Let it Rot

skip2004 | fsy jiaxun when

2023 年 3 月 10 日

目录

1 图论

1.1 二分图匹配 | 最小边覆盖

```
// 匈牙利, 左到右单向边, O(M|match|)
std::vector<int> edge[N];
inline bool dfs(int x, std::vector<int> & vis, std::vector<int> & match) {
   for(int y : edge[x]) if(!vis[y])
      if(vis[y] = 1, !match[y] || dfs(match[y], vis, match))
         return match[y] = x, 1;
   return 0;
}
inline std::vector<int> match(int nl, int nr) {
   std::vector<int> vis(nr + 1), match(nr + 1), ret(nl + 1);
   for(int i = 1;i <= nl;++i) if(dfs(i, vis, match))
      memset(vis.data(), 0, vis.size() << 2);
   for(int i = 1;i <= nr;++i) ret[match[i]] = i;
   return ret[0] = 0, ret;
}
/* 最小边覆盖, 很可能没用, 真别抄, 抄这个不用抄 match */
inline std::pair<std::vector<int>, std::vector<int>> minedgecover(int nl, int nr) {
   std::vector<int> vis(nr + 1), match(nr + 1), ret(nl + 1);
   for(int i = 1;i <= nl;++i) if(dfs(i, vis, match))
      memset(vis.data(), 0, vis.size() << 2);
   for(int i = 1;i <= nr;++i) ret[match[i]] = i;
   ret[0] = 0;
   for(int i = 1;i <= nl;++i) if(!ret[i]) dfs(i, vis, match);
   std::vector<int> le, ri;
   for(int i = 1;i <= nl;++i) if(ret[i] && !vis[ret[i]]) le.push_back(i);
   for(int i = 1;i <= nr;++i) if(vis[i]) ri.push_back(i);
   return std::make_pair(le, ri);
}
// -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
// 匈牙利, 左到右单向边, bitset, O(n^2/w |match|)
// -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
using set = std::bitset<N>;
set edge[N];
inline bool dfs(int x, set & unvis, std::vector<int> & match) {
   for(set z = edge[x];;) {
      z &= unvis;
      int y = z._Find_first();
      if(y == N) return 0;
      if(unvis.reset(y), !match[y] || dfs(match[y], unvis, match))
         return match[y] = x, 1;
   }
}
```

```
inline std::vector<int> match(int nl, int nr) {
   set unvis; unvis.set();
   std::vector<int> match(nr + 1), ret(nl + 1);
   for(int i = 1;i <= nl;++i)
      if(dfs(i, unvis, match))
         unvis.set();
   for(int i = 1;i <= nr;++i) ret[match[i]] = i;
   return ret[0] = 0, ret;
}
// -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
// HK, 左到右单向边, O(M sqrt(|match|))
// -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
std::vector<int> edge[N];
std::vector<int> L, R, vis, matchl, matchr;
inline bool bfs(int nl, int nr) {
   L.assign(nl + 1, 0), R.assign(nr + 1, 0);
   static std::queue<int> Q;
   for(int i = 1;i <= nl;++i)
      if(!matchl[i]) L[i] = 1, Q.push(i);
   int succ = 0;
   for(;Q.size();) {
      int x = Q.front(); Q.pop();
      for(int i : edge[x]) if(!R[i]) {
         R[i] = L[x] + 1;
         if(int p = matchr[i]) L[p] = R[i] + 1, Q.push(p);
         else succ = 1;
      }
   }
   return succ;
}
inline bool dfs(int x) {
   for(int i : edge[x]) if(R[i] == L[x] + 1 && !vis[i]) {
      vis[i] = 1;
      if(!matchr[i] || dfs(matchr[i]))
         return matchl[x] = i, matchr[i] = x, 1;
   }
   return 0;
}
inline std::vector<int> match(int nl, int nr) {
   matchl.assign(nl + 1, 0);
   matchr.assign(nr + 1, 0);
   vis.resize(nr + 1);
   for(;bfs(nl, nr);) {
      memset(vis.data(), 0, vis.size() << 2);
      for(int i = 1;i <= nl;++i) if(!matchl[i]) dfs(i);
   }
```

```
    return matchl;
}
/* 最小边覆盖，很可能没用，真别抄 */
inline std::pair<std::vector<int>, std::vector<int>> minedgecover(int nl, int nr) {
    auto res = match(nl, nr);
    std::vector<int> ls, rs;
    for(int i = 1;i <= nl;++i) if(!L[i]) ls.push_back(i);
    for(int i = 1;i <= nr;++i) if(R[i]) rs.push_back(i);
    return std::make_pair(ls, rs);
}
```

