目录

1 概述

1.1 课程设计题目

贪婪算法解决 0/1 背包问题。

1.2 课程设计目的

明确课程设计的目的和重要性，认真领会课程设计的题目，读懂课程设计指导书的要求，学会设计的基本方法与步骤，学会如何运用前修知识与收集、归纳相关资料解决具体问题的方法。

1.3 课程设计要求

严格要求自己，要独立思考，按时、独立完成能力拓展训练任务。设计报告：要求层次清楚，整洁，规范,不得相互抄袭。

1.4 设计环境及工具

（1）开发平台：Linux.
（2）编程语言：C++.
(3) 开发工具：Vi 编辑器，G++编译器.
(4) 文档编写：LibreOffice.

2 总体设计

2.1 需求分析

2.1.1 问题描述

已尺寸大小分别为 W1 , W2, ..., Wn 的若干物品和容量为 C 的背包, 物品的价值分别为 P1,P2, , Pn。要求找出这 n 个物品的一个子集, 使其尽可能使选入背包的物品的价值最大, 即：

最大化: $\sum_{i=1}^{n} PiXi$，且满足$\sum_{i=1}^{n} WiXi \le C$。这里 Xi={0，1}，1<=i<=n。Xi=1 表示物品 i

被装入背包，Xi=0 表示物品未装入背包。

贪婪算法通过一系列的选择得到问题的解，在每一次总是做出在当前状态下看来是最好的选择，也就是希望通过局部的最优来达到一个全局的最优。这种启发式的策略并不总能获得最优解，然而在许多情况下确能达到预期的目的。而且对于很多N- P 问题来说，本身就不存在最优解。

2.1.2 贪婪策略

对于 0- 1 背包问题来说，贪婪的策略有常用的四种。每种贪婪策略都采用多步过程来完成背包的装入。在每一步过程中利用贪婪准则选择一个物品装入背包。

一种贪婪准则为：从剩余的物品中，选出可以装入背包的价值最大的物品，利用这种规则，价值最大的物品首先被装入(假设有足够容量)，然后是下一个价值最大的物品，如此继续下去。这种策略不能保证得到最优解。

第二种准则从剩下的物品中选择可装入背包的重量最小的物品，如此继续下去直到不能满足条件为止。在一般情况下不一定能得到最优解。

第三种是价值密度(价值重量比 Pi/W i) 贪婪算法，这种选择准则为：从剩余物品中选择可装入包的 Pi/W i 值最大的物品。这种策略也不能保证得到最优解。

第四种则为启发式贪婪算法。

2.2 算法描述

2.2.1 前三种贪婪策略

（1）忽略先前的物品排序，按照各贪婪策略的要求进行非升序排列

（2）重复以下步骤，直到不满足条件为止：对于当前的物品，如果重量小于包中剩余的容量，则放入，并置物品标志为1(表明被选中) ，否则就停止。

2.2.2 启发式贪婪策略算法

（1），（2）两步和贪婪算法的过程是一样的，这里就不再重复。

（3） 在{1，2，，n} 里取一个子集S ，其中的元素个数小于等于k(文中k = 2) ，计算其重量和价值，如果重量小于背包中的容量，则在剩下的物品中，按价值密度从大到小装入包中，直到背包不能装下物品为止，算出背包中的总价值和对应的解。否则转4) 。

