Template

Billy Wang

2025 年 9 月 5 日

目录

1 写在前面

1.1 基础模版

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define OPFI(x) freopen(#x".in", "r", stdin);\
                freopen(#x".out", "w", stdout)
#define REP(i, a, b) for(int i=(a); i<=(b); ++i)
#define REPd(i, a, b) for(int i=(a); i>=(b); --i)
inline ll rd(){
    ll r=0, k=1; char c;
    while(!isdigit(c=getchar())) if(c=='-') k=-k;
    while(isdigit(c)) r=r*10+c-'0', c=getchar();
    return r*k;
}
int main(){
    return 0;
}
```

1.2 vimrc

```
syntax on
set ts=4 et ai cin sw=4 nu sts=4 sm ru mouse=a title wim=list
" im <F1> <esc>:w<CR>
im <F5> <esc>:bel ter<CR>
" nn <F1> :w<CR>
nn <F5> :bel ter<CR>

im <C-S> <esc>:w<CR>
nn <C-S> :w<CR>
set mp=gnumake
com! Mk sil mak | uns redr! | cw
nn <C-M> :Mk<CR>

"   set shell=powershell
"   set backspace=indent,eol,start
"   set nocompatible
" set sh=powershell bs=indent,eol,start nocp
```

2 数据结构

2.1 zkw 线段树

单点修区间查

```
ll s[N<<2], a[N];
int M;

ll f(ll x, ll y){
    return x+y; // 改这
}

void build(){
    for(M=1; M<=n+1; M<<=1);
    REP(i, 1, n) s[i+M]=a[i];
    REPd(i, M-1, 1) s[i]=f(s[2*i], s[2*i+1]);
}

ll qrange(int l, int r, ll init){ // 根据 f 传 init
    ll res=init;
    for(l=l+M-1, r=r+M+1; l^r^1; l>>=1, r>>=1){
        if(~l&1) res=f(res, s[l^1]);
        if(r&1) res=f(res, s[r^1]);
    }
    return res;
}

void edit(int x, ll v){
    for(s[x+=M]=v, x>>=1; x; x>>=1){
        s[x]=f(s[2*x], s[2*x+1]);
    }
}

ll qpoint(int x){
    return s[x+M];
}
```

2.2 珂朵莉树

```
struct node{
```

```
    int l, r;
    mutable int v;
    bool operator<(const node& rhs) const { return l<rhs.l; }
};

set<node> odt;
typedef set<node>::iterator iter;

iter split(ll p){
    iter tmp=odt.lower_bound((node){p, 0, 0});
    if(tmp!=odt.end()&&tmp->l==p) return tmp;
    --tmp;
    int tl=tmp->l, tr=tmp->r, tv=tmp->v;
    odt.erase(tmp);
    odt.insert((node){tl, p-1, tv});
    return odt.insert((node){p, tr, tv}).first;
}

// 修改和查询注意 split 顺序
// iter itr=split(r+1), itl=split(l);
```

2.3 FHQ-Treap

以模版文艺平衡树为例

```
int n, m, clk, rt;
struct node{
    int key, val, sz, tag, ls, rs;
}t[N];
int newnode(int k){ return t[++clk]=(node){k, rand(), 1, 0}, clk; }
void down(int o){
    if(t[o].tag){
        t[t[o].ls].tag=1-t[t[o].ls].tag;
        t[t[o].rs].tag=1-t[t[o].rs].tag;
        swap(t[t[o].ls].ls, t[t[o].ls].rs);
        swap(t[t[o].rs].ls, t[t[o].rs].rs);
        t[o].tag=0;
    }
}
void up(int o){ t[o].sz=t[t[o].ls].sz+t[t[o].rs].sz+1; }
void split(int o, int x, int &L, int &R){
```

```
    if(o==0) return L=R=0, void(); down(o);
    if(t[t[o].ls].sz+1>=x) R=o, split(t[o].ls, x, L, t[o].ls);
    else L=o, split(t[o].rs, x-t[t[o].ls].sz-1, t[o].rs, R);
    up(o);
}
int merge(int L, int R){
    if(L==0||R==0) return L+R;
    if(t[L].val>t[R].val) return down(L), t[L].rs=merge(t[L].rs, R)
       , up(L), L;
    else return down(R), t[R].ls=merge(L, t[R].ls), up(R), R;
}
```

2.4 并查集

```
ll n, fa[N];
void init(){
    iota(fa+1, fa+n+1, 1);
}

int find(int x){
    if(x==fa[x]) return x;
    return fa[x]=find(fa[x]);
}

void merge(int x, int y){
    x=find(x), y=find(y);
    if(x!=y) fa[x]=y;
}
```

2.5 ST 表

```
ll n, a[N], st[N][SP+10], to[N][SP+10], l2g[N];
ll op(ll x, ll y){ return max(x, y); }

void init(){
    l2g[1]=0, to[n+1][0]=n+1;
    REP(i, 2, n) l2g[i]=l2g[i-1]+!(i&(i-1));
    REP(i, 1, n) st[i][0]=a[i], to[i][0]=i+1;
    REP(i, 1, SP){
        REP(j, 1, n){
```

