## 数组的操作：

Array.reverse() 可以调换顺序。

Sizeof(arr)/sizeof(int) 可以返回大小。

Array 通过[]直接进行操作。

使用vector而不使用array就是因为在函数中要获取array的数值，需要使用的是\*。解除引用运算符。

## 位操作符：

通过>>=来确定移动的位置

## Vector操作

通过size()返回数组的大小

通过创建vector<int>::iterator VecIte来创建索引器，并且赋值为vector.begin()和vector.end(),同时，vecIterator具有一个+1，类似函数指针，可以改变自己的当前指向位置。

具有.begin和.end操作，然后具有[]取值的功能。

注意：如果不是循环体，在外边不要随意使用i++和i--，尤其是在递归迭代函数中。因为你的本意是i+1，而使用i++会改变原来的值。

## 值传递和引用传递以及指针传递(C++)

值传递——形式参数是实际参数的拷贝，改变形式参数的值不会改变外部的实际参数的值，从被调用的函数的角度来看，值传递是单向的，从实际参数到形式参数，而参数的值只能传入，不可传出。

指针传递——形式参数是指向实际参数的指针，当对形式参数的指向操作时候，相当于对实际参数本身做操作。

引用传递——形参相当于是实参的“别名”，对形参的操作其实就是对实参的操作，在引用传递过程中，被调函数的形式参数虽然也作为局部变量在栈中开辟了内存空间，但是这时存放的是由主调函数放进来的实参变量的地址。被调函数对形参的任何操作都被处理成间接寻址，即通过栈中存放的地址访问主调函数中的实参变量。正因为如此，被调函数对形参做的任何操作都影响了主调函数中的实参变量。

STL中的heap有大堆和小堆，最大堆就是母节点大于子节点，而最小堆就是母节点小于子节点。

注意：

Swap的是元素的值，所以要对指针进行解引用。

2018年6月2日08:05:29

关于Scanf读取EOF的猜想，个人猜想，EOF其实就是win中的ct+z。默认的输入是会考虑这一点的。

关于Scanf中空格问题，与printf中不一样的是，scanf回忽视所有的空格

关于bool类的输出，一种是C11以后的版本，是自带bool的，一种是不带bool的，对于不带bool的，你要不就是提前定义一个string存储true或者是false，要不然就是直接在print中进行判断。

1011题

首先注意数据类型 long long int 以及对应的lld输出

此外是一种全新的输出true或者是false的方式。也就是：

输出一个bool类型的方式。

1012题

使用vector进行操作，极大的解决了最后不知道有没有对应类型的数据而不知道是不是要输出N的情况。更明确的说，应该是个让vector[i].push\_back();这样不断的增加元素。

为什么使用vector，其实使用一个统计变量进行判断也可，这种题其实就是毫无技术含量的题。

1013

注意不是输出从M到N的素数，而是从M个素数开始，一直输出到N，

也就是说，要从1开始遍历，然后，数到M个素数的时候，开始输出，然后数到第N个素数的时候，停止输出。

判断素数，number对从2开始的数进行整除，起点2，终点的时候，i就是除数。

使用vector存储合适的number，素数的判断肯定是要从2开始，但是由于最后的输出并不是说从2开始的素数全部输出，因而使用vector存储可以输出的素数。

输出，这种要求的输出，一定要用for循环判断，然后整个输出的队列其实就是空格结合元素进而结合末尾的\n组合而成的。

1014 福尔摩斯字符串

注意，每一次循环之后，都要增加变量的值，并且注意，循环中 和 break的使用。

1015 德才论，也就是algorithm的cmp和vector的使用，类似于1013

分为两部分，将满足要求的人放到对应的数组中去，然后在输出之前，要用algorihtm进行排序，然后排序之后逐个输出vector。其实是蛮简单的。

很显然，首先排除这些DE&CAI都小于L的才是最明智的想法。

1016

统一使用long long int 为了应对越界问题。

1017

对于这个1000位的整数，不应该用数值类型来存储，而是应该使用string来存储。

【本质上是编写笔算的除法】

注：使用char数组进行string的存储，而不是使用string

**模拟手动除法的过程，每次用第一位去除以B，如果得到的商不是0就输出，否则就\*10+下一位，直到最后的数为余数～**

1018

第一点：Awin代表Blost，Bwin代表Alost，而Tie是N-Awin-Bwin，这样可以极大的简化变量声明。

第二点：Awin的时候，只需要统计A当前的手势是啥

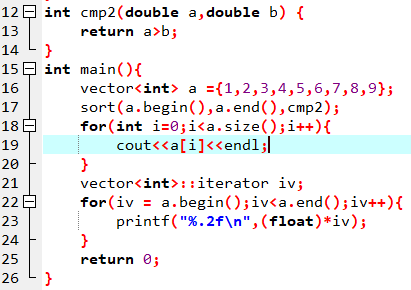
第三点：使用字典存储不同手势出现的次数是比较合理的选择

第四点：要学会使用Algorithm对于字典进行排序

第五点：要学会字典中元素的不同情况的输出。

1025 月饼

1. 字典排序，按照value进行排序
2. 使用vector进行存储，然后有用的一个是单价一个是库存量
3. 由于最后只需要利润，而不需要具体的种类，相对来说一个map<int,int>就行了
4. 对于浮点数，使用double，对于整数，使用long long int类型即可。
5. 我发现这个[**柳婼 の blog**](https://www.liuchuo.net/)特别喜欢使用结构体。。。。
6. 本题除了使用结构体作为数据结构之外，还可以使用字典，尤其是字典按照value进行排序sort。
7. //计算单价其实单纯是为了排序，不管是字典还是struct都是这样



