目录

[逐题解析 2](#_Toc1933811)

[难点解析 6](#_Toc1933812)

[PAT经验 6](#_Toc1933813)

[常见英文 7](#_Toc1933814)

[贪心 7](#_Toc1933815)

[归并排序的递归和非递归实现 11](#_Toc1933816)

[A1044 12](#_Toc1933817)

[A1093 13](#_Toc1933818)

[A1101 13](#_Toc1933819)

[A1069 13](#_Toc1933820)

[A1104 14](#_Toc1933821)

[A1049 14](#_Toc1933822)

[A1081 14](#_Toc1933823)

[GCD 14](#_Toc1933824)

[A1089 15](#_Toc1933825)

[A1029 15](#_Toc1933826)

[A1096 17](#_Toc1933827)

[A1059 17](#_Toc1933828)

[数组初始化vector 18](#_Toc1933829)

[随机快速排序 19](#_Toc1933830)

[无序序列中的第K大的值 21](#_Toc1933831)

[STL 21](#_Toc1933832)

[A1060 25](#_Toc1933833)

[Stack的使用 26](#_Toc1933834)

[计算器，中缀表达式和后缀表达式的转换 26](#_Toc1933835)

[Queue的使用 26](#_Toc1933836)

[链表处理 26](#_Toc1933837)

[空间释放 27](#_Toc1933838)

[动态链表和静态链表之间的差别 27](#_Toc1933839)

[静态链表的处理流程 28](#_Toc1933840)

[A1032 28](#_Toc1933841)

[A1052 29](#_Toc1933842)

[A1074 29](#_Toc1933843)

[A1097 29](#_Toc1933844)

[A1063 29](#_Toc1933845)

[C++ String 和 数值类型的变化 29](#_Toc1933846)

[A1100 30](#_Toc1933847)

[A1054 30](#_Toc1933848)

[字符串分割处理问题 30](#_Toc1933849)

[C库函数，字符串处理 31](#_Toc1933850)

[A1051 32](#_Toc1933851)

[A1056 32](#_Toc1933852)

[DFS 32](#_Toc1933853)

[思考DFS BFS DP 贪心 分治 33](#_Toc1933854)

[TREE 33](#_Toc1933855)

[Typedef struct 和 struct 34](#_Toc1933856)

[非递归中序遍历A1086 35](#_Toc1933857)

[A1119 35](#_Toc1933858)

[A1102 35](#_Toc1933859)

[A1090 A1079 A1094（树的静态写法） 36](#_Toc1933860)

[A1064 BST 36](#_Toc1933861)

[A1076和 A1034 36](#_Toc1933862)

[A1130 37](#_Toc1933863)

[并查集缩短路径 37](#_Toc1933864)

[A1107 37](#_Toc1933865)

[图的遍历 38](#_Toc1933866)

[剑指offer 38](#_Toc1933867)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Set |  |  |  |  |  |  |  |  |  |
| 静态链表 | A1032 | A1052 | A1074 | A1097 |  |  |  |  |  |
| String | A1060 |  |  |  |  |  |  |  |  |
| Map | A1100 |  |  |  |  |  |  |  |  |
| TREE | 1020 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

# 逐题解析

|  |  |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 | 根据前序序列和中序序列建立二叉树 |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |
| 30 |  |
| 31 |  |
| 32 |  |
| 33 |  |
| 34 |  |
| 35 |  |
| 36 |  |
| 37 |  |
| 38 |  |
| 39 |  |
| 40 |  |
| 41 |  |
| 42 |  |
| 43 |  |
| 44 |  |
| 45 |  |
| 46 |  |
| 47 |  |
| 48 |  |
| 49 |  |
| 50 |  |
| 51 |  |
| 52 |  |
| 53 |  |
| 54 |  |
| 55 |  |
| 56 |  |
| 57 |  |
| 58 |  |
| 59 |  |
| 60 |  |
| 61 |  |
| 62 |  |
| 63 |  |
| 64 |  |
| 65 |  |
| 66 |  |
| 67 |  |
| 68 |  |
| 69 |  |
| 70 |  |
| 71 |  |
| 72 |  |
| 73 |  |
| 74 |  |
| 75 |  |
| 76 |  |
| 77 |  |
| 78 |  |
| 79 |  |
| 80 |  |
| 81 |  |
| 82 |  |
| 83 |  |
| 84 |  |
| 85 |  |
| 86 |  |
| 87 |  |
| 88 |  |
| 89 |  |
| 90 |  |
| 91 |  |
| 92 |  |
| 93 |  |
| 94 |  |
| 95 |  |
| 96 |  |
| 97 |  |
| 98 |  |
| 99 |  |
| 100 |  |
| 101 |  |
| 102 |  |
| 103 |  |
| 104 |  |
| 105 |  |
| 106 |  |
| 107 | 并查集 |
| 108 |  |
| 109 |  |
| 110 |  |
| 111 |  |
| 112 |  |
| 113 |  |
| 114 | 并查集 |
| 115 |  |
| 116 | Map的使用 |
| 117 | 看不懂这个题 |
| 118 | 并查集 或者是麻烦的set  判断不同集合是否有公共的点，可以理解为，两个图，判断两个图是不是属于一个联通分量，要用并查集 |
| 119 |  |
| 120 |  |
| 121 |  |
| 122 |  |
| 123 |  |
| 124 |  |
| 125 |  |
| 126 |  |
| 127 | 根据前序和后序序列建立二叉树 |
| 128 |  |
| 129 |  |
| 130 |  |
| 131 |  |
| 132 |  |
| 133 |  |
| 134 |  |
| 135 |  |
| 136 |  |
| 137 |  |
| 138 |  |
| 139 |  |
| 140 |  |
| 141 |  |
| 142 |  |
| 143 |  |
| 144 |  |
| 145 |  |
| 146 |  |
| 147 |  |
| 148 |  |
| 149 |  |
| 150 |  |
| 151 |  |
| 152 |  |
| 153 |  |
| 154 |  |
| 155 |  |
| 156 |  |
| 157 |  |
| 158 |  |
|  |  |
|  |  |

# 难点解析

|  |  |
| --- | --- |
| A1022 | 数据结构 输入 map<string,set<int>> |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# PAT经验

1 给的数据不一定全用，A1032

2 存在使用映射关系的题，学会使用map而不是传统的数组

3 一旦出现那种多解问题，一定要认真审题，考虑题目所给出的条件

比如对于A1103，要求的是K的和最大的序列，这就要求你自己提前设置一个Kmax变量

4 scanf对于string类型的数据的读取是有问题的，而且问题很大，所以C++中读取string类型要用cin和空格吸收getchar() ， C中的string类型读取要逐个字符

5 sort《algo》 本身是一个左闭右开的东西

6 double对应的scanf操作符是%lf，而不是%llf

7 string类型是不可以用scanf的，而是也是应该使用cin的。

8 一个非常容易出现的问题就是，原本的单等号被意外的变成了双等号，就很尴尬。

# 常见英文

Suffix 后缀

Prefix 前缀

Deduplication 数据冗余

Distinct 不同

A graph which is connected and acyclic can be considered a tree.

