

华中科技大学

2022

算法设计与分析实践报告

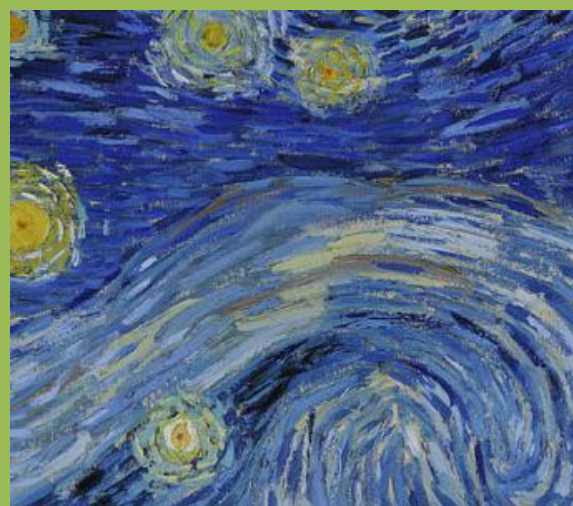
专 业： 计算机科学与技术

班 级： 大数据 2101 班

学 号： U202115638

姓 名： 马耀辉

完成日期： 2022-12-17



华中科技大学课程设计报告

目 录

1. 完成情况	1
2. POJ1753 解题报告	2
2.1 题目分析	2
2.2 算法设计	2
2.3 性能分析	6
2.4 运行测试	6
3. POJ3295 解题报告	7
3.1 题目分析	7
3.2 算法设计	7
3.3 性能分析	10
3.4 运行测试	11
4. POJ1201 解题报告	11
4.1 题目分析	11
4.2 算法设计	12
4.3 性能分析	16
4.4 运行测试	16

华中科技大学课程设计报告

5. POJ3040 解题报告	17
5.1 题目分析	17
5.2 算法设计	17
5.3 性能分析	21
5.4 运行测试	22
6. 总结	22
6.1 实验总结	22
6.2 心得体会和建议	23

华中科技大学课程设计报告

1. 完成情况

在整个实验中，完成题单上 25 个题，实际最后通过 25 个题，其中编程语言使用 Java 语言，测试环境为 WINDOWS10 系统下使用编译器 IntelliJ IDEA 2021.2(Ultimate Edition)进行代码提交前的测试。为了便于在编译器目录列表进行处理，因此题目类名使用题目编号进行命名，在 POJ 平台上测试时需要将类名修改为 Main，同时由于 POJ 的 Java 版本过低，需要去除注解等高版本特性。

User	Problem	Result	Memory	Time	Language
1563081082	1201	Accepted	10756K	5063MS	Java
1563081082	3040	Accepted	1980K	1547MS	Java
1563081082	1745	Accepted	9432K	5219MS	Java
1563081082	3660	Accepted	4844K	2641MS	Java
1563081082	1062	Accepted	4568K	1985MS	Java
1563081082	2387	Accepted	8856K	2485MS	Java
1563081082	1860	Accepted	3896K	3797MS	Java
1563081082	2586	Accepted	4980K	766MS	Java
1563081082	1700	Accepted	4720K	266MS	Java
1563081082	1328	Accepted	4908K	813MS	Java
1563081082	1042	Accepted	4944K	4532MS	Java
1563081082	1185	Accepted	7440K	2391MS	Java
1563081082	1088	Accepted	4780K	2032MS	Java
1563081082	3579	Accepted	5696K	3407MS	Java
1563081082	3579	Accepted	5696K	3438MS	Java
1563081082	3269	Accepted	4964K	2750MS	Java
1563081082	1723	Accepted	4996K	2157MS	Java
1563081082	2506	Accepted	3600K	422MS	Java
1563081082	3233	Accepted	5028K	1782MS	Java
1563081082	1050	Accepted	4712K	469MS	Java

图 1.1 完成情况 1

User	Problem	Result	Memory	Time	Language
1563081082	2503	Accepted	19936K	3047MS	Java
1563081082	2366	Accepted	6508K	2844MS	Java
1563081082	3295	Accepted	4644K	407MS	Java
1563081082	1753	Accepted	1880K	3360MS	Java
1563081082	1005	Accepted	2272K	188MS	Java
1563081082	1000	Accepted	1876K	782MS	Java

