常用算法 & 数据结构模板

大部分参考 Acwing 提供的模板，细节处有修改。

1 快速排序

```cpp
void qsort(std::vector<int>& a, int l, int r) {
    if (l >= r)
        return;

    int i = l - 1, j = r + 1, x = a[l + r >> 1];
    while (i < j) {
        do i ++; while (a[i] < x);
        do j --; while (a[j] > x);
        if (i < j) std::swap(a[i], a[j]);
    }

    qsort(a, l, j), qsort(a, j + 1, r);
}
```

2 归并排序

```cpp
void merge_sort(int l, int r) {
    if (l >= r)
        return;

    int mid = l + r >> 1;
    merge_sort(l, mid), merge_sort(mid + 1, r);

    int i = l, j = mid + 1, k = 0;
    while (i <= mid && j <= r) {
        if (a[i] <= a[j]) tmp[k ++] = a[i ++];
        else tmp[k ++] = a[j ++];
    }

    while (i <= mid) tmp[k ++] = a[i ++];
    while (j <= r) tmp[k ++] = a[j ++];

    for (int i = l; i <= r; ++ i)
        a[i] = tmp[i - l];
}
```

3 整数二分

```cpp
while (l < r) {
    int mid = l + r + 1 >> 1;
    if (a[mid] <= x) l = mid;
    else r = mid - 1
}

while (l < r) {
    int mid = l + r >> 1;
    if (a[mid] >= x) r = mid;
    else l = mid + 1;
}
```

4 高精度加法

```cpp
std::vector<int> add(std::vector<int> &A, std::vector<int> &B) {
    std::vector<int> res;

    int t = 0;
    for (int i = 0; i < A.size() || i < B.size(); ++ i) {
        t += (i < A.size() ? A[i] : 0) + (i < B.size() ? B[i] : 0);
        res.push_back(t % 10);
        t /= 10;
    }

    while (t) {
        res.push_back(t % 10);
        t /= 10;
    }

    while (res.size() > 1 && res.back() == 0) {
        res.pop_back();
    }

    return res;
}

// e.g.
auto C = add(A, B);

std::reverse(C.begin(), C.end());
for (auto &x : C) {
    std::cout << x;
}
```

5 高精度减法

```cpp
std::vector<int> sub(std::vector<int>& A, std::vector<int>& B) {
    std::vector<int> res;

    int t = 0;
    for (int i = 0; i < A.size() || i < B.size(); ++ i) {
        t = (i < A.size() ? A[i] : 0) - (i < B.size() ? B[i] : 0) - t;
        res.push_back((t + 10) % 10);
        t = t < 0;
    }

```

```cpp
    while (res.size() > 1 && res.back() == 0) {
        res.pop_back();
    }

    return res;
}

bool cmp(std::vector<int>& A, std::vector<int>& B) {
    if (A.size() != B.size()) {
        return A.size() > B.size();
    } else {
        for (int i = A.size() - 1; i >= 0; -- i)
            if (A[i] != B[i])
                return A[i] > B[i];
    }

    return true;
}

// e.g.
bool flag = cmp(A, B);

std::vector<int> C;

if (flag) C = sub(A, B);
else C = sub(B, A);

std::reverse(C.begin(), C.end());

if (!flag) std::cout << '-';
for (auto &x : C) {
    std::cout << x;
}
```

6 高精度乘法

```cpp
std::vector<int> mul(std::vector<int> &A, int b) {
    std::vector<int> res;

    int t = 0;
    for (auto &x : A) {
        t += x * b;
        res.push_back(t % 10);
        t /= 10;
    }

    while (t) {
        res.push_back(t % 10);
        t /= 10;
    }

    while (res.size() > 1 && res.back() == 0) {
        res.pop_back();
    }

    return res;
```

```cpp
}

// e.g.
auto C = mul(A, b);

std::reverse(C.begin(), C.end());
for (auto &x : C) {
    std::cout << x;
}
```

7 高精度除法

```cpp
PVI div(std::vector<int>& A, int b) {
    std::vector<int> res;

    int t = 0;
    for (int i = A.size() - 1; i >= 0; -- i) {
        t = t * 10 + A[i];
        res.push_back(t / b);
        t %= b;
    }

    std::reverse(res.begin(), res.end());

    while (res.size() > 1 && res.back() == 0) {
        res.pop_back();
    }

    return std::make_pair(res, t);
}

// e.g.
auto [C, r] = div(A, b);

std::reverse(C.begin(), C.end());
for (auto &x : C) {
    std::cout << x;
}
std::cout << '\n';
std::cout << r << '\n';
```

