Tricks

目录

Preface

记录遇到的各种 trick。

upd 2025.10.5 鉴于本文过于臃肿的情况，将本文按时间顺序 divide 成章节。

第一章 (2024.7 ~ 2024.8)

$n-\left\lfloor\frac{a}{b}\right\rfloor=\left\lceil\frac{nb-a}{b}\right\rceil$。

对于 $x\in Z\wedge y\in R$，有：

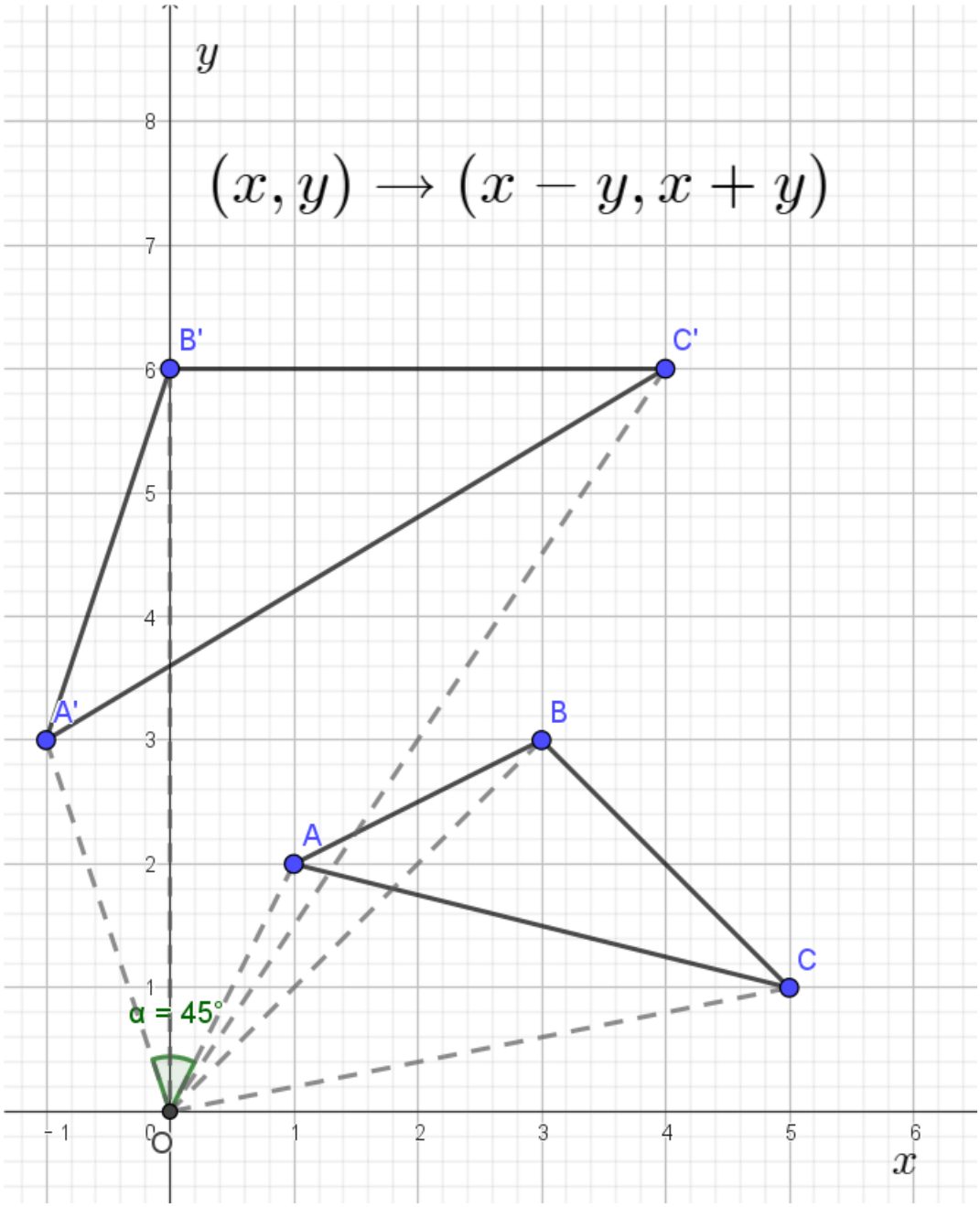
1. $x\le\lfloor y\rfloor\iff x\le y$
1. $x\ge\lceil y\rceil\iff x\ge y$
1. $x<\lceil y\rceil\iff x<y$
1. $x>\lfloor y\rfloor\iff x>y$

$(-1)^x=1-2\times(x\bmod 2)$。

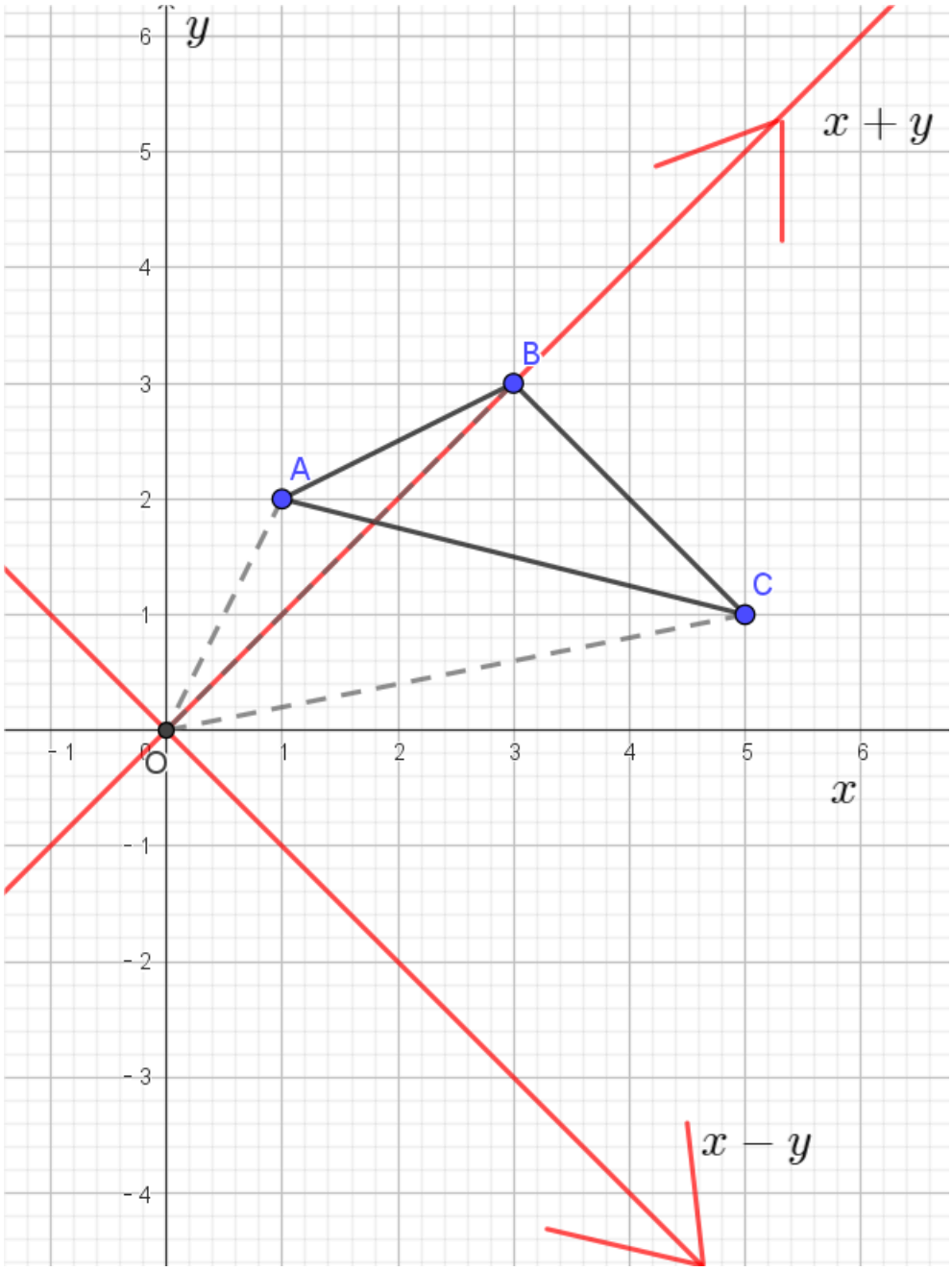
$\left\lfloor\frac{\lfloor x\rfloor}{y}\right\rfloor=\left\lfloor\frac{x}{y}\right\rfloor(x\in\mathbb{R}\wedge y\in\mathbb{Z}^+)$。

> Note
>
> 注意 $\left\lfloor\frac{a\lfloor x\rfloor}{b}\right\rfloor=\left\lfloor\frac{ax}{b}\right\rfloor(x\in R\wedge a,b\in Z^+)$ 不成立！

在平面直角坐标系中，对所有点做变换 $(x,y)\to(x-y,x+y)$，相当于将所有点绕原点逆时针旋转 $45°$。



如果懒得画出旋转后的点，也可以画出 $y = x$ 和 $y = -x$ 这两条直线，其中 $y = x$ 是 $x + y$ 轴（即变换后的坐标系中的 y 轴），$y = -x$ 是 $x - y$ 轴（即变换后的坐标系中的 x 轴），如图所示。