图 1.2 完成情况 2

华中科技大学课程设计报告

2. POJ1753 解题报告

2.1 题目分析

题目给出了一个固定大小的 4*4 棋盘，由于对同一个点位进行偶数次翻转相当于不进行翻转，因此对每个点位仅存在两种情况：翻转/不翻转。共有 2 的 16 次方种情况，即 256 种情况，因此是可以进行暴力枚举的。

2.2 算法设计

```
import java.util.Scanner;
```

```
/**
```

```
 * @author MYH
```

```
 * @date 2022-11-16
```

```
 */
```

```
public class P1753 {
```

```
    public static int[][] direction = new int[][]{{1,0},{-1,0},{0,1},{0,-1},{0,0}};
```

```
    public static void main(String[] args){
```

```
        Scanner scan = new Scanner(System.in);
```

```
        int[][] board = new int[4][4];
```

```
//        read data: w->0 b->1
```

```
        for(int i = 0;i < 4;i++){
```

```
            String c = scan.next();
```

```
            for(int j = 0;j < 4;j++){
```

```
                if(c.charAt(j) == 'w'){
```

```
                    board[i][j] = 0;
```

```
                }else{
```

华中科技大学课程设计报告

```
        board[i][j] = 1;
    }
}
}

//      function test
//      test function flip
//      int[][] newBoard = flip(board,1);
//      for(int i = 0;i < 4;i++){
//          System.out.println(Arrays.toString(newBoard[i]));
//      }
//      test isFinished
//      System.out.println(isFinished(flip(board,0)));
int ans = dfs(board,0,0);
System.out.println(ans==17?"Impossible":ans);
}

private static int dfs(int[][] board,int cur,int count){
    if(isFinished(board)){
        return count;
    }
    if(cur >= 16){
        return 17;
    }
    // don't flip current node
    int t1 = dfs(board,cur+1,count);
    // flip current node
```

华中科技大学课程设计报告

```
        flip(board,cur);

        int t2 = dfs(board,cur+1,count+1);

        flip(board,cur);

//        return min flip count

        return Math.min(t1,t2);

    }

// flip cur node

private static void flip(int[][] board,int cur){

    // calculate current x and y

    int x = cur/4;

    int y = cur-x*4;

    for(int[] dir:direction){

        // make sure x and y in the range

        if(x+dir[0] >=0 && x+dir[0] < 4 && y+dir[1]>=0 && y+dir[1] < 4){

            board[x+dir[0]][y+dir[1]] ^= 1;

        }

    }

}

// judge if there is different number

private static boolean isFinished(int[][] board){

    int x = board[0][0];

    for(int i = 0;i < 4;i++){

        for(int j = 0;j < 4;j++){

            if(board[i][j] != x) {

                return false;

            }

        }

    }

    return true;

}
```

```
        }  
    }  
}  
return true;  
}  
}
```

- isFinished

判断函数，比较棋盘上的每个元素是否与第一个元素相同，从而判断是否达到目标

- flip

翻转函数，负责完成一次翻转操作，遍历五个方向(上下左右以及本身)，对每个方向进行翻转，为了简化代码，读入的数据采用 0 和 1 存储，由于 $0^1=1, 1^1=0$ ，所以可以利用按位异或的性质来对每一位进行翻转，从而不必写额外的 if 判断语句

- dfs

枚举过程使用递归的形式完成，分别需要考虑对一个点翻转和不翻转的情况，并取两种情况可能得到的最小值

遍历点时仅使用一个数据 cur 来完成，通过 $x=cur/4, y=cur-x*4$ ，来计算出 cur 表示的位置

结束条件有两种

- 。 达成目标，isFinished 判断棋盘已经翻转完成
- 。 已经遍历完了每一个点

2.3 性能分析

时间复杂度，dfs 最多遍历全部 256 种情况，每次都会调用 isFinished 函数判断棋盘是否完成翻转，判断需要对 board 数组的 16 个空间逐个进行判断，复杂度为 16。其次，dfs 中最差时需要有一半情况进行翻转，即翻转 128 次，每次翻转需要需要操作 5 个方向，因此最差情况时，dfs 中需要操作 $256*26+128*5=4736$ 次。

