```
{
    "IO": {
        "prefix": "io",
        "body": [
            "namespace IO{",
            "    char ibuf[1<<21],*ip=ibuf,*ip_=ibuf;",
            "    char obuf[1<<21],*op=obuf,*op_=obuf+(1<<21);",
            "    inline char gc(){",
            "        if(ip!=ip_)return *ip++;",
            "        ip=ibuf;ip_=ip+fread(ibuf,1,1<<21,stdin);",
            "        return ip==ip_?EOF:*ip++;",
            "    }",
            "    inline void pc(char c){",
            "        if(op==op_)fwrite(obuf,1,1<<21,stdout),op=obuf;",
            "        *op++=c;",
            "    }",
            "    inline int read(){",
            "        int x=0,ch=gc(),w=1;",
            "        for(;ch<'0'||ch>'9';ch=gc())if(ch=='-')w=-1;",
            "        for(;ch>='0'&&ch<='9';ch=gc())x=x*10+ch-48;",
            "        return w*x;",
            "    }",
            "    template<class I>",
            "    inline void write(I x){",
            "        if(x<0)pc('-'),x=-x;",
            "        if(x>9)write(x/10);pc(x%10+'0');",
            "    }",
            "    class flusher_{",
            "    public:",
            "        ~flusher_(){if(op!=obuf)fwrite(obuf,1,op-obuf,stdout);}",
            "    }IO_flusher;",
            "}",
            "using namespace IO;"
        ],
        "description": "IO"
    },
    "dijkstra": {
        "prefix": "dij",
        "body": [
            "namespace dij{",
            "    const int N = 1e7 + 10;",
            "    const int INF = 0x3f3f3f3f;",
            "    struct Edge",
            "    {",
            "        int u, v, w, ne;",
            "    }e[N << 2];",
            "    int h[N], cnt, dist[N], vis[N];",
            "",
            "    void add(int u, int v, int w)",
            "    {",
            "        e[cnt].u = u;",
            "        e[cnt].v = v;",
            "        e[cnt].w = w;",
            "        e[cnt].ne = h[u];",
            "        h[u] = cnt++;",
            "    }",
            "    ",
            "    void dijkstra(int s, int d[]){",
            "        memset(vis, 0, sizeof vis);",
            "        memset(d, 0x3f, sizeof dist);",
            "        priority_queue <PII, vector<PII>, greater<PII>> q;",
            "        d[s] = 0;",
            "        q.push({0, s}); ",
            "        while(q.size())",
            "        {",
            "            auto ns = q.top();",
            "            q.pop();",
            "            int x = ns.second;",
            "            if(vis[x]) continue;",
            "            vis[x] = 1;",
            "            for(int i = h[x]; ~i ; i = e[i].ne)",
```

```
            "            {",
            "                int to = e[i].v;",
            "                if(d[to] > d[x] + e[i].w)",
            "                {",
            "                    d[to] = d[x] + e[i].w;",
            "                    q.push({d[to], to});",
            "                }",
            "            }",
            "        }",
            "    }",
            "}",
            "using namespace dij;"
        ],
        "description": "dijkstra"
    },
    "二分—找右": {
        "prefix": "erfen1",
        "body": [
            "int l = 1, r = n;",
            "while(l < r){",
            "\tint mid = l + r + 1 >> 1;",
            "\tif(check(mid)) l = mid;",
            "\telse r = mid - 1;",
            "}",
            "cout << l << endl;"
        ],
        "description": "二分—找右"
    },
    "二分—找左": {
        "prefix": "erfen2",
        "body": [
            "int l = 1, r = n;",
            "while(l <= r){",
            "\tint mid = l + r >> 1;",
            "\tif(check(mid)) r = mid - 1;",
            "\telse l = mid + 1;",
            "}",
            "if(l >= 1 && l <= n) cout << l << endl;",
            "else //l不存在"
        ],
        "description": "二分—找左"
    },
    "二进制枚举": {
        "prefix": "binary_enumeration",
        "body": [
            "void binary_enumeration(int n, int a[]){",
            "    int sum = 0;",
            "    for(int i = 0; i < (1 << n); i++){",
            "        sum = 0;",
            "        for(int j = 0; j < n; j++){",
            "            if( i & (1 << j) ){",
            "                sum += a[j];",
            "            }",
            "        }",
            "        //if() {",
            "            // 满足什么条件",
            "        //}",
            "    }",
            "}"
        ],
        "description": "二进制枚举"
    },
    "求因子和": {
        "prefix": "yinzi_sum",
        "body": [
            "//以线性筛为基础",
            "",
            "int jet_num[N];//用来记录素数的幂是多少的",
            "",
            "LL get_yin_zi_sum(LL n)",
            "{",
```

```
            "    LL ans = 1;",
            "    for(int i = 1; (LL) prime[i] * prime[i] <= n; i++){",
            "        if(!(n % prime[i])){",
            "            LL jet = 1, sum = 1;",
            "            while(!(n % prime[i])){",
            "                jet_num[prime[i]]++;",
            "                jet *= prime[i];",
            "                sum += jet;",
            "                n /= prime[i];",
            "            }",
            "            ans *= sum;",
            "        }",
            "    }",
            "    if(n > 1) {",
            "        jet_num[n]++;",
            "        ans *= (n + 1);",
            "    }",
            "    return ans;",
            "}",
            ""
        ],
        "description": "求因子和"
    },
    "求因子个数": {
        "prefix": "yinzi_num",
        "body": [
            "//以线性筛为基础",
            "LL get_yin_zi_num(LL n)",
            "{",
            "    LL ans = 1;",
            "    for(int i = 1; (LL) prime[i] * prime[i] <= n; i++){",
            "        if(!(n % prime[i])){",
            "            LL  cnt = 0;",
            "            while(!(n % prime[i])){",
            "                cnt ++;",
