贪心算法

引入：对于大部分最优化问题，使用动态规划来求最优解有点浪费，可以使用更加高效、简单的算法——贪心算法。

> 贪心算法在每一步的选择中，都选择当时最佳的情况。它并不能保证得到最优解，但是很多问题确实可以求得最优解。

Part 1 活动选择问题

有n个活动的集合S={a1,a2,...,an}，这些活动使用同一个资源(如同一个教室)，但它同时只能提供给一个活动，每个活动有一个**开始时间** *si* 和**结束时间** *fi*，任务ai发生在半开区间 [*si,fi*)。

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

1. 用动态规划方法

*c[i,j]*表示最优解的集合的大小。 则得到递归式：***c[i,j]=c[i,k]+c[k,j]+1*** 如果不知道 ***sij*** 的最优解包含 ***ak*** ，则：

$$c[i,j]=\begin{cases}0 & \text{若 } S_{ij}=\varnothing \\ \max\limits_{a_k\in S_{ij}}\{c[i,k]+c[k,j]+1\} & \text{若 } S_{ij}\neq\varnothing\end{cases}$$

2. 贪心算法

1. **最优子结构** *令Sij表示在ai结束之后开始，且在aj开始之前结束的那些活动的集合,其最优解Aik已知包含活动ak,*则对其子问题*Sik*和*Skj*的最优解*Aik*和*Akj*，有*Aij=Aik∪{ak}∪Akj*。

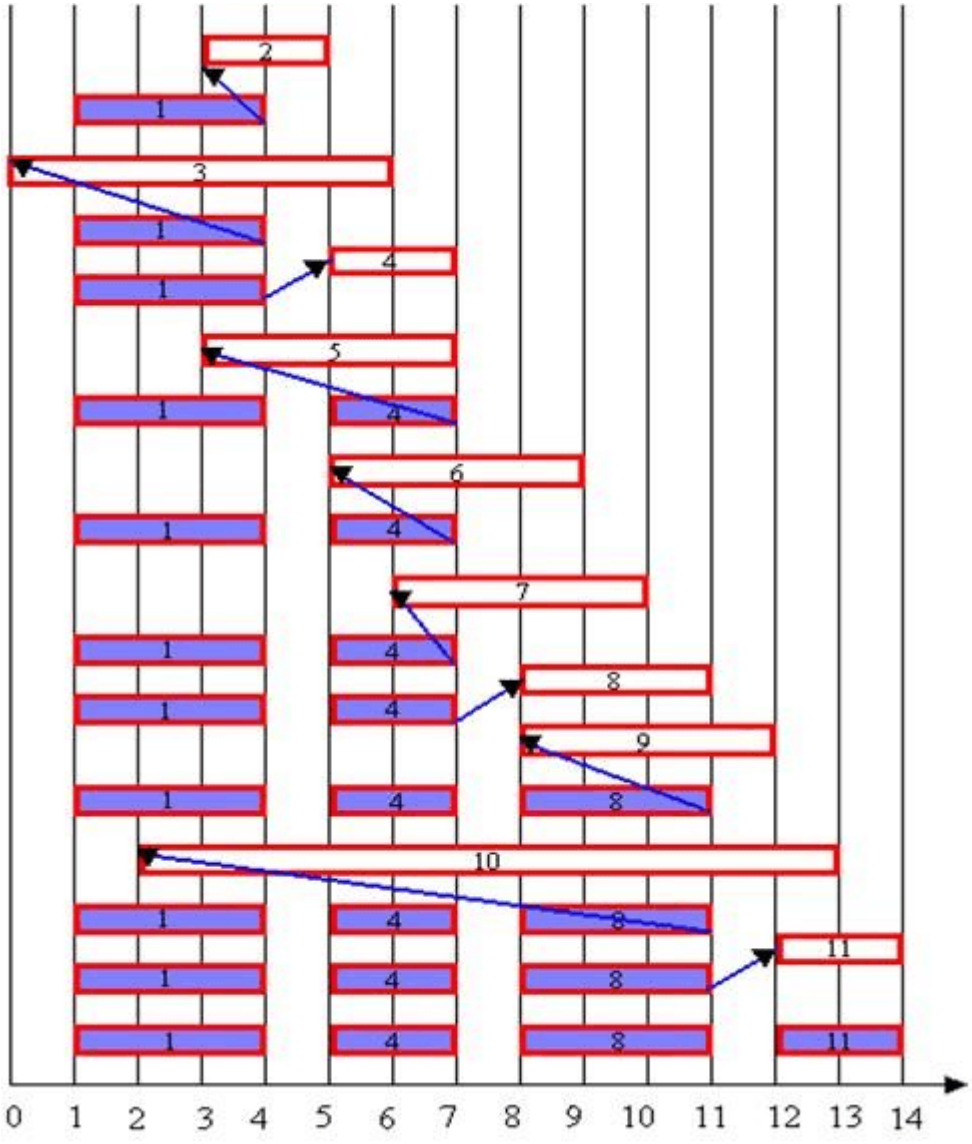
> 反证法：若存在|*Aij*'|>|*Aij*|,即存在|*Aik*'|+|*Akj*'|+1>|*Aik*|+|*Akj*|+1,与*Aik*,*Akj*为最优解相矛盾。

