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

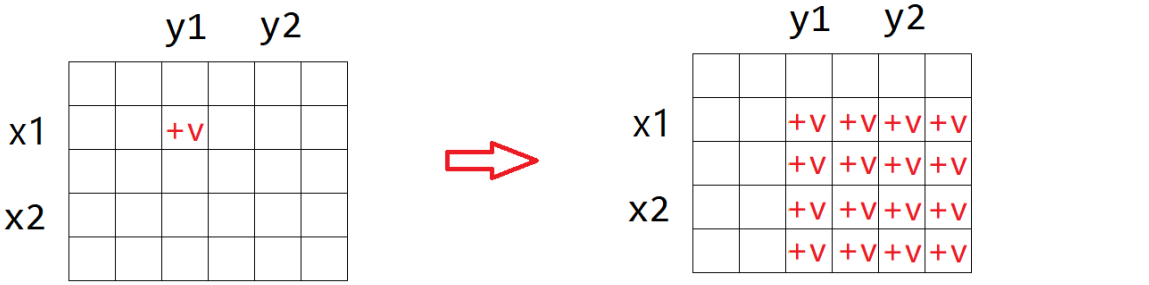
```
struct DATA {
        int a[N];
        inline int &operator()(int i, int j, int k) {
                return a[(i * B + j) * C + k];
        }
        inline void clear() {
                memset(a, 0, sizeof(a));
        }
} H, dif, sum;
```

#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

①

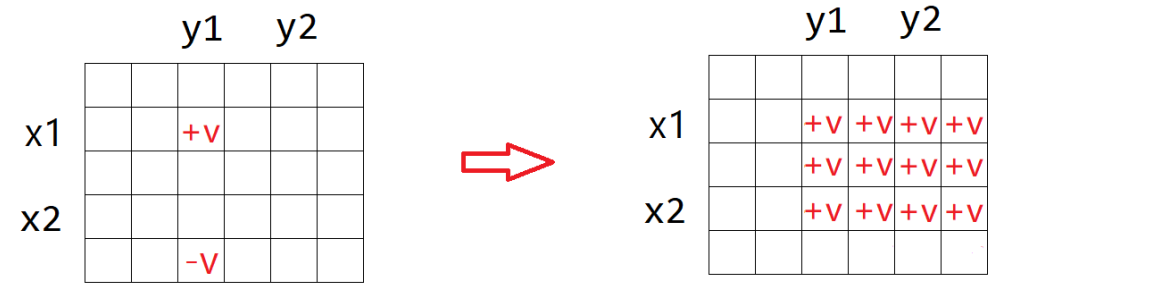


d[x1][y1] += v

CSDN @melonyzzZ

二维差分　　二维差分的前缀和

②



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

CSDN @melonyzzZ

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

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

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
```

```
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

#点权和最大的联通分量

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