

# 目录

目录	1
基础与经验	1
常系数齐次线性递推	1
字符串散列	1
字符串匹配 KMP	1
最长回文子串 Manacher	1
贪心	2
区间贪心问题	2
二元贪心	2
动态规划	2
一维	2
二维	2
三维	2
优化	2
背包问题	2
数据结构	2
分数 Fraction	2
高精度整数	2
堆	2
并查集	2
线段树 Segment Tree	2
树状数组 Binary Indexed Tree	2
左偏树 Leftist Tree	2
哈夫曼树	3
图论	3
单源非负最短路 Dijkstra	3
SPFA	3
最小生成树理论基础	3
最小生成树顶点优先 Prim	3
最小生成树边优先 Kruskal	3
网络流	3
最大流 Dinic	3
计算几何	3
向量	3
数论	3
欧拉函数	3
Miller-Rabin 素性测试	4
拓展欧几里得 $ax + by = \gcd(a, b)$	4
单变元模线性方程组 $ax \equiv b \pmod n$	4
语言及黑科技	4
C++	4
字符串格式工具	4
正则表达式 Regit	4
IO 优化	4
时空优化	4
Java	4

## 基础与经验

枚举 折半 搜索 模拟 打表 公式 二分 尺取 构造 离散化 染色

扫描 顺向 逆向 旗帜  
枚举后单调 [扫雷]

二元对 左-1 右正 [WF-Comma]  
环的处理

区间查询  
- 区间和 树状数组 线段树  
- 静态区间最值查询 稀疏表  
- 区间和是否整除模 考察前缀和

中位数定理 [输油管道问题]

自然数列  
- [Hybrid Crystal] 取数列中的元素，如果可以凑出  $[1...sum]$  区间中的任何一个数，向数列加入新数  $x \leq sum+1$ ，可以凑出  $[1...sum+x]$  中的任何一个数。

斐波那契数列 斐波那契数列第  $n$  项

$$\begin{pmatrix} f(n) \\ f(n+1) \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \begin{pmatrix} f(0) \\ f(1) \end{pmatrix} \quad \text{或} \quad \begin{bmatrix} f(n) & f(n+1) \end{bmatrix} = \begin{bmatrix} f(0) & f(1) \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n$$

通项公式  $a_n = \frac{1}{\sqrt{5}} \left[ \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right]$

常系数齐次线性递推

已知  $f_x = a_0 f_{x-1} + a_1 f_{x-2} + \dots + a_{n-1} f_{x-n}$  和  $f_0, f_1, \dots, f_{n-1}$ ，给定  $t$ ，求  $f_t$

$$\text{构造矩阵 } A = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_0 \end{bmatrix}, B = \begin{pmatrix} f_{x-n} \\ f_{x-n+1} \\ \vdots \\ f_{x-2} \\ f_{x-1} \end{pmatrix}$$

字符串散列

简易 利用 `unordered_map<string, int>` 为字符串编号

字符串匹配 KMP

输入模式串  $p$ ，文本串  $s$ ，在  $O(N+M)$  内求解模式串在文本串内的所有匹配位置的下标。注意文本串中匹配的模式串可以重叠。

```
int pre[maxN];
char s[maxN], p[maxN]; // 文本串、模板串

void prepare() {
    fill(pre, pre+maxN, -1);
    for (int i=1, j=-1; p[i]; ++i) {
        while (j>=0 && p[i] != p[j+1]) j = pre[j];
        if (p[i] == p[j+1]) ++j;
        pre[i] = j;
    }
}

void kmp(vector<int> &match) {
    match.clear(); prepare();
    for (int i=0, j=-1; s[i]; ++i) {
        while (j>=0 && s[i] != p[j+1]) j = pre[j];
        if (s[i] == p[j+1]) ++j;
        if (!p[j+1]) { // 匹配成功
            match.push_back(i-j);
        }
    }
}
```

最长回文子串 Manacher

优化暴力匹配

```
// 1-based: scanf("%s", str+1);
int solve() {
    int i = 0, mx = 1; str[0] = '*';
    while (str[i]) {
        int p = i;
        while (str[i+1] == str[i]) ++i;
        int q = i; // q之前不可能有更强的回文中心

        while (str[q-1] == str[p+1]) --q, ++p;
        mx = max(mx, q-p+1);
    }
}
```

```

    ++i;
}
return mx;
}