            "                n /= prime[i];",
            "            }",
            "            ans *= (1 + cnt);",
            "        }",
            "    }",
            "    if(n > 1)  ans *= 2;",
            "    return ans;",
            "}",
            ""
        ],
        "description": "求因子个数"
    },
    "素因子分解": {
        "prefix": "prime_factorization",
        "body": [
            "vector < int > v;",
            "for(int j = 2; j <= a / j; j++){",
            "    while(a % j == 0){",
            "        a /= j;",
            "        v.push_back(j);",
            "    }",
            "}",
            "if(a > 1) v.push_back(a);",
            ""
        ],
        "description": "素因子分解"
    },
    "01背包": {
        "prefix": "bag_01",
        "body": [
            "for(int i = 1; i <= n; i++){",
            "    for(int j = v; j >= c[i]; j--){",
            "        dp[j] = max(dp[j], dp[j - c[i]] + w[i]);",
            "    }",
            "}"
        ],
```

```
        "description": "01背包"
    },
    "二分图匹配": {
        "prefix": "erfen_graph_matching",
        "body": [
            "bool vis[N];",
            "int match[N];",
            "bool dfs(int u)",
            "{",
            "    for(int i = h[u]; ~i; i = e[i].ne){",
            "        int v = e[i].v;",
            "        if(!vis[v]){",
            "            vis[v] = true;",
            "            if(!match[v] || dfs(match[v])){",
            "                match[v] = u;",
            "                return true;",
            "            }",
            "        }",
            "    }",
            "    return false;",
            "}"
        ],
        "description": "二分图匹配"
    },
    "链式前向星": {
        "prefix": "graph",
        "body": [
            "const int N = 1e6 + 10;",
            "struct Edge",
            "{",
            "    int v, ne;",
            "}e[N << 2];",
            "int h[N];",
            "int cnt;",
            "void add(int u, int v)",
            "{",
            "    e[cnt].v = v;",
            "    e[cnt].ne = h[u];",
            "    h[u] = cnt++;",
            "}",
            "void init(){",
            "    memset(h, -1, sizeof(h));",
            "    cnt = 0;",
            "}"
        ],
        "description": "链式前向星"
    },
    "矩阵快速幂": {
        "prefix": "q_Matrix",
        "body": [
            "namespace Q_Matrix{",
            "    const int MAX = 10;",
            "    typedef struct{",
            "        LL m[MAX][MAX];",
            "    }Matrix;",
            "",
            "    Matrix P;//构造出的矩阵",
            "",
            "    LL k,mod,a[MAX];",
            "    Matrix matrixmul(Matrix a,Matrix b) //矩阵乘法",
            "    {",
            "        int i,j,k;",
            "        Matrix c;",
            "        for (i = 0 ; i < MAX; i++)",
            "            for (j = 0; j < MAX;j++)",
            "            {",
            "                c.m[i][j] = 0;",
            "                for (k = 0; k < MAX; k++)",
            "                    c.m[i][j] =(c.m[i][j]+(a.m[i][k]*b.m[k][j]))%mod;",
            "                c.m[i][j] %=mod;",
            "            }",
```

```
            "        return c;",
            "    }",
            "",
            "    Matrix quickpow(Matrix m , LL n)",
            "    {",
            "        Matrix b;//单位矩阵在这构造也可以",
            "        for(int i=0;i<MAX;i++)",
            "            for(int j=0;j<MAX;j++)",
            "            {",
            "                if(i==j)b.m[i][j]=1;",
            "                else b.m[i][j]=0;",
            "            }",
            "        while (n >= 1)",
            "        {",
            "            if (n & 1)",
            "                    b = matrixmul(b,m);",
            "            n = n >> 1;",
            "            m = matrixmul(m,m);",
            "        }",
            "        return b;",
            "    }",
            "}",
            "using namespace Q_Matrix;",
            ""
        ],
        "description": "矩阵快速幂"
    },
    "树状数组": {
        "prefix": "tree_array",
        "body": [
            "const int N = 1e6 + 10;",
            "int tree[N], n;",
            "void add(LL x,LL d)",
            "{",
            "    while(x <= n){",
            "        tree[x] += d;",
            "        x += lowbit(x);//查询x的后继们",
            "    }",
            "}",
            "LL sum(LL x)//前缀和",
            "{",
            "    LL sum = 0;",
            "    while(x > 0){",
            "        sum += tree[x];",
            "        x -= lowbit(x);//查询x的前驱们",
            "    }",
            "    return sum;",
            "}",
            ""
        ],
        "description": "树状数组"
    },
    "最小生成树": {
        "prefix": "kruskal",
        "body": [
            "namespace kruskal{",
            "    int n, m, u, v;",
            "    LL w, ans, cnt;",
            "    const int N = 1e6 + 10;",
            "    int fa[N];",
            "    struct sa{",
            "        int u,v;",
            "        LL w;",
            "    }e[N];",
            "",
            "    bool cmp(struct sa x,struct sa y){",
            "        return x.w < y.w;",
            "    }",
            "",
            "    void init(){",
            "        // n点，m条边",
```

```
            "        cin >> n >> m;",
            "        for(int i = 1; i <= m; i++) cin >> e[i].u >> e[i].v >> e[i].w;",
            "        sort(e + 1, e + 1 + m, cmp);",
            "        for(int i = 1; i <= n; i++) fa[i] = i;",
            "    }",
            "",
            "    int find(int x){",
            "        return fa[x] == x ? x : fa[x] = find(fa[x]);",
            "    }",
            "",
            "",
            "    void solve(){",
            "        for(int i = 1; i <= m; i++){",
            "            if(cnt == n - 1) break;",