对于一个 01 串，要构造任意连续三个字符不能全部相等（i.e. 不能出现子串 000 或 111），只需要令 $\forall 2i \le n, a_{\{2i-1\}} \neg = a_{\{2i\}}$。例题：ZR2085。

$\mu * \mathrm{Id} = \varphi$。

给定正整数 $x, x+1, a, b, m$，求 $\max_{i \le a \wedge j \le b \wedge ix+j(x+1) \le m}(ix + j(x+1))$，可以先用 x 尽量填满，再用 $x+1$ 去尽量填满，然后可以尝试用 $x+1$ 去替换 x，计算出最大能替换的数量，每替换一次会使答案 $+1$。题目：CF1995B2。

Code:

```
ll u = min(a, m / x), v = min(b, (m - u * x) / (x + 1));
ll k = m - u * x - v * (x + 1);
ans = m - (k - min({k, u, b - v}));
```

令原序列 $a_i \geq a_i + a_{i+1}, a_{i+2} \geq a_{i+2} + a_{i+1}, a_{i+1} \geq -a_{i+1}$，相当于令前缀和序列 $s_i \geq s_i + a_{i+1}$，$s_{i+1} \geq s_{i+1} - a_{i+1}$，相当于交换前缀和序列中的 s_i 和 s_{i+1}。（2024.7.26，ZR2148 【21 ZR 联赛集训 day18】游戏）

字符串中对每个 r 求 右端点为 r 的 形如 AA 的 子串数量，题目：ZR2427 24ab-day1 呼啸浪潮。

CONTENT

sol1

枚举 A 的长度，记为 l，则所有的 AA 一定都会覆盖某个 il 和 $(i+1)l$。

对于 il 和 $(i+1)l$ 两个位置，使用 二分+哈希 或 SA 求出 $\text{LCSuf}(s[1,il], s[1,(i+1)l])$ 和 $\text{LCP}(s[il,n], s[(i+1)l,n])$，就可以计算出覆盖 il 和 $(i+1)l$ 的 AA 数量，使用差分加到对应的 r 上即可。

枚举 l 并枚举 il 和 $(i+1)l$ 的 TC 是 $O(n \ln n)$ 的，如使用 二分+哈希 则总 TC 是 $O(n \ln n \log n)$，如使用 SA 则总 TC 是 $O(n \ln n + n \log n)$。

CONTENT

sol2

SA + 最值分治 + 树状数组维护二维数点，两只 log。

详见 ZR 上的题解。

求满足条件的区间数量时，可以将区间 $[l,r]$ 映射到平面直角坐标系上的一个点 (l,r)，方便思考。注意到直线 $l=r$ 下方是没有点的，有时候可以利用这一性质来去除一维限制。例题：ZR2947 24ab-day8 洄游。（2024.8.26，ZR2947 24ab-day8 洄游）

$\sum_{i=2}^{n} \lfloor \ln_i n \rfloor = \sum_{i=2}^{n} \sqrt[i]{n} \leq O(\sqrt{n})\}$，因为 $\sqrt[3]{n}, \sqrt[4]{n}, \ldots$ 的量级都远小于 $\sqrt{n}$，可以忽略不计。（2024.9.1）

STATEMENT

要求＜线性基中 能凑出的异或和＞中＜＜满足＜某些给定的位为 0，某些给定的位为 ＞的数＞中的第 中的第 k 小的数＞＞。

SOLUTION

如果只有＜某些给定的位为 1＞的限制，将 线性基中的每个数 都 & 上＜这些位组合而成的数＞ 后 得到的数 都 插入 一个新的线性基 里面。然后求（**未完待续**）

不建议将＜多个 有大量内容重复的代码＞写在代码里，因为这样如果你要修改某个地方，那么就要对每处代码都修改一遍，容易少修改到某处代码。

建议使用 `#define`，其中可以使用 \ 来表示换行，如：

```
#define func0() for (int k = 0; k ⩽ 1; ++k) { \
  int j_nxt, l_nxt; \
  if (i > pn) j_nxt = f[j][l][k], l_nxt = g[j][l][k]; \
  else j_nxt = ((j << 1) | k), l_nxt = (m % 2 == 1 ? 1 : 0); \
  if (l_nxt < pn) { \
    eadd(Q[j_nxt][l_nxt][0], P[j][l][0]); \
  } \
}

#define func1(p) for (int k = 0; k ⩽ p; ++k) { \
  int j_nxt, l_nxt; \
  if (i > pn) j_nxt = f[j][l][k], l_nxt = g[j][l][k]; \
  else j_nxt = ((j << 1) | k), l_nxt = (m % 2 == 1 ? 1 : 0); \
  if (l_nxt < pn) { \
    eadd(Q[j_nxt][l_nxt][k == p], P[j][l][1]); \
  } \
}
```

然后主函数：

```
for (int j = 0; j ⩽ S; ++j) {
  for (int l = 0; l ⩽ pn - 1; ++l) {
    if (P[j][l][0] > 0) {
      if (a[i] == '0' || a[i] == '?') {
        func0();
      }
      if (a[i] == '1' || a[i] == '?') {
        func0();
      }
    }
    if (P[j][l][1] > 0) {
      if (a[i] == '0' || a[i] == '?') {
        func1(0);
      }
      if (a[i] == '1' || a[i] == '?') {
        func1(1);
      }
    }
  }
}
```

或者定义 `lambda` 函数，像这样写：

```
for (int j = 0; j ⩽ S; ++j) {
  for (int l = 0; l ⩽ pn - 1; ++l) {
    if (P[j][l][0] > 0) {
      auto func = [&]() {
        for (int k = 0; k ⩽ 1; ++k) {
          int j_nxt, l_nxt;
          if (i > pn) j_nxt = f[j][l][k], l_nxt = g[j][l][k];
          else j_nxt = ((j << 1) | k), l_nxt = (m % 2 == 1 ? 1 : 0);
          if (l_nxt < pn) {
            eadd(Q[j_nxt][l_nxt][0], P[j][l][0]);
          }
```

```
        }
      };
      if (a[i] == '0' || a[i] == '?') {
        func();
      }
      if (a[i] == '1' || a[i] == '?') {
        func();
      }
    }
    if (P[j][l][1] > 0) {
      auto func = [&](int p) {
        for (int k = 0; k <= p; ++k) {
          int j_nxt, l_nxt;
          if (i > pn) j_nxt = f[j][l][k], l_nxt = g[j][l][k];
          else j_nxt = ((j << 1) | k), l_nxt = (m % 2 == 1 ? 1 : 0);
          if (l_nxt < pn) {
            eadd(Q[j_nxt][l_nxt][k == p], P[j][l][1]);
          }
        }
      };
      if (a[i] == '0' || a[i] == '?') {
        func(0);
      }
      if (a[i] == '1' || a[i] == '?') {
        func(1);
      }
    }
  }
}
```

不过定义 `lambda` 函数可能会带来较大的额外的时间开销，但是一般会被编译器优化掉（编译器会直接把 `lambda` 函数里的代码填进去），所以速度上和 `#define` 或 直接写多处重复代码 并没有任何区别。

第二章 (2024.9 ~ 2024.9)

一种经典反悔贪心：

Statement

给定 $\{a_n\},\{b_n\}$，有一个变量 s，给定 s 的初始值。 对于 $i:1\to n$（按顺序枚举），

- 你有两种选择：
- 选取 i 这一位置，让 $s\leftarrow s-a_i$。
- 不选 i 这一位置，让 $s\leftarrow s+b_i$。
- 做完这次操作后，你要保证 $s\geq 0$。

你要使得最终选取的位置的数量尽可能大，求出这个数量。

Solution

考虑用大根堆维护所有选的位置的 a_i+b_i。

核心思想是：对于每个位置，若选取当前位置后 $s\geq 0$ 则直接选取当前位置，否则若大根堆中最大的数大于当前位置的 a_i+b_i，则 选取当前位置 且 反悔地去除掉大根堆中最大的数对应的位置，并更新 s，答案不变。

具体实现比核心思想简略一些，很 simple，详见代码。

```
ans = 0;
For(i, 1, n) {
    s -= a[i]; // 假设先选取当前位置
    q.push(a[i] + b[i]); // 把当前位置加入大根堆
    if (s < 0) s += q.top(), q.pop(); // 若 s < 0，则选取当前位置，反悔地去除掉大根堆中最大的数对应的位置。如果当前位置的 a[i] + b[i] 本身就是最大的，则相当于不选当前位置，维持原方案不变。
    else ++ans; // 否则 s ⩾ 0，可以直接选取当前位置而不反悔地去除掉任何一个位置
}
```

