OI 笔记

头文件

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef __int128 lll;
//#define int long long
#define Inf 0x3f3f3f3f
#define INF 0x3f3f3f3f3f3f3f3f
```

荣耀笔记历史部分

2023/10/17 区间dp; 按位运算;
2023/10/18 暴力枚举; 翻倍记录线段 搜索;
2023/10/19 简单贪心;
2023/10/20 树(重)链剖分LCA dfs序 set 卡常(快读 左孩子右兄弟表示法);
2023/10/21 二维差分 二维前缀和 最大连续子序列dp; 计数dp(图计数-定一移二-连锁递推) 组合数学 or 压缩打表;
2023/10/22 置换 数论 组合 计数dp 快速幂;
2023/10/24 简单贪心; 构造 欧拉回路 dfs 链式前向星;
2023/10/25 计数dp(块状态 逐个插入法); 线性筛 欧拉函数;
2023/10/27 区间dp 前缀和;
2023/10/28 数位dp 记忆化搜索;
2023/10/29 简单dp 枚举 bitset; 可持久化Trie树 链表;
2023/10/30 树链剖分+线段树 链式前向星;
2023/10/31 单调队列; 模拟;
2023/11/1 递归 局部记忆化 or (递推)矩阵快速幂;
2023/11/2 博弈论(绝对胜算) 对称性; 简单贪心; 威尔逊定理 阶乘分解 化归;
2023/11/3 费马小定理 威尔逊定理 卢卡斯定理 快速幂;
2023/11/4 卢卡斯定理模板题;
2023/11/5 线段树的dfs序 取模细节;
2023/11/6 分块暴力 分块O(1); Dijkstra算法(堆优化);
2023/11/7 计算几何(暴力) 数据去重; 数位dfs; 简单递推dp; 桶; 线段树(懒标记);
2023/11/8 枚举 度; 链表 信息压缩; 树链剖分 树上前缀和;
2023/11/9 计数 贡献归功; 并查集 路径压缩;

2023/11/10 充分统计;
2023/11/11 背包dp; 逆序对 线段树统计; 随机生成数据; 填空: 模拟 暴力枚举 bfs最短路记录;
2023/11/12 模拟 二叉树的遍历; 计数dp; 巧用数据大小限制 or 数列线性性质 桶计数; 数列求和 方程求根 or 二分法求根;

2023/11/13

#全错位排列数
由容斥原理:

$$D_n = C_n^0 A_n^n - C_n^1 A_{n-1}^{n-1} + C_n^2 A_{n-2}^{n-2} - C_n^3 A_{n-3}^{n-3} + \ldots + (-1)^n C_n^n A_0^0$$

即:

$$D_n = n! \left[\frac{1}{0!} - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots + \frac{(-1)^n}{n!} \right]$$

#康托展开

$$p = \sum_{i=1}^{n} \lambda_i \, (n-i)!$$

p — 当前全排列的序数（$0 \leqslant p < n!$）

λ_i — i位置后面有多少个数字比它小（逆序数）

P8687 [蓝桥杯 2019 省 A] 糖果
#状压dp
状压打表:

```
for (int i = 0; i < n; ++i) {
        for (int j = 0; j < k; ++j) {
                int ai;
                cin >> ai;
                a[i] |= 1 << (ai - 1);
        }
}
```

状压转移:

```
dp[0] = 0;
for (int i = 0; i < (1 << m); ++i) {
    if (dp[i] == Inf)
        continue;
    for (int j = 0; j < n; ++j) {
        int nxt = i | a[j];
        dp[nxt] = min(dp[nxt], dp[i] + 1);
```

```
    }
}
```

2023/11/14

#最小生成树

结点定义：

```
struct P {
        int x, w;
        bool operator<(const P &p) const { return w > p.w; }
};
```

堆优化：

```
priority_queue<P> q;
q.push(P{1, 0});
int ans = 0;
while (!q.empty()) {
        int now = q.top().x;
        int mny = q.top().w;
        q.pop();
        if (vis[now]) continue;
        vis[now] = true;
        ans += mny;
        for (int i = 1; i ≤ n; ++i) {
            if (vis[i]) continue;
            q.push(P{i, cost(now, i)});
        }
}
```

#约数

复杂度$O(\sqrt{n})$：

$$找n的约数：从1\sim\sqrt{n}中的i，检查i（同时就有\frac{n}{i}）$$

2023/11/15

P8666 [蓝桥杯 2018 省 A] 三体攻击

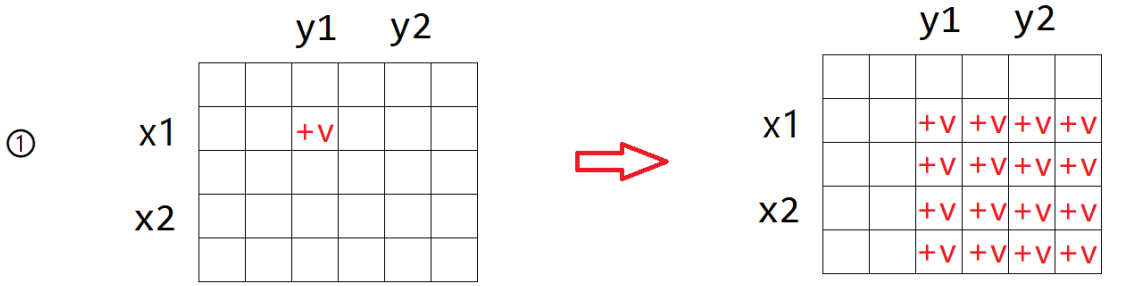
#拓展线性表

