---

1. 后缀排序：DC3

DC3 后缀排序算法，时空复杂度 $\Theta(n)$。字符串本体 s 数组、sa 数组和 rk 数组都要求 3 倍空间。下标从 0 开始，字符串长度为 n，字符集 Σ 为 $[0, m]$。partial_sum 需要标准头文件 numeric。

```
#define CH(i, n) i < n ? s[i] : 0
static int ch[NMAX + 10][3], seq[NMAX + 10];
static int arr[NMAX + 10], tmp[NMAX + 10], cnt[NMAX + 10];
inline bool cmp(int i, int j) {
    return ch[i][0] == ch[j][0] && ch[i][1] == ch[j][1] && ch[i][2] == ch[j][2];
}
inline bool sufcmp(int *s, int *rk, int n, int i, int j) {
    if (s[i] != s[j]) return s[i] < s[j];
    if ((i + 1) % 3 && (j + 1) % 3) return rk[i + 1] < rk[j + 1];
    if (s[i + 1] != s[j + 1]) return s[i + 1] < s[j + 1];
    return rk[i + 2] < rk[j + 2];
}
void radix_sort(int n, int m, int K, bool init = true) {
    if (init) for (int i = 0; i < n; i++) arr[i] = i;
    int *a = arr, *b = tmp;
    for (int k = 0; k < K; k++) {
        memset(cnt, 0, sizeof(int) * (m + 1));
        for (int i = 0; i < n; i++) cnt[ch[a[i]][k]]++;
        partial_sum(cnt, cnt + m + 1, cnt);
        for (int i = n - 1; i >= 0; i--) b[--cnt[ch[a[i]][k]]] = a[i];
        swap(a, b);
    }
    if (a != arr) memcpy(arr, tmp, sizeof(int) * n);
}
void suffix_sort(int *s, int n, int m, int *sa, int *rk) {
    s[n] = 0; n++;
    int p = 0, q = 0;
    for (int i = 1; i < n; i += 3, p++) for (int j = 0; j < 3; j++)
        ch[p][2 - j] = CH(i + j, n);
    for (int i = 2; i < n; i += 3, p++) for (int j = 0; j < 3; j++)
        ch[p][2 - j] = CH(i + j, n);
    radix_sort(p, m, 3);
    for (int i = 0; i < p; i++) {
        if (!q || (q && !cmp(arr[i - 1], arr[i]))) q++;
        s[n + arr[i]] = q;
    }
    if (q < p) suffix_sort(s + n, p, q, sa + n, rk + n);
    else {
        for (int i = 0; i < p; i++) sa[n + s[n + i] - 1] = i;
        for (int i = 0; i < p; i++) rk[n + sa[n + i]] = i + 1;
    }
    m = max(m, p);
    p = q = 0;
    for (int i = 1; i < n; i += 3, p++) rk[i] = rk[n + p];
    for (int i = 2; i < n; i += 3, p++) rk[i] = rk[n + p];
    for (int i = 0; i < n; i++) if (i % 3) seq[rk[i] - 1] = i;
    for (int i = 0; i < n; i += 3, q++) {
        ch[i][0] = i + 1 < n ? rk[i + 1] : 0;
        ch[i][1] = s[i];
        arr[q] = i;
    }
    radix_sort(q, m, 2, false);
    for (int i = seq[0] == n - 1, j = arr[0] == n - 1, k = 0; i < p || j < q; k++) {
```

```
        if (i == p) sa[k] = arr[j++];
        else if (j == q) sa[k] = seq[i++];
        else if (sufcmp(s, rk, n, seq[i], arr[j])) sa[k] = seq[i++];
        else sa[k] = arr[j++];
    }
    for (int i = 0; i < n - 1; i++) rk[sa[i]] = i + 1;
}
```