空间复杂度，用于记录方向的数组 direction 占固定大小为 5 的空间，棋盘占 $4*4$ 大小为 16 的空间，其余均为变量，因此空间复杂度总体为 $O(1)$ 。

2.4 运行测试

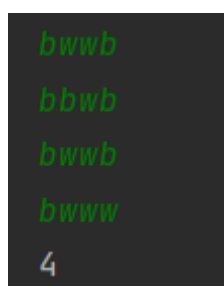


图 2.1 运行测试

华中科技大学课程设计报告

3. POJ3295 解题报告

3.1 题目分析

此题中仅给出了 5 个逻辑变量，每个变量有 True 和 False 两种情况，枚举共有 32 种情况，数量较小，因此可以对全部情况进行枚举。

3.2 算法设计

```
import java.util.*;

/**
 * @author MYH
 * @date 2022-11-16
 */
public class P3295 {
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        String s = scan.next();
        while(!s.equals("0")){
            if(help(s)){
                System.out.println("tautology");
            }else{
                System.out.println("not");
            }
            s = scan.next();
        }
    }
}
```

华中科技大学课程设计报告

```
// Enumerate each case

private static boolean help(String str){

    Map<Character,Character> map = new HashMap<Character,Character>();

    char[] b = new char[]{'0','1'};

    for(char p:b){

        map.put('p',p);

        for(char q:b){

            map.put('q',q);

            for(char r:b){

                map.put('r',r);

                for(char s:b){

                    map.put('s',s);

                    for(char t:b){

                        map.put('t',t);

                        if(!judge(str,map)){

                            return false;

                        }

                    }

                }

            }

        }

    }

    return true;

}
```

```
// judge if the str is true
```

华中科技大学课程设计报告

```
private static boolean judge(String str, Map<Character, Character> map) {  
    for(Map.Entry<Character, Character> entry : map.entrySet()){  
        str = str.replace(entry.getKey(), entry.getValue());  
    }  
    while(str.length() != 1){  
        if(str.indexOf("K") != -1) {  
            str = str.replace("K11", "1");  
            str = str.replace("K10", "0");  
            str = str.replace("K01", "0");  
            str = str.replace("K00", "0");  
        }  
        if(str.indexOf("A") != -1) {  
            str = str.replace("A11", "1");  
            str = str.replace("A10", "1");  
            str = str.replace("A01", "1");  
            str = str.replace("A00", "0");  
        }  
        if(str.indexOf("C") != -1) {  
            str = str.replace("C11", "1");  
            str = str.replace("C10", "0");  
            str = str.replace("C01", "1");  
            str = str.replace("C00", "1");  
        }  
        if(str.indexOf("E") != -1) {  
            str = str.replace("E11", "1");  
            str = str.replace("E10", "0");  
        }  
    }  
}
```

```
        str = str.replace("E01", "0");  
        str = str.replace("E00", "1");  
    }  
    if(str.indexOf("N") != -1){  
        str = str.replace("N1","0");  
        str = str.replace("N0","1");  
    }  
}  
return str.equals("1");  
}  
}
```

算法整体思路为，遍历五个逻辑变量为 0 或 1 的所有情况，对每种情况进行判断是否为真，当出现假时停止，说明该式不是永真式，直到遍历完全部情况，说明该式为永真式。

判断逻辑表达式的常规思路是使用栈来进行内容读取，但是此题的逻辑表达式对于每个逻辑操作符，没有很好的可以用来判断结束的标志，例如括号的右括号，因此这里枚举题干表中给出的所有情况，使用 replace 方法将式子进行不断化简，直到最简，即 ‘0’ 或 ‘1’ 为止。

3.3 性能分析

时间复杂度 $O(nm^2)$

其中，n 为输入逻辑表达式的数量，m 为逻辑表达式的长度

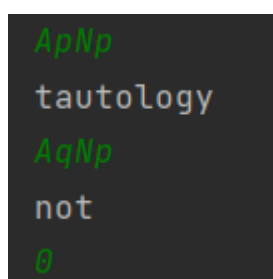
华中科技大学课程设计报告

对于每一组逻辑表达式，遍历逻辑变量的所有情况共需 2 的 5 次方，即 32 次

对于每一次判断，需要将长为 m 的字符串化简至长度为 1 为止，每次 replace 可以将三个字符化为一个字符，因此 $1+2t=m$ ，replace 的时间复杂度为 $O(m)$ ，所以总的时间复杂度为 $O(m^2)$

