✓ 常用芝士

1、C++基本语法

（1）、输入输出、命名空间

```
#include<iostream>
std::cin >> a >> b;
std::cout << a << b;

using namespace std;
cin >> a >> b;
cout << a << b;

ios::sync_with_stdio(false), cin.tie(0),
cout.tie(0);
//关闭同步流，一旦使用cin、cout尽量关闭同步流，节省时间
//在codeforces等平台中，关闭同步流时不能混用C（scanf
等）和C++（cin等）的IO

while(cin >> a)      //多组输入
    //

getline(cin, s);     //读行

freopen("out.txt", "r", stdin);      //C语言文件读写
```

（2）、C++新的基本数据类型：bool

```
bool flag = true;
flag = false;
```

(3)、类和结构体

在下一个部分中，将会有大量的C++帮你写好的类，类相当于结构体，把很多相关的变量集合在一起

在C++中，结构体的定义和声明更加方便，并且在结构体中可以写成员函数，通过定义的实例对象来调用

```
struct node
{
    int min;
    int sec;
    int getseconds()        //成员函数
    {
        return min * 60 + sec;
    }
}a;
int main()
{
    node b;     //a、b就是实例对象
    a.min = 1;
    a.sec = 19;
    cout << a.getseconds();
}
```

（4）、引用

引用就相当于给一个变量/对象起了另一个名字，也就是说，通过引用我们可以得到变量/对象的值，并且可以修改

对于比较复杂的类型，如果直接拷贝赋值的话会消耗很多时间，引用就可以省去拷贝

引用类型常用于传参或者使用容器时使用

（5）、auto

auto可以在声明变量的时候根据变量初始值的类型自动为此变量选择匹配的类型，例如：

```
int a = 1;
auto b = a;      //定义b为int类型，并将a赋值给b
```

通常我们再写迭代器等复杂的数据类型时会使用auto来提高书写速度

```
map<int, int>mp;
//map<int, int>::iterator it1 = mp.end();
auto it2 = mp.end();
auto& it2 = mp.begin();
```

（6）、默认初始化

C++对于所有自带的数据类型，都有设定默认初始化，例如int类型默认初始化为0，string类型默认初始化为空串

对于开在全局的基本数据类型（int，bool等）都使用默认初始化，但局部的不会，需要手动初始化、赋值

```
int a[maxn];    //初始化全为0
int main()
{
    int cnt;    //没有初始化
    bool flag = true;   //初始化为true
}
```

（7）、自定义排序、比较函数

内置快速排序函数：`sort`，默认按照数组大小从小到大排序

- 对于自定义普通数组的排序：

```
int cmp(int x1, int x2)
{
    return x1 > x2;
}

sort(a, a + n, cmp);    //从大到小排序

int cmp(int i1, int i2)
{
    return b[i1] < b[i2];
}

sort(a, a + n, cmp);    //对a数组按照b数组大小进行排序
```

- 对于自定义结构体的排序：

可以选择自定义比较函数（可以写Lambda函数）、或者重载小于比较运算符（sort内部靠<实现比较）：

如果有没有定义排序的维度？不同编译器结果不同！

```
int cmp(node a, node b)
{
    if (a.r != b.r)
        return a.r < b.r;
    else
        return a.l < b.r;
}
sort(q + 1, q + 1 + n, cmp);
```

```
struct node
{
    int l, r;
    friend bool operator<(const node& a, const
node& b) //friend声明友元函数
    {
        if (a.r != b.r)
            return a.r < b.r;
        else
            return a.l < b.r;
    }
    //bool operator<(const node& b) const
    //{
    //  if (r != b.r)
    //      return r < b.r;
    //  else
    //      return l < b.r;
    //}
}q[maxn];
sort(q + 1, q + 1 + n);
```

- C++内置比较函数

C++内置 `less<Type>` 从小到大排序， `greater<Type>` 从大到小排序

```
sort(a + 1, a + 1 + n, greater<int>());     //将a数组中的数从大到小排序
```

2、命名、习惯

在对变量命名时，我们常使用英文全拼的部分简写，例如answer（答案）记为ans，infinity（无穷大）记为inf，count（次数）记为cnt，这样既可以在明确变量含义的同时节约时间和码量，同时免去了与关键字重复的可能