图：关于使用iterator进行输出，以及使用sort自定义排序

注：

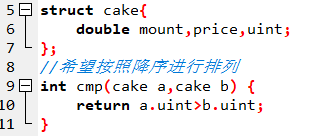
关于vector的初始化，可以使用

这种方式进行初始化。

甚至我跟你讲，可以用vector<int> a(3,10);来进行初始化。

注：struct的灵活使用

灵活使用struct作为数据结构，并且可以构造比较函数进行比较，有时候可以简化做题。



图：使用Algorithm进行排序操作【挺认真的啊】

1021题

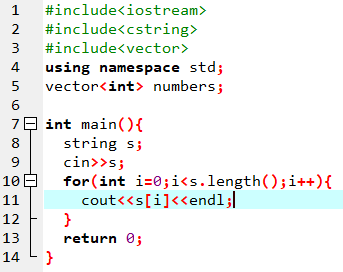
1 首先这个1000位数让我想起来了，还是应该使用string进行存储。

2. 然后输出的形式让我感觉可以使用map，而使用map在目前是完全可以使用struct进行代替的。

3. 事实证明，的确可以使用string按照数组的方式进行操作·。

stoi可以将string转化为int之类的。

除此之外呢，可以用to\_string来进行转化工作。



图：对于string按照数组的方式进行操作

Cin后面最好加上一个getchar() 否则会出大问题

1022 D进制

给的是10进制，基本操作。

1. 使用stack存储的确是一个很好的思路，但是使用vector，然后逆序输出一样可以解决问题。
2. 然后请审题，题目所说是2的30次方，两个数相加绝对会在int范围里面，没必要使用long long int。正常的int应该是4字节长度，也就是32位，也就是最大表示2的31次方。别忘了补码表示法还有一位是符号。
3. 最愁人的就是如果说mod的结果是0也就是说，最后的新的数应该是10.也就是涉及到了进位。
4. Mdzz，你已经获取了数字，存在了stack里面，为啥非要计算，计算多费时间啊，而且涉及到0之后会更加的麻烦。
5. 未解之谜——关于转化为0时的数值的处理。

1023

审题，我再说一遍审题，你真的是严重的不审题。所以最后做的完全是不对的。

然后就是基本操作了。

1024科学计数法

首先是按照固定的顺序输入的，相对比这个判断是不是科学技术要简答的多了。

正则表达式[+-][1-9]"."[0-9]+E[+-][0-9]+

也就是固定的一位整数，多位小数，然后是多位指数位置。

也就是

+1.23400E-03这种，一共是

+ 符号1

1 整数

234000 小数

E- 符号2

03 指数

一共5部分，感觉使用int[]或者是使用stack都比较不错。

【这个破题没有任何价值和实际的算法，不学也罢，考试考这种题的几率非常小，因为没有任何的算法含量】

//进一步说，分为数值和指数。数值用小数存储，而指数可以理解为string后面加的几个0或者是小数点之前要加的多少个0；

//额外需要注意的是，全程操作使用string，这样方便处理0值。防止0被消除。

【string就是容器string，不要觉得是字符数组】

【C++中的4个字符数组，包括char[] && const char\* && string &&char\* 】

Algorithm

1符号位输出 cout

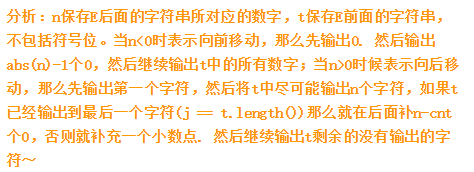
2 判断数值a（string形式保存）（E之前） 和 指数部分b（int形式保存）（E之后）

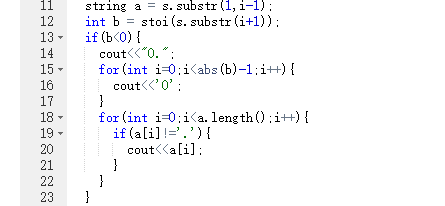
3 根据指数部分的数值，如果指数部分的数值是正的，也就是说，需要先输出a,然后输出与b有关的个数个0.如果指数部分的数值是负的，也就是说要先输出小数点和与b数目有关的0，然后输出a的数字。

3.1 如果说是n<0,首先应该输出的是“0.”，然后是abs(n)-1个0；然后是输出a中的所有的数字。从整数位开始的输出

3.2 如果说是N>0，两种情况，一个是a的长度其实是比你的b还多的，这种情况下，会出现小数点，小数点之后的数是a没有输出完的东西，如果是a的长度小于b，那么就在后面加上指定个数的0就好了。