```
            to[j][i]=to[to[j][i-1]][i-1];
            st[j][i]=op(st[j][i-1], st[to[j][i-1]][i-1]);
        }
    }
}

ll query(ll l, ll r){ // [l, r] 闭区间
    ++r;
    ll d=l2g[r-l];
    return op(st[l][d], st[r-(1<<d)][d]);
}
```

2.6 树状数组

```
ll n, fwt[N];
ll prod(ll x, ll d){ return x+d; }
ll op(ll x, ll y){ return x+y; }

void edit(int x, ll d){
    for(; x<=n; x+=x&-x) fwt[x]=prod(fwt[x], d);
}

ll query(int x){
    assert(1<=x&&x<=n);
    ll res=fwt[x]; x-=x&-x;
        // 这种写法不用考虑最大或最小值的初值问题
    for(; x; x-=x&-x) res=op(res, fwt[x]);
    return res;
}
```

2.7 线段树

3 数学

3.1 快速幂

```
const ll MOD=998244353; // 改模数

ll qpow(ll a, ll x){
    ll res=1;
```

```
    a%=MOD;
    while(x){
        if(x&1) res=res*a%MOD;
        a=a*a%MOD, x>>=1;
    }
    return res;
}

ll inv(ll x){ return qpow(x, MOD-2); } // 模数为质数时
```

3.2 高斯消元

```
const int N=110;
ll n;
double a[N][N], b[N];
void work(){
    n=rd();
    REP(i, 1, n){
        REP(j, 1, n) a[i][j]=rd();
        b[i]=rd();
    }
    REP(i, 1, n){
        int t=i;
        REP(j, i+1, n) if(abs(a[j][i])>1e-7&&(abs(a[t][i])>abs(a[j
            ][i])||abs(a[t][i])<1e-7)) t=j;
        REP(j, i, n) swap(a[t][j], a[i][j]);
        if(abs(a[i][i])<1e-7){
            puts("No Solution");
            return 0;
        }
        swap(b[t], b[i]);
        double e=a[i][i];
        REP(j, i, n) a[i][j]/=e;
        b[i]/=e;
        REP(j, i+1, n){
            double d=a[j][i];
            REP(k, i, n) a[j][k]-=d*a[i][k];
            b[j]-=d*b[i];
        }
    }
```

```
    REPd(i, n, 1) REP(j, 1, i-1) b[j]-=a[j][i]*b[i], a[j][i]=0;
    // REP(i, 1, n) printf("%.2f\n", b[i]);
    // b[1...n] 保存 Ax=b 的解
}
```

3.3 筛法

3.3.1 埃式筛

```
bitset<N> b;
ll n;
vector<ll> prime;
void erato(){
    REP(i, 2, n) if(!b[i]){
        prime.push_back(i);
        for(int j=i+i; j<=n; j+=i)
            b[j]=1;
    }
}
```

3.3.2 线性筛

```
bitset<N> b;
ll n, phi[N];
vector<ll> prime;
void euler(){
    REP(i, 2, n){
        if(!b[i]){
            prime.push_back(i);
            phi[i]=i-1;
        }
        for(int p: prime){
            if(p*i>n) break;
            b[p*i]=1;
            if(i%p==0){
                phi[p*i]=phi[i]*p;
                break;
            }
            phi[p*i]=phi[i]*phi[p];
        }
    }
```

```
}
```

3.4 类欧几里得

3.5 递推组合数

```
ll C[N][N];
void calcC(int n){
    C[0][0]=1;
    REP(i, 1, n){
        C[i][0]=C[i][i]=1;
        REP(j, 1, i-1) C[i][j]=(C[i-1][j]+C[i-1][j-1])%MOD;
    }
}
```

3.6 矩阵快速幂

```
struct mat{
    ll n, m;
    vector<vector<ll>> val; // 注意下标从 0 开始
    mat(ll _n, ll _m): n(_n), m(_m){
        val.resize(_n);
        REP(i, 0, _n-1) val[i].resize(_m, 0);
    }
    mat(ll _n, ll _m, vector<vector<ll>> _val):
        n(_n), m(_m), val(_val){}
};

mat mul(const mat& x, const mat& y, ll mod){
    assert(x.m==y.n);
    mat res(x.n, y.m);
    REP(i, 0, res.n-1) REP(j, 0, res.m-1){
        REP(k, 0, x.m-1){
            res.val[i][j]=(res.val[i][j]+x.val[i][k]*y.val[k][j]%
               mod)%mod;
        }
    }
    return res;
}

```

```
mat qpow(mat a, ll x, ll mod){
    assert(a.n==a.m);
    mat res(a.n, a.n);
    REP(i, 0, a.n-1) res.val[i][i]=1;
    REP(i, 0, a.n-1) REP(j, 0, a.m-1) a.val[i][j]%=mod;
    while(x){
        if(x&1) res=mul(res, a, mod);
        a=mul(a, a, mod), x>>=1;
    }
    return res;
}
```

3.7 扩展欧几里得

