1.memset逐字节赋值，因此除了char型数组，其余类型的只能赋值0，或者-1；

2.algorithm里面含有max，min，swap

3.cctype,cstdio可以不使用std名称空间

4.getline(cin,line)(line的type是string型的)

5.template<typename T>

6.longlong 的输入和输出均使用lld。

7.lower\_bound(it,it,key)返回的是iterator，要想知道下标，可以-a。另外，前提是已经排好序，lowb返回第一个大于等于key的值，upb返回第一个大于key的值

【L，R），L为lower\_bound,R为upper\_bound。R-L为key的个数。L指向大于等于key的第一个值的下标。本质是二分查找。

8.优先队列的实现方法priority\_queue**<int,vector<int>,greater<int> >** pq;

9.char str[10],cin.getline(str,10). String str; getline(cin,str)

10.顺序容器和关联容器中都有的函数有erase和clear

10’顺序容器只有vector，list，dequeue，关联有set，map和multi版本的。

11.顺序容器中的front，back，push\_back,pop\_back,

12.stack,queue,pq,均为迭代器适配器，均不支持迭代器。

13.非随机访问迭代器不支持<，>等运算符，可以理解为指针所指空间随机，不连续，无法像随机访问迭代器那样比较大小

14.insert(it,ele) insert(it,itb,ite) erase(it) erase(itb,ite).

15deque含有pop\_front和pop\_back

17.inserter插入迭代器，用法与迭代器类似，只不过会插入元素。

Inserter（container，it）eg：inserter（x,x.begin()）;

Back\_inserter(container),在容器的后面插入元素。Front\_inserter(x)

18.cin>>s,和scanf一开始遇到空白后，会自动掠过。在最后遇到空白后，停止，并且把空白留在缓冲区。后面如果是getline(cin,str)由于刚才的\n仍然留在缓冲区，因此这个str里面将不会有字符。Getline遇到\n停止读入，并且把\n丢弃。即下次读入时读\n后面的字符。而cin和scanf保留后面的空白。Gets与getline类似。Gets会把最后读入的\n转换为\0。

fgets(s,530,stdin);注意是啥都读，毫无改变的读，包括空格和\n。会把前导空格也给读入，并且最后的\n也会被读入s中。如果遇到需要把前导空格也读入的题，可以使用fgets(),然后s[--len]=0。

19.在使用dfs时，vis[i]=1在一开始就写出，这个与在遍历时再写出，如果最开始进dfs时，没vis[i]=1，是不同的。

20 long long mp=a[i]\*p;虽然好像是结果没有丢失，但是，实际上a[i]\*p是按照int型保存在int中的因此，需要写成long long mp=(long long)a[i]\*p;

21.longlong的最大值为（1LL<<63-1）=9223372036854775807。也可以写成9e18。

22.double的精度最多保持15到16位。

23.在判断素数时，不要自作聪明，把int bound =（int）sqrt（n）+1，因为这样2就会被判断为不是素数.

24.对于哈希的冲突解决方法中的平方探测法，如果a+k\*k中的k到了Tsize还没有找到，那么就找不到位置了。只有k∈[0,Tsize)均验证了即可。

25.最短路算法总结：

最好用dijkstra+dfs，这样打印，思考均十分方便。

dijkstra需要的变量：G[][]，d[]，vis[]

G[][]保存边权，注意一开始要初始化为inf。Fill(G[0],G[0]+maxn\*maxn,inf)

D[]保存最短路径，注意在dijikstra开头初始化为inf。最终d[ed]为0，很可能是G和D没有初始化。

Dijkstra算法大致流程：首先要初始化vis，d，以及d[s]；就这3个变量，接着n重循环：先找最小元素,设立u，mi。mi=d[u=v]可以完成两步操作简洁代码。接着，如果u==-1，直接返回。否则数组vis[u]=1会数组越界。接着进行松弛操作。

Dfs需要的变量:pre[maxn],tmppath,path。其中path保持最优解。

流程：先判断是否到了起点，是的话就判断最优性，更新path。否则遍历这个点的所有pre元素。注意元素v在dfs里面进行push\_back，dfs最后结束才pop\_back.

