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 № 11
дисциплины «Алгоритмизация»

Выполнила:
Кубанова Ксения Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: динамическое программирование алгоритмов

Цель: изучить и реализовать алгоритмы, используя динамическое программирование

Порядок выполнения работы:

Задание 1.

Нахождение числа Фибоначчи при помощи динамического программирования

Вход: число

Выход: число Фибоначчи

```
Инициализация
F[0...n] = [−1, −1, ..., −1]

Функция FibTD(n)
если F[n] = −1:
  если n ≤ 1:
    F[n] ← n
  иначе:
    F[n] ← FibTD(n − 1) + FibTD(n − 2)
вернуть F[n]
```

Рисунок 1 – Алгоритм нахождения числа Фибоначчи (назад)

Реализация этого алгоритма представлена в файле fib.cpp на строчках 5-11.

```
Функция FibBU(n)
создать массив F[0...n]
F[0] ← 0, F[1] ← 1
для i от 2 до n:
  F[i] ← F[i − 1] + F[i − 2]
вернуть F[n]
```

Рисунок 2 – Алгоритм нахождения числа Фибоначчи (вперёд)

Реализация этого алгоритма представлена в файле fib.cpp на строчках 14-22.

Ниже представлен алгоритм динамического программирования с уменьшение количества потребления памяти (рис.3):

```
Функция FibBUImproved(n)
если n ≤ 1:
  вернуть n

prev ← 0
curr ← 1

повторить (n − 1) раз:
  next ← prev + curr
  prev ← curr
  curr ← next
вернуть curr
```

Рисунок 3 – Алгоритм нахождения числа Фибоначчи

Реализация этого алгоритма представлена в файле fib.cpp на строчках 5-25-39.

```
Число Фибоначчи(10):
fibonacci(10, 0) = 55
fibonacci(10, 1) = 55
fibonacci(10, 2) = 55
```

Рисунок 4 – Результат работы файла fib.cpp

Задание 2.

Нахождение длинны НВП и самой НВП

Вход: последовательность A[1…n] = [a1, a2, …, an].

Выход: НВП: ai1, ai2 … ain, что i1 < i2 < … ik.

Ниже представлен алгоритм поиска длинны НВП (рис.5):

```
Функция LISBottomUp(A[1...n])
создать массив D[1...n]
для i от 1 до n:
  D[i] ← 1
  для j от 1 до i − 1:
    если A[j] < A[i] и D[j] + 1 > D[i]:
      D[i] ← D[j] + 1
ans ← 0
для i от 1 до n:
  ans ← max(ans, D[i])
вернуть ans
```

Рисунок 5 – Алгоритм нахождения длинны НВП

Реализация этого алгоритма представлена в файле list.cpp на строчках 6-19.

Ниже представлен алгоритм восстановления с помощью списка prev:

```
Восстановление ответа
создать массив L[1...ans]   {индексы НВП}
k ← 1
для i от 2 до n:
  если D[i] > D[k]:
    k ← i
j ← ans
пока k > 0:
  L[j] ← k
  j ← j − 1
  k ← prev[k]
```

Рисунок 6 – Алгоритм нахождения НВП

Реализация этого алгоритма представлена в файле list.cpp на строчках 21-32.

Ниже представлен алгоритм восстановления без помощи списка prev:

```
Функция LISBOTTOMUP2(A[1...n])
создать массивы D[1...n] и prev[1...n]
для i от 1 до n:
  D[i] ← 1, prev[i] ← −1
  для j от 1 до i − 1:
    если A[j] < A[i] и D[j] + 1 > D[i]:
      D[i] ← D[j] + 1, prev[i] ← j
ans ← 0
для i от 1 до n:
  ans = max(ans, D[i])
вернуть ans
```

Рисунок 7 – Алгоритм нахождения НВП

Реализация этого алгоритма представлена в файле list.cpp на строчках 34-51.

```
5
5
Using prev: [1, 3, 5, 9, 11]
Without prev: [2, 3, 5, 10, 11]
```

Рисунок 8 – Результат работы файла list.cpp

Задание 3.