```
ll exgcd(ll a, ll b, ll &x, ll &y){
    if(b==0){
        x=1, y=0;
        return a;
    }
    ll xx=0, yy=0;
    ll res=exgcd(b, a%b, xx, yy);
    y=xx-(a/b)*yy, x=yy;
    return res;
}

ll inv(ll x, ll mod){
    ll xx, yy;
    ll d=exgcd(x, mod, xx, yy);
    assert(d==1);
    return (xx%mod+mod)%mod;
}
```

4 图论

4.1 倍增

```
void dfs(int x, int fa){
    pa[x][0]=fa; dep[x]=dep[fa]+1;
    REP(i, 1, SP) pa[x][i]=pa[pa[x][i-1]][i-1];
    for(int& v:g[x]) if(v!=fa){
```

```
        dfs(v, x);
    }
}

int lca(int x, int y){
    if (dep[x]<dep[y]) swap(x, y);
    int t=dep[x]-dep[y];
    REP(i, 0, SP) if(t&(1<<i)) x=pa[x][i];
    REPd(i, SP-1, -1){
        int xx=pa[x][i], yy=pa[y][i];
        if (xx!=yy) x=xx, y=yy;
    }
    return x==y?x:pa[x][0];
}
```

4.2 网络流

不是我写的，但是看着还好

其中 ll 是我改的，不敢保证有没有漏改，但是过了洛谷模版题

4.2.1 最大流

```
constexpr ll INF = LLONG_MAX / 2;

struct E {
    int to; ll cp;
    E(int to, ll cp): to(to), cp(cp) {}
};

struct Dinic {
    static const int M = 1E5 * 5;
    int m, s, t;
    vector<E> edges;
    vector<int> G[M];
    int d[M];
    int cur[M];

    void init(int n, int s, int t) {
        this->s = s; this->t = t;
        for (int i = 0; i <= n; i++) G[i].clear();
        edges.clear(); m = 0;
```

```
20      }
21
22      void addedge(int u, int v, ll cap) {
23          edges.emplace_back(v, cap);
24          edges.emplace_back(u, 0);
25          G[u].push_back(m++);
26          G[v].push_back(m++);
27      }
28
29      bool BFS() {
30          memset(d, 0, sizeof d);
31          queue<int> Q;
32          Q.push(s); d[s] = 1;
33          while (!Q.empty()) {
34              int x = Q.front(); Q.pop();
35              for (int& i: G[x]) {
36                  E &e = edges[i];
37                  if (!d[e.to] && e.cp > 0) {
38                      d[e.to] = d[x] + 1;
39                      Q.push(e.to);
40                  }
41              }
42          }
43          return d[t];
44      }
45
46      ll DFS(int u, ll cp) {
47          if (u == t || !cp) return cp;
48          ll tmp = cp, f;
49          for (int& i = cur[u]; i < G[u].size(); i++) {
50              E& e = edges[G[u][i]];
51              if (d[u] + 1 == d[e.to]) {
52                  f = DFS(e.to, min(cp, e.cp));
53                  e.cp -= f;
54                  edges[G[u][i] ^ 1].cp += f;
55                  cp -= f;
56                  if (!cp) break;
57              }
58          }
59          return tmp - cp;
```

```
    }

    ll go() {
        ll flow = 0;
        while (BFS()) {
            memset(cur, 0, sizeof cur);
            flow += DFS(s, INF);
        }
        return flow;
    }
} DC;
```

4.2.2 费用流

```
constexpr ll INF = LLONG_MAX / 2;

struct E {
    int from, to; ll cp, v;
    E() {}
    E(int f, int t, ll cp, ll v) : from(f), to(t), cp(cp), v(v) {}
};

struct MCMF {
    static const int M = 1E5 * 5;
    int n, m, s, t;
    vector<E> edges;
    vector<int> G[M];
    bool inq[M];
    ll d[M], a[M];
    int p[M];

    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        REP (i, 0, n + 1) G[i].clear();
        edges.clear(); m = 0;
    }

    void addedge(int from, int to, ll cap, ll cost) {
        edges.emplace_back(from, to, cap, cost);
        edges.emplace_back(to, from, 0, -cost);
```

```
27          G[from].push_back(m++);
28          G[to].push_back(m++);
29      }
30
31      bool BellmanFord(ll &flow, ll &cost) {
32          REP (i, 0, n + 1) d[i] = INF;
33          memset(inq, 0, sizeof inq);
34          d[s] = 0, a[s] = INF, inq[s] = true;
35          queue<int> Q; Q.push(s);
36          while (!Q.empty()) {
37              int u = Q.front(); Q.pop();
38              inq[u] = false;
39              for (int& idx: G[u]) {
40                  E &e = edges[idx];
41                  if (e.cp && d[e.to] > d[u] + e.v) {
42                      d[e.to] = d[u] + e.v;
43                      p[e.to] = idx;
44                      a[e.to] = min(a[u], e.cp);
45                      if (!inq[e.to]) {
46                          Q.push(e.to);
47                          inq[e.to] = true;
48                      }
49                  }
50              }
51          }
52          if (d[t] == INF) return false;
53          flow += a[t];
54          cost += a[t] * d[t];
55          int u = t;
56          while (u != s) {
57              edges[p[u]].cp -= a[t];
58              edges[p[u] ^ 1].cp += a[t];
59              u = edges[p[u]].from;
60          }
61          return true;
62      }
63
64      pair<ll, ll> go() {
65          ll flow = 0, cost = 0;
66          while (BellmanFord(flow, cost));
```

