Практическая работа №6

Тема: Структуры данных «Дерево».

Цель работы: Изучить СД типа «дерево», научиться их программно реализовывать и использовать.

Дерево – одна из наиболее широко распространённых структур данных в информатике, эмулирующая древовидную структуру в виде набора связанных узлов. Является связным графом, не содержащим циклы. Большинство источников также добавляют условие на то, что рёбра графа не должны быть ориентированными. В дополнение к этим трём ограничениям, в некоторых источниках указывается, что рёбра графа не должны быть взвешенными.

Для реализации «дерева» в ширину, воспользуемся кодом, представленым ниже:

```
graph = {
  'A' : ['B','C'],
  'B' : ['D', 'E'],
  'C' : ['F'],
  'D' : [],
  'E' : ['F'],
  'F' : []
}

visited = []
queue = []

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    s = queue.pop(0)
    print (s, end = " ")

    for neighbour in graph[s]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

bfs(visited, graph, 'A')
```

Breath-first search (BFS) - это алгоритм, используемый для обхода дерева на графах или структурах данных дерева. BFS можно легко реализовать с помощью рекурсии и структур данных, таких как словари и списки.

Алгоритм:

1. Выберем любой узел, посетим соседнюю не посещенную вершину, отметим ее как посещенную, отобразим ее и встанем в очередь.

2. Если оставшихся соседних вершин не осталось, удалим первую вершину из очереди.

Повторим шаги 1 и 2, пока очередь не опустеет или не будет найден нужный узел.

Реализация была использована для графика, представленного на рисунке 1.

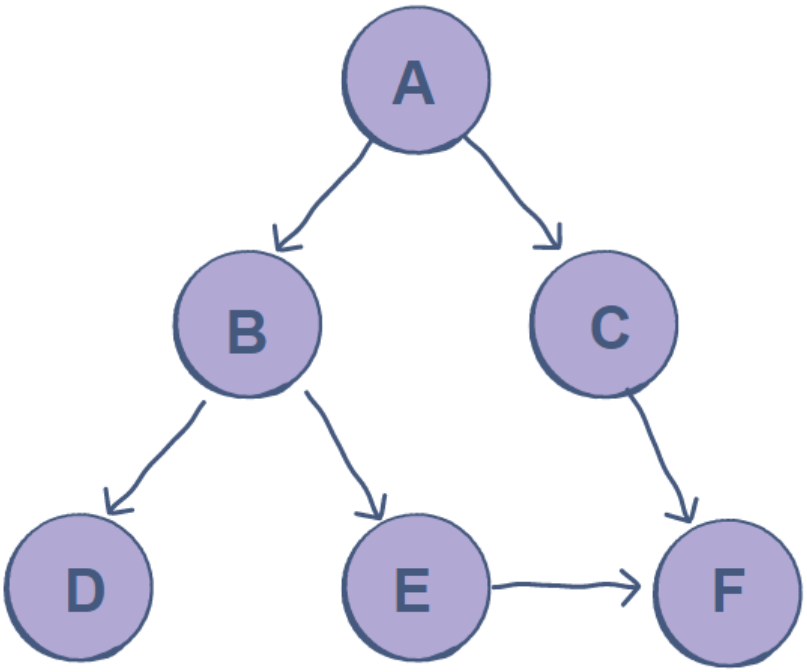

Рисунок 1 – График, который реализован в коде

Диаграмма деятельности для данного кода представлена на рисунке 2.

