基础算法思想 I

基础-提高衔接计划

览遍千秋

2025-07-26

洛谷

www.luogu.com.cn

课前提示

- 上课的时候专心听讲解，**不要跟着老师抄代码**，下课后独立完成。
- 不使用 AI 做题，AI 会做不等于自己会。
- 不抄袭题解（含对照题解抄一遍），抄对不等于会做。
- 看完题解后，关闭题解独立练习。
- 练习中途遇到问题，应当分析题目及自己的思路，而非回忆题解或再次参考题解。

目录

前缀和与差分

前缀

- 给定长度为 n 的数列 $(a_1, a_2, \cdots, a_n)$
- 前缀是指以 a_1 为开头的连续的子序列

- $a = [1,7,4,5]$
- 长度为 2 的前缀 $[1,7]$

- 长度为 i 的前缀
- $(a_1, a_2, \cdots, a_i)$

前缀和

- 前缀和，顾名思义，前缀的和
- 用 $S_i/S[i]/S(i)$ 表示长度为 i 的前缀的和

$$S_i = a_1 + a_2 + \cdots + a_i = \sum_{j=1}^{i} a_j$$

前缀和

$$S_i = a_1 + a_2 + \cdots + a_i = \sum_{j=1}^{i} a_j$$

- 暴力计算前缀和的方法非常直观

```
for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= i; j++) {
        S[i] += a[j];
    }
}
```

- $O(n^2)$

前缀和

- S_{i-1} 和 S_i 的关系
- $S_i = S_{i-1} + a_i (i \geq 2)$

- 定义 $S_0 = 0$
- $S_i = S_{i-1} + a_i (i \geq 1)$
- $O(n)$

```
S[0] = 0;
for(int i = 1; i <= n; i++) {
    S[i] = S[i - 1] + a[i];
}
```

前缀和

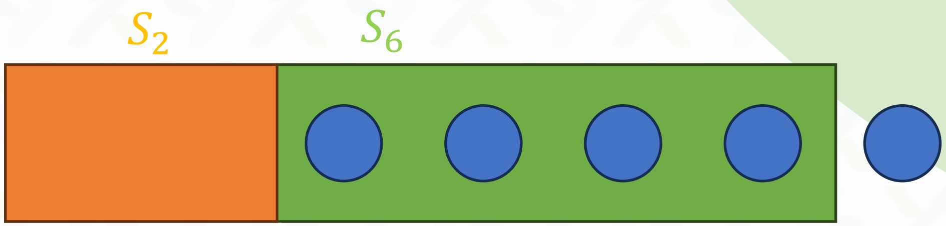
- 如何计算数列 a 下标在 $[l,r]$ 内的元素和？
- $a_l + a_{l+1} + \cdots + a_{r-1} + a_r$

- 暴力计算：循环枚举，$O(n)$

- 前缀和的定义
- $S_r = a_1 + a_2 + \cdots + a_r$
- $S_{l-1} = a_1 + a_2 + \cdots + a_{l-1}$
- $S_r - S_{l-1} = a_l + a_{l+1} + \cdots + a_{r-1} + a_r$
- $O(1)$

前缀和

$$S_r - S_{l-1} = \sum_{i=l}^{r} a_i$$



二维前缀和

- $S[i][j]$ 表示 $(1,1)$ 到 (i,j) 子矩形的和
- 暴力计算 $O(n^4)$

二维前缀和

- $S[i][j]$ 的递推关系略复杂
- $S[i][j] = a[i][j] + S[i-1][j] + S[i][j-1] - S[i-1][j-1]$

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

二维前缀和

- 求左上角为 (l_1, r_1)，右下角为 (l_2, r_2) 子矩阵的和 v
- $v = S[l_2][r_2] - S[l_1 - 1][r_2] - S[l_2][r_1 - 1] + S[l_1 - 1][r_1 - 1]$

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

前缀和

- 减是加的逆运算
- 从大的去掉多算的，补足重复扣除的

- 给定数列 a，计算 $[l,r]$ 的累积、异或和
- $M(i) = a_1 \cdot a_2 \cdot \cdots \cdot a_i$，$M(0) = 1$
- $X(i) = a_1 \oplus a_2 \oplus \cdots \oplus a_i$，$X(0) = 0$
- $\frac{M(r)}{M(l-1)}$
- $X(r) \oplus X(l-1)$