```
        return make_pair(flow, cost);
    }
} MM;
```

4.3 二分图最大匹配

ps. 建单向图（即只有左部指向右部的边）

```
struct MaxMatch {
    int n;
    vector<int> G[N];
    int vis[N], left[N], clk;

    void init(int n) {
        this->n = n;
        REP (i, 0, n + 1) G[i].clear();
        memset(left, -1, sizeof left);
        memset(vis, -1, sizeof vis);
    }

    bool dfs(int u) {
        for (int v: G[u])
            if (vis[v] != clk) {
                vis[v] = clk;
                if (left[v] == -1 || dfs(left[v])) {
                    left[v] = u;
                    return true;
                }
            }
        return false;
    }

    int match() {
        int ret = 0;
        for (clk = 0; clk <= n; ++clk)
            if (dfs(clk)) ++ret;
        return ret;
    }
} MM;
```

4.4 Tarjan 强连通分量缩点

```
int low[N], dfn[N], clk, B, bl[N];
vector<int> bcc[N];
void init() { B = clk = 0; memset(dfn, 0, sizeof dfn); }
void tarjan(int u) {
    static int st[N], p;
    static bool in[N];
    dfn[u] = low[u] = ++clk;
    st[p++] = u; in[u] = true;
    for (int& v: G[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (in[v]) low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
        ++B;
        while (1) {
            int x = st[--p]; in[x] = false;
            bl[x] = B; bcc[B].push_back(x);
            if (x == u) break;
        }
    }
}
```

4.5 树直径

```
ll n, dep[N], mxdep[N];
vector<int> g[N];
vector<int> dmt;
void dfs1(int x, int fa){
    dep[x]=dep[fa]+1;
    mxdep[x]=1;
    for(int u: g[x]) if(u!=fa){
        dfs1(u, x);
        mxdep[x]=max(mxdep[u]+1, mxdep[x]);
    }
}

```

```
void dfs2(int x, int fa){ // 找一条直径，如果只需要直径长度则不用
    dmt.push_back(x);
    int nxt=-1;
    for(int u: g[x]) if(u!=fa){
        if(nxt==-1||mxdep[u]>mxdep[nxt]) nxt=u;
    }
    if(nxt!=-1)
        dfs2(nxt, x);
}

void diameter(){
    dep[0]=0;
    dfs1(1, 0);
    int rt=max_element(dep+1, dep+n+1)-dep;
    dfs1(rt, 0);
    dfs2(rt, 0);
}
```

4.6 树重心

```
ll n, sz[N], mxsz[N], G;
vector<int> g[N];
void dfs(int x, int fa){
    sz[x]=1, mxsz[x]=0;
    for(int u: g[x]) if(u!=fa){
        dfs(u, x);
        sz[x]+=sz[u];
        mxsz[x]=max(sz[u], mxsz[x]);
    }
    mxsz[x]=max(mxsz[x], n-sz[x]);
    if(G==-1||(mxsz[x]<mxsz[G]||(mxsz[x]==mxsz[G]&&x<G)))
        G=x;
}
```

4.7 树链剖分

4.8 最短路

4.8.1 Floyd（最小环）

```
ll n, m, mincycle;
```

```
ll g[N][N], dis[N][N];
void floyd(){
    // 如果 g[i][j] 之间没边则存 inf
    // 注意 inf 的三倍不能爆 long long
    mincycle=LLONG_MAX/4;
    REP(i, 1, n) REP(j, 1, n) dis[i][j]=g[i][j];
    REP(k, 1, n){
        REP(i, 1, k-1)
            REP(j, i+1, k-1){
                mincycle=min(mincycle, dis[i][j]+g[j][k]+g[k][i]);
            }
        REP(i, 1, n)
            REP(j, 1, n)
                dis[i][j]=min(dis[i][j], dis[i][k]+dis[k][j]);
    }
}
```

4.8.2 Spfa（判负环）

```
ll n, m, dis[N], inq[N], cnt[N];
vector<pair<int, ll>> g[N];
bool spfa(int s){
    // 如果有负环则 return true
    queue<int> q;
    fill_n(dis+1, n, LLONG_MAX/2);
    fill_n(inq+1, n, 0);
    fill_n(cnt+1, n, 0);
    dis[s]=0, inq[s]=cnt[s]=1;
    q.push(s);
    while(!q.empty()){
        int u=q.front(); q.pop();
        inq[u]=0;
        for(auto [v, w]: g[u]){
            if(dis[v]>dis[u]+w){
                dis[v]=dis[u]+w;
                cnt[v]=cnt[u]+1;
                if(cnt[v]>n) return true;
                if(!inq[v]){
                    inq[v]=1, q.push(v);
                }
```

```
            }
        }
    }
    return false;
}
```

4.8.3 Dijkstra

```
ll n, m, dis[N], vis[N];
vector<pair<int, ll>> g[N];
void dijkstra(int s){
    fill_n(dis+1, n, LLONG_MAX/2);
    fill_n(vis+1, n, 0);
    dis[s]=0;
    priority_queue<pair<ll, int>> q;
    q.push(make_pair(0, s));
    while(!q.empty()){
        int u=q.top().second; q.pop();
        if(vis[u]) continue;
        vis[u]=1;
        for(auto [v, w]: g[u]){
            if(vis[v]) continue;
            if(dis[v]>dis[u]+w){
                dis[v]=dis[u]+w;
                q.push(make_pair(-dis[v], v));
            }
        }
    }
}
```

4.9 拓扑排序