3.3 特别注意的就是：关于n>0的情况，虽然是先输出字符，但是a的长度和b的长度相比是无法确定的，a长度小于b的时候，只需要在最后补充0就行了





图：截取部分元素并且输出b小于0的时候的情况，也就是带有小数点的情况



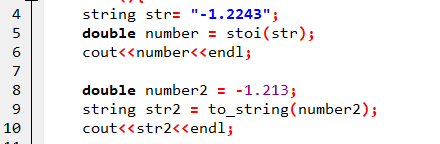
图：关于b大于0的时候的情况，注意，这个时候需要讨论一下，思考关于到底最后有没有小数点参与计算。如果是存在小数点参与运算，判断条件应该是a和b之间的关系属于：

也就是说a的长度并不是全部和b进行比较！！！！

注：

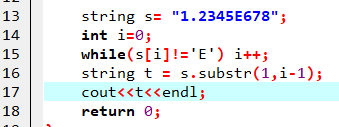
to\_string 和 stoi 的功能：

1. 将字符串转换为int 【但是不能是double类型的数据】
2. 转化为double是stof或者是strtoXXX
3. double/int 变成string

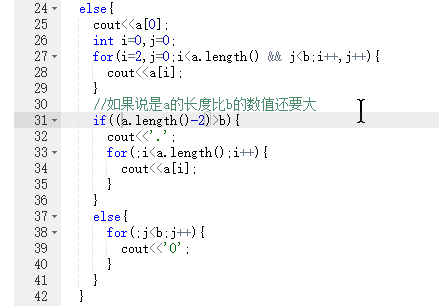


图：stoi和to\_strin的实际操作

SubStr操作



注意：坐标开始的位置仍然是0，本题中取1只是了取小数位置。



**Length 和 Size（）和sizeof()**

关于string以及其他数据结构在length和size上面的差别。

似乎length= size/sizeof(uint);

历史上来说：length是一开始出现的，出现在string中。

后来的C++为了兼容，创建了size和length兼容，都返回一个size\_t类型的东西

然后对于string来说，二者返回的都是字节，如果想要获取元素的数目，额外的需要使用

1025 根据K长度，反转列表

审题：

输入

——第一行给出第1个节点的位置，所有节点的个数以及K的数值

——然后是N个节点的实际情况

——节点的情况包括——自己的地址 自己的数值 下一个节点的地址

思考：链表的实际情况真的会影响到你的数据的分布情况吗？

你的链表，不管是一个什么样的分布情况，最后的影响真的大吗？

思考：

，居然有些节点不属于这个链表！！！！！

审题：

最重要的就是你要重点看这个数据中的大小的限制情况，比如说：这个链表中的地址最大是多少，方便你创建对应的数据结构。

1025 进阶解答

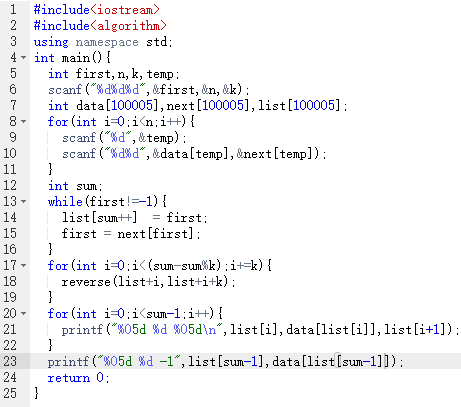
关于 或者是说 的使用



这个东西，首先是array的指针形式，然后reverse是对于指针进行的操作。

1025 特别注意：

PAT考试的输出printf或者是cout是非常恶心人的，非常容易遗忘的就是printf后面的\n也就是换行。关于这个\n是非常容易被忘记的一个情况，和积分中的C以及中的绝对值符号容易被遗忘是一个道理。对于黑盒测试，最重要的就是要小心输入和输出的一个实际情况。



1026 计算时间嘛

很简单的啦，最要命的其实是四舍五入的一个情况，在这个程序中，由于你的基础常数是100，所以你四舍五入的方法很简单，就是让你的东西+50，然后去做除法，除法的结果会被强制转换为int。

【计算除法的一个方式】

1027

至于说打印，其实这里更多的是数学问题。

你如何确认自己的结果是正确的，那么我告诉你，首先不要打印字符，单纯做到打印当前行的空行以及打印当前行的符号的数目就可以了。

如果当前行的符号的数目没有问题，说明你可以进行下一步也就是正式的打印了。

1028 oldest and youngest

可以用struct/数组/字典

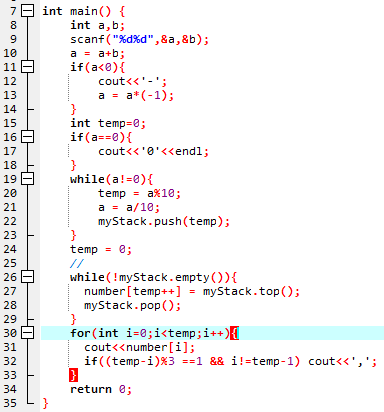
我们的思路还是有点僵化，正确的编程思路应该是：每次读取的时候就进行判断，而不是淡出的存储为一个数据结构，之后对于数据结构单独操作。