2. **贪心选择** 选择一个活动使得选出它后剩下的资源能被尽量多的其他任务所用。根据直觉，我们首选的活动应该是S中最早结束的活动。

伪代码

递归贪心算法

```
RECURSIVE-ACTIVITY-SELECTOR(s,f,k,n)
m=k+1
while m<=n and s[m]<f[k]
    m=m+1
if m <= n
    return {am} U RECURSIVE-ACTIVITY-SELECTOR(s,f,m,n)
else return empty
```



迭代贪心算法

```
GREEDY-ACTIVITY-SELECTOR(s,f)
n=s.length
A={a1}
k=1
for m=2 to n
    if s[m]>=f[k]
        A=A U {am}
        k=m
return A
```

Part 2 贪心算法原理

步骤：

1. 将最优化问题转换为这样的形式：对其做出一些选择后，只剩下一个子问题需要求解。
2. 证明做出贪心选择后原问题总是存在最优解，即贪心选择总是安全的。
3. 证明做出贪心选择后剩余的子问题满足性质：其最优解与贪心选择组合即可得到原问题的最优解，这样就得到了最优子结构。

1. **贪心选择性质** 可以通过做出局部最优解来构建全局最优解，而不用考虑子问题的解。

> 贪心算法进行选择时可能依赖之前做出的选择，但不依赖任何将来的选择或是子问题的解。

2. **最优子结构** 一个问题的最优解能够包含其子问题的最优解。

> 这个性质是能否应用动态规划和贪心算法的关键要素。

贪心算法和动态规划的微小差别

可以通过两个相似问题的比较来进行分辨。

1. 0-1背包问题(动态规划)

   > 只能对商品完整的拿取或留下。(金条)

2. 分数背包问题(贪心算法)

> 对商品可以只拿走一部分。(金砂)

其中第 i 个商品价值 vi 美元，重 wi 磅，背包最多容纳 W 磅重的商品。vi,wi 都是整数。

Part3 赫夫曼编码

考虑二进制编码

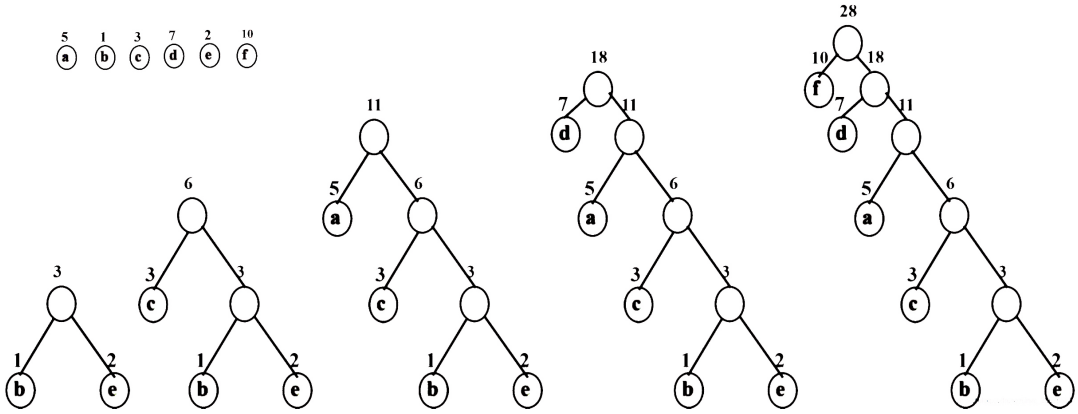
1. 定长编码 每个字符的编码长度相同。
2. 变长编码 对高频字符赋予短码，低频字符赋予长码。