（2024.9.3，ZR1276 【20 省选十联测 day9】Farm of Monsters）

三元组 (a,b,c) 使得 $a,b,c\le n\wedge a\mid b\wedge b\mid c$ 的个数不超过 $n\ln^2 n$。

PROOF

证明 设 $i=\frac{b}{a}$，则三元组的个数为

$$\begin{aligned}&\sum_{a=1}^{n}\sum_{i=1}^{\lfloor\frac{n}{a}\rfloor}\left\lfloor\frac{n}{ai}\right\rfloor\\=&\sum_{a=1}^{n}\sum_{i=1}^{n}\left\lfloor\frac{n}{ai}\right\rfloor\\\le&\sum_{a=1}^{n}\sum_{i=1}^{n}\left\lfloor\frac{n}{ai}\right\rfloor\\=&n\sum_{a=1}^{n}\sum_{i=1}^{n}\left\lfloor\frac{1}{ai}\right\rfloor\\=&n\sum_{a=1}^{n}\frac{1}{a}\sum_{i=1}^{n}\frac{1}{i}\\=&n\ln^2 n\end{aligned}$$

统计一个联通块里面的点和边的个数可以这么写：（边是无向边）

```
ll cnt_v, cnt_e;
bool vis_pre[N];
void dfs_pre(ll x) {
  vis_pre[x] = 1;
  ++cnt_v;
  for (ll i = H[x]; i > 0; i = E[i].nxt) {
    ll v = E[i].t;
    ++cnt_e;
    if (vis_pre[v]) continue;

    dfs_pre(v);
  }
}

cnt_e /= 2; // 别忘了给 cnt_e 除以 2!
```

注意最后统计答 案时 `cnt_e` 要除以 2！（2024.9.9，ZR2984 【24noip 十连测 day2】翻转 (rev)）

一棵树上，给定一个节点 u 和一个点集 S，你需要选择 S 内的一个点 v，最大化 $\mathrm{dist}(u,v)$。设这个最大值为 M，则对于点集 S 中的每条直径，都满足这条直径的两个端点中有一个端点离 u 的距离是 M。

根据这个结论，我们也可以实现在加点的过程中维护直径。具体地，只用维护任意一条直径的两个端点，加入新点时分别计算新点与直径两个端点的距离，若其中较大值大于当前的直径，则取这个值对应的那个端点和新点作为新的直径的两个端点。

时间复杂度 $O(n)$ 或 $O(n\log n)$，由求两点距离的时间复杂度而定。（2024.9.9，ZR2983 【24noip 十连测 day2】树 (tree)）

STATEMENT

给定一棵树，有若干次修改和查询，查询一个点 x 的答案可以由它的每一个儿子 v 的子树内的信息之和合并而来（也就是说可以通过遍历 x 的所有儿子 v 来回答一个对 x 的查询）。（例：ZR1221 【20 省选十联测 day2】LCA on Tree，弱化后的题意是：对 x 的询问的答案是 $\sum_{v \text{ is a son of } x}(s_v(\mathrm{sz}_x-\mathrm{sz}_v))$，其中 $s_x=\sum_{u\text{在}x\text{ 的子树里}} w_u$，修改操作会单点修改 w_x）

SOLUTION

考虑我们有两种暴力：

- 暴力 1：修改操作时枚举 x 的祖先 k，计算对 fa_k 的答案造成的贡献，并记录到 fa_k 的答案上。
- 暴力 2：询问时枚举 x 的所有儿子 v 并计算答案。

考虑将这两种暴力结合起来。具体的，重链剖分，修改时我们只对祖先路径上的轻边 $k-\mathrm{fa}_k$ 修改，询问时单独计算 x 的重儿子的贡献，将它与 x 原本维护的答案合并起来作为询问的答案。

由于一个点祖先路径上的轻边条数是 $O(\log n)$ 的，故时间复杂度为 $O(m\log n)$（m 为操作个数）。

（2024.9.12，ZR1221 【20 省选十联测 day2】LCA on Tree）

如果 $1\sim i$ 的前缀和会爆 `int`，但是能保证查询的一段数的和不会爆 `unsigned int`，那么可以使用 `unsigned int` 来存储前缀和数组，因为查询时两个前缀和相减后得到的答案会被减回 `unsigned int` 的值域里，得到的答案仍然是正确的。（2024.9.15，ZR2994 [24noip 十连测 day3]茵蒂克丝）

对于 dp：

- 转移式：$f_{i,j}=\sum_{k=j-1}^{i-1} f_{i-1,k}(1\le j\le i)$（或者说 $f_{i,j}\to f_{i+1,k}(1\le k\le j+1)$）
- 初始条件：$f_{1,1}=1, f_{i,0}=0$

有结论：$\sum_{j=0}^{i} f_{i,j} = f_{i+1,1} = C_i$。（$C_i$ 为卡特兰数的第 i 项。假设卡特兰数的下标从 0 开始，即 $C_0 = 1, C_1 = 1, C_2 = 2, C_3 = 5, \ldots$）

> Proof
>
> 考虑其组合意义，$f_{i,j}$ 表示有 i 个左括号（不论是否匹配）、j 个未匹配的左括号的括号序列的个数，使得其添加 0 或正整数个右括号后能成为合法的括号序列。从 $f_{i,j}$ 转移到 $f_{i+1,k}$ 的组合意义就是先添加一个左括号，再添加 $j+1-k$ 个右括号。

于是对于 $f_{i,j}$，可以将其添加 j 个右括号补全为恰好有 i 个左括号和 i 个右括号的合法的括号序列。所以 $\sum_{j=0}^{i} f_{i,j} = C_i$。（2024.9.16，ZR2997 [24noip 十连测 day3]食蜂操祈）

第三章 (2024.10 ~ 2024.11)

看到中位数（假定中位数为第 $\lfloor \frac{n+1}{2} \rfloor$ 小的元素），想到：枚举 mid，给元素赋权 $\mathrm{val}_x = \begin{cases} 1 \text{ if } x > \mathrm{mid} \\ -1 \text{ if } x \leq \mathrm{mid} \end{cases}$，当 $\sum \mathrm{val}_x \leq 0$ 时，**中位数** $\leq$ mid；当 $\sum \mathrm{val}_x > 0$ 时，**中位数** $>$ mid。（2024.10.1）

对于边权为 1 的无向图，它的 bfs 生成树满足每条横叉边的两个节点的深度差为 0 或 1。（2024.10.4）

对于任意正整数序列 A, B，$\prod A_{\{i\}} + \prod B_{\{i\}} = \prod \min\left(A_{\{i\}}, B_{\{i\}}\right) + \prod \max\left(A_{\{i\}}, B_{\{i\}}\right)$ 成立当且仅当 $\forall i, A_{\{i\}} \leq B_{\{i\}}$ 或 $\forall i, A_{\{i\}} \geq B_{\{i\}}$。

(2024.10.7，ZR2963 Heavenly Altitudes)

$n (2 \mid n)$ 阶完全图的完美匹配数为

$$(n-1) \times (n-3) \times (n-5) \times \ldots \times 1 = (n-1)!! = \frac{n!}{2^{\frac{n}{2}} \times (\frac{n}{2})!}$$

(!! 是双阶乘)

(2024.10.10，ZR1498 20AB-day3 Funny Cost)

有向图判断有没有环可以用 dfs 实现，注意实现时要开两个 `bool` 数组，一个存历史上有没有访问过，另一个存是否在栈里，实现如下：

```
bool ost[N], vis[N]; // ost 表示是否在栈里，vis 表示历史上是否访问过
void dfs(int x) {
  if (ost[x]) {
    printf("Yes");
    exit(0);
  }
  if (vis[x]) {
    return;
  }
  vis[x] = 1;
  ost[x] = 1;

  for (int v : E[x]) {
    dfs(v);
```

```
  }

  ost[x] = 0;
}
```

注意必须要存历史上是否访问过，否则时间复杂度会退化到指数级别。（2024.10.11，P2444 [POI2000] 病毒）

封装过的手写哈希表:

```
template<typename K, typename T>
struct hsh_map {
  static const ll Hsh = 1919839;
  static const ll S = 1e7 + 10;
  ll num, H[Hsh], nxt[S];
  K X[S]; T Y[S];

  inline void clr() {
    for (ll i = 1; i ⩽ num; ++i) H[X[i] % Hsh] = 0;
    num = 0;
  }
  inline T& operator [] (const K &x) {
    ll p = x % Hsh;
    for (ll i = H[p]; i > 0; i = nxt[i]) {
      if (X[i] == x) return Y[i];
    }

    ++num;
    X[num] = x;
    Y[num] = T();
    nxt[num] = H[p];
    H[p] = num;
    return Y[num];
  }
  inline bool count(const K &x) {
    ll p = x % Hsh;
    for (ll i = H[p]; i > 0; i = nxt[i]) {
      if (X[i] == x) return 1;
    }
    return 0;
  }
  inline T at(const K &x) {
    ll p = x % Hsh;
    for (ll i = H[p]; i > 0; i = nxt[i]) {
      if (X[i] == x) return Y[i];
    }
    return T();
  }
};
```

注意当 `T` 的类型不是整数时，要重载运算符 `%` 作为哈希函数，例如:

```
inline ll operator % (const pll &p, ll Hsh) {
  return (p.first * 13331 % Hsh + p.second) % Hsh;
}
```