首先应该判断的是，这个东西是不是一个合理的数值。然后是对于数据进行操作。

甲级1001

我记的这个number在转化为数组存在一个atoi的函数，然后关于int也存在着to\_string的操作。

【先研究一下这个不用atoi以及to\_string形式的操作结果，然后再尝试使用to\_string的操作方式】



【注意一种特殊的情况就是a和b都是0的情况。】

尝试使用to\_string() 进行操作

### STL中的list

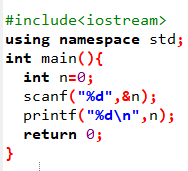
### STL中·的·array

### STL中的·vector

# 要求：

1. 掌握Vector List Array等等的常用操作
2. 掌握基本的Stack的操作
3. 掌握String的操作
4. 熟悉Algorithm以及Iterator的操作

## 标准开局



关于图

图中是存在点权和边权这两个东西的，并且存在点和点直接的连接关系。

首先就是图中的点的关系的存储，可以使用邻接表或者是邻接矩阵来实现。

其中邻接矩阵较为容易实现，但是由于实际的内存的限制，最多也就是1000\*1000

然后邻接表可以使用链表或者是vector实现【使用int没有权重，使用struct有权重】

对于图的遍历无论是DFS还是BFS，都有两种实现的方式，一个是使用stack或者是queue等作为辅助，一个是使用递归的方式。

记住一点，所谓的遍历，是DFS或者是BFS，肯定是使用递归的方式进行遍历。然后比较重要的就是遍历之前是需要使用vis[SIZE]来存储你是不是已经被遍历到了的，或者给Node增加一个属性也是可以的。然后就是很一般的循环遍历。

从起点寻找自己所需要的所有的点，然后是从这些点循环的遍历所有的其余的点。

1034

这个题的基础逻辑是：每一个人都可能是一个head，然后就判断从这个人开始可以便利的到所有的人有多少，总共的权重之和有多少。

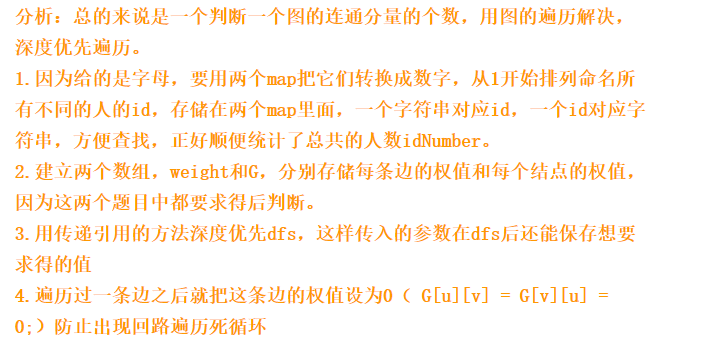
2018年6月12日13:54:05

这个题根据目前的理解，首先是图中联通分量的一个遍历，然后是图中的一个权重的累计。

关于权重的累计，首先，你要知道，如果已经累积了一个联通分量之后，要把权重变为0，以防止出现重复的相加。【这其实也是因为每次都是从第一个点开始遍历，而没有真正的分离开所有的联通变量】。

然后，你输入的是string，但是实际操作中要将string转换为int。

简单的说，从最底层的数据结构开始就决定了你上面的样子。



1068 关于背包问题

这种DP问题，并不是说，说你随便拿，而是有自己的限制条件，也就是说你要按照重量来拿，也就是说，~~Dp[i]的答案到底是Dp[i-1]还是Dp[i-2]+value[i]~~.

区别于偷东西。。。。。偷东西是要获取最大值。比较的是Dp[i]和 Dp[i-1]

这里面考虑到了一个重量的影响。

这种东西，Dp[i]表示的是质量为i的时候可以拿到的最大的东西

所以不是普通的+

最长回文串的求解：

1. Manacher Algorithm
2. 二分查找+Hash
3. 后缀数组的方法
4. 使用Dp动态规划

## Manacher Algorithm

有几个东西比较重要：

一个是Max，是表示的当前所找到的所有的子串中，最右边的边界位置。

然后就是Max所对应的子串的一个对称中心，也就是对称轴的位置的情况，用POS表示。

然后围绕着上面的Mx和Pos，我们开始使用i从0开始进行便利。

遍历i的目的就是为了不断的更新RL数组，也就是

RL[i] = mx>i?min(p[2\*pos-i],mx-i):1

然后，不断的更新当前便利i所产生的最大数组——resPos和resLength这两个东西。

重点：只要求出来了这个RL数组，RL数组-1就是当前字符的最大的回文字串的长度。

所以关键在于求出增广字符的每一个子串的长度。

马拉车的核心就是：

P[i] = mx>i?min(p[2\*id-i],mx-i):1

最核心的部分就是关于这个id的对称位置的i和j，是最关键的。

此外，最长的回文字串可以使用hash+二分，也就是一种减少匹配，多利用以前的一个结论。

然后,你也可以采用这个叫做DP来做。或者是后缀数组来做。

对于id的理解，id是所有的回文的字串中，可以延伸到最右端的回文字串的中心点的位置。

MaxRight是当前的所有的回文字串中，所有的字串中的，触及到的最右边的位置，然后，触及到这个位置的字串，然后肯定不是只有一个串会到这个位置，但是，我们只记录，在这些字串中的最长的字符字串中对称轴的位置。