也就是有向无环图可以被认为是一棵树

Infix expression 中缀表达式

Indice 指数 “或者是 exp”

# 贪心

B1042:

逐个读取while(scanf (“%c”,&char\_name )!=EOF)或者是cin.get(array\_name,maxsize)

A1070

本题中需要对price进行排序，但是price如果存储在数组中，而不是unordered\_map中就很难做到二者同时排序，所以用struct。

Struct的地位就是一个算法中的类，一个比较低级的数据结构。

注意:月饼的总库存和总需求虽然给的例子都是正整数，但是最后可能会出现浮点数的情况，这也提示我们，在出现浮点数和正整数的情况下，还是尽量要使用一个浮点数的格式来存储而不是使用正整数的格式来存储。

A1033

1我们只关心当前车站的下一个车站，如果下一个车站和当前车站的距离超过了最大距离，退出·。

2如果下一个车站比我当前的车站的油价低或者相等，继续开车到下一个加油站。

3如果下一个车站比我当前的车站的油价高，那就要继续寻找下一个车站，回到循环12中。

有两种做法，一种做法是给汽车一个油量的设置，另外一种做法是我们只考虑从当前车站到下一个车站的距离。这是两种是不一样的。

A1037

首先对两个东西进行排序，从小到大的顺序进行排序，首先是指针从左边的负数部分进行排序，然后是指针从右边的非负数部分进行排序，

用最小的组最小的，用最大的组最大的，肯定是没错的。不要想太多。

A1067

从头开始遍历，只要不是正确的位置就进行交换，但是不管交换过来的是啥，都不管了，只要去遍历下一个就行了。

注意，如果只可以用swap当然可以这样，但是你请注意，你只可以用swap(0,\*)来进行交换。

在while循环中

如果0不在0号位置，在index位置，就寻找arr，寻找值为index的数字和0进行交换。

直到0在0号位置的时候退出循环

遍历所有的元素，看看是不是index和元素值都相等。如果有不相等的情况，就返回到while循环。

所以这个程序最少要三层循环。也许用STL的vector的find可以加速寻找。

A1038

似乎对于string这种东西，想要输入进去的方式就是使用cin这种东西。

字符的比较一定要注意，char ans[maxn]

你用 ans[0]==0 就是错的，你用ans[0]==‘0’就是对的

A1085

Vector<int> array(size,0);

表示创建一个size大小的vector并且将所有的元素初始化为0

本题可以用二分也可以用双指针，

双指针的方法就是一个指向最小值一个指向最大值，然后二者

在c++中的数据结构中，int和long是等价的，然后long long是另外一种形式。

Int和long都是

A1010

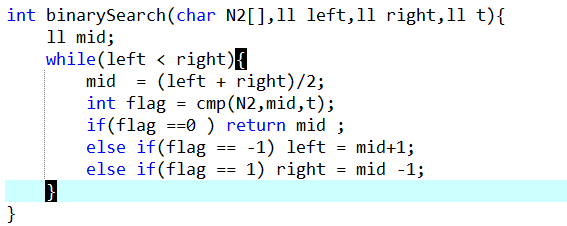
这个东西的进制不是36封顶，你的数值可以是35封顶，但是进制完全可以更大。

关于进制的范围，下界是N2中最大数字+1，上界呢？上界说不好，如果你的N1是INT\_MAX，而你的N2只有一个数，那么上界至少应该是INT\_MAX,所以上界的范围是max(下界,N1)+1;

下一步，所谓的二分查找也单纯是提高速度，更加关键的问题是long long越界问题

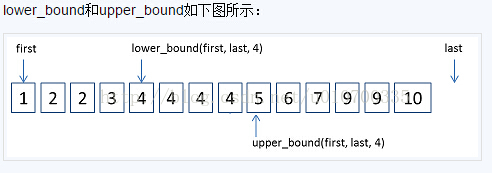
那就是，有些数值，如果用十进制表示，即使是longlong也许都会越界。

你把一个36进制的数值转化为10进制，如果36进制的数值足够长，即使是longlong也无法承受转换之后的结果，这种情况下输出impossible。【但我个人认为其实不是imposible，其实如果不转化为10进制而是统一转化为36进制，肯定可以解】



你的进制越大，你最后转化的数值也就越大，如果转换的结果大于N1，那就是进制高了。需要让top = mid-1；反之是，bottom = mid+1;此外一个最大的循环就是while(left <= right)

long long inf = (1ll << 63) -1; 代表了long long的最大值



Lower返回的是第一个不小于（>=）你的key的元素的指针，upper返回的是第一个大于（>）你的key的元素的指针。

A1044

就是在数组中选择最小的连续数值，选择连续序列。跟二分查找的关系就是，从左往右进行遍历，对每一个数进行遍历。

----------------------------------------------------

以第i位（1≤i≤N）作为开头的有N个子串，而其中至多只有一个子串是符合条件的   
**因为：子串和是单调递增的（所有项都是正整数），而要求是找到和大于等于给定数字，且和最小的子串，因此第一个使和达到要求的子串便是以这位数为开头的符合要求的子串**

----------------------------------------------------

以这种算法的话，最坏情况下复杂度是O（n²），而这道题对运行时间有严格要求，如果将序列改写成增量数组的话，便可以使数组成为有序的，因此可以用二分查找，使程序时间复杂度降到O（nlgn）

---------------------------------------------------

简单来说，我觉得，我觉得，其实二分是基于顺序的，你顺序查找肯定是要从左向右进行一波遍历的，你二分也要从左到右进行遍历。

--------------------------------

首先寻找一遍是不是存在理想的状况，其实只要有理想的（0），就可以剪枝操作。

然后再次进行for循环，寻找到符合S的，对于当前的元素合适的上界。(STL upper\_bound)

--------------------------------

对于upper\_bound和 lower\_bound的使用要很熟悉很熟悉包括algo自带的binary\_search和 手写的二分查找的双闭 双开 左闭右开

16 15

3 2 1 5 4 6 8 7 16 10 15 11 9 12 14 13

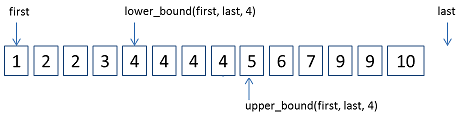
实践证明，完全是可以用algorithm的upper\_bound的。

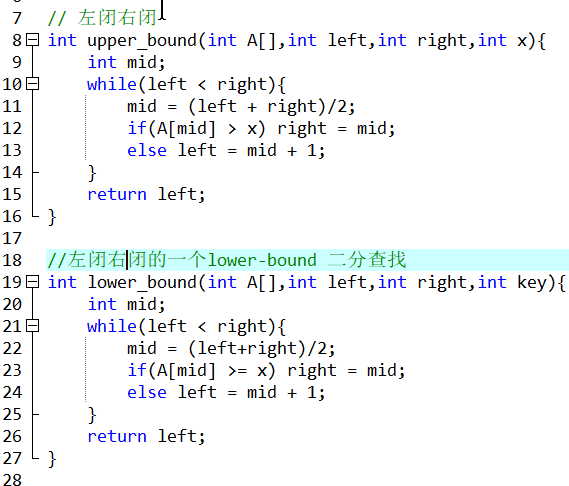
2019年1月19日23:46:04

2019年1月22日18:44:42

Lower-bound 第一个大于等于key的元素指针

Upper-bound第一个大于key的指针





寻找第一个满足条件的点的问题，就是二分查找，不管是lower\_bound的大于等于，还是说upper-bound的大于，其实都是二分查找问题。

2019年1月22日20:15:24

# 归并排序的递归和非递归实现

归并排序，mergeSort首先将整个序列递归的分成大小为1的序列，之后进行merge

Void merge(int A[], int L1, int R1, int L2, int R2)

