

算法实现题

4-1 会场安排问题

算法思路：

用贪心策略，每次考虑可能的最早结束的时间，再找这段会场结束后开始的最早结束事件，直到没有可以安排的为止。

```
struct Activity {
    int start;
    int end;
};

// 比较函数，用于按活动结束时间排序
bool compareEndTime(const Activity& a, const Activity& b) {
    return a.end < b.end;
}

// 贪心算法实现
int greedyActivitySelector(const std::vector<Activity>& activities) {
    if (activities.empty()) {
        return 0;
    }
    // 按结束时间排序活动
    std::vector<Activity> sortedActivities = activities;
    std::sort(sortedActivities.begin(), sortedActivities.end(),
compareEndTime);

    int numActivities = 1; // 至少有一个活动
    int lastEndTime = sortedActivities[0].end;
    for (size_t i = 1; i < sortedActivities.size(); ++i) {
        if (sortedActivities[i].start >= lastEndTime) {
            numActivities++;
            lastEndTime = sortedActivities[i].end;
        }
    }
    return numActivities;
}
```

4-2 最优合并问题

思路：

minComparisons 函数：

采用贪心策略，每次选择长度最小的两个序列进行合并。

使用 `std::sort` 对序列长度数组排序，然后取前两个最小的长度进行合并，更新总比较次数并更新数组，直到只剩下一个序列。

maxComparisons 函数:

采用贪心策略, 每次选择长度最大的两个序列进行合并。

使用 `std::sort` 并传入 `std::greater<int>()` 对序列长度数组从大到小排序, 然后取前两个最大的长度进行合并, 更新总比较次数并更新数组, 直到只剩下一个序列。

```
// 计算最少比较次数
int minComparisons(std::vector<int> lengths) {
    int sum = 0;
    while (lengths.size() > 1) {
        // 找到最小的两个长度
        std::sort(lengths.begin(), lengths.end());
        int a = lengths[0];
        int b = lengths[1];
        sum += a + b - 1;
        lengths.erase(lengths.begin());
        lengths[0] = a + b;
    }
    return sum;
}

// 计算最多比较次数
int maxComparisons(std::vector<int> lengths) {
    int sum = 0;
    while (lengths.size() > 1) {
        // 找到最大的两个长度
        std::sort(lengths.begin(), lengths.end(), std::greater<int>());
        int a = lengths[0];
        int b = lengths[1];
        sum += a + b - 1;
        lengths.erase(lengths.begin());
        lengths[0] = a + b;
    }
    return sum;
}
```

4-4 磁盘文件最优存储问题

思路: 根据题目, 输入的是表示文件检索概率的整数数组, 需要先将其归一化, 得到真正的检索概率;

贪心策略, 将概率大的文件优先放置在能使期望检索时间更优的位置;

再确定存储位置, 将概率最大的文件 `f_1` 放在中心磁道, 然后概率次大的 `f_2` 和 `f_3` 分居 `f_1` 两侧, `f_4` 在 `f_2` 左侧, `f_5` 在 `f_3` 右侧, 依次类推确定所有文件的存储位置。

```
double greedy(std::vector<int> p) {
    int n = p.size();
```

```

// 计算概率总和
int sum = std::accumulate(p.begin(), p.end(), 0);
// 归一化概率
std::vector<double> probs(n);
for (int i = 0; i < n; ++i) {
    probs[i] = static_cast<double>(p[i]) / sum;
}
// 按概率从大到小排序
std::vector<int> indices(n);
for (int i = 0; i < n; ++i) {
    indices[i] = i;
}
std::sort(indices.begin(), indices.end(), [&probs](int a, int b) {
    return probs[a] > probs[b];
});

// 确定文件存储位置
std::vector<int> positions(n);
int mid = n / 2;
positions[mid] = indices[0];
int left = mid - 1, right = mid + 1;
for (int i = 1; i < n; ++i) {
    if (i % 2 == 1) {
        positions[left--] = indices[i];
    } else {
        positions[right++] = indices[i];
    }
}

// 计算期望检索时间
double expectedTime = 0.0;
for (int i = 0; i < n; ++i) {
    for (int j = i; j < n; ++j) {
        double probProduct = probs[positions[i]] * probs[positions[j]];
        int distance = std::abs(i - j);
        expectedTime += probProduct * distance;
    }
}
return expectedTime;
}

```

4-6 最优服务次序问题

思路：n 个顾客同时等待，计时的起点相同，每个人的等待时间就是他接受服务前所有被服

务的人 t_i 之和。 t_i 最大的人放在最后一个被服务，其他人等待的时间之和最少。去掉最后一个，则剩下的人里 t_i 最大的应该放在最后。以此类推，故次序按服务时间升序排列就行。

方法：直接使用快排 `sort()`

算法分析：时间复杂度为 $O(n \log n)$

```
// 计算最优服务次序下的最小平均等待时间
double greedy(std::vector<int> x) {
    int n = x.size();
    // 对服务时间从小到大排序
    std::sort(x.begin(), x.end());
    // 计算每个顾客的累计等待时间
    for (int i = 1; i < n; ++i) {
        x[i] += x[i - 1];
    }
    double t = 0;
    // 计算总等待时间
    for (int i = 0; i < n; ++i) {
        t += x[i];
    }
    // 计算平均等待时间
    t /= n;
    return t;
}
```

