**BUAA\_OhMyGold**

# DS&STL

目录

[DS&STL 0](#_Toc308013909)

[数据结构 2](#_Toc308013910)

[树状数组 2](#_Toc308013911)

[二维树状数组 2](#_Toc308013912)

[三维树状数组 2](#_Toc308013913)

[扩展树状数组(区间加值，区间查询) 4](#_Toc308013914)

[左偏树 6](#_Toc308013915)

[Treap 8](#_Toc308013916)

[线段树类 11](#_Toc308013917)

[Splay 12](#_Toc308013918)

[STL 20](#_Toc308013919)

[Map/Multimap 20](#_Toc308013920)

[Set/Multiset 22](#_Toc308013921)

[Priority\_queue/deque 23](#_Toc308013922)

[Bitset 23](#_Toc308013923)

# 数据结构

树状数组

## 二维树状数组

int c[N][N], n;

int lowbit(int x)

{

return x & (x ^ (x - 1));

}

bool insert(int x, int y, int delta)

{

for (int i = x; i <= n; i += lowbit(i))

for (int j = y; j <= n; j += lowbit(j))

c[i][j] += delta;

return true;

}

int getsum(int x, int y)

{

int s = 0;

for (int i = x; i > 0; i -= lowbit(i))

for (int j = y; j > 0; j -= lowbit(j))

s += c[i][j];

return s;

}

int getsum(int x1, int y1, int x2, int y2)

{

return getsum(x2, y2) - getsum(x1 - 1, y2) - getsum(x2, y1 - 1) + getsum(x1 - 1, y1 - 1);

}

## 三维树状数组

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

int c[130][130][130], n;

int lowbit(int x)

{

return (x & (x ^ (x - 1)));

}

void insert(int x, int y, int z, int delta)

{

for (int i = x; i <= n; i += lowbit(i))

for (int j = y; j <= n; j += lowbit(j))

for (int k = z; k <= n; k += lowbit(k))

c[i][j][k] += delta;

}

int getsum(int x, int y, int z)

{

int s = 0;

for (int i = x; i; i -= lowbit(i))

for (int j = y; j; j -= lowbit(j))

for (int k = z; k; k -= lowbit(k))

s += c[i][j][k];

return s;

}

int getsum(int x1, int y1, int z1, int x2, int y2, int z2)

{

return getsum(x2, y2, z2) - getsum(x1 - 1, y2, z2) - getsum(x2, y1 - 1, z2) - getsum(x2, y2, z1 - 1) + getsum(x1 - 1, y1 - 1, z2) + getsum(x1 - 1, y2, z1 - 1) + getsum(x2, y1 - 1, z1 - 1) - getsum(x1 - 1, y1 - 1, z1 - 1);

}

int main()

{

int k, x1, y1, z1, x2, y2, z2, m;

while (~scanf("%d", &n))

{

memset(c, 0, sizeof(c));

while (scanf("%d", &m), (m != 3))

{

if (m == 1)

{

scanf("%d%d%d%d", &x1, &y1, &z1, &k);

x1++, y1++, z1++;

insert(x1, y1, z1, k);

}

else if (m == 2)

{

scanf("%d%d%d%d%d%d", &x1, &y1, &z1, &x2, &y2, &z2);

x1++, y1++, z1++, x2++, y2++, z2++;

printf("%d\n", getsum(x1, y1, z1, x2, y2, z2));

}

}

}

return 0;

}

## 扩展树状数组(区间加值，区间查询)

#include <iostream>

#include <cstdio>

#include <cstring>

#define LL long long

using namespace std;

LL lowbit(LL x)

{

return x & (x ^ (x - 1));

}

struct BIT{

LL c[100010], n;

BIT(LL \_n)

{

memset(c, 0, sizeof(c));

n = \_n;

}

BIT(){}

void insert(LL x, LL delta)

{

while (x <= n)

{

c[x] += delta;

x += lowbit(x);

}

}

LL getsum(LL x)

{

LL s = 0;

while (x)

{

s += c[x];

x -= lowbit(x);