2. AC 自动机

时间复杂度 $O(n+m+z+n|\Sigma|)$，n 是模板串总长度，m 是目标串长度，z 是总匹配次数，Σ 是字符集。如果想移掉 $n|\Sigma|$ 这一项，需要使用哈希表。传入的字符串下标从 0 开始。

```
struct Node {
    Node() : mark(false), suf(NULL), nxt(NULL) {
        memset(ch, 0, sizeof(ch));
    }
    bool mark;
    Node *suf, *nxt, *ch[SIGMA];
};
void insert(Node *x, char *s) {
    for (int i = 0; s[i]; i++) {
        int c = s[i] - 'a';
        if (!x->ch[c]) x->ch[c] = new Node;
        x = x->ch[c];
    }
    x->mark = true;
}
void build_automaton(Node *r) {
    queue<Node *> q;
    for (int c = 0; c < SIGMA; c++) {
        if (!r->ch[c]) continue;
        r->ch[c]->suf = r;
        q.push(r->ch[c]);
    }
    while (!q.empty()) {
        Node *x = q.front();
        q.pop();
        for (int c = 0; c < SIGMA; c++) {
            Node *v = x->ch[c]; if (!v) continue;
            Node *y = x->suf;
            while (y != r && !y->ch[c]) y = y->suf;
            if (y->ch[c]) y = y->ch[c];
            v->suf = y;
            if (y->mark) v->nxt = y;
            else v->nxt = y->nxt;
            q.push(v);
}}}
void search(Node *x, char *s) {
    for (int i = 0; s[i]; i++) {
        int c = s[i] - 'a';
        while (x->suf && !x->ch[c]) x = x->suf;
        if (x->ch[c]) x = x->ch[c];
        if (x->mark) print(i + 1, x->data);
        for (Node *y = x->nxt; y; y = y->nxt) print(i + 1, y->data);
}}
```

3. 后缀排序：倍增算法

倍增法后缀排序，时间复杂度为 $\Theta(n \log n)$。`suffix_sort` 是本体，结果输出到 `sa` 数组和 `rk` 数组（排名数组）。参数 `s` 是字符串，下标从 0 开始，n 是字符串长度，m 是字符集大小（一般为 255，字符集为 $\Sigma = \{0, 1, 2, ..., m\}$，0 是保留的 `$` 字符）。算法运行完毕后 `sa` 数组里面存的是从 0 开始的下标，`rk` 数组里面存的是从 1 开始的排名值。

另外附带一个线性求 `lcp` 数组的代码。`lcp` 数组下标从 1 开始，实际上只有在 2 到 n 范围内的才是有效值。参数意义与 `suffix_sort` 相同。

```
static int sa[NMAX + 10], rk[NMAX + 10], lcp[NMAX + 10];
void suffix_sort(const char *s, int n, int m) {
    static int x[NMAX + 10], y[NMAX + 10], cnt[NMAX + 10], i;
    for (i = 0; i < n; i++) cnt[s[i]]++;
    for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
    for (i = 0; i < n; i++) sa[--cnt[s[i]]] = i;
    for (i = 1, m = 1, rk[sa[0]] = 1; i < n; i++) {
        if (s[sa[i - 1]] != s[sa[i]]) m++;
        rk[sa[i]] = m;
    }
    for (int l = 1; l < n; l <<= 1) {
        memset(cnt, 0, sizeof(int) * (m + 1));
        for (i = 0; i < n; i++) cnt[y[i] = i + l < n ? rk[i + l] : 0]++;
        for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for (i = n - 1; i >= 0; i--) x[--cnt[y[i]]] = i;
        memset(cnt, 0, sizeof(int) * (m + 1));
        for (i = 0; i < n; i++) cnt[rk[i]]++;
        for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for (i = n - 1; i >= 0; i--) sa[--cnt[rk[x[i]]]] = x[i];
        for (i = 1, m = 1, x[sa[0]] = 1; i < n; i++) {
```

```
            if (rk[sa[i - 1]] != rk[sa[i]] || y[sa[i - 1]] != y[sa[i]]) m++;
            x[sa[i]] = m;
        }
        memcpy(rk, x, sizeof(int) * n);
}}
void compute_lcp(const char *s, int n) {
    int j = 0, p;
    for (int i = 0; i < n; i++, j = max(0, j - 1)) {
        if (rk[i] == 1) {
            j = 0;
            continue;
        }
        p = sa[rk[i] - 2];
        while (p + j < n && i + j < n && s[p + j] == s[i + j]) j++;
        lcp[rk[i]] = j;
}}
```