（4） 如果子集还没有取完，转3) 。

（5） 在所有的解中，找出最优的解，即是算法的最终解。

3 详细设计

3.1 类设计

3.1.1Qvector 类

```
class Qvector{
   public:
    struct Qpair; //Qvector内部类，用于保存各物品的重量，价值等信息。
    private:
    int theSize;
    int theCapacity;
    Qpair *objects;
    public:
    Qvector(int i=0):theSize(0),theCapacity(theSize+15); //构造函数
    ~Qvector(); //析构函数
    const Qvector& operator=(const Qvector &rhs); //重运算符载=（参数必须为引用，
    解释略）
    Qvector( Qvector &rhs):objects(NULL); //拷贝构造函数
    Qpair & operator[](int index); //重载运算符[]
    const Qpair& operator[](int index)const//运算符重载[]
    int size()const;
    void push_back(struct Qpair x);
    void sort(int s); //根据所选取的贪心策略，进行排序，并输出结果
    private :
    void initSel(); //对objects的各个Qpair对象进行恢复至初始值。
    void select(int s)//对输入的每个物品进行按某种策略进行排序（冒泡排序）
    void Greedy(); //进行贪心选择
   void outPut(int s); //对结果进行输出
   void Deal(Qvector& temp)//启发贪婪算法的处理函数
   void Rec(Qvector& temp,int po,int rt)//启发式贪婪算法的从po到temp.size()-1中
   选取rt个Qpair对象
};
```

3.1.2 内部类 Qpair

```
struct Qpair{//Qvector内部类
       private:
          char sym;//物品的标识（如：'a','b','c','d','e','f'）
          double w;//物品的重量
          double v;//物品的价值
```

```
        double avg;//avg=w/v
        bool sel;//表示物品是否装入背包,0表示没有装入，1表示装入
    public:
     Qpair(); //构造函数
     Qpair(const Qpair &rhs); //拷贝构造函数
     Qpair& operator=(const Qpair &rhs); //重载运算符=
     double getW()const; //获取物品的质量
     double getV()const; //获取物品的价值
     char getSym(); //获取物品的标识符
     double getAvg()const;//获取物品的单位价值
     double getSel()const;/获取物品是否装入背包
     void setSym(char ch);//设置物品的标识符
     void setAvg(bool b);//设置物品的单位价值

     void setW(double w);//设置物品的重量

     void setV(double v);//设置物品的价值

     void setSel(bool b);//设置物品是否装入背包

     bool lessThan(const Qpair& rhs,int i);//i用来表示用那个数据成员进行比
      较
    friend ostream& operator<<(ostream& out,const Qpair &rhs);//重运算符载<<
};
```

3.2main 函数及其流程图

3.2.1main 函数

```
int main()
{
    cout<<"/*贪婪算法解决0/1背包问题__0904班__秦超*/"<<endl<<endl;
    Qvector vec;
    char symbol;//物品的表示符
    double weight,value,Max;
    cout<<"请输入背包的容量（如：50）"<<endl;
        cin>>Max; //Max为背包的最大容量
    while(Max<=0)
    {
        cout<<"输入错误，请重新输入。"<<endl;
        cin>>Max;
    }
```

```
    vec.push_back(Qvector::Qpair('#',Max,0));
    //vec[0].w表示经过每一次贪心选择后背包的剩余容量
    //vec[0].v用来存放其每次贪心选择后的总价值
    cout<<"请输入每个物品的标识符，重量以及其价值，最后以(#,0,0)表示输入结束。
        "<<endl
        <<"（如：(A 20 60)，(B 10 70)，(C 30 120)，(#,0,0)）"<<endl;
    cin>>symbol>>weight>>value;
    while(symbol!='#')//以#表示输入结束
    {
        if((weight>0)&&value>=0)
            vec.push_back(Qvector::Qpair(symbol,weight,value));
        else
            cout<<"输入错误，输入不满足条件：weight>0 and value>=0;请重新输入!
               "<<endl;
        cin>>symbol>>weight>>value;
    }
    int s;
    cout<<"请选择贪心策略(0,1,2,3):"<<endl
        <<"(0)使用所有贪心策略"<<endl
        <<"(1)贪心策略：选取重量最小."<<endl
        <<"(2)贪心策略：选取价值最大者."<<endl
        <<"(3)贪心策略: 选取单位重量价值最大的物品."<<endl
        <<"*(4)贪心策略：启发式贪婪算法."<<endl;
    cin>>s;
    while((s>4)||(s<0))
    {
        cout<<"输入错误，请重新输入（0，1，2，3,4）："<<endl;
        cin>>s;
    }
    cout<<endl;
    vec.sort(s);
    //对输入的每个物品按(Value/Weight)的大小进行排序，并按相应策略进行贪心选择，
    最后输出结果。
    return 0;
}
```