```
struct DATA {
        int a[N];
        inline int &operator()(int i, int j, int k) {
                return a[(i * B + j) * C + k];
        }
        inline void clear() {
                memset(a, 0, sizeof(a));
        }
} H, dif, sum;
```

#三维差分

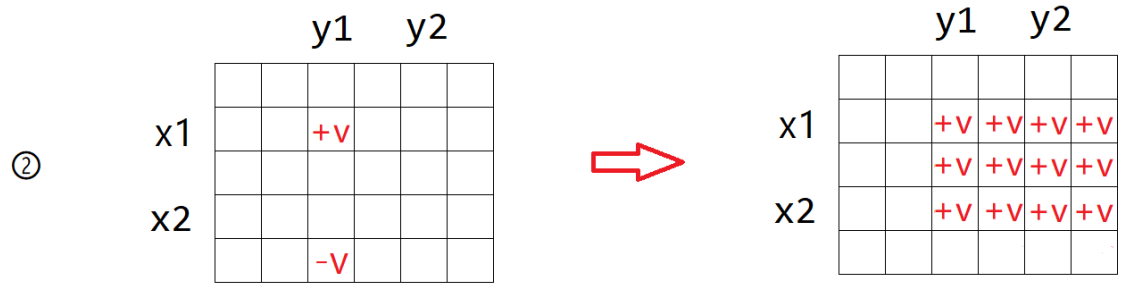
不管多少维，只要类比二维，就可以推广出原理：

二维差分　　二维差分的前缀和



d[x1][y1] += v

二维差分　　二维差分的前缀和



1. d[x1][y1] += v
2. d[x2 + 1][y1] -= v

记忆：全部顶点，按边角位置，符号交错，进行求和

```
inline void attk(int p) {
        const ATTACK &o = atk[p];
        dif(o.al, o.bl, o.cl) += o.h;
        dif(o.ar + 1, o.bl, o.cl) -= o.h;
        dif(o.al, o.br + 1, o.cl) -= o.h;
        dif(o.al, o.bl, o.cr + 1) -= o.h;
        dif(o.ar + 1, o.br + 1, o.cl) += o.h;
```

```
        dif(o.ar + 1, o.bl, o.cr + 1) += o.h;
        dif(o.al, o.br + 1, o.cr + 1) += o.h;
        dif(o.ar + 1, o.br + 1, o.cr + 1) -= o.h;
}

inline int qry(int i, int j, int k) {
        return sum(i, j, k) = dif(i, j, k) + sum(i - 1, j, k) + sum(i, j -
1, k) + sum(i, j, k - 1) - sum(i - 1, j - 1, k) - sum(i - 1, j, k - 1) -
sum(i, j - 1, k - 1) + sum(i - 1, j - 1, k - 1);
}
```

#二分答案

```
int l = 0, r = m;
while (r - l > 1) {
        int mid = l + r >> 1;
    if (check(mid))
            r = mid;
    else
            l = mid;
}
cout << r;
```

2023/11/16

#bitset

支持位运算（请注意位移运算的方向）

```
bitset<N> b;
b <<= 1; // 相当于bool数组右移
b ^= b; // 异或
```

#找规律

P8763 [蓝桥杯 2021 国 ABC] 异或变换

t范围太大，所以考虑会出现循环。模拟并观察，或者无脑暴力检查是否循环并计数，最后取模再跑一遍，有 `bitset` 优化，问题不大：

```
ll cnt = 0;
while (t--) {
        b ^= b << 1;
        ++cnt;
        if (b == b0) break;
}
if (t != -1) {
```

```
    t %= cnt;
        while (t--) b ^= b << 1;
}
```

2023/11/17

#gcd

cpp内置库函数：`int __gcd(int a, int b)`，需要`#include <algorithm>`
或者自己写：

```
inline int gcd(int a, int b) {
        if (b) while ((a %= b) && (b %= a));
        return a + b;
}
```

#完全背包

一定物品，不限数量，能组合成的所有价值的分布（价值上限为S），用dp桶：

```
for (int i = 1; i < S; ++i) {
    if (!dp[i])
      continue;
    for (int j = 1; j ≤ n; ++j) {
            if (i + a[j] ≥ S) break;
            dp[i + a[j]] = true;
    }
}
```

2023/11/18

#桶计数

```
for (int i = 1; i ≤ n; ++i) {
    int a;
    cin >> a;
    cnt[a]++;
}
```

2023/11/19

P8746 [蓝桥杯 2021 省 A] 分果果

#记忆化搜索

有两个优化目标mi和ma，可选择每次固定mi（mi的取值范围小），优化ma。

```
int maxmi = 2 * sum[n] / m, ans = Inf;
for (mi = maxmi; mi ≥ 1; --mi) {
    memset(stf, 0x3f, sizeof(stf));
    int ma = f();
    ans = min(ans, ma - mi);
    if (ans == 0) break;
}
cout << ans;
```

难以写出转移方程则无脑进行记忆化搜索，要注意本原部分和转移部分的选取，大大发挥记忆化的功能：

