

## Исследование поиска с ограничением глубины

**Цель работы:** приобретение навыков по работе с поиском с ограничением глубины с помощью языка программирования Python версии 3.x.

```
11 def dls_search(node, goal, limit, depth=0):
12     """Поиск с ограничением глубины в бинарном дереве"""
13     if node is None:
14         return False
15
16     if node.value == goal:
17         return True
18
19     if depth >= limit:
20         return False
21
22     # Рекурсивно ищем в левом и правом поддеревьях
23     if dls_search(node.left, goal, limit, depth + 1):
24         return True
25
26     if dls_search(node.right, goal, limit, depth + 1):
27         return True
28
29     return False
30
31
32 def main():
33     root = BinaryTreeNode(
34         1,
35         BinaryTreeNode(2, None, BinaryTreeNode(4)),
36         BinaryTreeNode(3, BinaryTreeNode(5), None)
37     )
38
39     goal = 4
40     limit = 2
41
```

Рисунок 1 - Результат работы программы «Система навигации робота-пылесоса»

```

11 def depth_limited_search(node, goal, limit, depth=0):
12     """Поиск с ограничением глубины. Возвращает узел цели, если найден в пределах глубины limit, иначе None."""
13     if node is None:
14         return None
15
16     if node.value == goal:
17         return node
18
19     if depth >= limit:
20         return None
21
22     # Рекурсивный поиск в левом поддереве
23     left_result = depth_limited_search(node.left, goal, limit, depth + 1)
24     if left_result:
25         return left_result
26
27     # Рекурсивный поиск в правом поддереве
28     right_result = depth_limited_search(node.right, goal, limit, depth + 1)
29     if right_result:
30         return right_result
31
32     return None
33
34
35 def main():
36     root = BinaryTreeNode(
37         1,
38         BinaryTreeNode(2, None, BinaryTreeNode(4)),
39         BinaryTreeNode(3, BinaryTreeNode(5), None)
40     )
41
42     goal = 4
43     limit = 2
44

```

PROBLEMS 45 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 45

Цель найдена: <4>  
(venv) PS C:\!lab4\lab4> []

Рисунок 2 - Результат работы программы «Система управления складом»

```

11 def find_max_at_depth(node, limit, current_depth=0):
12     """Находим максимальное значение на указанной глубине в бинарном дереве"""
13     if node is None:
14         # Возвращаем минус бесконечность для пустых узлов чтобы не влиять на сравнение
15         return float('-inf')
16
17     if current_depth == limit:
18         return node.value
19
20     left_max = find_max_at_depth(node.left, limit, current_depth + 1)
21     right_max = find_max_at_depth(node.right, limit, current_depth + 1)
22
23     # Возвращаем максимальное значение из поддеревьев
24     return max(left_max, right_max)
25
26 def main():
27     root = BinaryTreeNode(
28         3,
29         BinaryTreeNode(1, BinaryTreeNode(0), None),
30         BinaryTreeNode(5, BinaryTreeNode(4), BinaryTreeNode(6))
31     )
32
33     limit = 2
34
35     max_value = find_max_at_depth(root, limit)
36

```

PROBLEMS 68 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 68

(venv) PS C:\!lab4\lab4> python investment\_system.py  
Максимальное значение на указанной глубине: 6

Рисунок 3 - Результат работы программы «Система автоматического управления инвестициями»

```
(venv) PS C:\!lab4\lab4> python romania_graph.py
● Анализ графа Румынии поиск с ограничением глубины (DLS)
Начальный город: Arad
Целевой город: Bucharest

Эксперимент 1: Ограничение глубины = 3
Путь найден: Arad -> Sibiu -> Fagaras -> Bucharest
Стоимость: 450 км
Глубина: 3

Эксперимент 2: Ограничение глубины = 4
Путь найден: Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest
Стоимость: 418 км
Глубина: 4

Эксперимент 3: Ограничение глубины = 5
Путь найден: Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest
Стоимость: 418 км
Глубина: 4
Сравнение с ручным решением:
Ручное решение (A* алгоритм):
    Путь: Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest
    Стоимость: 418 км
    Длина пути: 5 городов

Решение DLS:
    Путь: Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest
    Стоимость: 418 км
    Длина пути: 5 городов

Сравнение:
    Разница в стоимости: 0 км
    Алгоритм DLS нашел оптимальное решение!
```

Рисунок 4 - Результат сравнения поиска с ограничением глубины и ручным решением

### Ответы на вопросы:

1. Поиск с ограничением глубины - это алгоритм поиска, который ограничивает максимальную глубину исследования дерева поиска. Он предотвращает бесконечные ветви устанавливая предельную глубину для рекурсивного спуска.
2. Основная цель ограничения глубины предотвратить уход алгоритма в бесконечный поиск по ветвям, которые не имеют конца.

3. Поиск в глубину исследует путь до конца пока не достигнет тупика.

Поиск с ограничением глубины останавливается на заданной глубине даже если путь может продолжаться.

4. Проверка глубины узла определяет достигнут ли установленный лимит глубины чтобы прекратить дальнейшее углубление по этому пути.

5. Функция возвращает обрезание чтобы указать что поиск был прерван из-за достижения лимита глубины, а не из-за отсутствия решений.

6. Поиск с ограничением глубины может не найти решение если оно находится на глубине превышающей установленный лимит.

7. Поиск в ширину использует очередь FIFO, а поиск в глубину использует стек LIFO.

8. Поиск с ограничением глубины не является оптимальным, потому что он может найти не самый короткий путь, а первый попавшийся в пределах глубины.

9. Итеративное углубление многократно запускает поиск с ограничением глубины постепенно увеличивая лимит что гарантирует нахождение оптимального решения.

10.Итеративное углубление эффективнее поиска в ширину, когда решение находится на большой глубине, а память ограничена.

11.Основная цель алгоритма поиска с ограничением глубины найти решение за разумное время избегая бесконечных циклов и чрезмерного использования памяти.

12.Функция depth\_limited\_search принимает problem объект задачи и limit максимальную глубину поиска.

13.Параметр limit по умолчанию равен 10.

14.Frontier это структура данных, стек содержащая узлы для исследования. Алгоритм извлекает из него узлы для проверки.

15.LIFOQueue это стек. Он используется для реализации поиска в глубину, где последний добавленный узел исследуется первым.

16.Переменная result инициализируется значением failure означающим что решение ещё не найдено.

17.Цикл while завершается, когда frontier пуст, то есть нет больше узлов для исследования.

18.Frontier.pop извлекает последний добавленный узел что соответствует стратегии поиска в глубину.

19.Если найден узел удовлетворяющий цели, функция немедленно возвращает этот узел как решение.

20.Условие проверяет достиг ли узел лимита глубины. Если да, то возвращается cutoff.

21.Если узел достиг ограничения глубины он не расширяется, а возвращается cutoff.

22.Проверка на циклы предотвращает зацикливание не позволяя расширять узлы которые уже присутствуют в пути от корня.

23.Дочерние узлы добавляются в frontier для дальнейшего исследования.

24.Если целевой узел не найден функция возвращает result, который может быть cutoff или failure.

25.Failure означает что решения нет в графе. Cutoff означает что решение возможно существует, но не было найдено из-за ограничения глубины.