# 11，谷歌面试精讲

2018年4月27日 19:51

- 关于面试
- 一些例题
  - 例1 被3和5整除的数的和
  - 例2 合法单词
  - 例3 和0交换的序列排序
  - 例4 放到结尾的序列排序
  - 例5 BFS及其推广
  - 例6 单词对
- 总结
- 关于面试
- 各个公司有没有自己的题库
  - 题库里的题目来源
    - 员工
    - 网络
- 笔试和面试
  - 笔试：没有交流
  - 面试
    - 展现思路
    - 给面试官好印象

- 一些例题
  - 例1 被3和5整除的数的和

```
/*问题： 给定一个数n, 求不超过n的所有的能被3或者5整除的数的和。例如: n = 9，答案
3 + 6 + 5 + 9 = 23。


分析：这是一个数学问题
    首先被3整除的数：3,6,9,...(n/3)*3.一直到n除3往下取整再乘3
    5同理：5,10,...,(n/5)*5
    但要减去重复的数，即同时是3和5的倍数，15,30,...,(n/15)×15
```

我们要注意的点是问面试官n的范围，是否超过int，要用longlong来存储

然后等差数列的求和公式为，设x为首项，y为项数，d为公差
(x + x+d(y-1))*y /2 ,即首项+尾项 × 项数 再处以2(该公式项数为0也适用)
这里我们关键是求得项数
x = 3,d = 3,y = n/3
x = 5,d = 5,y = n/5
x = 15,d = 15,y = n/15

*/

```
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    long long solution(int n)
    {
        long long res = 0;
        res = res + cal(3,n/3,3);
        res = res + cal(5,n/5,5);
        res = res - cal(15,n/15,15);
        return res;
    }
private:
    long long cal(int x,int y,int d)//首项，项数，公差
    {
        long long res = (x + x+d*(y-1))*y/2;
    }
};

int main()
{
    int n = 9;
    Solution s;
    cout<<s.solution(n)<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

- 

- ## 例2 合法单词

/*问题： 字符串只有可能有A、B、C三个字母组成，如果任何紧邻的三个字母相同，就非法。求长度为n的合法字符串有多少个？
比如： ABBBCA是非法，ACCBCCA是合法的。

分析：这道题使用动态规划的思路，
dp[i][0]表示长度为i，最后两位不同的合法字符串个数
dp[i][1]表示长度为i，最后两位相同的合法字符串个数
则dp[i][0] = dp[i-1][0]*2 + dp[i-1][1]*2;//这里前面这块，i-2和i-1不同，因此i处的取法只要不和i-1一样就是合理的，两种取法

然后后面这块，首先i-1和i-2位置都是同一字母，为要求合法且不同，i这里也只有两种取法
dp[i][1] = dp[i-1][0];//这里dp[i-1][1]已经i-2,i-1相同了，再让i-1和i同，明显不合法。
因此只有前面这个，然后前面这个是i-2和i-1不同，然后要求i与i-1同，只有一种选法

初值：从2开始好理解些
dp[1][0] = 3;
dp[1][1] = 0;
dp[2][0] = 6;
dp[2][1] = 3;

结果:dp[n][0]+dp[n][1]

时间复杂度On

*/

```
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    int solution(int n)
    {
        vector<vector<int> > dp(n+1,vector<int>(2));
        dp[2][0] = 6;
        dp[2][1] = 3;
        for(int i = 3;i<=n;i++)
        {
            dp[i][0] = 2*dp[i-1][0] + 2*dp[i-1][1];
            dp[i][1] = dp[i-1][0];
        }
        return dp[n][0]+dp[n][1];
    }
};


int main()
{
    int n = 3;
    Solution s;
    cout<<s.solution(n)<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

- 

- ## 例3 和0交换的序列排序

/*问题： 一个整数组里包含0-(n-1)的排列 (0到(n-1)恰好只出现一次)，如果每次只允许把任意数和0交换，求排好顺序至少交换多少次。 (PAT 1067)