只记录max的位置，以及max所在的最长的字串中的对称轴的位置，

如何求RL数组，最简单的方式就是使用回文串的对称性，扩展回文串。

max 和 pos是实施更新的，但是max所对应的字符串却不一定是最长的字符串。

i是你每次便利的一个东西。

在使用j给i赋值时候，所比较的一个核心名词——边界，边界是啥。边界是有右边的max还是说边界也包括了左边的一个pos，目前我觉得，边界更明确的说是在右边的pos。

# 图专题Graph

## 1034 Head of a Gang【连通分量、图的遍历DFS】

这个题要干啥，这个题就是给你一个图（点集），然后让你对这个图进行遍历。具体的操作方式就是对每一个点进行操作，寻找这个点可以到达的可能的位置。

然后在遍历的过程中寻找一个最大的通量的和。为什么使用DFS？反正使用BFS也是可以的。

## 1003 救援队【Dijkstra算法】

第一原则，从点到点的一个最短路径，在遍历的过程中同时计算权重的和。

第二原则：在出现多条路径的时候，寻找最大的权重。

#### 1013 Battle Over Cities【DFS遍历，统计联通分量的个数】

城市作为点，公路作为线，然后如果一个城市被敌人占领，那么这个城市的所有的公路都要被关闭。如果图中一个点被占领，那么保持其余城市联系所需要的公路条数。

比如：一共有三个城市City1，City2，City3，然后有公路是City1-City2以及City2-City3，如果是到道路City1City2坏了，那么保持City2和City3的连接，就需要额外的修建一条公路。

第一原则——城市之间必须要用线连接。

并行第一原则——如果一个点被毁坏，就要将于这个点所有相关的线设置为0，而且是双向设置为0；

返回值——需要新增的点和点之间的连线。

对于所有的点的遍历，然后返回的是每一个点的联通分量，其实说到底，就是要给你一个图，返回这个图所有的联通分量。

2018年6月14日15:29:44

1013 可笑的要死，一个图连最基本的权重都不用录入，完全就是遍历联通·分量的个数，这种题考试考了怕不是要笑死人。

有两点需要注意：一个是城市的这个起始的index是1而不是0，另外一个是说，关于你的遍历过程中的边界条件，也就是说，第一个就是在所有的节点进行遍历的时候，在等于自身的时候应该跳过而不应该进行遍历，一个是在dfs的时候，除了这个点没有被遍历过之外更重要的是你一定要做到对于这些点的联通性的判断。如果有必要的话，在一次遍历之后，需要将权重双向设置为0；

## 并查集

能判断点集中的点是不是同属于一个集合，做不了的事情——无法将点从集合中删除，或者说，本来点1和点2之间存在中的联系（属于一个集合），你没办法让两个点属于两个不同的集合。【砍边的操作是后续的一个高级数据结构】

核心操作：判断两个节点是不是属于同一个集合，首先将同一个集合中的东西组织成为一个树，然后判断这两个节点的根节点是不是同一个节点。【判断树的根节点是不是一样】

1. 两大操作：

第一个操作，判断两个点是不是同属于一个集合，第二个就是把两个点加入一个集合中去。

1. 并茶几每一次操作，都会让原来很深的一个树变得更矮，因为原来的深层的节点，指向了一个父节点。
2. 带权重的并茶几，会将子节点的权重转移到父节点上面。
3. 然后如果是x == Father(x)，只说明一个问题就是，你已经到了根节点了。
4. 一个是Find()不断的自己调用自己，一个是Union，也就是将两个点合并。

## 1021 【树的直径+并茶几+DFS】

大概和我想的差不多，就是说还是类似于1013的那种遍历方式，但是1013的目的是为了统计联通分量的个数，而本题是为了统计一个节点所对应的联通分量的实际深度。也就是说，需要记录没一个节点所在的联通分量的一个里面的元素的个数。

这个题是啥意思？

第一原则：判断联通分量的个数，如果联通分量的个数比1要多，说明肯定不是一个树。

这个时候需要输出联通分量的个数。

题意：给出一张图，判断该图能否形成一棵树，若能则求出树的最长简单路（树的直径），否则求出连通分量的个数。

解决这道题的整体思路是：   
1. 先**通过并查集判断连通分量的个数**，若连通分量只有1个就说明不存在森林；   
2. 对剩下的这些连通顶点做**两遍搜索**。第一遍任意选一个顶点u做起点，通过BFS找到距离u最远的顶点v，第二遍再从v出发做同样的BFS，搜到的最长路即树的直径，可以得到直径的端点，也就是题目所说的Deepest Root。

值得注意的点：

* 为了找到距离u最深的顶点v，在BFS过程中用level数组记录每个顶点的深度，即其前驱节点的深度+1，这样当BFS完成后，level里就记录了所有顶点与u的距离。
* 非常需要小心的是，这两遍BFS都可能会找到多个终点，因此每遍BFS以后都要记录下来；但是*有些顶点可能会被记录两次！*（测试点1、3、4）举个栗子，若第一遍选择的顶点恰好在直径的中间（或者图只有一个顶点），则两端的顶点都可能是终点，如果用vector来记录就会被push两次，导致重复输出。
* 由于最终输出要求排序，结合上面这点，可以用最简单的布尔数组root[i]来标记i是否为端点，能避免排序和重复的问题。

A graph which is connected and acyclic can be considered a tree. The height of the tree depends on the selected root.

