

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 ПРАКТИЧЕСКОЙ РАБОТЕ №11
дисциплины «Алгоритмизация»
Вариант 14

Выполнил:
Касимов Асхаб Арсенович
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.

Ход работы

Задание 1. Нахождение числа Фибоначчи при помощи динамического программирования

Динамическое программирование назад

Инициализация

$$F[0 \dots n] = [-1, -1, \dots, -1]$$

Функция $FIBTD(n)$

```
если  $F[n] = -1$ :  
    если  $n \leq 1$ :  
         $F[n] \leftarrow n$   
    иначе:  
         $F[n] \leftarrow FIBTD(n-1) + FIBTD(n-2)$   
вернуть  $F[n]$ 
```

Рисунок 1 – Алгоритм нахождения числа Фибоначчи

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

Динамическое программирование вперед:

Функция $FIBBU(n)$

```
создать массив  $F[0 \dots n]$   
 $F[0] \leftarrow 0, F[1] \leftarrow 1$   
для  $i$  от 2 до  $n$ :  
     $F[i] \leftarrow F[i-1] + F[i-2]$   
вернуть  $F[n]$ 
```

Рисунок 2 – Алгоритм нахождения числа Фибоначчи

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

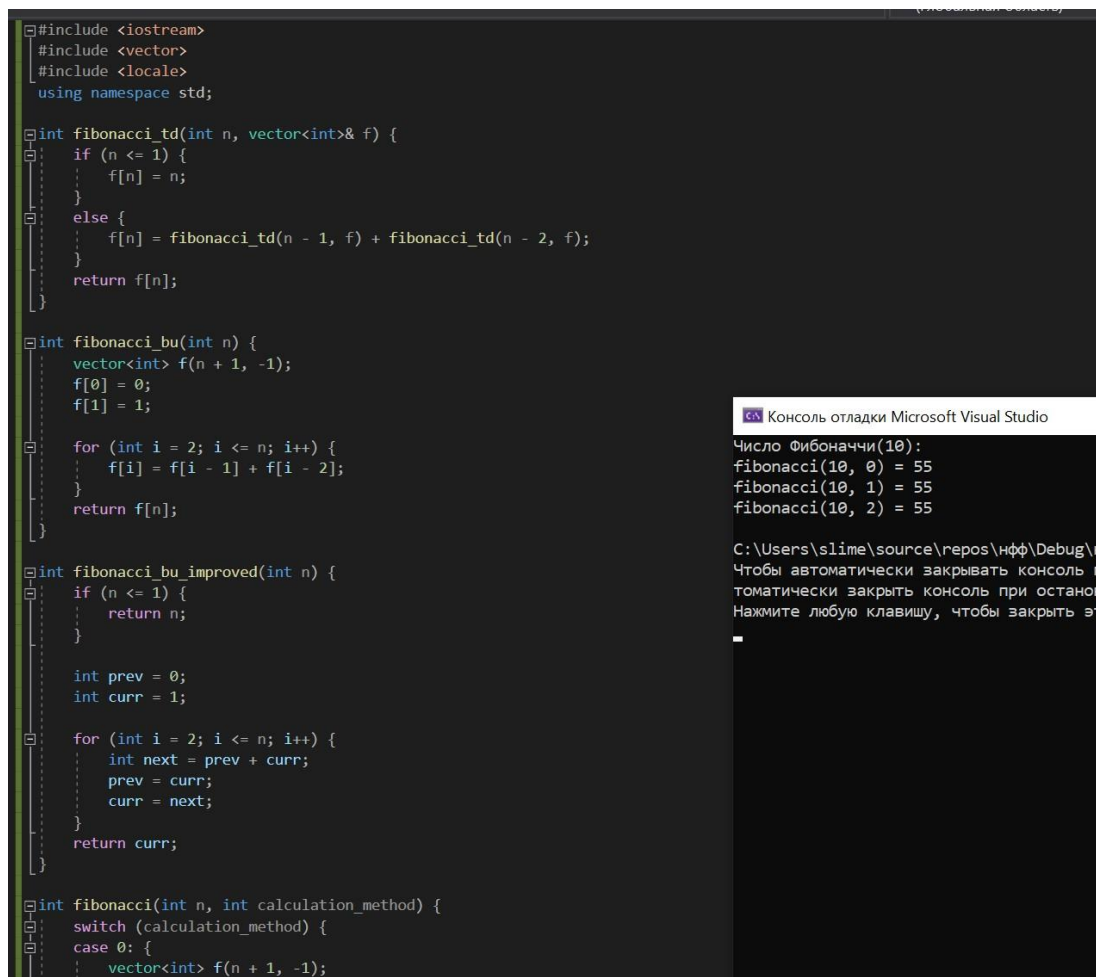
Алгоритм динамического программирования с уменьшением количества потребления памяти:

