

# Template

Billy Wang

2025 年 9 月 5 日

## 目录

|          |                     |           |
|----------|---------------------|-----------|
| <b>1</b> | <b>写在前面</b>         | <b>3</b>  |
| 1.1      | 基础模版 . . . . .      | 3         |
| 1.2      | vimrc . . . . .     | 3         |
| <b>2</b> | <b>数据结构</b>         | <b>4</b>  |
| 2.1      | zkw 线段树 . . . . .   | 4         |
| 2.2      | 珂朵莉树 . . . . .      | 4         |
| 2.3      | FHQ-Treap . . . . . | 5         |
| 2.4      | 并查集 . . . . .       | 6         |
| 2.5      | ST 表 . . . . .      | 6         |
| 2.6      | 树状数组 . . . . .      | 7         |
| 2.7      | 线段树 . . . . .       | 7         |
| <b>3</b> | <b>数学</b>           | <b>7</b>  |
| 3.1      | 快速幂 . . . . .       | 7         |
| 3.2      | 高斯消元 . . . . .      | 8         |
| 3.3      | 筛法 . . . . .        | 9         |
| 3.3.1    | 埃式筛 . . . . .       | 9         |
| 3.3.2    | 线性筛 . . . . .       | 9         |
| 3.4      | 类欧几里得 . . . . .     | 10        |
| 3.5      | 递推组合数 . . . . .     | 10        |
| 3.6      | 矩阵快速幂 . . . . .     | 10        |
| 3.7      | 扩展欧几里得 . . . . .    | 11        |
| <b>4</b> | <b>图论</b>           | <b>11</b> |
| 4.1      | 倍增 . . . . .        | 11        |
| 4.2      | 网络流 . . . . .       | 12        |
| 4.2.1    | 最大流 . . . . .       | 12        |
| 4.2.2    | 费用流 . . . . .       | 14        |
| 4.3      | 二分图最大匹配 . . . . .   | 16        |

|          |                          |           |
|----------|--------------------------|-----------|
| 4.4      | Tarjan 强连通分量缩点 . . . . . | 17        |
| 4.5      | 树直径 . . . . .            | 17        |
| 4.6      | 树重心 . . . . .            | 18        |
| 4.7      | 树链剖分 . . . . .           | 18        |
| 4.8      | 最短路 . . . . .            | 18        |
| 4.8.1    | Floyd (最小环) . . . . .    | 18        |
| 4.8.2    | Spfa (判负环) . . . . .     | 19        |
| 4.8.3    | Dijkstra . . . . .       | 19        |
| 4.9      | 拓扑排序 . . . . .           | 19        |
| 4.10     | 最小生成树 . . . . .          | 19        |
| 4.11     | 欧拉路径/回路 . . . . .        | 19        |
| 4.12     | 01 图黑白染色 . . . . .       | 19        |
| <b>5</b> | <b>字符串</b>               | <b>19</b> |
| 5.1      | KMP . . . . .            | 19        |
| 5.2      | Trie 树 . . . . .         | 19        |
| <b>6</b> | <b>STL</b>               | <b>19</b> |
| 6.1      | 算法库 . . . . .            | 19        |

# 1 写在前面

## 1.1 基础模版

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define OPFI(x) freopen(#x".in", "r", stdin);\
5                 freopen(#x".out", "w", stdout)
6 #define REP(i, a, b) for(int i=(a); i<=(b); ++i)
7 #define REPd(i, a, b) for(int i=(a); i>=(b); --i)
8 inline ll rd(){
9     ll r=0, k=1; char c;
10    while(!isdigit(c=getchar())) if(c=='-') k=-k;
11    while(isdigit(c)) r=r*10+c-'0', c=getchar();
12    return r*k;
13 }
14 int main(){
15     return 0;
16 }
```

## 1.2 vimrc

