

# 目录

|                          |   |
|--------------------------|---|
| 目录                       | 1 |
| 基础                       | 1 |
| 常系数齐次线性递推                | 1 |
| 贪心                       | 1 |
| 区间贪心问题                   | 1 |
| 二元贪心                     | 1 |
| 动态规划                     | 1 |
| 背包问题                     | 1 |
| 数据结构                     | 1 |
| 分数 Fraction              | 1 |
| 高精度整数                    | 1 |
| 线段树 Segment Tree         | 1 |
| 树状数组 Binary Indexed Tree | 1 |
| 左偏树 Leftist Tree         | 2 |
| 哈夫曼树                     | 2 |
| 图论                       | 2 |
| 单源非负最短路 Dijkstra         | 2 |
| 最小生成树 Prim               | 2 |
| 最小生成树 Kruskal            | 2 |
| 网络流                      | 2 |
| 计算几何                     | 2 |
| 数论                       | 2 |
| 拓展欧几里得                   | 2 |
| 单变元模线性方程组                | 2 |
| 黑科技                      | 2 |
| IO 优化                    | 2 |
| 时空优化                     | 3 |

# 基础

枚举 折半 搜索 模拟 打表 公式 二分 尺取 构造 离散化

斐波那契数列第 n 项

$$\begin{bmatrix} f(n+2) & f(n+1) \\ f(n+1) & f(n) \end{bmatrix} = \begin{bmatrix} f(2) & f(1) \\ f(1) & f(0) \end{bmatrix}^n \text{ 或 } \begin{pmatrix} f(n) \\ f(n+1) \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \begin{pmatrix} f(0) \\ f(1) \end{pmatrix}$$

常系数齐次线性递推

# 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

$$dp[step+1][x1][x2] = \max\{dp[step][x1'][x2']\} + v[x1][y1] + v[x2][y2]$$

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  $dp[len] = \min\{tail\}$

## 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

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

## 单源非负最短路 Dijkstra

升级: 堆优化

## SPFA

```
//todo
```

## 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

# 计算几何

## 向量

点乘 叉乘

## 线段相交判定

# 数论

## 拓展欧几里得

```
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;
    }
}
```

## 单变元模线性方程组

```
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;
}
```

# 黑科技

## 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")
```