Функция FIBUIIMPROVED(n)

```
если  $n \leq 1$ :  
    вернуть  $n$   
 $prev \leftarrow 0$   
 $curr \leftarrow 1$   
  
повторить  $(n - 1)$  раз:  
     $next \leftarrow prev + curr$   
     $prev \leftarrow curr$   
     $curr \leftarrow next$   
вернуть  $curr$ 
```

Рисунок 3 – Алгоритм нахождения числа Фибоначчи

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



```
#include <iostream>
#include <vector>
#include <locale>
using namespace std;

int fibonacci_td(int n, vector<int>& f) {
    if (n <= 1) {
        f[n] = n;
    }
    else {
        f[n] = fibonacci_td(n - 1, f) + fibonacci_td(n - 2, f);
    }
    return f[n];
}

int fibonacci_bu(int n) {
    vector<int> f(n + 1, -1);
    f[0] = 0;
    f[1] = 1;

    for (int i = 2; i <= n; i++) {
        f[i] = f[i - 1] + f[i - 2];
    }
    return f[n];
}

int fibonacci_bu_improved(int n) {
    if (n <= 1) {
        return n;
    }

    int prev = 0;
    int curr = 1;

    for (int i = 2; i <= n; i++) {
        int next = prev + curr;
        prev = curr;
        curr = next;
    }
    return curr;
}

int fibonacci(int n, int calculation_method) {
    switch (calculation_method) {
        case 0: {
            vector<int> f(n + 1, -1);
            return fibonacci_td(n, f);
        }
        case 1: {
            return fibonacci_bu(n);
        }
        case 2: {
            return fibonacci_bu_improved(n);
        }
    }
}
```

Консоль отладки Microsoft Visual Studio

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

C:\Users\slime\source\repos\нф\Debug\n
Чтобы автоматически закрывать консоль п
томатически закрыть консоль при остано
Нажмите любую клавишу, чтобы закрыть эт

Рисунок 4 – Результат работы файла fib.cpp

Задание 2. Нахождение длины НВП и самой НВП

Ниже представлен алгоритм поиска длины НВП:

Функция LISBOTOMUP($A[1 \dots n]$)

```
создать массив  $D[1 \dots n]$ 
для  $i$  от 1 до  $n$ :
     $D[i] \leftarrow 1$ 
    для  $j$  от 1 до  $i - 1$ :
        если  $A[j] < A[i]$  и  $D[j] + 1 > D[i]$ :
             $D[i] \leftarrow D[j] + 1$ 
ans  $\leftarrow 0$ 
для  $i$  от 1 до  $n$ :
    ans  $\leftarrow \max(ans, D[i])$ 
вернуть ans
```

Рисунок 5 – Алгоритм нахождения длины НВП

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

Алгоритм восстановления с помощью списка prev:

Восстановление ответа

```
создать массив  $L[1 \dots ans]$     {индексы НВП}
 $k \leftarrow 1$ 
для  $i$  от 2 до  $n$ :
    если  $D[i] > D[k]$ :
         $k \leftarrow i$ 
 $j \leftarrow ans$ 
пока  $k > 0$ :
     $L[j] \leftarrow k$ 
     $j \leftarrow j - 1$ 
     $k \leftarrow prev[k]$ 
```

Рисунок 6 – Алгоритм нахождения НВП

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

Алгоритм восстановления без помощи списка prev:

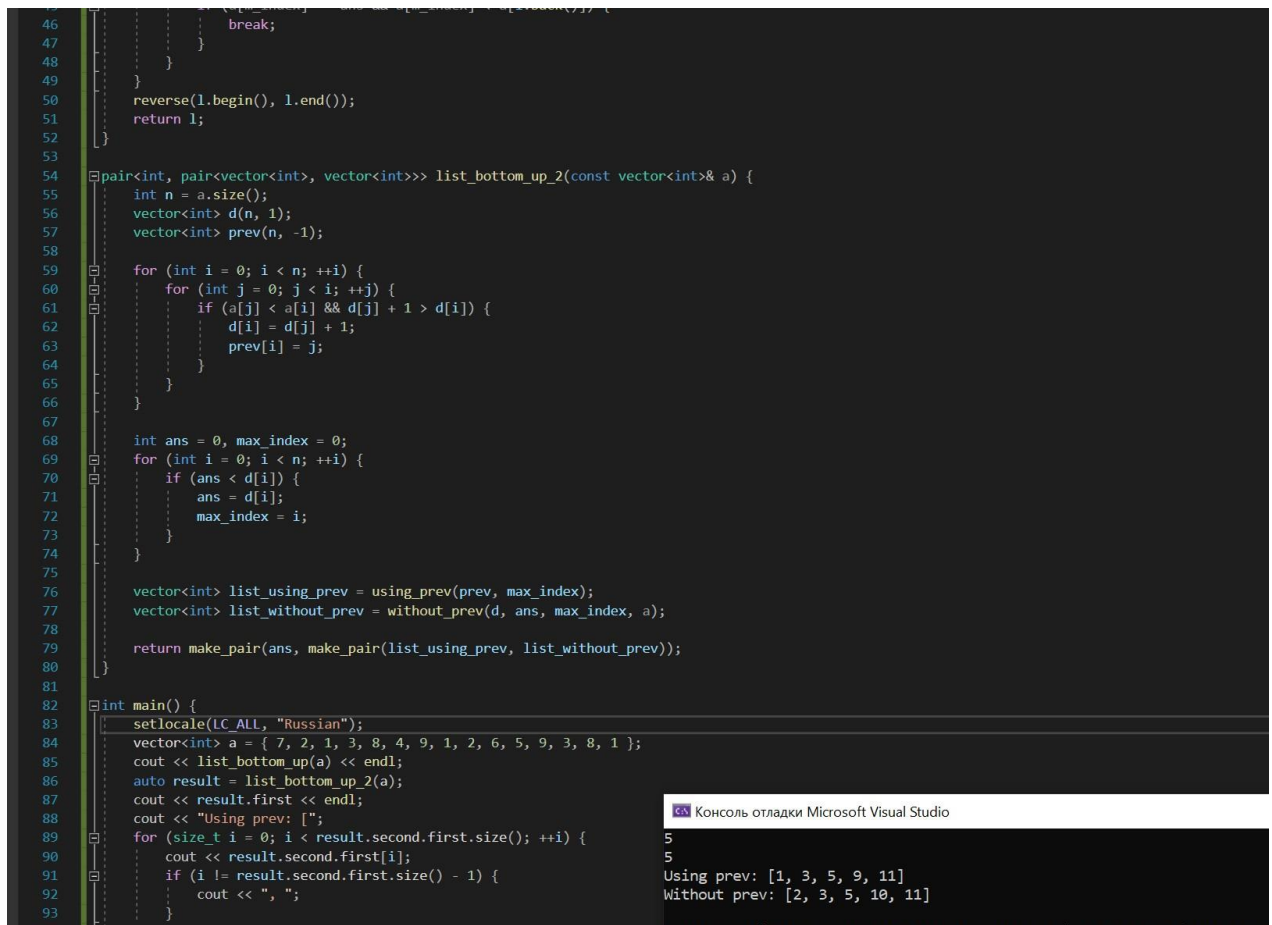
Функция LISBOTTOMUP2($A[1 \dots n]$)

создать массивы $D[1 \dots n]$ и $prev[1 \dots n]$
для i от 1 до n :
 $D[i] \leftarrow 1$, $prev[i] \leftarrow -1$
 для j от 1 до $i - 1$:
 если $A[j] < A[i]$ и $D[j] + 1 > D[i]$:
 $D[i] \leftarrow D[j] + 1$, $prev[i] \leftarrow j$