            "            w = e[i].w;",
            "            u = find(e[i].u);",
            "            v = find(e[i].v);",
            "            if(u != v){",
            "                fa[u] = v;",
            "                ans += w;",
            "                cnt ++;",
            "            }",
            "            if(cnt == n - 1) break;",
            "        }",
            "    }",
            "}",
            "using namespace kruskal;"
        ],
        "description": "最小生成树"
    },
    "并查集": {
        "prefix": "DSU",
        "body": [
            "namespace union_set{",
            "    const int N = 1e6 + 10;",
            "    int fa[N], n;",
            "    void init ()",
            "    {",
            "        for(int i = 1;i <= n; i++) fa[i] = i;",
            "    }",
            "    int find(int x){",
            "        return fa[x] == x ? x : fa[x] = find(fa[x]);",
            "    }",
            "    void join(int a,int b)",
            "    {",
            "        int a1 = find(a),b1 = find(b);",
            "        if(a1 != b1) fa[a1] = b1;",
            "    }",
            "}",
            "using namespace union_set;"
        ],
        "description": "并查集"
    },
    "素数筛": {
        "prefix": "prime_sieve",
        "body": [
            "namespace prime_sieve",
            "{",
            "    const int N = 5e4 + 5;",
            "    int cnt, prime[N];",
            "    bool flag[N];",
            "    inline void init()",
            "    {",
            "        memset(flag, 1, sizeof(flag));",
            "        flag[0] = flag[1] = 0;",
            "        for (int i = 2; i <= N; i++)",
            "        {",
            "            if (flag[i])",
            "            {",
            "                prime[++cnt] = i;",
            "            }",
```

```
            "            for (int j = 1; j <= cnt && 1LL * prime[j] * i <= N; j++)",
            "            {",
            "                flag[prime[j] * i] = 0;",
            "                if (i % prime[j] == 0)",
            "                    break;",
            "            }",
            "        }",
            "    }",
            "}",
            "using namespace prime_sieve;"
        ],
        "description": "素数筛"
    },
    "快速幂": {
        "prefix": "q_pow",
        "body": [
            "LL qm (LL a, LL b, LL c){",
            "    LL ret = 1 % c;",
            "    while(b){",
            "        if(b & 1)",
            "            ret = ret * a % c;",
            "        a = a * a % c;",
            "        b = b >> 1;",
            "    }",
            "    return ret;",
            "}",
        ],
        "description": "快速幂"
    },
    "按权相加法": {
        "prefix": "jinzhi",
        "body": [
            "int base_conversion(string str, int k)",
            "{",
            "    int ans = 0;",
            "    for(int i = 0; i < str.size(); i++){",
            "        ans = ans * k + (str[i] - '0');",
            "    }",
            "    return ans;",
            "}",
            ""
        ],
        "description": "按权相加法"
    },
    "spfa": {
        "prefix": "spfa",
        "body": [
            "namespace spfa_{",
            "    const int N = 1e7 + 10;",
            "    const int inf = 0x3f3f3f3f;",
            "    struct Edge",
            "    {",
            "        int u, v, w, ne;",
            "    }e[N << 2];",
            "",
            "    int h[N], number[N];",
            "    int cnt;",
            "",
            "    void init(){",
            "        memset(h, -1, sizeof(h));",
            "        memset(number, 0, sizeof(number));",
            "        cnt = 0;",
            "    }",
            "",
            "    void add(int u, int v, int w)",
            "    {",
            "        e[cnt].u = u;",
            "        e[cnt].v = v;",
            "        e[cnt].w = w;",
            "        e[cnt].ne = h[u];",
            "        h[u] = cnt++;",
```

```
            "    }",
            "    int dis[N], vis[N], n;",
            "",
            "    int spfa(int s, int d[]){",
            "        queue < int > q;",
            "        memset(vis, 0, sizeof(vis));",
            "        for(int i = 0; i <= n; i++) d[i] = inf; //最短路",
            "        d[s] = 0;",
            "        vis[s] = 1;",
            "        q.push(s);",
            "        while(!q.empty()){",
            "            int u = q.front();",
            "            q.pop();",
            "            vis[u] = 0;",
            "            for(int i = h[u]; ~i; i = e[i].ne){",
            "                int v = e[i].v, w = e[i].w;",
            "                if(d[v] > d[u] + w){  //最短路",
            "                    d[v] = d[u] + w;",
            "                    if( !vis[v] ){",
            "                        vis[v] = 1;",
            "                        q.push(v);",
            "                        number[v] ++;",
            "                        if(number[v] == n) return 0;",
            "                    }",
            "                }",
            "            }",
            "        }",
            "        return 1;",
            "    }",
            "}",
            "using namespace spfa_;"
        ],
        "description": "spfa"
    },
    "素数筛+欧拉函数": {
        "prefix": "prime_sieve_phi",
        "body": [
            "namespace prime_sieve_phi{",
            "    const int N = 5e4 + 5;",
            "    int  cnt, ans, prime[N], pre[N], phi[N];",
            "    bool flag[N];",
            "    inline void init()",
            "    {",
            "        memset(flag, 1, sizeof(flag));",
            "        flag[1] = 0;",
            "        cnt = 0;",
            "        phi[1] = 1;",
            "        for(int i = 2; i <= N; i++)",
            "        {",
            "            if(flag[i])",