Void mergeSort(int A[], int left, int right)

非递归的形式主要是基于步长从2开始到step/2 > n为结束的

其实就是不断的进行归并，重复性操作非常多。简化的写法可以借助algo的sort进行。

2019年1月25日13:02:34

# A1044

还是A1044 二分查找问题，本质上是存在性的问题。

//upper\_bound寻找的是大于key的元素位置，左闭右开，返回指针

//Lower\_bound寻找的是大于等于key的元素位置，左闭右开，返回指针

A1048

第一种:构造hash数组，统计硬币的分布情况，并且进行顺序遍历查找

第二种：对硬币序列进行排序，判断是不是存在这个硬币

注意lower\_bound的使用有一个左边界，有一个右边界，还有一个key值。

其中的左边界对于lower来说一定不要是0，而是要当前的元素+1的位置

二分查找，mid = left+ (right – left)/2；对于奇数位于正中，对于偶数，位于偏左的位置。

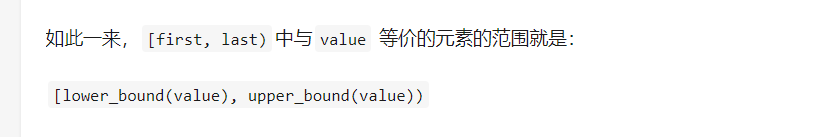
二分查找要记住，left+(right-left)/2，奇数情况正中间，偶数情况偏左边。

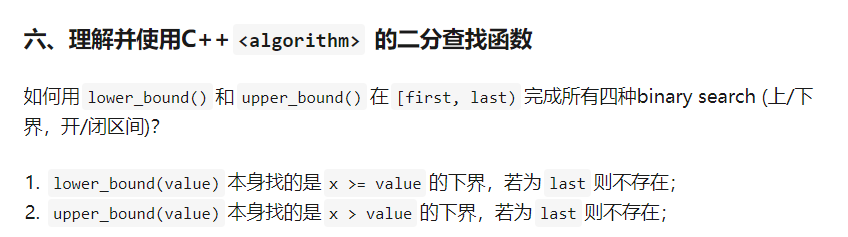
寻找左边界和中间情况的时候，直接用这个mid就行。

但是如果是寻找右边界，就需要mid = l + (r-l)>>2 +1;让中间位置无论是奇数还是偶数都是偏右的.



左闭右开的区间中，和要寻找的key值等价的元素的范围就是lower\_bound(key)和upper\_bound(key)，lower返回的是第一个大于等于key的，uppper\_bound返回的是第一个大于key的。也就是说满足要求的区间也是左闭右开的。





**所以，不要使用STL的binary\_search,因为本身lower和upper就是二分，binary\_search单纯用于判断有没有相同的元素，**

# A1093

本题是套路题

# A1101

对每一个元素进行遍历，填补leftmax和rightmin这两个数组。

# A1069

String的两大操作，转int的stoi，stof，stoll

以及转string的 string str = to\_string(int number);

以及string.insert(int position,const& char[] ,int length);

//其中的length是要插入的长度

string str(string str, int position);

string str(char ch [], int length);

string str(string str, int position, int length);

string string.substr(int initial\_position, int length);

string string.substr(int initial\_position);

string.insert(int position, string str, int length);

string.erase(string.begin()+10, string.end()-10);

//也就是删除从开始10号到结束10号的所有的元素

String.erase(int position, int length);

//也就是从指定的位置开始进行删除

//后面追加

String.append();

String.assign(another\_string, int init\_position,int length); [3]

String.assign(iter\_begin, iter\_end); [2]

# A1104

是不是可以首先计算累计的和？就是从第一个元素到当前位置的所有的元素的和。

# A1049

数学规律

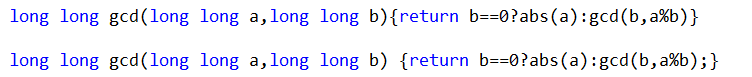
# A1081

分子可以大于分母

但是处理办法肯定是首先都加进去，所以数据结构要使用long long，甚至ll都可能不够

cstlib 中的gcd需要使用，专用的最大公约数函数

# GCD

最大公约数，的C++内置函数是\_\_gcd（a,b）  


# A1089

判断是插入排序还是归并，其中给了原始的序列和其中某一中间过程的序列。

输出的要求不是最终的结果，而是将当前的序列再运行一次，得到下一个中间过程。

要求最后不要出现任何的换行符号。

判断merge sort，肯定是要进行不同步长的一个判断，首先是判断每两个之间是不是存在着顺序，然后是每4个之间，以此类推。

但是当前步长符合merge sort不能说一定不是insert sort。

做法：要进行两种插入方式的模拟。

其中：insert可以变成不同长度的algo的sort。

Merge个人认为可以写一个函数。

对于vector使用数组的输入方式的前提是你给vector分配了空间

否则应该是用pushback，然而pushback并不快

还有一种方式是首先输入数组，然后用数组对vector进行初始爱

要注意一种情况就是如果已经是最后的结果了，那么久没必要进行排序了。就是剪枝判断。

【思路】

首先判断是不是insert sort，然后判断是不是一个merge sort。

对于insert sort，判断是不是会出现一个

使用内置sort进行merge sort的话，需要使用sort(I,i+2\*gap)这种东西，

**数组的数值的拷贝使用的是memcpy函数**

**数组数值的比较推荐使用memcmp函数**

# A1029

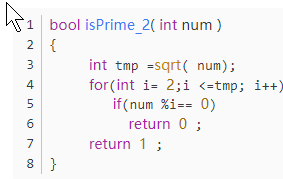
**主要是将两个东西都读进来然后排序肯定会超内存**

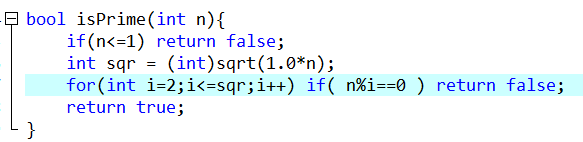
**所以，既然我们知道了元素的总数目，完全可以两个数组，然后两个指针，两个数组分别进行排序，然后分别进行统计。**

**B1007**

**首先只有负数有资格称为一个素数。**

**上述判断方法，明显存在效率极低的问题。对于每个数n，其实并不需要从2判断到n-1，我们知道，一个数若可以进行因数分解，那么分解时得到的两个数一定是一个小于等于sqrt(n)，一个大于等于sqrt(n)，据此，上述代码中并不需要遍历到n-1，遍历到sqrt(n)即可，因为若sqrt(n)左侧找不到约数，那么右侧也一定找不到约数。**