不规范示例：

```
int a, aa, b, bb;       //容易用错，并且不好debug

int xueliang, gongjili;     //中文?
```

常用模板（这里提供一种模板，也可以向其他学长请教）：

```
#include<bits/stdc++.h>
typedef long long ll;
const ll maxn = 5e3 + 7;    //一般开到题目的数据范围，然后多开一点
const ll maxm = 2e6 + 7;
constexpr ll mod = 1e9 + 7;          //模数
const ll inf = 0x3f3f3f3f;           //代表int范围内无穷大
const ll INF = 0x3f3f3f3f3f3f3f3f;  //代表ll范围内无穷大
const double eps = 1e - 10;          //精度
using namespace std;
int a[maxn];        //使用常变量定义数组大小
void solve()
{
    //是否压行
    //对于关键性代码使用注释
}
signed main()
{
```

```
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);     //关闭同步流
    int t = 1;
    //cin >> t;      //题目是否t组样例考虑是否注释
    while (t--)
    {
        solve();
    }
    return 0;
}
```

3、时间复杂度、时空概念

TLE or MLE？算好复杂度不存在的

在XCPC中，每道题都会限定时间和空间范围，据此我们可以选择我们的做法

空间

1MB = 1024KB = (1024 * 1024)B

int -> 4B -> (-1e9~1e9)

long long -> 8B -> (-1e18~1e18)

char,bool ->1B

大部分题不会刻意卡空间

时间

在算法竞赛上，我们可以认为测评机一秒可以跑1e8数量级的程序，不同的测评机、不同的语言版本效率不同、相同代码也可能跑出不同的时间

我们用n来代表数据规模的大小（注意t组样例的情况），则我们有：

当$n = 1e5 - 1e6$时我们考虑$O(n)$或$O(nlogn)$算法：

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

当$n = 1e3 - 5e3$时我们考虑$O(n^2)$算法：

```
for (int i = 1; i <= n; i++)
{
    for (int j = i; j <= n; j++)
    {
        //
    }
}
```

当$n = 1e9$或者更大数量级时，我们考虑$O(1), O(logn), O(\sqrt{n})$的复杂度：

```
for (ll i = 1; i * i <= n; i++)
{
    //
}
```

当$n = 20$左右时我们可以考虑$O(2^n)$的算法($2^{20} \approx 1e6$)：

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

一般情况下我们认为：

$O(n!) > O(2^n) > O(n^3) > O(n^2) > O(nlogn) > O(n) > O(logn) > O(1)$

更多情况下，一道题目的时间复杂度是非常复杂的，一个程序可能有多个模块组成，这就要求我们综合分析代码的时间复杂度，例如：$O(n*2+n^2)$我们通常认为是$O(n^2)$的复杂度，因为对于n^2来说，n根本不是一个数量级的，可以直接忽略

同时还要考虑常数的影响，例如：$O(20*n^2)$我们可以看作$O(n^2)$，当$n=1e3$时，复杂度有$2e7$可以接受，但当$n=5e3$时复杂度达到$5e8$可能会导致TLE

C++一些内置的函数、STL容器和一些不漂亮的写法也可能导致我们的常数比较大

学习一个算法不仅要学习算法的过程，更要学会这个算法的时间复杂度如何计算

例如，内置的排序算法$sort$的时间复杂度为$O(nlogn)$；前缀和的预处理时间$O(n)$，询问区间$O(1)$

总体来说分析时间复杂度是我们写代码之前必须要做的事情之一

✓ 终极STL

·· 1、C++的优势

丰富的函数库、模板库

兼容C语言，是C语言的拓展

相较于python、java更加通用、方便（涉及大数运算可以考虑python）

·· 2、容器、迭代器

··· （1）、基本概念

- 什么是容器?

可以理解为存储元素（数据）的一种结构，数据可以是int类型，也可能是结构体，也可以是另外的容器，这也就要求我们在定义一个容器的时候同时指明这个容器里面装的是什么东西（一个容器不能装载不同的数据类型）

C++用类来实现了各种各样的容器，我们可以通过调用类的成员函数来对容器进行操作，头文件就是对应名字，例如引入vector头文件即 `#include<vector>`

- 什么是迭代器?