> 使用**前缀码** (即没有如何码字是其他码字的前缀)，可以保证达到最优数据压缩率，也不会使编码丧失一般性，使编码文件的开始部分无歧义。

对于一个文件，最优的编码方案总是一颗**满二叉树**。

构造赫夫曼编码

```
HUFFMAN(C)
n=|C|
Q=C
for i=1 to n-1
    allocate a new node z
    z.left=x=EXTRACT-MIN(Q)//寻找Q中频率最低的数作为左孩子
    z.right=y=EXTRACT-MIN(Q)
    z.freq=x.freq+y.freq
    INSERT(Q,z)
return EXTRACT-MIN(Q)
```



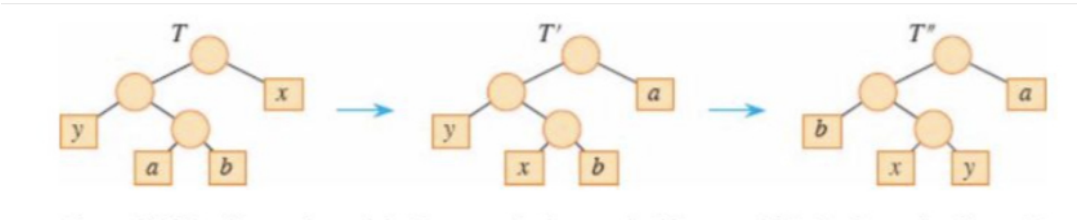
判断正确性

要证明赫夫曼编码是正确的，就要证明最优前缀码具有**贪心选择**和**最优子结构**性质。

证明贪心选择

引理： 给定一个字母表*C*，每个字符*c*的频率为*c.freq*，若*x*和*y*是*C*中频率最低的两个字符，则存在一个*C*的最优前缀码，其中*x*和*y*的码字长度相同，且只有最后一个二进制位不同。

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T'}(c) \\
&= (a.freq - x.freq)(d_T(a) - d_T(x)) \\
&\geq 0
\end{aligned}
$$



证明最优子结构

给定一个字母表 C，每个字符*c*的频率为*c.frep**，*若x和y是 C 中频率最低的两个字符，定义字符z且z.frep = x.frep + y.frep，定义字母表 C"=(C - {*x,y})U{z}*，字母表 C 的任一最优前缀无关编码对应的一棵二叉树 T'，将 T' 中的字符*z*对应的叶结点修改为一个内部结点且两个孩子结点分别为字符*x*和*y*对应的叶结点，这样可以得到字母表 C 的某一最优前缀无关编码对应的一棵二叉树 T。

$$
\begin{aligned}
C' &= (C - \{x, y\}) \cup \{z\} \\
z.freq &= x.freq + y.freq \\
d_T(x) &= d_T(y) = d_{T'}(z) + 1
\end{aligned}
$$

证明：由 $B(T) = \sum_{c \in C} c.freq \cdot d_T(c)$ ，可得 $B(T) = B(T') + x.freq + y.freq$ 。

$$
B(T') = \sum_{c \in C'} c.freq \cdot d_{T'}(c)
$$

Part4 离线缓存

缓存：容量比主存小，速度比主存快的**存储器**。 计算机通过把需要访问的数据的一部分储存在缓存中，可以减少数据的访问时间。缓存将数据有组织地存放在**缓存块**中，缓存块大小一般是32、64或128字节

