1. 优化与文件头

1.0. 基础模板

Description: 一个基本的代码模板，带有典型的 t 次循环，最大数组上限定义，int 替换为 long long 等竞赛基本属性。 本地使用文件读写测试时，在文件头加一行#define localDebug 即可。“总是那么好用，硬要说的话，就像板烧鸡腿堡。”

```
#include <bits/stdc++.h>
#define int long long
const int MaxN = 1e7 + 10;

signed main(){
 #ifdef localDebug
    freopen("file.in","r",stdin);
    freopen("file.out","w",stdout);
 #endif
    int t;
    scanf("%lld",&t);
    while(t--){

 }
}
```

1.1. 读入优化

一般来说，cin 大幅慢于 scanf，可以使用 ios::sync_with_stdio(false);关闭流同步进行优化，会略快于 scanf。 部分题目使用 scanf("%lld")读入10^{18}的数据会 WA，cin 或者%I64d 可以解决。

Description: 典型的兼容多类型的快速读入，兼容空格回车。“Tasty and crispy!”

```
template <class T>
inline bool fscan(T &vari){
 char ch;
 int sgn;
 if(ch=getchar(),ch==EOF) return false; //检测到EOF，返回False
 while((ch!='-')&&((ch<'0')||(ch>'9'))) ch=getchar(); //跳过非法字符
 sgn=(ch=='-')?-1:1; //符号处理
 vari=(ch=='-')?0:(ch-'0');
 while(ch=getchar(),((ch>='0')&&(ch<='9'))) vari=vari*10+ch-'0'; //数字处理
 vari*=sgn;
 return true;
}
```

1.2. STL 容器

简单来说，就是料理包。不可口但是方便，有时需要一定的优化才能把时间控制在比较合理的范围内。

1.2.1. string

因为存在 string 与 C-string 两个相似且都广泛使用的库，在此进行比较。

	string	C-string
定义	string str;	char str[LENGTH]
长度	str.size(); str.length();	strlen(str);
连接	str.push_back(char); str.append(str); str=str1+str2;	strcat(str1,str2);
拷贝	str=str1	strcpy(target,standard);
比较	str1>str2str1<str2	strcmp(str1,str2);//如果 str1>str2,返回正数，相等返回 0，否则返回负数。
插入	str.insert(pos,char)	
定位	str.find(string, pos); \\从左查找 str.rfind(string, pos); \\从右查找 \\如果找不到将会返回-1	strchr(str,string);
替换	str.replace(pos,n,string);\\把当前字符串从索引 pos 开始的 n 个字符替换为 string	
删除	str.erase(pos,len);	

RELATED: tolower(char); \\转换为小写 toupper(char); \\转换为大写
TIPS: string str2("123456789"); //生成"1234456789"的复制品 string str3("12345", 0, 3); //结果为"123" ，从 0 位置开始，长度为 3 string str4("123456", 5); //结果为"12345" ，长度为 5 string str5(5, '2'); //结果为"22222" ,构造 5 个字符'2'连接而成的字符串 string str6(str2, 2); //结果为"3456789"，截取第三个元素（2 对应第三位）到最后

1.2.2. vector

可以简单的认为，向量是一个能够存放任意类型的动态数组。vector 是一块连续分配的内存，随着元素的不断插入，按每次一倍的机制扩充空间。 每次容器的增长，==并不是在原有连续的内存空间后再进行简单的叠加， 而是重新申请一块更大的新内存，并把现有容器中的元素逐个复制过去，然后销毁旧的内存==。 这时原有指向旧内存空间的迭代器已经失效，所以当操作容器时，迭代器要及时更新。

操作	语句
定义	vector<Type> v;
大小	v.size();
尾部插入	v.push_back();
尾部删除	v.pop_back();
头元素迭代器	v.begin();
尾元素迭代器	v.end();
插入	v.insert(pos,element);
擦除	v.erase(pos);v.erase(begin, end);

WARNING:

1.vector 自带了 insert()和 erase()操作来在 vector 中任意位置来插入元素，但是离尾部越远效率越低，所以需要频繁的在元素中任意位置插入删除元素可以选用其它的容器，比如 list。 2.通过下标访问时不要越界访问！ 3.pushback 有可能导致迭代器失效！ **TIPS:** 双层嵌套 vector（如 vector<vector<int>> v;）可以通过访问二维数组的方式直接访问。也可以直接使用 vector 数组

1.2.3. set

set 的原型是数学中的集合。set 作为一个容器也是用来存储同一数据类型的数据类型，并且能从一个数据集合中取出数据，在 set 中每个元素的值都唯一，而且系统能根据元素的值自动进行排序。 应该注意的是 set 中元素实际为常量。