容器里有很多元素，为了遍历、访问这些元素，我们可以使用迭代器

迭代器就类似于指针，我们可以让他指向该容器的某一元素

- 一般的成员函数:

下面会讲述C++基本所有容器都会共有的一些成员函数（下用container代表容器）如果需要更多函数可以自行百度，下面会列出常用的一些成员函数

```
container::iterator it;      //该容器的迭代器类型
*it                 //解引用符号，获取该迭代器内元素是什么，即如果是int返回int类型

container.size()              //返回该容器的大小，注意这个数是无符号类型
container.empty()             //返回该容器是否空

container.erase()             //传参一个迭代器，删除迭代器指定的元素；传参两个迭代器，删去两个迭代器之间的部分（左闭右开）
container.clear()             //删除容器内全部元素

container1 == container2          //所有容器都支持==、!=运算符O(size)

container.begin()         //返回容器的头迭代器
container.end()           //返回容器最后一个元素的下个元素的迭代器
```

··· （2）、有序容器

顺序容器顾名思义就是按顺序存储元素的容器

1>、vector和string

`vector` 可以理解为数组，我们可以快速地在数组末尾添加元素，也可以利用下标快速访问第i个元素

可以简单的理解为： `vector<char>` 等价于 `string` ，即vector里面存储字符就是字符串

```
vector<int>a;         //定义一个空的数组
vector<int>a(n);      //定义一个有n个元素的数组，默认初始化值为0
vector<int>a[100];    //开了一百个空的数组
```

```
a.insert(a.begin() + k, val);        //在第k个位置插
入元素val，O(size-k)
a.push_back(val);    //将元素val放入数组a的末尾，O(1)
a.pop_back();        //删去尾部元素


for (int i = 0; i < a.size(); i++)  //三种遍历方式
    //a[i]
for (auto val : a)
    //val
for (auto it = a.begin(); it != a.end(); ++it)
    //*it
```

> 利用已有知识，如何存图？
>
> 什么是图？什么是树？
>
> https://csacademy.com/app/graph_editor/

string也有多于vector的一些字符串操作函数

```
string s = "";       //定义一个空串

string sub1 = s.substr(3);           //表示从下标为3
的位置取后面所有字符
string sub2 = s.substr(3,6);         //表示从下标为3
的位置取后面6个字符

s.find(args);                 //查找s中args第一次出现
的位置（下标），没有返回-1

s1 < s2;          //两个字符串是可以按照字典序比较的

int nums = to_string(val);
int val = stoi(s);       //字符串转化为数值，另外还有
stoll, stod
```

2>、deque

`deque` 名为双端队列，我们暂时把它类比为vector（支持下标访问），不同的是我们可以更快地从头部删除和添加元素

```
deque<int>q;
q.push_back();
q.pop_back();
q.push_front();
q.pop_front();
```

3>、array

`array` 是相对于vector的定长数组，即定义之后数组长度不能改变，用的不多，需要的时候自行学习

4>、list、forward_list

`list` 可以快速$O(1)$在中间插入元素，用的也不多，需要的时候自行学习

··· （3）、容器适配器

容器适配器是指基于顺序容器实现的一些数据结构

`stack` 和 `queue` 是基于deque实现的， `priority_queue` 是基于vector实现的

1>、queue

什么是队列？先进先出、后进后出

常用函数：

```
queue<int>q;                //建立一个空队列
q.front(),q.back();     //返回首元素或尾元素，只适用于queue
q.push(val);              //在队列末尾添加元素
q.pop();                    //删除队列首元素
```

2>、stack

什么是栈？先进后出、后进后出

常用函数：

```
stack<int>s;              //建立一个空栈
s.top();                    //返回栈顶元素
s.push(val);              //创建一个新元素压入栈顶
s.pop();                    //删除栈顶元素，但不返回该元素值
```

3>、priority_queue

优先队列默认大根堆，即队列首一定是所有在优先队列的元素中最大的元素

常用函数：

```
priority_queue <int>q;
q.top();                    //返回最高优先级的元素
q.push(val);                //将val元素放入优先队列
q.pop();                    //优先队列的最高优先级的元素
```

如果我想按照从小到大取出元素：

```
priority_queue<int, vector<int>, greater<int>>q;
//即将小于号重载为大于号
```