```
1 syntax on
2 set ts=4 et ai cin sw=4 nu sts=4 sm ru mouse=a title wim=list
3 " im <F1> <esc>:w<CR>
4 im <F5> <esc>:bel ter<CR>
5 " nn <F1> :w<CR>
6 nn <F5> :bel ter<CR>
7
8 im <C-S> <esc>:w<CR>
9 nn <C-S> :w<CR>
10 set mp=gnumake
11 com! Mk sil mak | uns redr! | cw
12 nn <C-M> :Mk<CR>
13
14 " set shell=powershell
15 " set backspace=indent,eol,start
16 " set nocompatible
17 " 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

```
1 ll s[N<<2], a[N];
2 int M;
3
4 ll f(ll x, ll y){
5     return x+y; // 改这
6 }
7
8 void build(){
9     for(M=1; M<=n+1; M<<=1);
10    REP(i, 1, n) s[i+M]=a[i];
11    REPd(i, M-1, 1) s[i]=f(s[2*i], s[2*i+1]);
12 }
13
14 ll qrange(int l, int r, ll init){ // 根据 f 传 init
15     ll res=init;
16     for(l=l+M-1, r=r+M+1; l^r^1; l>>=1, r>>=1){
17         if(~l&1) res=f(res, s[l^1]);
18         if(r&1) res=f(res, s[r^1]);
19     }
20     return res;
21 }
22
23 void edit(int x, ll v){
24     for(s[x+=M]=v, x>>=1; x; x>>=1){
25         s[x]=f(s[2*x], s[2*x+1]);
26     }
27 }
28
29 ll qpoint(int x){
30     return s[x+M];
31 }
```

### 2.2 珂朵莉树

```
1 struct node{
```

```

2     int l, r;
3     mutable int v;
4     bool operator<(const node& rhs) const { return l<rhs.l; }
5 };
6
7 set<node> odt;
8 typedef set<node>::iterator iter;
9
10 iter split(ll p){
11     iter tmp=odt.lower_bound((node){p, 0, 0});
12     if(tmp!=odt.end()&&tmp->l==p) return tmp;
13     --tmp;
14     int tl=tmp->l, tr=tmp->r, tv=tmp->v;
15     odt.erase(tmp);
16     odt.insert((node){tl, p-1, tv});
17     return odt.insert((node){p, tr, tv}).first;
18 }
19
20 // 修改和查询注意 split 顺序
21 // iter itr=split(r+1), itl=split(l);

```

## 2.3 FHQ-Treap

以模版文艺平衡树为例

```

1 int n, m, clk, rt;
2 struct node{
3     int key, val, sz, tag, ls, rs;
4 }t[N];
5 int newnode(int k){ return t[++clk]=(node){k, rand(), 1, 0}, clk; }
6 void down(int o){
7     if(t[o].tag){
8         t[t[o].ls].tag=1-t[t[o].ls].tag;
9         t[t[o].rs].tag=1-t[t[o].rs].tag;
10        swap(t[t[o].ls].ls, t[t[o].ls].rs);
11        swap(t[t[o].rs].ls, t[t[o].rs].rs);
12        t[o].tag=0;
13    }
14 }
15 void up(int o){ t[o].sz=t[t[o].ls].sz+t[t[o].rs].sz+1; }
16 void split(int o, int x, int &L, int &R){

```

```

17     if(o==0) return L=R=0, void(); down(o);
18     if(t[t[o].ls].sz+1>=x) R=o, split(t[o].ls, x, L, t[o].ls);
19     else L=o, split(t[o].rs, x-t[t[o].ls].sz-1, t[o].rs, R);
20     up(o);
21 }
22 int merge(int L, int R){
23     if(L==0||R==0) return L+R;
24     if(t[L].val>t[R].val) return down(L), t[L].rs=merge(t[L].rs, R)
        , up(L), L;
25     else return down(R), t[R].ls=merge(L, t[R].ls), up(R), R;
26 }

```

## 2.4 并查集

```

1 ll n, fa[N];
2 void init(){
3     iota(fa+1, fa+n+1, 1);
4 }
5
6 int find(int x){
7     if(x==fa[x]) return x;
8     return fa[x]=find(fa[x]);
9 }
10
11 void merge(int x, int y){
12     x=find(x), y=find(y);
13     if(x!=y) fa[x]=y;
14 }

```

## 2.5 ST 表

```

1 ll n, a[N], st[N][SP+10], to[N][SP+10], l2g[N];
2 ll op(ll x, ll y){ return max(x, y); }
3
4 void init(){
5     l2g[1]=0, to[n+1][0]=n+1;
6     REP(i, 2, n) l2g[i]=l2g[i-1]+!(i&(i-1));
7     REP(i, 1, n) st[i][0]=a[i], to[i][0]=i+1;
8     REP(i, 1, SP){
9         REP(j, 1, n){