空间复杂度，计算过程仅开辟常数级空间，因此空间复杂度为 $O(1)$

3.4 运行测试



```
ApNp
tautology
AqNp
not
0
```

图 3.1 运行测试

4. POJ1201 解题报告

4.1 题目分析

题目意为有 n 个区间 $[a_i, b_i]$ ，每个区间至少需要选 c_i 个数. 求出新的集合 Z 在满足所有约束条件下，至少包含多少个数。用 S_i 表示 $0 - i$ 中被选出的数的个数。约束条件：

华中科技大学课程设计报告

1. $S_i \geq S_{i-1}$, 从 1 到 i 中选的数必定不会比从 1 到 $i-1$ 中选的数少。

2. $S_i - S_{i-1} \leq 1 \Rightarrow S_{i-1} \geq S_i - 1$, 相邻之间选的数的个数差不会超过 1 个数。

3. $S_b - S_{a-1} \geq c \Rightarrow S_b \geq S_{a-1} + c$, $[a, b]$ 中至少选 c 个数。

4. 从源点出发, 一点可以走到所有边

注意到 $S_b \geq S_{a-1} + c$ 这个条件, 与单源最短路中的三角形不等式 $\text{dist}[y] \leq \text{dist}[x] + w$ 非常相似, 因此可以尝试使用差分约束建图来完成此题

为满足条件 1, 从 $i-1$ 向 i 连接一条 权值为 0 的边, $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \cdots 50001$ 因此源点 0 是可以到达所有边的。

题目中 a_i, b_i 从 0 开始的, 因此用 0 来作为源点。

由于求的是最小值, 因此求一遍最长路就行了, 结果为 $\text{dist}[50001]$ 。

4.2 算法设计

```
import java.util.*;
```

```
/**
```

华中科技大学课程设计报告

* @author MYH

* @date 2022-12-02

*/

```
public class P1201 {
```

```
    // the distance from zero to index
```

```
    static int[] dis = new int[50009];
```

```
    // the end point of the idxth edge
```

```
    static int[] e = new int[150009];
```

```
    // the weight of edge
```

```
    static int[] w = new int[150009];
```

```
    // find the next idx
```

```
    static int[] ne = new int[150009];
```

```
    // Record the index of the last edge starting with a
```

```
    static int[] h = new int[150009];
```

```
    static int idx;
```

```
    static void add(int a,int b,int c){
```

```
        e[idx] = b;
```

```
        w[idx] = c;
```

```
        ne[idx] = h[a];
```

```
        h[a] = idx++;
```

```
    }
```

```
    static void spfa(){
```

```
        Arrays.fill(dis,-0x3f);
```


华中科技大学课程设计报告

```
dis[0] = 0;

Queue<Integer> queue = new LinkedList<Integer>();

Set<Integer> set = new HashSet<Integer>();

queue.offer(0);

set.add(0);

while(queue.size()!=0){

    int cur = queue.poll();

    set.remove(cur);

    for(int i = h[cur];i>=0;i=ne[i]){

        int j = e[i];

        if(dis[j] < dis[cur] + w[i]){

            dis[j] = dis[cur] + w[i];

            if(!set.contains(j)){

                set.add(j);

                queue.offer(j);

            }

        }

    }

}
```

```
public static void main(String[] args){

    Scanner scan = new Scanner(System.in);

    int n = scan.nextInt();

    idx = 0;

    Arrays.fill(h,-1);
```

```
for(int i = 0;i < 50001;i++){
    add(i,i+1,0);
    add(i+1,i,-1);
}
for(int i = 0;i < n;i++){
    add(scan.nextInt(),scan.nextInt()+1,scan.nextInt());
}
spfa();
System.out.println(dis[50001]);
}
}
```

根据对题目的分析，使用差分约束来解决这个问题，首先需要建立有向带权图，用数组 e 存取第 i 条边的终点，数组 w 存取第 i 条边的权值（即 c_i ），数组 ne 存取以 a 为起点的上一条边的索引 i ，数组 h 存取以 a 为起点的最后一条边的索引 i ，通过这种存取方式，可以快速遍历所有以 a 为起点的边，加快运行速度。