8 离散化

```cpp
std::vector<int> all;
std::sort(all.begin(), all.end());
all.erase(std::unique(all.begin(), all.end()), all.end());
```

9 模拟链表

```cpp
int h = -1, e[N], ne[N], idx;

void insert(int a) {
    e[idx] = a, ne[idx] = h, h = idx ++;
}

void remove() {
    h = ne[h];
}
```

10 模拟栈

```cpp
int stk[N], tt = 0;

// insert
stk[++ tt] = x;

// pop
-- tt

// if empty then ...
if (tt > 0) {
    // ...
}

// pop and query
std::cout << stk[tt --] << '\n';
```

11 模拟队列

```cpp
int q[N], hh = 0, tt = -1;

// insert to front
q[++ tt] = x;

// pop front
++ hh;

// query front
std::cout << q[hh] << '\n';

// if empty then ...
if (hh <= tt) {
    // ...
}
```

12 单调栈

```cpp
std::stack<int> stack;

for (int i = 0; i < n; ++ i) {
    while (stack.size() && stack.top() >= a[i]) {
        stack.pop();
    }

    std::cout << (stack.size() ? stack.top() : -1) << ' ';

    stack.push(a[i]);
}
```

13 单调队列

```cpp
std::deque<int> q;
for (int i = 0; i < n; ++ i) {
    if (q.size() && q.front() + len - 1 < i)
        q.pop_front();

    while (q.size() && a[q.back()] >= a[i])
        q.pop_back();

    q.push_back(i);

    if (i >= len - 1)
        std::cout << a[q.front()] << ' ';
}
```

14 KMP 算法

```cpp
void init() {
    int i = 1, j = 0;

    while (i <= m) {
        if (j == 0 || s[i] == s[j])
            next[++ i] = ++ j;
        else
            j = next[j];
    }
}

// usage:
// 此处假设字符串的下标从 1 开始。当在待查文本的第 i 位，模式串的第 j 位失配时，令 j = next[j]，即将模式串中下一个要比较的字符对齐 i。

// e.g.
i = 1, j = 1;
while (i <= n && j <= m) {
    if (i == 0 || p[i] == s[j]) {
        ++ i, ++ j;
    } else {
        i = next[i];
    }

    if (i == n + 1) {
        std::cout << j - i << ' ';
```

```
        i = next[i];
    }
}
```

15 Trie 树

```
int son[26][N], cnt[N], idx;

void insert(const std::string& s) {
    int p = 0;
    for (auto &c : s) {
        if (son[c - 'a'][p])
            p = son[c - 'a'][p];
        else
            p = son[c - 'a'][p] = ++ idx;
    }
    ++ cnt[p];
}

int query(const std::string& s) {
    int p = 0;
    for (auto &c : s) {
        if (son[c - 'a'][p])
            p = son[c - 'a'][p];
        else
            return 0;
    }
    return cnt[p];
}
```

16 并查集

```
int find(int x) {
    return p[x] == x ? p[x] : p[x] = find(p[x]);
}

// init
for (int i = 1; i <= n; ++ i) {
    p[i] = i;
}
```

17 Hash

17.1 一般 Hash

```cpp
// N \in primes

int find(int x)
{
    int t = (x % N + N) % N;
    while (h[t] != null && h[t] != x)
    {
        t ++ ;
        if (t == N) t = 0;
    }
    return t;
}
```

17.2 字符串 Hash

```cpp
constexpr u64 N = 1e5 + 10, P = 13331;

u64 h[N], p[N] = { 1 };

int get(int l, int r) {
    return h[r] - h[l - 1] * p[r - l + 1];
}

// usage
char c;
for (int i = 1; i <= n; ++ i) {
    std::cin >> c;

    h[i] = h[i - 1] * P + c;
    p[i] = p[i - 1] * P;
}
```

18 链式前向星

```cpp
int h[N], e[N], ne[N], idx;

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

// init
memset(h, -1, sizeof h)

// usage
void dfs(int u) {
    if (st[u]) return;
    st[u] = true;

    std::cout << u << '\n';

    for (int i = h[u]; i != -1; i = ne[i]) {
        int v = e[i];

        dfs(v);
    }
}
```

19 拓扑排序

```cpp
std::queue<int> q;
std::vector<int> topo;

for (int i = 1; i <= n; ++ i) {
    if (in[i] == 0) {
        q.push(i);
    }
}

while (q.size()) {
    auto t = q.front();
    q.pop();

    topo.push_back(t);

    for (int i = h[t]; i != -1; i = ne[i]) {
        int j = e[i];

        if (-- in[j] == 0) {
            q.push(j);
        }
    }
}

if (topo.size() == n) {
    for (auto &x : topo) {
        std::cout << x << ' ';
    }
} else {
    std::cout << -1 << '\n';
}
```