```
ll n, in[N];
vector<int> g[N];
vector<int> res;
void topo(){
    // 不要忘记在建图时记录入度
    queue<int> q;
    REP(i, 1, n)
        if(in[i]==0) q.push(i);
```

```
    while(!q.empty()){
        int u=q.front(); q.pop();
        res.push_back(u);
        for(int v: g[u]){
            --in[v];
            if(in[v]==0) q.push(v);
        }
    }
}
```

4.10 最小生成树

Kruskal 算法

```
ll n, m, fa[N], weight_sum;
pair<ll, pair<int, int>> edge[M];
int find(int x){
    if(fa[x]==x) return x;
    return fa[x]=find(fa[x]);
}

void kruskal(){
    weight_sum=0;
    sort(edge+1, edge+m+1);
    iota(fa+1, fa+n+1, 1);
    REP(i, 1, m){
        auto [u, v]=edge[i].second;
        u=find(u), v=find(v);
        if(u==v) continue;
        fa[u]=v;
        weight_sum+=edge[i].first;
    }
}
```

4.11 欧拉路径/回路

```
ll n, m, cnt[N];
vector<int> g[N];
vector<int> path;
void dfs(int x){
    while(cnt[x]<g[x].size()){
```

```
6           int u=g[x][cnt[x]];
7           ++cnt[x];
8           dfs(u);
9       }
10      path.push_back(x);
11      // 最后要 reverse(path.begin(), path.end());
12  }
```

5 字符串

5.1 KMP

