

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
дисциплины «Алгоритмизация»
Вариант 29

Выполнил:
Саенко Андрей Максимович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., канд. технических
наук, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____
Ставрополь, 2023 г.

Тема: Жадные алгоритмы

Цель: изучить некоторые жадные алгоритмы

Порядок выполнения работы:

1. Составлена программа, которая принимает на вход массив чисел, а выводит массив с отрезками единичной длины, содержащий минимальное их количество, достаточное для покрытия всех определённых входным массивом точек.

Алгоритм создаёт цикл, на каждой итерации которого определяет границы нового отрезка, добавляет его в итоговый массив и удаляет из входного массива все элементы, которые лежат в границах этого отрезка. Если длина входного массива становится нулевой, цикл останавливается и выводится результат.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def points_cover(arr):
    result = []
    new_arr = arr.copy()

    while len(new_arr) != 0:
        x_left = min(new_arr)
        x_right = x_left + 1
        result.append([x_left, x_right])

        for i in arr:
            if i in new_arr and i >= x_left and i <= x_right:
                new_arr.remove(i)

    return result

if __name__ == "__main__":
    arr = [0.1, 1, 2, 3, 4, 5, 6, 7, 9, 11.1, 90]
    result = points_cover(arr)
    print(f"Массив на входе: {arr}")
    print("Результат:")
    print(*result, sep="\n")
```

Результат работы программы:

```
Массив на входе: [0.1, 1, 2, 3, 4, 5, 6, 7, 9, 11.1, 90]
Результат:
[0.1, 1.1]
[2, 3]
[4, 5]
[6, 7]
[9, 10]
[11.1, 12.1]
[90, 91]
PS C:\Users\HAIER>
```

Рисунок 1 – Результат работы программы

2. Составлена улучшенная версия первой программы. Новый алгоритм работает быстрее.

Алгоритм сортирует входной массив, создаёт цикл, на каждой итерации которого делает левой границей отрезка i -тое число, а правой – i -тое число+1, после чего добавляет отрезок в результирующий массив. После этого алгоритм увеличивает значение счётчика, пока i -тый элемент массива меньше значения правой границы отрезка и пока i меньше длины массива. После того, как начинает выполняться условие $i \geq \text{len}(\text{arr})$, функция возвращает результат.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def points_cover_upgrade(arr):
    result = []
    arr.sort()
    i = 0

    while i < len(arr):
        x_left = arr[i]
        x_right = x_left + 1
        result.append([x_left, x_right])
        i += 1

        while i < len(arr) and arr[i] <= x_right:
            i += 1

    return result
```

```

if __name__ == "__main__":
    arr = [0.1, 1, 2, 3, 4, 5, 6, 7, 9, 11.1, 90]
    result = points_cover_upgrade(arr)
    print(f"Массив на входе: {arr}")
    print("Результат:")
    print(*result, sep="\n")

```

Результат работы программы:

```

Массив на входе: [0.1, 1, 2, 3, 4, 5, 6, 7, 9, 11.1, 90]
Результат:
[0.1, 1.1]
[2, 3]
[4, 5]
[6, 7]
[9, 10]
[11.1, 12.1]
[90, 91]

```

Рисунок 2 – Результат работы программы

3. Написана программа для нахождения непересекающихся отрезков, количество которых должно быть максимально возможным.

Алгоритм заключается в том, что в наборе отрезков из входного массива ищется отрезок с наименьшим значением координаты правого конца, после чего он добавляется к результату, а из массива исключаются те отрезки, координата левого конца которых меньше координаты правого конца добавленного к результату отрезка.

Код программы:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def act_sel(arr):
    result = []

    while len(arr) > 0:
        min_right_item = arr[0]
        min_right = arr[0][1]

        for i in arr:
            if min_right > i[1]:
                min_right = i[1]
                min_right_item = i.copy()

        arr.remove(min_right_item)
        result.append(min_right_item)

```

```

new_arr = []
for i in arr:
    if i[0] > min_right:
        new_arr.append(i)

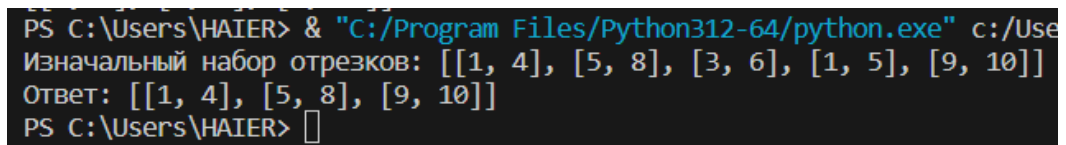
arr = new_arr.copy()

return result

if __name__ == "__main__":
    arr = [[1, 4], [5, 8], [3, 6], [1, 5], [9, 10]]
    print(f"Изначальный набор отрезков: {arr}")
    print(f"Ответ: {act_sel(arr)}")

```

Результат работы программы:



