**Code Library**

Wang Lulu

July, 2015

**Contents**

1. 经验与技巧

A.

B.

C.

1. 常用STL

A: 定义自己的类型

B: vector

C: stack

D: queue

E: priority\_queue

F: set/multiset

G: map/multimap

H: algorithm

1. 结构及操作

A

B

C

1. 通用算法

A

B

C

1. 数学方法

A

B

C

1. 特化问题及其算法

A

B

C

1. 经验与技巧

1.int数组快速初始化：

#include<string.h> //注意这里是string.h

int m[100][200];

memset(m,-1,sizeof(m)); //将m全初始化为-1

memset(m,0,sizeof(m)); //将m全初始化为0

1. 常用STL

**A: 自己定义类型**

struct Type{

int a;

bool b;

char c;

...

//------------------------

Type(){ //构造函数(可选)

a = xx;

b = xx;

...

}

Type(int \_a, bool \_b){ //带参构造函数(可选)

a = \_a;

b = \_b;

...

}

//----------------------

/\*使Type具有判等性(可选)\*/

friend bool operator==(Type x,Type y){

//自己定义相等的概念

if(x.a==y.a&&x.b==y.b&&x.c==y.c&&...)

return true;

else

return false;

}

/\*使Type自身具有可比性(可以比较大小)(可选)\*/

friend bool operator<(Type x,Type y){

//自己定义<的概念，如果返回TRUE认为x<y,

//否则认为x>=y

//eg. return x.a<y.a && x.c<y.c;

}

};

**B: vector**

1. ***vector中的元素在内存中是顺序排列的，对查找的开销小，对插入的开销大，但在尾部插入仍是O(1)。***
2. ***vector没有越界检查，要自己控制以防越界。***
3. ***vector开销大，尽量用数组代替它***

#include<vector> //header

vector<Type> v; //declear

v.assign(size,value); //用size个value初始化v

v.assign(first\_iter,last\_iter);//用两个iterator之间的元素初始化v

//左闭右开：[first\_iter last\_lter)

v.clear(); //清空v中的元素

v.empty(); //v为空返回true否则返回false

v.push\_back(e); //push e to the back of v

v[index]; //read or write v use index

v.insert(iter,e); //把e插入iter所处位置，iter和之后 //的元素后移

v.front(); //return the first obj of v

v.back(); //return the last obj of v

v.size(); //返回v中的元素数

v.begin(); //return the iterator of the first obj

v.end(); //return the iterator after the last obj

//左必右开：[v.begin v.end)

***ADD: 迭代器iterator的用法：***

1.iterator是指针的扩充，用法和指针很相似。创建方式如下：

容器类型<元素类型>::iterator 名称；

eg: set<int>::iterator iter;

vector<char>::iterator aaa;

2.支持的操作如下：

访问内容： \*(iterater)

移向下一个元素： iterater++/++iterater

求两迭代器之间的元素数： last\_iter - first\_lter

\*注意：迭代器不支持加减运算，这一点与指针不同。迭代器的++/--操作对线性容器和关联容器都适用。

**B: stack**

#include<stack> //header

stack<Type> s; //declear

s.push(e); //push a obj

s.top(); //return the top of a stack

s.pop(); //pop a obj from the stack

s.size(); //return the size of the stack

s.empty(); //return true if the stack is empty(s.size=0), //false if not

**C: queue**

#include<queue> //header

queue<Type> q; //declear

q.push(e); //push a obj

q.front(); //return the front ob of the queue

q.back(); //return the last obj of the queue

q.pop(); //pop the first obj of the queue

q.empty(); //return true if the stack is empty(s.size=0),

//false if not

q.size(); //return the size of the queue

**D: priority\_queue**

***1.因为priority\_queue涉及到排序，因此要为Type制定优先级比较方式，否则会按Type本身的值比较结构作为优先级。***

***2.priority\_queue是大顶堆，即优先级最高的元素在最前面，通过制定不同的优先级比较方式可以得到该种优先级意义下的大顶堆。***

#include<queue> //header(the same with queue)

priority\_queue<Type> q; //declear(使用自身的值比较作为优先级,灵活性 //很差[不推荐！])

priority\_queue<Type,vector<Type>,my\_prior> q; //declear(使用 //my\_prior提供的优先级函数进行比较[推荐！])

q.push(e); //push a obj

q.top(); //return the top obj of the heap

q.pop(); //pop the top obj of the heap

q.empty(); //return true if the size is 0

q.size(); //return the size

**ADD: 提供优先级的my\_prior的写法：**

***注意my\_prior实际上是结构体(与sort函数的情况不同)，和operator的特殊写法。***

struct my\_prior{

bool operator()(Type x,Type y){

//返回TRUE认为x优先级低于y

//否则认为x优先级>=y

}

}

**E: set/multiset**

1. ***set与multiset本质是平衡搜索树，它是有序的，并且插入和查找时间都为log(n),解决了向量插入和列表查找时开销大的问题。***
2. ***set的元素不能重复，multiset的元素可以重复。***
3. ***set/multiset中的元素按优先级：左子树<根节点<右子树 的顺序构建，默认使用元素自身的比较方式作为优先级，可以自己指定优先级函数.***
4. ***set只能通过iterator访问元素的值***

