

中国矿业大学计算机学院

2016 级本科生实验报告

课程名称 算法设计与分析

报告时间 12 月 4 日

学生姓名 刘宏波

学 号 08163288

专 业 计算机科学与技术 16-7 班

任课教师 王志晓

实验内容	指标点 3.3	指标点 4.3	成绩
实验一			
实验二			
实验三			
实验四			
总成绩			

实验一 0-1 背包问题的贪心算法

一、实验目的

通过本实验加深对贪心算法的理解。

二、实验内容

掌握贪心算法的概念和基本思想，掌握并分析“0-1”背包问题的贪心算法。

三、源程序

```
#include<iostream.h>

#define max 100

void sort (int n,float a[max],float b[max])

{

    int j,h,k;

    float t1,t2,t3,c[max];

    for(k=1;k<=n;k++)

        c[k]=a[k]/b[k];

    for(h=1;h<n;h++)

        for(j=1;j<=n-h;j++)

            if(c[j]<c[j+1])

            {

                t1=a[j];a[j]=a[j+1];a[j+1]=t1;

                t2=b[j];b[j]=b[j+1];b[j+1]=t2;

                t3=c[j];c[j]=c[j+1];c[j+1]=t3;

            }

}
```

```

void knapsack(int n,float limitw,float v[max],float w[max],int x[max])

{

    float c1;

    int i;

    sort(n,v,w);

    cout<<"货物按价值密度排序后为: "<<endl;

    cout<<"价值: ";

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

    {

        cout<<v[i]<<" ";

    }

    cout<<endl<<"重量: ";

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

    {

        cout<<w[i]<<" ";

    }

    c1=limitw;

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

    {

        if(w[i]>c1)

            continue;

        x[i]=1;

        c1=c1-w[i];

    }

}

```

```

void main()

{

    int n,i,x[max];

    float v[max],w[max],totalv=0,totalw=0,limitw;

    while(true)

    {

        cout<<"请输入货物数量:"<<endl;

        cin>>n;

        cout<<"背包最大载重:"<<endl;

        cin>>limitw;

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

            x[i]=0;

        cout<<"请依次输入物品的价值: "<<endl;

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

            cin>>v[i];

        cout<<"请依次输入物品的重量: "<<endl;

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

            cin>>w[i];

        knapsack (n,limitw,v,w,x);

        cout<<endl<<"装载结果为:";

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

```

```

    {

        cout<<x[i]<<" ";

        if(x[i]==1)

        {

            totalw=totalw+w[i];

            totalv=totalv+v[i];

        }

    }

    cout<<endl;

    cout<<"背包的总重量为: "<<totalw<<endl;

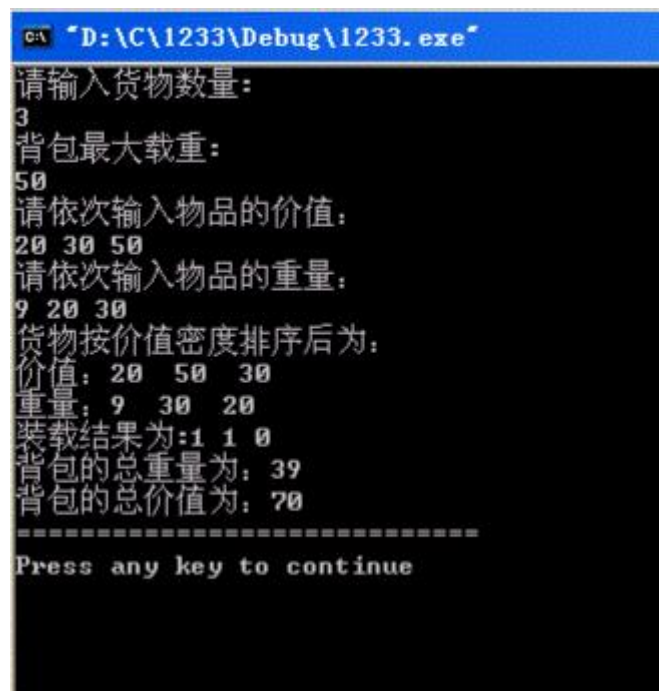
    cout<<"背包的总价值为: "<<totalv<<endl;

    cout<<"===== "<<endl;

}