```

PS C:\Users\HAIER> & "C:/Program Files/Python312-64/python.exe" c:/Use
Изначальный набор отрезков: [[1, 4], [5, 8], [3, 6], [1, 5], [9, 10]]
Ответ: [[1, 4], [5, 8], [9, 10]]
PS C:\Users\HAIER>

```

Рисунок 3 – Результат работы программы

4. Написана программа по улучшенному алгоритму для нахождения непересекающихся отрезков, количество которых должно быть максимально возможным.

В новом алгоритме набор отрезков сортируется по правой координате, после чего отрезок добавляется в ответ, если не пересекает ранее добавленный.

Код программы:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def act_sel_upgrade(arr):
    arr.sort(key=lambda item: item[1])
    result = []
    result.append(arr[0])

    for i in arr:
        if i[0] > result[-1][1]:
            result.append(i)

    return result

```

```

if __name__ == "__main__":
    arr = [[1, 4], [5, 8], [3, 6], [1, 5], [9, 10]]
    print(f"Изначальный набор отрезков: {arr}")
    print(f"Ответ: {act_sel_upgrade(arr)}")

```

5. Написана программа для нахождения максимального независимого множества вершин дерева.

Код программы:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def max_independent_set(tree):
    if tree == {}:
        return []

    leaves = []
    branches = set()

    def traverse(t, path):
        for node, child in t.items():
            current_path = path + [node]
            if child == {}:
                if len(current_path) != 1:
                    branches.add(tuple(current_path[:-1]))
            else:
                branches.add((current_path[-1],))

            leaves.append(node)

            traverse(child, current_path)

    traverse(tree, [])
    list_br = list(branches)
    temp_list_branch = []
    sorted_branches = sorted(list_br, key=len)

    for branch in sorted_branches:
        temp_branch = []

        for i in range(len(branch)):
            if not branch[i] in temp_list_branch:
                temp_branch.append(branch[i])

        parent = tree
        if len(temp_branch) != 1:
            for node in temp_branch[:-1]:
                temp = parent

```

```

        parent = parent[node]
    else:
        temp = tree

    for key, value in parent[temp_branch[-1]].copy().items():
        if value == {}:
            del parent[temp_branch[-1]][key]
        elif len(temp_branch) != 1:
            temp[node][key] = value
        else:
            temp[key] = value

    del parent[temp_branch[-1]]
    temp_list_branch.append(temp_branch[-1])

list_leave = (max_independent_set(tree))
leaves.extend(list_leave)


return leaves

if __name__ == '__main__':
    tree = {1: {2: {4: {}, 5: {}, 6: {}}, 3: {7: {}, 8: {}, 9: {}}}}
    print(max_independent_set(tree))

    tree = {1: {2: {3: {4: {5: {6: {7: {8: {9: {10: {}}}}}}}}}}}}
    print(max_independent_set(tree))

```

Результат работы программы:



The screenshot shows a Python IDE with a dark theme. The editor displays two code snippets. The first snippet defines a tree structure and prints the result of max_independent_set(tree), which is [4, 5, 6, 7, 8, 9, 1]. The second snippet defines a more complex tree structure and prints the result of max_independent_set(tree), which is [10, 8, 6, 4, 2]. Below the editor, the terminal window shows the command prompt and the execution of the Python script, displaying the same two output lists.

```

62 if __name__ == '__main__':
63     tree = {1: {2: {4: {}, 5: {}, 6: {}}, 3: {7: {}, 8: {}, 9: {}}}}
64     print(max_independent_set(tree))
65
66     tree = {1: {2: {3: {4: {5: {6: {7: {8: {9: {10: {}}}}}}}}}}}}
67     print(max_independent_set(tree))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\HAIER> & "C:/Program Files/Python312-64/python.exe" c:/Users/HAIE
[4, 5, 6, 7, 8, 9, 1]
[10, 8, 6, 4, 2]

```

Рисунок 4 – Результат работы программы

6. Написана программа, использующая жадный алгоритм для решения задачи о рюкзаке.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def knapsack(items, weight):
    items = sorted(items, key=lambda x: x[0]/x[1], reverse=True)
    result = []

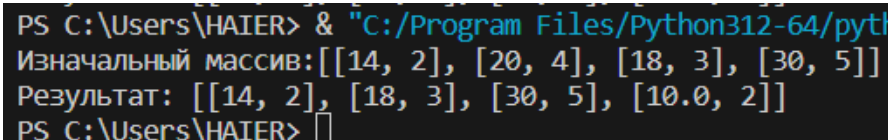
    for item in items:
        if item[1] <= weight:
            weight -= item[1]
            result.append(item)

        else:
            coefficient = item[0]/item[1]
            result.append([coefficient * weight, weight])
            weight = 0
            break

    return result

if __name__ == "__main__":
    items = [[14, 2], [20, 4], [18, 3], [30, 5]]
    weight = 12
    print(f'Изначальный массив: {items}')
    print(f'Результат: {knapsack(items, weight)}')
```

Результат работы программы:



```
PS C:\Users\HAIER> & "C:/Program Files/Python312-64/python.exe" knapsack.py
Изначальный массив: [[14, 2], [20, 4], [18, 3], [30, 5]]
Результат: [[14, 2], [18, 3], [30, 5], [10.0, 2]]
PS C:\Users\HAIER>
```

Рисунок 5 – Результат работы программы

Вывод

В ходе выполнения работы была изучена концепция жадных алгоритмов. При работе данные алгоритмы делают на каждом шаге лучший выбор, что чаще всего приводит к оптимальному решению.