20 单源最短路经算法

正确性的证明可参考 单源最短路径算法正确性的证明。

20.1 Dijkstra 算法

此处给出堆优化版的 Dijkstra 算法

20.1.1 伪代码

DIJKSTRA(G, w, s)

INITIALIZE_SINGLE_SOURCE(G, s)
　$S = \varnothing$
　$Q = G.V$
　while $Q \neq \varnothing$
　　$u =$ EXTRACT_MIN(Q)
　　$S = S \cup \{u\}$
　　for each vertex $v \in G.\mathrm{Adj}[u]$
　　　RELAX(u, v, w)

20.1.2 代码实现

```cpp
auto dijkstra = [&] () {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    std::priority_queue<std::pair<int, int>> q;
    q.push({ 0, 1 });

    while (q.size()) {
        auto [_, t] = q.top();
        q.pop();

        if (st[t])
            continue;

        st[t] = true;

        for (int i = h[t]; i != -1; i = ne[i]) {
            int j = e[i];

            if (dist[j] > dist[t] + w[i]) {
                dist[j] = dist[t] + w[i];

                q.push({ -dist[j], j });
            }
        }
    }

    return dist[n];
};
```

20.2 Bellman-Ford 算法以及 SPFA 算法

20.2.1 伪代码

BELLMAN-FORD(G, w, s)
1 INITIALIZE_SINGLE_SOURCE(G, s)
2 **for** $i = 1$ **to** $|G.V| - 1$
3   **for** each edge$(u, v) \in G.E$
4   RELAX(u, v, w)
5 **for** each edge$(u, v) \in G.E$
6   **if** $v.d > u.d + w(u, v)$
7   **return** FALSE
8 **return** TRUE

SHORTEST-PATH-FAST-ALGORITHM(G, w, s)

INITIALIZE_SINGLE_SOURCE(G, s)
$Q = \langle v_1, v_2, \cdots, v_k \rangle$ // $v_i \in G.V,\ k = |G.V|$
COUNT $= []$
// COUNT is an array used to store the number of nodes on a certain shortest path
// to determine whether the graph has a negative cycle.
while $Q \neq \varnothing$
  $u =$ EXTRACT_FRONT(Q)
  for each vertex $v \in G.\mathrm{Adj}[u]$
    if RELAX$(u, v, w) =$ TRUE
      COUNT$[v] =$ COUNT$[u] + 1$
      if COUNT$[v] \geq |G.V|$
        return FALSE
      if $v \notin Q$
        $Q = Q$ push v
return TRUE

20.2.2 代码实现

```cpp
auto bellman_ford = [&] () {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    for (int i = 1; i <= n; ++ i) {
        for (auto &[a, b, w] : edge) {
            if (dist[b] > bak[a] + w) {
                dist[b] = bak[a] + w;
            }
        }
    }

    return dist[n];
};
```

```cpp
auto spfa = [&] () {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    std::queue<int> q;

    for (int i = 1; i <= n; ++ i) {
        q.push(i), st[i] = true;
    }

    while (q.size()) {
        int t = q.front();
        q.pop(), st[t] = false;

        for (int i = h[t]; i != -1; i = ne[i]) {
            int j = e[i];

            if (dist[j] > dist[t] + w[i]) {
                dist[j] = dist[t] + w[i];

                cnt[j] = cnt[t] + 1;
```

```
                if (cnt[j] >= n) {
                    return false;
                }

                if (!st[j]) {
                    q.push(j), st[j] = true;
                }
            }
        }
    }

    return true;
};
```

21 最小生成树

21.1 Prim 算法

```
auto prim = [&] () {
    memset(dist, 0x3f, sizeof dist);

    int res = 0;

    for (int i = 1; i <= n; ++ i) {
        int t = -1;

        for (int j = 1; j <= n; ++ j)
            if (!st[j] && (t == -1 || dist[j] < dist[t]))
                t = j;

        if (t > 1 && dist[t] == INF)
            return 0;

        if (t > 1)
            res += dist[t];

        st[t] = true;

        for (int j = 1; j <= n; ++ j)
            dist[j] = std::min(dist[j], g[t][j]);
    }

    return res;
};
```

21.2 Kruskal 算法

```
using Edge = std::array<int, 3>;
std::vector<Edge> edge(m);

for (auto &[a, b, w] : edge) {
    std::cin >> a >> b >> w;
}

std::sort(edge.begin(), edge.end(), [] (Edge a, Edge b) {
```

```
    return a.at(2) < b.at(2);
});

int res = 0, cnt = 0;

for (auto &[a, b, w] : edge) {
    a = find(a), b = find(b);

    if (a != b) {
        res += w, ++ cnt;
        p[a] = b;
    }
}
```