P1387 最大正方形

https://www.luogu.com.cn/problem/P1387

- 给定一个 $n \times m$ 的 01 矩阵
- 计算最大全 1 子正方形的边长
- $1 \leq n, m \leq 100$

P1387 最大正方形

- 暴力
- 枚举子正方形的左上角和边长
- 枚举子矩阵内部元素
- $O(n^5)$

P1387 最大正方形

- 如果 $r \times r$ 的正方形全为 1
- 那么这个正方形的和应该为 r^2
- 可以用二维前缀和优化枚举子矩阵内所有元素的过程
- $O(n^3)$

P1387 最大正方形

```
for(int i = 1; i <= n; i++) {
  for(int j = 1; j <= m; j++) {
    s[i][j] = s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1] + a[i][j];
  }
}
for(int i = 1; i <= n; i++) {
  for(int j = 1; j <= m; j++) {
    for(int r = 1; i + r - 1 <= n && j + r - 1 <= m; r++) {
      int x = i + r - 1, y = j + r - 1;
      int S = s[x][y] - s[x][j - 1] - s[i - 1][y] + s[i - 1][j - 1];
      if(S == r * r) {
        ans = max(ans, r);
      }
    }
  }
}
```

差分

- 一种相对于前缀和的策略
- a 为原数组，b 为差分数组

$$b_i = a_i - a_{i-1} (2 \le i \le n)$$
$$b_1 = a_1$$

- b 数组的前缀和为 a 数组
- $b_1 + b_2 + \cdots + b_i = a_1 + (a_2 - a_1) + \cdots + (a_i - a_{i-1}) = a_i$

差分

- 差分可以解决区间修改问题
- 将 a 数组的 $[l,r]$ 全部加 k
- 转化为 b 数组 $b[l]$ 加 k，$b[r+1]$ 减 k

差分

- 某学校拟订购报纸，第 0 天时订购 0 份
- 第 1 天在第 0 天的基础上增加 b_1 份，此时为 a_1 份
- 第 2 天在第 1 天的基础上增加 b_2 份，此时为 $b_1 + b_2 = a_2$ 份
- ……
- 第 i 天在第 $i-1$ 天的基础上增加 b_i 份，此时总计 a_i 份

- 第 $l \sim r$ 天增加 k 份
- 从第 l 天开始增加，即 b_l 增加 k
- 从第 $r+1$ 天恢复原状，即 b_{r+1} 减少 k

二维差分

- 原数组是差分数组的前缀和
- $a[i][j] = b[i][j] + a[i-1][j] + a[i][j-1] - a[i-1][j-1]$

- $b[i][j] = a[i][j] + a[i-1][j-1] - a[i-1][j] - a[i][j-1]$

二维差分

- 对以 (l_1,r_1) 为左上角，(l_2,r_2) 为右下角的子矩阵全部加 k

- $b[l_1][r_1]$ 增加 k
- $b[l_1][r_2+1]$ 减少 k
- $b[l_2+1][r_1]$ 减少 k
- $b[l_2+1][r_2+1]$ 增加 k

二维差分

- $b[l_1][r_1]$ 增加 k
- $b[l_1][r_2+1]$ 减少 k
- $b[l_2+1][r_1]$ 减少 k
- $b[l_2+1][r_2+1]$ 增加 k

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

P4552 IncDec Sequence

https://www.luogu.com.cn/problem/P4552

- 给定长度为 n 的数列 a
- 每次可以选择区间 $[l,r]$ 并将其全部增加 1 或减少 1
- 计算最少多少次操作可以使 a 的数全部相同
- 在操作次数最少的情况下，a 有多少种可能取值

P4552 IncDec Sequence

- 计算 a 的差分数组 b
- 对 $[l,r]$ 的操作转化为 b_l, b_{r+1} 的操作
- a 全部相同时 $b_2 \sim b_n$ 均为 0
- b_1 为 a 最终的值

P4552 IncDec Sequence

- 每次在 $b_2 \sim b_n$ 中选择一对正数与负数进行操作
- 只有正数/负数和 0 后，再对 b_1/b_{n+1} 操作

- $b_2 \sim b_n$ 中，正数的和为 pos，负数的和的绝对值为 neg
- 相消操作次数 $\min(pos, neg)$
- 与 b_1/b_{n+1} 操作次数 $|pos - neg|$
- 总操作次数 $\max(pos, neg)$

- 在 b_1 上最多操作 $|pos - neg|$ 次
- b_1 还有初值，答案为 $|pos - neg| + 1$

P4552 IncDec Sequence

```
for(int i = 1; i <= n; i++) {
    cin >> a[i];
    b[i] = a[i] - a[i - 1];
    if(i == 1) continue;
    if(b[i] > 0) pos += b[i];
    else neg -= b[i];
}
cout << max(pos, neg) << endl;
cout << abs(pos - neg) + 1 << endl;
```