```

```

10         to[j][i]=to[to[j][i-1]][i-1];
11         st[j][i]=op(st[j][i-1], st[to[j][i-1]][i-1]);
12     }
13 }
14 }
15
16 ll query(ll l, ll r){ // [l, r] 闭区间
17     ++r;
18     ll d=l2g[r-1];
19     return op(st[l][d], st[r-(1<<d)][d]);
20 }

```

## 2.6 树状数组

```

1 ll n, fwt[N];
2 ll prod(ll x, ll d){ return x+d; }
3 ll op(ll x, ll y){ return x+y; }
4
5 void edit(int x, ll d){
6     for(; x<=n; x+=x&-x) fwt[x]=prod(fwt[x], d);
7 }
8
9 ll query(int x){
10     assert(1<=x&&x<=n);
11     ll res=fwt[x]; x-=x&-x;
12     // 这种写法不用考虑最大或最小值的初值问题
13     for(; x; x-=x&-x) res=op(res, fwt[x]);
14     return res;
15 }

```

## 2.7 线段树

# 3 数学

## 3.1 快速幂

```

1 const ll MOD=998244353; // 改模数
2
3 ll qpow(ll a, ll x){
4     ll res=1;

```

```

5     a%=MOD;
6     while(x){
7         if(x&1) res=res*a%MOD;
8         a=a*a%MOD, x>>=1;
9     }
10    return res;
11 }
12
13 ll inv(ll x){ return qpow(x, MOD-2); } // 模数为质数时

```

### 3.2 高斯消元

```

1  const int N=110;
2  ll n;
3  double a[N][N], b[N];
4  void work(){
5      n=rd();
6      REP(i, 1, n){
7          REP(j, 1, n) a[i][j]=rd();
8          b[i]=rd();
9      }
10     REP(i, 1, n){
11         int t=i;
12         REP(j, i+1, n) if(abs(a[j][i])>1e-7&&(abs(a[t][i])>abs(a[j]
13             ][i])||abs(a[t][i])<1e-7)) t=j;
14         REP(j, i, n) swap(a[t][j], a[i][j]);
15         if(abs(a[i][i])<1e-7){
16             puts("No Solution");
17             return 0;
18         }
19         swap(b[t], b[i]);
20         double e=a[i][i];
21         REP(j, i, n) a[i][j]/=e;
22         b[i]/=e;
23         REP(j, i+1, n){
24             double d=a[j][i];
25             REP(k, i, n) a[j][k]-=d*a[i][k];
26             b[j]-=d*b[i];
27         }
28     }
29 }

```



```

28     REPd(i, n, 1) REP(j, 1, i-1) b[j]-=a[j][i]*b[i], a[j][i]=0;
29     // REP(i, 1, n) printf("%.2f\n", b[i]);
30     // b[1...n] 保存  $Ax=b$  的解
31 }

```

### 3.3 筛法

#### 3.3.1 埃式筛

```

1  bitset<N> b;
2  ll n;
3  vector<ll> prime;
4  void erato(){
5      REP(i, 2, n) if(!b[i]){
6          prime.push_back(i);
7          for(int j=i+i; j<=n; j+=i)
8              b[j]=1;
9      }
10 }

```

#### 3.3.2 线性筛

```

1  bitset<N> b;
2  ll n, phi[N];
3  vector<ll> prime;
4  void euler(){
5      REP(i, 2, n){
6          if(!b[i]){
7              prime.push_back(i);
8              phi[i]=i-1;
9          }
10         for(int p: prime){
11             if(p*i>n) break;
12             b[p*i]=1;
13             if(i%p==0){
14                 phi[p*i]=phi[i]*p;
15                 break;
16             }
17             phi[p*i]=phi[i]*phi[p];
18         }
19     }

```

```
20 | }
```

### 3.4 类欧几里得

### 3.5 递推组合数

```
1 | ll C[N][N];
2 | void calcC(int n){
3 |     C[0][0]=1;
4 |     REP(i, 1, n){
5 |         C[i][0]=C[i][i]=1;
6 |         REP(j, 1, i-1) C[i][j]=(C[i-1][j]+C[i-1][j-1])%MOD;
7 |     }
8 | }
```