4-8 d 森林问题

思路：根据贪心策略，为了使删除顶点集 S 后得到的 d 森林中树的从根到叶路长不超过 d ，我们优先处理距离根较远（叶节点方向）的顶点。

`Solve()` 函数使用 `isRemoved` 数组标记顶点是否已被删除。从每个叶节点开始，沿着父节点路径向上回溯计算路径长度 `pathLen`，当路径长度超过 d 时，若当前顶点未被删除，则将其标记为删除并增加计数 `count`。最后根据计算结果判断是否能得到 d 森林并返回相应结果。

```
// 存储图的邻接表结构，每个顶点对应一个邻接顶点和边权的列表
unordered_map<int, vector<pair<int, int>>> graph;
// 存储每个顶点的父节点
vector<int> parent;
// 存储每个顶点到其父节点的边权
vector<int> parlen;
// 存储叶节点编号
vector<int> leaf;
// 顶点数
int n;
// 给定的距离 d
int d;

// 读取初始数据
void readin() {
```

```

ifstream fin("input.txt");
fin >> n;
parent.assign(n + 1, 0);
parlen.assign(n + 1, 0);
for (int i = 1; i <= n; ++i) {
    int deg;
    fin >> deg;
    if (deg == 0) {
        leaf.push_back(i);
    }
    for (int j = 0; j < deg; ++j) {
        int p, len;
        fin >> p >> len;
        graph[i].push_back({p, len});
        parent[p] = i;
        parlen[p] = len;
    }
}
fin >> d;
fin.close();
}

```

// 计算最小顶点集 S 的大小

```

int solve() {
    vector<bool> isRemoved(n + 1, false);
    int count = 0;
    for (int leafNode : leaf) {
        int cur = leafNode;
        int pathLen = 0;
        while (cur != 1) {
            pathLen += parlen[cur];
            if (pathLen > d) {
                if (!isRemoved[cur]) {
                    isRemoved[cur] = true;
                    count++;
                }
                break;
            }
            cur = parent[cur];
        }
    }
    if (count == 0 && pathLen <= d) {
        return 0;
    }
}

```

```
    return count;
}
```

4-9 虚拟汽车加油

思路：由于加油站是由近到远的顺序排列的，所以每一步都优先选离当前最远的能达到的加油站。

```
#include <iostream>
using namespace std;
const int K = 10050;
int gst[K]; // gasoline station

int greedy(int n, int k, int gst[]) {
    int cnt = 0, tmp = 0;
    for (int i = 0; i <= k; i++) {
        if (tmp + gst[i] > n) {
            cnt++;
            tmp = 0;
        }
        tmp += gst[i];
    }

    return cnt;
}

int main() {
    int n;
    cin >> n;
    int k;
    cin >> k;
    for (int i = 0; i <= k; i++) {
        cin >> gst[i];
    }
    cout << greedy(n, k, gst) << endl;
    return 0;
}
```

4-11 去除数的问题

思路：最近下降点优先，即从最高位开始遍历，每次删去比低位数字大的数，直到删掉 k 个数。

```
#include <iostream>
#include <string.h>
using namespace std;
```

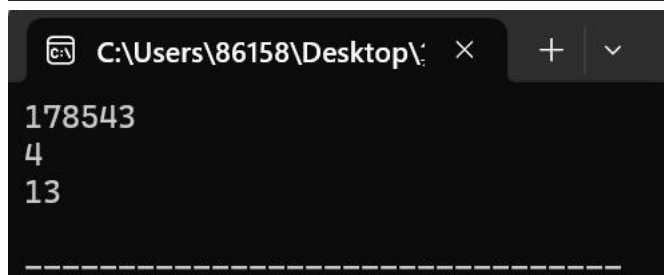
```

void deletex(string &a, int k) {
    int len = a.size();
    if (k >= len) {
        a.erase();
        return;
    }
    while (k > 0) {
        int i = 0;
        for (; (i < a.size() - 1) && (a[i] <= a[i + 1]); i++);
        a.erase(i, 1);
        k--;
    }
    while (a.size() > 1 && a[0] == '0')
        a.erase(0, 1);
}

int main() {
    string a;
    int k;
    cin >> a >> k;
    deletex(a, k);
    cout << a << endl;

    return 0;
}

```



```

C:\Users\86158\Desktop\  ×  +  ▾
178543
4
13
-----

```

4-15 最优分解问题：

思路：

$a+b$ 为常量时， $|a-b|$ 就越小， ab 越大。

限制条件是子数不相等，所以要将 n 分成尽可能多的数（除了 1）。

贪心策略：将 n 分成从 2 开始的连续自然数的和。

如果最后剩下一个数是前面使用过的，就将该数均匀分给前面各项（从后往前）。

代码：

```

//4-15
#include <iostream>

```

```

using namespace std;
int a[10050]; //存放分解的数

void maxdivision(int n) {
    int k = 1;
    //不合法的输入
    if (n < 3) {
        a[1] = 0;
        return;
    }

    //n 比较小时只有一种分法，特殊讨论：
    if (n < 5) {
        a[k] = 1;
        a[++k] = n - 1;
    }

    a[1] = 2;
    n -= 2;
    while (n > a[k]) {
        k++;
        a[k] = a[k - 1] + 1;
        n -= a[k];
    }
    if (n == a[k]) {
        a[k]++;
        n--;
    } //考虑 a[k]>k-1 (个数) 的情况

    for (int i = 0; i < n; i++) {
        a[k - i]++;
    }
    return;
}

int main() {
    int n;
    cin >> n;
    maxdivision(n);
    int ans = 1, idx = 1;
    while (a[idx]) {
        ans *= a[idx++];
    }
}

```



```
cout << ans << endl;  
}
```



A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\86158\Desktop\" followed by a close button (X), a plus sign (+), and a dropdown arrow (v). The command prompt displays the output of a program: the number "10" on the first line and "30" on the second line. Below the output, there is a dashed line representing the command prompt's input line.

```
10  
30  
-----
```