提高算法学习效率——高效C++编程

原创 乔祥硕 LOA算法学习笔记 2021-12-08 16:25

进行算法设计时，我们最常用的编程语言就是C++。

但有些时候，即便我们头脑中已经有了一个明确且正确的算法，但可能受限于编程能力，不能很快地将这些算法实现。这样可能导致我们更多地关注了算法的实现上，花费大量的时间用在实现一个本可以简单调用函数实现的步骤上（比如最简单的排序，需要每次都手写一个快排吗？当我们的算法重点不在于排序时，更好的方式是直接使用sort函数），而不是重点关注算法本身。这是不利于算法学习的。由笔者的编程经验，C++中存在一些必须要掌握的库函数、STL等用法以及一些编程技巧，掌握了它们，可以帮助我们提高C++编程效率，从而促进我们算法的学习与进步。

下面对这些技巧进行简单的总结。分为四部分：头文件、STL、编程模板与编程技巧。

一、头文件

万能头文件，顾名思义，用了这个头文件就不再需要其他头文件了，简单粗暴。但前提是oj需要支持该头文件才能使用。虽然方便，但我还是不大推荐这个头文件，因为直接使用万能头文件，会让我们习惯性地忽略很多东西。

```
#include <bits/stdc++.h> //C++

//STL专用的头文件，使用了哪种数据结构，就要用到哪种头文件。比方说，使用map，就要写一个#include <map>。
#include <map> //C++
#include <vector> //C++
#include <set> //C++

//C++必备的头文件，cin、cout及其相关的函数都在这里。
#include <isotream> //C++

//strlen()、strcat()等字符串操作函数在这里。在C++中是#include <cstring>，在C语言中是#include <string.h>。
#include <cstring> //C++
#include <string.h> //C语言

//min()、max()在这里。注意C++和C语言中的形式不同。
#include <cstdlib> //C++
#include <stdlib.h> //C语言

//scanf()、printf()以及和它们长得像的函数在这里。虽然C++中直接使用cin、cout非常方便，但有些要求（比如固定几位小数点
#include <cstdio> //C++
#include <stdio.h> //C语言

//sort()在这里！sort函数是必须掌握的用法，在下文中会有详细的介绍。
#include <algorithm>> //C++

//log()、sin()、pow()等数学运算的函数在这里。
#include <cmath>> //C++
#include <math.h> //C语言

// INT_MAX、INT_MIN、LLONG_MAX在这里，这样当我们想要表示正无穷、负无穷时，直接用就可以了。
#include <climits>

//cctype：字符处理库，定义了有关字符判断与处理的库函数
#include <cctype>
```

二、STL

a) Vector

```
vector<int> v; //一维
vector<int> vt(100); //指定长度，适用于已知个数，可以直接cin>>vt[i]
vector<int> vt1(100,1); //指定长度并进行初始化，非常好用
vector<vector<int> > v2; //二维
vector<int> v22[100]; //二维，第一维已知，非常适合作为邻接表
v.push_back(1); //插入到最后
v[0]; //取v中的值
v.front(); //返回v数组第一个元素的值
v.back(); //返回v数组最后一个元素的值
v.pop_back(); //删除最后一个元素
int k=1,a=1;
v.erase(v.begin()+k);//删除下标为k的数，返回下一个位置的下标，必须用迭代器
v.insert(v.begin()+k,a); //将数字a插入到指定位置k
v.insert(v.begin()+k,5,a); //将5个数字a插入到指定位置k
sort(v.begin(), v.end()); //快排
sort(v.begin(),v.end(),greater<int>());//降序快排
v.erase(unique(v.begin() , v.end()), v.end()); //去重，在此之前要先排序
reverse(begin(v), end(v)); //实现逆序，原地更改
is_sorted(v.begin(), v.end()); //是否非递减
v.resize(10); //重新指定大小，适用于v先定义为全局变量，main中通过输入才得知大小
cout<<accumulate(v.begin(), v.end(), 0); //从0开始计算总和，numeric头文件
```

b) Stack

```
stack<int> s;
s.push(1); //入栈
cout<<s.top()<<endl;//取栈顶元素，不出栈
s.pop();//出栈，不返回值
```

c) Queue

```
queue<int> q;
q.push(1); //入队
q.pop(); //出队，不返回值
q.front(); //取队头元素，不出队
q.back(); //取队尾元素，不出队
```

d) priority_queue

```
priority_queue<int> maxQ;//大顶堆
priority_queue<int,vector<int>,greater<int> > minQ;//小顶堆
maxQ.push(1);//入堆
cout<<maxQ.top()<<endl;//取堆顶元素，不出堆
maxQ.pop();//出堆，不返回值
```

e) map

```
map<int,int> m;
m.insert(pair<int,int>(1,2));//字典的插入方法1
m[2]=3;//字典的插入方法2，建议这样用
map<int,int>::iterator iter=m.find(1);//在字典中查找某一元素
if(iter!=m.end()) {
    cout<<iter->second;//找到了
}
m.erase(iter);//删除
for(auto iter=m.begin(); iter!=m.end(); iter++) //遍历，iter用auto类型
    cout<<iter->first<<' '<<iter->second<<endl; //获取键first-值second
```

f) unordered_map

```
unordered_map<int, int> um;
um[0]=1; //插入
if(um[2]) { //查找
    cout<<um[2];
}
```