3.2.2main()函数流程图

3.3 关键函数设计

由于前三种贪婪策略思想简单，容易实现，故不再加以说明，仅对启发式贪婪算法进行解释说明。

（1）Rec()函数

```
//启发式贪婪算法的从 po 到 temp.size()-1 中选取 rt 个 Qpair 对象
void Rec(Qvector& temp,int po,int rt)
    {
        if(0==rt)//递归结束条件
        {
            Deal(temp);
            return;
        }
        if((temp.size()-po)==rt)//递归结束条件
        {
            for(int i=po;i<temp.size();++i)
            {
                temp[i].setSel(true);
            }
            Deal(temp);
        return;
        }
      temp[po].setSel(true);
      Rec(temp,po+1,rt-1);
      for(int k2=po+1;k2<temp.size();++k2)//注意
          temp[k2].setSel(false);
      temp[po].setSel(false);
      Rec(temp,po+1,rt);
      for(int k3=po+1;k3<temp.size();++k3)//注意
          temp[k3].setSel(false);
    }
```

（2）Deal()函数

```
//每当从 n 个 Qpair 选择 k 个对象，执行启发式贪婪算法的第三步（详细说明见 2.2.2）。
 void Deal(Qvector& temp)
{
        double tw=0,tv=0;
        int initW=0,initV=0;
        initW=temp[0].getW();
        initV=temp[0].getV();
        for(int i=1;i<temp.size();++i)
        {
            if(temp[i].getSel())
            {
                tw+=temp[i].getW();
```

```
                tv+=temp[i].getV();
            }
        }
        //当 tw>temp[0].getW()时，丢弃。
        if(tw<=temp[0].getW())//如果 S 子集物品总重量小于背包容量
        {    temp[0].setW(temp[0].getW()-tw);
            temp[0].setV(temp[0].getV()+tv);
            if(ttemp[0].getW()>0)
            {

                for(int j=1;j<temp.size();++j)//进行贪心选择
                {
                    if(temp[j].getSel())
                        continue;
                     //当物品 j 的重量小于当前背包的容量并且价值大于 0 时。
                    if((temp[j].getW()<=temp[0].getW())&&(temp[j].getV()>0))
                        {
                     temp[0].setW(temp[0].getW()-temp[j].getW());
                     temp[0].setV(temp[0].getV()+temp[j].getV());
                     temp[j].setSel(true);
                    }
                }
                if((*this)[0].getV()<temp[0].getV())//最优解保存在（*this）中。
                {
                    (*this)=temp;
                }
                temp[0].setW(initW);//对 temp[0]进行恢复
                temp[0].setV(initV);
            }
        }
       /*for(int ss=1;ss<temp.size();++ss)//主要用来验证运算过程的正确性。
            cout<<temp[ss].getSel()<<' ';
        cout<<endl;*/
}
```

4 源程序清单

见附录 1，2。

5 系统实施、调试

5.1 设计问题

```
 Qpair(const Qpair &rhs1)//拷贝构造函数
{
 operator=(rhs1);
}
//重载运算符=（参数必须为引用)
//解释：如果重载运算符=（参数必须不是引用)例如在如下情况下：
//Qpair q1(参数略);
//Qpair q2=q1；(此时调用拷贝构造函数，过程如下：）
//Qpair&rhs1=q1;
//Qpair rhs2=rhs1;调用拷贝构造函数，如此便构成死循环
Qpair& operator=(const Qpair &rhs2)
{//实现略。
    return *this;
}
```