            "            {",
            "                prime[++cnt] = i;",
            "                pre[i] = cnt;",
            "                phi[i] = i - 1;",
            "            }",
            "            for(int j = 1; j <= cnt && 1LL * prime[j] * i <= N; j++)",
            "            {",
            "                flag[prime[j] * i] = 0;",
            "                if(i % prime[j] == 0) {",
            "                    phi[i * prime[j]] = phi[i] * prime[j];",
            "                    break;",
            "                }",
            "                phi[i * prime[j]] = phi[i] * (prime[j] - 1);",
            "            }",
            "        }",
            "    }",
            "}",
            "using namespace prime_sieve_phi;"
        ],
        "description": "素数筛+欧拉函数"
    },
```

```
    "素数筛+莫比乌斯": {
        "prefix": "prime_sieve_mu",
        "body": [
            "namespace prime_sieve_mu{",
            "    const int N = 5e4 + 5;",
            "    int  cnt, ans, prime[N], pre[N], mu[N];",
            "    bool flag[N];",
            "    inline void init()",
            "    {",
            "        memset(flag, 1, sizeof(flag));",
            "        flag[1] = 0;",
            "        mu[1] = 1;",
            "        cnt = 0;",
            "        for(int i = 2; i <= N; i++)",
            "        {",
            "            if(flag[i])",
            "            {",
            "                prime[++cnt] = i;",
            "                pre[i] = cnt;",
            "                mu[i] = -1;",
            "            }",
            "            for(int j = 1; j <= cnt && 1LL * prime[j] * i <= N; j++)",
            "            {",
            "                flag[prime[j] * i] = 0;",
            "                if(i % prime[j] == 0) {",
            "                    mu[prime[j] * i] = 0;",
            "                    break;",
            "                }",
            "                else{",
            "                    mu[prime[j] * i] = -mu[i];",
            "                }",
            "            }",
            "        }",
            "    }",
            "}",
            "using namespace prime_sieve_mu;"
        ],
        "description": "素数筛+莫比乌斯反演"
    },
    "线性筛+因子个数": {
        "prefix": "prime_sieve_sum_yinzi",
        "body": [
            "namespace prime_sieve_sum_yinzi{",
            "\tconst int N = 5e4 + 5;",
            "\tint  cnt, ans, prime[N], pre[N], d[N];",
            "\tbool flag[N];",
            "\tinline void init()",
            "\t{",
            "\t\tmemset(flag, 1, sizeof(flag));",
            "\t\td[1] = 1;",
            "\t\tflag[1] = 0;",
            "\t\tcnt = 0;",
            "\t\tfor(int i = 2; i <= N; i++)",
            "\t\t{",
            "\t\t\tif(flag[i])",
            "\t\t\t{",
            "\t\t\t\tprime[++cnt] = i;",
            "\t\t\t\tpre[i] = cnt;",
            "\t\t\t\td[i] = 2;",
            "\t\t\t}",
            "\t\t\tfor(int j = 1; j <= cnt && 1LL * prime[j] * i <= N; j++)",
            "\t\t\t{",
            "\t\t\t\tflag[prime[j] * i] = 0;",
            "\t\t\t\tif(i % prime[j] == 0) {",
            "\t\t\t\t\tint k = i;",
            "\t\t\t\t\twhile(k % prime[j] == 0) k /= prime[j];",
            "\t\t\t\t\td[i * prime[j]] = d[i] + d[k];",
            "\t\t\t\t\tbreak;",
            "\t\t\t\t}",
            "\t\t\t\telse d[i * prime[j]] = d[i] * d[prime[j]];",
            "\t\t\t}",
```

```
            "\t\t}",
            "\t}",
            "}",
            "using namespace prime_sieve_sum_yinzi;"
        ],
        "description": "线性筛+因子个数,因d也为积性函数，故也可以线性筛出来"
    },
    "线性筛+莫比乌斯+欧拉+前缀和": {
        "prefix": "prime_sieve_phi_mu",
        "body": [
            "namespace prime_sieve_phi_mu{",
            "    const int N = 5e4 + 5;",
            "    int  cnt, ans, prime[N], pre[N], phi[N], mu[N];",
            "    bool flag[N];",
            "    LL sum_phi[N], sum_mu[N];",
            "    inline void init()",
            "    {",
            "        memset(flag, 1, sizeof(flag));",
            "        flag[1] = 0;",
            "        cnt = 0;",
            "        phi[1] = 1;",
            "        mu[1] = 1;",
            "        for(int i = 2; i <= N; i++){",
            "            if(flag[i]){",
            "                prime[++cnt] = i;",
            "                pre[i] = cnt;",
            "                phi[i] = i - 1;",
            "                mu[i] = -1;",
            "            }",
            "            for(int j = 1; j <= cnt && 1LL * prime[j] * i <= N; j++){",
            "                flag[prime[j] * i] = 0;",
            "                if(i % prime[j] == 0) {",
            "                    phi[i * prime[j]] = phi[i] * prime[j];",
            "                    mu[prime[j] * i] = 0;",
            "                    break;",
            "                }",
            "                phi[i * prime[j]] = phi[i] * (prime[j] - 1);",
            "                mu[prime[j] * i] = -mu[i];",
            "            }",
            "        }",
            "        for(int i = 1; i <= N; i++){",
            "            sum_phi[i] = sum_phi[i - 1] + phi[i];",
            "            sum_mu[i] = sum_mu[i - 1] + mu[i];",
            "        }",
            "    }",
            "}",
            "using namespace prime_sieve_phi_mu;"
        ],
        "description": "线性筛+莫比乌斯+欧拉+前缀和"
    },
    "杜教筛（欧拉+莫比乌斯）": {
        "prefix": "djs",
        "body": [
            "unordered_map < LL, LL > phi_w;",
            "unordered_map < LL, LL > mu_w;",
            "",
            "LL djs_phi(LL x){",
            "    if(x <= N) return sum_phi[x];",
            "    if(phi_w[x]) return phi_w[x];",
            "    LL ans = x * (x + 1) / 2;",
            "    for(LL l = 2, r; l <= x; l = r + 1){",
            "        r = x / (x / l);",