### 3.6 矩阵快速幂

```
1 | struct mat{
2 |     ll n, m;
3 |     vector<vector<ll>> val; // 注意下标从 0 开始
4 |     mat(ll _n, ll _m): n(_n), m(_m){
5 |         val.resize(_n);
6 |         REP(i, 0, _n-1) val[i].resize(_m, 0);
7 |     }
8 |     mat(ll _n, ll _m, vector<vector<ll>> _val):
9 |         n(_n), m(_m), val(_val){}
10 | };
11 |
12 | mat mul(const mat& x, const mat& y, ll mod){
13 |     assert(x.m==y.n);
14 |     mat res(x.n, y.m);
15 |     REP(i, 0, res.n-1) REP(j, 0, res.m-1){
16 |         REP(k, 0, x.m-1){
17 |             res.val[i][j]=(res.val[i][j]+x.val[i][k]*y.val[k][j]%
18 |                 mod)%mod;
19 |         }
20 |     }
21 |     return res;
22 | }
```

```

23 mat qpow(mat a, ll x, ll mod){
24     assert(a.n==a.m);
25     mat res(a.n, a.n);
26     REP(i, 0, a.n-1) res.val[i][i]=1;
27     REP(i, 0, a.n-1) REP(j, 0, a.m-1) a.val[i][j]%=mod;
28     while(x){
29         if(x&1) res=mul(res, a, mod);
30         a=mul(a, a, mod), x>>=1;
31     }
32     return res;
33 }

```

### 3.7 扩展欧几里得

```

1 ll exgcd(ll a, ll b, ll &x, ll &y){
2     if(b==0){
3         x=1, y=0;
4         return a;
5     }
6     ll xx=0, yy=0;
7     ll res=exgcd(b, a%b, xx, yy);
8     y=xx-(a/b)*yy, x=yy;
9     return res;
10 }
11
12 ll inv(ll x, ll mod){
13     ll xx, yy;
14     ll d=exgcd(x, mod, xx, yy);
15     assert(d==1);
16     return (xx%mod+mod)%mod;
17 }

```

## 4 图论

### 4.1 倍增

```

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

```

```

5         dfs(v, x);
6     }
7 }
8
9 int lca(int x, int y){
10     if (dep[x]<dep[y]) swap(x, y);
11     int t=dep[x]-dep[y];
12     REP(i, 0, SP) if(t&(1<<i)) x=pa[x][i];
13     REPd(i, SP-1, -1){
14         int xx=pa[x][i], yy=pa[y][i];
15         if (xx!=yy) x=xx, y=yy;
16     }
17     return x==y?x:pa[x][0];
18 }

```

## 4.2 网络流

不是我写的，但是看着还好

其中 11 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

```

1 constexpr ll INF = LLONG_MAX / 2;
2
3 struct E {
4     int to; ll cp;
5     E(int to, ll cp): to(to), cp(cp) {}
6 };
7
8 struct Dinic {
9     static const int M = 1E5 * 5;
10    int m, s, t;
11    vector<E> edges;
12    vector<int> G[M];
13    int d[M];
14    int cur[M];
15
16    void init(int n, int s, int t) {
17        this->s = s; this->t = t;
18        for (int i = 0; i <= n; i++) G[i].clear();
19        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;

```

```

60     }
61
62     ll go() {
63         ll flow = 0;
64         while (BFS()) {
65             memset(cur, 0, sizeof cur);
66             flow += DFS(s, INF);
67         }
68         return flow;
69     }
70 } DC;

```

#### 4.2.2 费用流

```

1  constexpr ll INF = LLONG_MAX / 2;
2
3  struct E {
4      int from, to; ll cp, v;
5      E() {}
6      E(int f, int t, ll cp, ll v) : from(f), to(t), cp(cp), v(v) {}
7  };
8
9  struct MCMF {
10     static const int M = 1E5 * 5;
11     int n, m, s, t;
12     vector<E> edges;
13     vector<int> G[M];
14     bool inq[M];
15     ll d[M], a[M];
16     int p[M];
17
18     void init(int _n, int _s, int _t) {
19         n = _n; s = _s; t = _t;
20         REP (i, 0, n + 1) G[i].clear();
21         edges.clear(); m = 0;
22     }
23
24     void addedge(int from, int to, ll cap, ll cost) {
25         edges.emplace_back(from, to, cap, cost);
26         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));

```

```

67         return make_pair(flow, cost);
68     }
69 } 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）