}

return s;

}

}a, b;

LL sum[100010];

LL n, q, x, y, z;

char str[3];

LL getsum(LL s)

{

if (s == 0) return 0;

return a.getsum(s) \* (s + 1) - b.getsum(s);

}

int main()

{

while (~scanf("%lld%lld", &n, &q))

{

a = BIT(n), b = BIT(n);

memset(sum, 0, sizeof(sum));

for (LL i = 1; i <= n; ++i)

{

scanf("%lld", &x);

sum[i] = sum[i - 1] + x;

}

for (LL i = 1; i <= q; ++i)

{

scanf("%s", str);

if (str[0] == 'C')

{

scanf("%lld%lld%lld", &x, &y, &z);

a.insert(x, z);

a.insert(y + 1, -z);

b.insert(x, z \* x);

b.insert(y + 1, -z \* (y + 1));

}

else

{

scanf("%lld%lld", &x, &y);

printf("%lld\n", getsum(y) - getsum(x - 1) + sum[y] - sum[x - 1]);

}

}

}

return 0;

}

左偏树

/\*

可合并的优先队列(小堆)

所有操作都是log(n)的

\*/

#define typec int // type of key val

#define N 50005

const int na = -1;

struct node

{

typec key;//值

int l, r, f, dist;

//左子，右子，父亲，距离

}tr[N];

//左偏树的根结点的父亲为na

int ary[N];

// 获得结点i的根

int iroot(int i)

{

if (i == na) return i;

while (tr[i].f != na) i = tr[i].f;

return i;

}

// 合并两棵左偏树, param: rx ry (two root)

int merge(int rx, int ry)

{

if (rx == na) return ry;

if (ry == na) return rx;

if (tr[rx].key < tr[ry].key)

swap(rx, ry);

int r = merge(tr[rx].r, ry);

tr[rx].r = r; tr[r].f = rx;

if (tr[r].dist > tr[tr[rx].l].dist)

swap(tr[rx].l, tr[rx].r);

if (tr[rx].r == na) tr[rx].dist = 0;

else tr[rx].dist = tr[tr[rx].r].dist + 1;

return rx;// return new root

}

//插入一个新节点

//标号为i的结点看做新树，合并到原树中

int ins(int i, typec key, int root)

{

tr[i].key = key;

tr[i].l = tr[i].r = tr[i].f = na;

tr[i].dist = 0;

return root = merge(root, i);// return new root

}

// 删除某个已知结点（不常用）

int del(int i)

{

if (i == na) return i;

int x, y, l, r;

l = tr[i].l; r = tr[i].r; y = tr[i].f;

tr[i].l = tr[i].r = tr[i].f = na;

tr[x = merge(l, r)].f = y;

if (y != na && tr[y].l == i) tr[y].l = x;

if (y != na && tr[y].r == i) tr[y].r = x;

for ( ; y != na; x = y, y = tr[y].f) {

if (tr[tr[y].l].dist < tr[tr[y].r].dist)

swap(tr[y].l, tr[y].r);

if (tr[tr[y].r].dist + 1 == tr[y].dist) break;

tr[y].dist = tr[tr[y].r].dist + 1;

}

if (x != na) return iroot(x);// return new root

else return iroot(y);

}

// 获取最小结点

node top(int root)

{

return tr[root];

}

// 取得并删除最小结点

node pop(int &root)

{

node out = tr[root];

int l = tr[root].l, r = tr[root].r;

tr[root].l = tr[root].r = tr[root].f = na;

tr[l].f = tr[r].f = na;

root = merge(l, r);

return out;

}

// 增/减一个结点的键值

int add(int i, typec val)

{

if (i == na) return i;

if (tr[i].l == na && tr[i].r == na && tr[i].f == na)

{

tr[i].key += val;

return i;

}

typec key = tr[i].key + val;

int rt = del(i);

return ins(i, key, rt);

}

// 初始化数据，每个点一棵树

void init(int n)

{

for (int i = 0; i < n; i++)