Поиск максимальной стоимости предметов в рюкзаке

Вход: веса и стоимости n предметов, вместимость рюкзака W

Выход: максимальная стоимость предметов веса не более W

Ниже представлен алгоритм поиска максимальной стоимости предметов в рюкзаке при том, что предметы могут повторяться:

```
Функция
KNAPSACKWITHREPSBU(W, w_1, ..., w_n, c_1, ..., c_n)
создать массив D[0...W] = [0, 0, ..., 0]
для w от 1 до W:
  для i от 1 до n:
    если w_i ≤ w:
      D[w] ← max(D[w], D[w − w_i] + c_i)
вернуть D[W]
```

Рисунок 9 – Алгоритм поиска максимальной стоимости предметов в рюкзаке

Реализация этого алгоритма представлена в файле knapsack.cpp на строчках 6-16.

Ниже представлен алгоритм поиска максимальной стоимости предметов в рюкзаке при том, что предметы не могут повторяться:

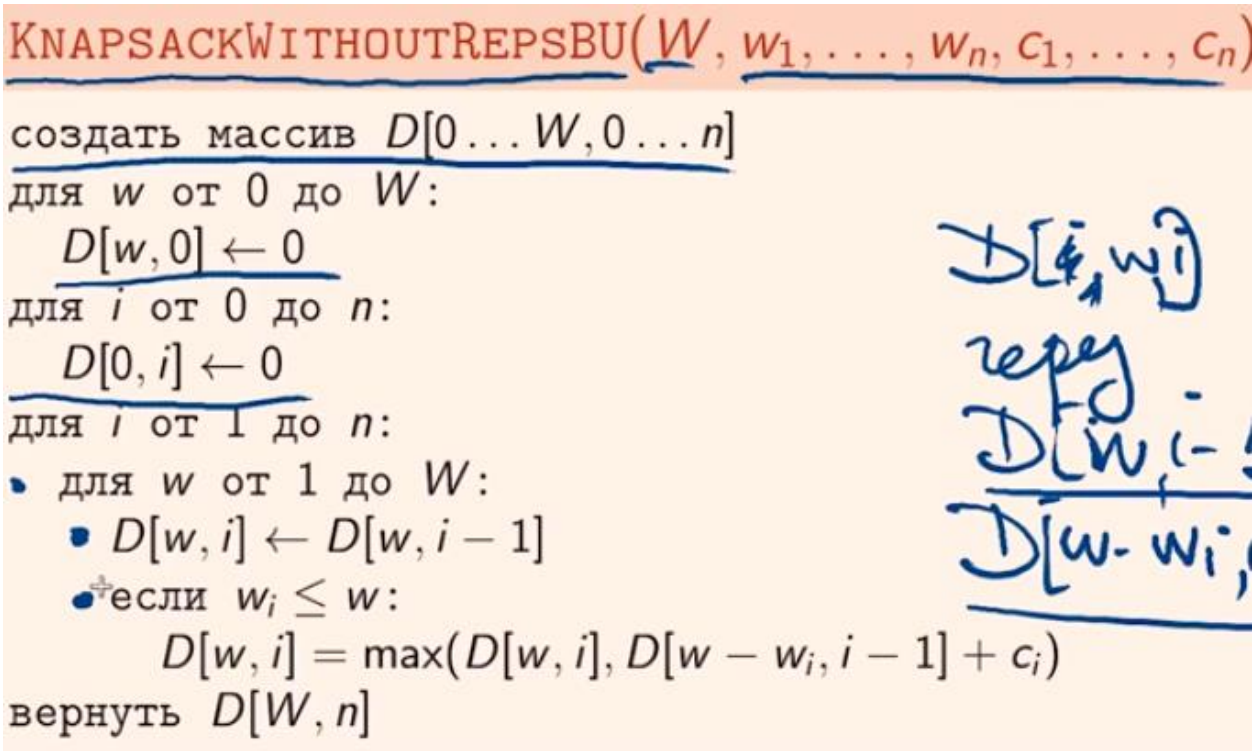

Рисунок 10 – Алгоритм поиска максимальной стоимости предметов в рюкзаке

Реализация этого алгоритма представлена в файле knapsack.cpp на строчках 18-48.

```
with_rep_bu = 48
without_rep_bu = 46, solution = [1, 0, 1, 0]
```

Рисунок 11 – Результат работы файла knapsack.cpp

Вывод: в ходе выполнения работы были исследованы алгоритмы динамического программирования: нахождение числа Фибоначчи, нахождение длинны НВП и самой НВП и алгоритм расчёта максимальной стоимости предметов в рюкзаке.