```
inline int f(int end1 = 0, int end2 = 0, const int now = 1) {
    ... // 边界计算
    int re = Inf - 1;
    // 转移
    const int maxi = min(n, end2 + 1);
    for (int i = maxi; i ≥ end1 + 2; --i) {
        re = min(re, f(i - 1, end2, now));
    }
    // 本原
    for (int i = end1 + 1, j = end2; j ≤ n; ++j) {  // j从end2开始由性质
        int s = S(i, j);
        if (s < mi) continue;
        if (s ≥ re) break;
        re = min(re, max(s, f(j, end2, now + 1)));
    }
    return stf[end1][end2][now] = re;
}
```

2023/11/20

#贝叶斯公式

$$P(A_i|B)=\frac{P(A_iB)}{P(B)}=\frac{P(B|A_i)P(A_i)}{\sum_j P(B|A_j)P(A_j)}$$

注意考察清楚$P(B|A_j)$。

2023/11/21

#线性筛

```
int ps[N], cur = 0;
inline void init()
{
    for (int i = 2; i < N; ++i) {
        if (!vis[i])
                ps[++cur] = i;
        for (int j = 1; j ≤ cur && i * ps[j] < N; ++j) {
            vis[i * ps[j]] = true;
            if (i % ps[j] == 0)
                break;
        }
    }
}
```

#约数个数

正整数N的质因数分解：

$$N = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n}$$

则它的约数个数为：

$$(\alpha_1 + 1)(\alpha_2 + 1) \dots (\alpha_n + 1)$$

#本质递增子序列计数dp

```
for (int i = 1; i ≤ n; ++i) {
    dp[i] = 1;
    for (int j = 1; j < i; ++j) {
        if (s[j] < s[i])
            dp[i] += dp[j];
        else if (s[j] == s[i])
            dp[i] -= dp[j];
    }
}
ll ans = 0;
for (int i = 1; i ≤ n; ++i)
    ans += dp[i];
```

2023/11/22

P8776 [蓝桥杯 2022 省 A] 最长不下降子序列

#最长不下降子序列

tmp[]维护了一个不下降的数据堆（并不是按照数组中的原序），每扫描一个数组元素，则会对tmp[]进行相应的更改，并判断最大子序列长度len是否加1……

```
int tmp[N], len;
int  main(){
        // ...
        tmp[0] = a[1]; len = 1;
        L[1] = 1;
    for (int i = 2; i ≤ n; ++i) {
        int t = upper_bound(tmp, tmp + len, a[i]) - tmp;
        if (t == len)
            tmp[len++] = a[i];
        else
            tmp[t] = a[i];
        L[i] = t + 1;
    }
    // for (int i = 1; i ≤ n; ++i) cout << L[i] << ' ';
    // ...
}
```

#树状数组

这里的功能是维护前缀最大值：

```
int bt[A];

inline void add(int p, int k){
    for (; p < A; p += p & -p)
        bt[p] = max(bt[p], k);
}

inline int qry(int p){
    int re = 0;
    for (; p; p -= p & -p)
        re = max(re, bt[p]);
    return re;
}
```

可作为桶，动态加点，加快区间查询速度：

```
for (int i = k + 2; i ≤ n; ++i) {
    add(a[i - k - 1], L[i - k - 1]);
    ans = max(ans, qry(a[i]) + k + R[i]);
}
```

2023/11/23

#数位dp

P8766 [蓝桥杯 2021 国 AB] 异或三角

debug半天，数位dp状态真多，我是这样记录的：

```
ll dfs(int now = 30, int flag = 0, bool full = true, bool have = false)
{ ... }
```

题解大佬的办法，用一个二进制数压缩前三个限制的当前状态，记忆化搜索实现dp，膜：

```
ll dfs(int cur, int state, bool fulc) {
    if (cur < 0)
        return state == 7;  // 同时满足所有限制
    if (~f[cur][state][fulc])
        return f[cur][state][fulc];
    int dig = fulc ? a[cur] : 1;
    i64 res = 0;
    for (int k = 0; k ≤ dig; k++) {
        if (k == 0) {
            int b = state & 1, c = state >> 1 & 1;
            res += dfs(cur - 1, state, fulc && (k == dig));  // (a_i, b_i,
c_i) = (0, 0, 0)
            if (b == 1 && c == 1)
                res += dfs(cur - 1, state | 4, fulc && (k == dig));  //
(a_i, b_i, c_i) = (0, 1, 1)
        }
        if (k == 1) {
            res += dfs(cur - 1, state | 1, fulc && (k == dig));  // (a_i,
b_i, c_i) = (1, 0, 1)
            res += dfs(cur - 1, state | 2, fulc && (k == dig));  // (a_i,
b_i, c_i) = (1, 1, 0)
        }
    }
    return f[cur][state][fulc] = res;
}
```

2023/11/24

#找规律

P8700 [蓝桥杯 2019 国 B] 解谜游戏

分类：证明同类元素可以按照规则来交换以达到任意目标，而不同类的元素不可能进行任何交换。所以对每个类中的元素进行计数并判断即可。

2023/11/25

P8769 [蓝桥杯 2021 国 C] 巧克力

#贪心

一道有意思的贪心题。

有一个很容易想到但是有误的贪心：从第1天开始，每次选择单价最低的购买，直到第x天。

但如果有一些单价较低且保质期极短的商品，和一些单价最低但保质期较长的商品，这个贪心就不会选择到单价较低的商品。

如果我们使时间逆流，就不会出现这样的问题，即从第x天开始，选当前单价最小的即可。

维护当前的最小值，优先队列和 set 都可以。

时间复杂度：$O(x \log n)$

#桶计数

配合优先队列，不要直接对优先队列中的元素进行修改。