```

四、运行结果



```

D:\C\1233\Debug\1233. exe
请输入货物数量:
3
背包最大载重:
50
请依次输入物品的价值:
20 30 50
请依次输入物品的重量:
9 20 30
货物按价值密度排序后为:
价值: 20 50 30
重量: 9 30 20
装载结果为:1 1 0
背包的总重量为: 39
背包的总价值为: 70
=====
Press any key to continue

```

实验二、0-1 背包问题的动态规划算法

一、实验目的

通过本实验加深对动态规划算法的理解。

二、实验内容

掌握动态规划算法的概念和基本思想，掌握并分析“0-1”背包问题的动态规划算法。

三、源程序

```
#include<iostream.h>

int c[10][100];

int knapsack(int wmax,int n)
{
    int i,j,w[10],p[10];

    cout<<"请输入每个物品的重量: "<<endl;

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

        cin>>w[i];

    cout<<"请输入每个物品的价值: "<<endl;

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

        cin>>p[i];

    for(i=0;i<10;i++)

    {
        for(j=0;j<100;j++)

            c[i][j]=0;
    }
}
```

```

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

        {

            for(j=1;j<=wmax;j++)

            {

                if(w[i]<=j)

                {

                    if(p[i]+c[i-1][j-w[i]]>c[i-1][j])

                        c[i][j]=p[i]+c[i-1][j-w[i]];

                    else

                        c[i][j]=c[i-1][j];

                }

                else

                    c[i][j]=c[i-1][j];

            }

        }

        return(c[n][wmax]);

    }

    int main()

    {

        int wmax,n,i,j;

        cout<<"请输入背包的载重: "<<endl;

        cin>>wmax;

        cout<<"请输入货物数量: "<<endl;

```



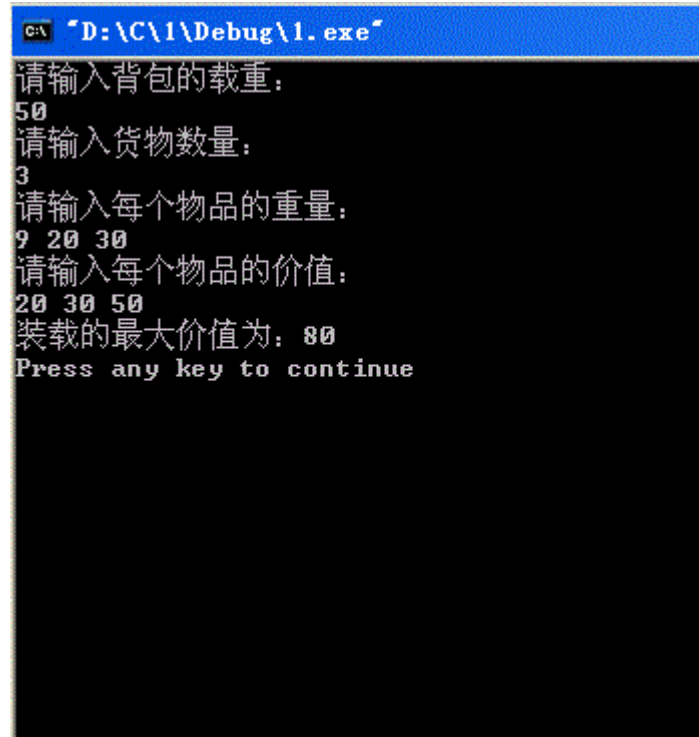
```
    cin>>n;

    cout<<"装载的最大价值为: "<<knapsack(wmax,n)<<endl;

    return 0;

}
```

四、运行结果



```
C:\> "D:\C\1\Debug\1.exe"
请输入背包的载重:
50
请输入货物数量:
3
请输入每个物品的重量:
9 20 30
请输入每个物品的价值:
20 30 50
装载的最大价值为: 80
Press any key to continue
```

实验三、0-1 背包问题的回溯算法

一、实验目的

通过本实验加深对回溯算法的理解。

二、实验内容

掌握回溯算法的概念和基本思想，掌握并分析“0-1”背包问题的回溯算法。

三、源程序

```
#include<iostream>

using namespace std;

class Knap

{

    friend int Knapsack(int p[],int w[],int c,int n );

public:

    void print()

    {

        for(int m=1;m<=n;m++)

            cout<<bestx[m]<<" ";

        cout<<endl;

    };

private:

    int Bound(int i);

    void Backtrack(int i);
```

```
int c;//背包容量

int n; //物品数

int *w;//物品重量数组

int *p;//物品价值数组

int cw;//当前重量

int cp;//当前价值

int bestp;//当前最优值

int *bestx;//当前最优解

int *x;//当前解

};
```

```
int Knap::Bound(int i)

{ //计算上界

    int cleft=c-cw; //剩余容量

    int b=cp;    //以物品单位重量价值递减序装入物品

    while(i<=n&&w[i]<=cleft)

    {

        cleft-=w[i];

        b+=p[i];

        i++;

    } //装满背包

    if(i<=n)
```

```

        b+=p[i]/w[i]*cleft;

    return b;

}