{

scanf("%d", &ary[i]);

tr[i].key = ary[i];

tr[i].l = tr[i].r = tr[i].f = na;

tr[i].dist = 0;

}

/\*\*构建初始左偏树：O(n)

将n个节点（每个节点作为一棵左偏树）放入先进先出队列。

 不断地从队首取出两棵左偏树，将它们合并之后加入队尾。

 当队列中只剩下一棵左偏树时，算法结束。

\*/

}

Treap

/\*

有一点点随机成分。。。当AVL树用就成了。

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

#include <ctime>

#include <cstdlib>

#define MAX 20000000

using namespace std;

typedef struct

{

int l,r,key,fix;

}node;

class treap

{

public:

node p[MAX];

int size,root;

treap()

{

//srand(time(0));人品摇号机。。。

size = -1;

root = -1;

}

void rot\_l(int &x)

{

int y = p[x].r;

p[x].r = p[y].l;

p[y].l = x;

x = y;

}

void rot\_r(int &x)

{

int y = p[x].l;

p[x].l = p[y].r;

p[y].r = x;

x = y;

}

void insert(int &k,int tkey)

{

if (k == -1)

{

k = ++size;

p[k].l = p[k].r = -1;

p[k].key = tkey;

p[k].fix = rand();

}

else if (tkey < p[k].key)

{

insert(p[k].l,tkey);

if (p[ p[k].l ].fix > p[k].fix)

rot\_r(k);

}

else

{

insert(p[k].r,tkey);

if (p[ p[k].r ].fix > p[k].fix)

rot\_l(k);

}

}

void remove(int &k,int tkey)

{

if (k == -1) return;

if (tkey < p[k].key)

remove(p[k].l,tkey);

else if (tkey > p[k].key)

remove(p[k].r,tkey);

else

{

if (p[k].l == -1 && p[k].r == -1)

k = -1;

else if (p[k].l == -1)

k = p[k].r;

else if (p[k].r == -1)

k = p[k].l;

else if (p[ p[k].l ].fix < p[ p[k].r ].fix)

{

rot\_l(k);

remove(p[k].l,tkey);

}

else

{

rot\_r(k);

remove(p[k].r,tkey);

}

}

}

void remove\_max(int &k)

{

if (k == -1) return;

if (p[k].r == -1)

{

k = p[k].l;

return;

}else remove\_max(p[k].r);

}

void remove\_min(int &k)

{

if (p[k].l == -1)

{

k = p[k].r;

return;

}

else remove\_min(p[k].l);

}

void print(int k)

{

if (p[k].l != -1)

print(p[k].l);

cout << p[k].key << " : " << p[k].fix << endl;

if (p[k].r!=-1)

print(p[k].r);

}

};

treap T;

线段树类

#define N 500005

struct NODE

{

int L,R;

int val;

};

struct SegmentTree

{

int n;NODE v[3\*N];

void init()

{

Build(1,1,n);

}

void Build(int p,int l,int r)

{

v[p].L = l;v[p].R = r;

v[p].val = 0;

if (l == r)

;

else

{

int mid = (l + r)/2;

Build(p\*2,l,mid);

Build(p\*2 + 1,mid + 1,r);

}

}

void Add();

void Chg();

void Query();

};

Splay

#include <iostream>

#include <cstring>

#include <algorithm>

#include <vector>

#include <cmath>

#include <string>

#include <map>

#include <fstream>

using namespace std;

typedef long long ll;

const int MAXN = 102000 \* 2;

const int INF = (1 << 28);

int ary[MAXN];

struct Node{

int val, sz;

int sum, ml, mr, mc;

Node \*ch[2], \*pre;

};

class Splay{

public:

Node \*null, \*root, \*S[MAXN], data[MAXN];

int cnt, top;

Node \* getNewNode(int x){

Node \*p;

if(top){

p = S[top--];

}else{

p = &data[cnt++];

}

p->val = p->sum = p->ml = p->mr = p->mc = x;

p->sz = 1;

p->ch[0] = p->ch[1] = p->pre = null;

return p;