**·**



**A1015**

**需要一个将十进制转化为指定进制的函数**

**需要一个将指定进制转化为十进制的函数**

**需要一个判断素数的函数**

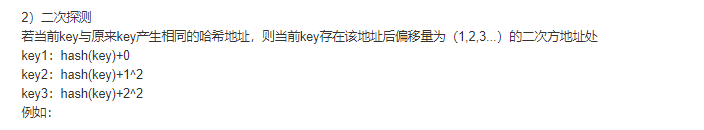
**由于是需要进行翻转，翻转是什么意思，比如你37首先获取的是7，然后你可以顺便构造73这个数。**

**但是对于二进制呢？其实也是一样的啊，唯一的就是你的乘数和除数不是10而是一个规定的值**

**本题没有给出例子的数目，所以需要使用while---EOF来判断**

**A1078**

**本题的hash二次探测法，**



**据说本题是需要mod size的，而且你的add从1到size-1，不是无上限的加。**

**注意：**

**双等号跟等号的作用很不易一样，会引起编译器无法察觉的错误。很要命，所以要求等号和双等号明确的写出来。**

# A1096

**首先是最多12个数字，最少一个数字（它本身？）**

**然后免不了要暴力的。**

**Prime就是素数的意思**

# A1059

本身就是短除法，实现的方式，外层循环判断是不是大于1，内层循环不断的从2开始进行除法。放心，2不能除，4肯定除不了。

短除法———统计法———输出法



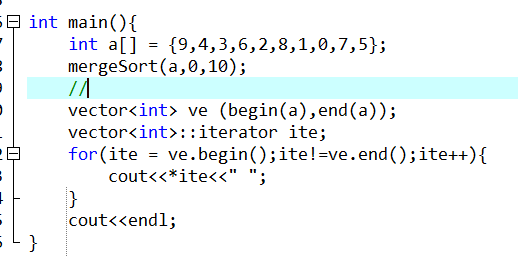
**注意：char array[maxn] 是可以直接用cin的，并且大小不是maxn而是有效数据范围**

**归并排序流程**

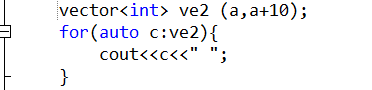
**首先进行分解，用递归的方式分解到一个元素，——然后就是归并的操作。**

# 数组初始化vector

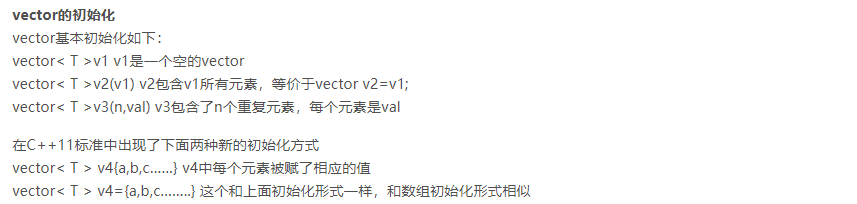
构造函数式：



简化构造函数式：



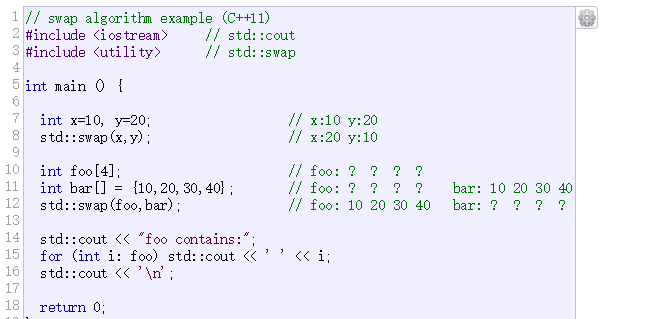
**vector 对象不能用来初始化数组** ，这个过程是不可逆的。



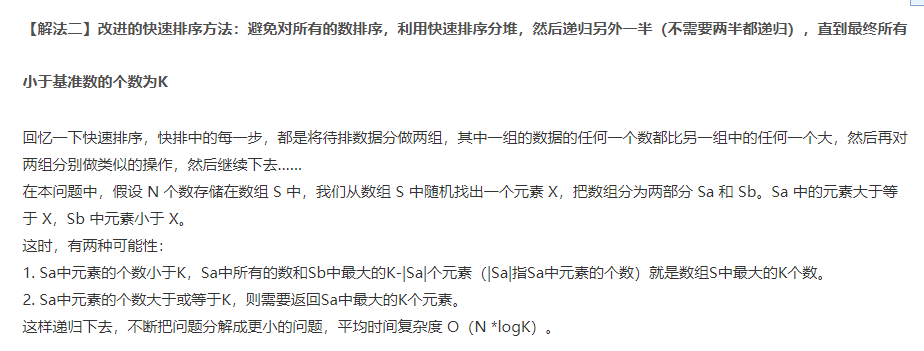
# 随机快速排序

快速排序之所以在完全有序的时候是最差劲的，就是因为，每次都是固定让第一个元素作为主元（或者是枢轴）。所以改进版本就是主元随机选择。

其实就是生成一个left ~~right范围里面的随机数，然后跟a[left]进行交换



# 无序序列中的第K大的值



**注意，字符数组的scanf不需要使用&地址引用符号，而一般的数组必需的。**

**此外，即使是char[]，也可以使用cin，而不一定是要用string**

**模拟加减法主要的就是：要考虑到进位的问题。**

**加法**

**处理进位，从低位开始，————处理额外多出来的进位**

**减法**

**处理借位，从低位开始，————最后将高位的0，改变len的长度**

**乘法**

**处理进位，从低位开始，————处理额外多出来的进位**

**除法**

**结果的len和a的len相等，从高位开始计算**

**A1013**

**在删除了某一个点和相连接的边之后，为了保持图的连通性，应该增加哪些边。**

# STL

**对于所有的STL容器，size是一个通用的函数，其作用就是返回元素的个数**

**基于vector实现邻接表**

**set的作用——自动去掉重复元素 自动排序**

**可以用iterator遍历vector set string**

**容器元素的·通用遍历方式是那个经典的迭代器for循环，除此之外对于string 和 vector可以使用数组方式进行遍历**

**Erase删除区间元素，删除固定元素，根据迭代器进行删除**

**Erase也可以根据元素的值直接删除对应的元素**

**Find根据数值二分查找到对应元素的迭代器**



**Set用于解决需要去掉重复元素，但是由于元素的类型难以直接开数组的情况，【这时如果使用struct的后果就是判断重复的操作过于复杂】，即使是使用struct也是非常难受。**

**所以，使用set可以接收复杂类型元素，自动去掉重复。而排序的功能其实用algo的cmp也是可以解决的。**

**String和vector是STL中唯二可以用数组和迭代器遍历的方式**

**String**

**string 不要赋值为NULL 进行初始化，也就是 不要出现String str = NULL 这样的话**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **遍历** | **作用** | **特殊** | **函数** |
| **Vector** | **下标 迭代器** |  |  |  |
| **Set** | **迭代器** | **复杂类型元素去掉重复** | **Find(value)函数** |  |
| **String** | **下标 迭代器** |  | **可以使用+=进行拼接**  **可以使用>= == != <= < > 比较字典序的大小** | **Substr（pos,length）**  **Str2.find(str1) 也就是在str2中找到str1所在的位置**  **Replace(pos, len, str2) 表示从pos位置，将len长度的元素替换为str2的元素** |
| **Map** | **特殊访问方式** | **建立复杂元素类型的key-value映射关系** |  | **Find函数** |
| **Queue** |  | **先进先出** | **只能用front()和back()进行操作** | **Pop和front使用前判断empty** |
| **Priority\_queue** |  | **Heap，优先级高的在上面** |  |  |
|  |  |  |  |  |

