东南大学
网络空间安全学院

算法作业 #3

学生姓名: 李盛; 学号: *229221*

Course: 算法设计与分析 – Professor: 陶军
Due date: *4* 月 *15* 日, *2022* 年

第一题

给定 n 个物品, 物品价值分别为 $P_1, P_2, \ldots, Pn$, 物品重量分别 $W_1, W_2, \ldots, Wn$, 背包容量为 M_0 每种物品可部分装入到背包中。输出 $X_1, X_2, \ldots, X_n, 0 \leq X_i \leq 1$, 使得 $\sum_{1\leq i} \leq nP_iX_i$ 最大，且 $\sum_{1\leq i\leq n} W_iX_i \leq M$ 。试设计一个算法求解该问题，分析算法的正确性。

(a) 本题是“部分背包”问题，跟 01 背包问题有些不同，下面介绍一下异同点:

(b) 首先介绍 0-1 背包问题。假设一共有 N 件物品，第 i 件物品的价值为 V_i，重量为 W_i。一个小偷有一个最多只能装下重量为 W 的背包，他希望带走的物品越有价值越好，请问: 他应该选择哪些物品?

0-1 背包问题的特点是: 对于某件 (更适合的说法是: 某类) 物品，要么被带走 (选择了它)，要么不被带走 (没有选择它)，不存在只带走一部分的情况。

0-1 背包问题中的物品想象的一个金子，你要么把它带走，要么不带走它; 而部分背包问题中的物品则是一堆金粉末，可以取任意部分的金粉末。

(c) 部分背包问题可以用贪心算法求解，且能够得到最优解。

贪心策略是什么呢? 将物品按单位重量所具有的价值排序。总是优先选择单位重量下价值最大的物品。

单位重量所具有的价值: V_i/W_i。选择性价比最高的，并按从大到小排序，依次放入背包直至背包放满。

(d) 正确性证明: 使用该贪心策略，可以获得最优解。在这里，最优解就是带走的物品价值最大。分三个部分进行证明: (1) 证明符合贪心选择的特性 (2) 证明符合归纳法结构 (Inductive Structure)(3) 证明最优子结构 (Optimal Substructure)。

(e) 证明 (1) 符合贪心选择的特性: 让物品 i^* 为最高密度的物品，假设这里存在一个最优解用到了 $\omega = min(\omega_i, M)$ 这么多重量的物品 i^*, 令 π^* 为最优解，如果 π^* 用到了 $\omega = min(\omega_i, M)$ 多的物品 i^*，那么已经证明了我们的第一个选择 (Greedy Choice，First Choice) 包含在某些最优解中。如果 π^* 没有用到 $\omega = min(\omega_i, M)$ 多的物品 i^*，那么我们可以将背包中任意一部分和物品 i^* 进行交换，因为我们是按照物品密度排序，而且我们已经定义物品 i^* 为最高密度的物品，所以与物品 i^* 交换的物品必然比物品 i^* 的密度要小，所以等量交换后，我们得到的解 π' 会比 π^* 更优，这个结论与和 π^* 为最优解的假设冲突，所以最优解 π^* 中必然含有 $\omega = min(\omega_i, M)$ 多的物品 i^*。因此符合贪心选择的特性 (Greedy Choice Property)。

(*f*) 证明 (2) 符合归纳法结构 (Inductive Structure)：在完成第一个选择（贪心选择 $\hat{c}$ 之后，子问题 P' 和原问题 P 还是同一类问题，意味着我们的选择不改变问题的结构，并且子问题的解可以和第一个选择（贪心选择）$\hat{c}$ 合并。证明：子问题含有除了物品 i^* 之外的所有物品，背包容量变成了 $M' = M - \omega_i^*$，因此物品 i^* 可以和子问题 P' 的解合并。

(*g*) 证明 (3) 最优子结构（Optimal Substructure）：定义 P 为原问题，P' 为在完成第一个选择（贪心选择）$\hat{c}$ 之后的子问题,π' 为子问题 P' 的最优解，那么 $\pi = \pi' \cup \hat{c}$ 为原问题 P 的最优解。证明：让 $v^* = w \cdot d_i$ 为贪心选择 $\hat{c}$ ((是 $w = min(w_i, M)$), 那么 value(π)=value(π') + v^*。假设 π 不是最优解，有一个其他的最优解 π^*，因为我们已经证明了算法符合贪心选择的特性，所以我们知道最优解 π^* 中一定含有贪心选择 $\hat{c}$。那么 $\pi^* - \hat{c}$ 就应该是子问题 P' 的解, 所以 $value(\pi^* - \hat{c}) = value(\pi^*) - v^* > value(\pi) - v^* = value(\pi')$。但是这与 π' 为子问题 P' 的最优解的定义产生冲突，所以 $\pi = \pi' \cup \hat{c}$ 不可能不是最优解，因为 $= \pi' \cup \hat{c}$ 为原问题 P 的最优解。