```

```

void Knap::Backtrack(int i)

```

```

{

    if(i>n)

    {

        if(bestp<cp)

        {

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

                bestx[j]=x[j];

            bestp=cp;

        }

        return;

    }

    if(cw+w[i]<=c) //搜索左子树

    {

        x[i]=1;

        cw+=w[i];

        cp+=p[i];

        Backtrack(i+1);

    }

}

```

```

        cw-=w[i];

        cp-=p[i];

    }

    if(Bound(i+1)>bestp)//搜索右子树

    {

class Object

    {

        friend int Knapsack(int p[],int w[],int c,int n);

public:

        int operator<=(Object a)const

        {

            return (d>=a.d);

        }

private:

        int ID;

        float d;

    };

int Knapsack(int p[],int w[],int c,int n)

{ //为 Knap::Backtrack 初始化

    int W=0;

    int P=0;

```

```

int i=1;

Object *Q=new Object[n];

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

{

    Q[i-1].ID=i;

    Q[i-1].d=1.0*p[i]/w[i];

    P+=p[i];

    W+=w[i];

}

if(W<=c)

    return P;//装入所有物品

//依物品单位重量排序

float f;

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

    for(int j=i;j<n;j++)

    {

        if(Q[i].d<Q[j].d)

        {

            f=Q[i].d;

            Q[i].d=Q[j].d;

            Q[j].d=f;

        }

    }

```

```

}

Knap K;

K.p = new int[n+1];

K.w = new int[n+1];

K.x = new int[n+1];

K.bestx = new int[n+1];

K.x[0]=0;

K.bestx[0]=0;

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

{

    K.p[i]=p[Q[i-1].ID];

    K.w[i]=w[Q[i-1].ID];

}

K.cp=0;

K.cw=0;

K.c=c;

K.n=n;

K.bestp=0; //回溯搜索

K.Backtrack(1);

K.print();

delete [] Q;

delete [] K.w;

```

```

        delete [] K.p;

        return K.bestp;
    }

void main()

{

    int *p;

    int *w;

    int c=0;

    int n=0;

    int i=0;

    //char k;

    while(true)

    {

        cout<<"请输入背包容量: "<<endl;

        cin>>c;

        cout<<"请输入物品的个数: "<<endl;

        cin>>n;

        p=new int[n+1];

        w=new int[n+1];

        p[0]=0;

        w[0]=0;
    }
}

```



```

        cout<<"请输入物品的价值: "<<endl;

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

            cin>>p[i];

        cout<<"请输入物品的重量: "<<endl;

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

            cin>>w[i];

        cout<<"最优解为: "<<endl;

        //cout<<"最优值为(bestp): "<<endl;

        cout<<Knapsack(p,w,c,n)<<endl;

        cout<<"===== "<<endl;

        //cout<<"[s] 重新开始"<<endl;

        //cout<<"[q] 退出"<<endl;

        //cin>>k;

    }

}

```

四、运行结果

```
C:\ "D:\C\1\Debug\1.exe"
请输入背包容量:
50
请输入物品的个数:
3
请输入物品的价值:
20 30 50
请输入物品的重量:
9 20 30
最优解为:
0 1 1
80
=====
请输入背包容量:
100
请输入物品的个数:
5
请输入物品的价值:
20 30 40 50 60
请输入物品的重量:
13 23 35 51 55
最优解为:
1 1 0 0 1
110
=====
请输入背包容量:
-
```

五、实验感想

贪心算法志考虑眼前的利益而不考虑长久利益，得到的可能不是最优解，所以编程时一步一步顺着走下来就好了，相对简单。在编写 0-1 背包回溯和动态规划算法的程序时，最重要的时先滤清思路，比如在纸上画一颗树，根据算法走一遍，搞清楚每一步时怎么走的，参数怎么变化等，最好在一张纸上一步一步记录清楚，编程时避免出现很多问题。