**所有的大小都使用size进行判断**

**Erase对于vector可能是删除某一个数值的元素，对于string却是删除某一个位置之后的元素**

**Map<set<int>, string> mp;**

**Map<string ,int> mp;**

**Map也可以用迭代器进行访问**

**Map<string,int> str2int 其中str2int[string s] ==0表示不存在这一对键值关系**

**也就是说map<key,value> 的键值关系表现为map[key]是不是为0**

|  |  |  |  |
| --- | --- | --- | --- |
| **STL erase的使用方式** | | | |
| **Vector** | **删除某一个迭代器所指向的元素** | **删除某一个区间内的所有元素，迭代器** |  |
| **Set** | **删除迭代器所指向的单个元素 FIND** | **根据某一个具体的元素值删除元素** | **删除某一个区间内的所有元素，迭代器** |
| **String** | **删除迭代器指向的单个元素** | **删除迭代器区间元素** | **删除从某一个迭代器开始的，某一个长度的元素**  **Str.Erase(pos,length);** |
| **Map** | **删除单个元素，迭代器** | **删除单个元素，按照数值进行寻找** | **删除区间内的所有元素，左右迭代器** |
|  |  |  |  |
|  |  |  |  |

**Clear函数在STL中表示将容器清空**

**Map<typename1,typename2> mp;**

**map<typename1,typename2>::iterator it = mp.begin() ;**

**printf(“%c %d\n”,it->first,it->second);**

**//用这种->方式，**

**Find（key）返回的是指向key的迭代器**

|  |
| --- |
| **Size方法是通用的 ，对于string还有strlen** |
| **Erase方法是通用的** |
| **指向键值对的指针，其打印方式，使用->first 和 ->second这两种方式** |
| **Count 对于map和set寻找的是目标数值对应的元素的个数** |
|  |
|  |
|  |

|  |  |
| --- | --- |
| **Queue的作用** | |
| **Push** | **在队首插入元素** |
| **Front back** | **获取元素的值** |
| **Pop** | **命令fornt元素出栈** |
| **Empty** | **检测是不是空的** |
| **使用front和pop首先要使用empty进行判断** | |
| **广度优先搜索借助的是队列，深度优先搜索借助的是stack** | |

|  |  |
| --- | --- |
| **Priority\_queue 优先队列** | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **STACK #include<stack>** | |
| **Top** | **访问栈顶元素** |
| **Push** | **入栈** |
| **Pop** | **出站** |
| **Empty** | **判空** |
| **Size** | **大小** |
|  |  |

|  |  |
| --- | --- |
| **#include<utility>** | |
| **Pair<string,int > p(“haha”,5)；** |  |
| **P = Make\_pair(“haha”,5);** |  |
| **Cout<<p.first<<” “<<p.second<<endl;** |  |
| **可以使用 == >= != < > 进行判断** | **比较原则：首先比较first，然后比较second** |
| **用处：代替二元结构体** |  |
| **对map进行插入** |  |

|  |  |
| --- | --- |
| **#include<algorithm>** | |
| **Max(x,y) min(x,y) abs(x)** |  |
| **Swap(x,y);** |  |
| **Reserve(it1,it2)** | **将it1和it2范围内的元素进行反转** |
| **Next\_permutation** | **给出全排列的下一个序列** |
| **Fill(it1,it2,value)** | **跟memset功能类似** |
| **Sort()** | **需要cmp函数的辅助帮助** |
| **Sort包括对数组，对结构体的排序，也可以对vector string 进行排序** |  |
| **Lower\_bound(first, last, value)** | **也就是大于等于value的第一个元素，也就是左界，闭** |
| **Upper\_bound（first，last，value）** | **大于value的最后一个元素，也就是有界，开** |
|  |  |

|  |  |
| --- | --- |
| **STACK** | |
| **Push** | **进栈** |
| **Pop** | **出栈** |
| **Empty** | **判断是不是空的** |
| **Top** | **输出栈顶元素** |
| **Clear** | **清空站内元素 top = -1；** |
| **Size** | **判断栈内的元素个数** |



**这句话是标准的清空栈的方式**

**对于stack queue set等等pop或者是top之前要判断是不是empty。**

**Memset其实是只能对int赋值0和 -1 ，对于bool也是只能true和false，其中设计了计算机编码问题**

|  |  |
| --- | --- |
| **#include<set> 不仅仅是去除重复的元素，而且自动排序递增序列** | |
| **Insert** |  |
| **Find** | **查找规定的value，返回一个迭代器** |
| **Upper\_bound** | **上界元素 大于 规定的元素值** |
| **Lower\_bound** | **下界元素 大于等于 规定的元素值** |
|  |  |
|  |  |
|  |  |

**Map.find(value) != map.end() 是用于判断map中是不是存在某一个元素的映射的很好的方法。**

# A1060

**最笨的方法，首先判断小数点前面有没有非零的数字，从而判断是大于1的还是小于1的，然后分别处理。**

# Stack的使用

# 计算器，中缀表达式和后缀表达式的转换

**不同数据结构在Cal和Change中的作用**

**计算器计算过程中要注意的细节：**

**Cal首先弹出temp2，然后弹出temp1**

**预处理汇总**

# Queue的使用

**Pop get\_front get\_rear使用之前都是需要有一个判空操作的**

**Queue的清空队列，可以使用while循环进行操作**

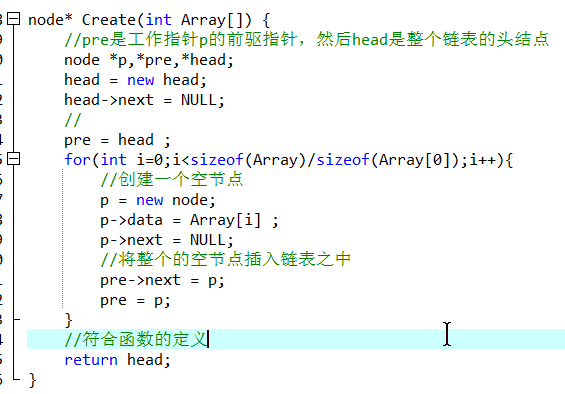
**Queue的数组实现和链表实现，主要是需要使用两个工作指针front和rear这两个东西。**

# 链表处理

一定要注意，链表的最后的一个东西是指向空节点的。

Node->next = NULL;