双指针

双指针

- 比较灵活的算法
- 同时使用两个指针，在序列、链表结构上指向的是位置，在树、图结构中指向的是节点，通过或同向移动，或相向移动来维护、统计信息。

滑动窗口

- l,r 两个指针维护区间窗口 $[l,r]$
- 每次 r 向右移动的同时，维护信息数组
- 按照题目要求向右移动 l

- 利用单调性、子串连续性等要求来实现【题目要求】

B4006 宝箱

https://www.luogu.com.cn/problem/B4006

小杨发现了 n 个宝箱，其中第 i 个宝箱的价值是 a_i。

小杨可以选择一些宝箱放入背包并带走，但是小杨的背包比较特殊，假设小杨选择的宝箱中最大价值为 x，最小价值为 y，小杨需要保证 $x - y \leq k$，否则小杨的背包会损坏。

小杨想知道背包不损坏的情况下，自己能够带走宝箱的总价值最大是多少。

B4006 宝箱

- 对 a 排序
- 选择的宝物一定是排序后 a 的连续一段

- $[l, r]$ 宝物中最大价值为 $a[r]$，最小价值为 $a[l]$
- 题目要求满足 $a[r] - a[l] \le k$

- 每次将 r 向右移动 1
- 向右移动 l 直到满足 $a[r] - a[l] \le k$

B4006 宝箱

```
sort(a + 1, a + n + 1);
for(int i = 1; i <= n; i++) {
    S[i] = S[i - 1] + a[i];
}
int l = 1, ans = 0;
for(int r = 1; r <= n; r++) {
    while(l < r && a[r] - a[l] > k) l++;
    ans = max(ans, S[r] - S[l - 1]);
}
```

对撞指针

- $[l, r]$ 一开始指向 $[1, n]$
- 每次缩小 $[l, r]$ 一长度（或两长度），缩减答案统计范围
- $[l+1, r]$ 或 $[l, r-1]$ 或 $[l+1, r-1]$

判断回文串

- 判别 S 是否为回文串
- $l = 1, r = |S|$
- 判断 S_l 与 S_r 是否相同，不相同则不是回文串
- $l \leftarrow l + 1, r \leftarrow r - 1$

快慢指针

- l, r 同向移动
- l 慢一些，r 快一些

- 通过指针的快慢差异来维护一些信息

将零移动至数列末尾

- 给定数列 a，将数列 a 中所有的 0 移动至数列末尾
- 不允许使用新的数组
- 例如 [4,0,0,8,7,0]，移动后变为 [4,8,7,0,0,0]

- 用 l 表示下一个非 0 数应当换到的位置
- 用 r 表示正在处理第 r 个数
- 如果 $a[r] \neq 0$ 则交换 $a[l], a[r]$
- l 移动的比 r 慢

B3773 完美字符串

https://www.luogu.com.cn/problem/B3773

- 给定字符串 S
- 求包含全部 26 个字母的最短子串长度

B3773 完美字符串

- $cnt[i]$ 表示字母 i 出现次数
- val 表示已经出现的字母种类数

- l, r 指向第 l 个与第 r 个单词

- 每次向右移动 r
- 很容易维护 cnt 与 val
- 尽力向右移动 l
- 直到 $l = r$ 或 向右移动 l 会导致 val 减少

B3773 完美字符串

- https://www.luogu.com.cn/record/222009225

双指针的时间复杂度

```
int l = 1, ans = 0;
for(int r = 1; r <= n; r++) {
    while(l < r && a[r] - a[l] > k) l++;
    ans = max(ans, S[r] - S[l - 1]);
}
```

