使用宽度优先搜索找所有方案

主讲人 令狐冲

一个方案=一条路径

求所有方案=求所有路径

BFS 善于解决求连通块问题

把路径看做点，把路径的变化关系看做点的连接关系

这样就把找所有路径问题变成了找所有连通点的问题

全子集问题

http://www.lintcode.com/problem/subsets

求一个集合的所有子集

画图了解两种不同的搜索树

Java 代码 - 第一种 BFS


```java
public List<List<Integer>> subsets(int[] nums) {
    if (nums == null) {
        return new ArrayList<>();
    }

    List<List<Integer>> queue = new ArrayList<>();
    int index = 0;

    Arrays.sort(nums);
    queue.add(new ArrayList<Integer>());
    while (index < queue.size()) {
        List<Integer> subset = queue.get(index++);
        for (int i = 0; i < nums.length; i++) {
            if (subset.size() != 0 && subset.get(subset.size() - 1) >= nums[i]) {
                continue;
            }
            List<Integer> newSubset = new ArrayList<>(subset);
            newSubset.add(nums[i]);
            queue.add(newSubset);
        }
    }

    return queue;
}
```

把初始节点的 new ArrayList 换成 new LinkedList 行不行？

Java 代码 - 第二种 BFS


```java
public List<List<Integer>> subsets(int[] nums) {
    if (nums == null) {
        return new ArrayList<>();
    }

    List<List<Integer>> queue = new ArrayList<>();
    queue.add(new LinkedList<Integer>());
    Arrays.sort(nums);

    for (int num : nums) {
        int size = queue.size();
        for (int i = 0; i < size; i++) {
            List<Integer> subset = new ArrayList<>(queue.get(i));
            subset.add(num);
            queue.add(subset);
        }
    }

    return queue;
}
```

Python 代码 - 两种 BFS


```python
def subsets(self, nums):
    if not nums:
        return [[]]

    queue = [[]]
    index = 0
    while index < len(queue):
        subset = queue[index]
        index += 1
        for num in nums:
            if subset and subset[-1] >= num:
                continue
            queue.append(subset + [num])

    return queue
```

```python
def subsets(self, nums):
    if not nums:
        return [[]]

    queue = [[]]
    for num in sorted(nums):
        for i in range(len(queue)):
            subset = list(queue[i])
            subset.append(num)
            queue.append(subset)

    return queue
```

二叉树的序列化

https://www.lintcode.com/problem/serialize-and-deserialize-binary-tree

实现 serialize 和 deserialize 函数来序列化和反序列化二叉树

什么是序列化？

将“**内存**”中结构化的数据变成“**字符串**”的过程

序列化：object to string

反序列化：string to object

什么时候需要序列化？

1. 将内存中的数据持久化存储时

内存中重要的数据不能只是呆在内存里，这样断电就没有了，所需需要用一种方式写入硬盘，在需要的时候，能否再从硬盘中读出来在内存中重新创建

2. 网络传输时

机器与机器之间交换数据的时候，不可能互相去读对方的内存。只能讲数据变成字符流数据(字符串)后通过网络传输过去。接受的一方再将字符串解析后到内存中。

常用的一些序列化手段:

- XML
- Json
- Thrift (by Facebook)
- ProtoBuf (by Google)

序列化算法

一些序列化的例子：

- 比如一个数组，里面都是整数，我们可以简单的序列化为”[1,2,3]”
- 一个整数链表，我们可以序列化为，”1->2->3”
- 一个哈希表(HashMap)，我们可以序列化为，”{\”key\”: \”value\”}”

序列化算法设计时需要考虑的因素：

- **压缩率**。对于网络传输和磁盘存储而言，当然希望更节省。
  - 如 Thrift, ProtoBuf 都是为了更快的传输数据和节省存储空间而设计的。
- **可读性**。我们希望开发人员，能够通过序列化后的数据直接看懂原始数据是什么。
  - 如 Json，LintCode 的输入数据

二叉树如何序列化？

你可以使用任何你想要用的方法进行序列化，只要保证能够解析回来即可。

LintCode 采用的是 BFS 的方式对二叉树数据进行序列化，这样的好处是，你可以更为容易的自己画出整棵二叉树。

算法描述：

http://www.lintcode.com/en/help/binary-tree-representation/

题目及解答：

http://www.lintcode.com/problem/binary-tree-serialization/

http://www.jiuzhang.com/solutions/binary-tree-serialization/

Java 代码


```java
public String serialize(TreeNode root) {
    if (root == null) {
        return "{}";
    }

    List<TreeNode> queue = new ArrayList<TreeNode>();
    queue.add(root);

    for (int i = 0; i < queue.size(); i++) {
        TreeNode node = queue.get(i);
        if (node == null) {
            continue;
        }
        queue.add(node.left);
        queue.add(node.right);
    }

    return queueToString(queue);
}
```

```java
private String queueToString(List<TreeNode> queue) {
    while (queue.get(queue.size() - 1) == null) {
        queue.remove(queue.size() - 1);
    }

    List<String> items = new ArrayList<>();
    for (TreeNode node : queue) {
        if (node == null) {
            items.add("#");
        } else {
            items.add("" + node.val);
        }
    }

    return "{" + String.join(",", items) + "}";
}
```

```java
public TreeNode deserialize(String data) {
    if (data.equals("{}")) {
        return null;
    }
    String[] vals = data.substring(1, data.length() - 1).split(",");
    ArrayList<TreeNode> queue = new ArrayList<TreeNode>();
    TreeNode root = new TreeNode(Integer.parseInt(vals[0]));
    queue.add(root);
    int index = 0;
    boolean isLeftChild = true;
    for (int i = 1; i < vals.length; i++) {
        if (!vals[i].equals("#")) {
            TreeNode node = new TreeNode(Integer.parseInt(vals[i]));
            if (isLeftChild) {
                queue.get(index).left = node;
            } else {
                queue.get(index).right = node;
            }
            queue.add(node);
        }
        if (!isLeftChild) {
            index++;
        }
        isLeftChild = !isLeftChild;
    }
    return root;
}
```

Python 代码


```python
def serialize(self, root):
    if root is None:
        return "{}"

    queue = [root]
    index = 0
    while index < len(queue):
        if queue[index] is not None:
            queue.append(queue[index].left)
            queue.append(queue[index].right)
        index += 1

    while queue[-1] is None:
        queue.pop()

    return '{%s}' % ','.join([str(node.val) if node is not None else '#'
                              for node in queue])
```

```python
def deserialize(self, data):
    data = data.strip('\n')

    if data == '{}':
        return None

    vals = data[1:-1].split(',')

    root = TreeNode(int(vals[0]))
    queue = [root]
    isLeftChild = True
    index = 0

    for val in vals[1:]:
        if val is not '#':
            node = TreeNode(int(val))
            if isLeftChild:
                queue[index].left = node
            else:
                queue[index].right = node
            queue.append(node)

        if not isLeftChild:
            index += 1
        isLeftChild = not isLeftChild

    return root
```