}

void init(){

cnt = top = 0;

null = getNewNode(-INF);

null->sum = null->sz = 0;

root = getNewNode(-INF);

root->sum = 0;

root->ch[1] = getNewNode(INF);

root->ch[1]->sum = 0;

root->ch[1]->pre = root;

updata(root);

}

void updata(Node \*p){

p->sz = p->ch[0]->sz + p->ch[1]->sz + 1;

p->sum = p->ch[0]->sum + p->ch[1]->sum + p->val;

//这玩意怎么和线段树那么像啊！

p->ml = max(p->ch[0]->ml, p->ch[0]->sum + p->val + max(p->ch[1]->ml, 0));

p->mr = max(p->ch[1]->mr, p->ch[1]->sum + p->val + max(p->ch[0]->mr, 0));

//ml 左边连续的最大值, mr 右边连续的最大值, mc 该点作为根时连续的最大值

p->mc = max(p->ch[0]->mc, p->ch[1]->mc);

p->mc = max(p->mc, max(p->ch[0]->mr + p->ch[1]->ml, 0) + p->val);

p->mc = max(p->mc, max(p->ch[0]->mr, p->ch[1]->ml) + p->val);

}

void rotate(Node \*x, int c){

//c = 1 是右旋 0 是左旋 , 将 x 节点旋转到它父亲的位置

Node \*y = x->pre;

y->ch[!c] = x->ch[c];

if(x->ch[c] != null){

x->ch[c]->pre = y;

}

x->pre = y->pre;

if(y->pre != null){

if(y->pre->ch[0] == y){

y->pre->ch[0] = x;

}else {

y->pre->ch[1] = x;

}

}

x->ch[c] = y;

y->pre = x;

if(y == root){

root = x;

}

updata(y);

}

void splay(Node \*x, Node \*f){

//把节点 x 旋转为 f 的儿子，若 f 为 null 则把 x 旋转到根的位置

while(x->pre != f){

if(x->pre->pre == f){//zig

rotate(x, x->pre->ch[0] == x);

/\*if(x->pre->ch[0] == x){

rotate(x, 1); //右旋

}else {

rotate(x, 0); //左旋

} \*/// <==> rotate(x, x->pre->ch[0] == x);

}else {

Node \*y = x->pre;

Node \*z = y->pre;

if(z->ch[0] == y){

if(y->ch[0] == x){

rotate(y, 1), rotate(x, 1);//zig-zig

}else{

rotate(x, 0), rotate(x, 1);//zig-zag

}//

}else {

if(y->ch[1] == x){

rotate(y, 0), rotate(x, 0);//zig-zig

}else {

rotate(x, 1), rotate(x, 0); //zig-zag

}

}

}

}

updata(x);

}

void select(int kth, Node \*f){

//找到第 k 个元素，并把该元素旋转使之成为 f 的儿子

int tmp;

Node \*t = root;

while(true){

tmp = t->ch[0]->sz;

if(tmp + 1 == kth){//find the right one

break;

}

if(kth <= tmp){

t = t->ch[0];

}else {

kth -= tmp + 1;

t = t->ch[1];

}

}

splay(t, f);

}

void insert(int pos, int val){

//在 pos 之前插入一个节点

Node \*tmproot = getNewNode(val);

select(pos - 1, null);

select(pos, root);

root->ch[1]->ch[0] = tmproot;

tmproot->pre = root->ch[1];

splay(root->ch[1], null);

}

void del(int k){

//删除第 k 个元素, k 不能为 1;

select(k, null);

Node \*old\_root = root;

root = root->ch[1];

root->pre = null;

select(1, null);

root->ch[0] = old\_root->ch[0];

root->ch[0]->pre = root;

updata(root);

S[++top] = old\_root;

}

void replace(int x, int y){

//把第 x 位置的元素值替换为 y

select(x, null);

root->val = y;

updata(root);

}

Node \*build(int l, int r){

// 这不就是线段树的建树过程吗？

if(l > r){

return null;

}

int m = (l + r) >> 1;

