ФГБОУ ВПО
ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ
ИМПЕРАТОРА АЛЕКСАНДРА I

Кафедра "Информационные и вычислительные системы"

ОТЧЕТ
по лабораторной работе №2

Динамическое программирование

Выполнил студент ____________ Зайцев Л.А.
(подпись)

Группа ИВБ-811

Отчет принял ____________ Баушев А.Н.
(подпись)

Санкт-Петербург, 2020 г.

1 Алгоритма поиска наибольшей возрастающей подпоследовательности

1.1 Постановка задачи

Дан массив из n чисел: a[1 … n]. Требуется найти в этой последовательности строго возрастающую подпоследовательность наибольшей длины.

1.2 Описание алгоритма

Возьмем массив d[1 … n], где d[i] - это длина наибольшей возрастающей подпоследовательности, оканчивающейся именно в элементе с индексом i.
Пусть текущий индекс — i, т.е. мы хотим посчитать значение d[i], а все предыдущие значения d[1] … d[i-1] уже подсчитаны. Тогда заметим, что у нас есть два варианта:

- d[i] = 1, т.е. искомая подпоследовательность состоит только из числа a[i].
- d[i] > 1. Тогда перед числом a[i] в искомой подпоследовательности стоит какое-то другое число. Давайте переберём это число: это может быть любой элемент a[j] (j = 1 … i-1), но такой, что a[j] < a[i]. Пусть мы рассматриваем какой-то текущий индекс j. Поскольку динамика d[j] для него уже подсчитана, получается, что это число a[j] вместе с числом a[i] даёт ответ d[j] + 1. Таким образом, d[i] можно считать по такой формуле:

$$d[i] = \max_{\substack{j=1...i-1 \\ a[j]<a[i]}} (d[j] + 1) \tag{1}$$

Объединяя эти два варианта в один, получаем окончательный алгоритм для вычисления d[i]:

$$d[i] = max(1, \max_{\substack{j=1...i-1 \\ a[j]<a[i]}} (d[j] + 1)) \tag{2}$$

1.3 Код программы

Листинг 1: Наибольшая возрастающая подпоследовательность

```
function [Res, Path] = LIS(array)
Res = 0;
Path = [];
```

```
n = length(array);
d = zeros(1, n);
prev = zeros(1, n);

for i = 1 : n
        d(i) = 1;
        prev(i) = -1;
        for j = 1 : n
                if array(j) < array(i) && 1 + d(j) > d(i)
                        d(i) = d(j) + 1;
                        prev(i) = j;
                end
        end
end

Res = d(1);
pos = 1;

for i = 1 : n
        if (d(i) > Res)
                Res = d(i);
                pos = i;
        end
end

i = 1;
while pos ~= -1
        Path(i) = pos;
        pos = prev(pos);
        i = i + 1;
end

n = length(Path);
for i = 1 : n / 2
        tmp = Path(i);
        Path(i) = Path(n - i + 1);
        Path(n - i + 1) = tmp;
end
end % End of 'LIS' function
```

1.4 Сложность алгоритма

O(n^2)

1.5 Результат работы

Листинг 2: Тест. Наибольшая возрастающая подпоследовательность

```
a(1) = 1;
a(2) = 5;
a(3) = 3;
a(4) = 7;
a(5) = 1;
a(6) = 4;
a(7) = 10;
a(8) = 15;

[ans, path] = LIS(a)
n = 100 : 100 : 10000
t1 = zeros(1, length(n));

for i = 1 : length(n)
        a = zeros(1, n(i));
        for k = 1 : n(i)
                a(k) = length(a) * rand(1, 1);
        end
        tic
        LIS(a);
        t1(i) = toc;
end
figure;
hold on;
plot(n, t1);
```

2 Алгоритм поиска расстояния редактирования

2.1 Постановка задачи

Расстояние редактирования - это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

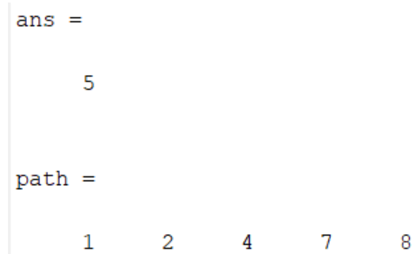

Рис. 1: Результат работы алгоритма по поиску наибольшей возрастающей подпоследовательности

