算法分析第四次作业

算法实现题

4-1 题答案：

答：先按照开始时间递增排序，若开始时间相同则按照结束时间递增排序。每次都优先安排序号靠前的活动，除非不相容，一遍扫完之后再扫第二遍，同时开第二个会场，直至所有活动都被安排。

伪代码：

```
struct Activity {
    int start;    // 活动开始时间
    int end;      // 活动结束时间
    bool if_arrange;    // 活动是否已安排
}
// 比较函数
function cmp(Activity a, Activity b) {
int main(){
    int k;
    scanf k;
    Activity a[10001];
    For(i 从 0 到 k – 1) {
        输入 a[i].start 和 a[i].end;
        a[i].if_arrange = false;
    }
    使用 cmp 函数对数组 a 进行排序;
    int ok_num = 0;    // 已成功安排的活动数量
    int now_end = 0; // 当前安排活动的结束时间
    int ans = 0;        // 需要的会场数量
    while(ok_num<k) {
        now_end = 0;
        ans = ans + 1;
        for( i 从 0 到 k - 1 ){
            if( a[i].if_arrange 为 false ){
                if(a[i].start 大于等于 now_end) {
                    a[i].if_arrange = true;
                    ok_num = ok_num + 1;
                    now_end = a[i].end;
                }
```

```
                }
            }
        }
        printf ans;
        return 0;
}
```

4-2 题答案：

答：由题目已知得合并长度为 m 和 n 需要比较次数为 m+n-1,要想得到最优合并顺序，根据贪心算法的思想，局部最优即全局最优，先考虑长度最短的两个数进行合并，合并后的数字加入数组中然后再次挑选长度最短的两个数合并，每次合并都让记录次数的值进行加法就可以，一直到将所有数据合并完为止。

首先用一个数组存储要合并的序列，如果要求最优合并，则升序排列，每次合并后的数字替代合并前的第二个数字，然后对后面的再次排序，重复前面的步骤。

如果是最差情况，则降序排列，从最大的开始，每次挑选最大的两个数字合并，合并后的数字加入原数组，再排序然后合并。

4-4 题答案：

答：子结构性质位较大的数离较小的数越近越优（相差较小的两个数越近越优），最优策略为：尽可能地将较小检索概率的文件放在较大检索概率的文件的最近位置。排序不是严格单调排列，而是将相应值的分布从中间往两边依次减少排列。

4-6 题答案：

答：需要服务时间短的顾客先接受服务,总的等待时间是所有顾客等待时间之和并且每位顾客的等待时间包括自己所需要的时间。

1、定义一个 time 数组记录每个服务点目前的时间，每服务一个客户 time[服务点号]加上该客户的服务时间。

2、随后对每个服务点的目前时间进行比较，数值最小的服务点即为最快能接受服务的服务点，该服务点进行服务，

该客户的等待时间 waited_time 即为该服务点的目前时间 time[服务点号]，并记录好该客户接受服务的服务点号 served_point

3、将每个用户的等待时间相加，再除以客户的总数，即得到平均服务时间。

伪代码：

```
const MAX_CUSTOMS = 100;
struct Customer {
    int id;
    float serviceTime;
}
// 服务处
struct ServicePoint {
    float remainingTime;
    int customerOrder[MAX_CUSTOMS];
    int customerCount;
}
// 比较函数
function compare(Customer a, Customer b) ;
function main() {
    int n, s;
    scanf n, s;
    Customer customers[MAX_CUSTOMS];
    ServicePoint points[MAX_CUSTOMS];
    for (i = 0; i < s; i++) {
        points[i].remainingTime = 0;
        points[i].customerCount = 0;
    }
    for (i = 0; i < n; i++) {
        scanf customers[i].id, customers[i].serviceTime;
    }
```

```
        Sort(customers, n, compare);
        for (i = 0; i < n; i++) {
            int minIndex = 0;
            for (j = 1; j < s; j++) {
                if (points[j].remainingTime < points[minIndex].remainingTime) {
                    minIndex = j;
                }
            }
            points[minIndex].customerOrder[points[minIndex].customerCount] =
customers[i].id;
            points[minIndex].customerCount++;
            points[minIndex].remainingTime += customers[i].serviceTime;
        }
        // 计算总等待时间
        float totalWait = 0;
        for (i = 0; i < s; i++) {
            float curTime = 0;
            for (j = 0; j < points[i].customerCount; j++) {
                for (k = 0; k < n; k++) {
                    if (customers[k].id == points[i].customerOrder[j]) {
                        totalWait += curTime + customers[k].serviceTime;
                        curTime += customers[k].serviceTime;
                        break;
                    }
                }
            }
        }
        printf totalWait / n;
        printf " ", points[i].customerOrder[j];
    }
}
```