# 图的分类

DAG 有向无环图 ，一种算法是关于其拓扑排序，一种算法是关于其的图转树的操作。

## 树的直径

定义：树的最长简单路，

求法：两边DFS，任选一个节点，开始DFS，然后终点是距离当前的点最远的点，第二次的DFS从这个最远的点开始进行遍历。第二次找到的最长的路径就是树的直径。

原理: 设起点为u,第一次BFS找到的终点v一定是树的直径的一个端点

证明:   
1) 如果u 是直径上的点，则v显然是直径的终点(因为如果v不是的话，则必定存在另一个点w使得u到w的距离更长，则于BFS找到了v矛盾)   
2) 如果u不是直径上的点，则u到v必然于树的直径相交(反证),那么交点到v 必然就是直径的后半段了

注意：不要说什么直径，其实就是最长的简单路径。

求法，

1. 任选一个点作为起点，然后对树进行DFS遍历，找出离u最远的点v
2. 以v为起点，再次进行DFS遍历，然后找到离v最远的点w，那么从v到w的路径长度

## 【大臣的旅费】

很久以前，T王国空前繁荣。为了更好地管理国家，王国修建了大量的快速路，用于连接首都和王国内的各大城市。

为节省经费，T国的大臣们经过思考，制定了一套优秀的修建方案，使得任何一个大城市都能从首都直接或者通过其他大城市间接到达。同时，如果不重复经过大城市，从首都到达每个大城市的方案都是唯一的。

聪明的J发现，如果不在某个城市停下来修整，在连续行进过程中，他所花的路费与他已走过的距离有关，在走第x千米到第x+1千米这一千米中（x是整数），他花费的路费是x+10这么多。也就是说走1千米花费11，走2千米要花费23。

J大臣想知道：他从某一个城市出发，中间不休息，到达另一个城市，所有可能花费的路费中最多是多少呢？、

<https://blog.csdn.net/wukk5201315/article/details/45815081>

【数据结构中的树的直径的问题】

<https://blog.csdn.net/qq_31917799/article/details/52404138>

【树的直径】

## 1010 数制转换【二分】

//所以二分的意义体现在哪里？这一点要明确。

1010题的判断很有问题，对于暴力的搜索完全可以得到大部分分数，还是那句话，二分查找的效果非常不明显。

## 关于string/字符数组/一般性数组的长度

第一种：strlen(char\*)函数求的是字符串的实际长度，它求得方法是从开始到遇到第一个'\0',如果你只定义没有给它赋初值，这个结果是不定的，它会从aa首地址一直找下去，直到遇到'\0'停止。

第二种：sizeof(),求所占总空间的字节数。

char[] a={'a','b','c'};

sizeof(a)的值应该为3。

char[] b={"abc"};

sizeof(b)的值应该是4。

若string str={'a','b','c','\0','X'};

那么sizeof(str)为5，strlen(str)为3。

有这么几个函数，却不知道区别。sizeof() 、strlen()、str.length();

(C/C++ strlen(str)和str.length()和str.size()都可以求字符串长度。

其中str.length()和str.size()是用于求string类对象的成员函数

strlen(str)是用于求字符数组的长度，其参数是char\*。)

## STL反向迭代器

  begin和end操作产生指向容器内第一个元素和最后一个元素的下一个位置的迭代器，如下所示。这两个迭代器通常用于标记包含容器中所有元素的迭代范围。

c.begin() 返回一个迭代器，它指向容器c的第一个元素

c.end() 返回一个迭代器，它指向容器c的最后一个元素的下一个位置

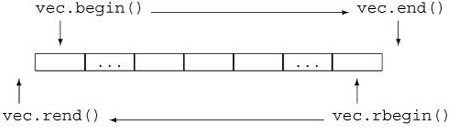
c.rbegin() 返回一个逆序迭代器，它指向容器c的最后一个元素

c.rend() 返回一个逆序迭代器，它指向容器c的第一个元素前面的位置

【真是开了眼了，没想到原来begin返回的仍然是一个迭代器】

然后迭代器可以返回一个const的迭代器，也可以是返回一个非const的迭代器。

反向迭代器是一种反向遍历容器的迭代器。也就是，从最后一个元素到第一个元素遍历容器。反向迭代器将自增（和自减）的含义反过来了：对于反向迭代器，++ 运算将访问前一个元素，而 -- 运算则访问下一个元素。（注意，begin并不是返回地址，begin返回对象的类型一样是一个迭代器，所以我们需要使用auto进行标注）



## CCTYPE

[**isalnum**](http://www.cplusplus.com/reference/cctype/isalnum/)

Check if character is alphanumeric (function )

[**isalpha**](http://www.cplusplus.com/reference/cctype/isalpha/)

Check if character is alphabetic (function )

[**isblank**](http://www.cplusplus.com/reference/cctype/isblank/)

Check if character is blank (function )

[**iscntrl**](http://www.cplusplus.com/reference/cctype/iscntrl/)

Check if character is a control character (function )

[**isdigit**](http://www.cplusplus.com/reference/cctype/isdigit/)

Check if character is decimal digit (function )