4. 后缀排序：SA-IS

SA-IS 后缀数组排序。字符串存在 `str` 中，下标从 1 开始，长度为 n，并且 `str[n + 1]` 为哨兵字符，编号为 `1`。后缀数组放在 `sa` 中，下标从 1 开始。时空复杂度为 $\Theta(n)$。其中使用了 `vector<bool>` 来优化缓存命中率。

```
#define rep(i, l, r) for (register int i = (l); i <= (r); ++i)
#define rrep(i, r, l) for (register int i = (r); i >= (l); --i)
#define PUTS(x) sa[cur[str[x]]--] = x
#define PUTL(x) sa[cur[str[x]]++] = x
#define LMS(x) (!type[x - 1] && type[x])
#define RESET memset(sa + 1, 0, sizeof(int) * (n + 1));
    memcpy(cur + 1, cnt + 1, sizeof(int) * m);
#define INDUCE rep(i, 1, m) cur[i] = cnt[i - 1] + 1;
    rep(i, 1, n + 1) if (sa[i] > 1 && !type[sa[i] - 1]) PUTL(sa[i] - 1);
    memcpy(cur + 1, cnt + 1, sizeof(int) * m);
    rrep(i, n + 1, 1) if (sa[i] > 1 && type[sa[i] - 1]) PUTS(sa[i] - 1);
void sais(int n, int m, int *str, int *sa) {
    static int id[NMAX + 10];
    vector<bool> type(n + 2);
    type[n + 1] = true;
    rrep(i, n, 1) type[i] = str[i] == str[i + 1] ? type[i + 1] : str[i] < str[i + 1];
    int cnt[m + 1], cur[m + 1], idx = 1, y = 0, rt, lrt, *ns = str + n + 2, *nsa = sa + n + 2;
    memset(cnt, 0, sizeof(int) * (m + 1));
    rep(i, 1, n + 1) cnt[str[i]]++;
    rep(i, 1, m) cnt[i] += cnt[i - 1];
    RESET rep(i, 2, n + 1) if (LMS(i)) PUTS(i); INDUCE
    memset(id + 1, 0, sizeof(int) * n);
    rep(i, 2, n + 1) if (LMS(sa[i])) {
        register int x = sa[i];
        for (rt = x + 1; !LMS(rt); rt++) ;
        id[x] = y && rt + y == lrt + x && !memcmp(str + x, str + y, sizeof(int) * (rt - x + 1)) ? idx : ++idx;
        y = x, lrt = rt;
    }
    int len = 0, pos[(n >> 1) + 1];
    rep(i, 1, n) if (id[i]) {
        ns[++len] = id[i];
        pos[len] = i;
    }
    ns[len + 1] = 1, pos[len + 1] = n + 1;
    if (len == idx - 1) rep(i, 1, len + 1) nsa[ns[i]] = i;
    else sais(len, idx, ns, nsa);
    RESET rrep(i, len + 1, 1) PUTS(pos[nsa[i]]); INDUCE
}
static int str[NMAX * 3 + 10], sa[NMAX * 3 + 10];
```

5. 线性筛 & 杜教筛

计算积性函数 $f(n)$ 的前缀和 $F(n)=\sum_{k=1}^n f(k)$：先选定辅助函数 $g(n)$ 进行 Dirichlet 卷积，得到递推公式：

$$F(n)=\frac{1}{g(1)}\left(\sum_{k=1}^n (f\times g)(k)-\sum_{k=2}^n g(k)F\left(\left\lfloor\frac{n}{k}\right\rfloor\right)\right)$$