Node \*p = getNewNode(ary[m]);

p->ch[0] = build(l, m - 1);

if(p->ch[0] != null){

p->ch[0]->pre = p;

}

p->ch[1] = build(m + 1, r);

if(p->ch[1] != null){

p->ch[1]->pre = p;

}

updata(p);

return p;

}

};

Splay sp;

int main()

{

int n, m, i, x, y;

char c;

sp.init();

scanf("%d", &n);

for(int i = 1; i <= n; ++i){

scanf("%d", &ary[i]);

}

Node \*troot = sp.build(1, n);

sp.root->ch[1]->ch[0] = troot;

troot->pre = sp.root->ch[1];

sp.updata(sp.root->ch[1]);

sp.updata(sp.root);

sp.splay(sp.root->ch[1], sp.null);

scanf("%d", &m);

for(int i = 1; i <= m; ++i){

scanf(" %c", &c);

if(c == 'I'){

scanf("%d %d", &x, &y);

sp.insert(++x, y);

}else if(c == 'D'){

scanf("%d", &x);

sp.del(++x);

}else if(c == 'R'){

scanf("%d %d", &x, &y);

sp.replace(++x, y);

}else if(c == 'Q'){

scanf("%d %d", &x, &y);

sp.select(++x - 1, sp.null);

sp.select(++y + 1, sp.root);

printf("%d/n", sp.root->ch[1]->ch[0]->mc);

}

}

return 0;

}

数组版：

const int maxNodeCnt = 111111;

struct SplayTree {

int nodeCnt, root, type[maxNodeCnt],

parent[maxNodeCnt], childs[maxNodeCnt][2],

size[maxNodeCnt], stack[maxNodeCnt],reversed[maxNodeCnt];

// ...

void clear(){

root = 0;

size[0] = 0;

nodeCnt = 1;

}

int malloc() {

type[nodeCnt] = 2;

childs[nodeCnt][0] = childs[nodeCnt][1] = 0;

size[nodeCnt] = 1;

reversed[nodeCnt] = 0;

return nodeCnt ++;

}

void update(int x) {

size[x] = size[childs[x][0]] + 1 + size[childs[x][1]];

// ...

}

void pass(int x) {

// NOTICE: childs[x][i] == 0

if (reversed[x]) {

swap(childs[x][0], childs[x][1]);

type[childs[x][0]] = 0;

reversed[childs[x][0]] ^= 1;

type[childs[x][1]] = 1;

reversed[childs[x][1]] ^= 1;

reversed[x] = 0;

}

// ...

}

void rotate(int x) {

int t = type[x],

y = parent[x],

z = childs[x][1 - t];

type[x] = type[y];

parent[x] = parent[y];

if (type[x] != 2) {

childs[parent[x]][type[x]] = x;

}

type[y] = 1 - t;

parent[y] = x;

childs[x][1 - t] = y;

if (z) {

type[z] = t;

parent[z] = y;

}

childs[y][t] = z;

update(y);

}

void splay(int x) {

int stackCnt = 0;

stack[stackCnt ++] = x;

for (int i = x; type[i] != 2; i = parent[i]) {

stack[stackCnt ++] = parent[i];

}

for (int i = stackCnt - 1; i > -1; -- i) {

pass(stack[i]);

}

while (type[x] != 2) {

int y = parent[x];

if (type[x] == type[y]) {

rotate(y);

} else {

rotate(x);

}

if (type[x] == 2) {

break;

}

rotate(x);

}

update(x);

}

int find(int x, int rank) {

while (true) {

pass(x);

if (size[childs[x][0]] + 1 == rank) {

break;

}

if (rank <= size[childs[x][0]]) {

x = childs[x][0];

} else {

rank -= size[childs[x][0]] + 1;

x = childs[x][1];

}

}

return x;

}

void split(int &x, int &y, int a) {

// NOTICE: x, y != 0

y = find(x, a + 1);

splay(y);

x = childs[y][0];

type[x] = 2;

childs[y][0] = 0;

update(y);

}

void split3(int &x, int &y, int &z, int a, int b) {

split(x, z, b);

split(x, y, a - 1);