1.2 网络最大流 | dinic

```
// S 编号最小，T 最大，或者改一下清空
struct Dinic {
    struct T {
        int to, nxt, v;
    } e[N << 3];
    int h[N], head[N], num = 1;
    inline void link(int x, int y, int v) {
        e[++num] = {y, h[x], v}, h[x] = num;
        e[++num] = {x, h[y], 0}, h[y] = num; // !!!
    }
    int dis[N];
    bool bfs(int s, int t) {
        std::queue<int> Q;
        for(int i = s;i <= t;++i) dis[i] = -1, head[i] = h[i]; //如果编号不是[S,T]，只要改这里
        for(Q.push(s), dis[s] = 0;!Q.empty();) {
            int x = Q.front(); Q.pop();
            for(int i = h[x];i;i = e[i].nxt) if(e[i].v && dis[e[i].to] < 0) {
                dis[e[i].to] = dis[x] + 1, Q.push(e[i].to);
            }
        }
        return dis[t] >= 0;
    }
    int dfs(int s, int t, int lim) {
        if(s == t || !lim) return lim;
        int ans = 0, mn;
        for(int & i = head[s];i;i = e[i].nxt) {
            if(dis[e[i].to] == dis[s] + 1 && (mn = dfs(e[i].to, t, std::min(lim, e[i].v)))) {
                e[i].v -= mn, e[i ^ 1].v += mn;
                ans += mn, lim -= mn;
                if(!lim) break;
            }
        }
        return ans;
    }
    inline int flow(int s, int t) {
        int ans = 0;
        for(;bfs(s, t);) ans += dfs(s, t, 1e9);
        return ans;
    }
} G;
```

1.3 最小费用流 | dijkstra

```
// S 编号最小，T 最大，或者改一下清空
namespace mcmf {
   using pr = std::pair<ll, int>;
   const int N = 10005, M = 1e6 + 10;
   struct edge {
      int to, nxt, v, f;
   } e[M << 1];
   int h[N], num = 1;
   inline void link(int x, int y, int v, int f) {
      e[++num] = {y, h[x], v, f}, h[x] = num;
      e[++num] = {x, h[y], 0, -f}, h[y] = num;
   }
   ll d[N], dis[N];
   int vis[N], fr[N];
   inline void spfa(int s, int t) {
      std::queue<int> Q;
      std::fill(d + s, d + t + 1, 1e18);
      for(d[s] = 0, Q.push(s);!Q.empty();) {
         int x = Q.front(); Q.pop(); vis[x] = 0;
         for(int i = h[x];i;i = e[i].nxt)
            if(e[i].v && d[e[i].to] > d[x] + e[i].f) {
               d[e[i].to] = d[x] + e[i].f;
               if(!vis[e[i].to]) vis[e[i].to] = 1, Q.push(e[i].to);
            }
      }
   }
   inline bool dijkstra(int s, int t) {
      std::priority_queue<pr, std::vector<pr>, std::greater<pr>> Q;
      std::fill(dis + s, dis + t + 1, 1e18);
      std::fill(vis + s, vis + t + 1, 0);
      for(Q.emplace(dis[s] = 0, s);!Q.empty();) {
         int x = Q.top().second; Q.pop();
         if(vis[x]) continue;
         vis[x] = 1;
         for(int i = h[x];i;i = e[i].nxt) {
            const ll v = e[i].f + d[x] - d[e[i].to];
            if(e[i].v && dis[e[i].to] > dis[x] + v) {
               fr[e[i].to] = i;
               Q.emplace(dis[e[i].to] = dis[x] + v, e[i].to);
            }
         }
      }
      return dis[t] < 1e17;
   }
```

```
    std::pair<ll, ll> EK(int s, int t) {
        spfa(s, t);
        ll f = 0, c = 0;
        for(;dijkstra(s, t);) {
            ll fl = 1e18;
            for(int i = fr[t];i;i = fr[e[i ^ 1].to]) fl = std::min<ll>(e[i].v, fl);
            for(int i = s;i <= t;++i) d[i] += dis[i];
            f += fl, c += fl * d[t];
            for(int i = fr[t];i;i = fr[e[i ^ 1].to])
                e[i].v -= fl, e[i ^ 1].v += fl;
        }
        return std::make_pair(f, c);
    }
}
```

1.4 最小费用流 | spfa

```cpp
using ll = long long;
// S 编号最小, T 最大, 或者改一下清空
namespace mcmf {
   using pr = std::pair<ll, int>;
   const int N = 10005, M = 1e6 + 10;
   struct edge {
      int to, nxt, v, f;
   } e[M << 1];
   int h[N], num = 1;
   inline void link(int x, int y, int v, int f) {
      e[++num] = {y, h[x], v, f}, h[x] = num;
      e[++num] = {x, h[y], 0, -f}, h[y] = num;
   }
   ll d[N];
   int vis[N], fr[N];
   inline bool spfa(int s, int t) {
      std::queue<int> Q;
      std::fill(d + s, d + t + 1, 1e18);
      for(d[s] = 0, Q.push(s);!Q.empty();) {
         int x = Q.front(); Q.pop(); vis[x] = 0;
         for(int i = h[x];i;i = e[i].nxt)
            if(e[i].v && d[e[i].to] > d[x] + e[i].f) {
               d[e[i].to] = d[x] + e[i].f, fr[e[i].to] = i;
               if(!vis[e[i].to]) vis[e[i].to] = 1, Q.push(e[i].to);
            }
      }
      return d[t] < 1e17;
   }
   inline std::pair<ll, ll> EK(int s, int t) {
      ll f = 0, c = 0;
      for(;spfa(s, t);) {
         ll fl = 1e18;
         for(int i = fr[t];i;i = fr[e[i ^ 1].to]) fl = std::min<ll>(e[i].v, fl);
         f += fl, c += fl * d[t];
         for(int i = fr[t];i;i = fr[e[i ^ 1].to])
            e[i].v -= fl, e[i ^ 1].v += fl;
      }
      return std::make_pair(f, c);
   }
}
```