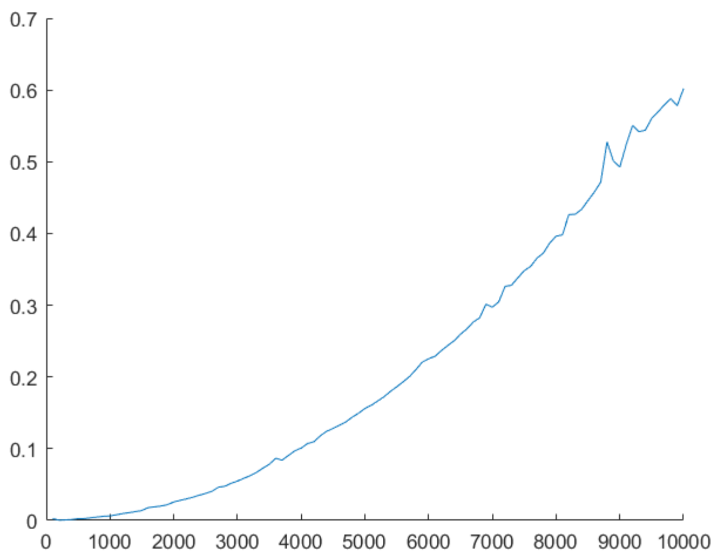

Рис. 2: Время работы алгоритма по поиску наибольшей возрастающей подпоследовательности

2.2 Описание алгоритма

Возьмем массив d[1 ... n][1 ... m], где m и n - длины строк (S_1 и S_2), а d[i][j] - расстояние между префиксами строк: первыми i символами строки S_1 и первыми j символами строки S_2. Оно вычисляется по формуле:

$$d[i][j] = \begin{cases} 0 & i = 1, j = 1; \\ i & j = 1, i > 1 \\ j & j > 1, i = 1 \\ d(i-1, j-1) & S_1[i] = S_2[j] \\ min(& \\ d(i, j-1) + 1, & \\ d(i-1, j) + 1, & \\ d(i-1, j-1) + 1) & j > 1, i > 1, S_1[i] \neq S_2[j] \end{cases} \tag{3}$$

2.3 Код программы

Листинг 3: Расстояние редактирования

```
function [Res] = LevDist(a, b)
n = length(a) + 1;
m = length(b) + 1;
Res = 0;
d = zeros(n, m);

for i = 2 : n
        d(i, 1) = i - 1;
end

for j = 2 : m
        d(1, j) = j - 1;
end

for i = 2 : n
        for j = 2 : m
                if a(i - 1) ~= b(j - 1)
                        d(i, j) = min(d(i, j - 1) + 1, min(d(i - 1
                else
                        d(i, j) = d(i - 1, j - 1);
                end
        end
end
```

```
d
Res = d(n, m);
end % End of 'LevDist' function
```

2.4 Сложность алгоритма

O(mn)

2.5 Результат работы

Листинг 4: Тест. Расстояние редактирования

```
LevDist('exponential', 'polynomial')
```

```
d =

     0     1     2     3     4     5     6     7     8     9    10
     1     1     2     3     4     5     6     7     8     9    10
     2     2     2     3     4     5     6     7     8     9    10
     3     2     3     3     4     5     6     7     8     9    10
     4     3     2     3     4     5     5     6     7     8     9
     5     4     3     3     4     4     5     6     7     8     9
     6     5     4     4     4     5     5     6     7     8     9
     7     6     5     5     5     4     5     6     7     8     9
     8     7     6     6     6     5     5     6     7     8     9
     9     8     7     7     7     6     6     6     6     7     8
    10     9     8     8     8     7     7     7     7     6     7
    11    10     9     8     9     8     8     8     8     7     6


ans =

     6
```

Рис. 3: Результат работы алгоритма по поиску расстояния редактирования

3 Задача о рюкзаке 0-1

3.1 Постановка задачи

Дано N предметов, W — вместимость рюкзака, w = $w_1, w_2, \ldots, w_N$ — соответствующий ему набор положительных целых весов, p = $p_1, p_2, \ldots, p_N$ — соответствующий ему набор положительных целых стоимостей. Нужно найти набор бинарных величин B=$b_1, b_2, \ldots, b_N$, где $b_i = 1$, если предмет n_i включен в набор, $b_i = 0$, если предмет n_i не включен, и такой что:

1. $b_1 w_1 + \ldots + b_N w_N \leq W$

2. $b_1 p_1 + \ldots + b_N p_N$ максимальна.

3.2 Описание алгоритма

Пусть A(k,s) есть максимальная стоимость предметов, которые можно уложить в рюкзак вместимости s, если можно использовать только первые k предметов, то есть $n_1, n_2, \ldots, n_k$, назовем этот набор допустимых предметов для A(k,s).

