二叉树序列化与反序列化

层序遍历方法

```
class Codec {
public:
  // Encodes a tree to a single string.
  std::string serialize(TreeNode *root) {
    if (root == nullptr) {
      return "#,";
    }
    std::string res;
    // 层序
    std::queue<TreeNode *> deque;
    deque.push(root);

    while (deque.size()) {
      TreeNode *tmp = deque.front();
      deque.pop();

      if (tmp == nullptr) { // 必须队列弹出的时候增加，否则就不是层序遍历
        res += "#,";
        continue;
      } else {
        res += std::to_string(tmp->val) + ",";
      }
      deque.push(tmp->left);
      deque.push(tmp->right);
    }

    return res;
  }

  // Decodes your encoded data to tree.
  TreeNode *deserialize(std::string data) {
    int len = data.size();
    if (len == 0) {
      return nullptr;
    }

    std::vector<std::string> str = split(data, ',');

    std::deque<TreeNode *> deque;
    if (str[0] == "#") {
      return nullptr;
    }
    TreeNode *root = new TreeNode(std::stoi(str[0]));
    deque.push_back(root);

    for (int i = 1; i < str.size();) {
      // 弹出头结点
```

```
        TreeNode *tmp = deque.front();
        deque.pop_front();

        // 左节点
        std::string left = str[i++];
        if (left == "#") {
          tmp->left = nullptr;
        } else {
          tmp->left = new TreeNode(std::stoi(left));
          deque.push_back(tmp->left);
        }

        // 右节点
        std::string right = str[i++];
        if (right == "#") {
          tmp->right = nullptr;
        } else {
          tmp->right = new TreeNode(std::stoi(right));
          deque.push_back(tmp->right);
        }
      }

      return root;
    }

  private:
    std::vector<std::string> split(std::string data, char seq) {
      std::vector<std::string> res;
      for (int i = 0; i < data.size();) {
        if (data[i] == seq) {
          i++;
          continue;
        }

        int         j = i;
        std::string tmp;
        while (data[j] != seq) {
          tmp.push_back(data[j++]);
        }
        res.push_back(tmp);
        i = j;
      }

      return res;
    }
};
```

先序遍历方法

```
class Codec {
public:
```

```
    // Encodes a tree to a single string.
    std::string serialize(TreeNode *root) {
      if (root == nullptr) {
        return "#";
      }
      std::string res;
      // 先序
      res += std::to_string(root->val) + ",";
      std::string left  = serialize(root->left);
      std::string right = serialize(root->right);

      res += left + "," + right;

      return res;
    }

    // Decodes your encoded data to tree.
    TreeNode *deserialize(std::string data) {
      int len = data.size();
      if (len == 0) {
        return nullptr;
      }

      std::list<std::string> str = split(data, ',');

      return deserialize(str);
    }

  private:
    TreeNode *deserialize(std::list<std::string> &data) {
      if (data.empty()) {
        return nullptr;
      }

      std::string str = data.front();
      data.pop_front();
      if (str == "#") {
        return nullptr;
      }

      TreeNode *root = new TreeNode(std::stoi(str));
      root->left     = deserialize(data);
      root->right    = deserialize(data);

      return root;
    }

    std::list<std::string> split(std::string data, char seq) {
      std::list<std::string> res;
      for (int i = 0; i < data.size();) {
        if (i < data.size() && data[i] == seq) {
          i++;
          continue;
        }
```

```
      int         j = i;
      std::string tmp;
      while (j < data.size() && data[j] != seq) {
        tmp.push_back(data[j]);
        if (j < data.size()) {
          j++;
        }
      }

      res.push_back(tmp);
      i = j;
    }

    return res;
  }
};
```

后序遍历方法

```
class Codec {
public:
  // Encodes a tree to a single string.
  std::string serialize(TreeNode *root) {
    if (root == nullptr) {
      return "#";
    }
    std::string res;
    // 后序
    std::string left  = serialize(root->left);
    std::string right = serialize(root->right);

    res += left + "," + right + "," + std::to_string(root->val);

    return res;
  }

  // Decodes your encoded data to tree.
  TreeNode *deserialize(std::string data) {
    int len = data.size();
    if (len == 0) {
      return nullptr;
    }

    std::list<std::string> str = split(data, ',');

    return deserialize(str);
  }

private:
  TreeNode *deserialize(std::list<std::string> &data) {
```

```
    if (data.empty()) {
      return nullptr;
    }

    std::string str = data.back();
    data.pop_back();
    if (str == "#") {
      return nullptr;
    }

    TreeNode *root = new TreeNode(std::stoi(str));
    root->right    = deserialize(data);
    root->left     = deserialize(data);

    return root;
  }

  std::list<std::string> split(std::string data, char seq) {
    std::list<std::string> res;
    for (int i = 0; i < data.size();) {
      if (i < data.size() && data[i] == seq) {
        i++;
        continue;
      }

      int         j = i;
      std::string tmp;
      while (j < data.size() && data[j] != seq) {
        tmp.push_back(data[j]);
        if (j < data.size()) {
          j++;
        }
      }

      res.push_back(tmp);
      i = j;
    }

    return res;
  }
};
```

中序遍历方法

中序遍历无法反序列化。