```

1  struct MaxMatch {
2      int n;
3      vector<int> G[N];
4      int vis[N], left[N], clk;
5
6      void init(int n) {
7          this->n = n;
8          REP (i, 0, n + 1) G[i].clear();
9          memset(left, -1, sizeof left);
10         memset(vis, -1, sizeof vis);
11     }
12
13     bool dfs(int u) {
14         for (int v: G[u])
15             if (vis[v] != clk) {
16                 vis[v] = clk;
17                 if (left[v] == -1 || dfs(left[v])) {
18                     left[v] = u;
19                     return true;
20                 }
21             }
22         return false;
23     }
24
25     int match() {
26         int ret = 0;
27         for (clk = 0; clk <= n; ++clk)
28             if (dfs(clk)) ++ret;
29         return ret;
30     }
31 } MM;

```



## 4.4 Tarjan 强连通分量缩点

```
1  int low[N], dfn[N], clk, B, bl[N];
2  vector<int> bcc[N];
3  void init() { B = clk = 0; memset(dfn, 0, sizeof dfn); }
4  void tarjan(int u) {
5      static int st[N], p;
6      static bool in[N];
7      dfn[u] = low[u] = ++clk;
8      st[p++] = u; in[u] = true;
9      for (int& v: G[u]) {
10         if (!dfn[v]) {
11             tarjan(v);
12             low[u] = min(low[u], low[v]);
13         } else if (in[v]) low[u] = min(low[u], dfn[v]);
14     }
15     if (dfn[u] == low[u]) {
16         ++B;
17         while (1) {
18             int x = st[--p]; in[x] = false;
19             bl[x] = B; bcc[B].push_back(x);
20             if (x == u) break;
21         }
22     }
23 }
```

## 4.5 树直径

```
1  ll n, dep[N], mxdep[N];
2  vector<int> g[N];
3  vector<int> dmt;
4  void dfs1(int x, int fa){
5      dep[x]=dep[fa]+1;
6      mxdep[x]=1;
7      for(int u: g[x]) if(u!=fa){
8          dfs1(u, x);
9          mxdep[x]=max(mxdep[u]+1, mxdep[x]);
10     }
11 }
12
```

```

13 void dfs2(int x, int fa){ // 找一条直径，如果只需要直径长度则不用
14     dmt.push_back(x);
15     int nxt=-1;
16     for(int u: g[x]) if(u!=fa){
17         if(nxt==-1 || mxdep[u]>mxdep[nxt]) nxt=u;
18     }
19     if(nxt!=-1)
20         dfs2(nxt, x);
21 }
22
23 void diameter(){
24     dep[0]=0;
25     dfs1(1, 0);
26     int rt=max_element(dep+1, dep+n+1)-dep;
27     dfs1(rt, 0);
28     dfs2(rt, 0);
29 }

```

## 4.6 树重心

```

1  ll n, sz[N], mxsz[N], G;
2  vector<int> g[N];
3  void dfs(int x, int fa){
4      sz[x]=1, mxsz[x]=0;
5      for(int u: g[x]) if(u!=fa){
6          dfs(u, x);
7          sz[x]+=sz[u];
8          mxsz[x]=max(sz[u], mxsz[x]);
9      }
10     mxsz[x]=max(mxsz[x], n-sz[x]);
11     if(G==-1 || (mxsz[x]<mxsz[G] || (mxsz[x]==mxsz[G]&& x<G)))
12         G=x;
13 }

```

## 4.7 树链剖分

## 4.8 最短路

### 4.8.1 Floyd (最小环)

```

1  ll n, m, mincycle;

```

```

2 | ll g[N][N], dis[N][N];
3 | void floyd(){
4 |     // 如果 g[i][j] 之间没边则存 inf
5 |     // 注意 inf 的三倍不能爆 long long
6 |     mincycle=LLONG_MAX/4;
7 |     REP(i, 1, n) REP(j, 1, n) dis[i][j]=g[i][j];
8 |     REP(k, 1, n){
9 |         REP(i, 1, k-1)
10 |            REP(j, i+1, k-1){
11 |                mincycle=min(mincycle, dis[i][j]+g[j][k]+g[k][i]);
12 |            }
13 |         REP(i, 1, n)
14 |            REP(j, 1, n)
15 |                dis[i][j]=min(dis[i][j], dis[i][k]+dis[k][j]);
16 |     }
17 | }

```

#### 4.8.2 Spfa (判负环)

#### 4.8.3 Dijkstra

#### 4.9 拓扑排序

#### 4.10 最小生成树

#### 4.11 欧拉路径/回路

#### 4.12 01 图黑白染色

### 5 字符串

#### 5.1 KMP

#### 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)