用数组创建一个链表



# 空间释放

C malloc 和 free

C++ new和 delete

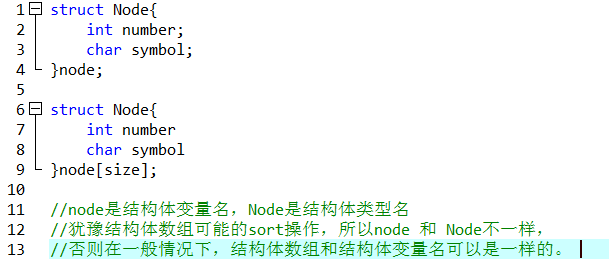
***以上两种操作***

# 动态链表和静态链表之间的差别

动态链表指的是用指针相连的链表，要声明结构体

静态链表？？？

结构体类型 和 结构体变量 和 结构体数组



二者最明显的差别体现在一个是struct作为结点，一个是声明struct [maxn]结构体数组进行操作。

# 静态链表的处理流程

数据结构 包括：地址 数值 下一个 XXX这四部分

1 输入数据

2 遍历吗，处理XXX数据域，XXX数据域主要是用于区分有效结点和 无效结点

很多情况下，数据的节点不一定都是在链表上的节点，对于有效节点要整理到数组的最左边

3 排序，有效结点，无效结点，以及有效节点的二次排序

4 对有效结点的具体处理

这边通用的出现的问题包括：

如果你的头节点指向的元素不存在，你应该怎么办？

# A1032

链表，而且是使用hash的静态链表而不是单纯的动态链表。

为了节省空间，我们可以在储存单词时使用相同的子链表储存共同后缀。题目给出两个链表，找出并输出公共后缀的起始位置或返回-1。

# A1052

静态链表

# A1074

# A1097

需要两个静态链表和一个unorder\_map存储数据出现过几次，或者是一个bool 数组也可

反正数据的大小不超过10000

思路很简单，读取数据，按照有效无效+值进行整理；根据数据的重复与否制作另外一个链表，如果是重复的就把这个节点插入到另外一个链表当中；之后进行两个链表分别的重新排序和重新的标号，最后就是打印了。

此外，两个链表会造成空间和时间的超出。所以，优化的方法包括，增加struct的参数，区分用了的和没有用到的数据。

# A1063

Nc代表的是两个组共享的不同数字

Nt代表的是两个组中出现过的不同数字

# C++ String 和 数值类型的变化

11

8\*13 = 104

# A1100

对于13的倍数·的处理

对于0的长度 是4而不是一般的数组的3的长度的处理问题

**字符串转数字**

字符串中tret字符的位置的不同会对结果产生影响

**数字转字符串**

优先处理0的特情

13的倍数 不应该加上后面的tret，

不要忘记如果字符串有两部分组成，应该加上中间的空格作为分隔符

可不可以在读入的时候进行处理，直接将字符串转变为第三类符号

# A1054

2的24次方是完全可以用 int装下的

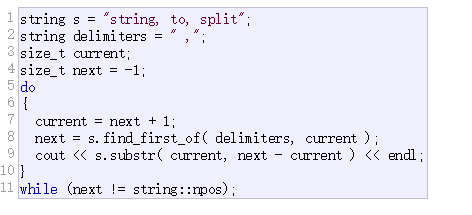
用普通的数组作为hash会超时

所以使用map<int,int >进行优化，也就是用map承担这个计数的工作

本题的核心问题就是映射关系的进阶，也就是 学会用 map<int,int> 建立映射关系

# 字符串分割处理问题

如何处理字符串的分割问题？其实最简单的方法就是按照字母逐个进行处理，如果都是字母，就拼成一个单词，一旦出现了某个分割标志，就重新开始，并且考虑分割表示连续出现的情况。



字符串的分割问题：使用逐个字母的处理方式 string push\_back等等

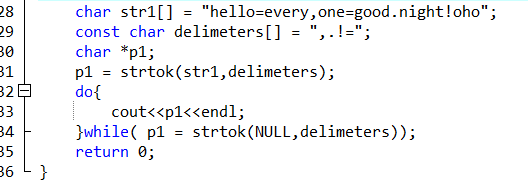
# C库函数，字符串处理

|  |  |
| --- | --- |
| STRCMP |  |
| STRCAT |  |
| STRCPY |  |
| STRTOK | 字符串分割函数，但是会出现空串，注意空串不是NULL |
| STRSTR |  |
|  |  |
|  |  |
|  |  |



Strtok需要使用const char\* 和 char[],返回的是char\*

Strtok第一次调用是str，后续调用是null



# A1051

用三种栈的不同状态进行研究，分别是空 不 不 ；不 空 不；不 不 不

# A1056

其实很好理解，就是题目的输入输出实在是太恶心人了。

其实就是简单的模拟

数据结构包括：

输入顺序 质量 第一次比赛的顺序 排名

结果输出 按照输入顺序进行输出

化繁为简，其实就是你给出输入质量，我给你排名。输入质量对应着排名。没毛病。

此外，要考虑大量存在的平局等等影响。

***所以本题的解决办法就是使用队列，要大量的使用队列***

# DFS

首先确定岔路口和死胡同

然后创建一个stack，用递归的方式进行遍历

递归程序中，进入死胡同的元素是一定要进行一个return的。

岔路口不可以加上return ,死胡同不可以缺少return

对于DFS的递归函数，将整个程序看做是两种东西拼凑起来的结果，一种是岔路口，一种是死胡同。

# 思考DFS BFS DP 贪心 分治

DFS经常是由递归形成，而且DFS感觉就是有点DP的意思

DP属于DFS的一种

也就是说，无论是DP还是DFS都是要考虑——死胡同是什么 岔路口是什么

DFS函数的核心，无非是两个东西，岔路口还是死胡同，岔路口不写return，死胡同不忘return

死胡同也有两种类型，一种是GE，一种是BE

外加两种岔路口，递归函数一共有四部分

深度优先用的是stack，因为要涉及到回溯，如果是那种多解选最符合标准的还好，但是如果要求你记录你的遍历轨迹，那就要声明一个全局的stack进行不断的pop和push了。

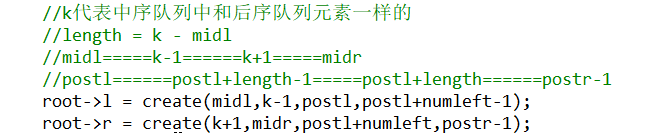
广度优先涉及的是queue，因为我个人感觉，广度优先和DP的关系不大，有时候不需要递归。