对于 Euler 函数 $\varphi(n)$，选定 $g(n)=1$，得：

$$\Phi(n)=\frac{n(n+1)}{2}-\sum_{k=2}^n \Phi\left(\left\lfloor\frac{n}{k}\right\rfloor\right)$$

对于 Mobius 函数 $\mu(n)$，选定 $g(n)=1$，得：

$$\mathrm{M}(n)=1-\sum_{k=2}^n \mathrm{M}\left(\left\lfloor\frac{n}{k}\right\rfloor\right)$$

如果没有预处理，时间复杂度为 $\Theta(n^{3/4})$，空间复杂度为 $\Theta(\sqrt{n})$。如果预处理前 $\Theta(n^{2/3})$ 项前缀和，则时空复杂度均变为 $\Theta(n^{2/3})$。下面的代码以 Euler 函数为例，能够在 1s 内计算 10^{10} 内的数据。可以多次调用。

```
#define S 17000000  // for F(10^10)
static int pc, pr[S + 10];
static i64 phi[S + 10];
static unordered_map<i64, i64> dat;
inline void sub(i64 &a, i64 b) { a -= b; if (a < 0) a += MOD; }
```

```cpp
inline i64 c2(i64 n) { n %= MOD; return n * (n + 1) % MOD * INV2 % MOD; }
i64 F(i64 n) {  // 杜教筛
    if (n <= S) return phi[n];
    if (dat.count(n)) return dat[n];
    i64 &r = dat[n] = c2(n);
    for (i64 i = 2, l; i <= n; i = l + 1) {
        i64 p = n / i;
        l = n / p;
        sub(r, (l - i + 1) * F(p) % MOD);  // (l - i + 1) % MOD?
    }
    return r;
}
phi[1] = 1;  // 线性筛
for (int i = 2; i <= S; i++) {
    if (!phi[i]) {
        pr[pc++] = i;
        phi[i] = i - 1;
    }
    for (int j = 0; pr[j] * i <= S; j++) {
        int p = pr[j];
        if (i % p) phi[i * p] = phi[i] * (p - 1);
        else {
            phi[i * p] = phi[i] * p;
            break;
}}}
for (int i = 2; i <= S; i++) add(phi[i], phi[i - 1]);
```

6. 类 Euclid 算法

类 Euclid 算法在模意义下计算：

$$\sum_{k=0}^{n} k^p \left\lfloor \frac{ak+b}{c} \right\rfloor^q$$

其中所有参数非负，在计算过程中始终保证 $K = p + q$ 不增，$a,\ c \geqslant 1$ 且 $b \geqslant 0$。需要 Bernoulli 数（$B_1 = +1/2$）来计算自然数幂前缀和 $S_p(x) = \sum_{k=1}^{x} k^p = \sum_{k=1}^{p+1} a_k^{(p)} x^k$，其中 $a_k^{(p)} = \frac{1}{p+1}\binom{p+1}{k} B_{p+1-k}$。代码中 has 为访问标记数组，每次使用前需清空，val 为记忆化使用的数组，qpow 是快速幂，S 是自然数幂前缀和，A 记录了 $a_k^{(p)}$，C 是组合数。时空复杂度为 $\mathrm{O}(K^3 \log \max\{a,\ c\})$。