}

void join(int &x, int y) {

// NOTICE x, y != 0

x = find(x, size[x]);

splay(x);

childs[x][1] = y;

type[y] = 1;

parent[y] = x;

update(x);

}

void join3(int &x, int y, int z) {

join(y, z);

join(x, y);

}

int getRank(int x) {

splay(x);

root = x;

return size[childs[x][0]];

}

void reverse(int a, int b) {

int x, y;

split3(root, x, y, a + 1, b + 1);

reversed[x] ^= 1;

join3(root, x, y);

}

} splayTree;

int main() {

return 0;

}

STL

## Map/Multimap

//牢记map中所有元素都是pair

map内部自建一颗红黑树(一种非严格意义上的平衡二叉树)，这颗树具有对数据自动排序的功能，所以在map内部所有的数据都是有序的。（内存也会占不少）

以map<int, string> mapStudent;为例。

通用函数：

//size(),clear(),empty();

//前向迭代器遍历

for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)

iter->first,iter->second

//反向迭代器遍历

for(iter = mapStudent.rbegin(); iter != mapStudent.rend(); iter++)

//数组下标式遍历

for(int i = 1; i <= mapStudent.size(); ++i)

{

cout<<mapStudent[i]<<endl;

}

1. 构造

map<键，键值> 名称;

注意键是结构体时，需要重载<

例：map<int, string> mapStudent;

1. 插入

第一种：

/\*当已有相同键的元素存在时，插入失败

可以使用pair检查插入是否成功。

pair<map<int, string>::iterator, bool> Insert\_Pair;

Insert\_Pair = mapStudent.insert(pair<int,string>(1,"student\_one"));

if(Insert\_Pair.second == true)则插入成功。

\*/

map<int, string> mapStudent;

mapStudent.insert(pair<int, string>(1, "student\_one"));

mapStudent.insert(pair<int, string>(2, "student\_two"));

mapStudent.insert(pair<int, string>(3, "student\_three"));

第二种方法：/\*有相同键元素存在，也插入失败\*/

map<int, string> mapStudent;

mapStudent.insert(map<int,string>::value\_type(1, "student\_one"));

mapStudent.insert(map<int,string>::value\_type(2, "student\_two"));

mapStudent.insert(map<int,string>::value\_type(3,"student\_three");

第三种：

/\*当有相同键的元素存在时，会覆盖原有值\*/

map<int, string> mapStudent;

mapStudent[1] = "student\_one";

mapStudent[2] = "student\_two";

mapStudent[3] = "student\_three";

1. 查找

使用函数.count(x)，map中只有0，1两种返回值

使用find函数或后面函数方法：

map<int, string>::iterator iter;

iter = mapStudent.find(1);

//查找键，返回指向该键的迭代器

if(iter != mapStudent.end())

//此元素在map里存在

1. 删除

用迭代器删：

map<int,string>::iterator iter;

iter = mapStudent.find(1);

mapStudent.erase(iter);

用关键字删：

int n = mapStudent.erase(1);//如果删除了会返回1，否则返回0

用迭代器，成片的删除

//一下代码把整个map清空

mapStudent.erase(mapStudent.begin(), mapStudent.end());

//成片删除要注意的是，也是STL的特性，删除区间是一个前闭后开的集合

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

注意multimap不支持下标[]访问

另一种查找方法，更适于multimap：

lower\_bound( const keytype& x ) 找到比指定键值x小等的键值的第一个元素

upper\_bound( const keytype& x ) 找到比指定键值x大的键值的第一个元素

返回值为该元素的游标。

细节： 当到达键值x已经是最大时，upper\_bound返回的是这个multimap的end游标。同理，当键值x已经是最小了，lower\_bound返回的是这个multimap的begin游标。

函数equal\_range：（基于二分）可以得到所有等于给定值的值对

返回游标对pair::first是由函数lower\_bound得到的x的前一个值，游标对pair::second的值是由函数upper\_bound得到的x的后一个值

