算法设计与分析

回溯法作业

课程名称：	算法设计与分析
姓名：	凌晨
学院：	软件学院
专业：	软件工程
学号：	2214414320

2023 年 10 月 28 日

目录

一、 5-5 **3**

1. 题目描述 . . . 3
2. 问题 1 . . . 3
3. 问题 2 . . . 3

二、 5-6 **6**

1. 题目描述 . . . 6
2. 问题 1 . . . 6
3. 问题 2 . . . 7

三、 5-21 **9**

1. 题目描述 . . . 9
2. 分析 . . . 10
3. 实现代码 . . . 10
4. 运行结果 . . . 12
5. 复杂度分析 . . . 14

一、 5-5

1. 题目描述

5-5 设 G 是一个有 n 个顶点的有向图，从顶点 i 发出的边的最大费用记为 $\max(i)$。

（1）证明旅行售货员回路的费用不超过 $\sum_{i=1}^{n}\max(i)+1$。

（2）在旅行售货员问题的回溯法中，用上面的界作为 bestc 的初始值，重写该算法，并尽可能地简化代码。

2. 问题 1

证明. 不妨假设旅行售货员回路的费用 p' 超过了 $\sum_{i=1}^{n} max(i)+1$

由于 max(i) 的定义，我们可以假设存在一条这样的回路，即每次售货员选择下一个地点的时候，均选择最大费用的路，而且这样的选择满足旅行售货员回路，这样的回路 p 的最大费用为 $\sum_{i=1}^{n} max(i)$。

由于 p'>p，说明 p' 回路一定不满足题目要求，走了回头路，因此假设不成立。

旅行售货员回路的费用 p' 一定不超过了 $\sum_{i=1}^{n} max(i)+1$ 证毕 □

3. 问题 2

为了进行对比，先给出书上的源代码：

```java
package src;

public class Bttsp {
    static int n; // 图G的顶点数
    static int[] x; // 当前解
    static int[] bestx; // 当前最优解
    static float bestc; // 当前最优值
    static float cc; // 当前费用
    static float[][] a; // 图G的邻接矩阵

    public static float tsp(int[] v){
        // 置x为单位排列
        x = new int[n+1];
        for (int i = 1; i <= n ; i++) {
            x[i] = i;
        }
        bestc = Float.MAX_VALUE;
        bestx = v;
        cc = 0;
        // 搜索x[2:n]的全排序
        backtrack(2);
        return bestc;
```

```
    }

    private static void backtrack(int i){
        if (i == n) {
            if (a[x[n-1]][x[n]] <Float.MAX_VALUE
                    && a[x[n]][1]<Float.MAX_VALUE
                    && (bestc==Float.MAX_VALUE || cc+a[x[n-1]][x[n]]+a[x[n]][1]<bestc) ) {
                for (int j = 1; j <= n ; j++) {
                    bestx[j] = x[j];
                }
                bestc = cc +a[x[n-1]][x[n]]+a[x[n]][1];
            }
        }
        else {
            for (int j = i; j <= n; j++) {
                // 是否可进入x[j]子树
                if (a[x[i-1]][x[j]] < Float.MAX_VALUE
                        && (bestc== Float.MAX_VALUE || cc+a[x[i-1]][x[j]]<bestc)){
                    swap(x,i,j);
                    cc+=a[x[i-1]][x[i]];
                    backtrack(i+1);
                    cc-=a[x[i-1]][x[i]];
                    swap(x,i,j);
                }
            }
        }
    }

    private static void swap(int[] x,int i,int j){
        int temp = x[i];
        x[i] = x[j];
        x[j] = temp;
        return;
    }

}
```