            "        ans -= (r - l + 1) * (djs_phi(x / l));",
            "    }",
            "    return phi_w[x] = ans;",
            "}",
            "",
            "LL djs_mu(LL x){",
            "    if(x <= N) return sum_mu[x];",
            "    if(mu_w[x]) return mu_w[x];",
            "    LL ans = 1;",
```

```
            "    for(LL l = 2, r; l <= x; l = r + 1){",
            "        r = x / (x / l);",
            "        ans -= (r - l + 1) * (djs_mu(x / l));",
            "    }",
            "    return mu_w[x] = ans;",
            "}"
        ],
        "description": "杜教筛（欧拉+莫比乌斯）"
    },
    "模运算": {
        "prefix": "mod",
        "body": [
            "const LL mod = 1e9 + 7;",
            "LL mul(LL x, LL y){return 1LL * x * y % mod;}",
            "LL dec(LL x, LL y){return x >= y ? x - y : x + mod - y;}",
            "LL add(LL x, LL y){return x + y >= mod ? x + y - mod : x + y;}",
            "LL pmod(LL x) {return (x + mod) % mod;}",
            ""
        ],
        "description": "模运算"
    },
    "逆元费马小定理": {
        "prefix": "inv_fm",
        "body": [
            "LL inv_fm(LL n, LL p) { return qm(n, p - 2, p); }"
        ],
        "description": "逆元费马小定理"
    },
    "埃氏筛": {
        "prefix": "prime_aishi_sieve",
        "body": [
            "namespace prime_aishi_sieve{",
            "    const int N = 1E6 + 10;",
            "    bool vis[N];",
            "    int prime[N], cnt;",
            "",
            "    void aishi_sieve(){",
            "        for(int i = 0; i <= N; i++) vis[i] = 1;",
            "        vis[0] = vis[1] = 0;",
            "        for(int i = 2; i <= N; i++){",
            "            if(vis[i]){",
            "                prime[++cnt] = i;",
            "                for(int j = i + i; j <= N; j += i){",
            "                    vis[j] = 0;",
            "                }",
            "            }",
            "        }",
            "    }",
            "}",
            "using namespace prime_aishi_sieve;"
        ],
        "description": "埃氏筛"
    },
    "线段树加法": {
        "prefix": "xds_add",
        "body": [
            "namespace xds_add{",
            "    const int N = 1e6 + 10;",
            "    LL a[N << 2], tr[N << 2], add_tag[N << 2], k;",
            "    int n, x, y;",
            "",
            "    inline void pushup(int i)",
            "    {",
            "        tr[i] = tr[ls] + tr[rs];",
            "    }",
            "",
            "    void bulid(int i, int l, int r)",
            "    {",
            "        add_tag[i] = 0;",
            "        if(l == r){",
            "            tr[i] = a[l];",
```

```
            "            return;",
            "        }",
            "        int mid = (l + r) >> 1;",
            "        bulid(ls, l, mid);",
            "        bulid(rs, mid + 1, r);",
            "        pushup(i);",
            "    }",
            "",
            "    inline void ADD(int i, int l, int r, LL k)",
            "    {",
            "        add_tag[i] = (add_tag[i] + k);",
            "        tr[i] = (tr[i] + (r - l + 1) * k);",
            "    }",
            "",
            "    inline void pushdown(int i, int l, int r, int mid)",
            "    {",
            "        if( (!add_tag[i]) ) return;",
            "        ADD(ls, l, mid, add_tag[i]);",
            "        ADD(rs, mid + 1, r, add_tag[i]);",
            "        add_tag[i] = 0;",
            "    }",
            "",
            "    inline void update_ADD (int i, int l, int r, int x, int y, LL k)",
            "    {",
            "        if(l > y || r < x) return;",
            "        if(l >= x && r <= y) return ADD(i, l, r, k);",
            "        int mid = (l + r) >> 1;",
            "        pushdown(i, l, r, mid);",
            "        update_ADD(ls, l, mid, x, y, k);",
            "        update_ADD(rs, mid + 1, r, x, y, k);",
            "        pushup(i);",
            "    }",
            "",
            "    LL query(int i, int l, int r, int x, int y)",
            "    {",
            "        LL res = 0;",
            "        if(l > y || r < x) return 0;",
            "        if(l >= x && r <= y) return tr[i];",
            "        int mid = (l + r) >> 1;",
            "        pushdown(i, l, r, mid);",
            "        if(x <= mid) res = res + query(ls, l, mid, x, y);",
            "        if(y > mid)  res = res + query(rs, mid + 1 , r, x, y);",
            "        return res;",
            "    }",
            "}",
            "using namespace xds_add;"
        ],
        "description": "线段树加法"
    },
    "线段树+最大值": {
        "prefix": "xds_max",
        "body": [
            "namespace xds_max{",
            "    const int N = 1e6 + 10;",
            "    LL a[N << 2], tr[N << 2];",
            "    int n;",
            "",
            "    inline void pushup(int i)",
            "    {",
            "        tr[i] = max(tr[ls], tr[rs]);",
            "    }",
            "",
            "    void bulid(int i, int l, int r)",
            "    {",
            "        if(l == r){",
            "            tr[i] = a[l];",
            "            return;",
            "        }",
            "        int mid = (l + r) >> 1;",
            "        bulid(ls, l, mid);",
            "        bulid(rs, mid + 1, r);",
```

```
            "        pushup(i);",
            "    }",
            "",
            "    inline void update (int i, int l, int r, int x, LL y)",
            "    {",
            "        if(l > x || r < x) return;",
            "        if(l == x && l == r) {tr[i] = y; return;}",
            "        int mid = (l + r) >> 1;",
            "        update(ls, l, mid, x, y);",
            "        update(rs, mid + 1, r, x, y);",
            "        pushup(i);",
            "    }",
            "",
            "    LL query(int i, int l, int r, int x, int y)",