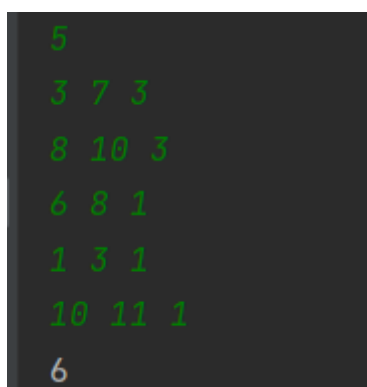
spfa 是求单源最短路径的一种算法，它是 Bellman-ford 的队列优化，建立一个队列，初始时队列里只有起始点，在建立一个表格记录起始点到所有点的最短路径（该表格的初始值要赋为极大值，该点到他本身的路径赋为 0）。然后执行松弛操作，用队列里有的点去刷新起始点到所有点的最短路，如果刷新成功且被刷新点不在队列中则把该点加入到队列最后。重复执行直到队列为空。

4.3 性能分析

时间复杂度，spfa 算法的最差时间复杂度为 $O(VE)$ ，其中 V 为定值 50000， E 为 $100000+n$ ，初始化图的时间复杂度为常数 $O(1)$ ，读取数据的时间复杂度为 $O(n)$ 。

空间复杂度，由题目知， n 的范围为 $[1, 50000]$ ，对于每组数据，需要 4 个空间来存储建图相关的数据，除此之外初始化时使用了 400000 大小的空间来对图进行初始化，同时因为 a, b 的范围为 $[1, 50000]$ 需要大小为 50000 的数组 dis 来存储每个点与源点的距离，同时还有 `queue` 以及 `set` 所占空间，但其空间重复利用程度高，所占空间相对数组较小，综合考虑空间复杂度为 $O(n)$ 。

4.4 运行测试



```
5
3 7 3
8 10 3
6 8 1
1 3 1
10 11 1
6
```

图 4.1 运行测试

5. POJ3040 解题报告

5.1 题目分析

给定每种钱的面额和数量，要求用这些钱尽可能多地组合出达到C元的组合，因此此题应该为将钱进行分组的一个策略问题。

由于其中给出了面额之间存在整除关系，确保了使用贪心算法的正确性，因此可以确定此题应该使用贪心算法。

5.2 算法设计

```
import java.util.*;

/**
 * @author MYH
 * @date 2022-12-01
 */
public class P3040 {

    // the class to record coin
    static class Money{
        // value
        int v;
        // num
        int b;

        Money(int v,int b){
            this.v = v;
            this.b = b;
        }
    }
}
```

```
    }  
}  
  
public static void main(String[] args){  
    Scanner scan = new Scanner(System.in);  
    int n = scan.nextInt();  
    int c = scan.nextInt();  
    // record all money  
    Money[] moneys = new Money[25];  
    // record how many ith money be used currently  
    int[] used = new int[25];  
    int ans = 0;  
    // the num of type of money  
    int idx = 0;  
    for(int i = 0; i < n; i++){  
        int v = scan.nextInt();  
        int b = scan.nextInt();  
        // if just one coin can pay, no need to record  
        if(v >= c){  
            ans += b;  
        }else{  
            Money money = new Money(v,b);  
            moneys[idx++] = money;  
        }  
    }  
}  
  
// sort moneys from small to large
```

华中科技大学课程设计报告

```
Arrays.sort(moneys,0,idx, new Comparator<Money>() {  
    public int compare(Money o1, Money o2) {  
        return o1.v-o2.v;  
    }  
});  
while(true){  
    // reset used  
    Arrays.fill(used,0);  
    // rest to pay  
    int rest = c;  
    // from max to min  
    for(int i = idx-1;i >=0;i--){  
        int tmp = Math.min(rest/moneys[i].v,moneys[i].b);  
        rest -= tmp*moneys[i].v;  
        used[i] = tmp;  
    }  
    // need to pay more  
    if(rest != 0){  
        // from min to max to avoid waste money  
        for(int i = 0;i < idx;i++){  
            if(moneys[i].b != 0 && moneys[i].v >= rest){  
                used[i]++;  
                rest = 0;  
                break;  
            }  
        }  
    }  
}
```

```
    }
    // no enough money pay c
    if(rest != 0){
        break;
    }
    // record current strategy can pay how many weeks
    int minx = Integer.MAX_VALUE;
    for(int i = 0; i < idx; i++){
        if(used[i] != 0){
            minx = Math.min(minx, moneys[i].b/used[i]);
        }
    }
    ans += minx;
    for(int i = 0; i < idx; i++){
        // sub the number of used money
        moneys[i].b -= minx*used[i];
    }
}
// print ans
System.out.println(ans);
}
}
```