算法主要分为三个情况，其中 $a \geqslant c$ 和 $b \geqslant c$ 的情况比较简单。当 $a,\ b < c$ 时，用 $j = \lfloor (ak+b)/c \rfloor$ 进行代换，注意最终要转化为 $\lfloor (c(j-1)+c-b-1)/a \rfloor < k \leqslant \lfloor (cj+c-b-1)/a \rfloor$，再进行一次分部求和即可。注意处理 $k \leqslant n$ 这个条件。

```cpp
i64 F(i64 n, i64 a, i64 b, i64 c, int p, int q, int d = 0) {
    if (n < 0) return 0;
    if (has[d][p][q]) return val[d][p][q];
    has[d][p][q] = true;
    i64 &ret = val[d++][p][q] = 0;  // 后面的 d 均加 1
    if (!q) ret = S(n, p) + (!p);   // 注意 p = 0 的边界情况
    else if (!a) ret = qpow(b / c, q) * (S(n, p) + (!p)) % MOD;
    else if (a >= c) {
        i64 m = a / c, r = a % c, mp = 1;
        for (int j = 0; j <= q; j++, mp = mp * m % MOD)
            add(ret, C[q][j] * mp % MOD * F(n, r, b, c, p + j, q - j, d) % MOD);
    } else if (b >= c) {
        i64 m = b / c, r = b % c, mp = 1;
        for (int j = 0; j <= q; j++, mp = mp * m % MOD)
            add(ret, C[q][j] * mp % MOD * F(n, a, r, c, p, q - j, d) % MOD);
    } else {
        i64 m = (a * n + b) / c;
        for (int k = 0; k < q; k++) {
            i64 s = 0;
            for (int i = 1; i <= p + 1; i++)
                add(s, A[p][i] * F(m - 1, c, c - b - 1, a, k, i, d) % MOD);
            add(ret, C[q][k] * s % MOD);
        }
        ret = (qpow(m, q) * S(n, p) - ret) % MOD;
    } return ret;
}
```

7. 构造圆方树

G 用于存图，T 是构造的圆方树。只有一个点的点双没有添加方点。

```cpp
static vector<int> G[NMAX + 10], T[NMAX + 10];
void bcc(int u, int f = 0) {
    static stack<Pair> stk;
    static bool marked[NMAX + 10];
    static int in[NMAX + 10], low[NMAX + 10], cur;
    in[u] = low[u] = ++cur;
    for (int v : G[u]) {
        if (v == f) f = 0;  // 应对重边
        else if (in[v]) low[u] = min(low[u], in[v]);
        else {
            stk.push(Pair(u, v));  // stk 内存储 DFS 树上的边
            bcc(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > in[u]) {  // 割边 u - v
                T[u].push_back(v);
```

```cpp
                T[v].push_back(u);
                stk.pop();
            } else if (low[v] >= in[u]) {  // 可能有点双了
                cnt++;
                int linked = 0, p = n + cnt;  // linked 点数, p 圆方树上的新方点
                auto add = [p, &linked](int x) {
                    if (!marked[x]) {
                        marked[x] = true;
                        T[p].push_back(x);
                        T[x].push_back(p);
                        linked++;
                }};
                while (!stk.empty()) {
                    Pair x = stk.top();
                    stk.pop();
                    add(x.u);
                    add(x.v);
                    if (x.u == u && x.v == v) break;
                }
                for (int v : T[p]) marked[v] = false;
                if (linked == 0) cnt--;  // 假点双
}}}}
```