26.对于二叉排序树bst，用算法笔记上的那个要初始化T为0，不要new。另外，注意二叉树的子树定义方法，最好分别定义。不要写成Node\* a,b;这的意思是只有a为指针，而b为Node。

27.一定要注意，写break的时候，一定要把break写在最后，不要在break后面加东西，否则后面的东西是不会执行的。

28.能不用getchar坚决不用getchar，这个真的是太恶心了。

29.bfs求最短路时，把dis打包在struct中，这样可以减少代码量和出错率.这样做的话，还需要加一个vis数组。

30.如果考虑使用bfs时，一定要考虑数据的范围，eg深度不能超过26，否则一定解题方法不是bfs。Eg：2014的binary tree这个题。虽然是binary tree，但是解法不是binary tree。

31.**字符串编程整理**：

1）读入

cin>>s,和scanf一开始遇到空白后，会自动掠过。在最后遇到空白后，停止，并且把空白留在缓冲区。后面如果是getline(cin,str)由于刚才的\n仍然留在缓冲区，因此这个str里面将不会有字符。

Getline与cin和scanf恰好相反，前导空白不会被掠过，照样被读入。遇到\n停止读入，并且把\n从缓冲区丢弃。Gets与getline类似。Gets会把最后读入的\n转换为\0。

fgets(s,530,stdin);会把前导空格也给读入，并且最后的\n也会被读入s中。如果遇到需要把前导空格也读入的题，可以使用fgets(s,100,stdin),然后s[--len]=0。

2）

Str.find(str,pos),找不到返回string::npos，值为-1.

Str.find\_first\_of(“0123456789”,start\_pos),找字符串中某个字符，第一次出现的位置。

Str.find\_first\_not\_of(“str”,start\_pos),找第一个没有出现的位置。

Str.replace(pos,size,str),在pos开始的size个大小替换为str

Str.erase(pos,size)

32.先序和后续遍历，确定树的个数。

如果pre[prel+1]==post[postr-1]则无法确定树是左是右，cnt++。但如果不是这样，那么一定是会有左右子树（这种情况下树是确定的），且pre[prel+1]一定是左子树的根结点，（假设是右子树的根节点，那么pre[prel+1]==post[postr-1]，与假设不符合）且post[i]是左子树的终点.最后,2\*\*cnt就是答案.

33.unordered\_map和unordered\_setC++11新出的。与map和set的底层实现不同。这两个本质上是哈希表，因此输出顺序与哈希有关，与输入无关。需要包含头文件<unordered\_map>与…

34.判断是否为完全二叉树：

（1）判断max是否等于n，即pre(rt,num)，左是num\*2，右是num\*2+1，找出最大的num。

（2）用bfs。每有一个子树入队，就cnt++，直到一棵树不存在左子树或者右子树为止，再break这个cnt如果等于n，那么就是完全二叉树。

35.注意，floor返回的是浮点数，因此，要打印时，用%d打印的话，需要在它的前面加个int。

36.判断是否是欧拉图，不能只判断结点的度，还要判断是否是联通图。

37.做题不要想当然，一定要按照自己的理解，解释通测试用例，再编码，要不然很浪费时间。又一次，一定要确定结合测试用例，并且测试用例是符合自己的理解的，才去编写代码。

38.最好把数据都读入到数组中，不要边读入边处理，因为，一个break，就会导致读入发生偏移，因而使程序出现错误。

39.在图中，不可以用bfs的方法来求最短路径。主要是因为mark这个。而且，用bfs来求单位为1的最短路径，在队列中，总是距离最短的先被访问。如果增量不是1，可以考虑使用优先队列。

40.注意不要使用next以及end作为变量，devc可能检查不出来，但这个是不明确的，还有y1。

41.注意pat的题，输出时四位数字时，一定要写%04d，这个0千万不要忘记。

42.注意pat的链表题，不要认为n个结点都是互相连接的，而是要自己数数cnt,最后输出时绝对不能写成n。

43.拓扑排序：

队列保存入度为0的点，d[]保存每个点的入度，cnt保存出来的点是否达到n个，如果少于n个表示存在环。邻接表或者邻接矩阵保存图。（另一种实现方式见大数据18代码。）

可以解决确定n个元素的顺序的问题。如果有环，则是矛盾的。如果每次只有一个入度为0的点出来，那么这个顺序就是确定的，多于一个的话，这些点的顺序就不确定了。

44.注意，使用getline时，一定要注意吸收前面的换行，而且吸收用getchar时，即使本地运行正确，判题如果是Linux环境可能报错，所以还是用getline吸收比较保险。