主存：虚拟内存系统中，主内存可以被视为驻留在磁盘上的数据的缓存。这些主存块被称为页，页大小一般是4096字节。

当一个程序执行时，需要进行一系列的存储器请求。假设有*n*个访存请求，这些数据按照请求顺序分别在*b1,b2,...,bn*块中。事实上，这些请求不会完全不同，多个请求有可能需要访问同一个块。

当需要访问*bi*时，会出现以下三种情况：

情况一：由于之前访问过*bi*，*bi*已经在缓存中，当需要再次访问*bi*时，可以直接访问缓存，称为缓存命中**。 **情况二**：*bi*不在缓存中，缓存未满，当需要访问*bi*时，直接将*bi*填充到空闲的缓存块。 **情况三**：*bi*不在缓存中，缓存已满，当需要访问*bi*时，需要预先将某一个缓存块空出来，然后将*bi*填充到空闲的缓存块。

情况二和情况三称为**缓存未命中**。情况二称为**强制未命中**。

一般情况下，由于计算机无法知道未来的请求，因此缓存是一个在线问题。这里我们仅仅考虑缓存问题的离线版本，即已知完整的访存请求序列和缓存块数量，我们的目标是最小化缓存未命中，最大化缓存命中。

我们采用称为**将来最久**的贪心策略求解离线缓存问题，即如果缓存已满，那么选择已在缓存中且访存序列中将来最久到达的块进行置换。

离线缓存的最优子结构

定义子问题 (C,i)，缓存块集合为 C，访存请求序列为 $b_i,b_{i+1},\ldots,b_n$，当前需要请求 b_i，设可用的缓存块数量最多为 k，$|C|\le k$。子问题 (C,i) 的最优解要求使得缓存未命中数最小。

设子问题 (C,i) 的最优解为 S，请求 b_i 完成后缓存块集合为 C'，子问题 $(C',i+1)$ 的最优解为 S'。

定义 $R_{C,i}$ 为子问题 (C,i) 请求 b_i 完成后可能的缓存块集合的集合，有：

- 情况一：$R_{C,i}=\{C\}$
- 情况二：$R_{C,i}=\{C\cup\{b_i\}\}$
- 情况三：$R_{C,i}=\{(C-x)\cup\{b_i\}:x\in C\}$

定义 $miss(C,i)$ 为子问题 (C,i) 的最优解的缓存未命中数。递归式如下：

$$miss(C,i)=\begin{cases}0 & i=n\wedge b_n\in C\\ 1 & i=n\wedge b_n\notin C\\ miss(C,i+1) & i<n\wedge b_i\in C\\ 1+\min\{miss(C',i+1):C'\in R_{C,i}\} & i<n\wedge b_i\notin C\end{cases}$$

离线缓存的贪心选择性质

考虑子问题 (C,i)，缓存块集合为 C 有 k 个缓存块，即此时缓存已满，且出现缓存未命中。当前需要请求 b_i，设 $z=b_m$ 为 C 中将来最久被请求的缓存块，（如果有缓存块将来不再被访问，优先考虑这样的缓存块成为 z，所以可以增加虚缓存块，使得 $z=b_m=b_{n+1}$。）我们可以移除缓存块 z，然后将增加缓存块 b_i，如此操作可以得到子问题 (C,i) 的某一个最优解。

证明：设子问题 (C,i) 的最优解为 S，S 为增加缓存块 b_i 而移除缓存块 z。如果增加缓存块 b_i 而移除缓存块 x，设此最优解为 S'。

定义 $C_{S,j}$ 为执行最优解 S 后且请求 b_j 前的缓存块集合，同理 $C_{S',j}$ 为执行最优解 S' 后且请求 b_j 前的缓存块集合。

下面将说明根据如下性质构造 S'。

1. 对于 $j=i+1,\ldots,m$，令 $D_j=C_{S,j}\cap C_{S',j}$，$|D_j|\ge k-1$，$C_{S,j}$ 和 $C_{S',j}$ 最多有一个缓存块不同。若 $C_{S,j}$ 和 $C_{S',j}$ 不同，则 $C_{S,j}=D_j\cup\{z\}$ 且 $C_{S',j}=D_j\cup\{y\}$，其中 $y\ne z$。
2. 对于请求序列 $b_i,\ldots,b_{m-1}$ 中每个请求，若 S 缓存命中，则 S' 也缓存命中。
3. 对于所有 $j>m$，$C_{S,j}$ 的缓存块集合与 $C_{S',j}$ 的缓存块集合完全相同。
4. 对于请求序列 $b_i,\ldots,b_{m-1}$，S' 的缓存未命中数至多为 S 的缓存未命中数。

我们将用归纳法证明这些性质适用于每个请求.