操作	语句
定义	set<int> s; //升序 set<int,greater<int>> s; //降序
插入	s.insert(element); //单点插入 s.insert(iter.begin(), iter.end()); //指针迭代器插入
判空	s.empty(); //若空返回 True，否则返回 False
计数	s.count(); //返回个数，不存在返回 0
大小	s.size();
头元素迭代器	s.begin();
尾元素迭代器	s.end();

RELATED: cout<<*s.lower_bound(value)<<endl; //输出集合中第一个大于 value 的值 cout<<*s.upper_bound(value)<<endl; //输出集合中第一个大于等于 value 的值 **TIPS:** BTW，map 和 set 的插入删除效率都比其他容器要更高,复杂度$O(log\mathrm{N})$。 使用 unordered_set 和 unordered_map 可以

提高效率,复杂度$O(1)$,对于字符串复杂度为$O(len)$。 使用 muiltimap 和 multiset 可以存储相同元素。 同理，使用无序关联的容器也可以提高 muiltimap 和 multiset 的速度，分别为 unordered_multimap 和 unordered_multiset。 如果需要反向遍历容器，可以使用 rbegin()和 rend()。 —

1.2.4. map

使用 map，可以很方便的构建映射关系，它提供一对一的数据处理能力（有序键值对），第一个元素称为关键字，第二个称为关键字的值，其中关键字是唯一的。 map 是一类关联式容器。它的特点是增加和删除节点对迭代器的影响很小，除了那个操作节点，对其他的节点都没有什么影响。

对于迭代器来说，可以修改实值，而不能修改 key。

操作	语句
定义	map<type,type> mp;
插入	mp[key]=valuemp.insert(pair<type,type>(key,value)); mp.insert(make_pair(key,value));
头元素迭代器	mp.begin();
尾元素迭代器	mp.end();

1.3. STL 函数

1.3.1. swap

TIPS: 将两个容器的内容互换。

```
//原型如此
template <class T>
void swap(T& a, T& b)
{
    T tmp = a;
    a = b;
    b = tmp;
}
```

1.3.2. sort

TIPS: 对容器进行排序。在头文件 algorithm 内省排序和插入排序的结合，内省排序又是快速排序和堆排序的结合， 根据数据量决定具体每一步使用什么算法。由于使用泛型，并不是非常快。 默认使用 less<T>进行升序排序。 兼容迭代器

```
sort(begin,end,comp);//注意：区间左闭右开
```

1.3.3. reverse

TIPS: 反转容器内容。 用法与 sort 相同。

1.3.4. unique

TIPS: 去除容器内重复元素。 返回一个指针，指向去重后最后一个元素。 unique(begin,end);

1.3.5. lower_bound

TIPS: 返回第一个大于等于 value 的元素的迭代器。 lower_bound(begin,end,value); 支持使用迭代器

1.3.6. upper_bound

TIPS: 返回第一个大于 value 的元素的迭代器。 返回和 upper_bound 一样

1.3.7. equal_range

TIPS: 返回第一个大于等于 value 的元素的迭代器和第一个大于 value 的元素的迭代器。 返回值为迭代器对。 原型为 pair<iterator,iterator> equal_range(const T& value);

1.3.8. binary_search

TIPS: 判断元素是否在容器中。 使用二分查找，返回一个 bool 值。 binary_search(begin,end,value); 同样支持迭代器

1.3.9. merge

TIPS: 将两个容器合并成一个容器。 将两个迭代器区间合并到第三个容器

```
vector<T>v1;
vector<T>v2;
vector<T>v3;
merge(v1.begin(),v1.end(),v2.begin(),v2.end(),v3.begin());
```

1.3.10. GCD

TIPS: 计算两个数的最大公约数。 **使用：** __gcd(a,b);(*注意前面的是两个下划线*)或者 gcd(a,b);

1.3.11. fill

TIPS: 将容器内元素全部赋值为 value。 fill(begin,end,value);

1.3.12. Max & Min

TIPS: 返回容器内最大值和最小值。 原型：template <class T> T max(T a,T b); template <class T> T min(T a,T b);

1.3.13. next_permutation

TIPS: 全排列,每调用一次，返回下一个排列。 下一个排列字典序增大 使用：`next_permutation(begin,end);` next 换成 prev，增大字典序减小

1.3.14. bitset

TIPS: 将整数转换成二进制的字符串。 状态压缩的有效方法

```
bitset<8> bs(0xFF);
bitset<9>bs(string("101010101"));
```