A(k,1) = 0

A(1,s) = 0

Найдем A(k,s). Возможны 2 варианта:

1. Если предмет k не попал в рюкзак. Тогда A(k,s) равно максимальной стоимости рюкзака с такой же вместимостью и набором допустимых предметов $n_1, \ldots, n_{k-1}$, то есть A(k, s) = A(k - 1, s)

2. Если k попал в рюкзак. Тогда A(k,s) равно максимальной стоимости рюкзака, где вес s уменьшаем на вес k-ого предмета и набор допустимых предметов $n_1, n_2, \ldots, n_{k-1}$ плюс стоимость k, то есть $A(k-1, s-w_k) + p_k$

3.3 Код программы

Листинг 5: Тест. Расстояние редактирования

```
function [Res, Collection] = Knapsack01(w, p, Weight)
Collection = [];
Res = 0;
n = length(w);
A = zeros(n + 1, Weight + 1);

for i = 2 : n + 1
        for j = 1 : Weight + 1
                if j > w(i - 1)
```

```
                        A(i, j) = max(A(i - 1, j), A(i - 1, j - w(
                else
                        A(i, j) = A(i - 1, j);
                end
        end
end

A
Res = A(n + 1, Weight + 1);

i = n + 1;
j = Weight + 1;
index = 1;

while A(i, j) ~= 0
        if A(i - 1, j) == A(i, j)
                i = i - 1;
        else
                i = i - 1;
                j = j - w(i);
                Collection(index) = i;
                index = index + 1;
        end
end
end % End of 'Knapsack01' function
```

3.4 Сложность алгоритма

O(nW)

3.5 Результат работы

Листинг 6: Тест. Задача о рюкзаке без повторений

```
Knapsack01([3, 4, 5, 8, 9], [1, 6, 4, 7, 6], 13)
```

4 Задача о неограниченном рюкзаке

4.1 Постановка задачи

Теперь любой предмет можно брать неограниченное количество раз.

```
A =

     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     1     1     1     1     1     1     1     1
     0     0     0     1     6     6     6     7     7     7     7     7     7     7
     0     0     0     1     6     6     6     7     7    10    10    10    11    11
     0     0     0     1     6     6     6     7     7    10    10    10    13    13
     0     0     0     1     6     6     6     7     7    10    10    10    13    13


ans =

    13


coll =

     4     2
```

Рис. 4: Результат работы алгоритма задача о рюкзаке без повторений

4.2 Описание алгоритма

Пусть d(i,j) - максимальная стоимость любого количества вещей типов от 1 до i, суммарным весом до j включительно. Заполним d(0,j) нулями. Тогда меняя i от 2 до N, рассчитаем на каждом шаге d(i, j), для c от 1 до W, по рекуррентной формуле:

$$d(i,j) = \begin{cases} d(i-1,j) & forj = 0,\ldots,w_i - 1 \\ max(d(i-1,j), d(i,j-w_i)+p_i) & forj = w_i,\ldots,W; \end{cases} \tag{4}$$

4.3 Код программы

Листинг 7: Тест. Задача о неограниченном рюкзаке

```
function [Res, Collection] = Knapsack(w, p, Weight)
Res = 0;
Collection = [];

n = length(w);
d = zeros(n + 1, Weight + 1);

for i = 2 : n + 1
        for j = 1 : Weight + 1
                if j > w(i - 1)
                        d(i, j) = max(d(i - 1, j), d(i, j - w(i -
                else
```

```
                        d(i, j) = d(i - 1, j);
                end
        end
end
d
Res = d(i, j);

i = n + 1;
j = Weight + 1;
index = 1;

while d(i, j) ~= 0
        if d(i - 1, j) == d(i, j)
                i = i - 1;
        else
                j = j - w(i - 1);
                Collection(index) = i - 1;
                index = index + 1;
        end
end
end % End of 'Knapsack' function
```

4.4 Сложность алгоритма

O(nW)

4.5 Результат работы

Листинг 8: Тест. Задача о неограниченном рюкзаке

```
[ans, coll] = Knapsack([3, 4, 5, 8, 9], [1, 6, 4, 7, 6], 13)
```

```
d =

     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     2     2     2     3     3     3     4     4
     0     0     0     1     6     6     6     7    12    12    12    13    18    18
     0     0     0     1     6     6     6     7    12    12    12    13    18    18
     0     0     0     1     6     6     6     7    12    12    12    13    18    18
     0     0     0     1     6     6     6     7    12    12    12    13    18    18


ans =

    18


coll =

     2     2     2
```

Рис. 5: Результат работы алгоритма задача о неограниченном рюкзаке

5 Вывод

В ходе выполнения лабораторной работы мы познакомились с методом динамического программирования путем решения классических задач. Динамическое программирование - способ решения сложных задач путём разбиения их на более простые подзадачи. Мы научились составлять функциональные уравнения динамического программирования.

6 Библиографический список

1. https://ru.wikipedia.org/
2. https://neerc.ifmo.ru/