[**isgraph**](http://www.cplusplus.com/reference/cctype/isgraph/)

Check if character has graphical representation (function )

[**islower**](http://www.cplusplus.com/reference/cctype/islower/)

Check if character is lowercase letter (function )

[**isprint**](http://www.cplusplus.com/reference/cctype/isprint/)

Check if character is printable (function )

[**ispunct**](http://www.cplusplus.com/reference/cctype/ispunct/)

Check if character is a punctuation character (function )

[**isspace**](http://www.cplusplus.com/reference/cctype/isspace/)

Check if character is a white-space (function )

[**isupper**](http://www.cplusplus.com/reference/cctype/isupper/)

Check if character is uppercase letter (function )

[**isxdigit**](http://www.cplusplus.com/reference/cctype/isxdigit/)

Check if character is hexadecimal digit (function )

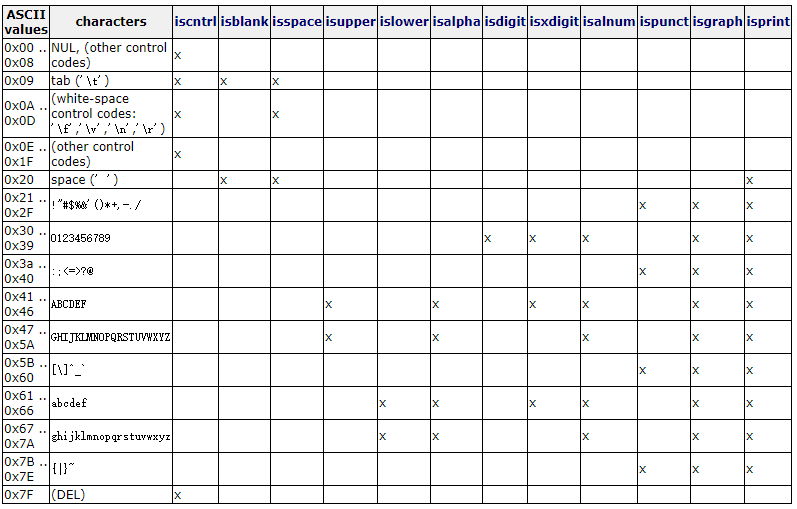
Cctype就是一个字符判断函数，以及字符状态操作函数

[**tolower**](http://www.cplusplus.com/reference/cctype/tolower/)

Convert uppercase letter to lowercase (function )

[**toupper**](http://www.cplusplus.com/reference/cctype/toupper/)

Convert lowercase letter to uppercase (function )



## Size\_type的本质



所以从上面可以明确的看出，vector<int>::size\_t其实就是对于类型的一个定义，类似于typedef的使用。

其实对于本质上来说vector<int>::size\_type就是一个int.

注:

首先这个东西是由string和vec来定义的，用于保存vec和str的长度，标准库中的size\_type定义的东西是unsigned的东西。

size（）的抽象意义是字符串的尺寸， string::size\_type抽象意义是尺寸单位类型。

String::size\_type在不同的机器上面的大小是不一样的

String.size() 返回的是str的一个字符数量，应该会比下标索引数大1，因为是从1开始累积的。

Size\_t不是容器的概念，而size\_type是容器概念，必须要在容器的条件下使用。



在string标准库中，有size\_type,然后在使用bitset的时候遇到了size\_t

标准string中有一个size(),返回的是字符串中的字符的个数。

Size()函数的返回类型是size\_type，不可以赋值给int。

string类类型和许多其他库类型都定义了一些**配套类型**（companion type）。通过这些配套类型，库类型的使用就能与机器无关。size\_type就是这些配套类型中的一种。

size\_type被定义为与unsigned型（unsigned int, unsigned long）具有相同的含义，而且可以保证足够大能够存储任意string对象的长度。为而来使用由string类型定义的size\_type类型。程序员必须加上作用于操作符来说明所使用的size\_type类型是由string类定义的。

我们为什么不适用int变量来保存string的size呢？

使用int变量的问题是：有些机器上的int变量的表示范围太小，甚至无法存储实际并不长的string对象。如在有16位int型的机器上，int类型变量最大只能表示32767个字符的string对象。而能容纳一个文件内容的string对象轻易就能超过这个数字，因此，为了避免溢出，保存一个string对象的size的最安全的方法就是使用标准库类型string：：size\_type().

什么是size\_t类型呢？其实本质上和size\_type没有多大区别

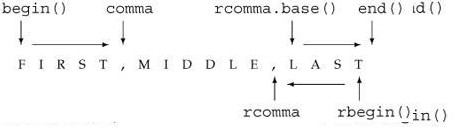
其实size\_t和size\_type类似，size\_t 类型定义在cstddef头文件中,该文件是C标准库的头文件stddef.h的C++版本.它是一个与机器相关的unsigned类型,其大小足以保证存储内存中对象的大小。用法如下：

1. **class** container
2. {
3. **public**:
4. **typedef** **size\_t** size\_type;
5. //...
6. };

