Zscaler

1. Why Zscaler

Zscaler (/ˈziːˌskeɪlər/) is an **information security company** that provides **network security services** all based on the cloud. Its flagship services, **Zscaler Internet Access**, and **Zscaler Private Access** create fast, secure connections between users and applications, regardless of device, location, or network. Glassdoor rating of 4.7/5.0 + 98% CEO Approval is incredible.

As a new graduate student, I'm always fascinated by new technologies. Working at Zscaler would have opportunities to solve the most challenging and interesting problems on Internet Security. I could learn a lot and making things with my team because I know the culture in Zscaler is amazing. Moreover, the growth of this company is pretty impressive. I believe there will be a lot of growth opportunities for engineers working at Zscaler.

2. Interview Question

1. Integer palindrome

Determine whether an integer is a palindrome.

```
class Solution(object):
    def isPalindrome(self, x):
        """
        :type x: int
        :rtype: bool
        """
        if x < 0:
            return False
        div = 1
        while x / div >= 10:
            div *= 10

        while x > 0:
            l = x // div
            r = x % 10
```

```
            if l != r:
                return False
            x %= div
            x //= 10
            div /= 100
        return True
```

2. Reverse Linked List

```
# Iterative
class Solution(object):
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head or not head.next:
            return head

        dummy = ListNode(-1)
        while head:
            tmp = head
            head = head.next
            tmp.next = dummy.next
            dummy.next = tmp
        return dummy.next
```

```
# Recursive
class Solution(object):
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head or not head.next:
            return head
        p = self.reverseList(head.next)
        head.next.next = head
        head.next = None
        return p
```

3. Reverse Bits

```python
class Solution:
    # @param n, an integer
    # @return an integer
    def reverseBits(self, n):
        res = 0
        for i in range(32):
            res = res << 1
            res = res | (n >> i & 1)
        return res
```

4. First Missing Positive

```python
class Solution(object):
    def firstMissingPositive(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        n, i = len(nums), 0
        if n == 0: return 1

        while i < n:
            # 获取当前位置的数据，减去1是为了得到 在list要插入的位置
            w = nums[i] - 1
            # 0 < nums[i] <= n 判断是否出界
            # nums[i] != nums[w] 如果相等或者是本身就没必要替换了，避免死循环
            if 0 < nums[i] <= n and nums[i] != nums[w]:
                nums[i], nums[w] = nums[w], nums[i]
            else:
            # 当前位置上的数，没有找到合适的位置，进行下一个位置
                i += 1

        for i in range(n):  # 遍历返回
            if i + 1 != nums[i]:
                return i + 1
        return (n + 1)
```

5.Minimum spanning tree

5.1 Prim Algorithm

Prim算法：设图G =（V，E），其生成树的顶点集合为U。

①、把v0放入U。

②、在所有u∈U，v∈V-U的边(u，v)∈E中找一条最小权值的边，加入生成树。

③、把②找到的边的v加入U集合。如果U集合已有n个元素，则结束，否则继续执行②。

```
# Prim's Minimum Spanning Tree (MST) algorithm.
# The program is for adjacency matrix representation of the graph

import sys

class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

 # A utility function to print the constructed MST stored in parent[]
    def printMST(self, parent):
        print "Edge \tWeight"
        for i in range(1,self.V):
            print parent[i],"-",i,"\t",self.graph[i][ parent[i] ]

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minKey(self, key, mstSet):

        # Initilaize min value
        min = sys.maxint

        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v

        return min_index

    # Function to construct and print MST for a graph
    # represented using adjacency matrix representation
    def primMST(self):

        #Key values used to pick minimum weight edge in cut
        key = [sys.maxint] * self.V
        parent = [None] * self.V # Array to store constructed MST
        # Make key 0 so that this vertex is picked as first vertex
        key[0] = 0
        mstSet = [False] * self.V
```

```python

        parent[0] = -1 # First node is always the root of

        for cout in range(self.V):

            # Pick the minimum distance vertex from
            # the set of vertices not yet processed.
            # u is always equal to src in first iteration
            u = self.minKey(key, mstSet)

            # Put the minimum distance vertex in
            # the shortest path tree
            mstSet[u] = True

            # Update dist value of the adjacent vertices
            # of the picked vertex only if the current
            # distance is greater than new distance and
            # the vertex in not in the shotest path tree
            for v in range(self.V):
                # graph[u][v] is non zero only for adjacent vertices of m
                # mstSet[v] is false for vertices not yet included in MST
                # Update the key only if graph[u][v] is smaller than key[v]
                if self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:
                        key[v] = self.graph[u][v]
                        parent[v] = u

        self.printMST(parent)

g = Graph(5)
g.graph = [ [0, 2, 0, 6, 0],
            [2, 0, 3, 8, 5],
            [0, 3, 0, 0, 7],
            [6, 8, 0, 0, 9],
            [0, 5, 7, 9, 0]]

g.primMST();
```

5.2 Kruskal’ s Algorithm

1. 把图中的所有边按代价从小到大排序；
2. 把图中的n个顶点看成独立的n棵树组成的森林；

3. 按权值从小到大选择边，所选的边连接的两个顶点ui,viui,vi,应属于两颗不同的树，则成为最小生成树的一条边，并将这两颗树合并作为一颗树。
4. 重复(3),直到所有顶点都在一颗树内或者有n-1条边为止

```
from collections import defaultdict

#Class to represent a graph
class Graph:

    def __init__(self,vertices):
        self.V= vertices #No. of vertices
        self.graph = [] # default dictionary
                                # to store graph

    # function to add an edge to graph
    def addEdge(self,u,v,w):
        self.graph.append([u,v,w])

    # A utility function to find set of an element i
    # (uses path compression technique)
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    # A function that does union of two sets of x and y
    # (uses union by rank)
    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)

        # Attach smaller rank tree under root of
        # high rank tree (Union by Rank)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot

        # If ranks are same, then make one as root
        # and increment its rank by one
        else :
            parent[yroot] = xroot
            rank[xroot] += 1

```

```
    # The main function to construct MST using Kruskal's
        # algorithm
    def KruskalMST(self):

        result =[] #This will store the resultant MST

        i = 0 # An index variable, used for sorted edges
        e = 0 # An index variable, used for result[]

            # Step 1:  Sort all the edges in non-decreasing
                # order of their
                # weight.  If we are not allowed to change the
                # given graph, we can create a copy of graph
        self.graph =  sorted(self.graph,key=lambda item: item[2])

        parent = [] ; rank = []

        # Create V subsets with single elements
        for node in range(self.V):
            parent.append(node)
            rank.append(0)

        # Number of edges to be taken is equal to V-1
        while e < self.V -1 :

            # Step 2: Pick the smallest edge and increment
                    # the index for next iteration
            u,v,w =  self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent ,v)

            # If including this edge does't cause cycle,
                        # include it in result and increment the index
                        # of result for next edge
            if x != y:
                e = e + 1
                result.append([u,v,w])
                self.union(parent, rank, x, y)
            # Else discard the edge

        # print the contents of result[] to display the built MST
        print "Following are the edges in the constructed MST"
        for u,v,weight  in result:
            #print str(u) + " -- " + str(v) + " == " + str(weight)
```

```
                print ("%d -- %d == %d" % (u,v,weight))
```