

0 6 6 6 6 6 6 6 6 6

0 6 6 9 9 9 9 9 9 9

0 6 6 9 9 9 9 11 11 14

0 6 6 9 9 9 10 11 13 14

0 6 6 9 9 12 12 15 15 15

跳跃点的计算过程如下：

初始时p[6]={(0,0)}

因此，q[6]=p[6]⊕(w[5],v[5])={(4,6)}

p[5]={(0,0),(4,6)}

q[5]=p[5]⊕(w[4],v[4])={(5,4),(9,10)}

p[5]与q[5]的并集p[5]∪q[5]={(0,0),(4,6),(5,4),(9,10)}中跳跃点(5,4)受控于跳跃点(4,6)。

将受控跳跃点(5,4)清除后，得到p[4] ={(0,0),(4,6),(9,10)}

q[4]=p[4]⊕(6，5)={(6，5)，(10，11)}

p[3]={(0，0)，(4，6)，(9，10)，(10，11)}

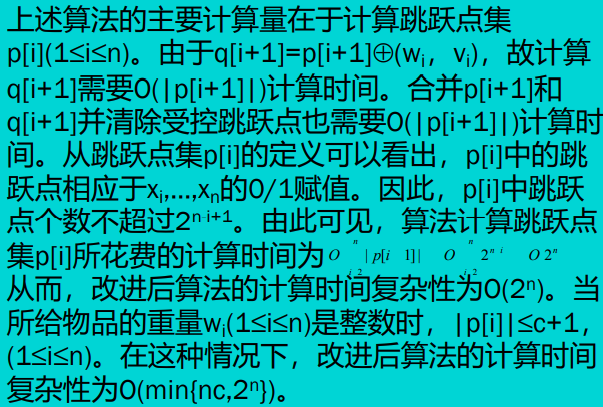
q[3]=p[3]⊕(2，3)={(2，3)，(6，9)}

p[2]={(0，0)，(2，3)，(4，6)，(6，9)，(9，10)，(10，11)}

q[2]=p[2]⊕(2，6)={(2，6)，(4，9)，(6，12)，(8，15)}

p[1]={(0，0)，(2，6)，(4，9)，(6，12)，(8，15)}

p[1]的最后的那个跳跃点(8,15)即为所求的最优值，m(n,C)=15



1. 源代码
2. 初步部分 11\_10.cpp

#include <iostream>  
#define V 500  
using namespace std;  
int weight[20 + 1];  
int value[20 + 1];  
int m[20 + 1][V + 1];  
int main()  
{  
   int n, C;  
   cout << "请输入可供选择的物品个数和背包所能容纳的最大容量"<<endl;  
   cin >> n>>C;  
   cout << "请分行输入" << n << "个物品的重量和价值，以空格间隔:" << endl;   
   for (int i = 1; i <= n; i++)   
  {  
       cin >> weight[i] >> value[i];  
  }  
   for (int i = 1; i <= n; i++)   
  {  
       for (int j = 1; j <= C; j++)   
      {  
           if (weight[i] > j)   
          {  
               m[i][j] = m[i - 1][j];  
          }  
           else   
          {  
               m[i][j] = m[i - 1][j] > m[i - 1][j - weight[i]] + value[i] ? m[i - 1][j] : m[i - 1][j - weight[i]] + value[i];  
          }  
      }  
  }  
   for(int i=1;i<=n;i++)  
  {  
       for(int j=1;j<=C;j++)  
      {  
           cout<<m[i][j]<<" ";  
      }  
       cout<<endl;  
  }  
   cout << "背包能放的最大价值为:" << m[n][C] << endl;  
   system("pause");  
   return 0;  
}