1.5 最小树形图

抄罗大的，返回值是边的集合，如果没有最小树形图会返回 -1，可以修改建图。

```
namespace DMST {
   struct edge {
      int u, v, id; ll w;
      inline bool operator < (const edge & y) const {
         return w < y.w;
      }
   } ent[N], val[M];
   int ls[M], rs[M], size[M], cc; ll tag[M];
   int fs[N], fw[N], rt[N];

   inline void put(int x, ll v) {
      if(x) val[x].w += v, tag[x] += v;
   }
   inline void pushdown(int x) {
      put(ls[x], tag[x]);
      put(rs[x], tag[x]);
      tag[x] = 0;
   }
   inline int merge(int x, int y) {
      if(!x || !y) return x | y;
      if(val[y] < val[x]) std::swap(x, y);
      pushdown(x), rs[x] = merge(rs[x], y);
      if(size[rs[x]] > size[ls[x]]) {
         std::swap(ls[x], rs[x]);
      }
      size[x] += size[y];
      return x;
   }
   inline void ins(int & x, const edge & z) {
      val[++cc] = z, size[cc] = 1;
      x = merge(x, cc);
   }
   inline void pop(int & x) { x = merge(ls[x], rs[x]); }
   edge top(int x) { return val[x]; }
   inline int find(int x, int * anc) {
      return anc[x] == x ? x : anc[x] = find(anc[x], anc);
   }
   inline void link(int u, int v, int w, int id) {
      ins(rt[v], {u, v, id, w});
   }
   int pa[N * 2], tval[N * 2], up[N * 2], end_edge[M], cmt, baned[M];
   std::vector<int> solve(int r) {
      std::queue<int> roots;
```

```
        for(int i = 1;i <= n;++i) {
            fs[i] = fw[i] = i, tval[i] = ++ cmt;
            if(i != r) roots.push(i);
        }
        std::vector<edge> H;
        std::vector<int> ret;
        for(;!roots.empty();) {
            int k = roots.front(); roots.pop();
            if(!rt[k]) return ret;
            edge e = top(rt[k]); pop(rt[k]);
            int i = e.u, j = e.v;
            if(find(i, fs) == k) roots.push(k);
            else {
                H.push_back(e); end_edge[e.id] = tval[k];
                if(find(i, fw) != find(j, fw)) {
                    fw[find(j, fw)] = i;
                    ent[k] = e;
                } else {
                    pa[tval[k]] = ++ cmt, up[tval[k]] = e.id;
                    put(rt[k], -e.w);
                    for(;(e = ent[find(e.u, fs)]).u;) {
                        int p = find(e.v, fs);
                        pa[tval[p]] = cmt;
                        up[tval[p]] = e.id;
                        put(rt[p], -e.w);
                        rt[k] = merge(rt[k], rt[p]);
                        fs[p] = k;
                    }
                    tval[k] = cmt;
                    roots.push(k);
                }
            }
        }
        reverse(H.begin(), H.end());
        for(edge i : H) if(!baned[i.id]) {
            ret.push_back(i.id);
            for(int j = i.v;j != end_edge[i.id];j = pa[j]) ++ baned[up[j]];
        }
        sort(ret.begin(), ret.end());
        return ret;
    }
}
```

1.6 缩点 | kasaraju

时间复杂度 $O(\frac{n^2}{w})$, 可以对于边修改不多的图快速计算。

```cpp
using set = std::bitset<N>;
// re 是反向边, 需要连好
set e[N], re[N], vis;
std::vector<int> sta;
inline void dfs0(int x, set * e) {
   vis.reset(x);
   for(;;) {
      int go = (e[x] & vis)._Find_first();
      if(go == N) break;
      dfs0(go, e);
   }
   sta.push_back(x);
}
inline std::vector<std::vector<int>> solve() {
   vis.set();
   for(int i = 1;i <= n;++i) if(vis.test(i)) dfs0(i, e);
   vis.set();
   auto s = sta;
   std::vector<std::vector<int>> ret;
   for(int i = n - 1;i >= 0;--i) if(vis.test(s[i])) {
      sta.clear(), dfs0(s[i], re), ret.push_back(sta);
   }
   return ret;
}
```

2 Math

2.1 万能欧几里得

```
// 万欧
// 前提 : r < q, r >= q 先提几个 U 出来再用
// 使用: Y * q <= X * p + r, 斜率 p/q, U表示向上，R表示到达一个顶点，先一些 U 再一个 R
template<class T>
inline T power(T a, ll k) {
    // 有效率需求可以改为半群乘法
    if(!k) return T();
    T res = a;
    for(--k;k;) {
        if(k & 1) res = res + a;
        if(k >>= 1) a = a + a;
    }
    return res;
}
template<class T>
T solve(ll p, ll q, ll r, ll l, T U, T R) {
    if (p >= q)
        return solve(p % q, q, r, l, U, power(U, p / q) + R);
    ll m = ((__int128)p * l + r) / q;
    if (!m) return power(R, l);
    ll cnt = l - ((__int128)q * m - r - 1) / p;
    return power(R, (q - r - 1) / p) + U + solve(q, p, (q - r - 1) % p, m - 1, R, U) + power(R,
        cnt);
}
```

