# Algorithm HW4

1. 设有 $n$ 个顾客，服务时间分别为 $t_1, t_2, \ldots, t_n$，一次只能服务一个人，服务不可中断。若服务顺序为某个排列 $\pi$，第 $\pi(k)$ 个顾客的等待时间为

   $$W_{\pi(k)} = \sum_{j=1}^{k-1} t_{\pi(j)}.$$

   因此总等待时间为

   $$W = \sum_{k=1}^{n} \sum_{j=1}^{k-1} t_{\pi(j)}.$$

   按服务时间从小到大排序，即**最短服务时间优先（SPT / Shortest Processing Time first）**，可以使总等待时间最小。**证明**：

   考虑任意一个非最优顺序，其中存在相邻的两个顾客 $i, j$，服务时间满足 $t_i > t_j$，但顺序却是 $i$ 在前、$j$ 在后。设在他们之前已完成的服务时间总和为 $S$。原顺序下，这两人对总等待时间的贡献为

   $$(S) + (S + t_i) = 2S + t_i.$$

   若交换二者顺序，贡献变为

   $$(S) + (S + t_j) = 2S + t_j.$$

   由于 $t_j < t_i$，交换后总等待时间减少了

   $$(2S + t_i) - (2S + t_j) = t_i - t_j > 0.$$

   因此，只要存在“长作业在短作业之前”的相邻对，交换它们就能严格减少总等待时间。不断进行这样的交换，最终必然得到按服务时间非递减排序的顺序，而此时已无法再改进，总等待时间达到最小。

2. 给定：字符 $(a \sim h)$ 的频率分别为前 8 个 Fibonacci 数：

   $$F_0 = 1, F_1 = 1, F_2 = 2, F_3 = 3, F_4 = 5, F_5 = 8, F_6 = 13, F_7 = 21$$

Huffman 算法每一步都合并当前最小的两个权值。由于 Fibonacci 数满足 $F_k = F_{k-1} + F_{k-2}$，每次合并后的新结点权值，正好等于下一个 Fibonacci 数。合并顺序为：

$$(F_0, F_1) \to F_2, \quad (F_2, F_2) \to F_3, \quad (F_3, F_3) \to F_4, \quad \dots$$

一直向上。结果频率最小的字符在最深层，频率最大的字符在最浅层，可得到如下编码：

| 字符 | 频率 | Huffman 编码 | 编码长度 |
|---|---|---|---|
| h | 21 | 0 | 1 |
| g | 13 | 10 | 2 |
| f | 8 | 110 | 3 |
| e | 5 | 1110 | 4 |
| d | 3 | 11110 | 5 |
| c | 2 | 111110 | 6 |
| b | 1 | 1111110 | 7 |
| a | 1 | 1111111 | 7 |

很容易推广到前 $n$ 个 Fibonacci 数的情况：Huffman 树退化为一棵**单支链，**频率最大的字符 $F_{n-1}$ 编码长度为 $1$，次大的为 $2$，.....，第 $k$ 小的编码长度为 $k$，最小的两个字符编码长度为 $n-1$。

3. 设程序长度按非降序排列：

$$a_1 \le a_2 \le \cdots \le a_n.$$

(1) 算法：按上述顺序依次加入程序，只要当前总长度不超过 $L$。设算法得到的集合为

$$Q = a_1, a_2, \dots, a_k, \quad \sum_{i=1}^{k} a_i \le L, \quad \sum_{i=1}^{k+1} a_i > L.$$

反证：假设存在另一集合 $Q'$，其程序个数为 $m > k$，且

$$\sum_{a_i \in Q'} a_i \le L.$$

由于 $Q'$ 含有 $m$ 个程序，而 $a_1, \ldots, a_m$ 是最小的 $m$ 个长度，有

$$\sum_{i=1}^{m} a_i \leq \sum_{a_i \in Q'} a_i \leq L.$$

但 $m > k$意味着

$$\sum_{i=1}^{k+1} a_i \leq \sum_{i=1}^{m} a_i \leq L,$$

这与 $\sum_{i=1}^{k+1} a_i > L$ 矛盾。因此不存在比 $Q$ 更大的可行集合，贪心算法得到的 $Q$ 在**程序个数意义下是最大子集合**。