```
while (i > 0 && !pq.empty()) {
    const P p = pq.top();
    if (--cnt[p.id] == 0) pq.pop();
    ans += p.a;
    --i;
    for (; now > 0 && ps[now].b ≥ i; --now) {
        cnt[ps[now].id] = ps[now].c;
        pq.push(ps[now]);
    }
}
```

2023/11/16

P8709 [蓝桥杯 2020 省 A1] 超级胶水

#Ad-hoc

结果与选择无关，题意迷惑人

#前缀和

注意开long long

2023/11/17

P8650 [蓝桥杯 2017 省 A] 正则问题

#栈

非递归求解正则问题：

```
#include <bits/stdc++.h>
// #define int long long
```

```cpp
using namespace std;
typedef long long ll;
typedef __int128 lll;
const int Inf = 0x3f3f3f3f;
const ll INF  = 0x3f3f3f3f3f3f3f3f;

stack<int> s;

inline void add0(int x)
{
    if (!s.empty() && s.top() ≥ 0) {
        int t = s.top();
        s.pop();
        s.push(t + x);
    }
    else {
        s.push(x);
    }
}

inline void add(char c)
{
    switch (c) {
        case 'x': {
            add0(1);
            break;
        }
        case '(': {
            s.push(-1);
            s.push(0);
            break;
        }
        case ')': {
            int a = s.top();
            s.pop();
            while (s.top() == -2) {
                s.pop();
                int b = s.top();
                s.pop();
                a = max(a, b);
            }
            s.pop();
            add0(a);
            break;
        }
        case '|': {
            s.push(-2);
            break;
```

```
        }
    }
}

signed main()
{
    add('(');
    char c;
    while (scanf("%c", &c) != EOF) {
        if (c == '\n')
            break;
        add(c);
        // cout<<s.top()<<'\n';
    }
    add(')');
    cout << s.top();
    return 0;
}
```

递归的话更简单，如题解：

```
#include <bits/stdc++.h>
using namespace std;
int dfs()
{
    char c;
    int s = 0;
    while (cin >> c) {
        if (c == 'x')
            s++;  // 长度加1
        else if (c == '(')
            s += dfs();  // 调用函数计算子正则表达式的长度。
        else if (c == ')')
            return s;  // 返回当前长度
        else
            return max(s, dfs());  // 调用函数，返回返回值与当前长度中较大的那一个
    }
    return s;  // 返回当前长度
}
int main()
{
    //  freopen(".in","r",stdin);
    //  freopen(".out","w",stdout);
    printf("%d", dfs());
    return 0;
}
```

2024/1/3

#点权和最大的子树
树上dp，

$$dp[u] = w[u] + \sum_{u \to v} dp[v](dp[v] > 0)$$

$$ans = \max_{i} dp[i]$$

#康托展开

2024/1/19

#记忆化搜索
也要注意初始条件和边界条件！！！
初始条件：

```
if (x == 1 && y == 1 && cnt == 0) return 1; // 函数中判断
...
dp[1][1][1][a[1][1]] = 1; // 函数外赋值
```

边界条件：

```
if (x == 0 || y == 0) return 0;
```

2024/1/23

P8767 [蓝桥杯 2021 国 A] 冰山
#map
稀疏桶计数
平移标记
上分位：`lower_bound()`；下分位：`upper_bound()-1`
#函数式编程 #模运算
我的代码：

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

const ll P = 998244353, N = 100005, K = 1000000009;
ll n, m, k;
```

```cpp
map<ll, ll> mp;
ll S = 0, X = 0;

inline void add(ll y, ll num = 1)
{
    if (num == 0)
        return;
    mp[y - X] = (mp[y - X] + num) % P;
    n = (n + num) % P;
    S = (S + y * num) % P;
}

inline void del(map<ll, ll>::iterator it)
{
    n = (n - it->second) % P;
    S = (S - (it->first + X) * it->second) % P;
    mp.erase(it);
}

inline void inc(ll x)
{
    X += x;
    S = (S + n * x) % P;
    map<ll, ll>::iterator it = mp.lower_bound(k - X);
    ll cnt = 0;
    while (it != mp.end()) {
        cnt += it->second;
        add(1, (it->first - k + X) * it->second);
        del(it++);
    }
    n %= P;
    add(k, cnt);
}

inline void dec(ll x)
{
    X -= x;
    S = (S - n * x) % P;
    map<ll, ll>::iterator it = mp.upper_bound(-X), tmp = it;
    while (it != mp.begin()) {
        --tmp;
        del(tmp);
        tmp = it;
    }
    n %= P;
    S %= P;
}
```

```
signed main()
{
    cin >> n >> m >> k;
    for (int i = 0; i < n; ++i) {
        ll a;
        cin >> a;
        mp[a]++;
        S += a;
    }
    while (m--) {
        ll x, y;
        cin >> x >> y;
        if (x > 0)
            inc(x);
        else if (x < 0)
            dec(-x);
        if (y > 0)
            add(y);

        cout << (S % P + P) % P << '\n';
    }
}
```