22 筛质数

22.1 试除法 $O(\sqrt{n})$

```
bool is_prime(int x) {
    if (x < 2) return false;

    for (int i = 2; i <= x / i; ++ i) {
        if (x % i == 0)
            return false;
    }
    return true;
}
```

22.2 埃氏筛法 $O(n \log \log n)$

```
for (int i = 2; i <= n; ++ i) {
    if (!st[i]) {
        primes.push_back(i);
        for (int j = i; j <= n / i; ++ j) {
            st[i * j] = true;
        }
    }
}
```

22.3 线性筛法 $O(n)$

```cpp
std::vector<int> primes;

for (int i = 2; i <= n; ++ i) {
    if (!st[i])
        primes.push_back(i);

    for (auto &p : primes) {
        if (p * i > n) break;
        st[p * i] = true;
        if (i % p == 0) break;
    }
}
```

23 欧几里得算法

```cpp
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}
```

24 欧拉函数

24.1 性质

积性函数：若 $\gcd(a,b)=1$ 则有 $\varphi(ab)=\varphi(a)\varphi(b)$。

对 $p \mid n$ 有 $\varphi(n\cdot p)=p\cdot\varphi(n)$。

对 $p \nmid n$ 有 $\varphi(n\cdot p)=(p-1)\cdot\varphi(n)$。

24.2 证明

对于 $p\in primes$ 易有 $\varphi(p)=p-1$。

又有对于 p^k，除了 p 的倍数的所有数都与 p^k 互质，而 p 的倍数有 p^{k-1} 个，所以有 $\varphi(p^k)=p^k-p^{k-1}=p^{k-1}(p-1)$ 。

对于一般的自然数，有

$$\begin{aligned}
\varphi(n) &= \prod \varphi(p_i^{k_i}) \\
&= \prod p_i^{k_i-1}(p-1) \\
&= \prod p_i^{k_i}(1-\frac{1}{p}) \\
&= \prod p_i^{k_i} \prod \frac{p-1}{p} \\
&= n\prod \frac{p-1}{p}
\end{aligned}$$

24.3 推广

欧拉定理：对 $\gcd(n, m) = 1$ 有 $n^{\varphi(m)} \equiv 1 \pmod m$。

特殊情况下有 **费马小定理**：$p \in primes$, $n^{p-1} \equiv 1 \pmod p$。常用于求乘法逆元。

24.3.1 证明

对于 m 的一个简化剩余系 $r_1, r_2, \ldots, r_{\varphi(m)}$，由于 $\gcd(n, m) = 1$，容易想到其与 $nr_1, nr_2, \ldots, nr_{\varphi(m)}$ 等价。

所以有 $n^{\varphi(m)} \cdot r_1 \cdot r_2 \cdot \cdots \cdot r_{\varphi(m)} \equiv r_1 \cdot r_2 \cdot \cdots \cdot r_{\varphi(m)} \pmod m$。

化简有 ${}^{\varphi(m)} \equiv 1 \pmod m$。

25 快速幂

```
i64 pmod(i64 a, i64 b, i64 p) {
    i64 res = 1;

    for (; b; b >>= 1) {
        if (b & 1) res = (res * a) % p;
        a = (a * a) % p;
    }

    return res;
}
```

26 扩展欧几里得算法

26.1 Bézout 定理

存在 $ax + by = \gcd(a, b)$。

26.1.1 证明

1. 对于 $\gcd(a, 0)$，存在 $x = 1, y = 0$。
2. 对于 $\gcd(a, b)$，假设对 $\gcd(b, a \bmod b)$，有 $b \cdot x' + (a - \lfloor \frac{a}{b} \rfloor b) \cdot y' = \gcd(b, a \bmod b)$。

   则有

$$
\begin{aligned}
b \cdot x' + (a - \lfloor \frac{a}{b} \rfloor b) \cdot y' &= a \cdot y' + b \cdot (x' - \lfloor \frac{a}{b} \rfloor y') \\
&= \gcd(b, a \bmod b) \\
&= \gcd(a, b)
\end{aligned}
$$

   其中 $x = y', y = x' - \lfloor \frac{a}{b} \rfloor y'$。

26.2 代码

用于求解 $ax + by = \gcd(a, b)$。

```cpp
int exgcd(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }

    int gcd = exgcd(b, a % b, y, x);

    y -= (a / b) * x;

    return gcd;
}
```