每一次发工资，都要考虑当前所能实现的最优策略。

华中科技大学课程设计报告

对 money 数组按照面值大小从小到大排序,然后将数值大于等于 C 的先提前计算。

为了能最大程度实现每次发的工资都尽可能贴近 C,我们需要先从大到小进行分配,分配过程中,费用不能超过 C;然后,如果费用不等于 C,也就是仍然需要面值小的进行补偿,就需要从小到大进行分析,这时的费用可以超过 C,因为是从大到小,我们可以将小面值用到极致,也就是能用的都用了。

贪心过程需要 used 数组记录每一个面值所使用的数目,这样就可以找到最优组合可以实现多少次,也就是 $\text{moneys}[i].b/\text{used}[i]$ 中的最小值。

当搭配完毕后,若出现费用仍然大于零的情况,就说明当前的钱不够分工资了,退出循环。

5.3 性能分析

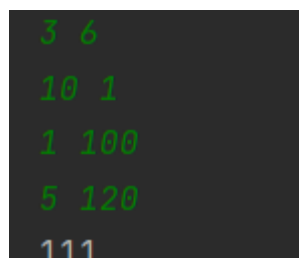
时间复杂度,在读取完输入数据后,由于输入的数据为随机输入,因此需要对其按照面额大小进行排序, `Arrays.sort()` 的时间复杂度为 $O(n\log n)$ 。

空间复杂度,首先需要两个大小为 20 ($n \leq 20$) 的数组 `moneys` 记录每次输入的钱的信息(面额以及数量),以及 `used` 记录当前消耗每

华中科技大学课程设计报告

种钱的数量，其余均为常数级变量空间的申请，因此空间复杂度为 $O(n)$ 。

5.4 运行测试



```
3 6
10 1
1 100
5 120
111
```

图 5.1 运行测试

6. 总结

6.1 实验总结

本次实验我总共从题单上选择了 25 个题目，并且全部都成功完成，写题目的过程中虽然会遇到一些困难，但是克服困难后也学到了许多有用的技术和技巧。

递归过程出现多个结束条件判断时，需要缜密选择判断的先后顺序，例如在 POJ1753 Flip Game 中，递归的判断条件有 1. 遍历完所有点 2. 达成目标，由于判断达成目标语句判断的是上一次递归结束后的结果，所以应该放在前面，这样才能得到正确的输出结果。

华中科技大学课程设计报告

Java 中将数组作为参数传入时，传入的是地址，在其中对数据进行操作会改变原数组，如果想要不影响原数组，可以使用 `Arrays.copyOf()` 来获得一个与传入数组值相同的新数组，在新数组上进行修改。

OJ 对输出的结果要求相对比较严格，一个空格的不正确抑或是换行符错误都会导致不通过，对于长字符串的输出，应该直接复制题目中的输出并进行修改，而不是手动去输入。

6.2 心得体会和建议

通过本次实验，我应用了自己在数据结构和算法设计与分析中的所学所得，是一次很好的对所学知识的实践应用，将仅仅作为理论知识掌握的算法成功进行了实现，学习到了很多程序设计技巧。其中也有难度很高的题目，并不是我一上手就可以解决的，但通过从网上查阅相关资料，学习相关算法及其优化后，最终还是能成功解决，并且获得更大的成就感，今后我也会持续保持自己在算法设计此方面的学习历练。

本次实验的题目选取的是恰到好处的，既有比较容易的，也有需要一定学习的，同时留给了同学们选择的余地，但 POJ 平台的使用确实有些一言难尽，除了简陋的前端，对 Java 的版本支持度有些低下，如果非要提建议的话，可能就是希望使用更加强健的 OJ 平台。