(*h*) Implementation code

```
int main()
{
    char name[7] . CA',.0','C','D',1E.,'F','61;
    double W;
    cin>>W;
    int n;
    cin>>n;
    double value[n],weight[n];
    for(int i = 0;i<n ;i++)
        cin»value[i]>>weight[i];
    deal(value,weight,n);
    sort(value,n);
    int j = 0;
    while(j<n) {
        if(W >= weight[j])
            W -= weight[j];
        cout<<name[j]<<" "<<weight[j]<<endl;
    }
    return 0;
}

void deal(double *value,double •weight,int n) {
    for(int i = 0;i<n;i++)
        value[i] /- weight[i];
}

void sort(double svalue,int n) {
    for(int i = 0;i<n-1;i++) {
        for(int j = 0;j<n-i-1;j++) {
            if(value[j]<value[j+1]) swap(value[j],value[j+1]);
        }
    }
}
```

第二题

海面上有一些船需要与陆地进行通信，需要在海岸线上布置一些基站。现将问题抽象为, 在 x 轴上方, 给出 N 条船的坐标 $p_1, p_2, \ldots, p_N, p_i = (x_i, y_i), x_i \geq 0, y_i \leq \mathrm{d}, 1 \leq i \leq N$ ，在 x 轴上安放的基站可以覆盖半径为 d 的区域内的所有点, 问在 x 轴上至少要安放几个点才可以将 x 轴上方的点都覆盖起来。试设计一个算法求解该问题, 并分析算法的正确性。

Answer.

(a) 首先将所有的点按横坐标从小到大进行排序。当点集不空的时候，每次取出最左边的点，将该点视做圆周上的点，以该点到 x 轴上距离为 d 的点 (位于右边) 作为圆心，以 d 作为半径，画出一个圆，然后去除掉包含在该圆内的点。然后继续选出一个最左的点，重复以上操作，直至点集为空，表示所有点都被覆盖，此时得出的圆的个数就是所求答案。

(b) 证明：(1) 最优子结构的证明：假设船的坐标集合 $S = x_1, x_2, ..., x_n$ 已经按 x 轴从小到大排完序。假设该问题的最优解为 $O(1,i) = a_1, a_2, ..., a_i$;$a_1, a_2, ..., a_i$ 是排好序的，设 s'=$x_m, x_2, ..., x_n$, 即 S' 为 S 除去第一个圆中包含的点构成的集合，即 S' 为 S 的一个子问题，则 $O(2,i) = a_2, ..., a_i$ 为其最优解；现在假设存在 $O^{'}(2,k)$=$a_2, ..., a_k$ 为 S' 的最优解，则 $Size(O'(2,k)) < Size(O(2,i))$，则 $O'(2,k) \bigcup a_1$ 为 S 的最优解，因为 $Size(O'(2,k)) + 1 < Size(O(2,i)) + 1 = size(O(1,i))$, 因此与 O(1,i) 为 S 的最优解相矛盾，因此算法具有最优子结构的性质。

(c) 证明：(2) 贪心选择性证明：假设船的坐标集合 $S = x_1, x_2, ..., x_n$ 已经按 x 轴从小到大排完序。第一次选取最小的横坐标 x_1 作为最左侧圆周上的点，去掉第一个圆中包含的点所构成的集合为 $S^{'} = x_i, x_{i+1}, ..., x_n$，此时选取 x_i 作为左侧圆周上的点，重复上述操作。因此每次选取子问题中最小的点可以得到问题的最优解 $O = a_1, a_2, ..., a_i$，既可以覆盖当前点，又可以尽可能多的覆盖右边的点，因此因该贪心的选择它，这就证明了如果用贪心策略来进行选择，得到的是最优解，从而证明了贪心算法的正确性。

(d) Implementation code

```
public class Point implements Comparable{
  private double x;
  private double y;

  public void setX(double x) {
    this.x = x;
  }

  public double getX() {
    return x;
  }

  public double getY() {
    return y;
  }
```

```
    public void setY(double y) {
      this.y * y;
    }

    public Point(double x, double y) {
      this.x = x;
      this.y * y;
    }

    public double getDistance(Point o){
      double distance * Math.scirt(Math.pow(x-o.getX0,2)
                                   +Math.pow(y-o.getY0,2));
      return distance;
    }
    @Override
    public int compareTo(Point o) {
     return (int) (this.x - o.getX());
}

public class pointCover {
    public static int count = 0 ;
    public static int d = 4;
    public static void main(String[] agrs){
        List list = new ArrayList<>();
        list.add(new Point(-10,4));
        list.add(new Point(7,3.5));
        list.add(new Point(5,3.1));
        list.add(new Point(1,2));
        list.add(new Point(-4,3));
        list.add(new Point(-2,4));
        //sort by x-coordinate
        collections.sort(list);
        System.out.println("All points are:");
        for(Point op : list)
          System.out.print(op+" ");
        System.out.println();
        List result = new ArrayList<>();

        Point p;
        while((p = getTop(list))!=null )(
          // Take the leftmost point
          double cent = p.getX()+
          Math.sort(Math.pow(d,2)-Math.pow(p.gert(),2));
          Point centerP = new Point(centX,0);
          result.add(centerP);
          count++;
          Iterator it = list.iterator();
          while(it.hastNext())(
            if(centerP.getDistance(it.next())<=d)
```

```
                    it.remove();
                }
            }
            System.out.println("Number of circles:"+count);
            system.out.print("Center of circles: ");
            for(Point o : result)
              System.out.print(" "+o);
            System.out.println();
            System.out.println("Radius: "+d);
        }

        public static Point getTop(List list)(
        if(list.size()>0) return list.get(0); else return null; )
    }
```