（此示例见附录 3）

6 运行截图

（1）输入背包容量以及各物品的重量及其价值

```
qinchao@qinchao-Lenovo-G450:~/Q10$ cd test
qinchao@qinchao-Lenovo-G450:~/Q10/test$ g++ -o Qin QinCh.cpp
qinchao@qinchao-Lenovo-G450:~/Q10/test$ ./Qin
/*贪婪算法解决0/1背包问题__0904班__秦超*/

请输入背包的容量（如：50）
50
请输入每个物品的标识符，重量以及其价值，最后以(#,0,0)表示输入结束。
（如：（A 20 60），（B 10 70），（C 30 120），（#,0,0）
a 10 70
b 20 60
c 30 120
# 0 0
```

（2）选择贪婪策略

```
请选择贪心策略(0,1,2,3):
(0)使用所有贪心策略
(1)贪心策略：选取重量最小.
(2)贪心策略：选取价值最大者.
(3)贪心策略：选取单位重量价值最大的物品.
*(4)贪心策略：启发式贪婪算法.
4

//贪心策略4：启发式贪婪算法(isSelected表示是否将该物品装入背包):
Symbol          Weigh           Value           Value/Weight    isSelected
a               10              70              7               1
c               30              120             4               1
b               20              60              3               0

背包的剩余容量为: 10
装入背包的物品数量为：2
装入背包物品的最大总价值为: 190

重新选择贪心策略？Y/N
```

（3）是否重新选择贪婪策略

```
重新选择贪心策略？Y/N
y
请选择贪心策略(0,1,2,3):
(0)使用所有贪心策略
(1)贪心策略：选取重量最小.
(2)贪心策略：选取价值最大者.
(3)贪心策略：选取单位重量价值最大的物品.
*(4)贪心策略：启发式贪婪算法.
3

//贪心策略3：选取单位重量价值最大的物品(isSelected表示是否将该物品装入背包):
Symbol          Weigh           Value           Value/Weight*   isSelected
a               10              70              7               1
c               30              120             4               1
b               20              60              3               0

背包的剩余容量为: 10
装入背包的物品数量为：2
装入背包物品的最大总价值为: 190

重新选择贪心策略？Y/N
n
qinchao@qinchao-Lenovo-G450:~/Q10/test$
```

（4）选择启发式贪婪策略

```
请输入背包的容量（如：50）
11
请输入每个物品的标识符，重量以及其价值，最后以(#,0,0)表示输入结束。
（如：（A 20 60），（B 10 70），（C 30 120），（#,0,0）
a 2 6
b 4 10
c 6 12
d 7 13
# 0 0
请选择贪心策略(0,1,2,3):
(0)使用所有贪心策略
(1)贪心策略：选取重量最小.
(2)贪心策略：选取价值最大者.
(3)贪心策略：选取单位重量价值最大的物品.
*(4)贪心策略：启发式贪婪算法.
4

//贪心策略4：启发式贪婪算法(isSelected表示是否将该物品装入背包):
Symbol          Weigh           Value           Value/Weight    isSelecte
d
a               2               6               3               0
b               4               10              2.5             1
c               6               12              2               0
d               7               13              1.85714         1

背包的剩余容量为：0
装入背包的物品数量为：2
装入背包物品的最大总价值为：23

重新选择贪心策略？Y/N
d
重新选择贪心策略？Y/N
n
qinchao@qinchao-Lenovo-G450:~/Q10/test$
```

7 总结

通过本次课程设计，首先认识到了自己的不足。在编码的过程中认识到了自己C/C++一些细节方面的不足，在以后高级语言的学习过程中在理解的同时还要做到对细节的注重；当然在本次课设中也有很多值得肯定的地方，比如：自己采用递归方法实现了从n个数中选取m个的算法（当然m<=n）。同时，觉得本次课设是十分有意义的，使自己所学的只是有了实践的地方。

8 参考文献

【1】林鑫. 基于0-1背包问题的讨论. 微机发展, 2005