2024/1/30

#树的直径

法1：两次dfs，从最远到最远（不适用于负边权）

法2：树上dp：

`dp[u]`表示从子树根`u`出发的最长路径，

```
dp[u] = max(dp[u], dp[v] + w[u][v]);
```

`d`为不断更新的树的直径，需要在上述代码之前进行：

```
d = max(d, dp[u] + dp[v] + w[u][v]);
```

实例：（来自OI-wiki）

```
void dfs(int u, int fa) {
        for (int v : E[u]) {
                if (v == fa) continue;
            dfs(v, u);
            d = max(d, dp[u] + dp[v] + w[u][v]);
            dp[u] = max(dp[u], dp[v] + w[u][v]);
```

```
        }
}
```

2024/2/1

#最长公共子序列

$f(m,n)$表示$s1[1..m]$与$s2[1..n]$的最长公共子序列。

转移方程：

$$f(m,n)=\max\{f(m-1,n-1)+(s1[m]==s2[n]),f(m-1,n),f(m,n-1))\}$$

2024/2/26

#状态压缩

一个串`string s`代表一个状态。

#记忆化搜索

记忆存储：`map<string, int> stf`

记忆查询：`if (stf.find(s) != stf.end()) return stf[s];`

函数传递：`int f(string s)`

2024/2/27

#分层图

普通图：1层，编号1~n

分层图：m层，编号1(0)~n(m-1)，一个点与一个编号一一对应

#Dijkstra

所以可以在分层图中使用Dijkstra算法：（一种状态的点为一个点）

```
struct point
{
    int v, d, t;
    bool operator<(const point& p) const
    {
        return d == p.d ? t > p.t : d > p.d;
    }
};

int dj1()
{
    bool vis[N];  // 1层图
```

```
    memset(vis, 0, sizeof(vis));
    priority_queue<point> pq;
    pq.push(point{1, 0, 0});
    while (!pq.empty()) {
        int u = pq.top().v, d = pq.top().d;  // cout<<d<<'\n';
        pq.pop();
        if (vis[u])
            continue;
        vis[u] = true;
        if (u == n)
            return d;
        for (int i = hed[u]; i; i = nxt[i]) {
            if (vis[to[i]] || typ[i])
                continue;
            pq.push(point{to[i], d + wgt[i], 0});  // 后继点
        }
    }
    return -1;
}

int dj2()
{
    int re = Inf;
    int cnt = 0;
    int vis[N][3];  // 3层图
    memset(vis, 0, sizeof(vis));
    priority_queue<point> pq;
    pq.push(point{1, 0, 0});
    while (!pq.empty()) {
        int u = pq.top().v, d = pq.top().d, t = pq.top().t;
        pq.pop();
        if (vis[u][t])
            continue;
        vis[u][t] = true;
        if (u == n) {
            re = min(re, d);
            if (++cnt == 3)
                return re;
            continue;
        }
        for (int i = hed[u]; i; i = nxt[i]) {
            if (t + typ[i] > 2 || vis[to[i]][t + typ[i]])
                continue;
            pq.push(point{to[i], d + wgt[i], t + typ[i]});  // 后继点
        }
    }
    return re;
}
```

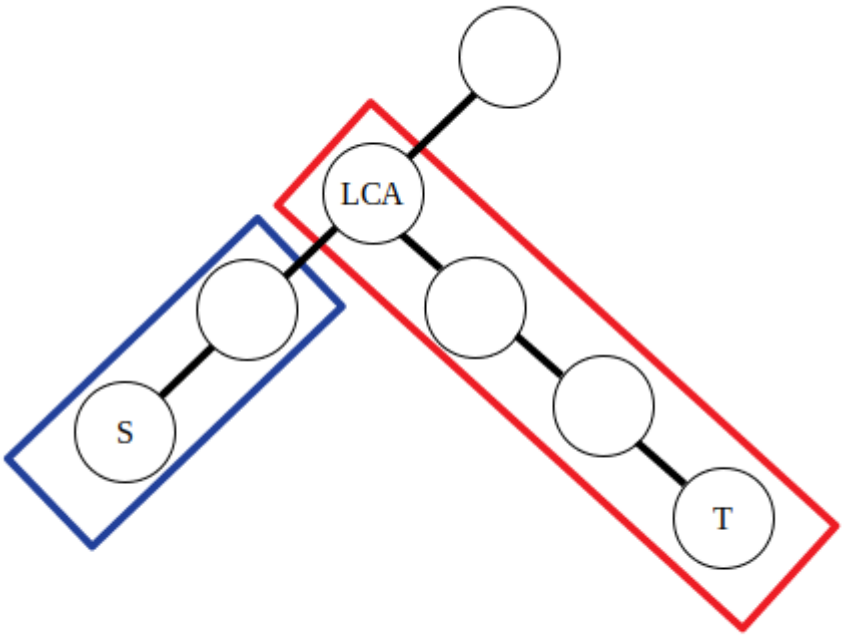
2024/2/28

#最长上升子序列
注意要用 `lower_bound`
构造序列需要用 `pre[N]` 数组
注意区分 `pos` 参数与 `rnk` 参数
前者用于预排序：`pair<item, int>`，其中 `item` 为元素，`int` 为 `pos` 参数，最终可得到 `rnk[N]`
后者用于进行最大上升子序列操作，参考 #最长不下降子序列 ，并同时构造 `pre[N]` 数组，`pre[i]=tmp[cur-1]`，其中 `i` 和 `tmp[cur-1]` 均为 `pos` 参数