(2) 磁带利用率定义为

$$\rho = \frac{\sum_{a_i \in Q} a_i}{L}.$$

由构造知：

$$\sum_{i=1}^{k+1} a_i > L \Rightarrow a_{k+1} > L - \sum_{i=1}^{k} a_i.$$

又因 $a_{k+1} \geq a_k$，得

$$\sum_{i=1}^{k} a_i \geq L - a_k \geq \frac{L}{2}.$$

因此

$$\frac{1}{2} \leq \rho < 1.$$

**结论**：该贪心算法的磁带利用率至少为 **50%**。

(3) 反例构造：令 $L = 10$，程序长度为 $a = 6, 6, 4$. 按非降序：$4, 6, 6$. 贪心选择：

$$Q = 4, \quad \sum a_i = 4, \quad \rho = 0.4.$$

但选择集合

$$Q' = 6, \quad \sum a_i = 6, \quad \rho = 0.6.$$

显然

$$\frac{\sum_{a_i \in Q} }{L} < \frac{\sum_{a_i \in Q'} }{L}.$$

**结论**：该贪心策略保证“程序数最多”，但**不保证磁带利用率最大**。

4. 伪代码：

```
Huffman(S):
    输入：字符集合 S，每个字符 x 有频率 w(x)
    输出：Huffman 树 T

    建立最小优先队列 Q
    for 每个 x ∈ S:
        创建结点 node(x)，权值 = w(x)
        Q.push(node)

    while Q.size > 1:
        x = Q.pop_min()
        y = Q.pop_min()
        z = new node
        z.weight = x.weight + y.weight
        z.left = x
        z.right = y
        Q.push(z)

    return Q.pop_min()   // Huffman 树根
```

C++ 实现：

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
```

```
    char ch;
    int freq;
    Node *left, *right;

    Node(char c, int f) : ch(c), freq(f), left(nullptr),
right(nullptr) {}
    Node(int f, Node* l, Node* r) : ch(0), freq(f), left
(l), right(r) {}
};

struct Cmp {
    bool operator()(Node* a, Node* b) {
        return a->freq > b->freq; // 小根堆
    }
};

void generateCode(Node* root, string code,
                  unordered_map<char, string>& huff) {
    if (!root) return;
    if (!root->left && !root->right) {
        huff[root->ch] = code;
        return;
    }
    generateCode(root->left, code + "0", huff);
    generateCode(root->right, code + "1", huff);
}

int main() {
    vector<char> chars =
{'a','b','c','d','e','f','g','h'};
    vector<int> freq  = {1,1,2,3,5,8,13,21};

    priority_queue<Node*, vector<Node*>, Cmp> pq;

    for (int i = 0; i < chars.size(); i++) {
        pq.push(new Node(chars[i], freq[i]));
    }
```

```
    while (pq.size() > 1) {
        Node* x = pq.top(); pq.pop();
        Node* y = pq.top(); pq.pop();
        pq.push(new Node(x->freq + y->freq, x, y));
    }

    Node* root = pq.top();

    unordered_map<char, string> huffmanCode;
    generateCode(root, "", huffmanCode);

    for (auto& p : huffmanCode) {
        cout << p.first << " : " << p.second << endl;
    }

    return 0;
}
```

6. 定义集合系统 $(E, \mathcal{I})$，其中

$$\mathcal{I} = F \subseteq E \mid F \text{ 不含环（即森林）}.$$

验证拟阵三公理：

1）**非空性：**空集不含环，$\varnothing \in \mathcal{I}$。

2）**遗传性：**若 $F \in \mathcal{I}$，任意子集 $F' \subseteq F$ 仍不含环，故 $F' \in \mathcal{I}$。

3）**交换性：**若 $F_1, F_2 \in \mathcal{I}$，且 $|F_1| < |F_2|$，则 $F_2$ 中必存在一条边 $e \notin F_1$，使 $F_1 \cup e$ 仍无环。这正是**图拟阵（Graphic Matroid）**。因此，MST 是一个**带权拟阵的最大独立集最小权问题**。

对每条边定义权值函数

$$w : E \to \mathbb{R}^+.$$

目标是在拟阵 $(E, \mathcal{I})$ 中，找一个**基（大小为 $|V| - 1$ 的独立集）**，使 $\sum_{e \in T} w(e)$ 最小。拟阵定理告诉我们：**在带权拟阵中，按权值递增顺序进行贪心选择，一定能得到最优解。**

**切割性质（安全边定理）：**对任意一个割 $(S, V \setminus S)$，跨越该割的**最小权值边**一定属于某棵最小生成树。

Kruskal 做的是：将边按权值从小到大排序；依次加入当前不产生环的最小边。由于“不产生环” ⇔ 保持在拟阵独立集内，每一步选的是某个割上的最小边，由切割性质，该边是**安全边**，必属于某个 MST，拟阵贪心定理保证整体最优。所以 Kruskal 是**全局按权值的拟阵贪心算法**。

Prim 做的是：维护一个已选顶点集合 $S$；每一步选连接 $S$ 与 $V \setminus S$ 的最小权边。由于当前 $(S, V \setminus S)$ 构成一个割，Prim 每一步都选该割上的最小权边，根据切割性质，这条边是安全边，不断加入安全边，最终得到 MST。所以 Prim 是**在“顶点扩展视角”下的局部贪心，但每一步仍是全局安全的**。

7. 多机调度 $m$ 台相同机器的 LPT（Longest Processing Time first）贪心伪代码如下：

```
LPT_Schedule(jobs[1..n], m):
    # jobs[i] has processing time p[i]
    sort jobs by non-increasing processing time (p descendi
ng)

    for k = 1..m:
        load[k] = 0
        machineJobs[k] = empty list

    for each job j in jobs (in sorted order):
        k = argmin_{t in 1..m} load[t]          # least loa
ded machine
        assign job j to machine k
        machineJobs[k].append(j)
        load[k] = load[k] + p[j]

    makespan = max_{k=1..m} load[k]
    return machineJobs, load, makespan
```