4-8 题答案：

答：双亲表示法建立一棵树，从叶节点开始遍历，如果一个节点的每个子树都已经被遍历到，那么这个节点成为叶节点入队列，直到队列为空。树从叶节点开始遍历，将出度为 0 的节点入队列，一个个出队列，出队列时如果到父节点路长大于 d，切掉父节点，父节点的父节点出度 - 1，如果到父节点的路长不大于 d，并且到父节点的路长

大于父节点到叶节点的路长，更新父节点到叶节点的路长，并且父节点出度 - 1。如果一个父节点的儿子节点都出队列，那么这时候得到的父节点到叶节点的路长就是到所有儿子节点的最长路长，将这个值保存下来，将父节点入队列。

4-9 题答案：

答：对于到达每一个加油站时的处理：如果目前的油够到达下一站，就不再加油了，把油留到后面加，如果目前的油不够到达。

```
    int n, k, temp;//n 满油行驶距离，k 加油站数量
    scanf n, k;
    int rest_oil = n; //  初始化剩余油量为加满油的量
    scanf temp;
    int ans = 0;//加油次数
    for (i = 0; i < k; i++) {
        int now_input;
        scanf now_input;//到下一个距离
        if (now_input > n) {
            printf "No Solution";
            return 0;
        }
        if (rest_oil - now_input >= 0) {
            rest_oil = rest_oil - now_input;
        }
        else {
            rest_oil = n;
            rest_oil = rest_oil - now_input;
            ans = ans + 1;
        }
    }
    printf ans;
    return 0;
}
```

4-11 题答案：

答：

4-15 题答案：

答：首先将仍无依照其截止时间非递减排序,设对于任务 1，2，……，i。截止时间为 d 的最小误时惩罚为 p[i][d]，则 p[i][d]符合最优子结构性质，且有递推式 p[i][d] = min{ p[i-1][d] + w[i], p[i-1][min{d,d[i]} - t[i]] }。

边界条件有：考虑 p[1][d]

1.对于 t[1] <= d 时，截止时间大于完成任务的时间，而且只有一个任务，该任务是可以完成的，所以没有惩罚。

2.对于 t[1] > d 时，由于第一个工作无法完成，必然导致惩罚，罚时为 w[1]。

```
int MAXINT = 99999;
struct Task {
    int t; // 任务所需时间
    int d; // 任务截止时间
    int w; // 任务惩罚权重
}
function cmp(Task a, Task b);
int n;
scanf n;
Task task[1000];
for (i = 0; i < n; i++) {
    scanf task[i].t, task[i].d, task[i].w;
}
Sort task array from task[0] to task[n - 1] using cmp function;
int maxd = task[n - 1].d;
int dp[n][maxd + 1];
// 初始化动态规划数组，将所有值设为极大值
for (i = 0; i < n; i++) {
    for (j = 0; j <= maxd; j++) {
        dp[i][j] = MAXINT;
    }
}
for (j = 0; j <= maxd; j++) {
    if (j < task[0].d) {
        dp[0][j] = task[0].w;
    } else {
```

```
                dp[0][j] = 0;
            }
        }

        for (i = 1; i < n; i++) {
            for (j = 0; j <= maxd; j++) {
                // 不选择当前任务的惩罚
                int drop = dp[i - 1][j] + task[i].w;
                dp[i][j] = drop;
                // 如果可以在截止时间内完成
                if (min(j, task[i].d) - task[i].t >= 0) {
                    // 选择当前任务的惩罚
                    int fetch = dp[i - 1][min(j, task[i].d) - task[i].t];
                    dp[i][j] = min(drop, fetch);
                }
            }
        }
        // 输出最终的最小惩罚
        printf dp[n - 1][maxd];
        return 0;
}
```