1. 选做部分
2. 上述算法要求所给物品的重量必须是整数，而实际处理问题时无法避免物品的重量是小数的情况，试编写一个能够处理重量为小数的情况。

11\_11-2.cpp

#include <iostream>  
#define V 500  
using namespace std;  
int weight[20 + 1];  
int value[20 + 1];  
double FWeight[20+1];  
double FValue[20+1];  
int m[20 + 1][V + 1];  
int main()  
{  
   int n,C;  
   double xc;  
   cout << "请输入可供选择的物品个数和背包所能容纳的最大容量"<<endl;  
   cin >> n>>xc;  
   C=xc\*100;  
   cout << "请分行输入" << n << "个物品的重量和价值，以空格间隔:" << endl;   
   for (int i = 1; i <= n; i++)   
  {  
       //cin >> weight[i] >> value[i];  
       cin >> FWeight[i] >> FValue[i];  
       weight[i]=FWeight[i]\*100;  
       value[i]=FValue[i];  
  }  
   for (int i = 1; i <= n; i++)   
  {  
       for (int j = 1; j <= C; j++)   
      {  
           if (weight[i] > j)   
          {  
               m[i][j] = m[i - 1][j];  
          }  
           else   
          {  
               m[i][j] = m[i - 1][j] > m[i - 1][j - weight[i]] + value[i] ? m[i - 1][j] : m[i - 1][j - weight[i]] + value[i];  
          }  
      }  
  }  
   for(int i=1;i<=n;i++)  
  {  
       for(int j=1;j<=C;j++)  
      {  
           cout<<m[i][j]<<" ";  
      }  
       cout<<endl;  
  }  
   cout << "背包能放的最大价值为:" << m[n][C] << endl;  
   system("pause");  
   return 0;  
}

1. 当背包容量c很大时，算法需要计算的时间很大，该算法的时间复杂度在c>2^n时为n\*c; 在算法中，注意到m(i,j)是阶梯状单调不减函数。请试图改进该算法，提高算法复杂度。

11\_11.cpp

#include <iostream>  
​  
#define V 500  
using namespace std;  
int weight[20 + 1];  
int value[20 + 1];  
int m[20 + 1][V + 1];  
​  
template<class Type>  
int pack(int n,Type c,Type v[],Type w[],int \*\*p,int x[]);  
template<class Type>  
void trace(int n,Type w[],Type v[],Type \*\*p,int \*head,int x[]);  
​  
int main()  
{  
   int n, C;  
   cout << "请输入可供选择的物品个数和背包所能容纳的最大容量"<<endl;  
   cin >> n>>C;  
   cout << "请分行输入" << n << "个物品的重量和价值，以空格间隔:" << endl;   
   for (int i = 1; i <= n; i++)   
  {  
       cin >> weight[i] >> value[i];  
  }  
   int x[20+1]; //针对物品而言，xi被装进去，就置1；否则置0  
​  
   int \*\*p=new int \*[V];  
   for(int i=0;i<V;i++)  
  {  
       p[i]=new int[2]; //记录跳跃点，第一个int存重量，第二个int存价值  
  }  
   cout << "背包能放的最大价值为:" << pack(n,C,value,weight,p,x) << endl;  
   cout<<"背包装下的物品编号为：";  
for(int i=1; i<=n; i++)  
{  
if(x[i]==1)  
{  
cout<<i<<" ";  
}  
}  
cout<<endl;  
​  
for(int i=0; i<V; i++)  
{  
delete p[i];  
}  
​  
delete[] p;  
   system("pause");  
   return 0;  
}  
​  
template<class Type>  
int pack(int n,Type c,Type v[],Type w[],int \*\*p,int x[])  
{  
int \*head = new int[n+2];//下标从1开始，记录n+1个结点   
head[n+1]=0;  
​  
p[0][0]=0;//第一个记录重量，第二个记录价值  
p[0][1]=0;  
​  
// left 指向p[i+1]的第一个跳跃点，right指向最后一个，next指向下一个跳跃点要存放的位置  
int left = 0,right = 0,next = 1;  
head[n]=1;  
​  
for(int i=n; i>=1; i--)  
{  
int k = left;//k指向p[ ]中跳跃点,移动k来判断p[]与p[]+（w v）中的受控点  
for(int j=left; j<=right; j++)  
{  
if(p[j][0]+w[i]>c)   
               break;//背包装不下第i个物品，直接退出循环  
Type y = p[j][0] + w[i],m = p[j][1] + v[i];  
​  
//若p[k][0]较小则(p[k][0] p[k][1])一定不是受控点，将其作为p[i]的跳跃点存储  
while(k<=right && p[k][0]<y)  
{  
p[next][0]=p[k][0];  
p[next++][1]=p[k++][1];  
}  
​  
//受控点，不存  
if(k<=right && p[k][0]==y)  
{  
if(m<p[k][1])//对（p[k][0]   p[k][1]）进行判断  
{  
m=p[k][1];  
}  
k++;  
}  
​  
// 若p[k][0]>=y且m> =p[k][1],判断是不是当前i的最后一个跳跃点的受控点  
//若不是跳跃点，则作为i的跳跃点存储  
if(m>p[next-1][1])  
{  
p[next][0]=y;  
p[next++][1]=m;  
}  
​  
//若是，则对下一个元素进行判断。  
while(k<=right && p[k][1]<=p[next-1][1])  
{  
k++;  
}  
}  
​  
while(k<=right)  
{  
p[next][0]=p[k][0];  
p[next++][1]=p[k++][1];//将i+1剩下的跳跃点作为做为i的跳跃点存储  
}  
​  
       //更改left和right  
left = right + 1;  
right = next - 1;  
​  
// 第i-1个物品第一个跳跃点的位置 head[n]指第n个物品第一个跳跃点的位置  
head[i-1] = next;  
}  
​  
trace(n,w,v,p,head,x); //回溯踪迹，便于输出  
return p[next-1][1];  
}  
​  
​  
template<class Type>  
void trace(int n,Type w[],Type v[],Type \*\*p,int \*head,int x[])  
{  
//初始化j,m为最后一个跳跃点对应的第0列及第1列  
Type j = p[head[0]-1][0],m=p[head[0]-1][1];  
for(int i=1; i<=n; i++)  
{  
x[i]=0;// 初始化数组；  
for(int k=head[i+1]; k<=head[i]-1; k++) // 初始k指向p[2]的第一个跳跃点（0 0）  
{  
//判断物品i是否装入，装入就置1  
if(p[k][0]+w[i]==j && p[k][1]+v[i]==m)  
{  
x[i]=1;//物品i被装入，则x[i]置1  
j=p[k][0];  
m=p[k][1];  
break;  
}  
}  
}  
}