```
1   char s1[N];
2   int n, pi[N];
3   void solve(){
4       // 字符串存于 s1[0, n)
5       pi[0]=0;
6       REP(i, 1, n-1){
7           int j=pi[i-1];
8           while(j>0&&s1[j]!=s1[i]) j=pi[j-1];
9           if(s1[j]==s1[i]) ++j;
10          pi[i]=j;
11      }
12  }
```

5.2 Trie 树

6 STL

6.1 算法库

不修改序列的操作

批量操作

在标头 <algorithm> 定义

`for_each`

应用一元函数对象到范围中元素 (函数模板)

`ranges::for_each` (C++20)

应用一元函数对象到范围中元素 (算法函数对象)

`for_each_n` (C++17)

应用函数对象到序列的前 N 个元素 (函数模板)

`ranges::for_each_n` (C++20)

应用函数对象到序列的前 N 个元素 (算法函数对象)

搜索操作

在标头 <algorithm> 定义

`all_of` (C++11)

`any_of` (C++11)

`none_of` (C++11)

检查谓词是否对范围中所有、任一或无元素为 `true` (函数模板)

`ranges::all_of` (C++20)

`ranges::any_of` (C++20)

`ranges::none_of` (C++20)

检查谓词是否对范围中所有、任一或无元素为 `true` (算法函数对象)

`ranges::contains` (C++23)

`ranges::contains_subrange` (C++23)

检查范围是否包含给定元素或子范围 (算法函数对象)

`find`

`find_if`

`find_if_not` (C++11)

查找首个满足特定条件的元素 (函数模板)

`ranges::find` (C++20)

`ranges::find_if` (C++20)

`ranges::find_if_not` (C++20)

查找首个满足特定条件的元素 (算法函数对象)

`ranges::find_last` (C++23)

`ranges::find_last_if` (C++23)

`ranges::find_last_if_not` (C++23)

查找最后一个满足特定条件的元素 (算法函数对象)

`find_end`

查找元素序列在特定范围中最后一次出现 (函数模板)

`ranges::find_end` (C++20)

查找元素序列在特定范围中最后一次出现 (算法函数对象)

`find_first_of`

搜索一组元素中任一元素 (函数模板)

`ranges::find_first_of` (C++20)

搜索一组元素中任一元素 (算法函数对象)

`adjacent_find`

查找首对相同（或满足给定谓词）的相邻元素 (函数模板)

`ranges::adjacent_find` (C++20)

查找首对相同（或满足给定谓词）的相邻元素 (算法函数对象)

`count`

`count_if`
返回满足特定条件的元素数目 (函数模板)
`ranges::count` (C++20)
`ranges::count_if` (C++20)
返回满足特定条件的元素数目 (算法函数对象)
`mismatch`
查找两个范围的首个不同之处 (函数模板)
`ranges::mismatch` (C++20)
查找两个范围的首个不同之处 (算法函数对象)
`equal`
判断两组元素是否相同 (函数模板)
`ranges::equal` (C++20)
判断两组元素是否相同 (算法函数对象)
`search`
搜索元素范围的首次出现 (函数模板)
`ranges::search` (C++20)
搜索元素范围的首次出现 (算法函数对象)
`search_n`
搜索元素在范围中首次连续若干次出现 (函数模板)
`ranges::search_n` (C++20)
搜索元素在范围中首次连续若干次出现 (算法函数对象)
`ranges::starts_with` (C++23)
检查一个范围是否始于另一范围 (算法函数对象)
`ranges::ends_with` (C++23)
检查一个范围是否终于另一范围 (算法函数对象)
折叠操作 (C++23 起)
在标头 <algorithm> 定义
`ranges::fold_left` (C++23)
左折叠范围中元素 (算法函数对象)
`ranges::fold_left_first` (C++23)
以首元素为初值左折叠范围中元素 (算法函数对象)
`ranges::fold_right` (C++23)
右折叠范围中元素 (算法函数对象)
`ranges::fold_right_last` (C++23)
以末元素为初值右折叠范围中元素 (算法函数对象)
`ranges::fold_left_with_iter` (C++23)
左折叠范围中元素，并返回 `pair`（迭代器，值）(算法函数对象)
`ranges::fold_left_first_with_iter` (C++23)
以首元素为初值左折叠范围中元素，并返回 `pair`（迭代器，optional）(算法函数对象)
修改序列的操作

复制操作

在标头 <algorithm> 定义

`copy`

`copy_if` (C++11)

复制范围中元素到新位置 (函数模板)

`ranges::copy` (C++20)

`ranges::copy_if` (C++20)

复制范围中元素到新位置 (算法函数对象)

`copy_n` (C++11)

复制若干元素到新位置 (函数模板)

`ranges::copy_n` (C++20)

复制若干元素到新位置 (算法函数对象)

`copy_backward`

从后往前复制范围中元素 (函数模板)

`ranges::copy_backward` (C++20)

从后往前复制范围中元素 (算法函数对象)

`move` (C++11)

将范围中元素移到新位置 (函数模板)

`ranges::move` (C++20)

将范围中元素移到新位置 (算法函数对象)

`move_backward` (C++11)

从后往前将范围中元素移到新位置 (函数模板)

`ranges::move_backward` (C++20)

从后往前将范围中元素移到新位置 (算法函数对象)

交换操作

在标头 <algorithm> 定义 (C++11 前)

在标头 <utility> 定义 (C++11 起)

在标头 <string_view> 定义

`swap`

在标头 <algorithm> 定义

交换两个对象的值 (函数模板)

`swap_ranges`

交换两个范围的元素 (函数模板)

`ranges::swap_ranges` (C++20)

交换两个范围的元素 (算法函数对象)

`iter_swap`

交换两个迭代器所指向的元素 (函数模板)

变换操作

在标头 <algorithm> 定义

`transform`

应用函数到元素范围，并在目标范围存储结果 (函数模板)

`ranges::transform` (C++20)

应用函数到元素范围 (算法函数对象)

`replace`

`replace_if`

替换所有满足特定条件的值为另一个值 (函数模板)

`ranges::replace` (C++20)

`ranges::replace_if` (C++20)