下面为将 $\sum_{i=1}^{n} max(i)+1$ 作为 bestc 的初始值，重写的算法：

```
package src;

public class Bttsp5_5 {
    static int n; // 图G的顶点数
    static int[] x; // 当前解
    static int[] bestx; // 当前最优解
    static float bestc; // 当前最优值
    static float cc; // 当前费用
    static float[][] a; // 图G的邻接矩阵
```

```java
static float MAX_VALUE;

public static float tsp(int[] v){
    // 置x为单位排列
    x = new int[n+1];
    for (int i = 1; i <= n ; i++) {
        x[i] = i;
    }
    // 设置上界
    MAX_VALUE = maxC();
    bestc = MAX_VALUE;
    bestx = v;
    cc = 0;
    // 搜索x[2:n]的全排序
    backtrack(2);
    return bestc;
}

private static void backtrack(int i){
    if (i == n) {
        if (a[x[n-1]][x[n]] <MAX_VALUE
                && a[x[n]][1]<MAX_VALUE
                && cc+a[x[n-1]][x[n]]+a[x[n]][1]<bestc){
            for (int j = 1; j <= n ; j++) {
                bestx[j] = x[j];
            }
            bestc = cc +a[x[n-1]][x[n]]+a[x[n]][1];
        }
    }
    else {
        for (int j = i; j <= n; j++) {
            // 是否可进入x[j]子树, 与源程序进行了改变
            if (a[x[i-1]][x[j]] < MAX_VALUE
                    && cc+a[x[i-1]][x[j]]<bestc){
                swap(x,i,j);
                cc+=a[x[i-1]][x[i]];
                backtrack(i+1);
                cc-=a[x[i-1]][x[i]];
                swap(x,i,j);
            }
        }
    }
}

private static void swap(int[] x,int i,int j){
    int temp = x[i];
    x[i] = x[j];
    x[j] = temp;
    return;
}
```

```
    // 得到上界
    private static float maxC(){
        float sum = 0;
        for (int i = 1; i <= n; i++) {
            float t = a[i][0];
            for (int j = 1; j <= n; j++) {
                t = Math.max(t,a[i][j]);
            }
            sum+=t;
        }
        return sum+1;
    }

}
```

二、 5-6

1. 题目描述

5-6 设 G 是一个有 n 个顶点的有向图，从顶点 i 发出的边的最小费用记为 $\min(i)$。

（1）证明图 G 的所有前缀为 $x[1:i]$的旅行售货员回路的费用至少为：

$$\sum_{j=2}^{i} a(x_{j-1},x_j)+\sum_{j=i}^{n}\min(x_j)$$

155

式中，$a(u,v)$是边(u,v)的费用。

（2）利用上述结论设计一个高效的上界函数，重写旅行售货员问题的回溯法，并与教材中的算法进行比较。

2. 问题 1

证明. 易得，走过前缀为 x[1:i] 的费用一定为 $\sum_{j=2}^{i} a(x_{j-1},x_j)$

假设剩下的 x[i+1:n] 费用 p' 少于 $\sum_{j=i}^{n} min(x_j)$。

由于 min(i) 的定义，我们可以假设存在一条这样的回路，即每次售货员选择下一个地点的时候，均选择最小费用的路，而且这样的选择满足旅行售货员回路，这样的回路 p 的费用为 $\sum_{j=i}^{n} min(x_j)$

由于 p'<p，说明 p' 回路一定不满足题目要求，即存在没有到达的点，因此假设不成立。

所以，所有前缀为 x[1:i] 的旅行售货员回路的最小费用至少为 $\sum_{j=2}^{i} a(x_{j-1}, x_j) + \sum_{j=i}^{n} min(x_j)$