            "    {",
            "        LL res = 0;",
            "        if(l > y || r < x) return 0;",
            "        if(l >= x && r <= y) return tr[i];",
            "        int mid = (l + r) >> 1;",
            "        if(x <= mid) res = max(res, query(ls, l, mid, x, y));",
            "        if(y > mid)  res = max(res, query(rs, mid + 1 , r, x, y));",
            "        return res;",
            "    }",
            "};",
            "using namespace xds_max;"
        ],
        "description": "线段树+最大值"
    },
    "线段树+最小值": {
        "prefix": "xds_min",
        "body": [
            "namespace xds_min{",
            "    const int N = 1e6 + 10;",
            "    LL a[N << 2], tr[N << 2];",
            "    int n;",
            "",
            "    inline void pushup(int i)",
            "    {",
            "        tr[i] = min(tr[ls], tr[rs]);",
            "    }",
            "",
            "    void bulid(int i, int l, int r)",
            "    {",
            "        if(l == r){",
            "            tr[i] = a[l];",
            "            return;",
            "        }",
            "        int mid = (l + r) >> 1;",
            "        bulid(ls, l, mid);",
            "        bulid(rs, mid + 1, r);",
            "        pushup(i);",
            "    }",
            "",
            "    inline void update (int i, int l, int r, int x, LL y)",
            "    {",
            "        if(l > x || r < x) return;",
            "        if(l == x && l == r) {tr[i] = y; return;}",
            "        int mid = (l + r) >> 1;",
            "        update(ls, l, mid, x, y);",
            "        update(rs, mid + 1, r, x, y);",
            "        pushup(i);",
            "    }",
            "",
            "    LL query(int i, int l, int r, int x, int y)",
            "    {",
            "        LL res = INF;",
            "        if(l > y || r < x) return 0;",
            "        if(l >= x && r <= y) return tr[i];",
            "        int mid = (l + r) >> 1;",
            "        if(x <= mid) res = min(res, query(ls, l, mid, x, y));",
            "        if(y > mid)  res = min(res, query(rs, mid + 1 , r, x, y));",
```

```
            "        return res;",
            "    }",
            "}",
            "using namespace xds_min;"
        ],
        "description": "线段树+最小值"
    },
    "KMP": {
        "prefix": "KMP",
        "body": [
            "namespace Kmp{",
            "    const int N = 1E7 + 10;",
            "    string t;",
            "    int Next[N];",
            "    void get_next(string t){",
            "        int i = 0, j = -1;",
            "        int n = t.length();",
            "        Next[0] = -1;",
            "        while (i < n){",
            "            if (j == -1 || t[i] == t[j]){",
            "                i++, j++;",
            "                Next[i] = j;",
            "            }",
            "            else",
            "                j = Next[j];",
            "        }",
            "    }",
            "    bool kmp(string s, string ss){",
            "        int i = 0, j = 0;",
            "        int slen = s.length(), sslen = ss.length();",
            "        get_next(ss);",
            "        while (i < slen && j < sslen){",
            "            if (j == -1 || s[i] == ss[j]){",
            "                i++, j++; //i是主串下标，j是模式串下标",
            "            }",
            "            else",
            "                j = Next[j]; //如果不匹配了，就移动模式串",
            "        }",
            "        if (j == sslen)",
            "            return 1;",
            "        else",
            "            return 0;",
            "    }",
            "}",
            "using namespace Kmp;"
        ],
        "description": "KMP"
    },
    "快速幂+快速乘": {
        "prefix": "q_pow_mm",
        "body": [
            "namespace q_pow_mm{",
            "    LL mm(LL a, LL b, LL m){",
            "        LL ret = 0;",
            "        while(b){",
            "            if(b & 1) ",
            "                ret = (ret + a) % m;",
            "            a = (a * 2) % m;",
            "            b >>= 1;",
            "        }",
            "        return ret;",
            "    }",
            "    LL q (LL a, LL b){",
            "        LL ret = 1;",
            "        while(b){",
            "            if(b & 1)",
            "                ret = ret * a;",
            "            a = a * a;",
            "            b = b >> 1;",
            "        }",
            "        return ret;",
```

```
            "    }",
            "    LL qm (LL a, LL b, LL c){",
            "        a = a % c;",
            "        LL ret = 1 % c;",
            "        while(b){",
            "            if(b & 1)",
            "                ret = mm(ret, a, c) % c;",
            "            a = mm(a, a, c) % c;",
            "            b = b >> 1;",
            "        }",
            "        return ret;",
            "    }",
            "}",
            "using namespace q_pow_mm;"
        ],
        "description": "快速幂+快速乘"
    },
    "欧几里得+拓展": {
        "prefix": "gcd+exgcd",
        "body": [
            "LL gcd(LL a, LL b) {return b == 0 ? a : gcd(b, a % b);}",
            "",
            "LL exgcd(LL a, LL b, LL &x, LL &y){",
            "    if(!b){",
            "        x = 1;",
            "        y = 0;",
            "        return a;",
            "    }",
            "    LL r = exgcd(b, a % b, x, y);",
            "    LL tmp = y;",
            "    y = x - (a / b) * y;",
            "    x = tmp;",
            "    return r;",
            "}"
        ],
        "description": "欧几里得+拓展"
    },
    "逆元exgcd": {
        "prefix": "inv_exgcd",
        "body": [
            "LL exgcd(LL a, LL b, LL &x, LL &y){",
            "    if(!b){",
            "        x = 1;",