8. 最小树形图：朴素算法

给定一张 n 个点 m 条边的带权有向图，求以 r 为根的最小树形图上的边权总和，如果不存在输出 `-1`。时间复杂度为 $\mathrm{O}(nm)$。调用 `mdst(r)` 获得答案，调用前需清空 `id` 数组。如要求不定根的最小树形图，可以额外添加一个节点，向原图中的每个点连接一条边权为 ∞ 的边。

```cpp
static int n, m, G[NMAX + 10], nxt[MMAX + 10];
static struct Edge { int u, v, w; } E[MMAX + 10], *in[NMAX + 10];
static int id[NMAX + 10], mark[NMAX + 10];
int find(int x) { return id[x] ? id[x] = find(id[x]) : x; }
int dfs(int x) {
    mark[x] = 1; int ret = 1;
    for (int i = G[x]; i; i = nxt[i])
        if (!mark[E[i].v]) ret += dfs(E[i].v);
    return ret;
}
inline int detect(int x) {
    mark[x] = x;
    for (int y = in[x]->u; in[y]; y = in[y]->u)
        if (mark[y]) return mark[y] == x ? y : 0;
        else mark[y] = x;
    return 0;
}
int mdst(int r) {
    if (dfs(r) < n) return -1;
    int ret = 0;
    while (true) {
        memset(in, 0, sizeof(in));
        memset(mark, 0, sizeof(mark));
        for (auto *e = E + 1; e <= E + m; e++)
            if (e->u != e->v && e->v != r && (!in[e->v] || e->w < in[e->v]->w))
                in[e->v] = e;
        int p = 0, t = 0;
        for (int x = 1; x <= n; x++, t |= p) if (!mark[x] && in[x]) {
            if (!(p = detect(x))) continue;
            ret += in[p]->w;
            for (int x = in[p]->u; x != p; x = in[x]->u)
                id[find(x)] = p, ret += in[x]->w;
            for (auto *e = E + 1; e <= E + m; e++) {
                int u = find(e->u), v = find(e->v);
                if (u != p && v == p) e->w -= in[e->v]->w;
                e->u = u; e->v = v;
        }}
        if (!t) break;
    }
    for (int x = 1; x <= n; x++) if (in[x]) ret += in[x]->w;
    return ret;
}
```

9. 单纯型

```cpp
#define EPS 1e-10
#define INF 1e100

class Simplex {
 public:
    void initialize() {
        scanf("%d%d%d", &n, &m, &t);

        memset(A, 0, sizeof(A));
        for (int i = 1; i <= n; i++) {
            idx[i] = i;
            scanf("%Lf", A[0] + i);
```

```
        }

        for (int i = 1; i <= m; i++) {
            idy[i] = n + i;
            for (int j = 1; j <= n; j++) {
                scanf("%Lf", A[i] + j);
                A[i][j] *= -1;
            }

            scanf("%Lf", A[i]);
        }
    }

    void solve() {
        srand(time(0));

        while (true) {
            int x = 0, y = 0;
            for (int i = 1; i <= m; i++) {
                if (A[i][0] < -EPS && (!y || (rand() & 1)))
                    y = i;
            }

            if (!y)
                break;

            for (int i = 1; i <= n; i++) {
                if (A[y][i] > EPS && (!x || (rand() & 1)))
                    x = i;
            }

            if (!x) {
                puts("Infeasible");
                return;
            }

            pivot(x, y);
        }

        while (true) {
            double k = INF;
            int x, y;
            for (x = 1; x <= n; x++) {
                if (A[0][x] > EPS)
                    break;
            }

            if (x > n)
                break;

            for (int i = 1; i <= m; i++) {
                double d = A[i][x] > -EPS ? INF : -A[i][0] / A[i][x];
                if (d < k) {
                    k = d;
                    y = i;
                }
            }

            if (k >= INF) {
                puts("Unbounded");
                return;
            }

            pivot(x, y);
        }

        printf("%.10Lf\n", A[0][0]);

        if (t) {
            static double ans[NMAX + 10];
            for (int i = 1; i <= m; i++) {
                if (idy[i] <= n)
                    ans[idy[i]] = A[i][0];
            }

            for (int i = 1; i <= n; i++) {
                printf("%.10Lf ", ans[i]);
            }
            printf("\n");
        }
    }

private:
    void pivot(int x, int y) {
        swap(idx[x], idy[y]);
```

```
            double r = -A[y][x];
            A[y][x] = -1;
            for (int i = 0; i <= n; i++) {
                A[y][i] /= r;
            }

            for (int i = 0; i <= m; i++) {
                if (i == y)
                    continue;

                r = A[i][x];
                A[i][x] = 0;
                for (int j = 0; j <= n; j++) {
                    A[i][j] += r * A[y][j];
                }
            }
        }

    int n, m, t;
    double A[NMAX + 10][NMAX + 10];
    int idx[NMAX + 10], idy[NMAX + 10];
};
```