替换所有满足特定条件的值为另一个值 (算法函数对象)

`replace_copy`

`replace_copy_if`

复制范围，并将满足特定条件的元素替换为另一个值 (函数模板)

`ranges::replace_copy` (C++20)

`ranges::replace_copy_if` (C++20)

复制范围，并将满足特定条件的元素替换为另一个值 (算法函数对象)

生成操作

在标头 <algorithm> 定义

`fill`

以复制的方式赋给定值到范围中所有元素 (函数模板)

`ranges::fill` (C++20)

赋给定值到范围中元素 (算法函数对象)

`fill_n`

以复制的方式赋给定值到范围中 N 个元素 (函数模板)

`ranges::fill_n` (C++20)

赋给定值到若干元素 (算法函数对象)

`generate`

赋连续函数调用结果到范围中所有元素 (函数模板)

`ranges::generate` (C++20)

将函数结果保存到范围中 (算法函数对象)

`generate_n`

赋连续函数调用结果到范围中 N 个元素 (函数模板)

`ranges::generate_n` (C++20)

保存 N 次函数应用的结果 (算法函数对象)

移除操作

在标头 <algorithm> 定义

`remove`

`remove_if`

移除满足特定条件的元素 (函数模板)

`ranges::remove` (C++20)

`ranges::remove_if` (C++20)

移除满足特定条件的元素 (算法函数对象)

`remove_copy`

`remove_copy_if`

复制范围并忽略满足特定条件的元素 (函数模板)

`ranges::remove_copy` (C++20)

`ranges::remove_copy_if` (C++20)

复制范围并忽略满足特定条件的元素 (算法函数对象)

`unique`

移除范围中连续重复元素 (函数模板)

`ranges::unique` (C++20)

移除范围中连续重复元素 (算法函数对象)

`unique_copy`

创建某范围的不含连续重复元素的副本 (函数模板)

`ranges::unique_copy` (C++20)

创建某范围的不含连续重复元素的副本 (算法函数对象)

顺序变更操作

在标头 <algorithm> 定义

`reverse`

逆转范围中的元素顺序 (函数模板)

`ranges::reverse` (C++20)

逆转范围中的元素顺序 (算法函数对象)

`reverse_copy`

创建范围的逆向副本 (函数模板)

`ranges::reverse_copy` (C++20)

创建范围的逆向副本 (算法函数对象)

`rotate`

旋转范围中的元素顺序 (函数模板)

`ranges::rotate` (C++20)

旋转范围中的元素顺序 (算法函数对象)

`rotate_copy`

复制并旋转元素范围 (函数模板)

`ranges::rotate_copy` (C++20)

复制并旋转元素范围 (算法函数对象)

`shift_left` (C++20)

`shift_right` (C++20)

迁移范围中元素 (函数模板)

`ranges::shift_left` (C++23)

`ranges::shift_right` (C++23)

迁移范围中元素 (算法函数对象)

`random_shuffle` (C++17 前)

`shuffle` (C++11)

随机重排范围中元素 (函数模板)

`ranges::shuffle` (C++20)

随机重排范围中元素 (算法函数对象)

采样操作

在标头 <algorithm> 定义

`sample` (C++17)

从序列中随机选择 N 个元素 (函数模板)

`ranges::sample` (C++20)

从序列中随机选择 N 个元素 (算法函数对象)

排序和相关操作

要求

部分算法要求由实参表示的序列"已排序"或"已划分"。未满足要求时行为未定义。

序列 `[start, finish)` 在满足以下条件时已按表达式 `f(e)` 划分：存在一个整数 `n`，使得对于 `[0, std::distance(start, finish))` 中的所有整数 `i`，`f(*(start + i))` 当且仅当 `i < n` 时是 `true`。

划分操作

在标头 <algorithm> 定义

`is_partitioned` (C++11)

判断范围是否已按给定谓词划分 (函数模板)

`ranges::is_partitioned` (C++20)

判断范围是否已按给定谓词划分 (算法函数对象)

`partition`

将范围中元素分为两组 (函数模板)

`ranges::partition` (C++20)

将范围中元素分为两组 (算法函数对象)

`partition_copy` (C++11)

复制范围并将元素分为两组 (函数模板)

`ranges::partition_copy` (C++20)

复制范围并将元素分为两组 (算法函数对象)

`stable_partition`

将元素分为两组，同时保留其相对顺序 (函数模板)

`ranges::stable_partition` (C++20)

将元素分为两组，同时保留其相对顺序 (算法函数对象)

`partition_point` (C++11)

定位已划分范围的划分点 (函数模板)

`ranges::partition_point` (C++20)

定位已划分范围的划分点 (算法函数对象)

排序操作

在标头 <algorithm> 定义

`sort`
将范围按升序排序 (函数模板)
`ranges::sort` (C++20)
将范围按升序排序 (算法函数对象)
`stable_sort`
将范围中元素排序，同时保持相等元之间的顺序 (函数模板)
`ranges::stable_sort` (C++20)
将范围中元素排序，同时保持相等元之间的顺序 (算法函数对象)
`partial_sort`
将范围中前 N 个元素排序 (函数模板)
`ranges::partial_sort` (C++20)
将范围中前 N 个元素排序 (算法函数对象)
`partial_sort_copy`
复制范围中元素并部分排序 (函数模板)
`ranges::partial_sort_copy` (C++20)
复制范围中元素并部分排序 (算法函数对象)
`is_sorted` (C++11)
检查范围是否已按升序排列 (函数模板)
`ranges::is_sorted` (C++20)
检查范围是否已按升序排列 (算法函数对象)
`is_sorted_until` (C++11)
找出最大的有序子范围 (函数模板)
`ranges::is_sorted_until` (C++20)
找出最大的有序子范围 (算法函数对象)
`nth_element`
将给定范围部分排序，确保其按给定元素划分 (函数模板)
`ranges::nth_element` (C++20)
将给定范围部分排序，确保其按给定元素划分 (算法函数对象)

二分搜索操作（在已划分范围上）

在标头 <algorithm> 定义

`lower_bound`
返回首个不小于给定值的元素的迭代器 (函数模板)
`ranges::lower_bound` (C++20)
返回首个不小于给定值的元素的迭代器 (算法函数对象)
`upper_bound`
返回首个大于给定值的元素的迭代器 (函数模板)
`ranges::upper_bound` (C++20)