27 高斯消元

```cpp
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n;
    std::cin >> n;

    for (int i = 0; i < n; ++ i)
        for (int j = 0; j <= n; ++ j)
            std::cin >> g[i][j];

    auto gauss = [&] () {
        int col = 0, row = 0;

        for (; col < n; ++ col) {
            int t = row;

            for (int i = t + 1; i < n; ++ i)
                if (std::abs(g[i][col]) > std::abs(g[t][col]))
                    t = i;

            if (std::abs(g[t][col]) < eps)
                continue;

            for (int i = col; i <= n; ++ i)
                std::swap(g[t][i], g[row][i]);

            for (int i = n; i >= col; -- i)
                g[row][i] /= g[row][col];

            for (int i = row + 1; i < n; ++ i)
                for (int j = n; j >= col; -- j)
                    g[i][j] -= g[row][j] * g[i][col];

```

```
            ++ row;
        }

        if (row < n) {
            for (int i = row; i < n; ++ i)
                if (std::abs(g[i][n]) > eps)
                    return 0;
            return 2;
        }

        for (int i = row - 1; i >= 0; -- i)
            for (int j = n - 1; j > i; -- j)
                g[i][n] -= g[i][j] * g[j][n];

        return 1;
    };

    int flag = gauss();

    if (flag == 0) {
        std::cout << "No solution" << '\n';
    } else if (flag == 2) {
        std::cout << "Infinite group solutions" << '\n';
    } else {
        for (int i = 0; i < n; ++ i)
            std::cout << std::fixed << std::setprecision(2) << g[i][n] << '\n';
    }

    return 0;
}
```

28 Lucas 定理

对 $p \in primes$，$C_n^m = C_{n \bmod p}^{m \bmod p} \cdot C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} \pmod p$。

28.1 证明

将 m, n 看作 p 进制数，有 $m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_0 p_o$ ，$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_0 p_o$ 。

我们使用类似生成函数的思想，对 $(1+x)^p$ 二项式展开，有

$$\begin{aligned}(1+x)^p &= C_p^0 x^0 + C_p^1 x^1 + \cdots + C_p^p x^p \\ &\equiv 1 + x^p \pmod p\end{aligned}$$

所以对 $(1+x)^n$ ，有

$$\begin{aligned}(1+x)^n &\equiv (1+x)^{n_0 p^0} \cdot (1+x)^{n_1 p^1} \cdot \cdots \cdot (1+x)^{n_k p^k} \pmod p \\ &\equiv (1+x^{p_0})^{n_0} \cdot (1+x^{p_1})^{n_1} \cdot \cdots \cdot (1+x^{p_k})^{n_k} \pmod p\end{aligned}$$

由二项式和组合数的性质，要构造出 $(1+x)^m$ 的展开式，需要从 $(1+x^{p_0})^{n_0}$ 中选出 x^{p_0} 项的 m_0 次方，从 $(1+x^{p_1})^{n_1}$ 中选出 x^{p_1} 项的 m_1 次方......其中系数分别为 $C_{n_0}^{m_0}, C_{n_1}^{m_1}, \ldots, C_{n_k}^{m_k}$。

由此我们得出 $C_n^m \equiv C_{n_0}^{m_0} \cdot C_{n_1}^{m_1} \cdot \cdots \cdot C_{n_k}^{m_k} \pmod p$。由进制转换的性质表示成递归形式也即

$$C_n^m = C_{n \bmod p}^{m \bmod p} \cdot C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} \pmod p$$

29 线性同余方程组的求解

29.1 中国剩余定理

$$\begin{cases} x & \equiv a_1 \pmod{m_1} \\ x & \equiv a_2 \pmod{m_2} \\ & \vdots \\ x & \equiv a_n \pmod{m_n} \end{cases}$$

29.1.1 证明

对于 m_i 两两互质的情况，令

$$m = \prod_{i=1}^{n} m_i$$
$$M_i = \frac{m}{m_i}$$
$$M_i t_i \equiv 1 \pmod{m_i}$$

$a_i M_i t_i$ 是 $\forall k \neq i,\ m_k$ 的倍数。

$$\begin{cases} a_i M_i t_i & \equiv 0 \pmod{m_k} \\ a_i M_i t_i & \equiv a_i \pmod{m_i} \end{cases}$$

有

$$x = \sum_{i=1}^{n} a_i M_i t_i$$

29.2 对于一般的线性同余方程组的求解

对于 m_i 不两两互质的情况，我们考虑分别求解单个线性同余方程，并将各个线性同余方程的解联系起来。

假设已求出前 $k-1$ 项的一个解 x，我们令 $m = \mathrm{lcm}_{i=1}^{k-1} m_i$，则前 $k-1$ 项的通解为 $x + m \cdot i \quad (i \in \mathbb{Z})$。