使用上面的手写哈希表，我们也可以实现只支持插入和判断是否存在的简易版 `set`，如下:

```
template<typename T, typename Container = hsh_map<T, bool> >
struct hsh_set {
  Container c;
  inline void clr() {
    c.clr();
  }
  inline void insert(const T &x) {
    c[x];
  }
  inline bool count(const T &x) {
    return c.count(x);
  }
};
```

(2024.10.11, ZR1496 20AB-day3 Good Subsegments)

封装过的邻接表:

```
template<typename T>
struct ACN {
  static const int fm = /**/;
  static const int M = fm + 10;
  int num, H[N], nxt[M];
  T E[M];

  inline void add(int x, const T &y) {
    ++num;
    E[num] = y;
    nxt[num] = H[x];
    H[x] = num;
  }

  struct iterator {
    ACN<T> &ptr;
    int i;
    inline operator int() {
      return i;
    }
    inline T operator * () {
      return ptr.E[i];
    }
    inline void operator ++ () {
      i = ptr.nxt[i];
    }
  };

  inline iterator operator [] (int x) {
    return (iterator){*this, H[x]};
  }
};
```

遍历时要这样写:

```
for (auto it = E[x]; it; ++it) {
  int v = *it;
  // do something...
}
```

或者要想使用 `for` 范围循环，可以这样写：

```
template<typename T>
struct ACN {
  static const int fm = /*∗/;
  static const int M = fm + 10;
  int num, H[N], nxt[M];
  T E[M];

  inline void add(int x, const T &y) {
    ++num;
    E[num] = y;
    nxt[num] = H[x];
    H[x] = num;
  }

  struct iterator {
    ACN<T> &ptr;
    int i;
    inline operator int() {
      return i;
    }
    inline T operator * () {
      return ptr.E[i];
    }
    inline void operator ++ () {
      i = ptr.nxt[i];
    }
  };

  struct iterator_range {
    ACN<T> &ptr;
    int x;
    inline iterator begin() {
      return (iterator){ptr, ptr.H[x]};
    }
    inline iterator end() {
      return (iterator){ptr, 0};
    }
  };

  inline iterator_range operator [] (int x) {
    return (iterator_range){*this, x};
  }
};
```

然后就可以直接像下面这样遍历：

```
for (int v : E[x]) {
  // do something...
}
```

（2024.10.11，P2414 [NOI2011] 阿狸的打字机）

对于一个 01 序列，要找到其中任意一组相邻的位置使得这两个位置上的数不同，可以找到最大的与 $s[n]$ 不同的位置（记为 ind），则 ind 与 ind $+1$ 就是一组解。因为 $[\text{ind}+1, n]$ 都与 $s[n]$ 相同，而 $s[\text{ind}]$ 与 $s[n]$ 不同，也就是 $s[\text{ind}]$ 与 $s[\text{ind}+1]$ 不同。

例题：

STATEMENT

对于给定的参数 a,b，定义 $f(x)=(x\oplus a)-b$，其中 $\oplus$ 表示异或符号。

给定 n 个变量，给出 m 组询问，对于每组询问，给定 a,b，请你输出一个 i，满足 $i\in[1,n)$，且有 $f(x_i)\times f\left(x_{\{i+1\}}\right)\le 0$，如果无解则输出 -1，如果有多组解输出任意一个即可。（https://mna.wang/contest/1639/problem/1）

SOLUTION

首先，如果存在 $i\in[1,n]$ 使得 $f(x_i)=0$，则直接将它与任意一个与它相邻的位置作为答案即可。

否则，将 $f(x_i)<0$ 看做 0，将 $f(x_i)>0$ 看做 1，则构成了一个长度为 n 的 01 序列。

我们只要找到最后一个与 $f(x_n)$ 的正负性不同的数即可。用 Trie 求出答案。

时间复杂度 $O(n+m)\log V$。（这个做法是从 lottle 那里学来的，%%%lottle）

（2024.10.18，https://mna.wang/contest/1639/problem/1）

一个局面的 SG 值小于等于其剩余能操作的次数的上限。（2024.10.28，ZR3027 分手快乐）

Model：有一颗树，求有多少排列满足 $P_x<P_{\mathrm{fa}_x}$。 Sol：这个问题可以简单的树形 dp 得到一个式子 $\frac{n!}{\prod \mathrm{sz}_x}$。（2024.10.31，[ZR3061 不如不见](http://zhengruioi.com/problem/3061)）

Model：对于一张 DAG，每次操作随机一个剩余的点 x，并删除 x 和所有 x 能到达的点。问将图删光的操作次数的期望。 考虑将问题转化为：随机一个排列 P，第 i 次操作若 P_i 未被删除则＜删除 P_i 和所有 P_i 能到达的点，并记入次数＞，如果 P_i 已经被删除则不做任何事。 由期望的线性性可得答案即为＜＜每个点被记入次数的概率＞之和＞。考虑点 x 被记入次数的概率即为＜＜P 中由＜x 和所有能到达 x 的点＞组成的子序列＞中 x 是第一个元素＞的概率。记能到达 x 的点（包含 x）的数量为 cnt_x，则答案即为 $\sum\left(\frac{1}{\mathrm{cnt}_x}\right)$。（2024.10.31，ZR3062 一丝不挂）

$[x=1]=\frac{2^x \bmod 4}{2}(x\ge 1)$。 这个 trick 经常被用在只要求答案 $\bmod 2$ 的时候。例如，如果只要求 $(\sum[\star=1])\bmod 2(\star\ge 1)$ 的结果，那么 $\frac{(\sum 2^\star)\bmod 4}{2}$) 即为答案。然后再通过组合意义等将 $\sum 2^\star$ 转换掉即可。（2024.11.9，ZR3069 城市）

有一些区间，则一个区间不被任何其他区间包含的充要条件是：＜将所有区间以 l 从小到大排序，若 l 相等则按 r 从大到小排序＞后，该区间的 r 是前缀中严格最大的（i.e. 对于所有在该区间前面的区间，其 r 小于该区间的 r）。

~~有一些区间，则一个区间不包含任何其他区间的充要条件是：＜将所有区间以 l 从小到大排序，若 l 相等则按 r 从小到大排序＞后，该区间的 r 是前缀中不严格最大的（i.e. 对于所有在该区间前面的区间，其 r 小于等于该区间的 r）。~~（2024.11.9，ZR3070 数据结构）

树上，距离的奇偶性 -> 黑白染色。（2024.11.11，ZR3045 白雪赞歌）

对于一个 k 进制下的数 x，设其在 k 进制下所有数位之和为 p，则有 $x \equiv p(\mathrm{mod}(k-1))$。

Proof

设 $x = a_0 + a_1 k + a_2 k^2 + a_3 k^3 + ...$，则 $p = a_0 + a_1 + a_2 + a_3 + ...$，

$$x - p = a_1(k-1) + a_2(k-1)(k+1) + a_3(k-1)(k^2+k+1) + ... \ = \sum_{i=1}^{\infty}\left(a_{i(k-1)}\sum_{j=0}^{i-1}(k^j)\right)$$

也即 $x - p \equiv 0(\mathrm{mod}(k-1)) \Longrightarrow x \equiv p(\mathrm{mod}(k-1))$。

（2024.11.12，ZR3099 可爱喵喵有力量）

一个括号序列的经典做法：见 ZR3056 新·括号王国 题解的第一句话：

Solution

假设确定了 a_i，那么求价值可以理解成如下过程：选择第 1 个位置，然后每次将第 $2i$ 个和第 $2i+1$ 个位置放入优先队列，再从优先队列中选出最大的值并弹出作为价值。

（2024.11.13，ZR3056 新·括号王国 & ZR3101 豪宅集训）

序列 a **严格**单调递增 $\iff$ 序列 $b_i = a_i - i$ **不严格**单调递增。（2024.11.23，CF713C Sonya and Problem Wihtout a Legend）

$\binom{n}{m} \bmod 2 = [m$ **的二进制表示** $\subseteq n$ **的二进制表示**$]$。（2024.11.28）

---

x^2 可以转换为：有 x 个盒子，有一个红球和一个蓝球，每种球都可以放在任意一个盒子里，两种球之间独立，放置球的方案数。$x^3, x^4, ...$ 同理。（2024.11.29）

$a_i \geq a_{i-1} + a_{i+1} - a_i$ 等价于 $\mathrm{swap}(d_i, d_{i+1})$（$d$ 为 a 的差分数组）。（2024.11.29，[P7962 [NOIP2021] 方差](https://www.luogu.com.cn/problem/P7962)）

第四章 (2024.12 ~ 2025.7.16)

$\sum_{i=0}^{n}\binom{n}{u}(-1)^i = [n=0]$。（2024.12.4，P11363 [NOIP2024] 树的遍历）

点 x 在路径 $u \rightsquigarrow v$ 上 $\iff \mathrm{lca}(u,v) = x \vee [u \in x\text{'s subtree}] \neq [v \in x\text{'s subtree}]$。（2024.12.17，17，P5344 【XR-1】逛森林）