g) string

```
string str2="abc";
str2+="d";//直接用加号进行字符串连接
str2.substr(2,1);//取子串，两个参数分别为：start,length；不写length就是到结束
str2.find("ab",1); //从下标1开始，查找子串"ab"的位置，没找到则为string::npos
str2.length(); //字符串长度
str2.size() ; //字符串的长度
str2.c_str(); //转换为C语言中的char[]
sort(str2.begin(), str2.end()); //字符串sort排序
//字符串转换成其他格式：#include<string> stoi() stod() stof() stol() stold() stoll() stoul() stoull()
//其他格式转换成字符串：to_string()
str2.erase(str2.begin()+1); //移除指定位置的字符
str2.insert(0,10,'0'); //在字符串指定位置增加10个'0'字符，不需要迭代器
str2.erase(0,3); //删除下标0开始的共3个字符
str2.pop_back(); //删除最后一个字符
```

h) list

```
list<int> l;
l.push_back(1);
l.push_back(5); //插入到最后
l.push_front(0); //插入到最前
list<int>::iterator iter2;//也可以用auto，注意iter是指针可以++
for(iter2 = l.begin(); iter2 != l.end(); iter2++) {
    cout <<*iter2 << endl; //遍历
}
l.insert(iter2,3); //在指定位置插入
l.erase(iter2); //在指定位置删除
l.sort(); //排序
```

i) set

```
set<int> set1 {0,1,2}; //可以在定义时赋初值
set1.insert(3); //向集合中插入数据
set1.size();  //获取集合大小
set1.erase(3); //按值删除
set1.erase(set1.begin()) ; //按位置删除
set1.find(5)!=set1.end(); //查找某一元素是否在集合中
for(auto it=set1.begin(); it!=set1.end(); it++) {
    cout<<*it<<endl; //获取每一个元素值
}
set1.clear(); //清空
```

三、常用编程模板

a) 判断某一数字是否为素数（最简单的判断方式）：

```
bool isPrime(int n) {
    if(n<=1)
        return false;
```

```
    for(int i=2; i<=sqrt(n); i++)
        if (n % i == 0) return false;
    return true;
}
```

b) 求两个数的最大公约数：

```
int gcd(int x,int y) {
    return y==0?x:gcd(y,x%y);
}
```

c) 快速进行素数打表：

```
vector<int> primeTable(int n) {
    vector<int> isprime(n, 1); //初始化为n个1
    for (int i=2; i*i<n; i++) //外层循环i从2到根号n
        for (int j=2; j*i<n; j++) //内层循环j从2开始到i*j<n
            isprime[i*j]=0; //把i的j倍都标记为0
    return isprime;
}
```

d) 比较函数：

```
bool cmp(const Student &a, const Student &b) {
    if(a.pno!=b.pno)
        return a.pno<b.pno;
    return a.score>=b.score;
}
```

一般与sort函数共同使用，注意这里写成引用形式会更快。

e) 并查集：

```
int fa[100]; //定义为全局变量，初始化为自己

// 查找父亲，含路径压缩
int findFather(int x) {
    int t = x;
    while(x != fa[x])
        x = fa[x];
    //路径压缩，直接将路径上所有元素的父亲改为最终父亲
    while(t != fa[t]) {
        int z = t;
        t = fa[t];
        fa[z] = x;
    }
    return x;
}

// 合并，即建立它们父亲间的关系
void Union(int a, int b) {
    int faA = findFather(a);
    int faB = findFather(b);
    if(faA != faB) fa[faA] = faB;
}
```

f) Sort函数

```
sort(a,a+10,cmp);
sort(arr.begin(),arr.end(),greater<int>());
sort(arr.begin(),arr.end(),less<int>());
```

四、编程技巧

a) 解决爆栈问题，手动加栈，必须放在头文件之前：

```
#pragma comment(linker,"/STACK:1024000000,1024000000")
```

b) 可以使用typedef定义long long的缩写为ll：

```
typedef long long ll;
```

这样在用到long long时，直接使用ll代替，节省时间。

c) 学会使用pair：

```
typedef pair<string, int> psi;
```

当一个数组或者vector中只有两个元素时，使用pair最为简单。pair的定义可以进一步简化，比如我们可以用long long简化为ll相同的方式，把string与int组成的pair简写为psi，这样每次使用的时候相当方便。

d) 运算符重载：

```
class Student {
    public:
        Student(int p) {
            this->pno=p;
        }
        int pno;
        int score;
        //运算符重载，从而可以sort排序
        bool operator < (const Student& s) const {
            return this->pno<s.pno;
        }
};
```

运算符重载的技巧可以与sort排序相结合，从而实现直接对类进行sort排序。

e) 提高cin与cout的速度：

```
ios::sync_with_stdio(false);
```

不过可惜的是，这样还是不如scanf快，因此如果时间要求苛刻的话，还是老老实实用scanf和printf吧！

f) C++中的输入方法总结：

```
char ch=cin.get();//读入单个字符
string str;
getline(cin,str);//#include<string>，读入一行，可以读入空格
char cha[100];
cin.getline(cha,100); //#include <iostream>，读入一行，可以读入空格
gets(cha);  //可以读入空格
```

g) 关于复制粘贴：

```
freopen("1.txt","r",stdin);
freopen("2.txt","w",stdout);
```

某些oj无法直接复制输入样例，或者某些终端没法直接粘贴，这样的话，可以将C++标准IO流变为文件IO流，但是用完记得删掉，不然就wa了！

h) 不确定输入个数时的读入方法：

```
while(cin>>n) {}
```