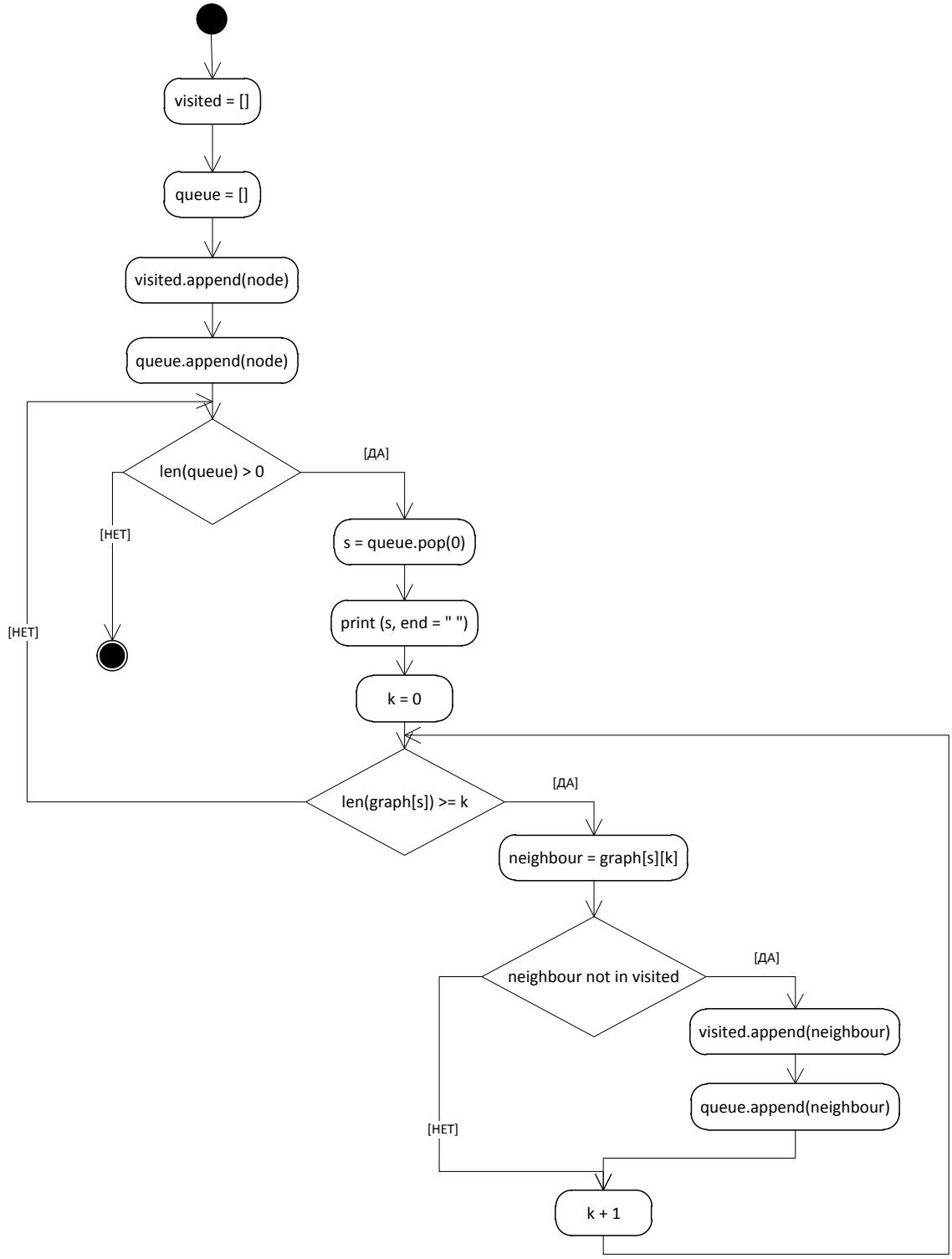

Рисунок 2 – Диаграмма деятельности для реализации дерева в ширину

При помощи функции printPath покажем путь от узла data к узлу parent.

```
def printPath(Data, parent):
    if parent[Data] == Data:
        return
    printPath(parent[Data], parent)
    print(parent[Data], end = " ")
```

Диаграмма деятельности для функции printPath показана на рисунке 3.

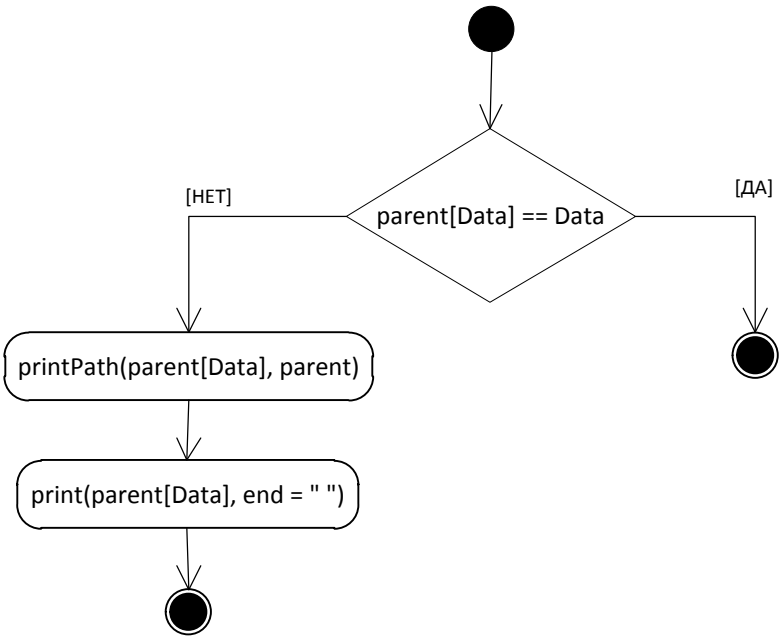

Рисунок 3 – Диаграмма деятельности для функции printPath

При помощи функции leftMostShortest определим самый короткий путь.

```
def leftMostShortest(root):
    q = []
    q.append(root)
    LeafData = -1
    temp = None
    parent = {}
    parent[root.data] = root.data
    while len(q) != 0:
        temp = q.pop(0)
        if not temp.left and not temp.right:
            LeafData = temp.data
            break
        else:
            if temp.left:
                q.append(temp.left)
                parent[temp.left.data] = temp.data
            if temp.right:
                q.append(temp.right)
                parent[temp.right.data] = temp.data
    printPath(LeafData, parent)
    print(LeafData, end = " ")
```

Диаграмма деятельности для функции leftMostShortest показана на рисунке 4.