- 两重循环，时间复杂度是 $O(n^2)$ 吗？
- l,r 均单调地从 1 移动到 n
- 时间复杂度为 $O(n)$

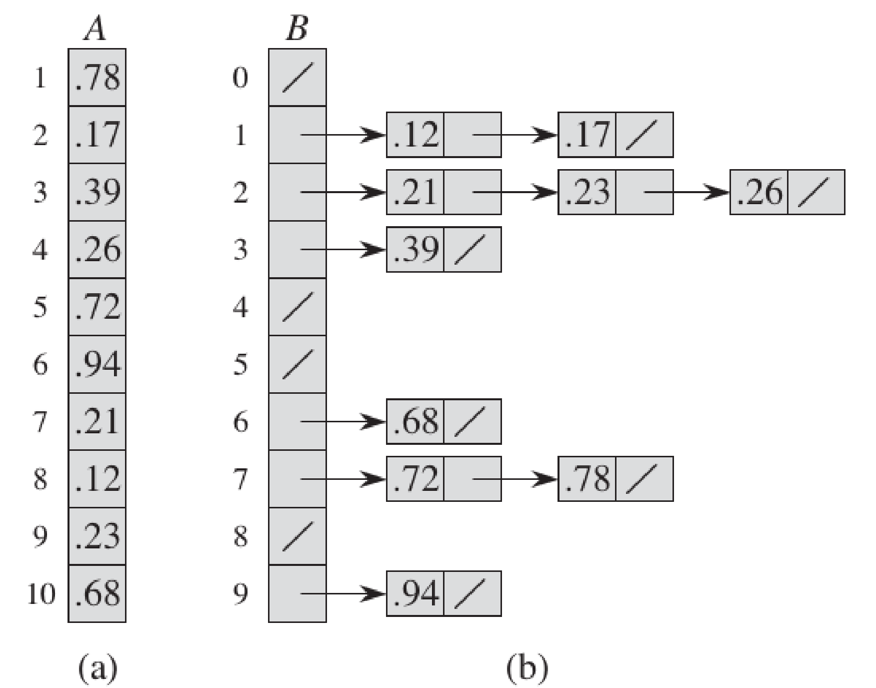
计数排序与 sort

计数排序

- 用 $cnt[i]$ 表示数字 i 出现的次数
- 每次输入 x，增加 $cnt[x]$

计数排序与桶排序

- 计数排序：统计每个元素出现的次数
- 桶排序：按大小将元素划分为不同区间，每个区间对应一个桶，用链表存储数据



sort

- 头文件 algorithm 提供的快速排序函数
- 底层结合快速排序与堆排序
- 时间复杂度 $O(n\log n)$，不存在快速排序的退化问题
- 不稳定排序

- stable_sort

sort

- sort 的范围是左闭右开，即开始位置参数在排序范围内，而终止位置参数不在排序范围内
- 对数组 $a[l] \sim a[r]$ 排序
- `sort(a + l, a + r + 1);`
- 默认从小往大排序
- 对 vector 排序
- 不能用 vec[0] 和 vec[vec.size()]-1 来表示位置
- `sort(vec.begin(), vec.end());`
- vec.begin() 返回指向第一个元素的迭代器
- vec.end() 返回指向最后一个元素下一个位置的迭代器

sort

- 自定义排序顺序
- 如果需要从大到小排序
- `sort(a + l, a + r + 1, vector <int>, greater<int>());`
- `sort(vec.begin(), vec.end(), vector<int>, greater<int>());`

- 也可以编写自定义比较函数 cmp
- `bool cmp(int a, int b) {return a > b;}`
- `sort(a + l, a + r + 1, cmp);`
- `sort(vec.begin(), vec.end(), cmp);`

sort

- 结构体排序
- struct T {/* some definition */};
- 自定义比较函数或重载运算符

- bool cmp(T a, T b) {
-     if(a.一关 != b.一关) return a.一 > b.一;
-     if(a.二关 != b.二关) return a.二 < b.二;
-     return a.第 X 关键字 < b.第 X 关键字;
- }

sort

- 重载运算符

```
struct T {
    string name;
    int score;
    bool operator <(const T& a) const {
        return score == a.score? name < a.name:
                                  score > a.score;
    }
}
```

贪心

哈夫曼树与哈夫曼编码

- 树的带权路径长度（WPL）：设二叉树有 n 个带权叶节点，从根节点到各叶节点的路径长度与相应叶节点权值的乘积之和
- 哈夫曼树：对于给定一组具有确定权值的叶节点，可以构造出不同的二叉树，其中 WPL 最小的二叉树称为哈夫曼树（Huffman Tree）
- 哈夫曼编码：左子树 0，右子树 1，到根结点路径自然形成的编码
- 性质：对于一篇文章，使用哈夫曼编码期望长度最短
- 哈夫曼编码不存在歧义，即不存在一个字符的编码是另一个字符编码的前缀

哈夫曼编码

10. 假设有一组字符{a,b,c,d,e,f}，对应的频率分别为 5%、9%、12%、13%、16%、45%。请问以下哪个选项是字符 a,b,c,d,e,f 分别对应的一组哈夫曼编码？（　）