45.cctype

isalnum()  如果参数是字母数字，即字母或者数字，函数返回true

isalpha()  如果参数是字母，函数返回true

isdigit()  如果参数是数字（0－9），函数返回true

islower()  如果参数是小写字母，函数返回true

ispunct()  如果参数是标点符号，函数返回true

isspace()  如果参数是标准空白字符，如空格、换行符、水平或垂直制表符，函数返回true

isupper()  如果参数是大写字母，函数返回true

tolower()  如果参数是大写字符，返回其小写，否则返回该参数

toupper()  如果参数是小写字符，返回其大写，否则返回该参数

46.注意，如果是改变的是递归函数的名字，里面的函数名也要修改啊。

47.在递归枚举时，如果是枚举的数字可以重复，那么直接循环即可；如果是要枚举全排列A(m,n)，可以使用mark数组来标志有的数字是否出现在这次排列中，而如果是组合数C（m，n），一共有几种组合，那么就要,再多加一个dfs(cur,start)，每次i从start开始遍历，然后dfs(cur,i+1)。

48.prim算法，一个贪心算法，每次都把离所生成的树（一开始是任意一个结点），最近的的结点u（用d[u],其实是每个权值）保存给加入到新的树中，然后更新u周围的d[v]为min(G[u][v],d[v]),直到v-1次循环即所有的点都进入为止。

48.C++中不可以使用’y1’这个变量。

49.0-1背包注意要从下标1开始读数，如果压缩为1维度，要从后面往前走，

无限背包可以用记忆化搜索维度为1，也可用0-1背包的套路，但是如果压缩为1维要从前开始。滑雪可以记忆化搜索全部，找出最大值。

50．扩展欧几里得算法：ax+by=c。如何求出所有解呢？当c不被最大共因数d整除时，无解。x0,y0为x,y扩大c/g倍。然后dx=b/g,dy=a/g作为增量，即可求出所有解。

51.做字符串题要小心，如果现在没人做，那就先别做，等知道正确率后再做，因为很可能因为处理细节，而浪费大量时间。另外，无论是string还是字符数组，稍微越界如s[-1],s[n]，基本上都没问题，不会报错。一般是0，string(5,’\*’)可以快速构造重复字符串，注意一定是先为个数，后面才是元素，vector也一样。

52.注意：有些输入会很坑，比如a，b的值不一定就是一大一下，不可以擅自假定，还有最短路，两个点之间有多条路，要选择最短的路。

53.注意：int（float）会自动向下取整，而不是四舍五入。因此需要用round函数进行四舍五入。

54.在in，pre数组时，可以不建立树，直接拿数组进行遍历。

55.注意保留2位小数的题目，.2lf过不了可以试试.2f

56.bfs可以抽象成常见的状态搜索，mark可以增加维数以增加状态，注意使用pq时，要把小于定义为大于，才是小顶堆，一般要把S转换为其他字符。或者另加判断。

57.看通过率少并且做的人少的千万别做，很大可能有坑。

58.二分搜索，注意全部用<,[,)区间搜索，注意lo只取mid+1，hi只取mid，注意求一系列相同值时，可求左边界，或者右边界（需要减1）。注意需要判断函数的可以把其看作在0，1的序列中搜索（000000111111），然后就可以判断是左边界，还是右边界了。

59.运行时错误，runtime error可能是除0错误。

60.仅仅mp[x]，就会在map中添加一个新的结点，而这个新的结点的second是0。在百炼oj上，可以使用C++11。

61.异常退出先看scanf是否加了取地址符。

62.注意，涉及到位运算一定要加上括号，因为位运算是最低级的。

63.runtime error可能是数组开的不够。

64.Node a ={4,5}这个就是把内存中连续存放的变为a,pair<int,int>(4,5),这个就相当于构造函数。Q.push((Node){4,5})不行。

65.有一种题叫做种类并查集。

66.快慢指针法：判断条件是fast&&fast.next则为中间后面的那个元素，如果是fast.next和fast.next.next则是中间前面的元素。注意都是要从**第一个元素**开始。

67.C++的是s.substr(begin,lenth)。

68.如果用string &s,可以不超时，而用string s那么就会超时，因为复制需要的时间太多了，另外，做必要的剪枝很有必要。