1. 对于*j=i+1,...,m**，*归纳基础为CS,j和CS',j完全相同。对于需要请求 bi，S 移除缓存块z，S'移除缓存块x，CS,i+1和CS,i+1至多有一个缓存块不同。若CS,i+1和CS',i+1不同，则C*S,i+1=*D*i+1* ∪{*z}*且 CS',*i+1*=D*i+1* ∪{*x}*，其中*x≠z*。归纳步骤将证明S'在请求*bj*下如何运行，其中*i+1 <= J <= m-1*。归纳假设请求*bj*后性质1能够保持。由于*z=bm*为CS,*i*中将来最久被请求的缓存块，已知*bj≠z*。我们考虑以下几种情况：

- 若 $C_{S,j} = C_{S',j}$ （此时 $|D_j| = k$），则对于请求 b_j， S 和 S' 做出的选择相同，故 $C_{S,j+1} = C_{S',j+1}$。
- 若 $|D_j| = k - 1$ 且 $b_j \in D_j$，则 $b_j \in C_{S,j} \wedge b_j \in C_{S',j}$，$S$ 和 S' 均缓存命中，故 $C_{S,j+1} = C_{S,j} \wedge C_{S',j+1} = C_{S',j}$。
- 若 $|D_j| = k - 1$ 且 $b_j \notin D_j$，则 $C_{S,j} = D_j \cup \{z\} \wedge b_j \neq z$，$S$ 缓存不命中。故此时需要移除缓存块 z 或者别的某一缓存块 $w \in D_j$。
  - 若移除缓存块 z，则 $C_{S,j+1} = D_j \cup \{b_j\}$。根据 $b_j = y$ 还是 $b_j \neq y$，分成两种情况讨论：
    - 若 $b_j = y$，则 S' 缓存命中，有 $C_{S',j+1} = C_{S',j} = D_j \cup \{b_j\}$。此时 $C_{S,j+1} = C_{S',j+1}$。
    - 若 $b_j \neq y$，则 S' 缓存未命中，此时需要移除缓存块 y，有 $C_{S',j+1} = D_j \cup \{b_j\}$。此时 $C_{S,j+1} = C_{S',j+1}$。
  - 若移除别的某一缓存块 $w \in D_j$，则 $C_{S,j+1} = (D_j - \{w\}) \cup \{b_j, z\}$。根据 $b_j = y$ 还是 $b_j \neq y$，分成两种情况讨论：
    - 若 $b_j = y$，则 S' 缓存命中，有 $C_{S',j+1} = C_{S',j} = D_j \cup \{b_j\}$。由于 $w \in D_j$ 且 w 未被 S' 移除，有 $w \in C_{S',j+1}$。又 $w \notin D_{j+1}$ 且 $b_j \in D_{j+1}$，因此 $D_{j+1} = (D_j - \{w\}) \cup \{b_j\}$。此时 $C_{S,j+1} = D_{j+1} \cup \{z\}$ 且 $C_{S',j+1} = D_{j+1} \cup \{w\}$，因为 $w \neq z$，所以当需要请求 b_{j+1} 时，性质1能够保持。（换句话说，w 替换了性质1中的 y。）
    - 若 $b_j \neq y$，则 S' 缓存未命中，此时需要移除缓存块 w，有 $C_{S',j+1} = (D_j - \{w\}) \cup \{b_j, y\}$。又 $w \notin D_{j+1}$ 且 $b_j \in D_{j+1}$，因此 $D_{j+1} = (D_j - \{w\}) \cup \{b_j\}$。此时 $C_{S,j+1} = D_{j+1} \cup \{z\}$ 且 $C_{S',j+1} = D_{j+1} \cup \{y\}$。