某些有关背包的方案计数 dp 题，如果合并容量那一维是 $O(n^2)$ 的，可以把容量那一维用生成函数代替，即设 $f = \sum \mathrm{dp}_i \times x^i$，然后任取 **容量上限** $+1$ 个 x 代入并用没有容量一维的 dp 计算出结果，再将所得的结果用拉格朗日插值求出对应的多项式，即为要求的 dp 数组。这样做可以在时间复杂度中省去一个 n。（2025.1.4，ZR2014 [zr 联赛十连测 day3] 修建道路）

---

mex 不容易快速插入或快速合并，但可以快速删除。假如我们要求一个集合 S 的 mex，已知集合 T 的 mex（$S \subseteq T$），且支持快速查询 S 里是否存在某个元素。做法是设 ans 为 T 的 mex，我们遍历 $T \setminus S$ 中的元素 x，若 $x \notin S$，则将 ans 对 x 取 min。（2025.1.6，ZR1249【省选特训 5】教科书般的亵渎）

若操作是在序列中第 u 个元素前插入一个元素 k，且可以离线，则可以使用**树状数组上二分**来求出每个元素在最终序列中的位置，这样就不用写平衡树了。具体的，维护一个初始时每个位置都是 1 的树状数组，我们时光倒流，对于在 u 个元素前插入的操作，我们用树状数组上二分求出树状数组中第 u 个 1，设它的位置为 p，则该操作插入的元素在最终序列中的位置为 p，然后我们将树状数组上位置 p 的值 $-= 1$。（2025.1.6，ZR1249【省选特训 5】教科书般的亵渎）

---

37. 若 $x \geq 3$，则 $\varphi(x)$ 为偶数。（2025.1.7，ZR2022 [zr 联赛集训 day14]函数）

若 $a \perp b$，设 a 的既约剩余系为 A，b 的既约剩余系为 B，则 ab 的既约剩余系为 $\{pb + qa \mid p \in A, q \in B\}$（定义 x 的既约剩余系为 $\{k \mid 1 \leq k \leq n \wedge k \perp x\}$）

> Proof
>
> 显然 $pb \perp a$，$qa \perp b$。则 $\gcd(pb + qa, a) = \gcd((pb + qa) \bmod a, a) = \gcd(pb, a) = 1$，即 $(pb + qa) \perp a$。同理有 $(pb + qa) \perp b$。所以 $(pb + qa) \perp ab$。

（2025.1.7，ZR2022 [zr 联赛集训 day14]函数）

在二维平面上，与 (x, y) 点积为定值 k 的所有点 (a, b) 构成一条直线。证明可以考虑直线的一般式方程 $ax + by = c$ 也可以看做 (x, y) 与 (a, b) 的点积为 c。（2025.1.10，ZR1501 20AB-day4 Exercise，from PPP）

$d(xy) = \sum_{i \mid x} \sum_{j \mid y} [\gcd(i, j) = 1]$。证明：对于质数 p，设其在 x 里的次数为 k_1，在 y 里的次数为 k_2，则其对 $d(xy)$ 的贡献为乘以 $(k_1 + k_2 + 1)$。容易发现在枚举 i, j 的过程中，$\gcd(i, j) = 1$ 的 (i, j) 一定满足 i 中 p 的次数在 $[0, k_1]$ 且 $p \nmid j$，或 j 中 p 的次数在 $[0, k_2]$ 且 $p \nmid i$，容易发现 p 对 $\gcd(i, j) = 1$ 的 (i, j) 个数的贡献为乘以 $(k_1 + k_2 + 1)$。（2025.1.12，P3327 [SDOI2015] 约数个数和）

---

41. $\varphi(xy) = \frac{\varphi(x)\varphi(y)\gcd(x,y)}{\varphi(\gcd(x,y))}$，证明可以考虑对单个质数 p 的贡献 分讨 x, y 分别是否整除 p。（2025.1.12）

所有点的度数相同的二分图必定存在完美匹配。证明可以使用 Hall 定理。（2025.1.27，xtq 讲课）

求一个集合内点两两斜率的最大值，可以按 x 轴排序后求相邻点的斜率的最大值。（最小值亦即可）（2025.2.15，ZR3026 灯，ZR2318 [22 AB Day10] 签到）

某一些划分集合的状压 dp 问题，如果集合之间的顺序无关的话，并且确定了剩余的元素要划分为多少个集合，可以钦定每次新加的集合为大小最小的那个，从而其大小 $\le \lfloor |U \setminus S| \rfloor$，来减少复杂度，可以将时间复杂度从 $O(3^n)$ 减少一些。（2025.2.15，ZR2320 [22 AB Day10] 简单

一些高精度题，如果需要高精度取模、高精度除法等复杂操作，可以先求出答案分别模足够多个 `int` 以内的小质数的结果，再使用中国剩余定理合并，且使用 CRT 合并的过程中只需要使用 **高精度加减法、高精度乘单精度、高精度数大小比较** 这些简单操作。见这篇题解：https://www.luogu.com.cn/article/0jh3mdy4（2025.3.6，P2144 [FJOI2007] 轮状病毒）

$$\gcd(ka, a+b) = \gcd(a,b) \times \gcd\left(k, \frac{a+b}{\gcd(a,b)}\right)$$

Proof

感性理解可以考虑将数整体除以 $\gcd(a,b)$ 进行缩放。

或者推式子：

$$\begin{aligned}\gcd(ka, a+b) &= \gcd(a,b) \times \gcd\left(k\frac{a}{\gcd(a,b)}, \frac{a+b}{\gcd(a,b)}\right)\\ &= \gcd(a,b) \times \gcd\left(k, \frac{a+b}{\gcd(a,b)}\right)\end{aligned}$$

（最后一步是因为：设 $u = \frac{a}{\gcd(a,b)}, v = \frac{b}{\gcd(a,b)}$，因为 u, v 互质，所以 $u, u+v$ 互质，所以 u 的每个大于一的因子都无法整除 $u+v$、无法造成贡献，所以可以在 ku 中去除 u）

（2025.3.6，[SP419 TRANSP - Transposing is Fun](https://www.luogu.com.cn/problem/SP419)）

---

47. $x^{\overline{k}} = (-x)^{\underline{k}}(-1)^k$，同理 $x^{\underline{k}} = (-x)^{\overline{k}}(-1)^k$。（2025.3.11）

$xy = \binom{x+y}{2} - \binom{x}{2} - \binom{y}{2}$。（2025.4.1，P6295 有标号 DAG 计数）

对于模数为质数的情况，计算一个数乘 2 的逆元可以这么写：

```
constexpr inline ll div2(ll x) { return (x + (x & 1) * MO) >> 1; }
```

也就是在 $2 \mid x$ 时，答案即为 $\frac{x}{2}$；在 $2 \nmid x$ 时，答案为 $\frac{x+P}{2}$。

（虽然在 P4717 【模板】快速莫比乌斯/沃尔什变换 (FMT/FWT) 上的测试结果显示，疑似直接乘 2 的逆元会被编译器优化到跟这玩意一样快。）。

同理，也可以用同样的方法计算乘 $3, 4, 5, \ldots$ 的逆元，计算一个数乘 k 的逆元的时间复杂度是 $O(k)$ 的（因为要枚举模 k 的所有余数）。

一般只建议用这玩意加速乘 2 的逆元，否则可能比直接乘逆元并取模更慢。

（2025.4.21, P4717 【模板】快速莫比乌斯/沃尔什变换 (FMT/FWT)

Statement

对于一个长为 n 的序列，一个子区间在一定条件下是**好的**，且若一个区间是**好的**，则它的所有子区间都是**好的**。要求把整个序列划分为尽可能多个不交子区间，求子区间数量的最大值。

假设可以在 $O(L)$ 的时间内 check 一个长为 L 的区间是否是好的，则可以在 $O(n)$ 的时间复杂度内解决该问题。具体的，当我们划分完 $[1,i)$ 时，要快速求出以 i 为左端点的最大的**好的**区间的右端点。注意到直接二分会时间爆炸，我们先倍增，对于 $j=0\to\log n$ 依次 check 最大的**好的**区间的长度是否在 $[2^j,2^{j+1})$ 内，也就是找到最小的 j 使得长为 2^j 的区间是**好的**，长为 2^{j+1} 的区间不是**好的**。求出区间长度的范围后，再在 $[2^j,2^{j+1})$ 这个范围内二分即可。容易发现倍增和二分的时间复杂度都为 $2^0+2^1+\ldots+2^j\le 2^{j+1}\le 2\times$ **最大的好的区间的长度**，由于我们在给序列分段，因此总时间复杂度为 $O(n)$。（2025.5.8，ZJCPC2025 K. Wavelike Finding）

原树上两点 x,y 的路径上 一定包含 点分树上 x,y 的 LCA。 **推论：** 对于一个原树上的连通块 S，设 p 为 S 中在点分树上深度最小的点，则点分树上 p 的子树一定包含点集 S。
（2025.5.11，[P5311 Ynoi2011 成都七中]）

$O(n)$ 求排列逆序对数的奇偶性：将排列视为置换，则答案为 $(n-$ **置换的环数**$)\bmod 2$。 证明：交换任意两数，逆序对奇偶性 ^= 1。对于一个大小为 len 的置换环要交换恰好 len -1 次使它归位，于是总共要交换 $n-$ **置换的环数** 次，于是排列逆序对数的奇偶性为 $(n-$ **置换的环数**$)\bmod 2$。（2025.5.12，从 UOJ 群友处学到的）