分析：这个问题是组合数学里的圈，比如0占了1的位置，1占了2的位置，2占了0的位置，总能划分为若干个不相交的圈

然后这个圈长度m，如果有0,则最少交换m-1次即可，若没有，则需交换m+1次

代码中我们如何找环呢？

*/

```
#include<bits/stdc++.h>
using namespace std;


class Solution
{
public:
    //bool visited
    int solution(vector<int> a,int n)
    {
        int res = 0;
        vector<bool> visited(n,false);//元素是否被访问过，记录的是元素本身，
        for(int i = 0;i<n;i++)
            res+=count(a[i],a,visited);
        return res;
    }
private:
    int count(int x,vector<int> &a,vector<bool> &visited)//计算每个元素交换次数
    {
        int r = 0;//所在环元素个数
        bool have = false;//环内是否有0
        for(;!visited[x];r++)
        {
            if(x == 0) have = true;
            visited[x] = true;
            x = a[x];//这里很巧妙，x取新的值。比如这里x为3,我们将3加入环中之后，再去取位置3上的值，依次循环，看是否已被加过环，
            //直到遍历到有被访问过得元素时停止，成环


        }
        return have?r-1:(r<=1)?0:r+1;
    }
};


int main()
{
    int n = 12;
    vector<int> a;
    for(int i = 0;i<n;i++)
        a.push_back(i);
    random_shuffle(a.begin(),a.end());
    Solution s;
    cout<<s.solution(a,n)<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

- **例4 给定一个1-n的排列，每次只能把一个数放到序列末尾，至少几次能排好顺序？(O(n)时间内解决的问题（下）)**

```
/*
问题1：给定一个1-n的排列，每次只能把一个数放到序列末尾，至少几次能排好顺序？
        为什么要移动1？其他都排好了，1自然就好了
        如果要移动x，则之后我们必须把(x + 1)，(x + 2).. n都移动到末尾。
        因此，从1-(x-1)必须有序的
        因此我们的目的是找到尽可能大的一个x，让其前面的1-x-1的数都是有序的。


问题2：给定一个1-n的排列，每次可以把一个数放到序列开头，也可以放到结尾，至少几次能排好序？
        分析
        我们可以把1..y移动到开头
        然后把x..n移动到末尾
        但要求[y + 1.. x - 1]必须按顺序出现，因此，我们仍需统计一下多少数是按顺序出现的，这里只不过不用从1开始
        dp[x]表示从x开始在原数组中往后按顺序出现的最长长度
            即dp[x]的值表示：x, x + 1, …x + dp[x] - 1按顺序出现
            倒着循环i,  dp[a[i]] = dp[a[i] + 1] + 1


*/


class Solution1
{
public:
    int solution(vector<int> a)
    {
        int n = a.size(),want =1 ;//want为x的值,初始化为1
        for(int i = 0;i<n;i++)
        {
            if(a[i] == want)//这里want很精妙，相当于在数组范围内找到了所有有序的部分，
            //比如这里，找到了1,就该接着看1后面能不能找到2,一直看找到多少有序的数
                want++;
        }
        return n-want+1;//want,...,n-1都是要被移动的, 相当于n-(want-1)
    }
};