塞入结构体？

（4）、关联容器

C++内置 `pair` 结构体，相当于自定义的有两个元素的一个结构体（已经重载了小于号）

```
pair<int, int>a;
a.first, a.second;          //访问pair的第一元素、第二元素
make_pair(val1, val2);      //以val1、val2作为第一、二元素建立pair
```

关联容器相对于顺序容器，我们更关注元素的值而不是元素加入的顺序

1>、set、map

`set` 和 `map` 可以理解为一个放入元素自动排序、去重的容器。当然是有时间复杂度的代价的，只要使用自带O(logn)

与之前的顺序容器一样，我们可以决定set中要放什么数据类型 `<type>` ，我们按照这个数据类型进行从小到大排序（默认）

map类似于set，不过map中塞的是 `pair<type1,type2>` ，我们称这个为键值对，第一个元素type1叫关键字（键），第二个元素type2叫值，在该容器内我们按照关键字从小到大排序（默认）

```
set<int>st;                 //建立一个空的set容器
map<string, int>mp;       //建立一个空的map容器

st.insert(val);           //set容器中插入val元素
mp.insert(make_pair(val1, val2));    //mp中插入{val1, val2}这个键值对
mp.insert({val1, val2});

mp[val];        //返回关键字val对应的值的引用（即可以知道这个值是多少、也可以修改这个值），如果map中没有这个键值对，则会新建立一个默认初始化的值

st.erase(it); //传参it迭代器，表示在set中删除it所指的元素，map同理但用的不多

c.clear();     //初始化，将容器中元素全部删去

c.find(val);  //返回一个迭代器，指向第一个关键字为val的元素,若不存在返回尾后迭代器
c.count(val); //返回关键字为val的元素的数量，对于不重复关键字的容器返回值为0或1


c.lower_bound(val); //返回一个迭代器，指向第一个关键字不小于val的元素
```

```
c.upper_bound(val); //返回一个迭代器，指向第一个关键字大于val的元素

*it      //解引用迭代器，获得元素内容
++it, --it;          //得到将迭代器指向下/上一个元素
//非顺序容器，不支持迭代器相减;

c.begin();       //获得第一个迭代器，如果按默认排序则是最小元素
c.rbegin();      //获得第一个反向迭代器，即最后一个元素，如果按默认排序则是最大元素
```

2>、multi-、unordered-

`multiset` 和 `multimap` 代表着放入的关键字是可以重复的，但是依旧排序

`unordered_set` 和 `unordered_map` 代表着放入的关键字是不排序（基于哈希实现）的，但是去重

一般常用的是multiset和unordered_map

> map：基于红黑树，复杂度O(logn)
>
> unordered_map：基于哈希表，复杂度依赖于散列函数产生的冲突多少，多数情况下其复杂度接近于O(1)，但稳定性差，可能被卡成O(n)
>
> https://blog.csdn.net/weixin_52093215/article/details/121055519

此外还有 `unordered_multiset` 和 `unordered_multiset` ，很少用

3、算法

主要头文件： `algorithm`

函数兼容 `数组` 和 `顺序容器` （如果想知道为什么可以兼容，可以学习C++重载函数）

```
count(a, a + n, x);
count(a.begin(), a.end(), x);
//返回在范围内寻找x的出现次数

//以下用beg代表范围起始位置、end代表范围结束位置、val代表值、cmp代表比较函数

lower_bound(beg, end, val, cmp);
//返回一个地址/迭代器，代表在一个有序范围内查找第一个大于等于val的元素，如果不存在这样的元素返回最后一个元素下一个元素（地址、迭代器）、cmp可以自定义
upper_bound(beg, end, val, cmp);
//返回第一个大于val的元素

fill(beg, end, val);
//范围内每一个赋值val

sort(beg, end, cmp);
//排序函数，见上
is_sorted(beg, end, cmp);
//返回bool值，判断是否排序

unique(beg, end);
//对于相邻的重复元素，通过覆盖来实现去重，因此通常结合sort

reverse(beg, end);
//反转区间

next_permutation(beg, end, cmp);
//如果序列已经是最后一个排列、返回false，否则返回下一个排列结果，通常用于枚举情况

max(val1, val2);
min(val1, val2);
min_element(beg, end, comp);
max_element(beg, end, comp);
//查找范围内最小/大的元素

swap(val1,val2);    //交换两个元素，O(1)
```