# TREE

根据后序遍历和中序遍历构造二叉树

对二叉树进行层序遍历

树一般都要用链表进行解决



# Typedef struct 和 struct





树的建立和恢复

# 非递归中序遍历A1086

只要左孩子不是空的，就入栈

左孩子空了——栈里还有——pop——找右孩子入栈

左孩子空了——栈也空了——结束

# A1119

根据后续和先序序列，判断中序遍历结果是不是唯一的，如果不是唯一的，就随便输出一个中序序列就行。

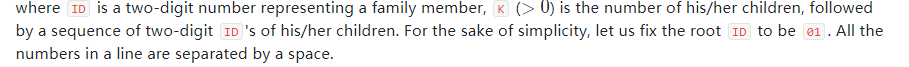
# A1102

# A1090 A1079 A1094（树的静态写法）

都是二叉树的静态，也就是用二维数组记忆二叉树，其中第一个维度代表所有的节点，第二个维度代表节点的子节点情况

而且这几个题都不一定是二叉树，都是多叉树的结构。

对于这个二维的数组，可以有三种实现方式，包括两种vetor 和 一种二维数组



给你一个树的题，第一步要做的就是首先判断到底用静态存储结构还是动态存储结构。

二叉排序树的中序遍历结果就是一个有序的序列，所以说，利用一个有序序列建立的二叉树理论上就应该是一个二叉排序树

# A1064 BST

对于二叉树的中序遍历输出的是一个有序序列，也就是说，中序遍历一个BST，在应该输出的时候进行赋值，就可以把一个存储二叉树的数据结构赋值为一个BST。

本题的二叉树结构其实不一定要进行数组的存储，但是利用数组的存储先对来说是最简单的。因为用动态存储方式，要提前搭建结构，由于我们知道元素的数目，因而在理论上建立链表的树是完全可行的，但是实际上，操作复杂。

当然按照，按照二叉排序树的元素，进行不断的insert建树也是完全可行的。但是不保证是一个完全的树。

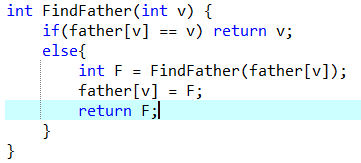
# A1076和 A1034

一个是广度优先遍历，一个是深度优先遍历

# A1130

根据二叉树构建序列，输出的过程其实就是一个前序遍历的过程。

# 并查集缩短路径



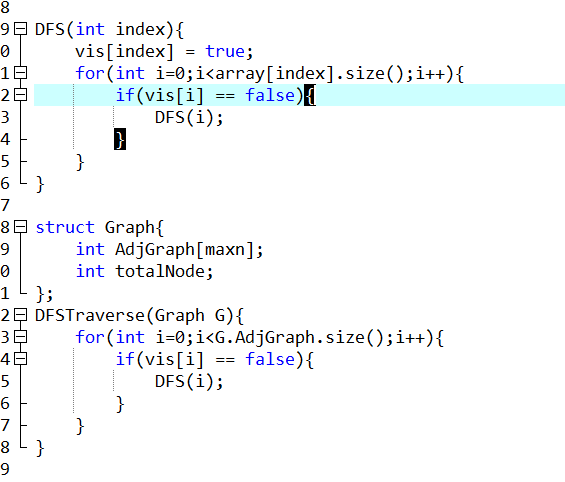
总的目的就是利用迭代，从最底层返回根节点，然后逐层进行赋值

# A1107

一个爱好有一个数组，数组中的元素也就是数组中的一个节点代表一个人。

所谓的社交圈子也无非就是人的集合。

# 图的遍历



# 图的最短距离

Dijstra主要是两个数组，已经遍历的节点和没有遍历过的节点，和一个最小距离数组，

可以用邻接表或者是邻接矩阵进行实现。

除了使用邻接表和使用邻接矩阵进行dij之外

要求你对三种变形有掌握

此外，学会记录pre数组，来对路径的恢复操作。

1 最短路径 + 最多物资

一个是当前城市自己的物资数目，一个是从起点到某个城市可以收集到的物资的数目

2 最短路径+ 路径移动的最小花费（花费跟距离无关）

一个是路径的花费，一个是到达某一个点的花费是多少

3 最短路径+ 最短路径的条数

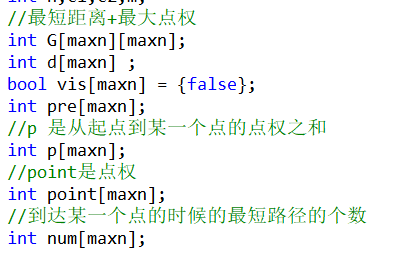
只增加一个——也就是达到某一个点的时候，经历的路径的个数情况

4 最短路径+最多或者最少经过的点

【核心就是，上面的这些东西都是在核心模板距离更新的时候进行更新的，这些附加数组都是根据这个核心距离进行更新的】

牢记dij算法的根本框架就是两个数组一个距离，两个数组中的元素不断的进行移动，然后距离随着每次节点的移动而更换位置。

然后两个数组之间的区分方法就是，按照vis是不是为true进行区分，一开始所有的元素都是vis为true，之后将最小距离的点设置为true，并且逐渐扩大到整个群体，而在d更新的部分，也只对vis为false的进行处理，而不管vis为true的部分。

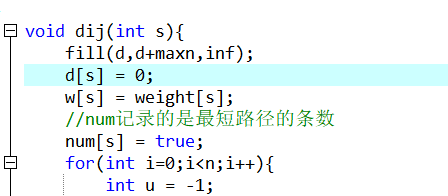


涉及到路径个数，也就是num个数的情况，首先将num[s]初始化为1.其余的路径初始化为0。

涉及到路径花费，初始化c[s]为0，而将其余的东西，比如c[其余]初始化为inf。这样就行了。

Dij中的d数组也是要初始化全部为inf的。因为初始情况认为这几个点之间不存在任何的联系，也就是inf的距离才能到达。要不然每次都是fill(d,d+maxn,inf)是为了啥。你说是不是。

Dij代码中易错的地方还有——vis[s]==false,也就是说，你第一次大循环是对根节点也就是V0节点进行的操作，不能一开始就vis[s]= true;



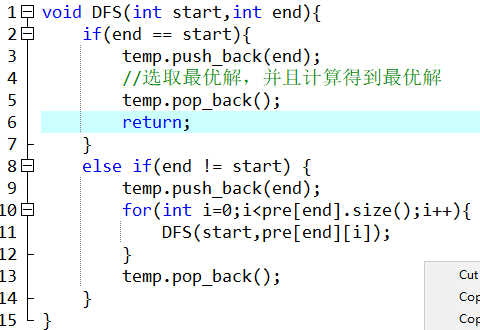
这个地方绝对不允许有vis[s]= true，因为你第一次的操作相当于是对所有的元素进行的初始话。

