1 实验八 分支限界法——装载问题

1-1 一、实验目的

1、通过分支限界法的示例程序理解回溯法的基本思想

2、运用分支限界法解决实际问题进一步加深对分支限界法的理解和运用

1-2 二、实验内容

（一）装载问题

I 1. 问题描述

用回溯法编写一个递归程序解决如下装载问题：

有 n 个集装箱要装上 2 艘载重分别为 c1 和 c2 的轮船，其中集装箱 i 的重量为 wi（1≤ i ≤ n），且 ∑ ◈◈ ≤ ◈1 + ◈2 ◈ ◈=1 。

问是否有一个合理 的装载方案可以将这 n 个集装箱装上这 2 艘轮船？如果有，请给出装载方案。

举例：当`n=3`，`c1 = c2 = 50`，且`w = [10,40,40]`时，可以将集装箱 1 和 2 装到第一艘轮船上， 集装箱 3 装到第二艘轮船上；如果`w=[20,40,40]`时，无法将这 3 个集装箱都装上轮船。

> 注意：用队列式分支限界法和优先队列式分支限界法两种方法分别实现。

输入格式

`输入的第一行整数c，第一艘轮船载重量。 第二行是集装箱个数 第三行为每个集装箱的重量`

输出格式

`输出2行，第一行包含一个整数，表示最大载重量 接下来依次输出装入的集装箱序号`

样例输入

70 4 20 10 26 15

样例输出

最优装载重量为：61 最优装载顺序为（1 表示装入，0 表示未装入）：1 0 1 1

II 2. 要求

（1） 写出问题的分析过程

（2） 写出程序代码

（3） 贴出程序结果

1-3 三、实验总结（写出本次实验的收获，遇到的问题等）

I 代码

```java
import java.util.Scanner;

/*
 * 若尘
 */

/**
 * 装载问题
 *
 * @author ruochen
 * @version 1.0
 */
public class Load {
  /** 集装箱数 */
  static int n;
  /** 集装箱重量数组 */
  static int[] w;
  /** 第一艘轮船的载重量 */
  static int c1;
  /** 第二艘轮船的载重量 */
  static int c2;
  /** 当前载重量 */
  static int cw;
  /** 当前最优载重量 */
  static int bestw;
  /** 剩余集装箱重量 */
  static int r;
  /** 当前解 */
  static int[] x;
  /** 当前最优解 */
  static int[] bestx;

  public static void main(String[] args) {
    // 下标从1开始，所以第一个元素为0
    System.out.println("请输入第一艘轮船的载重量");
    Scanner weight1 = new Scanner(System.in);
    int cc1 = weight1.nextInt();

    System.out.println("请输入集装箱的个数: ");
    Scanner number = new Scanner(System.in);
    // w是集装箱的个数
    int w = number.nextInt() + 1;

    // 创建集装箱数组
    System.out.println("请输入集装箱重量...");
    Scanner weight = new Scanner(System.in);
    int[] ww = new int[w];
    ww[0] = 0;
    for (int i = 1; i < w; i++) {
      ww[i] = weight.nextInt();
    }

    int cc2 = cc1;
    System.out.println(maxLoading(ww, cc1, cc2));
    outPut();
    weight1.close();
    weight.close();
    number.close();
  }
```

```java
/** 返回不超过c 的最大子集和 */
public static int maxLoading(int[] ww, int cc1, int cc2) {
  n = ww.length - 1;
  w = ww;
  c1 = cc1;
  c2 = cc2;
  cw = 0;
  bestw = 0;
  x = new int[n + 1];
  bestx = new int[n + 1];
  r = 0;
  // r 初始值为全部集装箱总重
  for (int i = 1; i <= n; i++) {
    r += w[i];
  }
  // 计算最优载重量
  backTrack(1);
  return bestw;
}

/** 回溯算法 */
public static void backTrack(int i) {
  // 搜索第 i 层节点
  if (i > n) {
    // 到达叶节点
    if (cw > bestw) {
      for (int j = 1; j <= n; j++) {
        bestx[j] = x[j];
      }
      bestw = cw;
    }
    return;
  }
  // 搜索子树
  r -= w[i];
  if (cw + w[i] <= c1) {
    // 重量不超过 c
    // 搜索左子树
    x[i] = 1;
    cw += w[i];
    backTrack(i + 1);
    // 还原
    x[i] = 1;
    cw -= w[i];
  }
  // 只在右子树进行上界函数判断是因为其对左子树无影响
  // 左子树是选择放，上界函数 = cw(当前重量) + r(剩余重量)
  if (cw + r > bestw) {
    x[i] = 0;
    // 搜素右子树
    backTrack(i + 1);
  }
  r += w[i];
}

static void outPut() {
  int weight = 0;
  for (int i = 1; i <= n; i++) {
    if (bestx[i] == 0) {
      // 第一艘轮船装完后的剩余重量
```

```
        weight += w[i];
      }
    }
    if (weight > c2) {
      System.out.println("不能装入 ");
    } else {
      System.out.print("轮船一装入的货物为: ");
      for (int i = 1; i <= n; i++) {
        if (bestx[i] == 1) {
          System.out.print(i + " ");
        }
      }
      System.out.println();
      // Output 0 or 1
      for (int i = 1; i <= n; i++) {
        if (bestx[i] == 1) {
          System.out.print("1" + " ");
        } else {
          System.out.print("0" + " ");
        }
      }
      System.out.println();
      // System.out.print("轮船二装入的货物为:  ");
      // for (int i = 1; i <= n; i++) {
      // if (bestx[i] != 1) {
      // System.out.print(i + " ");
      // }
      // }
      // System.out.println();
    }
  }
}
```