A. 1111, 1110, 101, 100, 110, 0

B. 1010, 1001, 1000, 011, 010, 00

C. 000, 001, 010, 011, 10, 11

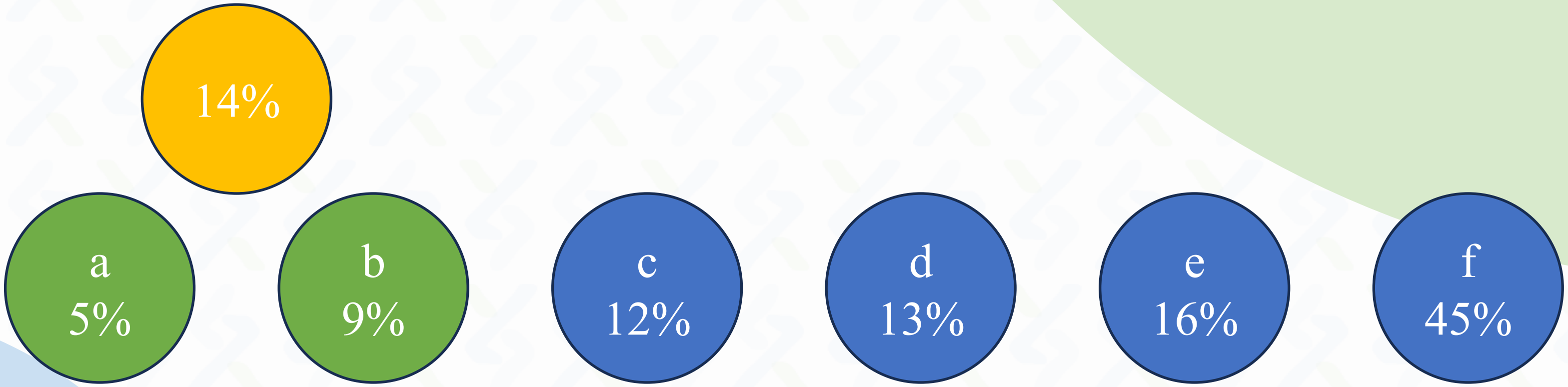
D. 1010, 1011, 110, 111, 00, 01

哈夫曼编码

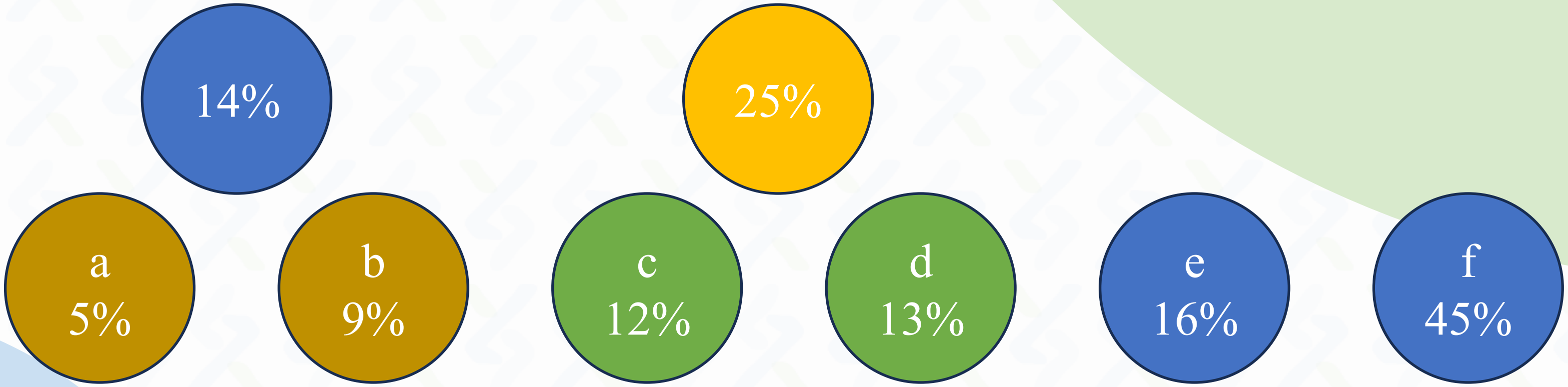
- 答案：A
- 考查要点：哈夫曼编码
- 哈夫曼编码是一种用于数据压缩的无损编码算法，它通过对数据中使用频率高的字符进行短编码，对使用频率低的字符进行长编码，从而达到压缩数据的目的。
- 构建哈夫曼编码的步骤：构建哈夫曼树
- 取当前频率最小的两个结点，插入一个新点作为其父节点，父节点频率为两子节点频率之和。