2.2 直线下点数 | 欧几里得

$n < 2^{32}, 1 \leq m < 2^{32}$

$$result = \sum_{i=0}^{n-1} \lfloor \frac{ai+b}{m} \rfloor \pmod{2^{64}}$$

```
u64 floor_sum(u64 n, u64 m, u64 a, u64 b) {
	u64 ans = 0;
	for(;;) {
		if(a >= m) {
			ans += n * (n - 1) / 2 * (a / m);
			a %= m;
		}
		if(b >= m) {
			ans += n * (b / m);
			b %= m;
		}
		u64 ymax = a * n + b;
		if(ymax < m) break;
		n = ymax / m;
		b = ymax % m;
		std::swap(m, a);
	}
	return ans;
}
```

2.3 扩展欧几里得

```
// exgcd
// result : -b < x < b AND -a < y <= a when a,b != 0
inline void exgcd(ll a, ll b, ll & x, ll & y) {
   if(!b) return x = 1, y = 0, void();
   exgcd(b, a % b, y, x), y -= a / b * x;
}
```

2.4 扩展中国剩余定理

```
ll exCRT(ll a1, ll p1, ll a2, ll p2) {
       ll a, b, gcd = std::gcd(p1, p2);
       if((a1 - a2) % gcd)
              return -1;
       exgcd(p1, p2, a, b);
       ll k = i128((a2 - a1) % p2 + p2) * (a + p2) % p2;
       return p1 / gcd * k + a1;
}
```

2.5 Miller-Rabin

```
using f64 = long double;
ll p;
f64 invp;
inline void setmod(ll x) {
	p = x, invp = (f64) 1 / x;
}
inline ll mul(ll a, ll b) {
	ll z = a * invp * b + 0.5;
	ll res = a * b - z * p;
	return res + (res >> 63 & p);
}
inline ll pow(ll a, ll x, ll res = 1) {
	for(;x;x >>= 1, a = mul(a, a))
		if(x & 1) res = mul(res, a);
	return res;
}
inline bool checkprime(ll p) {
	if(p == 1) return 0;
	setmod(p);
	ll d = __builtin_ctzll(p - 1), s = (p - 1) >> d;
	for(ll a : {2, 3, 5, 7, 11, 13, 82, 373}) {
		if(a % p == 0)
			continue;
		ll x = pow(a, s), y;
		for(int i = 0;i < d;++i, x = y) {
			y = mul(x, x);
			if(y == 1 && x != 1 && x != p - 1)
				return 0;
		}
		if(x != 1) return 0;
	}
	return 1;
}
```

2.6 Pollard-rho

```
inline ll rho(ll n) {
	if(!(n & 1))
		return 2;
	static std::mt19937_64 gen((size_t)"hehezhou");
	ll c = gen() % (n - 1) + 1, y = gen() % (n - 1) + 1;
	auto f = [&](ll o) {
		o = mul(o, o) + c;
		return o >= n ? o - n : o;
	};
	setmod(p);
	for(int l = 1;;l <<= 1) {
		ll x = y, g = 1;
		for(int i = 0;i < l;++i) y = f(y);
		const int d = 512;
		for(int i = 0;i < l;i += d) {
			ll sy = y;
			for(int j = 0;j < std::min(d, l - i);++j) {
				y = f(y), g = mul(g, (y - x + n));
			}
			g = std::__gcd(n, g);
			if(g == 1)
				continue;
			if(g == n)
				for(g = 1, y = sy;g == 1;)
					y = f(y), g = std::__gcd(n, y - x + n);
			return g;
		}
	}
}
inline std::vector<ll> factor(ll x) {
	std::queue<ll> q; q.push(x);
	std::vector<ll> res;
	for(;q.size();) {
		ll x = q.front(); q.pop();
		if(x == 1) continue;
		if(checkprime(x)) {
			res.push_back(x);
			continue;
		}
		ll y = rho(x);
		q.push(y), q.push(x / y);
	}
	sort(res.begin(), res.end());
	return res;
}
```

2.7 Fast Fourier Transform

```cpp
using db = double;
using C = std::complex<db>;
// C::real, C::imag, std::conj, std::arg
const db pi = std::acos(-1);
int rev[N], lim, invlim;;
C wn[N];

void init(int len) {
   lim = 2 << std::__lg(len - 1);
   invlim = mod - (mod - 1) / lim;
   for(static int i = 1;i < lim;i += i) {
      for(int j = 0;j < i;++j) {
         wn[i + j] = std::polar(1., db(j) / i * pi);
      }
   }
   for(int i = 1;i < lim;++i) {
      rev[i] = rev[i >> 1] >> 1 | (i % 2u * lim / 2);
   }
}
void DFT(C * a) {
   for(int i = 0;i < lim;++i) {
      if(rev[i] < i) std::swap(a[rev[i]], a[i]);
   }
   for(int i = 1;i < lim;i += i) {
      for(int j = 0;j < lim;j += i + i) {
         for(int k = 0;k < i;++k) {
            C x = a[i + j + k] * wn[i + k];
            a[i + j + k] = a[k + j] - x;
            a[k + j] += x;
         }
      }
   }
}
void IDFT(C * a) {
   DFT(a), std::reverse(a + 1, a + lim);
   for(int i = 0;i < lim;++i)
      a[i] /= lim;
}
```