若一个游戏的 SG 函数的值只有 0/1，则游戏时无论哪一方操作哪个游戏，最后的结果（谁胜谁负）一定都是固定的。于是这种游戏只需要计算操作成无法行动的局面所需要的步数即可判断胜负。

Proof

若一个局面的 SG 值为 1，则其后继状态的 SG 值必定只有 0；若一个局面的 SG 值为 0，则其后继状态的 SG 值必定只有 1。于是走向每个后继状态后的 SG 值都是相同的，也就是说怎么操作与操作后的 SG 值无关。（2025.5.25，CF1906G Grid Game 2）

要求两个 序列内部元素互不相同 的序列 a,b，它们之间能凑出多少对相同的数，可以将 a,b 内元素离散化使 a 内元素为 $1\sim n$ 的整数，然后查询 b 中在 $[1,n]$ 的数有多少个即可。
（2025.5.30，ZR3221 人员调度 CJR 的做法）

`~x` 等价于 `x - 1`。（2025.6.23，阅读 一种简单的 O(nlogn) 预处理 O(1) 查询静态树上链半群的方法 文中的代码时学来的）

若要计算 $\frac{x}{k}\bmod P$，其中 $\gcd(x,P)\neq 1$，无法使用逆元。（x 是一个整数，保证能够整除 k）这种情况下可以先计算 $x\bmod(kP)$ 的结果，再将其直接除以 k，即为所求的 $\frac{x}{k}\bmod P$。
（2025.7.4，P3726 [AHOI2017/HNOI2017] 抛硬币 的 这篇题解）

第五章 (2025.7.17 ~ ?)

支持区间加（从第一项开始的）斐波那契，询问区间和，对 998244353 取模。

等价于区间加等比数列（矩阵）

（2025.7.18，当天 wangzeyuan 的 C 班课件（标题为《线段树进阶》））

有树，点带权，设 s_x 表示 x 子树内的点权和。则整棵树中满足 $2s_x \geq s_{\text{root}}$ 的深度最大的点 x 一定是树的重心，且另一个重心如果存在则一定是 x 的父亲。

证明大概考虑：整棵树去掉 x 子树后的部分，点权和 $\leq \frac{1}{2}s_{\text{root}}$；$x$ 的每个儿子的子树，其 s 值也 $< \frac{1}{2}s_{\text{root}}$（若 $\geq \frac{1}{2}s_{\text{root}}$ 则该儿子深度大于 x，与 x 是满足 $2s_x \geq s_{\text{root}}$ 中的深度最大的点矛盾）。

（2025.7.21，P3345 [ZJOI2015] 幻想乡战略游戏）

有根树上要求 $\text{lca}(x, y)$ 到根节点的路径上的信息和，可以在已清空的数据结构上将 x 到根节点的路径上的信息激活（即：将路径上每个点的信息设为它应有的信息，将树上其他点的信息设为 0），再查询 y 到根节点的路径上的信息，这时候查询到的信息只会包含 $\text{lca}(x, y)$ 到根节点上的信息和。

（2025.7.21，P3345 [ZJOI2015] 幻想乡战略游戏，题解代码里使用的 trick，经 incra 点拨获得的 trick）

贪心做法要证明其正确性，可以通过证明 所有取到最优解的【结构/答案】 都可以通过某些变换 转换为 该贪心做法得到的【结构/答案】。（2025.7.23，ZR3244 宝宝巴士之小猫过河）

想一想把方案取反后的结果，例如 P6846 [CEOI 2019] Amusement Park 中，有：

> Content
>
> 首先发现，若用 x 次变为了 DAG，那么一定可以用 $m - x$ 次将其变为它的反图。于是题意转化为给无向边定向成 DAG 的方案数乘以 $\frac{m}{2}$。

（2025.7.26，P6846 [CEOI 2019] Amusement Park）

枚举 p 的超集：

```
for (int q = p | (p + 1); q < (1 << n); q = (q + 1) | p)
```

（2025.7.26，ZR2099【21 ZR 联赛集训 day4】删边方案）

求出点双后要求每条边所属的点双，可以这么写：

```
int n, m, U[N], V[N]; vector<int> E[N];
int dfn[N], lw[N], dfn_p;
int stk[N], *stk_l;
int idx; vector<int> F[N];
void tarjan(int x, int ek) {
  dfn[x] = lw[x] = ++dfn_p;
  *++stk_l = x;
```

```
  for (int i : E[x]) if (i ≠ ek) {
    int v = U[i] ^ V[i] ^ x;
    if (!dfn[v]) {
      tarjan(v, i);
      Min(lw[x], lw[v]);
      if (lw[v] ≥ dfn[x]) {
        ++idx, F[idx].clear();
        while (stk_l > stk) {
          int u = *stk_l--;
          F[idx].push_back(u), F[u].push_back(idx);
          if (u == v) break;
        }
        F[idx].push_back(x), F[x].push_back(idx);
      }
    }
    else Min(lw[x], dfn[v]);
  }
  return;
}

int fa[N];
void dfs_fa(int x, int fr) {
  fa[x] = fr;
  for (int v : F[x]) if (v ≠ fr) dfs_fa(v, x);
  return;
}

void proc_edges() {
  for (int i = 1; i ≤ m; ++i) {
    int u = U[i], v = V[i], p;
    if (fa[u] == fa[v]) p = fa[u];
    else if (fa[fa[u]] == v) p = fa[u];
    else p = fa[v];
    // (u, v) 是点双 p 内的一条边
  }
}
```

另外，这种写法会将 割边 及 它连接的两个点 视为一个点双。它还会建出圆方树，使得直接遍历方点的邻居（也就是 `F[p]`）就可以遍历该点双内部的点。

（另外我记得还有一种写法在 tarjan 中就求出每条边所属的点双，但是我忘了/ll）

（2025.7.30，P8456 「SWTR-8」地地铁铁）

对于正整数 x, y，若 $x \geq 2y$，则一定存在一个 k 使得 $x > 2^k \geq y$。（2025.7.31，P8265 [Ynoi Easy Round 2020] TEST_63）

求点集在树上的最小斯坦纳树大小，只需要将点按照 dfn 排序之后，求相邻点的距离之和再除以 2，即 $\frac{\sum_{i=1}^{k} \mathrm{dis}(p_i, p_{i \bmod k+1})}{2}$。（2025.8.3，QOJ5363 ZYB 的游览计划）

对于高斯消元，如果保证结果一定为整数，那么为了避免浮点数误差，可以在模意义下进行运算，只要保证计算过程中不会出现模数的倍数即可（所以模数取 998244353 或 $10^9 + 7$ 这种一般就够用了）。

可以把 `pair<int, unsigned int>` 压一起存成 `long long`，这样比较两个 `pair<int, unsigned int>` 和比较压位后的 `long long` 是等价的，且压位后的 `long long` 进行比较 会更快一些（因为 `pair<int, unsigned int>` 存储会先存 `unsigned int`，再存 `int`，就要考虑 `int` 的正负号带来的影响，不如直接比较 `long long` 来得快）。（2025.8.7, according to PPP）

拉插で使用点值求多项式系数时，如果复杂度瓶颈在拉插的 $O(n^2)$ 将点值转为多项式系数，可以考虑改成 NTT，因为 NTT 逆变换也是一种利用点值求系数的方法，且复杂度仅为 $O(n \log n)$，坏处只有 相对于拉插来说需要求 至多 2 倍数量 的点值。（2025.8.9，ZR3279 图）

有些树上背包，一次会选一整棵子树，这时可以考虑在 dfn 序上 dp， 把树上 dp 转换为序列 dp。有两种转移：

- 不选 dfn 序为 i 的点，相当于一个 $\mathrm{dp}[i-1] \to \mathrm{dp}[i]$ 的转移。
- 选一棵 dfn 序在 $[l, r]$ 的子树相当于一个 $\mathrm{dp}[l-1] \to \mathrm{dp}[r]$ 的转移。

（2025.8.10，ZR3289 连通块计数问题）

点集中的点数 − 点集中的边数 = 点集中的连通块个数 （2025.8.10）

括号匹配问题：

> Statement
>
> 有一个长为 n 的括号序列 $\{a_n\}$。
>
> 定义一对括号 a_i, a_j 能够匹配当且仅当 $i < j$，a_i 为左括号，a_j 为右括号。
>
> 每个括号至多只能在一个匹配中，求最多能匹配多少对括号。

> Solution
>
> 令左括号的权值为 1，右括号的权值为 -1，令 s_i 表示 $a_1, \ldots, a_i$ 的权值之和。则答案为 $\frac{n+s_n}{2} + \min_{i=0}^{n} s_i$。