返回首个大于给定值的元素的迭代器 (算法函数对象)
`equal_range`
返回匹配特定键值的元素范围 (函数模板)

`ranges::equal_range` (C++20)
返回匹配特定键值的元素范围 (算法函数对象)
`binary_search`
判断元素是否在偏序范围中 (函数模板)
`ranges::binary_search` (C++20)
判断元素是否在偏序范围中 (算法函数对象)
集合操作（在已排序范围上）
在标头 <algorithm> 定义
`includes`
当一个序列是另一个的子序列时返回 `true` (函数模板)
`ranges::includes` (C++20)
当一个序列是另一个的子序列时返回 `true` (算法函数对象)
`set_union`
计算两个集合的并集 (函数模板)
`ranges::set_union` (C++20)
计算两个集合的并集 (算法函数对象)
`set_intersection`
计算两个集合的交集 (函数模板)
`ranges::set_intersection` (C++20)
计算两个集合的交集 (算法函数对象)
`set_difference`
计算两个集合的差集 (函数模板)
`ranges::set_difference` (C++20)
计算两个集合的差集 (算法函数对象)
`set_symmetric_difference`
计算两个集合的对称差 (函数模板)
`ranges::set_symmetric_difference` (C++20)
计算两个集合的对称差 (算法函数对象)
归并操作（在已排序范围上）
在标头 <algorithm> 定义
`merge`
合并两个有序范围 (函数模板)
`ranges::merge` (C++20)
合并两个有序范围 (算法函数对象)
`inplace_merge`
就地合并两个有序范围 (函数模板)
`ranges::inplace_merge` (C++20)
就地合并两个有序范围 (算法函数对象)
堆操作

(C++20 前) 随机访问范围 `[first, last)` 在满足以下条件时是一个关于比较器 `comp` 的堆：对于 `(0, last - first)` 中的所有整数 `i`，`bool(comp(first[(i - 1)/ 2], first[i]))` 都是 `false`。

在标头 <algorithm> 定义

`push_heap`

添加元素到最大堆 (函数模板)

`ranges::push_heap` (C++20)

添加元素到最大堆 (算法函数对象)

`pop_heap`

移除最大堆中最大元 (函数模板)

`ranges::pop_heap` (C++20)

移除最大堆中最大元 (算法函数对象)

`make_heap`

从元素范围创建最大堆 (函数模板)

`ranges::make_heap` (C++20)

从元素范围创建最大堆 (算法函数对象)

`sort_heap`

将最大堆变成按升序排序的元素范围 (函数模板)

`ranges::sort_heap` (C++20)

将最大堆变成按升序排序的元素范围 (算法函数对象)

`is_heap`

检查给定范围是否为最大堆 (函数模板)

`ranges::is_heap` (C++20)

检查给定范围是否为最大堆 (算法函数对象)

`is_heap_until` (C++11)

查找能成为最大堆的最大子范围 (函数模板)

`ranges::is_heap_until` (C++20)

查找能成为最大堆的最大子范围 (算法函数对象)

最小/最大操作

在标头 <algorithm> 定义

`max`

返回给定值中较大者 (函数模板)

`ranges::max` (C++20)

返回给定值中较大者 (算法函数对象)

`max_element`

返回范围中最大元 (函数模板)

`ranges::max_element` (C++20)

返回范围中最大元 (算法函数对象)

`min`

返回给定值中较小者 (函数模板)

`ranges::min` (C++20)
返回给定值中较小者 (算法函数对象)
`min_element`
返回范围中最小元 (函数模板)
`ranges::min_element` (C++20)
返回范围中最小元 (算法函数对象)
`minmax` (C++11)
返回两个元素间的较小者和较大者 (函数模板)
`ranges::minmax` (C++20)
返回两个元素间的较小者和较大者 (算法函数对象)
`minmax_element` (C++11)
返回范围中的最小元和最大元 (函数模板)
`ranges::minmax_element` (C++20)
返回范围中的最小元和最大元 (算法函数对象)
`clamp` (C++17)
在一对边界值下夹逼一个值 (函数模板)
`ranges::clamp` (C++20)
在一对边界值下夹逼一个值 (算法函数对象)

字典序比较操作

在标头 <algorithm> 定义

`lexicographical_compare`
当一个范围字典序小于另一个时返回 `true` (函数模板)
`ranges::lexicographical_compare` (C++20)
当一个范围字典序小于另一个时返回 `true` (算法函数对象)
`lexicographical_compare_three_way` (C++20)
三路比较两个范围 (函数模板)

排列操作

在标头 <algorithm> 定义

`next_permutation`
生成元素范围的下一个字典序更大的排列 (函数模板)
`ranges::next_permutation` (C++20)
生成元素范围的下一个字典序更大的排列 (算法函数对象)
`prev_permutation`
生成元素范围的下一个字典序更小的排列 (函数模板)
`ranges::prev_permutation` (C++20)
生成元素范围的下一个字典序更小的排列 (算法函数对象)
`is_permutation` (C++11)
判断一个序列是否为另一个序列的排列 (函数模板)
`ranges::is_permutation` (C++20)
判断一个序列是否为另一个序列的排列 (算法函数对象)

数值运算

在标头 <numeric> 定义

`iota` (C++11)

从初始值开始连续递增填充范围 (函数模板)

`ranges::iota` (C++23)

从初始值开始连续递增填充范围 (算法函数对象)

`accumulate`

求和或折叠范围中元素 (函数模板)

`inner_product`

计算两个范围中元素的内积 (函数模板)

`adjacent_difference`

计算范围中相邻元素的差 (函数模板)

`partial_sum`

计算范围中元素的部分和 (函数模板)

`reduce` (C++17)

类似 `std::accumulate`，但不依序执行 (函数模板)

`exclusive_scan` (C++17)

类似 `std::partial_sum`，第 `i` 个和中排除第 `i` 个输入 (函数模板)

`inclusive_scan` (C++17)

类似 `std::partial_sum`，第 `i` 个和中包含第 `i` 个输入 (函数模板)

`transform_reduce` (C++17)

应用可调用对象，然后乱序规约 (函数模板)

`transform_exclusive_scan` (C++17)

应用可调用对象，然后计算排除扫描 (函数模板)

`transform_inclusive_scan` (C++17)

应用可调用对象，然后计算包含扫描 (函数模板)