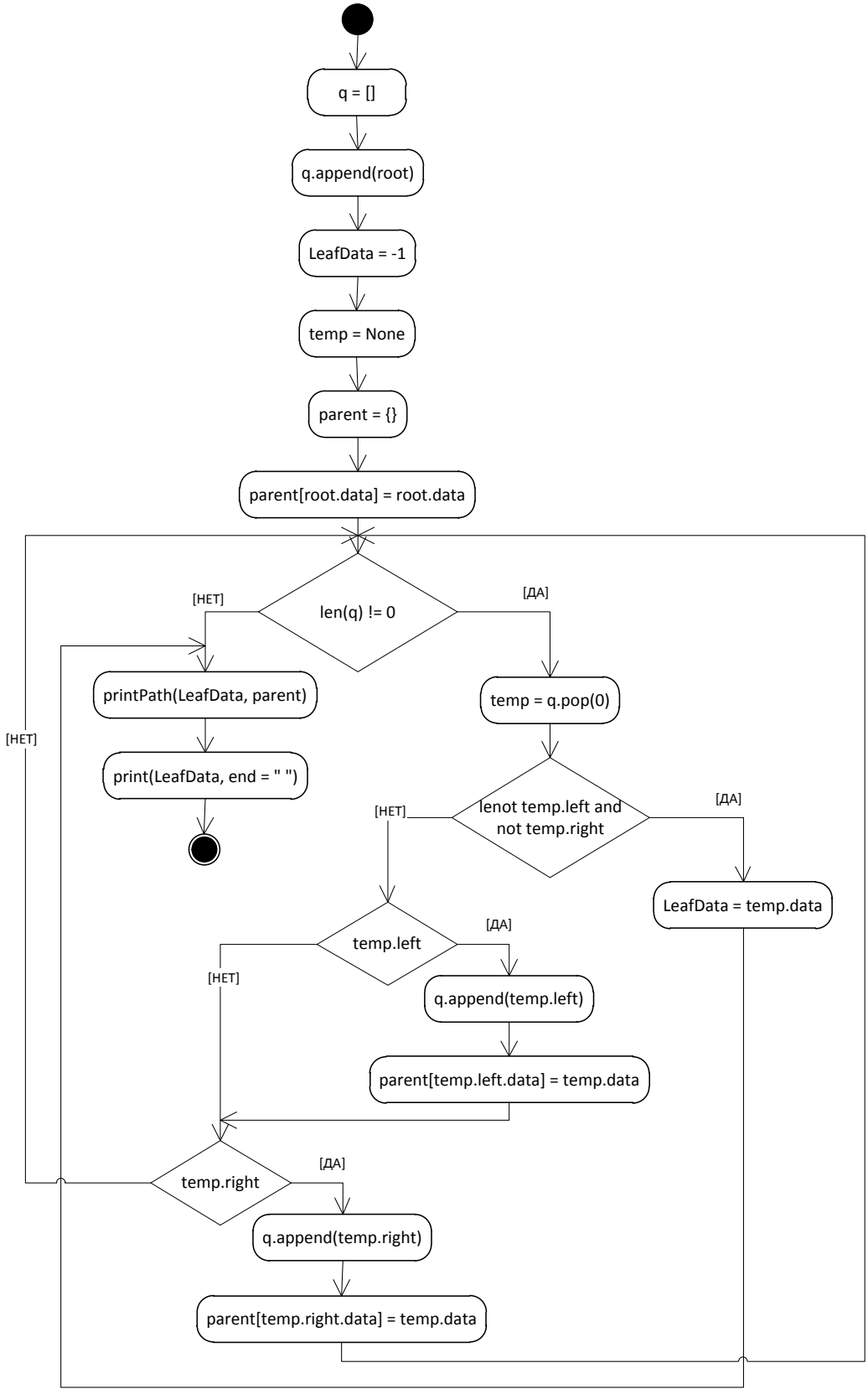

Рисунок 4 – Диаграмма деятельности для функции leftMostShortest

При помощи кода, представленного ниже, создадим дерево.

```
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.right.left = Node(5)
root.right.right = Node(7)
root.left.left.left = Node(10)
root.left.left.right = Node(11)
root.right.right.left = Node(8)
leftMostShortest(root)
```

В результате получим самый короткий путь: 1 3 5.

Вывод. Мы изучили СД типа «дерева», научились их программно реализовывать и использовать, строить дерево в ширину, находить кратчайший путь до какого-либо узла.