1. 实验结果及分析
2. 初步部分

请输入可供选择的物品个数和背包所能容纳的最大容量  
5 10  
请分行输入5个物品的重量和价值，以空格间隔:  
2 6  
2 3  
6 5  
5 4  
4 6  
0 6 6 6 6 6 6 6 6 6  
0 6 6 9 9 9 9 9 9 9  
0 6 6 9 9 9 9 11 11 14  
0 6 6 9 9 9 10 11 13 14  
0 6 6 9 9 12 12 15 15 15  
背包能放的最大价值为:15  
Press any key to continue . . .

1. 选做部分
2. 上述算法要求所给物品的重量必须是整数，而实际处理问题时无法避免物品的重量是小数的情况，试编写一个能够处理重量为小数的情况。

请输入可供选择的物品个数和背包所能容纳的最大容量  
5 0.1  
请分行输入5个物品的重量和价值，以空格间隔:  
0.02 6  
0.02 3  
0.06 5  
0.05 4  
0.04 6  
0 6 6 6 6 6 6 6 6 6  
0 6 6 9 9 9 9 9 9 9  
0 6 6 9 9 9 9 11 11 14  
0 6 6 9 9 9 10 11 13 14  
0 6 6 9 9 12 12 15 15 15  
背包能放的最大价值为:15  
Press any key to continue . . .

1. 当背包容量c很大时，算法需要计算的时间很大，该算法的时间复杂度在c>2^n时为n\*c; 在算法中，注意到m(i,j)是阶梯状单调不减函数。请试图改进该算法，提高算法复杂度

请输入可供选择的物品个数和背包所能容纳的最大容量  
5 10  
请分行输入5个物品的重量和价值，以空格间隔:  
2 6  
2 3  
6 5  
5 4  
4 6  
背包能放的最大价值为:15  
背包装下的物品编号为：1 2 5  
Press any key to continue . . .

1. 实验心得

在本次背包问题的解决中，我借助于老师上课的讲解和PPT、互联网的帮助，学习了动态规划中的典型问题——背包问题的设计求解。在学习过程中，我了解到除了最基础的0-1背包外，还有完全背包、多重背包等“背包九讲”，对背包九讲有了一定的了解，体会到了动态规划“上帝视角”、万变不离其宗的奇妙之处，受益匪浅。