**A1003 涉及到了点权最大，然后路径最短。以及输出最短路径的个数情况。**

**典型的DIJ算法，包括最短路径的个数num[]，以及最大的点的权重w[]和weight[]这两个东西。这里也就是要求你在dij初始部分 和 d数组更新的部分都要进行注意。**

**由于最短路径的题目经常都是不止一个判断条件，比如在最短路径的同时，还需要某些额外的条件，这也就要求使用Dijstra+DFS进行综合的判断。**

**涉及到递归形式输出最优路径的：首先区分到底是终点还是岔路口，其次，在终点的位置，涉及到全局最优解的比较和确定，在岔路口的位置，涉及到首先push，然后pop的常规的路径记录操作。**



**注意：dij+DFS的目的本来就是为了将第一判据和第二判据分开，这样才方便。**

**注意，如果是最原始的dij，他的pre数组是要首先初始化为自己的。但是如果是dij结合上DFS算法，它的pre是一个二维的数组，也就不需要初始化，相反，初始化会带来错误。**

**其实DFS+DIJ的方法，从理论上是更加适合多路径的，多前缀的。**

**三种变式问题——最短路径+最大边权，最短路径+ 最大点权，最短路径+路径个数。**

**DIJ算法可以拆分成三部分，一部分是初始化，包括fill d 以及 d[s] ==0 ,第二部分是寻找没有遍历过的节点中的最短的距离，采用的方法是for循环+if判断，寻找到的最短距离节点我们设置为u节点，第二部分和第三部分之间，对寻找到的u节点进行判断，如果u节点找到而循环仍在继续，说明是图中的联通分量不止一个。第三部分是根据u节点进行更新，具体表现就是，for循环遍历所有和u相连的节点，并且是没有vis过的节点，之后，进行一个距离的更新操作。这里呢，根据更新情况的不同，合理的更改pre前缀的情况。最后一部分也就是d更改的部分，是可以结合上很多的变式的，是dij算法中变化最大的部分。**

**必须注意的是初始化工作一定要做好，比如对于路径个数数组num，这个数组本身都是0，在初始化的时候num[s] ==1；此外，全篇一共涉及了3次vis数组，其中一次是赋值操作，两次是判断操作，也就是dij核心数据结构中，两个遍历未遍历数组，一个距离数组的体现。**

# 剑指offer

3 一个从左到右，从上到下递增的数组，如果要寻找其中的某一个确定的元素，做法就是考虑三种状态。也是用递归函数进行寻找。

如果当前元素的值大于目标元素，就在当前元素所在行的上边进行寻找。

如果当前元素的值小于目标元素，就在当前元素所在列的右边进行寻找。

寻找的话，固定从整个数组的右上角进行寻找

4 替换空格，首先可以用双指针，如果是要求在原来的数组中进行寻找的话，从前面向后面进行寻找的话，会让空间不够大，由于你涉及到了空间的增加，如果是从后面进行寻找的话，方便实时移动

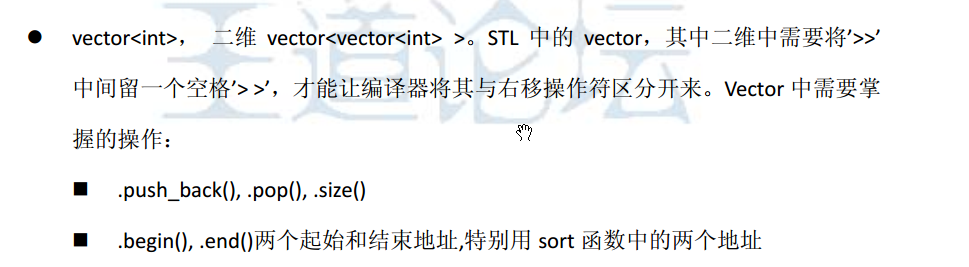
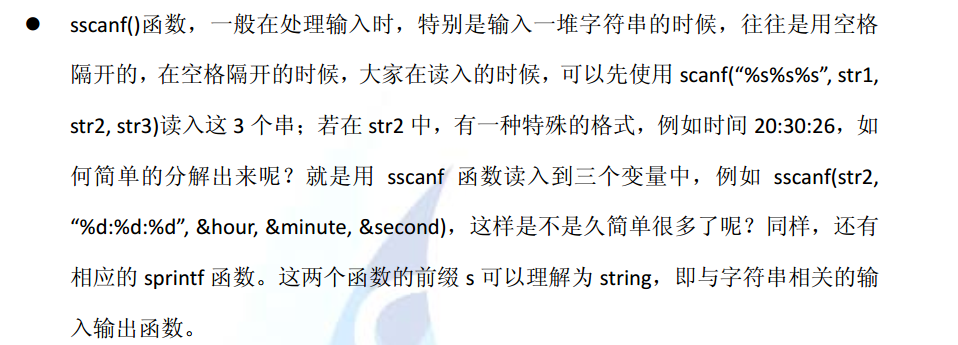
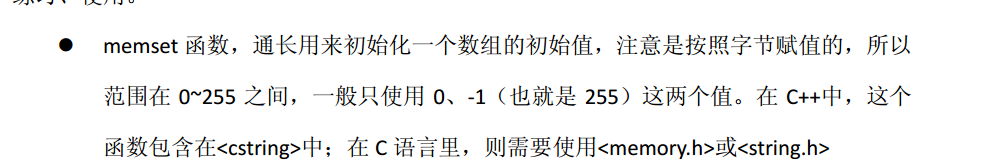
正向遍历，统计空格元素的个数；逆向遍历，将后面的元素直接移动到最终的位置

知道了空格的数目，就知道了替换之后的字符串的总长度，这个时候，元素就可以直接移动到最终的位置，而不需要中间过程

5 反转输出链表，一个是利用stack，在正向遍历链表的时候，将数值存储到链表中，另一种做法是利用递归函数。

6 前序遍历和中序遍历构造一个二叉树，其实就是递归的方式不断划分左右子树直到最终。

# 浙大计算机机试指南



# 图论

最小生成树，克鲁斯卡尔就是一种贪心的算法，因为最小生成树肯定是包括所有的最短边，但是有一个问题随着你不断的加最小边进来，就会导致出现环路的肯定性越来越大，所以每次你增加进来一个环路之后都要判断是不是构成了环路，构成了环路之后，就不能增加这个最小边，而是要选择更大的边进行操作。

然后就是，判断环路的方法很简单，就是使用并查集进行判断。

应该说，Kruskal算法和prim算法都有自己的长处，一个适合点多的，一个适合边多的。Kruskal算法就是ElogV的复杂度，而Prim就是VlogE的复杂度，所以kruskal算法适用于Edge少，Vertex多的，而Prim适用于Vertex多，而Edge少的。

你想啊，krual是不断的把最小的边增加进来，所以kruskal考虑的完全都是边的情况。而PRIM应该说从顶点的情况构建生成树。把所有的顶点集合分成了两个集合，一个是在树中的顶点集合，一部分是不在树中的顶点集合。

由于prim算法是不断的将不在树中的顶点增加到树中的，也就是说，不会出现环路。也可以看出，prim是不考虑任何的边的情况，只考虑顶点的情况。

**为什么PRIM不会构成环**

因为我们在寻找边时，只是找U与V-U中顶点所构成的边，而U中内部顶点的边是不会找的。找到最短边后，直接将另一头顶点加入U了，也就是说这两个顶点都在U中了，即使这2个顶点还有其他路径，后面都不会再找，也就不会构成回路了。

假设一个图G=(V,E)有n个节点，图G的每个节点的出度是一个固定的常数：k。由于E=kV=O(V) ，所以我们把符合E=O(V) 条件的图称为稀疏图。

    同理 ：

如果一个图G=(V,E)有n个节点，假设图G的每个节点的出度是关于n的一个小数，并且0<f<=1，我们把符合E=fV2(平方)=V2(平方)条件的图称为稠密图。

据说：邻接表是表示图的标准方法，原因是稠密图在实际应用中并不多见。

点多就是稠密图，点少就是稀疏图，稠密图用临界矩阵。