## Bitset标准库

主要是使用二进制对于这个字符串和数据进行操作。包括位操作符等等非常底层的操作、

## 反向迭代器



就是用反向迭代器来寻找，用正向迭代器来进行输出。

 图 2 显示的对象直观地解释了普通迭代器与反向迭代器之间的关系。例如，正如 line\_rbegin() 和 line.end() 一样，rcomma 和 rcomma.base() 也指向不同的元素。为了确保正向和反向处理元素的范围相同，这些区别必要的。从技术上来说，设计普通迭代器与反向迭代器之间的关系是为了适应左闭合范围  
（第 9.2.1 节）这个性质的，所以，[line.rbegin(), rcomma) 和[rcomma.base(), line.end()) 标记的是 line 中的相同元素。  
      反向迭代器用于表示范围，而所表示的范围是不对称的，这个事实可推导出一个重要的结论：使用普通的迭代器对反向迭代器进行初始化或赋值时，所得到的迭代器并不是指向原迭代器所指向的元素。

STL 中的vector容器的front和 back返回的是第一个元素和最后一个元素的引用。

## Struct——轻量级的类

你搞PAT肯定不会用class，时间根本不支持，但是使用一个struct还是可以的，所以可以尝试的高科技包括vector<node>这样的带有结构体的操作。

也就是让一个vector来存储node，然后经过实际尝试，可以使用iterator实现遍历。

## 在PAT中目前为止，基本上没用过指针，我严重怀疑我是不是还是没学过C++的感觉

判断一个数是不是一个素数isPrime（）

一个比较好的改进是，只用2到根号n的素数做为除数，这样可以减少不少除法的次数，算法的代码可以表示为：

另外如果一个数是偶数，可以直接判断这个数是一个false大大加速。

此外就是，Miller－Rabin

最终解决方案：

一个就是验证是不是一个偶数，然后就是在验证的时候，从3开始，每次都+2到偶数，然后最后就是到sqrt(number)为止。

1015 数制转换以及素数的判断

1014 银行排队

一个值得注意的地方就是说：注意数据结构的设计，把窗口前的M个位置设计为一个队列，然后设计一个数据结构，也就是struct里面就是用于存储这些queue<int> 以及第一个人结束的时间poptime，以及整个队列的结束时间endtime。【数据结构的设计非常的重要】

如果想让代码中的部分全部都是从1开始，那么如果你一共有N个人，你索性把数据结构设计为N+1就行了呗。比如是一个vector<int>，你设计为一个(n+1)不就行了吗。

【本题主要考察的就是STL以及抽象数据结构】

1015 【没啥技术含量】【一个数制转换+一个prime判定就解决问题了，有啥技术含量】

1016 电话号码

主要的难点就是排序，一个原则是按照人的姓名进行排序，另外一个原则就是按照时间进行排序。

此外就是时间要预先化成分钟，也就是一个月为单位的分钟。

【和1014银行存钱一样，都是考察对于自定义复杂数据结构。而没有涉及到过多的数据结构和算法方面的内容。】

//到底怎么一个排序？就是按照人名来拍，人名拍完了，就是按照绝对时间来排序。反正就是这两个原则。

## STL map的使用

Map的初衷有两点：

数组的映射关系，从int到int满足不了实际的需求，需要使用map建立任意类型(包括STL和struct)和任意类型之间的映射关系。

第二点就是当你要判断某些数字在一个文件中是不是出现过的时候，可以建立一个bool类型的hashTable[size]，通过判断hashTable[x]是true还是false从而判断这个x是不是曾经出现过。如果这些数字很大，有些情况下数组是没办法开这么大的。这个时候建立一个map就显得很重要了。

可以把数字当成一个字符，然后建立string到bool的映射关系，或者是直接的数字int到bool的映射关系

注意事项：

1. Mp中的key是唯一的东西。
2. Mp的用途，1、比如需要一个映射，而建立数组形式的映射不仅仅是耗费时间，而且代码量非常的大。
3. 判断一个数字是不是在文件中出现的时候，可以尝试建立map。
4. 而字符串和字符串的映射有些时候也会永达。

C11最新的标准是unordered\_map 用hash散列代替了原来的红黑树，这个unordered可以单纯做到映射，而不排序，速度比map快的多。

注意：map的迭代器可以修改数值，而不会出现修改key的情况。

根据key的查找也就是find函数的时间复杂度是nlogn。如果有1000个记录，最多查找10次。

为了使用方便，可以对模板类进行一下类型定义，

typedef map<int, CString> myMap;

myMAP enumMap;

//auto功能真的是非常强大，你原来需要额外的声明的遍历用的迭代器完全可以直接使用。

//注意map中的erase不可以删除一个不存在的key，如果删除一个不存在的key相当的不方便。

Map

根据key找value，可以find可以直接下标索引进行操作，除此之外，需要使用value找key的唯一的方法似乎就是for循环迭代操作了。

begin()和end()两个成员，分别代表map对象中第一个条目和最后一个条目，这两个数据的类型是iterator

然后erase操作其实是针对地址进行的操作。

//再次强调，注重auto的使用，

1017的学习

真的讲套路的话，其实1017和1014是一样的题，1014要求你把所有的人都要输出他们的结束时间，而1017也要获取每个人在银行的初始时间和结束时间。

因而，我们要做的就是首先要创建一个数据结构专门用于存储每个人到达的时间，终止的时间等等，此外呢，额外需要的就是要有一个queue甚至是queue<node>来专门存储这个节点。