2024/2/29

#重链剖分
第一次dfs求：`int hson[N], siz[N], dep[N], fa[N]`
第二次dfs求：`int top[N], dfn[N], rnk[N]`，全局辅助变量 `now = 0`
#树上差分
相比于树上前缀和，计算方向相反，为从树叶到树根的差分
#点差分
要使s到t的全部点权+k

```
dif[s] += k, dif[t] += k;
dif[lca] -= k, dif[fa[lca]] -= k;
```

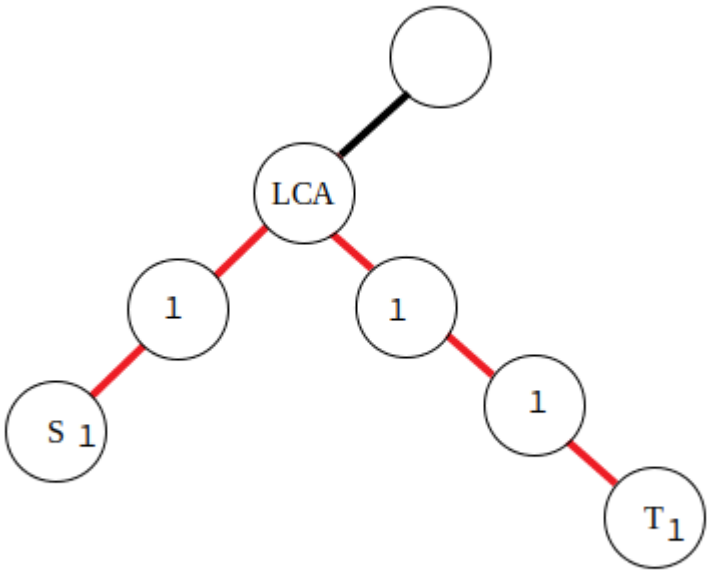


#边差分

边化归到其下方的结点

要使s到t的全部点权+k

```
dif[s] += k, dif[t] += k;
dif[lca] -= k << 1;
```



由差分数组还原权值数组，递归即可：

```
void dfs3(int u) {
        wgt[u] = dif[u];
        for (int i = hed[u]; i; i = nxt[i]) {
                v = to[i];
                if (v == fa[u]) continue;
                dfs3(v);
                wgt[u] += wgt[v];
        }
}
```

2024/3/1

#Floyd

```
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < i; ++j)
            dis[i][j] = dis[j][i] = min(dis[i][j], dis[i][k] + dis[k][j]);
    }
} // 无向图
```

#状态压缩

“我为人人”

`dp[s][u]`，用01串`s`代表已访问的状态，`u`表示当前所在状态

示例：

```
for (int i = 0; i < 1 << n; ++i)
    for (int j = 0; j < n; ++j)
        dp[i][j] = Inf;
dp[1][0] = 0;
for (int i = 1; i < 1 << n; ++i) {
    if (i & 1 == 0)
        continue;
    for (int u = 0; u < n; ++u) {
        if ((i >> u) & 1 == 0)
            continue;
        for (int v = 0; v < n; ++v) {
            if ((i >> v) & 1)
                continue;
            dp[i | (1 << v)][v] = min(dp[i | (1 << v)][v], dp[i][u] + dis[u]
[v]);
        }
    }
}
```

2024/3/3

#数位dp

`dp[x][y]`代表仅考虑数位`a[1..x]`且`a[y]`为状态`y`时的目标值，列出转移方程即可

2024/3/5

#欧拉函数

求单个：利用欧拉函数的积性和质因数分解，将求大数的欧拉函数化归到质数幂次的欧拉函数，$\varphi(p^k)=p^{k-1}(p-1)$

求多个：线性筛的时候根据欧拉函数的积性递推：

```
for (int i = 2; i < N; ++i) {
    if (!vis[i])
        ps[++n] = i, phi[i] = i - 1;
    for (int j = 1; j ≤ n; ++i) {
        int p = ps[j], t = i * p;
        if (t ≥ N)
            break;
        vis[t] = true;
        if (i % p)
            phi[t] = phi[i] * phi[p];
        else {
            phi[t] = phi[i] * p;
            break;
        }
    }
}
```

2024/3/13

#网格填充

#动态规划 #状态压缩

设前i行已填好，第i行为状态j（状压），试图找到$dp[i][j]$与$dp[i-1][k]$的关系，一行一行递推。

2024/3/15

#网格填充

条形区域（仅一个维度在变化），按后缀递推

#矩阵快速幂

把状态转移方程写成矩阵乘积形式，如：

$$\begin{cases} a_n = a_{n-1} + 2a_{n-2} + a_{n-3} + 2b_{n-1} + 2c_{n-1} \\ b_n = a_{n-2} + c_{n-1} \\ c_n = a_{n-2} + b_{n-2} \end{cases}$$

变成：

$$\begin{pmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ b_n \\ b_{n-1} \\ c_n \end{pmatrix} = \begin{pmatrix} 1 & 2 & 1 & 2 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_{n-1} \\ a_{n-2} \\ a_{n-3} \\ b_{n-1} \\ b_{n-2} \\ c_{n-1} \end{pmatrix}$$