证毕 □

3. 问题 2

改变在注释中有写，不再赘述，下面为代码：

```
package src;

public class Bttsp5_6 {
    static int n; // 图G的顶点数
    static int[] x; // 当前解
    static int[] bestx; // 当前最优解
    static float bestc; // 当前最优值
    static float cc; // 当前费用
    static float[][] a; // 图G的邻接矩阵
    static float[] mina; // 记录a每一行最小的值

    static float MAX_VALUE;

    public static float tsp(int[] v){
        // 置x为单位排列
        x = new int[n+1];
        for (int i = 1; i <= n ; i++) {
            x[i] = i;
        }
        // 设置上界
        MAX_VALUE = maxC();
        // 设置mina
        setMina();
        // 下面为源代码的程序
        bestc = MAX_VALUE;
        bestx = v;
        cc = 0;
        // 搜索x[2:n]的全排序
        backtrack(2);
        return bestc;
    }

    private static void backtrack(int i){
        if (i == n) { // 到达边界
            // 该判断条件不变
            if (a[x[n-1]][x[n]] <MAX_VALUE
                    && a[x[n]][1]<MAX_VALUE
                    && cc+a[x[n-1]][x[n]]+a[x[n]][1]<bestc){
                for (int j = 1; j <= n ; j++) {
                    bestx[j] = x[j];
                }
```

```java
                bestc = cc +a[x[n-1]][x[n]]+a[x[n]][1];
            }
        }
        else {
            for (int j = i; j <= n; j++) {
                // 是否可进入x[j]子树，该判断条件可以进行改变，即进行剪枝
                if (a[x[i-1]][x[j]] < MAX_VALUE
                        && cc+a[x[i-1]][x[j]]+calRest(i)<bestc){
                    swap(x,i,j);
                    cc+=a[x[i-1]][x[i]];
                    backtrack(i+1);
                    cc-=a[x[i-1]][x[i]];
                    swap(x,i,j);
                }
            }
        }
    }

    private static void swap(int[] x,int i,int j){
        int temp = x[i];
        x[i] = x[j];
        x[j] = temp;
        return;
    }

    private static float maxC(){
        float sum = 0;
        for (int i = 1; i <= n; i++) {
            float t = a[i][0];
            for (int j = 1; j <= n; j++) {
                t = Math.max(t,a[i][j]);
            }
            sum+=t;
        }
        return sum+1;
    }

    private static void setMina() {
        mina = new float[n + 1];
        for (int i = 1; i <= n; i++) {
            float t = a[i][0];
            for (int j = 1; j <= n; j++) {
                t = Math.min(t, a[i][j]);
            }
            mina[i] = t;
        }
        return;
    }

    private static float calRest(int i){
```

```
        float sum = 0;
        for (int j = i; j <= n ; j++) {
            sum += mina[x[j]];
        }
        return sum;
    }
}
```

分析如下：

与源代码相比，改变的核心为：

```
if (a[x[i-1]][x[j]] < MAX_VALUE
    && cc+a[x[i-1]][x[j]]+calRest(i)<bestc){
        swap(x,i,j);
        cc+=a[x[i-1]][x[i]];
        backtrack(i+1);
        cc-=a[x[i-1]][x[i]];
        swap(x,i,j);
    }
```

与教材中的算法比较，通过引入额外计算时间的计算后面 $\sum_{j=i}^{n} min(x_j)$，若不满足条件，则进行剪枝，节约了大量时间。

若在 i 处进行了剪枝，额外的计算时间为 $O(n)$，但剪枝节约的时间为 $O((n-i+1)!)$。因此，该算法减少了运行时间，但该算法的时间复杂度与教材中的算法还是保持一致，为 $O(n!)$。这是由于回溯法的本质导致的！

三、 5-21

1. 题目描述

5-21 子集树问题。

问题描述：试设计一个用回溯法搜索子集空间树的函数。该函数的参数包括结点可行性判定函数和上界函数等必要的函数，并将此函数用于解装载问题。

装载问题描述如下：有一批共 n 个集装箱要装上艘载重量为 c 的轮船，其中集装箱 i 的重量为 w_i。找出一种最优装载方案，将轮船尽可能装满，即在装载体积不受限制的情况下，将尽可能重的集装箱装上轮船。

算法设计：对于给定的 n 个集装箱的重量和轮船的重量，计算最优装载方案。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 2 个正整数 n 和 c。n 是集装箱数，c 是轮船的载重量。接下来的 1 行中有 n 个正整数，表示集装箱的重量。

结果输出：将计算的最大装载重量输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
5 10	10
7 2 6 5 4	

