Разбор задач

Гоша: Дерево — сложная структура данных. Кажется, обходить его можно разными способами.

Тимофей: Две основные стратегии обхода деревьев: обход в глубину *(англ. Depth-first search или dfs)* и обход в ширину *(англ. Breadth-first search или bfs)*.

Гоша: А почему их так называют?

Тимофей: Сейчас я расскажу про эти алгоритмы и станет понятно, откуда такие названия.

Начнём с поиска в глубину. Идея этого метода в том, чтобы спускаться от корня настолько глубоко, насколько это возможно. Эту процедуру удобно организовывать рекурсивно.

Начиная от узла N, проделываем такие шаги:

- Рекурсивно обходим левое поддерево. Шаг завершается, когда мы снова попадаем в узел N.
- Рекурсивно обходим правое поддерево. Этот шаг тоже завершается при попадании в узел N.
- Обрабатываем узел N. В нашем случае будем добавлять узел в путь.

Для представления узла дерева используем класс `Node`:

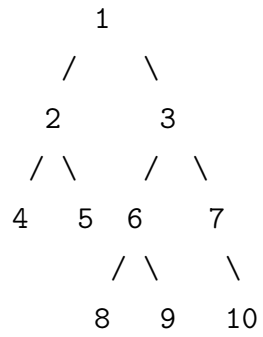
```
class Node:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

    def __repr__(self):
        return str(self.value)
```

Напишем код функции, реализующей описанный алгоритм:

```
def dfs_post_order(node, path=[]):
    if node.left:
        path = dfs_post_order(node.left, path)
    if node.right:
        path = dfs_post_order(node.right, path)
    path += [node]
    return path
```

Допустим, дано такое дерево:



При обходе дерева, который соответствует алгоритму, порядок посещения вершин будет такой: $4, 5, 2, 8, 9, 6, 10, 7, 3, 1$.

Этот метод обхода в глубину называется `post_order`. Обработка узла происходит после рекурсивных вызовов.

Существуют два других алгоритма обхода в глубину.

Можно сначала обработать вершину, а потом запускать рекурсивные вызовы. Такой метод называется `pre_order`.

```
def dfs_pre_order(node, path=[]):
    path += [node]
    if node.left:
        path = dfs_pre_order(node.left, path)
    if node.right:
        path = dfs_pre_order(node.right, path)
    return path
```

При обходе ранее рассмотренного дерева порядок посещения вершин будет таким: $1, 2, 4, 5, 3, 6, 8, 9, 7, 10$.

Последний вариант алгоритма поиска в глубину — `in_order`.

В этом случае сначала производится рекурсивный вызов для левого поддерева, потом обработка узла, затем — рекурсивный вызов для правого поддерева.

```
def dfs_in_order(node, path=[]):
    if node.left:
        path = dfs_in_order(node.left, path)
    path += [node]
    if node.right:
        path = dfs_in_order(node.right, path)
    return path
```

Порядок посещения вершин будет следующий: $4, 2, 5, 1, 8, 6, 9, 3, 7, 10$.

Перейдём к рассмотрению алгоритма обхода в ширину.

При обходе в ширину узлы посещаются уровень за уровнем. Каждый уровень обходится слева направо.

Для реализации алгоритма удобно использовать очередь или дек. Мы воспользуемся деком.

Алгоритм следующий:

Добавляем в пустую очередь корень дерева.

Пока в очереди есть элементы:

- Извлекаем самый левый, то есть самый ранее добавленный элемент.
- Если у этого элемента есть левый потомок, добавляем потомка в очередь.
- Аналогично поступаем с правым потомком.

```
from collections import deque

def bfs(root, path=[]):
    q = deque()
    q.append(root)
    while q:
        cur_node = q.popleft()
        path.append(cur_node)
        if cur_node.left:
            q.append(cur_node.left)
        if cur_node.right:
            q.append(cur_node.right)
    return path
```

Если применить этот алгоритм обхода к рассмотренному ранее дереву, очередность обхода вершин будет такая: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$.