> Proof
>
> 设 $p = \operatorname{argmin}_{i=0}^{n} s_i$，则容易发现不会有跨过 p 的匹配。
>
> 显然 $(p, n]$ 里的右括号都一定可以被 $(p, n]$ 里的左括号匹配，$[1, p]$ 里的左括号都一定可以被 $[1, p]$ 里的右括号匹配。
>
> 设 x 为 $[1, p]$ 里的左括号数量，y 为 $[1, p]$ 里的右括号数量。显然 $x + y = p, x - y = s_p$，故解得 $x = \frac{p+s_p}{2}$。由于 $[1, p]$ 里的左括号都一定可以被 $[1, p]$ 里的右括号匹配，所以 $[1, p]$ 内部产生的匹配数量为 $\frac{p+s_p}{2}$。类似地，可以求得 $(p+1, n]$ 内部产生的匹配数量为 $\frac{(n-p)-(s_n-s_p)}{2}$。
>
> 故总匹配数量为
>
> $$\frac{p+s_p}{2} + \frac{(n-p)-(s_n-s_p)}{2} = \frac{n+2s_p-s_n}{2} = \frac{n-s_n}{2} + s_p = \frac{n-s_n}{2} + \min_{i=0}^{n} s_i$$

（2025.8.10，ZR3288 括号匹配问题）

一种把点分治转为边分治的方法：

Solution

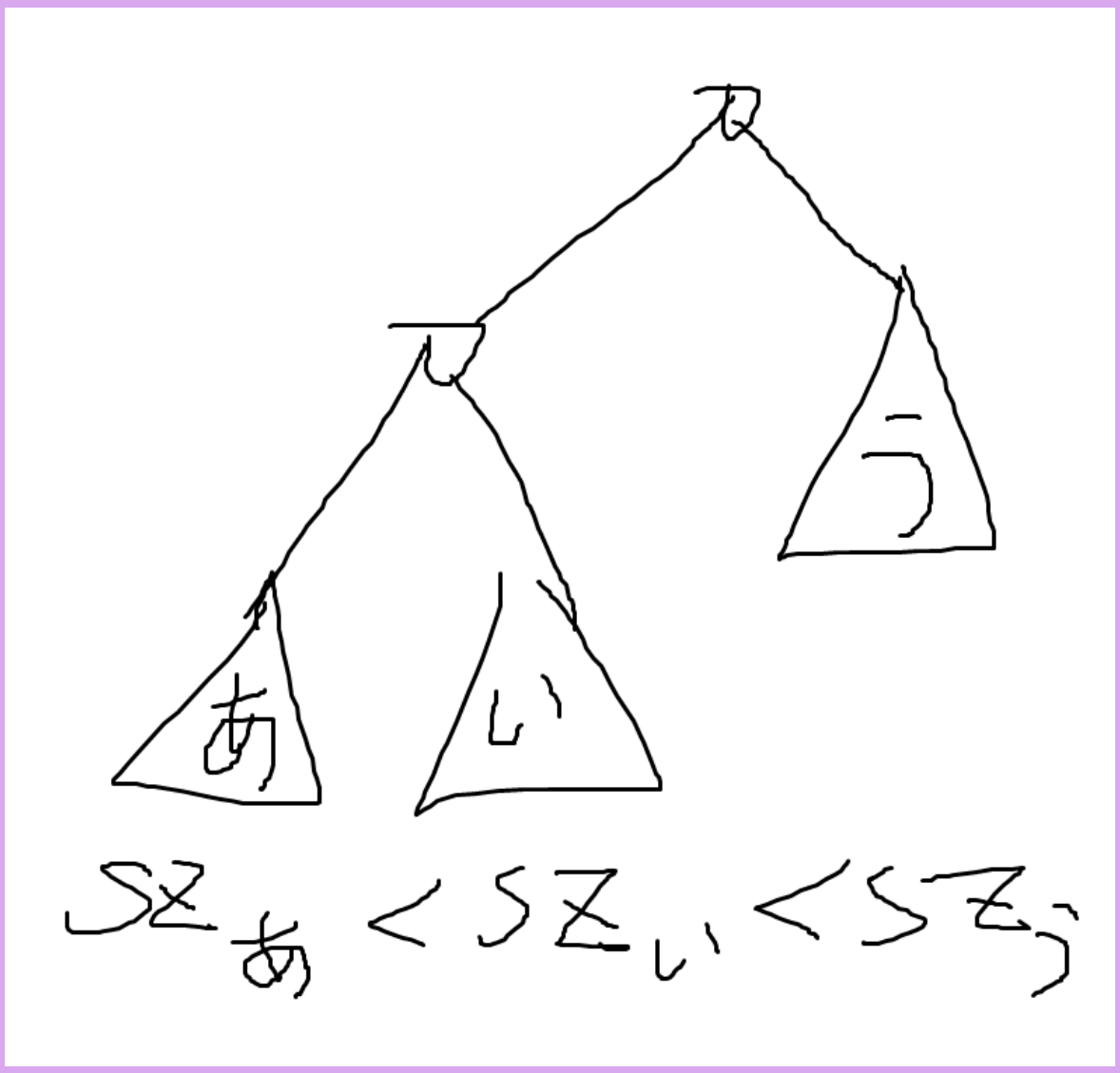
对分治子树进行哈夫曼合并，也就是：

每次取出最小的两个子树，将这两棵子树进行合并，合并完后将两棵子树当做一个整体（或者说当做一棵更大的子树，大小为两棵子树之和）放回子树堆里。这样每次会让子树数量减 1，不断执行上述操作直到只剩下一棵子树。

这样的复杂度比边分治多一个 log，也就是 $O(n \log^2 n)$ 的。证明如下：

Proof

我们只要证明每棵子树至多会被合并 $O(\log \textbf{子树数量})$ 次即可。



例如有如图所示的哈夫曼合并结构，有 あ、い、う 三棵子树，满足 $\text{sz}_{\text{あ}} < \text{sz}_{\text{い}} < \text{sz}_{\text{う}}$。

那么显然合并 あ、い 两棵子树时，由于 $\mathrm{sz}_{\text{あ}} < \mathrm{sz}_{\text{い}}$，$\mathrm{sz}_{\text{あ}}$ 的大小翻倍。

对于 い 这棵子树，虽然合并 あ、い 两棵子树时其大小没有翻倍，但是在合并 あい 的结合体 与 う 这棵子树时，由于 $\mathrm{sz}_{\text{い}} < \mathrm{sz}_{\text{う}}$，$\mathrm{sz}_{\text{い}}$ 的大小也翻倍了。

综上，一棵子树合并 $1 \sim 2$ 次后大小翻倍，故每棵子树至多会被合并 $2\log$ **子树数量** 次，也就是 $O(\log$ **子树数量**$)$ 次。

（2025.8.11，ZR3251 小 Z 的树）

对于计算一个字符串能按某些规则消成多少个本质不同的字符串的这类问题，可以推导出 s 能否消成 t 的判定流程，并根据这个写出计数 dp。

并且在推导出 s 能否消成 t 的判定流程之前，先推导出一个字符串能否被消为空有可能是有帮助的。

（2025.8.16，ZR3315 Compute）

一些问题可以建出最大流模型，然后使用最大流最小割定理转化为求最小割，再从最小割的问题模型转化出一个比较易做的问题。

（2025.8.16，ZR3314 Time）

消元时，可以选定一些未知数为**关键元**，将其他未知数都用 关键元加常数的线性表达式 表达出来（求出这些表达式的过程需要进行消元），然后把这些表达式代入其他方程，来消掉不是关键元的未知数，然后就变成了未知数集合为我们选定的关键元的方程组了，且方程组的秩不变。

虽然这个 trick 面对任意的给出的方程组并不能降低消元的复杂度，但是在一些题目上，你可以用这个 trick 结合题目中的方程组的性质降低复杂度，例如 P3706 [SDOI2017]硬币游戏 的这篇题解。

（2025.8.21，P3706 [SDOI2017] 硬币游戏）

可以用 `(!x - 1) & (MO - x)` 求 `x` 在模 `MO` 意义下的相反数（假设 $0 \le x < \mathrm{MO}$）。

（2025.8.21，《yhx-12243 的 NTT 究竟写了些什么（详细揭秘）》）

STATEMENT

维护一个 $x \times y$ 的数组，若干次修改或查询单点的值，x, y 都是 10^6 级别的。

用哈希表或 `std::map` 等显然可以在线维护，但是如果觉得哈希表常数太大了 或 不想写哈希表，且可以离线，那么可以这样做：

SOLUTION

把 x 相同的修改和查询放在一起处理，对于相同的 x，处理时只用开 y 这一维的桶即可。清空也是容易的，把用到过的位置清空即可。

并且这个做法可以拓展到任意维的情况，对于每一维把这一维相同的拉出来即可。例如，三维时，先把所有 x 相同的修改/查询放一起，对于单个 x，再把 y 相同的修改/查询放一起，然后用桶维护 z 即可。

并且这个做法可以拓展到最常见的 key 为较大整数的情况，用来代替常规的哈希表。只要设置 $O(n)$ 范围的 B，然后把 key 写成 B 进制数即可。例如，$\text{key} \le 10^{12}$ 时，相当于把一个数 key 拆成一个二元组 $\left(\left\lfloor \frac{\text{key}}{10^6} \right\rfloor, \text{key} \bmod 10^6\right)$，再用二维情况的上述做法处理即可。