对式 k，我们需要找到一个正整数 t，使得 $x + mt \equiv a_k \pmod{m_k}$，也即 $m \cdot t \equiv a_k - x \pmod{m_k}$，则对前 k 项有 $x' = x + mt$，在代码实现中需注意右式 $a_k - x$ 需大于零，且在 C 语言 中随时使用 `(x % p + p) % p` 来保证为最小整数解。

具体来说，原式可以化为 $m \cdot t + m_k \cdot y = a_k - x$，我们用扩展欧几里得算法可以求出 $m \cdot t + m_k \cdot y = \gcd(m, m_k)$ 的解，我们知道原式存在解当且仅当 $\gcd(m, m_k) \mid (a_k - x)$。

30 Nim 博弈

对于 $a_1, a_2, \ldots, a_n$，若

1. $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$ ，则先手必败。
2. $a_1 \oplus a_2 \oplus \cdots \oplus a_n \neq 0$，则先手必胜

30.1 证明

1. $0 \oplus 0 \oplus \cdots \oplus 0 = 0$

2. 当 $a_1 \oplus a_2 \oplus \cdots \oplus a_n = x \neq 0$ 时，一定能操作到等于零：

   一定存在一个 a_i 的第 k 位为 1，我们知道 $a_i \oplus x < a_i$，若令 a_i 减少 $a_i - a_i \oplus x$，则 a_i 变为 $a_i \oplus x$：

$$a_1 \oplus a_2 \oplus \cdots \oplus a_i \oplus x \oplus \cdots \oplus a_n = x \oplus x = 0$$

3. 当 $a_1 \oplus a_2 \oplus \cdots \oplus a_n = x = 0$ 时，无论怎么操作都不能令新的式子保持为 0:

   若原式变为 $a_1 \oplus a_2 \oplus \cdots \oplus a_i' \oplus \cdots \oplus a_n = 0$，而原式 $a_1 \oplus a_2 \oplus \cdots \oplus a_i \oplus \cdots \oplus a_n = 0$，则 $a_i \oplus a_i' = 0$，有 $a_i = a_i'$，矛盾。

换句话说，若先手面对的情况为 $a_1 \oplus a_2 \oplus \cdots \oplus a_n = x \neq 0$，那么后面面对的情况一定是先手不等于零而后手等于零的状态，即先手必胜。

30.2 SG 函数

在一个有向无环图中，终点为 0，函数 $\mathrm{SG}(x)$ 表示不在节点 x 的后继集合中的最小非负整数。

性质 若 $\mathrm{SG}(x) = 0$，则必败，反之必胜。

由于 $\mathrm{SG}(x)$ 能操作成任何小于 $\mathrm{SG}(x)$ 的非负整数，类比前面的证明，容易得出

$$\mathrm{SG}(x_1) \oplus \mathrm{SG}(x_2) \oplus \cdots \oplus \mathrm{SG}(x_n) = 0$$

时，先手必败，反之必胜。

31 ST 算法

用于解决区间最值问题。

```cpp
template <typename T>
class SparseTable {
    std::vector<std::vector<T>> ST;

public:
    SparseTable (const std::vector<T> &v) {
        int n = v.size(), t = std::log(n) / std::log(2) + 1;
        ST.assign(n, std::vector<T>(t, 0));

        for (int i = 0; i < n; ++ i) {
            ST[i][0] = v[i];
        }

        for (int j = 1; j < t; ++ j)
            for (int i = 0; i + (1 << (j - 1)) < n; ++ i)
                ST[i][j] = std::max(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
    }

    T query(int l, int r) {
        if (l == r)
            return ST[l][0];

        int k = std::log(r - l + 1) / std::log(2);
        return std::max(ST[l][k], ST[r - (1 << k) + 1][k]);
    }
};
```

32 树状数组

```
template <typename T>
class BIT {
private:
    std::vector<T> bit;
    int n;

    int lowbit(int x) {
        return x & -x;
    }

public:
    BIT (int n) : n(n) {
        bit.assign(n, 0);
    }

    T ask(int r) {
        int res = 0;
        for (; r > 0; r -= lowbit(r))
            res += bit[r];
        return res;
    }

    T ask(int l, int r) {
        if (l > r)
            std::swap(l, r);
        return ask(r) - ask(l - 1);
    }

    void add(int i, int v) {
        for (; i < n; i += lowbit(i))
            bit[i] += v;
    }
};
```