2. 分析

该问题和 0-1 背包异曲同工之妙，比 0-1 背包可以省略体积计算。
由于书上已经有分析，代码中也有详细的注释，不再给出解释。

3. 实现代码

```java
package src;

public class Bttsp5_6 {
    static int n; // 图G的顶点数
    static int[] x; // 当前解
    static int[] bestx; // 当前最优解
    static float bestc; // 当前最优值
    static float cc; // 当前费用
    static float[][] a; // 图G的邻接矩阵
    static float[] mina; // 记录a每一行最小的值

    static float MAX_VALUE;

    public static float tsp(int[] v){
        // 置x为单位排列
        x = new int[n+1];
        for (int i = 1; i <= n ; i++) {
            x[i] = i;
        }
        // 设置上界
        MAX_VALUE = maxC();
        // 设置mina
        setMina();
        // 下面为源代码的程序
        bestc = MAX_VALUE;
        bestx = v;
        cc = 0;
        // 搜索x[2:n]的全排序
        backtrack(2);
        return bestc;
    }

    private static void backtrack(int i){
        if (i == n) { // 到达边界
            // 该判断条件不变
            if (a[x[n-1]][x[n]] <MAX_VALUE
                    && a[x[n]][1]<MAX_VALUE
                    && cc+a[x[n-1]][x[n]]+a[x[n]][1]<bestc){
                for (int j = 1; j <= n ; j++) {
                    bestx[j] = x[j];
                }
                bestc = cc +a[x[n-1]][x[n]]+a[x[n]][1];
```

```
        }
    }
    else {
        for (int j = i; j <= n; j++) {
            // 是否可进入x[j]子树，该判断条件可以进行改变，即进行剪枝
            if (a[x[i-1]][x[j]] < MAX_VALUE
                    && cc+a[x[i-1]][x[j]]+calRest(i)<bestc){
                swap(x,i,j);
                cc+=a[x[i-1]][x[i]];
                backtrack(i+1);
                cc-=a[x[i-1]][x[i]];
                swap(x,i,j);
            }
        }
    }
}

private static void swap(int[] x,int i,int j){
    int temp = x[i];
    x[i] = x[j];
    x[j] = temp;
    return;
}

private static float maxC(){
    float sum = 0;
    for (int i = 1; i <= n; i++) {
        float t = a[i][0];
        for (int j = 1; j <= n; j++) {
            t = Math.max(t,a[i][j]);
        }
        sum+=t;
    }
    return sum+1;
}

private static void setMina() {
    mina = new float[n + 1];
    for (int i = 1; i <= n; i++) {
        float t = a[i][0];
        for (int j = 1; j <= n; j++) {
            t = Math.min(t, a[i][j]);
        }
        mina[i] = t;
    }
    return;
}

private static float calRest(int i){
    float sum = 0;
```

```java
        for (int j = i; j <= n ; j++) {
            sum += mina[x[j]];
        }
        return sum;
    }
}
```