2.8 Number Theoretic Transform

```
int rev[N], wn[N], lim, invlim;
int norm(int x) {
    return x >= mod ? x - mod : x;
}
int pow(int a, int b, int ans = 1) {
    for(;b;b >>= 1, a = (u64) a * a % mod) if(b & 1)
        ans = (u64) ans * a % mod;
    return ans;
}
void init(int len) {
    lim = 2 << std::__lg(len - 1);
    invlim = mod - (mod - 1) / lim;
    for(static int i = 1;i < lim;i += i) {
        wn[i] = 1;
        const int w = pow(3, mod / i / 2);
        for(int j = 1;j < i;++j) {
            wn[i + j] = (u64) wn[i + j - 1] * w % mod;
        }
    }
    for(int i = 1;i < lim;++i) {
        rev[i] = rev[i >> 1] >> 1 | (i % 2u * lim / 2);
    }
}
void DFT(int * a) {
    static u64 t[N];
    for(int i = 0;i < lim;++i) {
        t[i] = a[rev[i]];
    }
    for(int i = 1;i < lim;i += i) {
        for(int j = 0;j < lim;j += i + i) {
            for(int k = 0;k < i;++k) {
                const u64 x = t[i + j + k] * wn[i + k] % mod;
                t[i + j + k] = t[k + j] + mod - x, t[k + j] += x;
            }
        }
    }
    for(int i = 0;i < lim;++i) a[i] = t[i] % mod;
}
void IDFT(int * a) {
    DFT(a), std::reverse(a + 1, a + lim);
    for(int i = 0;i < lim;++i)
        a[i] = (u64) a[i] * invlim % mod;
}
```

2.9 Generating function

```
void cpy(int * a, int * b, int n) {
    if(a != b) memcpy(a, b, n << 2);
    memset(a + n, 0, (lim - n) << 2);
}
void inv(int * a, int * b, int n) { // mod x^n
    if(n == 1) return void(*b = pow(*a, mod - 2));
    static int c[N], d[N];
    int m = (n + 1) / 2;
    inv(a, b, m);
    init(n + m), cpy(c, b, m), cpy(d, a, n);
    DFT(c), DFT(d);
    for(int i = 0;i < lim;++i) c[i] = (u64) c[i] * c[i] % mod * d[i] % mod;
    IDFT(c);
    for(int i = m;i < n;++i) b[i] = norm(mod - c[i]);
}
void log(int * a, int * b, int n) {
    static int c[N], d[N];
    inv(a, c, n), init(n + n);
    for(int i = 1;i < n;++i) d[i - 1] = (u64) a[i] * i % mod;
    cpy(d, d, n - 1), cpy(c, c, n);
    DFT(c), DFT(d);
    for(int i = 0;i < lim;++i) c[i] = (u64) c[i] * d[i] % mod;
    IDFT(c), *b = 0;
    for(int i = 1;i < n;++i) b[i] = pow(i, mod - 2, c[i - 1]);
}
```

3 字符串

3.1 后缀自动机 | SAM

需要两倍点数量。

```
int c[N][26], mx[N], fail[N], tot = 1;
int append(int id, int w) {
    int p = id, now = ++ tot;
    //right[now] = id;
    for(mx[now] = mx[p] + 1;p && !c[p][w];p = fail[p])
        c[p][w] = now;
    if(!p) fail[now] = 1;
    else {
        int q = c[p][w];
        if(mx[q] == mx[p] + 1) fail[now] = q;
        else {
            int x = ++ tot; mx[x] = mx[p] + 1;
            memcpy(c[x], c[q], sizeof(c[0])), fail[x] = fail[q]; //right[x] = right[q];
            for(fail[q] = fail[now] = x;p && c[p][w] == q;p = fail[p])
                c[p][w] = x;
        }
    }
    return now;
}
void 后缀树() { // 倒着建
    for(int i = 2;i <= tot;++i)
        son[fa[i]][s[right[i] + mx[fa[i]]] - 'a'] = i;
}
```

3.2 回文自动机 | PAM

```
int c[N][26], fail[N], len[N], tot;
void init() {
    fail[0] = 1, len[++tot] = -1;
    // root is 1
}
int get_fail(int o, char * x) {
    for(;*x != x[-len[o] - 1];)
        o = fail[o];
    return o;
}
int append(int o, char * x) {
    o = get_fail(o, x);
    int & p = c[o][*x - 'a'];
    if(!p) {
        fail[++tot] = c[get_fail(fail[o], x)][*x - 'a'];
        len[p = tot] = len[o] + 2;
    }
    return p;
}
```