哈夫曼编码

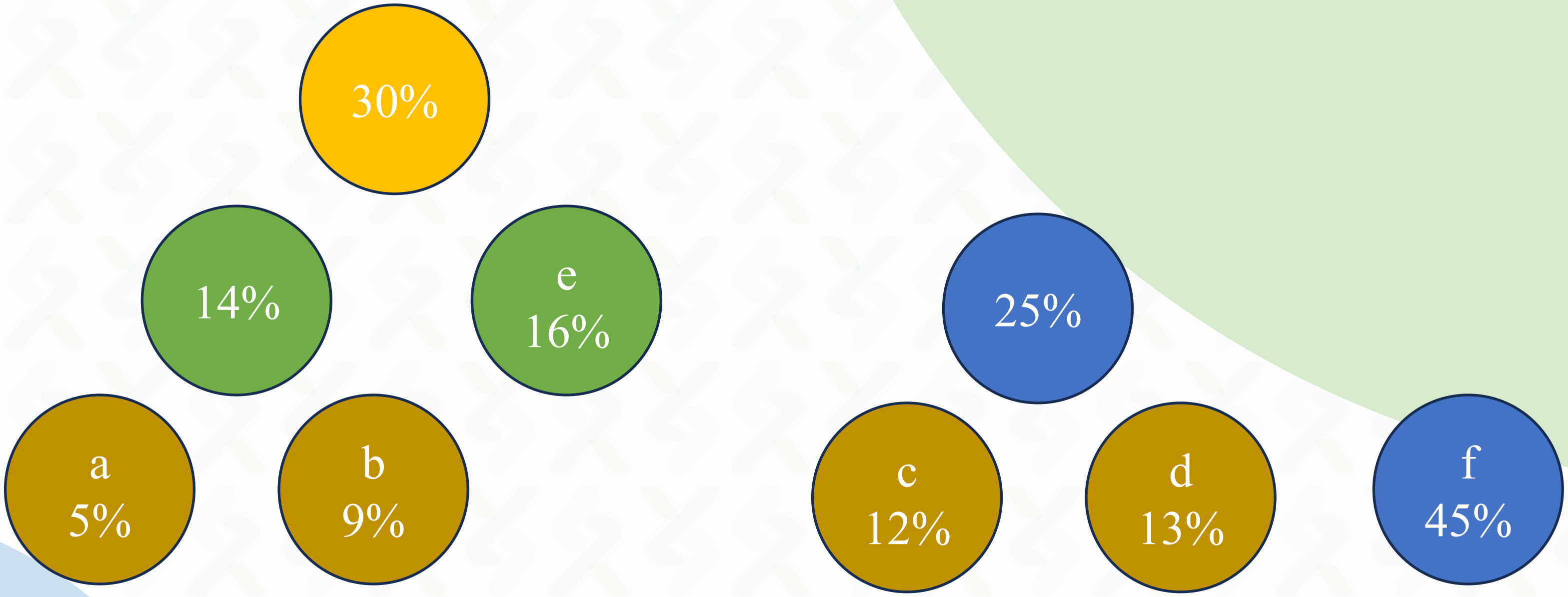
哈夫曼编码



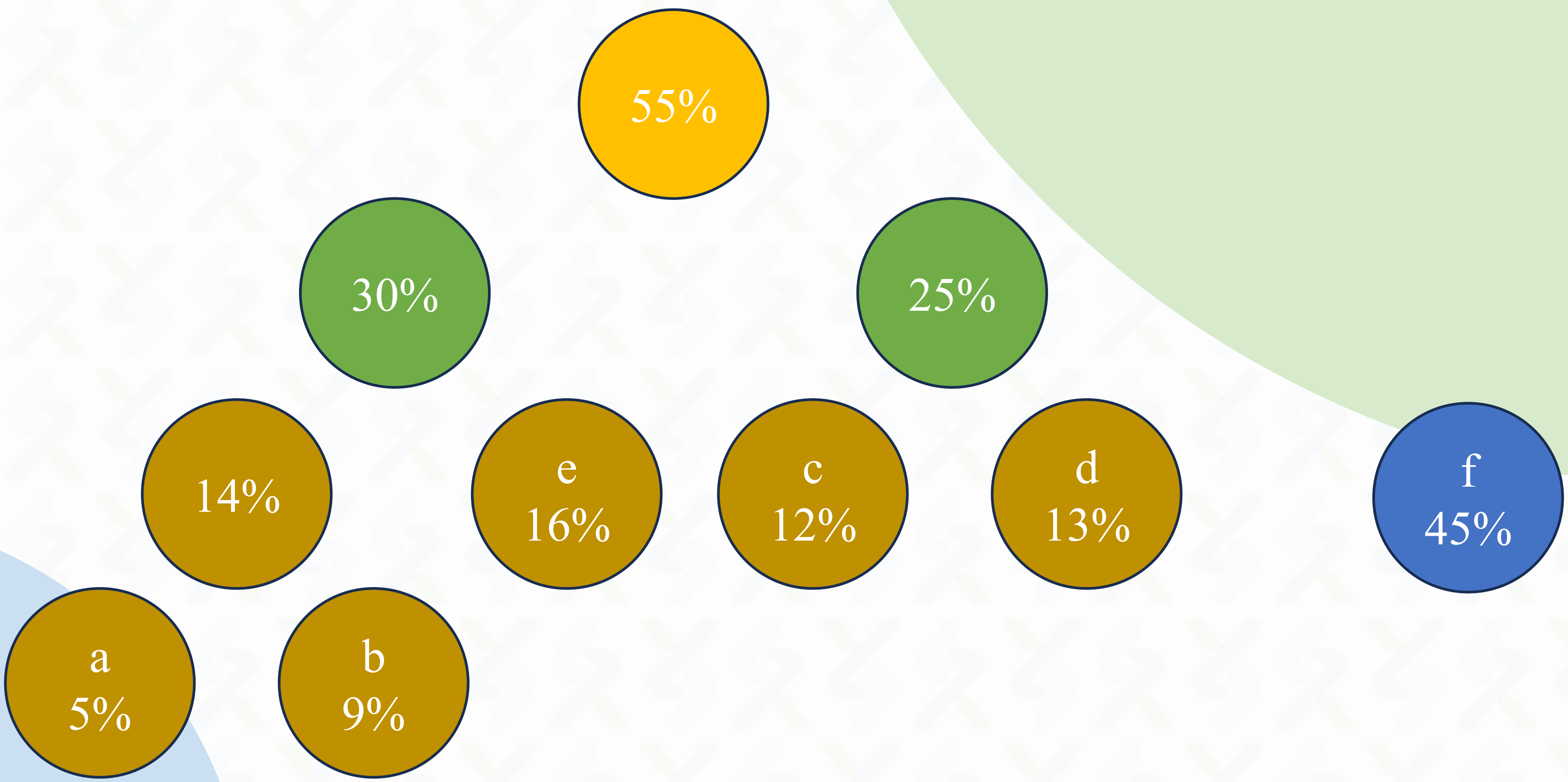
哈夫曼编码



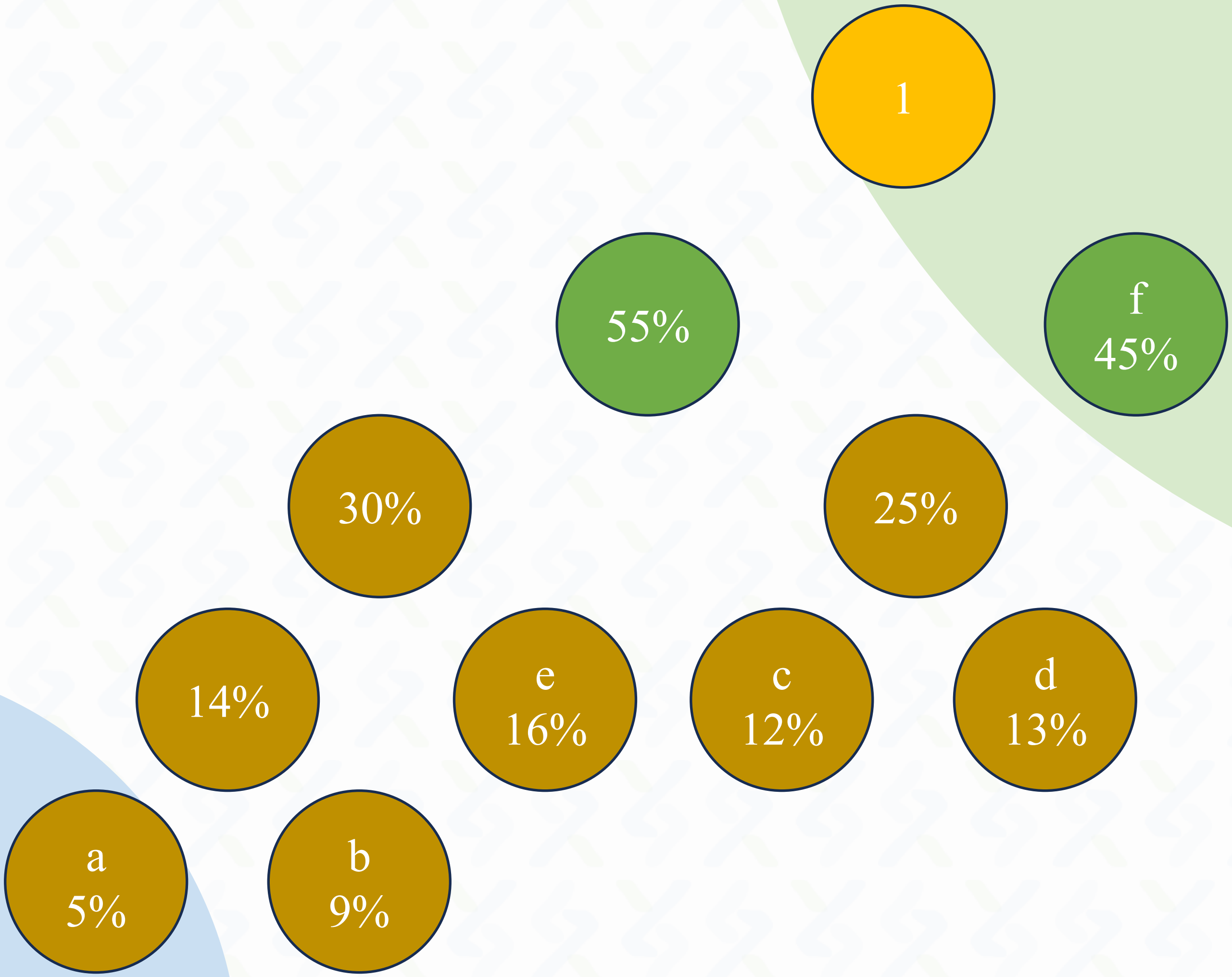
哈夫曼编码



哈夫曼编码



哈夫曼编码



哈夫曼编码

- 由最终构建的哈夫曼树，仅有 A 符合条件

字符	a	b	c	d	e	f
编码长度	4	4	3	3	3	1

哈夫曼树的构建过程

- 将全部节点放入小根堆（优先队列）
- 每次取出最小的两个，合并出新的节点
- 将新的节点插入堆

P2168 荷马史诗

https://www.luogu.com.cn/problem/P2168

- 使用 k 进制串对元素进行编码
- 要求加权长度尽可能短

P2168 荷马史诗

- 哈夫曼编码变形
- 二进制变为 k 进制
- 哈夫曼树由二叉树变为 k 叉树

- 每次从堆中取出权重最小的 k 个节点合并?
- 最后剩余的节点不足 k 个，会导致根结点不到 k 叉，导致结果不是最优