1.3.15. 字符串与整数

TIPS: 将整数转换成字符串。`to_int` 相反 `s.to_string(); stoi(s);` `s.to_int(); stoi(s);`

1.4. 打表

1.4.1. 打表

TIPS: 有时有些题数据范围小或者有潜在规律，这是可以在本地用穷举法找到问题的解。 禹姐姐：我能$O(1)$的解决所有问题，你说NP类问题？那只是没有找到好的算法而已

1.4.2. 对拍

TIPS: 如果找到了一个问题的不确定算法，可以使用对拍程序，将新算法与暴力程序的输入输出做对比，看看是否有差异。以验证新算法在一定数据量内是否正确,还可以快速找到 hack case

```
@echo off
set path=C:\MinGW\bin
g++ -o hash.exe hash.cpp
:loop
set path=C:\Users\hp\AppData\Local\Programs\Python\Python39
python aa.py >data.in
hash.exe <data.in >hash.out
python bb.py <data.in >py.out
set path=C:\Windows\System32
fc py.out hash.out
if errorlevel == 1 pause
goto loop
```

```
#!bin/bash
while true; do
    gcc hash.cpp -o hash.exe
    gcc sort.cpp -o sort.exe
    gcc makedata.cpp -o makedata.exe
    ./makedata.exe >data.in
    ./hash.exe <data.in >hash.out
```

```
    ./sort.exe <data.in >sort.out
    if diff hash.out sort.out; then
        echo OK
    else
        echo wrong
        break
    fi
done
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    system("g++ in.cpp -O2 -o text.exe");
    system("g++ std.cpp -O2 -o std.exe");
    for(int i=1;i<=10000;i++)
    {
        system("python data_maker.py>data.txt");
        system("test.exe<data.txt>tt.txt");
        double st=clock();
        system("std.exe<data.txt>std.txt");
        double ed=clock();
        if(system("fc tt.txt std.txt")) {printf("WA\n");break;}
        else printf("AC #%d Time:%.3lfms\n",i,ed-st);
    }
    return 0;
}
```

1.5. Python

> 如果让我归类的话，对于传统语言竞赛选手来说，Python 应该归为”乱搞“的范畴。Python 的显著优势在于自带高精度，以及它方便的无视类型特性，当然还有字典、列表。在比赛的时候可能会有奇效。但是需要注意的是，通常来说 Python 的运行效率会比较低，需要注意 Python 是否有专门的时间限制。“麻辣香锅，吃多了容易闹肚子，太久不吃又馋。”某人补充：Python 一时爽，一直 Python 一直爽。

Python 是弱类型的语言。他的类型会由它的赋值而改变。当然，你也可以使用经典的`(type)`进行强制类型转换。Python 并不需要类似 C 的大括号来标识代码块的开始与结束，以 `tab` 即四个空格为单位**严格划分每一层代码块**。

1.5.1 基础

操作	语句
输入	`a,b=map(type,input().split(char))`
输出	`print(value)`
函数定义	`VARIBLE=value`

1.5.2 函数

```
def func(a,b, key=0,*args,**kwargs):
 print(a,b,args,kwargs)
 # a 和 b 是必须的参数，args 和 kwargs 是可选的参数
 # arges 为元组，kwargs 为字典
 # 如果没有 key，默认为 0
 return a+b
```

1.5.3 列表

操作	语句
定义	li=[value,value,...]
插入	li.append(value) #尾部插入元素 li.insert(pos,value) #指定位置插入元素
访问	li[pos] #访问第 pos+1 个元素 li[start:end] #截取列表
检测存在	value in li #存在返回 True，不存在返回 False

```
li=[1,2,3,4,5]
li.append(6)  # 在列表末尾添加元素
li.insert(0,0)  # 在列表指定位置插入元素
li.pop()   # 删除列表末尾元素
li.pop(0)   # 删除列表指定位置元素
li.remove(1)  # 删除列表中指定元素
li.reverse()  # 列表反转
li.sort()   # 列表排序
li.sort(reverse=True) # 列表排序，反转
```

1.5.4 字典

```
di={1:1,2:2,3:3}
di[1]   # 获取字典中指定元素
di.get(1)  # 获取字典中指定元素
di.pop(1)  # 删除字典中指定元素
di.popitem() # 删除字典中随机元素
di.clear()  # 清空字典
di.setdefault(1,1) # 如果字典中不存在指定元素，则添加该元素
```

1.6. 其他

```
# 一种输入方式
[i for i in sys.stdin]

# 表达式
[i for i in range(1,10) if i%2==0] # 迭代器表达式
[i if i%2==0 else 0] # 类似于三元运算符
```