4. 运行结果

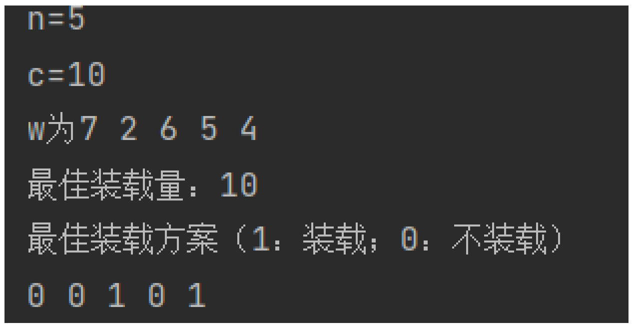
编写了 main 函数进行部分数据测试, 下面为代码

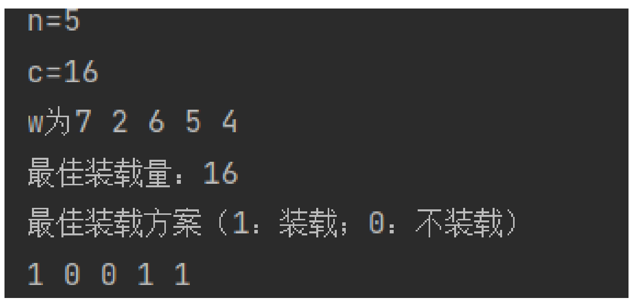
```java
package src;

public class Bttsp5_6 {
    static int n; // 图G的顶点数
    static int[] x; // 当前解
    static int[] bestx; // 当前最优解
    static float bestc; // 当前最优值
    static float cc; // 当前费用
    static float[][] a; // 图G的邻接矩阵
    static float[] mina; // 记录a每一行最小的值

    static float MAX_VALUE;

    public static float tsp(int[] v){
        // 置x为单位排列
        x = new int[n+1];
        for (int i = 1; i <= n ; i++) {
            x[i] = i;
        }
        // 设置上界
        MAX_VALUE = maxC();
        // 设置mina
        setMina();
        // 下面为源代码的程序
        bestc = MAX_VALUE;
        bestx = v;
        cc = 0;
        // 搜索x[2:n]的全排序
        backtrack(2);
        return bestc;
    }

    private static void backtrack(int i){
        if (i == n) { // 到达边界
            // 该判断条件不变
            if (a[x[n-1]][x[n]] <MAX_VALUE
                    && a[x[n]][1]<MAX_VALUE
                    && cc+a[x[n-1]][x[n]]+a[x[n]][1]<bestc){
```

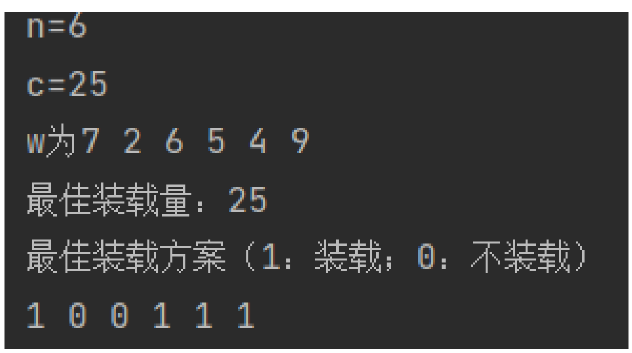
```
            for (int j = 1; j <= n ; j++) {
                bestx[j] = x[j];
            }
            bestc = cc +a[x[n-1]][x[n]]+a[x[n]][1];
        }
    }
    else {
        for (int j = i; j <= n; j++) {
            // 是否可进入x[j]子树，该判断条件可以进行改变，即进行剪枝
            if (a[x[i-1]][x[j]] < MAX_VALUE
                    && cc+a[x[i-1]][x[j]]+calRest(i)<bestc){
                swap(x,i,j);
                cc+=a[x[i-1]][x[i]];
                backtrack(i+1);
                cc-=a[x[i-1]][x[i]];
                swap(x,i,j);
            }
        }
    }
}

private static void swap(int[] x,int i,int j){
    int temp = x[i];
    x[i] = x[j];
    x[j] = temp;
    return;
}

private static float maxC(){
    float sum = 0;
    for (int i = 1; i <= n; i++) {
        float t = a[i][0];
        for (int j = 1; j <= n; j++) {
            t = Math.max(t,a[i][j]);
        }
        sum+=t;
    }
    return sum+1;
}

private static void setMina() {
    mina = new float[n + 1];
    for (int i = 1; i <= n; i++) {
        float t = a[i][0];
        for (int j = 1; j <= n; j++) {
            t = Math.min(t, a[i][j]);
        }
        mina[i] = t;
    }
    return;
```

```
    }

    private static float calRest(int i){
        float sum = 0;
        for (int j = i; j <= n ; j++) {
            sum += mina[x[j]];
        }
        return sum;
    }
}
```

下面为图片展示







可以看到，结果符合预期。

5. 复杂度分析

- 空间复杂度

算法仅设置了长度为 n 的数组，空间复杂度为 O(n);

- 时间复杂度

  要遍历整个递归树，O(2^n)

  要记录最佳装载方案，O(n)

  所以时间复杂度为 O($n2^n$);