33 线段树

```
struct Seg {
    i64 l, r, dat, sum, add;
};

#define sum(x) tree[x].sum
#define add(x) tree[x].add
#define l(x) tree[x].l
#define r(x) tree[x].r

const int N = 1e5+10;

Seg tree[N * 4];
i64 a[N];

void build(int p, int l, int r) {
    l(p) = l, r(p) = r;
```

```cpp

    if (l == r) {
        sum(p) = a[l];
        return;
    }

    int mid = l + r >> 1;
    build(p * 2, l, mid), build(p * 2 + 1, mid + 1, r);
    sum(p) = sum(p * 2) + sum(p * 2 + 1);
}

void spread(int p) {
    if (add(p)) {
        sum(p * 2) += add(p) * (r(p * 2) - l(p * 2) + 1);
        sum(p * 2 + 1) += add(p) * (r(p * 2 + 1) - l(p * 2 + 1) + 1);
        add(p * 2) += add(p);
        add(p * 2 + 1) += add(p);
        add(p) = 0;
    }
}

void change(int p, int l, int r, i64 v) {
    if (l <= l(p) && r >= r(p)) {
        sum(p) += v * (r(p) - l(p) + 1), add(p) += v;
        return;
    }

    spread(p);

    int mid = l(p) + r(p) >> 1;
    if (l <= mid) change(p * 2, l, r, v);
    if (r > mid) change(p * 2 + 1, l, r, v);
    sum(p) = sum(p * 2) + sum(p * 2 + 1);
}

i64 ask(int p, int l, int r) {
    if (l <= l(p) && r >= r(p))
        return sum(p);

    spread(p);

    int mid = l(p) + r(p) >> 1;

    i64 res = 0;
    if (l <= mid) res += ask(p * 2, l, r);
    if (r > mid) res += ask(p * 2 + 1, l, r);
    return res;
}
```

34 Treap 树

```cpp
struct item {
    int key, prior, cnt, size;
    item *l, *r;
    item () { }
```

```
    item (int key) : key(key), prior(std::rand()), l(nullptr), r(nullptr), cnt(1), size(1) { }
};

using pitem = item*;

pitem root = nullptr;

void update(pitem& x) {
    x->size = x->cnt + (x->l ? x->l->size : 0) + (x->r ? x->r->size : 0);
}

void zig(pitem& x) {
    pitem y = x->l;
    x->l = y->r, y->r = x, x = y;
    update(x), update(x->r);
}

void zag(pitem& x) {
    pitem y = x->r;
    x->r = y->l, y->l = x, x = y;
    update(x), update(x->l);
}

void insert(pitem& x, int y) {
    if (!x)
        return x = new item(y), void();
    if (x->key == y)
        return ++ x->cnt, update(x), void();
    if (y < x->key) {
        insert(x->l, y);
        if (x->l->prior > x->prior) zig(x);
    } else {
        insert(x->r, y);
        if (x->r->prior > x->prior) zag(x);
    }
    update(x);
}

void remove(pitem& x, int y) {
    if (y < x->key) remove(x->l, y);
    else if (y > x->key) remove(x->r, y);
    else {
        if (x->cnt > 1) -- x->cnt;
        else if (!x->l) x = x->r;
        else if (!x->r) x = x->l;
        else {
            zag(x);
            remove(x->l, y);
            if (x->l && x->l->prior > x->prior)
                zig(x);
        }
    }
    if (x) update(x);
}

```

```cpp
pitem getPre(int v) {
    pitem x = root, ans = new item(-1e9);
    while (x) {
        if (v == x->key)
            if (x->l) {
                x = x->l;
                while (x->r) x = x->r;
                ans = x;
            }
        if (x->key < v && x->key > ans->key)
            ans = x;
        x = v < x->key ? x->l : x->r;
    }
    return ans;
}

pitem getNxt(int v) {
    pitem x = root, ans = new item(1e9);
    while (x) {
        if (v == x->key)
            if (x->r) {
                x = x->r;
                while (x->l) x = x->l;
            }
        if (x->key > v && x->key < ans->key)
            ans = x;
        x = v > x->key ? x->r : x->l;
    }
    return ans;
}

int getValByRank(pitem& x, int rank) {
    if (!x) return 1e9;
    if ((x->l ? x->l->size : 0) >= rank)
        return getValByRank(x->l, rank);
    if ((x->l ? x->l->size : 0) + x->cnt >= rank)
        return x->key;
    return getValByRank(x->r, rank - (x->l ? x->l->size : 0) - x->cnt);
}

int getRankByVal(pitem& x, int v) {
    if (!x) return 0;
    if (v == x->key) return (x->l ? x->l->size : 0) + 1;
    if (v < x->key) return getRankByVal(x->l, v);
    return getRankByVal(x->r, v) + (x->l ? x->l->size : 0) + x->cnt;
}
```