#include<set> //header

set<Type> s; //同优先级队列，这里使用Type自己的比较方式 //作为优先级。

set<Type,my\_prior> s; //使用my\_prior提供的优先级函数进行比较。

multiset<Type> s; //允许元素重复，其余同上

multiset<Type,my\_prior> s;//同上

s.insert(e); //将e插入s中，set不保留重复的，multiset //会保留所有插入的元素

s.insert(first\_iter,last\_iter);//将[first\_iter last\_iter)内的所有 //指向的元素插入s

s.find(e); //返回一个iterator指向值为e的元素，没有发 //现返回s.end()

s.erase(e); //将值为e的元素删除。

s.erase(iter); //将iter指向的元素删除

s.erase(iter\_first,iter\_last); //将[first\_iter last\_iter)内所有指 //向的元素删除

s.clear(); //清空s中的所有元素

s.empty(); //s为空返回true,否则返回false

s.size(); //返回s中元素个数

s.begin()/s.end(); //同前面几节

s.count(e); //返回值为e的元素个数，这里只可能为1 //或0，multiset中可以为大于1的数

s.lower\_bound(e); //返回一个iterator，指向第一个大于等 //于e的元素(多用于multiset)

s.upper\_bound(e); //返回一个iterator，指向第一个大于 //e的元素(多用于multiset)

**F: map/multimap**

1. ***map即字典，实现字典通常有两种方式：hash表与二叉搜索树，map使用的是二叉搜索树。***
2. ***map可以看做每个元素都附加了一个属性值的set，即map中每一个元素都由key-value组成，如果忽略value只看key那么map和set几乎一样。***
3. ***与set相同，map中key值相同的元素只能有一个，而multiset可以有多个。***
4. ***map/multimap中元素的本质是一些pair，访问方式有两种：iterator和下标访问。***
5. ***同优先级队列，map/multimap中的元素按key的优先级：左子树<根节点<右子树 的顺序构建，默认使用元素中key的自身的比较方式作为优先级，可以自己指定优先级函数***

#include<map> //header

map<Type\_key,Type\_val> m;//同优先级队列，这里使用TypeA自己的比较方式 //作为优先级。

map<Type\_key,Type\_val,my\_prior> m;

//使用my\_prior为Type\_key提供的优先级函数 //进行比较。

multimap<Type\_key,Type\_val> m; //允许元素重复，其余同上

multimap<Type\_key,Type\_val,my\_prior> m; //同上

m.insert(pair\_e); //将pair\_e插入m中，map不保留重复的，而 //multimap会保留所有插入的元素

m.insert(first\_iter,last\_iter);//将[first\_iter last\_iter)内的所有 //指向的元素(pair类型)插入m

m.find(key); //返回一个iterator指向键值为key的元素，没 //有发现返回m.end()

m.erase(key); //将键值为key的元素删除。

m.erase(iter); //将iter指向的元素删除

m.erase(iter\_first,iter\_last); //将[first\_iter last\_iter)内所有指 //向的元素删除

m.clear(); //清空m中的所有元素

m.empty(); //m为空返回true,否则返回false

m.size(); //返回m中元素个数

m.begin()/s.end(); //同前面几节

m.count(key); //返回键值为key的元素个数，这里只可 //能为1或0，multimap中可以为大于1的数.

m.lower\_bound(key); //返回一个iterator，指向第一个键值大于等 //于key的元素(多用于multimap)

m.upper\_bound(key); //返回一个iterator，指向第一个键值大于 //key的元素(多用于multimap)

***ADD:***

***1. map/multimap的iterator指向的是pair，不能像别的容器那样直接用\*()访问其值，而是通过iter->first和iter->second来访问它的key与value.***

***2. map支持特有的下标访问方式，multimap不支持：***

***下标访问map与下标访问vector截然不同，下标访问vector类似于访问数组，而下标访问map中不存在的元素将导致在map中添加一个新元素，其key就是访问它的那个下标，value为对应类型下的默认初始值，如int为0，string为空串。***

1. 结构及操作
2. 通用算法
3. 数学方法

求最大公约数

1. int GCD(int x, int y) {
2. int t;
3. while(y > 0) {
4. t = x % y;
5. x = y;
6. y = t;
7. }
8. return x;
9. }

求最小公倍数：

1. x\*y/GCD(x,y)

判断是否为素数：

1. bool is\_prime(int u){
2. if(u==0||u==1) return false;
3. If(u==2) return true;
4. If(u%2==0) return false;
5. For(int i=3;i<=sqrt(u);i=i+2)
6. If(u%i==0) return false;
7. Return true;
8. }
9. 常见子问题