代码如下：

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define INF 0x3f3f3f3f3f3f3f3f

const ll P = 1e9 + 7;

struct Mat
{
    ll a[6][6];
    Mat()
    {
        for (int i = 0; i < 6; ++i) {
            for (int j = 0; j < 6; ++j) {
                a[i][j] = i == j;
            }
        }
    }
    void mul(const Mat m)
    {
        ll b[6][6];
        for (int i = 0; i < 6; ++i) {
            for (int j = 0; j < 6; ++j) {
                b[i][j] = 0;
                for (int k = 0; k < 6; ++k) {
                    b[i][j] = (b[i][j] + a[i][k] * m.a[k][j]) % P;
                }
            }
        }
```

```
        memcpy(a, b, sizeof(b));
    }
};

ll a0[6][6] = {
1, 2, 1, 2, 0, 2,
1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 0, 1,
0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 1, 0};

ll x2[6] = {3, 1, 1, 1, 0, 1};

signed main()
{
    Mat A;
    memcpy(A.a, a0, sizeof(a0));

    ll n;
    cin >> n;

    if (n == 1)
        return cout << 1, 0;
    else if (n == 2)
        return cout << 3, 0;

    n -= 2;

    Mat B;
    while (n) {
        if (n & 1)
            B.mul(A);
        A.mul(A);
        n >>= 1;
    }

    ll ans = 0;
    for (int k = 0; k < 6; ++k)
        ans += B.a[0][k] * x2[k];
    cout << ans % P;
}
```

2024/3/16

#1373. 川川超市

#01背包

```
fill(dp, dp + 15005, -100000);
dp[0] = 0;
for (int i = 1; i ≤ n; ++i) {
    for (int j = 15005; j ≥ w[i]; --j) {
        dp[j] = max(dp[j - w[i]] + v[i], dp[j]);
    }
}
```

#多重覆盖

```
vis[0] = true;
for (int i = 0; i < 7; ++i) {
    for (int j = 0; j < a[i]; ++j) {
        for (int k = 15005; k ≥ money[i]; --k) {
            vis[k] |= vis[k - money[i]];
        }
    }
}
```

2024/3/17

MC0204 世界警察

#双指针 #滑动窗口

考察子串`s[i..j]`，`j`向右移动，移动不了了则`i`向右移动，循环往复

2024/3/22

P8737 [蓝桥杯 2020 国 B] 质数行者

#求和

三重求和，标准求时间复杂度过高，可变换求和方式，降低复杂度，如：

$$\sum_{i=0}^{x}\sum_{j=0}^{y}\sum_{k=0}^{z}\frac{g(x,i)g(y,j)g(z,k)(i+j+k)!}{i!j!k!}=\sum_{l=0}^{x+y}\left(\sum_{i=\max(0,l-y)}^{\min(x,l)}\frac{g(x,i)g(y,j)}{i!j!}\sum_{k=0}^{z}\frac{g(z,k)(l+k)!}{k!}\right)$$

时间复杂度由$O(xyz)$降为了$O((x+y)(x+y+z))$

#动态规划

记忆化搜索的常数或者甚至复杂度会高于遍历更新，所以建议在能写出非递归方法时就不要写

递归，防止TLE：
记忆化搜索（递归）：

```
ll g(int x, int i)
{
    if (i ≥ 500)
        return 0;
    if (stg[x][i] ≠ INF)
        return stg[x][i];
    if (i == 0)
        return stg[x][i] = (x == 0);
    if (i == 1)
        return stg[x][i] = !notp[x];
    ll re = 0;
    for (int y = 2; y < x; ++y)
        re += g(y, 1) * g(x - y, i - 1) % P;
    return stg[x][i] = re % P;
}
```

遍历（非递归，“我为人人”）：

```
stg[0][0] = 1;
for (int i = 0; i < 500; ++i) {
    for (int x = 0; x ≤ 1000; ++x) {
        for (int j = 1; j ≤ nn; ++j) {
            int p = ps[j], y = x + p;
            if (y > 1000)
                break;
            stg[y][i + 1] = (stg[y][i + 1] + stg[x][i]) % P;
        }
    }
}
```

2024/3/23

#相对运动
在网格点上的运动题，讨论两点之间的情况时可以其中一点为参考系（坐标原点），就简化成了单点运动，初始准备：

```
const int dx[4] = {1, 0, -1, 1}, dy[4] = {0, 1, 0, -1};
int x[N], y[N], d[N]; // d = 0,1,2,3分别代表右、上、左、下
```

#switch
注意要写`break`

2024/3/25

2. 二进制王国【算法赛】

#排序

要求排序后数组整体要满足某个性质，且性质能化归到每对元素的比较上。
要求字符串数组整体的字典序最小，可这样写比较函数：

```
bool cmp(const string& s1, const string& s2) {
    return s1 + s2 < s2 + s1;
}
```

#贪心

6. 小蓝的跳跃【算法赛】

问在一个数组上跳跃取值能否可能满足条件，考察极端情况`min`，`max`

2024/4/4

#随机数生成器

以下函数`randrange(int l, int r)`将生成区间$[l,r)$中随机的整数（不均匀）

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

inline int randrange(int l, int r) {
    return rand() * rand() % (r - l) + l;
}

signed main() {
    // freopen("a.in", "w", stdout);
    srand(time(0));

    for (int i = 0; i < 100; ++i)
        cout << randrange(0, 10) << ' ';

    return 0;
}
```