35 快读快输

```cpp
namespace IO {
    template <typename T> inline T read() {
        char ch = getchar();
        T ret = 0, sig = 1;
        while(ch < '0' || ch > '9') { if(ch == '-') sig = -1; ch = getchar(); }
        while(ch >= '0' && ch <= '9') ret *= 10, ret += ch - 48, ch = getchar();
        return ret * sig;
```

```cpp
    }
    template <typename T> inline void write(T out) {
        if(!out) { putchar('0'), putchar(' '); return; }
        int stk[100], tt = 0;
        if(out < 0) out = -out, putchar('-');
        while(out) stk[tt++] = out % 10, out /= 10;
        for(register int i = --tt; i>=0; --i) putchar(stk[i] + 48);
        putchar(' ');
    }
    template <typename T> inline void read(T& ret) { ret = IO::read<T>(); }
    template <typename T, typename... Args> inline void read(T& x, Args&... args) { IO::read(x), IO::read(args...); }
    template <typename T, typename... Args> inline void write(T x, Args... args)  { IO::write(x), IO::write(args...); }
};
```

36 封装的高精度

```cpp
const int BASE = 1e4, WEIGHT = 4;

struct Bigint {
    std::vector<int> num;

    Bigint() {}

    Bigint(const char* x) {
        int len = strlen(x), tmp = 0;
        for (int i = 0, w = 1; i < len; w *= 10, ++ i) {
            if (w == BASE) w = 1, num.emplace_back(tmp), tmp = 0;
            tmp += w * (x[len - i - 1] - 48);
        }
        if (tmp) num.emplace_back(tmp);
    }

    Bigint(const int x) {
        char tmp[100];
        sprintf(tmp, "%d", x);
        *this = tmp;
    }

    friend std::ostream& operator << (std::ostream& os, const Bigint x) {
        if (!x.num.size()) {
            os << 0;
            return os;
        }
        for (int i = (int) x.num.size() - 1; i >= 0; -- i)
            if (i != (int) x.num.size() - 1) os << std::setfill('0') << std::setw(WEIGHT) << x.num[i];
            else os << x.num[i];
        return os;
    }

    friend std::istream& operator >> (std::istream& is, Bigint& x) {
        std::string str;
        is >> str;
        x = str.c_str();
```

```

        return is;
    }

    template <typename T>
    Bigint operator + (const T rst) {
        Bigint ret = 0, x = rst;
        int t = 0;
        for (int i = 0; i < (int) num.size() || i < (int) x.num.size() || t; ++ i) {
            if (i < (int) num.size()) t += num[i];
            if (i < (int) x.num.size()) t += x.num[i];
            ret.num.emplace_back(t % BASE);
            t /= BASE;
        }
        return ret;
    }

    Bigint operator * (const int rst) {
        if (!num.size()) return (Bigint) 0;
        Bigint ret = 0;
        int t = 0;
        for (int i = 0; i < (int) num.size() || t; ++ i) {
            if (i < (int) num.size()) t += num[i] * rst;
            ret.num.emplace_back(t % BASE);
            t /= BASE;
        }
        while (ret.num.back() == 0 && (int) ret.num.size() > 1) ret.num.pop_back();
        return ret;
    }

    template <typename T>
    Bigint operator * (const T rst) {
        Bigint ret = 0, x = rst;
        std::reverse(x.num.begin(), x.num.end());
        for (auto i : x.num) ret = ret * 10 + *this * i;
        return ret;
    }

    template <typename T>
    Bigint operator *= (const T rst) {
        *this = *this * rst;
        return *this;
    }

    template <typename T>
    Bigint operator += (const T rst) {
        *this = *this + rst;
        return *this;
    }

    template <typename T>
    bool operator < (T rst) {
        Bigint x = rst;
        if(num.size() > x.num.size()) return false;
        else if(num.size() < x.num.size()) return true;
        else {
```

```
            for(int i = num.size() - 1; i >= 0; -- i)
                if(num[i] < x.num[i]) return true;
            return false;
        }
        return false;
    }

    template <typename T>
    bool operator == (T rst) {
        Bigint x = rst;
        return num == x.num;
    }

    template <typename T>
    bool operator > (T rst) {
        Bigint x = rst;
        return !(*this == x || *this < x);
    }

    template <typename T>
    bool operator <= (T rst) {
        return *this == rst || *this < rst;
    }

    template <typename T>
    bool operator >= (T rst) {
        return *this == rst || !(*this < rst);
    }
};
```