（2025.9.3，P5607 [Ynoi2013] 无力回天 NOI2017）

等差乘等比数列求和公式：

（即求 $A = a + (a+b) \times p + (a+2\times b) \times p^2 + ... + (a+n\times b) \times p^n$ 的值）

$$(1) : A = a + (a+b) \times p + (a+2\times b) \times p^2 + ... + (a+n\times b) \times p^n$$

$$(2) : A \times p = a \times p + (a+b) \times p^2 + (a+2\times b) \times p^3 + ... + (a+n\times b) \times p^{n+1}$$

$$(1)-(2) : A \times (1-p) = a + b \times (p + p^2 + p^3 + ... + p^n) - (a+n\times b)p^{n+1}$$

$$A \times (1-p) = a + b \times p \times \frac{1-p^n}{1-p} - (a+n\times b) \times p^{n+1}$$

$$A = \frac{a}{1-p} + b \times \frac{p-p^{n+1}}{(1-p)^2} - \frac{(a+n\times b)\times p^{n+1}}{1-p}$$

（2025.9.4，CF908D New Year and Arbitrary Arrangement，来自于 这篇题解）

若选择 x 的 子树大小前 $\sqrt{n}$ 大的 所有儿子作为重儿子集合，则每个节点到根的链上的轻边数量均为 $O(1)$。

因为一个子树大小是 k 的点如果是它父亲的轻儿子，则他的父亲的子树大小则至少有 $O\left(kn^{\frac{1}{2}}\right)$。因此若一个节点上方有 p 条轻边，则顶端的子树大小将达到 $O\left(n^{\frac{p}{2}}\right)$。因此每个节点到根的链上的轻边数量均为 $O(1)$。这样莫队涉及的点的数量为 $O(n)$，时间复杂度为 $O(n\sqrt{m})$。

（2025.9.5，P6782 [Ynoi2008] rplexq，这篇题解）

要由排列想到二分图匹配。

（2025.9.9）

使用匿名 `namespace` 封装一个 **非 inline** 函数，开了 `-Wall` 的情况下，若这个函数没有被使用过则会报 warning，例如：

```
namespace {
  ll jc[N], ji[N], inv[N];
  void init_inv() {
    jc[0] = 1;
    for (int i = 1; i <= fn; ++i) jc[i] = 1ll * jc[i - 1] * i % MO;
    ji[fn] = get_inv(jc[fn]);
    for (int i = fn; i >= 1; --i) ji[i - 1] = 1ll * ji[i] * i % MO;
    inv[0] = 1;
    for (int i = 1; i <= fn; ++i) inv[i] = jc[i - 1] * ji[i] % MO;
    return;
```

```
  }
}
```

编译信息：

```
test.cpp:86:14: warning: ‘void MyNamespace::{anonymous}::init_inv()’ defined but not
used [-Wunused-function]
   86 |         void init_inv() {
      |              ^~~~~~~~
```

这个技巧可以用于避免「一些初始化函数忘了在主函数中调用」的情况。

注意，实测在 `void init_inv` 前面加上 `inline`，就不会报 warning!

（2025.9.16，CF961G Partitions）

$$[x^{n-m}]\frac{1}{(1-x)^{m+1}}=\binom{n}{m}$$

由此可得出：

$$\sum_{i=k}^{n}\binom{i}{k}\binom{n-i}{j}=\binom{n+1}{k+j+1}$$

PROOF

$$\binom{i}{k}=[x^{i-k}]\frac{1}{(1-x)^{k+1}}$$

$$\binom{n-i}{j}=[x^{n-i-j}]\frac{1}{(1-x)^{j+1}}$$

把这两个东西卷起来可以得到

$$\sum_{i=k}^{n}\binom{i}{k}\binom{n-i}{j}=[x^{n-k-j}]\frac{1}{(1-x)^{k+j+2}}=\binom{n+1}{k+j+1}$$

另外还有一个组合意义的推导：

PROOF

设有 $n+1$ 个物品按顺序排成一排，其中有 $k+j+1$ 个是特殊的。设 $k+j+1$ 个特殊物品中的第 $k+1$ 个物品是关键物品。

则 $\binom{n+1}{k+j+1}$ 显然是从 $n+1$ 个物品中选出 $k+j+1$ 个物品作为特殊物品的方案数。

$\sum_{i=k}^{n}\binom{i}{k}\binom{n-i}{j}$ 则是，枚举所有物品中排在关键物品前面的有 i 个物品，则 $\binom{i}{k}$ 和 $\binom{n-i}{j}$ 分别是「从排在关键物品前面的 i 个物品中选出 k 个作为特殊物品」和「从排在关键物品后面的 $n-i$ 个物品中选出 j 个作为特殊物品」的方案数，可以发现这种统计方式也不重不漏地统计了所有「从 $n+1$ 个物品中选出 $k+j+1$ 个物品作为特殊物品」的方案，故两式是相等的。$Q.E.D.$ 证毕。

（2025.9.16，P5825 排列计数 的 这篇题解）

二项式反演除了用 $O(n^2)$ 暴力求以外，还可以发现它是一个差卷积的形式，故可以用卷积在 $O(n\log n)$ 时间内求出。

具体地，假设有

$$g_k=\sum_{i=k}^{n}\binom{i}{k}f_i$$

那么

$$\begin{aligned}f_k&=\sum_{i=k}^{n}\binom{i}{k}(-1)^{i-k}g_i\\&=\frac{1}{k!}\sum_{i=k}^{n}i!g_i\times\frac{(-1)^{i-k}}{(i-k)!}\end{aligned}$$

由 $i-(i-k)=k$，容易发现是一个差卷积的形式。故卷积做即可。

（2025.9.16，P5825 排列计数）

输出构造方案的题，可以在代码里加 checker，如果自己输出的方案是错误的就 `assert(0)` 掉。

（2025.9.25，CF1680F Lenient Vertex Cover）

- $(a_{i-1},a_i,a_{i+1})\to(a_{i-1},a_{i-1}+a_{i+1}-a_i,a_{i+1})$ 作差分解决。
- $(a_{i-1},a_i,a_{i+1})\to(a_{i-1}+a_i,-a_i,a_{i+1}+a_i)$ 作前缀和解决。

（2025.9.25，CF1110E Magic Stones）

---

01 序列 想到 设 0 的权值为 -1，1 的权值为 $+1$，求出前缀和序列 想到 利用「前缀和序列形成的折线图」来刻画。

（2025.9.25，CF1458D Flip and Reverse）

在树上，令大小前 B 大的儿子为重儿子，其余点为轻儿子，则 $\sum_x x$ **的祖先中为轻儿子的节点的数量** 为 $O(n\log_{B+1}n)$ 级别的。

$B=1$ 时即为树链剖分。

当 $B=\sqrt{n}$ 时，$\sum_x x$ **的祖先中为轻儿子的节点的数量** 为 $O(n)$ 级别的。B 不一定要严格为 $\sqrt{n}$，也可以是 $n^{\frac{1}{3}},n^{\frac{2}{3}}$ 等等，都可以使 $\sum_x x$ **的祖先中为轻儿子的节点的数量** 成为 $O(n)$ 级别。实际使用中，也不需要取这么大的 B，令 B 为稍大一点的常数（例如 4 或 10），也可以使 $\sum_x x$ **的祖先中为轻儿子的节点的数量** 同样是近似 $O(n)$ 级别了，

（2025.10.1，P6782 [Ynoi2008] rplexq）

Erdős-Ginzburg-Ziv 定理 对于任意 $2n-1$ 个整数，总可以从中选出 n 个整数使它们的和为 n 的倍数。

（2025.10.4，CF1798F Gifts from Grandfather Ahmed）

对于图 (V, E)，将 E 拆分成若干个边集 $E_1, E_2, ..., E_k \left(\bigcup_{i=1}^{k} E_i = E\right)$。对每个边集求这个边集的 MST，得到新边集 $E'_1, E'_2, ..., E'_k$，再在只使用 $E'_1, E'_2, ..., E'_k$ 里的边的情况下求 MST。这样求 MST 与用 E 求 MST 是等价的。

（2025.10.5，AT_cf17_final_j Tree MST）

> Quote
>
> 相信你发现了，上面的算法甚至没有办法通过子任务 2。虽然这个子任务是给大家各显神通的，但是如果你仔细思考，你就会发现度数在整个题目的影响是非常大的。下面我们来介绍，如何处理 dijkstra 中，每个点度数很大，导致松弛边过多。
>
> 如果你做过 k 短路，或者类似的题，它们都有用到这个技巧：我们注意到由一个点出发的边，肯定是边权小的会先出堆，因此，如果我们预先将每个点的出边按照边权排序，我们可以只往堆中塞入第一条边，当第一条边出堆时候，再把第二条边塞入堆中。如此操作，堆中的边数就线性于出堆次数了，我们也就解决了这个艰难的问题。
>
> ——《PKUSC 2024 最短路径 题解》by skip2004

（2025.10.5，QOJ8673 最短路径）

由生日悖论可得，随机选取一列数字，出现重复数字需要的抽样规模的期望是 $O(\sqrt{n})$ 的。

（2025.10.5，QOJ8673 最短路径）