#并查集

一定要初始化`fa[i] = i`啊，否则合并会出问题！！！

```
// 寻找祖先
int find(int x) {
```

```
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}
// （非启发式）合并
void hb(int x, int y) {
    fa[find(x)] = find(y);
}
```

2024/4/6

#pow

内置函数`pow`函数返回的是一个浮点数！！！

请自己写`qpow`！！！

2024/4/16

#线段树

我的模版：（注意pushdown的位置）

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

const int N = 100005;
int n, m;
ll a[N];
ll tr[N << 2], lz[N << 2];

ll init(int l = 1, int r = n, int p = 1) {
    if (l == r)
        return tr[p] = a[l];
    int m = l + r >> 1, ps = p << 1;
    return tr[p] = init(l, m, ps) + init(m + 1, r, ps | 1);
}

void pd(int l, int r, int p) {
    if (l == r || lz[p] == 0)
        return;
    int m = l + r >> 1, ps = p << 1;
    lz[ps] += lz[p];
    lz[ps | 1] += lz[p];
    tr[ps] += lz[p] * (m - l + 1);
    tr[ps | 1] += lz[p] * (r - m);
    lz[p] = 0;
```

```
}

void add(int s, int t, int k, int l = 1, int r = n, int p = 1) {
    if (s == l && t == r) {
        tr[p] += k * (t - s + 1);
        lz[p] += k;
        return;
    }
    pd(l, r, p);
    int m = l + r >> 1, ps = p << 1;
    if (s ≤ m)
        add(s, min(t, m), k, l, m, ps);
    if (t > m)
        add(max(s, m + 1), t, k, m + 1, r, ps | 1);
    tr[p] = tr[ps] + tr[ps | 1];
}

ll qry(int s, int t, int l = 1, int r = n, int p = 1) {
    if (s == l && t == r)
        return tr[p];
    pd(l, r, p);
    int m = l + r >> 1, ps = p << 1;
    ll re = 0;
    if (s ≤ m)
        re += qry(s, min(t, m), l, m, ps);
    if (t > m)
        re += qry(max(s, m + 1), t, m + 1, r, ps | 1);
    return re;
}

signed main() {
    ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i ≤ n; ++i)
        cin >> a[i];
    init();
    while (m--) {
        int c;
        cin >> c;
        if (c == 1) {
            int x, y, k;
            cin >> x >> y >> k;
            add(x, y, k);
        }
        else {
            int x, y;
            cin >> x >> y;
            cout << qry(x, y) << '\n';
        }
```

```
    }

    return 0;
}
```

#树状数组

也可以用两个树状数组来实现，试图用差分数组直接算出前缀和。

`bt1[N]` 维护 `{d[1],d[2],...,d[n]}`

`bt2[N]` 维护 `{n*d[1],(n-1)*d[2],...,d[n]}`

（其中 `d[N]` 为原数列的前缀数组）

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

const int N = 100005;
int n, m;
ll a[N];
ll bt1[N], bt2[N];

void add(int x, ll k) {
    ll K = (n + 1 - x) * k;
    for (; x < N; x += x & -x) {
        bt1[x] += k;
        bt2[x] += K;
    }
}

void add(int l, int r, ll k) {
    add(l, k);
    add(r + 1, -k);
}

ll qry(int r) {
    ll sum = 0, dec = 0, k = n - r;
    for (; r; r -= r & -r) {
        dec += bt1[r];
        sum += bt2[r];
    }
    return sum - k * dec;
}

ll qry(int l, int r) {
    return qry(r) - qry(l - 1);
}

signed main() {
    ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
```

```
    cin >> n >> m;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
        add(i, a[i] - a[i - 1]);
    }

    while (m--) {
        int c;
        cin >> c;
        if (c == 1) {
            int x, y, k;
            cin >> x >> y >> k;
            add(x, y, k);
        }
        else {
            int x, y;
            cin >> x >> y;
            cout << qry(x, y) << '\n';
        }
    }

    return 0;
}
```

2024/4/19

#三分法

三分套三分，求二元函数的最小值：

```
const double e = 1e-10, kk = 1e-3;

double g(const P& p1, const P& p2, double k) {
    const double x = p1.x1 + k * (p1.x2 - p1.x1);
    const double y = p1.y1 + k * (p1.y2 - p1.y1);
    double l = 0, r = 1;
    double X, Y, a, b;
    while (r - l > e) {
        double m = (l + r) / 2, d = r - l, L = m - kk * d, R = m + kk * d;
        X = p2.x1 + L * (p2.x2 - p2.x1);
        Y = p2.y1 + L * (p2.y2 - p2.y1);
        a = hypot(x - X, y - Y) + hypot(X - p2.x, Y - p2.y, p2.h);
        X = p2.x1 + R * (p2.x2 - p2.x1);
        Y = p2.y1 + R * (p2.y2 - p2.y1);
        b = hypot(x - X, y - Y) + hypot(X - p2.x, Y - p2.y, p2.h);
        if (a < b)
            r = R;
```

```
        else
            l = L;
    }
    return hypot(x - p1.x, y - p1.y, p1.h) + min(a, b);
}

double f(const P& p1, const P& p2) {
    double l = 0, r = 1;
    double a, b;
    while (r - l > e) {
        double m = (l + r) / 2, d = r - l, L = m - kk * d, R = m + kk * d;
        a = g(p1, p2, L);
        b = g(p1, p2, R);
        if (a < b)
            r = R;
        else
            l = L;
    }
    return min(a, b);
}
```