            "        y = 0;",
            "        return a;",
            "    }",
            "    LL r = exgcd(b, a % b, x, y);",
            "    LL tmp = y;",
            "    y = x - (a / b) * y;",
            "    x = tmp;",
            "    return r;",
            "}",
            "",
            "LL inv_exgcd(LL n, LL p){",
            "    LL d, x, y;",
            "    d = exgcd(n, p, x, y);",
            "    if(d == 1)",
            "        return (x % p + p) % p;",
            "    else",
            "        return -1;",
            "}"
        ],
        "description": "逆元exgcd"
    },
    "逆元欧拉定理": {
        "prefix": "inv_euler",
        "body": [
            "LL euler(LL n){",
            "    LL ans = n;",
            "    for(int i = 2; i * i <= n; i ++){",
            "        if(!(n % i)){",
```

```
            "            ans = ans / i * (i - 1);",
            "            while(!(n % i)) n /= i;",
            "        }",
            "    }",
            "    if(n > 1) ans = ans / n * (n - 1);",
            "    return ans;",
            "}",
            "",
            "LL inv_euler(LL n, LL p) {return qm(n, euler(p) - 1, p);}"
        ],
        "description": "逆元欧拉定理"
    },
    "ls与rs宏定义": {
        "prefix": "ls",
        "body": [
            "#define ls                          i << 1",
            "#define rs                          i << 1 | 1"
        ],
        "description": "ls与rs宏定义"
    },
    "hashString": {
        "prefix": "hashString",
        "body": [
            "const int base = 131, N = 1e5 + 10;",
            "ULL h[N], p[N] = {1};",
            "",
            "ULL get(int l, int r){",
            "    return h[r] - h[l - 1] * p[r - l + 1];",
            "}"
        ],
        "description": "hashString"
    },
    "杨辉三角": {
        "prefix": "yanghuisanjiao",
        "body": [
            "void init(){",
            "    a[0][0] = 1;",
            "    for(int i = 0; i <= N; ++ i){",
            "        a[i][0] = a[i][i] = 1;",
            "        for(int j = 1; j <= i / 2; ++ j){",
            "            a[i][j] = a[i][i - j] = add(a[i - 1][j - 1], a[i - 1][j]);",
            "        }",
            "    }",
            "}",
            ""
        ],
        "description": "杨辉三角"
    },
    "Lucas求大组合数": {
        "prefix": "lucas",
        "body": [
            "LL inv_fm(LL n, LL p) { ",
            "    return qm(n, p - 2, p); ",
            "}",
            "",
            "LL C(LL n, LL m, LL p){",
            "    return m > n ? 0 : mul(fac[n], mul(inv_fm(fac[m], p), inv_fm(fac[n - m], p)));",
            "}",
            "",
            "LL lucas(LL n, LL m, LL p){",
            "    if(n < m) return 0;",
            "    if(n == m || m == 0) return 1;",
            "    else{",
            "        return mul(C(n % p, m % p, p), lucas(n / p, m / p, p));",
            "    }",
            "}",
            ""
        ],
        "description": "Lucas求大组合数"
    },
```

```
    "逆元阶乘+组合数": {
        "prefix": "inv_fac",
        "body": [
            "const LL N = 1e6 + 10;",
            "LL fac[N], finv[N];",
            "",
            "void init(){",
            "//  需保证 mod > N, 不然不能这么做",
            "    fac[0] = 1;",
            "    for(int i = 1; i <= N; ++ i){",
            "        fac[i] = mul(fac[i - 1], i); ",
            "    }",
            "    finv[N] = qm(fac[N], mod - 2, mod);",
            "    for(int i = N - 1; i >= 1; -- i){",
            "        finv[i] = mul(finv[i + 1], i + 1);",
            "    }",
            "}",
            "LL C(LL n, LL m, LL p){",
            "    return m > n ? 0 : mul(fac[n], mul(finv[m], finv[n - m]));",
            "}",
            ""
        ],
        "description": "逆元阶乘+组合数"
    },
    "逆元线性打表": {
        "prefix": "inv_xian",
        "body": [
            "void init()",
            "{",
            "    inv[1] = 1;",
            "    for (int i = 2; i <= N; i ++)",
            "        inv[i] = mul(dec(mod, mod / i), inv[mod % i]); ",
            "}",
            ""
        ],
        "description": "逆元线性打表"
    },
    "KM": {
        "prefix": "KM",
        "body": [
            "namespace KM{",
            "    const int N = 1e4 + 10;",
            "    int n, ex_L[N], ex_R[N], match[N], slack[N], w[N][N];",
            "    bool vis_L[N], vis_R[N];",
            "    const int INF = 0x3f3f3f3f;",
            "    ",
            "    bool dfs(int u){",
            "        vis_L[u] = true;",
            "        for(int v = 1; v <= n; ++ v){",
            "            if(vis_R[v]) continue;",
            "            int gap = ex_L[u] + ex_R[v] - w[u][v];",
            "            if(!gap){",
            "                vis_R[v] = true;",
            "                if(!match[v] || dfs(match[v])){",
            "                    match[v] = u;",
            "                    return true;",
            "                }",
            "            }",
            "            else{",
            "                slack[v] = min(slack[v], gap);",
            "            }",
            "        }",
            "        return false;",
            "    }",
            "",
            "    int km(){",
            "        memset(match, 0, sizeof match);",
            "        memset(ex_R, 0, sizeof ex_R);",
            "        for(int i = 1; i <= n; ++ i){",
            "            ex_L[i] = w[i][1];",
            "            for(int j = 1; j <= n; ++ j){",
```

```
            "                ex_L[i] = max(ex_L[i], w[i][j]);",
            "            }",
            "        }",
            "        for(int i = 1; i <= n; ++ i){",
            "            memset(slack, 0x3f, sizeof slack);",
            "            while(true){",
            "                memset(vis_L, false, sizeof vis_L);",
            "                memset(vis_R, false, sizeof vis_R);",
            "                if(dfs(i)) break;",