2. 在上述关于保持性质1的讨论中，S 只在前两种情况可能出现缓存命中，S'只有在缓存 S 命中的情况下才可能缓存命中。

3. 若 $C_{S,m} = C_{S',m}$，则对于请求 b_m， S 和 S' 做出的选择相同，故 $C_{S,m+1} = C_{S',m+1}$。若 $C_{S,m} \neq C_{S',m}$，则根据性质1，有 $C_{S,m} = D_m \cup \{z\}$ 且 $C_{S',m} = D_m \cup \{y\}$，此时 $y \neq z$。由于 $z = b_m$ 为 C 中将来最久被请求的缓存块，因此 S 缓存命中，有 $C_{S,m+1} = C_{S,m} = D_m \cup \{z\}$， S' 移除缓存块 y 并增加缓存块 z，有 $C_{S',m+1} = D_m \cup \{z\} = C_{S,m+1}$。综上，无论 $C_{S,m} = C_{S',m}$ 还是 $C_{S,m} \neq C_{S',m}$，都有 $C_{S,m+1} = C_{S',m+1}$。从请求 b_{m+1} 开始， S 和 S' 同步，两者做出的选择完全相同。

4. 根据性质2，对于请求序列 $b_i, \ldots, b_{m-1}$ 中每个请求，若 S 缓存命中，则 S' 也缓存命中。只剩下 $b_m = z$ 需要考虑，若对于请求 b_m， S 缓存未命中，则无论 S' 缓存命中还是缓存未命中， S' 的缓存未命中数至多为 S 的缓存未命中数。

若对于请求 b_m， S 缓存命中但 S' 缓存未命中，我们需要证明请求序列 $b_{i+1}, \ldots, b_{m-1}$ 中至少有一个请求会导致 S 缓存未命中但 S' 缓存命中。我们采用反证法，假设请求序列 $b_{i+1}, \ldots, b_{m-1}$ 中没有请求会导致 S 缓存未命中但 S' 缓存命中。

对于某个 $j > i$，有 $C_{S,j} = C_{S',j}$，由于 $b_m \in C_{S,m} \wedge b_m \notin C_{S',m}$，因此 $C_{S,m} \neq C_{S',m}$，所以 S 已经在请求序列 $b_i, \ldots, b_{m-1}$ 中将缓存块 z 移除了，因为只有这样，才能有 $C_{S,m+1} = C_{S',m+1}$。对于某个 $y \neq z$，有 $C_{S,j} = D_j \cup \{z\} \wedge C_{S',j} = D_j \cup \{y\}$， S 移除了某个缓存块 $w \in D_j$。此外，由于这些请求中任何一个请求都不会导致 S 缓存未命中但 S' 缓存命中，因此 $b_j = y$ 永远不可能发生。即对于 $b_{i+1}, \ldots, b_{m-1}$ 中任何一个请求 b_j 都不会导致 $y \in C_{S',j} - C_{S,j}$。请求 b_j 完成后，有 $C_{S',j+1} = D_{j+1} \cup \{y\}$，请求前后两个缓存块集合的差集并没有发生变化。让我们回到请求 b_i，有 $C_{S',i+1} = D_{i+1} \cup \{x\}$，由于此后直至 b_m 的请求序列都没有能够使得两个缓存块集合的差集并发生变化，因此有 $C_{S',j} = D_j \cup \{x\}$，其中 $j = i+1, \ldots, m$。

根据定义，请求 $z = b_m$ 在请求 x 之后，意味着 $b_{i+1}, \ldots, b_{m-1}$ 至少其中之一为 x，但对于 $j = i+1, \ldots, m$，有 $x \in C_{S',j} \wedge x \notin C_{S,j}$，所以其中至少有一次请求导致 S' 缓存未命中但 S 缓存命中，与假设请求序列 $b_{i+1}, \ldots, b_{m-1}$ 中没有请求会导致 S 缓存未命中但 S' 缓存命中矛盾。所以请求序列 $b_{i+1}, \ldots, b_{m-1}$ 中至少有一个请求会导致 S 缓存未命中但 S' 缓存命中。所以 S' 的缓存未命中数至多为 S 的缓存未命中数。由于假设 S 为最优解， S' 为最优解。