3.3 回文自动机 border 处理

```
int c[N][26], fail[N], len[N], tot;
void init() {
   fail[0] = 1, len[++tot] = -1;
   // root is 1
}
int get_fail(int o, char * x) {
   for(;*x != x[-len[o] - 1];)
      o = fail[o];
   return o;
}
int append(int o, char * x) {
   o = get_fail(o, x);
   int & p = c[o][*x - 'a'];
   if(!p) {
      fail[++tot] = c[get_fail(fail[o], x)][*x - 'a'];
      len[p = tot] = len[o] + 2;
   }
   return p;
}
BorderPam
info atom[N];
int c[N][26], fail[N], len[N], tot;
int diff[N], bigfail[N];
/*----------------------------------------*/
// 严格 logn 插入，但是很可能不要
void init() {
   bigfail[0] = fail[0] = 1, len[++tot] = -1;
   // root is 1
}
int get_fail(int o, char * x) {
   for(;*x != x[-len[o] - 1];) {
      o = (*x != x[-len[fail[o]] - 1] ? bigfail : fail)[o];
   }
   return o;
}
/*----------------------------*/
void init() {
   fail[0] = 1, len[++tot] = -1;
   // root is 1
}
int get_fail(int o, char * x) {
   for(;*x != x[-len[o] - 1];)
      o = fail[o];
   return o;
}
int append(int o, char * x) {
    o = get_fail(o, x);
    int & p = c[o][*x - 'a'];
    if(!p) {
        fail[++tot] = c[get_fail(fail[o], x)][*x - 'a'];
        len[p = tot] = len[o] + 2;
        diff[p] = len[p] - len[fail[p]];
        bigfail[p] = diff[p] == diff[fail[p]] ? bigfail[fail[p]] : fail[p];
    }
    return p;
}
info node[N];
info query(int x, int i) {
    info z = {};
    for(;x;x = bigfail[x]) {
        const int go = bigfail[x], fa = fail[x];
        info & t = node[x] = atom[i - len[go] - diff[x]];
        if(fa != go) t = node[fa] + t;
        z = t + z;
    }
    return z;
}
```

3.4 后缀数组 | SA

```
char s[N];
int rank[N], sa[N], h[N], n, L;
bool cmp(int a, int b) {
    if(rank[a] != rank[b]) return rank[a] < rank[b];
    return b + L <= n && (a + L > n || rank[a + L] < rank[b + L]);
}
void SA() { // s[n + 1] need '0'
    static int a[N], t[N];
    for(int i = 1;i <= n;++i) rank[i] = s[i];
    std::iota(a, a + n + 1, 0);
    for(L = 1;;L <<= 1) {
        std::sort(a + 1, a + n + 1, cmp);
        for(int i = 1, r = 0;i <= n;++i) t[a[i]] = r += !i || cmp(a[i - 1], a[i]);
        memcpy(rank + 1, t + 1, n << 2);
        if(t[a[n]] == n) break;
    }
    for(int i = 1;i <= n;++i) sa[rank[i]] = i;
    for(int i = 1, k = 0;i <= n;++i) if(rank[i] < n) {
        int j = sa[rank[i] + 1];
        for(k -= !!k;s[i + k] == s[j + k];++k);
        h[rank[i]] = k;
    }
}
```

3.5 AC 自动机

```
const int sig = 26;
int son[N][sig], fail[N], cnt;
int ins(const char * c) {
    int x = 0;
    for(;*c;++c) {
        int & p = son[x][*c - 'a'];
        if(!p) p = ++ cnt;
        x = p;
    }
    return x;
}
void build_ac() {
    std::queue<int> Q;
    for(int i = 0;i < sig;++i) if(son[0][i]) Q.push(son[0][i]);
    for(;Q.size();) {
        int x = Q.front(); Q.pop();
        for(int i = 0;i < sig;++i)
            if(son[x][i]) fail[son[x][i]] = son[fail[x]][i], Q.push(son[x][i]);
            else son[x][i] = son[fail[x]][i];
    }
}
```

4 数据结构

4.1 区间加区间求和树状数组

```
// 后缀加，前缀求和
struct BIT {
   ll a[N], b[N];
   inline void add(ll p, int v) {
      for(int i = p;i < N;i += i & -i)
         a[i] += v, b[i] += p * v;
   }
   inline ll qry(ll p) {
      ll res = 0;
      for(int i = p;i;i &= i - 1) res += (p + 1) * a[i] - b[i];
      return res;
   }
   inline void add(int l, int r, int v) {
      add(l, v), add(r + 1, -v);
   }
   inline ll qry(int l, int r) {
      return qry(r) - qry(l - 1);
   }
} bit;
```