$ans \leftarrow 0$
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



```
46         break;
47     }
48 }
49 }
50 reverse(l.begin(), l.end());
51 return l;
52 }
53
54 pair<int, pair<vector<int>, vector<int>>> list_bottom_up_2(const vector<int>& a) {
55     int n = a.size();
56     vector<int> d(n, 1);
57     vector<int> prev(n, -1);
58
59     for (int i = 0; i < n; ++i) {
60         for (int j = 0; j < i; ++j) {
61             if (a[j] < a[i] && d[j] + 1 > d[i]) {
62                 d[i] = d[j] + 1;
63                 prev[i] = j;
64             }
65         }
66     }
67
68     int ans = 0, max_index = 0;
69     for (int i = 0; i < n; ++i) {
70         if (ans < d[i]) {
71             ans = d[i];
72             max_index = i;
73         }
74     }
75
76     vector<int> list_using_prev = using_prev(prev, max_index);
77     vector<int> list_without_prev = without_prev(d, ans, max_index, a);
78
79     return make_pair(ans, make_pair(list_using_prev, list_without_prev));
80 }
81
82 int main() {
83     setlocale(LC_ALL, "Russian");
84     vector<int> a = { 7, 2, 1, 3, 8, 4, 9, 1, 2, 6, 5, 9, 3, 8, 1 };
85     cout << list_bottom_up(a) << endl;
86     auto result = list_bottom_up_2(a);
87     cout << result.first << endl;
88     cout << "Using prev: [";
89     for (size_t i = 0; i < result.second.first.size(); ++i) {
90         cout << result.second.first[i];
91         if (i != result.second.first.size() - 1) {
92             cout << ", ";
93         }
94     }
95     cout << "]\n";
96     cout << "Without prev: [";
97     for (size_t i = 0; i < result.second.second.size(); ++i) {
98         cout << result.second.second[i];
99         if (i != result.second.second.size() - 1) {
100             cout << ", ";
101         }
102     }
103     cout << "]\n";
104 }
```

Консоль отладки Microsoft Visual Studio

```
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

Алгоритм поиска максимальной стоимости предметов в рюкзаке при том, что предметы могут повторяться:

Функция
KNAPSACKWITHREPSBU($W, w_1, \dots, w_n, c_1, \dots, c_n$)

создать массив $D[0 \dots W] = [0, 0, \dots, 0]$
для w от 1 до W :
 для i от 1 до n :
 если $w_i \leq w$:
 $D[w] \leftarrow \max(D[w], D[w - w_i] + c_i)$
вернуть $D[W]$

Рисунок 9 – Алгоритм поиска максимальной стоимости предметов в рюкзаке

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

Алгоритм поиска максимальной стоимости предметов в рюкзаке при том, что предметы не могут повторяться:

KNAPSACKWITHOUTREPSBU($W, w_1, \dots, w_n, c_1, \dots, c_n$)

создать массив $D[0 \dots W, 0 \dots n]$
для w от 0 до W :
 $D[w, 0] \leftarrow 0$
для i от 0 до n :
 $D[0, i] \leftarrow 0$
для i от 1 до n :
 • для w от 1 до W :
 • $D[w, i] \leftarrow D[w, i - 1]$
 • если $w_i \leq w$:
 $D[w, i] = \max(D[w, i], D[w - w_i, i - 1] + c_i)$
вернуть $D[W, n]$

Handwritten notes:
 $D[i, w_i]$
перейти
 $D[w, i - 1]$
 $D[w - w_i, i - 1]$

Рисунок 10 – Алгоритм поиска максимальной стоимости предметов в рюкзаке

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

```
(d[w][i] == d[w - weight[i - 1]][i - 1] + cell[i - 1]) {  
    solution.push_back(1);  
    w -= weight[i - 1];  
}  
else {  
    solution.push_back(0);  
}  
with_rep_bu = 48  
without_rep_bu = 46, solution = [1, 0, 1, 0]  
e(solution.begin(), solution.end(), " ");
```

Рисунок 11 – Результат работы файла knapsack.cpp

Вывод: В ходе выполнения лабораторной работы были изучены алгоритмы динамического программирования, такие как нахождение числа Фибоначчи, нахождение длины наибольшей возрастающей подпоследовательности (НВП) и самой НВП, а также алгоритм расчета максимальной стоимости предметов в рюкзаке.