- 用频率为 0 的节点补齐，直到 $(n-1)$ 是 $(k-1)$ 的倍数
- 每次合并取出 k 个节点，插入 1 个节点，减少 $k-1$ 个节点
- 最终需要减少 $n-1$ 个节点

P1966 火柴排队

https://www.luogu.com.cn/problem/P1966

- 要求完成 60%，即 $n \leq 1000$ 的部分
- 给定数列 a, b。交换相邻的火柴，使得 $\sum(a_i - b_i)^2$ 最小。
- 求最小交换次数。

P1966 火柴排队

$$\sum(a_i - b_i)^2 = \sum\left(a_i^2 + b_i^2 - 2a_ib_i\right) = \sum a_i^2 + \sum b_i^2 - 2\sum a_ib_i$$

- 无论如何交换，$\sum a_i^2 + \sum b_i^2$ 的值都是固定的
- 想要距离最小，就需要 $\sum a_ib_i$ 最大

- 结论：如果 $x < y, p < q$，那么 $xp + yq > xq + yp$
- 证明：

$$p < q \to p - q < 0 \to x(p - q) > y(p - q)$$
$$\to xp - xq > yp - yq \to xp + yq > xq + yp$$

- 即，两小两大相乘和大于交叉相乘

P1966 火柴排队

- 可以推论，将 a 中最大的，与 b 中最大的配对；a 中第二大的，与 b 中第二大的配对……
- 根据 a 的大小关系，可以得到需要将 b 中每一个数交换到的位置
- $a = [1,3,4,2]$
- $b = [1,7,2,4]$
- 需要将 b 的次序由 $(1,2,3,4) \to (1,3,4,2)$
- 与 $(1,3,4,2) \to (1,2,3,4)$ 需要交换的次数相同
- 即将 $(1,3,4,2)$ 用冒泡排序为升序，需要交换的次数
- 模拟这样一个排序的过程是 $O(n^2)$ 的，可以通过 60% 的数据

P1966 火柴排队

- 冒泡排序交换次数本质是逆序对数，可以用树状数组维护
- 树状数组时间复杂度 $O(n\log n)$，可以通过 100% 的数据
- 树状数组超出了本课程要求的范围