            "                int d = INF;",
            "                for(int j = 1; j <= n; ++ j) ",
            "                    if (!vis_R[j]) d = min(d, slack[j]);",
            "                for(int j = 1; j <= n; ++ j){",
            "                    if(vis_L[j]) ex_L[j] -= d;",
            "                    if(vis_R[j]) ex_R[j] += d;",
            "                    else slack[j] -= d;",
            "                }",
            "            }",
            "        }",
            "        int ans = 0;",
            "        for(int i = 1; i <= n; ++ i){",
            "            if(match[i]) ans += w[match[i]][i];",
            "        }",
            "        return ans;",
            "    }",
            "}",
            "using namespace KM;"
        ],
        "description": "KM(不用重新定义n)"
    },
    "字典树": {
        "prefix": "Trie",
        "body": [
            "namespace Trie{",
            "    const int N = 1e5 + 10;",
            "    int son[N][26], cnt[N], idx;",
            "    char s[N];",
            "    void init(){",
            "        memset(son, 0, sizeof son);",
            "        memset(cnt, 0, sizeof cnt);",
            "        idx = 0;",
            "    }",
            "    void insert(char *s){",
            "        int p = 0;",
            "        for(int i = 0; s[i]; ++ i){",
            "            int u = s[i] - 'a';",
            "            if(!son[p][u]) son[p][u] = ++ idx;",
            "            p = son[p][u];",
            "        }",
            "        cnt[p] ++;",
            "    }",
            "    int query(char *s){",
            "        int p = 0;",
            "        for(int i = 0; s[i]; ++ i){",
            "            int u = s[i] - 'a';",
            "            if(!son[p][u]) return 0;",
            "            p = son[p][u];",
            "        }",
            "        return cnt[p];",
            "    }",
            "}",
            "using namespace Trie;"
        ],
        "description": "字典树"
    },
    "01字典树": {
        "prefix": "Trie01",
        "body": [
            "namespace Trie01{",
            "    const int N = 1e5 + 10;",
            "    int son[N * 32][2], val[N * 32], idx;",
```

```
            "    void init(){",
            "        idx = 0;",
            "        son[0][0] = son[0][1] = 0;",
            "    }",
            "    void insert(int x){",
            "        int p = 0;",
            "        for(int i = 31; i >= 0; -- i){",
            "            int u = (x >> i) & 1;",
            "            if(!son[p][u]){",
            "                son[p][u] = ++ idx;",
            "                p = son[p][u];",
            "                son[p][0] = son[p][1] = 0;",
            "            }",
            "            else p = son[p][u];",
            "        }",
            "        val[p] = x;",
            "    }",
            "    int query(int x){",
            "        int p = 0;",
            "        for(int i = 31; i >= 0; -- i){",
            "            int u = (x >> i) & 1;",
            "            if(son[p][u ^ 1]) p = son[p][u ^ 1];",
            "            else p = son[p][u];",
            "        }",
            "        return val[p];",
            "    }",
            "}",
            "using namespace Trie01;"
        ],
        "description": "01字典树"
    },
    "树链剖分": {
        "prefix": "TreeChain",
        "body": [
            "namespace TreeChain{",
            "    int w[N];",
            "    int pre[N], sizx[N], son[N], deep[N];",
            "    int dfn[N], top[N], a[N];",
            "    int cnx; // dfs2 pool",
            "",
            "    void dfs1(int u, int fa)",
            "    {",
            "        pre[u] = fa;",
            "        deep[u] = deep[fa] + 1;",
            "        sizx[u] = 1;",
            "        int maxson = -1;",
            "        for (int i = h[u]; ~i; i = e[i].ne)",
            "        {",
            "            int v = e[i].v;",
            "            if (v != fa)",
            "            {",
            "                dfs1(v, u);",
            "                sizx[u] += sizx[v];",
            "                if (maxson < sizx[v])",
            "                {",
            "                    maxson = sizx[v];",
            "                    son[u] = v;",
            "                }",
            "            }",
            "        }",
            "    }",
            "",
            "    void dfs2(int u, int t)",
            "    {",
            "        top[u] = t;",
            "        dfn[u] = ++cnx;",
            "        a[cnx] = w[u];",
            "        if (!son[u])",
            "            return;",
            "        dfs2(son[u], t);",
            "        for (int i = h[u]; ~i; i = e[i].ne)",
```

```
            "        {",
            "            int v = e[i].v;",
            "            if (v != pre[u] && v != son[u])",
            "            {",
            "                dfs2(v, v);",
            "            }",
            "        }",
            "    }",
            "",
            "    void mtre(int x, int y, int z)",
            "    {",
            "        while (top[x] != top[y])",
            "        {",
            "            if (deep[top[x]] < deep[top[y]]) ",
            "            {",
            "                swap(x, y);",
            "            }",
            "            modify(1, dfn[top[x]], dfn[x], z); ",
            "            x = pre[top[x]];                    ",
            "        }",
            "        if (deep[x] > deep[y])",
            "        { ",
            "            swap(x, y);",
            "        }",
            "        modify(1, dfn[x], dfn[y], z);",
            "    }",
            "}",
            "using namespace TreeChain;"
        ],
        "description": "树链剖分"
    }
}
```