4.2 zkw 线段树

```
struct seg {
   ll o[1 << 20]; int L;
   void upt(int x) {
      o[x] = o[x << 1] + o[x << 1 | 1];
   }
   void init(int n, int * w) {
      L = 2 << std::__lg(n + 1);
      for(int i = 1;i <= n;++i) o[i + L] = w[i];
      for(int i = L;i >= 1;--i) upt(i);
   }
   void upt(int p, int v) {
      for(o[p += L] += v;p >>= 1;upt(p));
   }
   ll qry(int l, int r) {
      l += L - 1, r += L + 1;
      ll ans = 0;
      for(;l ^ r ^ 1;l >>= 1, r >>= 1) {
         if((l & 1) == 0) ans += o[l ^ 1];
         if((r & 1) == 1) ans += o[r ^ 1];
      }
      return ans;
   }
   // if there is no I
   ll qry2(int l, int r) {
      if(l == r) return o[l + L];
      ll le = o[l + L], ri = o[r + L];
      l += L, r += L;
      for(;l ^ r ^ 1;l >>= 1, r >>= 1) {
         if((l & 1) == 0) le = le + o[l ^ 1];
         if((r & 1) == 1) ri = o[r ^ 1] + ri;
      }
      return le + ri;
   }
} sgt;
```

4.3 Link Cut Tree

```
int son[N][2], fa[N], rev[N];

int get(int x, int p = 1) {
    return son[fa[x]][p] == x;
}
void update(int x) {
}
int is_root(int x) {
    return !(get(x) || get(x, 0));
}
void rotate(int x) {
    int y = fa[x], z = fa[y], b = get(x);
    if(!is_root(y)) son[z][get(y)] = x;
    son[y][b] = son[x][!b], son[x][!b] = y;
    fa[son[y][b]] = y, fa[y] = x, fa[x] = z;
    update(y);
}
void put(int x) {
    if(x) rev[x] ^= 1, std::swap(son[x][0], son[x][1]);
}
void down(int x) {
    if(rev[x]) {
        put(son[x][0]);
        put(son[x][1]);
        rev[x] = 0;
    }
}
void pushdown(int x) {
    if(!is_root(x)) pushdown(fa[x]);
    down(x);
}
void splay(int x) {
    for(pushdown(x);!is_root(x);rotate(x)) if(!is_root(fa[x]))
        rotate(get(x) ^ get(fa[x]) ? x : fa[x]);
    update(x);
}
void access(int x) {
    for(int t = 0;x;son[x][1] = t, t = x, x= fa[x])
        splay(x);
}
void makeroot(int x) {
    access(x), splay(x), put(x);
}
```

5 计算几何

5.1 向量

```
using db = long double;
const db eps = 1e-10;

inline int add(int a, int b, int p) {
   return a += b, a >= p ? a - p : a;
}

inline db sgn(db x) {
   return x < -eps ? -1 : x > eps;
}
struct vec2 {
   db x, y;
   inline vec2() { }
   inline vec2(db a, db b) : x(a), y(b) { }
   inline db norm() const {
      return x * x + y * y;
   }
   inline db abs() const {
      return std::sqrt(x * x + y * y);
   }
};
inline vec2 operator + (const vec2 & x, const vec2 & y) {
   return vec2(x.x + y.x, x.y + y.y);
}
inline vec2 operator - (const vec2 & x, const vec2 & y) {
   return vec2(x.x - y.x, x.y - y.y);
}
inline vec2 operator / (const vec2 & x, db y) {
   return vec2(x.x / y, x.y / y);
}
inline db operator * (const vec2 & x, const vec2 & y) {
   return x.x * y.y - x.y * y.x;
}
inline vec2 operator * (const db & x, const vec2 & y) {
   return vec2(x * y.x, x * y.y);
}
inline vec2 operator * (const vec2 & y, const db & x) {
   return vec2(x * y.x, x * y.y);
}
inline db operator % (const vec2 & x, const vec2 & y) {
   return x.x * y.x + x.y * y.y;
}
```

```cpp
inline db dist(const vec2 & x, const vec2 & y) {
    return (x - y).abs();
}
// 逆时针极角排序
inline int half(const vec2 & x) {
    return x.y < 0 || (x.y == 0 && x.x <= 0);
}
inline bool cmp(const vec2 & A, const vec2 & B) {
    if(half(A) != half(B)) return half(B);
    return A * B > 0;
}
```

5.2 直线半平面

```
struct line {
    db a, b, c;
    // a * x + b * y + c (= or >) 0
    inline line(db A, db B, db C) : a(A), b(B), c(C) {}
    inline line(const vec2 & x, const vec2 & y) : a(x.y - y.y), b(y.x - x.x), c(x * y) { }
    // 左侧 > 0
    inline db operator ()(const vec2 & x) const { return a * x.x + b * x.y + c; }
    inline line perp() const { return line(b, -a, 0); }
    inline line para(const vec2 & o) { return line(a, b, c - (*this)(o)); }
    inline vec2 normVec() const { return vec2(a, b); }
    inline db norm() const { return normVec().norm(); }
};
inline vec2 operator & (const line & x, const line & y) {
    return vec2(vec2(x.c, x.b) * vec2(y.c, y.b), vec2(x.a, x.c) * vec2(y.a, y.c)) / -(vec2(x.a,
        x.b) * vec2(y.a, y.b));
}
inline vec2 proj(const vec2 & x, const line & l) {
    return x - l.normVec() * (l(x) / l.norm());
}
inline vec2 refl(const vec2 & x, const line & l) {
    return x - l.normVec() * (l(x) / l.norm()) * 2;
}
inline bool is_para(const line & x, const line & y) {
    return std::fabs(x.normVec() * y.normVec()) < eps;
}
inline bool is_perp(const line & x, const line & y) {
    return std::fabs(x.normVec() % y.normVec()) < eps;
}
inline bool online(const vec2 & x, const line & l) {
    return std::fabs(l(x)) < eps;
}
inline int ccw(vec2 a, vec2 b, vec2 c) {
    int sign = sgn((b - a) * (c - a));
    if(sign == 0) {
        if(sgn((b - a) % (c - a)) == -1) return 2;
        if((c - a).norm() > (b - a).norm() + eps) return -2;
    }
    return sign;
}

// 三条直线关系
db det(line a, line b, line c) {
    vec2 A = vec2(a), B = vec2(b), C = vec2(c);
    return c.c * (A * B) + a.c * (B * C) + b.c * (C * A);
}
db check(line a, line b, line c) { // sgn same as c(a & b), 0 if error
   return sgn(det(a, b, c)) * sgn(vec2(a) * vec2(b));
}
```