```

## 贪心

### 区间贪心问题

#### 活动安排问题

若干活动占用左闭右开的时间区间，在活动时间不重叠的情况下选择尽可能多的活动：**右端点越小的区间优先**（为后续区间让出空间）

#### 活动安排问题 2

若干活动占用左闭右开的时间区间，同一个教室安排的活动不能重叠，在使用教室尽可能少的情况下安排所有活动：**考虑活动在时间轴上的厚度**

### 二元贪心

#### 独木舟问题

若干人乘若干独木舟，独木舟有载重限制且只能乘坐两人。安排乘坐方案，使占用的独木舟数量最少：**最轻与最终若能同乘则同乘**（极端化，最优解可转化）

#### 任务执行顺序

若干任务，第  $i$  个任务计算时占用  $R[i]$  空间，完成计算后储存结果占用  $O[i]$  空间 ( $R[i] > O[i]$ )。安排任务，使占用的总空间尽可能少 => 设有整数  $N$ ，第  $i$  个操作时  $N$  减  $a[i]$  加  $b[i]$ ，安排操作顺序，在操作中不能出现负数的情况下  $N$  尽可能小： **$b[i]$  非递增排序** 任何可行方案不优于按  $b[i]$  非递增排序时的方案（最优解可转化）

## 动态规划

#### 树塔 矩阵取数 双向矩阵取数

$$dp[step + 1][x1][x2] = \max\{dp[step][x1'][x2']\} + v[...]$$

**最大子段和** 最大子矩阵和 循环数组最大子段和（总和 - 最小子段和）

#### 正整数分组

$$dp[i][j] = dp[i - 1][j - a[i]] \text{ or } dp[i - 1][j + a[i]]$$

或背包问题，背包容量  $sum/2$

#### 子序列的个数

$$dp[i] = \begin{cases} dp[i - 1] * 2 & \text{若 } a[i] \text{ 未出现} \\ dp[i - 1] * 2 - dp[j - 1] & \text{若 } a[i] \text{ 最近在 } j \text{ 位置出现} \end{cases}$$

#### 最长公共子序列 LCS

$$dp[i][j] = \begin{cases} dp[i - 1][j - 1] + 1 & \text{若 } x[i] = y[j] \\ \max\{dp[i][j - 1], dp[i - 1][j]\} & \text{若 } x[i] \neq y[j] \end{cases}$$

$$\text{编辑距离 } dp[i][j] = \min \begin{cases} dp[i - 1][j - 1] + \text{same}(i, j) & dp[0][0] = 0 \\ dp[i - 1][j] + 1 & dp[i][0] = i \\ dp[i][j - 1] + 1 & dp[0][j] = j \end{cases}$$

**最长单增子序列 LIS**  $dp[len] = \min\{tail\}$

### 一维

### 二维

**石子归并**  $dp[i][j] = \max_{i \leq k \leq j} \{dp[i][k] + dp[k][j]\} + \sum_{i \leq p \leq j} w[p]$

### 三维

[CSA-Two Rows]  $dp[r][c][turn]$

Turn 选择  $(r, c)$  的玩家  
Dp 从  $(r, c)$  到终点的总花费

### 优化

#### 改进状态表示

#### 四边形不等式

#### 斜率优化

### 背包问题

#### 01 背包问题

#### 多重背包问题

## 数据结构

### 分数 Fraction

Numerator 分子 Denominator 分母

构造函数接受分子 num 和分母 den 作为参数，确保符号在分子上集中，并且断言分母不为零，然后进行约分。

### 高精度整数

// 正在整理

### 堆

### 并查集

**经验** [圆环出列]

### 线段树 Segment Tree

// 正在整理

### 树状数组 Binary Indexed Tree

#### 区间求和单点更新

// todo

#### 区间求和区间更新

// todo

### 左偏树 Leftist Tree

编号为 0 的节点表示空节点

```

struct LeftTree
{
    const static int MXN = 100100;
    int tot = 0;
    int l[MXN], r[MXN], v[MXN], d[MXN];

    // 初始化为x的元素
    int init(int x)
    {
        tot++;
        v[tot] = x;
        l[tot] = r[tot] = d[tot] = 0;
        return tot;
    }
}

```

```

// 合并堆顶编号为x, y的堆
int merge(int x, int y)
{
    if (!x) return y;
    if (!y) return x;
    if (v[x] < v[y])
        swap(x, y);
    r[x] = merge(r[x], y);
    if (d[l[x]] < d[r[x]])
        swap(l[x], r[x]);
    d[x] = d[r[x]] + 1;
    return x;
}

// 向堆顶编号为x的堆中插入值为v的元素
int insert(int x, int v)
{
    return merge(x, init(v));
}

// 取编号为x的堆的堆顶元素
int top(int x)
{
    return v[x];
}

// 弹出编号为x的堆的堆顶元素, 返回新堆顶的编号
int pop(int x)
{
    return merge(l[x], r[x]);
}
};

```

## 哈夫曼树

以频率为节点权值维护节点队列。合并队列中权值最小的两个节点, 将合并的新节点放入队列中, 重复步骤, 直至队列中只存在一个节点。

## 图论

### 匹配

#### 二分图最大匹配匈牙利算法

```

vector<int> g[maxN]; // g[x]: 与x点相连的右侧的点
int from[maxN], tot; // from[x]: 与右侧点x匹配的左侧的点,
tot: 最大匹配数
bool use[maxN];

bool match(int x) // x: 待匹配的一个左侧的点
{
    for (unsigned i=0; i<g[x].size(); ++i) if
(!use[g[x][i]])
    {
        use[g[x][i]] = true;
        if (from[g[x][i]] == -1 ||
match(from[g[x][i]])) {
            from[g[x][i]] = x; return true;
        }
    }
    return false;
}

int hungary(int n) // n: 左侧点的个数
{
    memset(from, -1, sizeof(from)); tot = 0;
    for (int i=1; i<=n; ++i) {
        memset(use, 0, sizeof(use));
        if (match(i)) ++tot;
    }
    return tot;
}