class Solution2
{
public:
    int solution(vector<int> a)
    int n = a.size(),m = 0;//m为最长有序的长度
    vector<int> dp(n+2,0);//使用1,...,n+1,因为数是从1-n的，我们统计的是以数x开头往后的有序长度
```

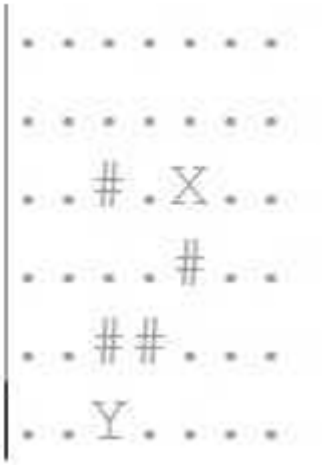
```
    for(int i = n-1;i>=0;i--)
    {
        m = max(m,dp[a[i]] = dp[a[i]+1]+1);//这个dp方程很奇妙，它计算每个以x开头的最大有序长度时，等于以x+1开头的最大有序长度+1。
        //当然这些有序长度最开始均初始化为0的，以 1 3 4 2为例，该方法算得dp2 = dp3 + 1 = 1, dp4 = dp5+1 = 1, dp3 = dp4+1 = 2, dp1 = dp2+1 = 2
        //至于这里为什么以倒序的顺序计算，是因为这样，能在计算dpx时，dpx+1的值是有效的
    }
    return n-m;
};
```

来自 <http://tool.oschina.net/highlight>

- 例5 BFS及其推广
- 例5 给定一个矩阵X表示起点，Y表示终点,#表示墙，从每个位置只能上下左右四个方向走，不能走出矩阵，

  （1）问至少多少步？

  （2）如果允许最多拆3堵墙，至少多少步？

- 
- 分析
  - （1） 很简单，就是直接BFS
  - （2） 枚举拆墙？
    - C($n^2$,3)  O($n^6$)
    - BFS  O($n^2$)
  - 重新构图？
    - 4层的有向图（0，1，2，3）
    - 每一层（相同）

- □ 每个点（包括墙）到它的非墙邻居有边
- □ 注意：墙有出边，无入边

- 第i层到第(i + 1)层 (i=0,1,2)
  - 第i层的任意位置的邻居如果是墙，则有一条从第i层该位置到第(i + 1)层对应墙位置的边。
  - 从第i层相当于“穿墙”到了第(i+1)层，虽然第(i+1)层该位置仍是墙，但是该位置可以出到别的位置。
  - 在这个“立体”图上做BFS
- 节点数$O(n^2)$, 边数$O(n^2)$, 时间复杂度$O(n^2)$

### 例6 单词对

*/*问题： 给定一个字典，找到两个单词，它们不包含相同的字母，且乘积尽可能大，允许预处理字典。*

*分析：*
*方法1：先枚举单词，时间复杂度n^2, 然后如何判断单词之间不包含相同字母？*
*为每个单词设定一个签名，即用2^26次方bitmap表示每个字母是否出现了，出现为1,没出现为0*
*假设单词a签名为x，b为y，那么xy求亦或，若结果为1,那么有重复字母*
*n约为1-2w时，时间复杂度可以接受*

*方法2：预处理单词，将每个单词用整数x = [0,....2……26]表示，二进制转为的整数，比如abc，111都出现，则为7*
*，ac，为101,为5。这里不考虑每个单词出现的次数，只考虑出不出现*

*然后给定状态s，表示单词的字母出现次数可以为s的字迹，比如s = 7（111），则说明只能出现abc，当然abc可以不出现，只要是子集就好*

*如何计算dp[s]?s= 7 ,包括abc，ab，ac，bc，a，b，c.因此，dp[s]的值应为所有这些子集单词中长度最长的。再次重申，s的值表示允许出现哪些字母*

*我们处理的过程如下，首先dp[*]初始化为0.*

*然后对于某个单词的签名x,dp[x] = max(dp[x],len(word)].//有单词的可以填初值了，就是单词的长度，比如abc，dp[7] = 3*

*然后更新s（0, 2^26-1）,s‘为s少一个二进制1的状态，比如*
*s = 10110,s’= 00110,10010,10100,*
*dp[s] = max(dp[s],dp[s’]).即子集的最优解，要么是它的子集中最好的，要么是它本身（预处理好了）*

*那么结果如何取呢？*

*max(len(x)*dp[~x&((1<<26)-1)])//前面表示x这个签名代表的单词的长度，后面表示取反后的子集，并只取前26位*/*

来自 <http://tool.oschina.net/highlight>

- 时间复杂度:
  - 每个单词签名 O(length * n)
  - 计算dp[s]  $O(2^{26} * 26)$
  - 最终求解 O(n)
- 空间复杂度 $O(2^{26})$

- 一定要和面试官交流
  - 不要把面试当成笔试
  - 给面试官积极的情绪
  - 没有标准答案——开放问题
  - 多提假设
    - 函数头部要自己写出
- 无固定套路
- 多总结、思考、归纳
- Goode luck