5.3 半平面交

```
std::vector<vec2> HalfPlaneI(std::vector<line> vs) {
    auto cmp = [](line a, line b) {
        if(paraS(a, b)) {
            return dist(a) < dist(b);
        }
        return ::cmp(vec2(a), vec2(b));
    };
    sort(vs.begin(), vs.end(), cmp);
    int ah = 0, at = 0, n = size(vs);
    std::vector<line> deq(n + 1);
    std::vector<vec2> ans(n);
    deq[0] = vs[0];
    for(int i = 1;i <= n;++i) {
        line o = i < n ? vs[i] : deq[ah];
        if(paraS(vs[i - 1], o)) {
            continue;
        }
        for(;ah < at && check(deq[at - 1], deq[at], o) < 0;) -- at;
        if(i != n)
        for(;ah < at && check(deq[ah], deq[ah + 1], o) < 0;) ++ ah;
        if(!is_para(o, deq[at])) {
            ans[at] = o & deq[at];
            deq[++at] = o;
        }
    }
    if(at - ah <= 2) return {};
    return {ans.begin() + ah, ans.begin() + at};
}
```

5.4 线段

```
struct seg {
    vec2 x, y;
    inline seg() {}
    inline seg(const vec2 & A, const vec2 & B) : x(A), y(B) {}
    inline bool onseg(const vec2 & o) const {
        return (o - x) % (o - y) < eps && std::fabs((o - x) * (o - y)) < eps;
    }
    inline line to_l() const {
        return line(x, y);
    }
};
inline bool is_isc(const seg & x, const seg & y) {
    return
        ccw(x.x, x.y, y.x) * ccw(x.x, x.y, y.y) <= eps &&
        ccw(y.x, y.y, x.x) * ccw(y.x, y.y, x.y) <= eps;
}
inline db dist(const seg & o, const vec2 & x) {
    vec2 z = proj(x, o.to_l());
    if(o.onseg(z)) {
        return dist(x, z);
    } else {
        return std::min(dist(o.x, x), dist(o.y, x));
    }
}
inline db dist(const seg & x, const seg & y) {
    if(is_isc(x, y)) return 0;
    return std::min({ dist(y, x.x), dist(y, x.y), dist(x, y.x), dist(x, y.y), });
}
```

5.5 多边形

```
using polygon = std::vector<vec2>;
// counter-clockwise
inline db area(const polygon & x) {
   db res = 0;
   for(int i = 2;i < (int) x.size();++i) {
      res += (x[i - 1] - x[0]) * (x[i] - x[0]);
   }
   return res / 2;
}
inline bool is_convex(const polygon & x, bool strict = 1) {
   // warning, maybe wrong
   const db z = strict ? eps : -eps;
   for(int i = 2;i < (int) x.size() + 2;++i) {
      if((x[(i - 1) % x.size()] - x[i - 2]) * (x[i % x.size()] - x[i - 2]) < z) return 0;
   }
   return 1;
}
inline int inpoly(const vec2 & x, const polygon & o) {
   bool in = false;
   for(int i = 0;i < (int) o.size();++i) {
      if(seg(o[i], o[add(i, 1, size(o))]).onseg(x)) {
         return 1;
      }
      auto a = o[i] - x, b = o[add(i, 1, size(o))];
   }
}
```

6 Dirty Hacks

6.1 Pragma

```
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2")
```

6.2 Barrett

```
struct DIV {
        u64 x;
        void init(u64 v) { x = -1ull / v + 1; }
};
// works while x*(y-1)<2^64
u64 operator / (const u64 & x, const DIV & y) {
        return (u128) x * y.x >> 64;
}
```

6.3 LCS

```
int lim;
struct bitset {
    static const int B = 63;
    u64 a[N / B + 1];
    inline void set(int p) { a[p / B] |= 1ull << (p % B); }
    inline bool test(int p) { return a[p / B] >> (p % B) & 1; }
    inline void run(const bitset & o) {
        u64 c = 1;
        for(int i = 0;i < lim;++i) {
            u64 x = a[i], y = x | o.a[i];
            x += x + c + (~y & (1ull << 63) - 1);
            a[i] = x & y, c = x >> 63;
        }
    }
} dp;
```