```

#### 单源非负最短路 Dijkstra

#### 升级 堆优化

#### SPFA

队列非空时, 队头出列; 松弛队头的边, 已松弛且不在队列中的顶点入队。入队超过 n 次则图中存在负环。

## 最小生成树理论基础

环定理 切分定理 最小权值边定理

### 最小生成树顶点优先 Prim

类似于 Dijkstra, 但维护的距离是顶点到已松弛顶点的集合的距离。

### 最小生成树边优先 Kruskal

维护顶点的集合  $S=V_0$ ,  $T=(V-S)$ 。边升序遍历, 对于每一条边  $(s, t)$ , 若  $s \in S$ ,  $t \in T$ , 则将边加入树中, 并将  $t$  并入  $S$ ;  $T$  中没有顶点时, 算法结束, 所得树为最小生成树。

## 网络流

### 最大流 Dinic

## 计算几何

**海伦公式**  $A = \sqrt{p\sqrt{p-a}\sqrt{p-b}\sqrt{p-c}}$   $p = \frac{a+b+c}{2}$

### 向量

点乘 叉乘  
两点共线的判定  
线段相交的判定

## 数论

**二项式定理**  $(x+a)^n = \sum_{k=0}^n \binom{n}{k} x^k a^{n-k}$

**组合数**  $C_n^k = \binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$   
 $C(n, m) = C(n-1, m) + C(n-1, m-1)$

**错排公式**  $D_n = (n-1)(D_{n-1} + D_{n-2})$

#### 费马小定理

若  $p$  为质数,  $a^p \equiv a \pmod{p}$   
若  $a$  不是  $p$  的倍数,  $a^{p-1} \equiv 1 \pmod{p}$   
引理,  $a^p \equiv 1 \pmod{p} \rightarrow a \equiv \pm 1 \pmod{p}$

**威尔逊定理**  $(p-1)! \equiv -1 \pmod{p}$

**自然数 N 因子个数  $f(n)$**  考虑分解质因数

#### 猜想

**Bertrand 猜想** 对于任意  $n > 3$ , 存在  $n < p < 2n$ , 其中  $p$  为质数

**质数间隔** 在  $1e13$  范围内, 相邻素数的最大间隔为 777

## 欧拉函数

## Miller-Rabin 素性测试

### 拓展欧几里得 $ax + by = \gcd(a, b)$

```
LL gcd(LL a, LL b, LL &x, LL &y)
{
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    else {
        LL r = gcd(b, a%b, y, x);
        y -= (a/b)*x;
        return r;
    }
}
```

$$x = x_0 + \frac{b}{\gcd(a, b)} \cdot t, \quad y = y_0 - \frac{a}{\gcd(a, b)} \cdot t$$

### 单变元模线性方程组 $ax \equiv b \pmod{n}$

相当于求解  $ax + ny = b$ , 当且仅当  $\gcd(a, n) | n$  时有解, 且有  $\gcd(a, n)$  个解。通解:

$$x_i = \left[ x_0 + i \cdot \left( \frac{n}{\gcd(a, n)} \right) \right] \pmod{n}, \quad i = 0, 1, 2, \dots, \gcd(a, n) - 1$$

```
vector<LL> line_mod_equation(LL a, LL b, LL n)
{
    LL x, y;
    LL d = gcd(a, n, x, y);

    vector<LL> ans;
    if (b%d == 0) {
        x %= n; x += n; x %= n;
        ans.push_back(x*(b/d)%(n/d));
        for (LL i=1; i<d; ++i)
            ans.push_back((ans[0]+i*(n/d))%n);
    }
    return ans;
}
```

## 语言及黑科技

### C++

```
set_intersection()
set_union()
set_difference()
```

### 字符串格式工具

```
string stoi stol stoll stod to string
*char atoi atol atof
```

### 正则表达式 Regit

### IO 优化

```
template<typename T = int>
inline T read() {
    T val = 0, sign = 1; char ch;
    for (ch = getchar(); ch < '0' || ch > '9'; ch = getchar())
        if (ch == '-') sign = -1;
    for (; ch >= '0' && ch <= '9'; ch = getchar())
        val = val * 10 + ch - '0';
    return sign * val;
}
```

## 时空优化

展开循环: 牺牲程序的尺寸加快程序的执行速度

```
#pragma GCC optimize("unroll-loops")
```

### Java

```
// BigInteger and BigDecimal
import java.math.*;
import java.util.Scanner;
```

add multiply subtract divide