在map中如果这两个迭代器相等的话，则说明map中不出现这个关键字

而multimap中比较有用。

例：multimap<int,int> a;

a.insert(pair<int,int>(1,11));

…//插入元素

multimap<int,int>::iterator p\_map;

pair<multimap<int,int>::iterator, multimap<int,int>::iterator> ret;

for(p\_map = a.begin() ; p\_map != a.end();)//注意这里没有p\_map++

{

cout<<p\_map->first<<" =>";

ret = a.equal\_range(p\_map->first);

//返回两个迭代器，可用distance(p.first, p.second)求个数

for(p\_map = ret.first; p\_map != ret.second; ++p\_map)

cout<<" "<< (\*p\_map).second;

cout<<endl; //输出所有键及它所对应的各个值

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## Set/Multiset

1. 构造

set <type> s;

如果是结构体需要重载<

也可以这样：

int a[5]={1,2,3,4,5};

set<int> set2(a,a+3);

set<int> set3(set2);

1. 插入

直接insert元素。

使用pair <set<type>::iterator,bool> p;

可以检查是否插入成功，而且提供所插入元素的迭代器位置

p = s.insert(1);

if(p.second)插入成功，p.first得到插入元素的迭代器

1. 查找

使用.count()函数，set返回0或1，multiset返回个数

使用find(),得到位置

1. 删除

使用erase();参数可以是迭代器，数值(multiset会把相等的全删掉)，迭代器范围。

1. 其它支持函数

begin()，end()，rbegin(),rend()正向，反向遍历

clear();size();empty();

equal\_range();lower\_bound();upper\_bound();查找

## Priority\_queue/deque

公有函数：size();empty();

priority\_queue函数

注意重载<

push();pop();top();

deque函数

第一个数据front();最后一个数据end();

可以用at(ind)或[ind]访问下标为ind的元素，注意防越界

pop\_back();pop\_front();

push\_back(elem);push\_front(elem);

clear();

.erase(pos) 删除pos(迭代器)位置的数据

.erase(beg,end) 删除[beg,end)区间的数据

传回下一个数据的位置。

.insert(pos,elem) 在pos位置插入一个elem拷贝，传回新数据位置。

## Bitset

构造函数

bitset<n> b;

b有n位，每位都为0.参数n可以为一个表达式.

如bitset<5> b0;则"b0"为"00000";

bitset<n> b(unsigned long u);

b有n位,并用u赋值;如果u超过n位,则顶端被截除

如:bitset<5>b0(5);则"b0"为"00101";

bitset<n> b(string s);

b是string对象s中含有的位串的副本

string bitval ( "10011" );

bitset<5> b0 ( bitval4 );

则"b0"为"10011";

bitset<n> b(s, pos);

b是s中从位置pos开始位的副本,前面的多余位自动填充0;

string bitval ("01011010");

bitset<10> b0 ( bitval, 3 );

则"b0" 为 "0000011010";

bitset<n> b(s, pos, num);

b是s中从位置pos开始的num个位的副本,如果num<n,则前面的空位自动填充0;

string bitval ("11110011011");

bitset<6> b0 ( bitval5, 3, 6 );

则"b0" 为 "100110";

其它可用函数

os << b

把b中的位集输出到os流

os >>b

输入到b中,如"cin>>b",如果输入的不是0或1的字符,只取该字符前面的二进制位.

bool any()

//是否存在置为1的二进制位？和none()相反

bool none()

//是否不存在置为1的二进制位,即全部为0？和any()相反.

size\_t count()//二进制位为1的个数.

size\_t size()//二进制位的个数

flip()//把所有二进制位逐位取反

flip(size\_t pos)//把在pos处的二进制位取反

bool operator[]( size\_type \_Pos )//类数组下标访问

set()//把所有二进制位都置为1

set(pos)//把在pos处的二进制位置为1

reset()//把所有二进制位都置为0

reset(pos)//把在pos处的二进制位置为0

test(size\_t pos)//在pos处的二进制位是否为1？

unsigned long to\_ulong()//用同样的二进制位返回一个unsigned long值

string to\_string ()//返回对应的字符串.