C++ with STL templates

compare

```cpp
#include<algorithm>

int greater=max(a,b);
int smaller=min(a,b);
```

sort

```cpp
#include<algorithm>
vector<int> arr;
arr.push_back(2);
arr.push_back(4);
arr.push_back(3);
arr.push_back(7);
sort(arr.begin(),arr.end());//默认从小到大排序
sort(nums.begin(),nums.end(),greater<int>());//从大到小排序
vector<vector<int>> grid;
grid.push_back([3,2]);
grid.push_back([5,3]);
grid.push_back([1,3]);
sort(grid.begin(),grid.end());//二维数组按照数组元素的第一个数字从小到大排序
//自定义排序：（优先队列的自定义排序也可以这么写）
struct cmp{
    bool operator()(vector<int>& a, vector<int>&b){//这里的两个参数随机应变
        return a[1]<b[1];//按照a、b数组的第二个元素从小到大排序
        //a<b: 从小到大;a>b: 从大到小
    }
};
sort(grid.begin(),grid.end(),cmp());
```

vector

```cpp
#include<vector>

vector<int> v;//声明
vector<int> v(20);//声明，且里面开始即留20个空间,默认初始化为0（及可以直接用v[i]访问）
vector<int> v(20,5);//默认初始化为20个5
//声明二维数组:
vector<vector<int>>(n,vector<int>(m,2));//n*m大小二维数组全部填满2
vector<vector<int>> move={{0,1},{1,0},{0,-1},{-1,0}};//在岛屿类题目中常见
//
v.push_back(num);//向容器末尾插入元素num
v.pop_back();//清除容器末尾的元素
v.back()//返回容器v最后一个元素
v.resize(n)//将v内部的元素个数改为n个，多出的删去，少的补（默认补0），本来就有且不超出的部分保留
v.resize(n,num)//补的部分补numv，其余同上
v.insert(v.begin(),num);//将num插入头部
v.insert(v.begin()+n,num);//将num插入某位置
```

```
v.empty()//是空则返回true
v.clear()//清空所有值
v.size()//v中的元素个数
v.capacity()//v占据的内存（注意和size()的区别）
//声明二维数组：
vector<vector<int>> v(n,vector<int>(m));
//在类的private中声明vector类型并初始化：
vector<int> v=vector<int>(5,5);
//迭代器的使用：
vector<int>::iterator it=v.begin();
it++;//迭代器向后移动一位
it<v.end();//判断结束
*it=10;//赋值给相对应迭代器的位置
//返回空数组：
return vector<int>(0);
```

链表

官方给的默认链表格式，自己也要会写

```
struct ListNode{
    int val;
    ListNode* next;
    ListNode(): val(0), next(nullptr){}
    ListNode(int v): val(v), next(nullptr){}
    ListNode(int v, ListNode* nex):val(v), next(nex){}
};
```

stack

```
stack<int> stk;
stk.push(1);
stk.empty();
stk.top();
stk.pop();
```

deque

```
#include<deque>

queue<int>q;

// 常用函数： front指队头，back指队尾（新插入的）
q.push_back(num);
q.push_front(num);
q.pop_back();
q.pop_front();
int num=q.front();
int num=q.back();
int n=q.size();
bool flag=q.empty();
```

multiset

```
#include<set>

multiset<int> mset;
mset.insert(3);//插入元素:O(log(n))
mset.insert(5);
mset.insert(1);
mset.insert(2);
mset.erase(2);//删除元素:O(log(n))，注意会删除所有等于改值的
mset.erase(mset.begin()+1);//删除某迭代器位置上的元素，即使有重复也只会删除一个
*mset.begin();//有序集合的首位元素（利用迭代器）
*mset.rbegin();//有序集合的末尾元素（利用迭代器）
//自定义排序：类似sort，不再重述
mset.find(3);//找某元素，找不到返回mset.end()
mset.count(3);//某元素数量
mset.upper_bound(5);//第一个大于5的元素的迭代器
mset.lower_bound(5);//第一个大于等于5的元素的迭代器
mset.count(5);//5的个数
```

priority_queue

```
# include<queue>
priority_queue<int> q;//默认大根堆
q.push();//插入元素
q.pop();//弹出元素
q.top();//返回堆顶元素
q.empty();//是否为空
q.size();//返回元素个数

//内部元素优先级设置：
priority_queue<int> q;//默认大根
priority_queue<int, vector<int>, less<int> > q;//大根
priority_queue<double, vector<double>, greater<double>> q;//小根
```

set & unordered_set

```
// 集合
set<int> s;

// 常用函数
int n=s.size();
s.insert(num);
s.erase(num);
s.count(num); // 常用来判断有无这一元素

// 遍历元素
set<int>::iterator it;
for(it = s1.begin(); it!=s1.end(); it++){ //自动排序元素
    cout<<*it<<endl;  // 这里的it是个指向数据的指针
}
//c++ 11
for(auto it : s){
```

```
    cout<<it<<endl;   // 这里的it就是数据本身
}
```

```
unordered_set<int> uset;
uset.insert(x);
uset.erase(x);//删去元素
uset.find(x);//找到返回迭代器，否则返回uset.end()
uset.count(x);//返回x的数目，0或者1
uset.size();//返回元素个数
uset.clear();//清空
uset.empty();//空返回1
//遍历：
unordered_set<int> set = {9,5,9,8,1,2,3,5,6,1,2,3,4,5,6,7,4,3,3};
for(unordered_set<int>::iterator it = set.begin();it!=set.end();it++)
{
    cout<<*it<<" ";
}
```

map & unordered_map

```
unordered_map<int, string> umap;//声明
//注意：直接对未初始化的umap[i]++，直接生成默认0再操作
umap.emplace(key,value);//向umap中添加键值对
umap[1]="aaa";//向umap中添加键值对
umap.find(key);//找到此key并返回其迭代器，否则返回umap.end()
//(注意是找到key不是找到其对应的value)
umap.count(key);//找到key返回1，否则0
umap.size();//返回哈希表大小
//遍历：
for(pair<int,string>key:umap)
{
    function(key.first,key.second);
}
//OR:
for(unordered_map::iterator it = map.begin();it!=map.end();it++){
    int front = it->first;//key
    int end = it->second;//value
}//（用迭代器进行遍历）
```

```
// map
map<string, int> m;

// 常用函数
int n=m.size();
m[key]=value; // 最简单的插入方法（个人认为）
m.erase(key);
s.count(key); // 常用来判断有无这一元素

// 遍历元素
map<string, int>::iterator it;
for (it = m.begin(); it != m.end(); it++) {
    cout<<it->first<<" "<<it->second<<endl;
}
```

```cpp
// c++ 11
for(auto it : m){
    cout<<it.first<<" "<<it.second<<endl;
}
```