Белорусский государственный технологический университет

Кафедра Информационных Систем и Технологий

**Курс «Математическое программирование»**

**Отчёт по лабораторной работе №4**

**Динамическое программирование**

**Вариант 12**

Выполнила: студентка 2 курса 4 группы

Мергель Каролина Андреевна

Проверил: Харланович Анастасия Владимировна

Минск 2021

**Лабораторная работа 4**

**ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ**

**ЦЕЛЬ РАБОТЫ:** освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.

**Ход выполнения работы**

**Задание 1.** На языке С++ сгенерировать случайным образом строку букв латинского алфавита  длиной  символов и длиной .

**Решение:**

string RandomString(int len)

{

srand(time(0));

string str = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

string newstr;

int pos;

while (newstr.size() != len) {

pos = ((rand() % (str.size() - 1)));

newstr += str.substr(pos, 1);

}

return newstr;

}

int main()

{

cout << "String 300" << endl;

string random\_str = RandomString(300);

cout << random\_str << endl;

cout << "String 250" << endl;

random\_str = RandomString(250);

cout << random\_str << endl;

system("pause");

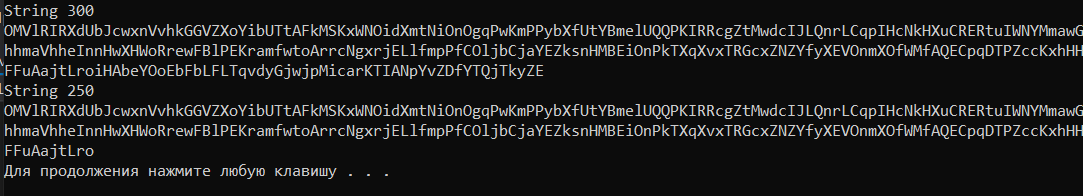
}

Рис.1 – пример генерации строк

**Задание 2.** Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – Левенштейна для , где - длина строки ,  - строка, состоящая из первых  символов строки . (копии экрана и код вставить в отчет).

**Решение:**

Ниже приведены варианты реализации нахождения дистанции Левенштейна при помощи динамического программирования и при помощи рекурсивного алгоритма.

Исходный код реализации через динамическое программирование:

int min3(int x1, int x2, int x3)

{ return std::min(std::min(x1,x2),x3); }

int levenshtein(int lx, const char x[], int ly, const char y[])

{

int \*d = new int[(lx + 1)\*(ly + 1)];

for (int i = 0; i <= lx; i++) DD(i, 0) = i;

for (int j = 0; j <= ly; j++) DD(0, j) = j;

for (int i = 1; i <= lx; i++)

for (int j = 1; j <= ly; j++)

{

DD(i, j) = min3(DD(i - 1, j) + 1, DD(i, j - 1) + 1,

DD(i - 1, j - 1) + (x[i - 1] == y[j - 1] ? 0 : 1));

}

return DD(lx, ly);

}

Пример реализации рекурсивным методом:

int min3(int x1, int x2, int x3)

{ return std::min(std::min(x1,x2),x3); }

int levenshtein\_r(int lx, const char x[],

int ly, const char y[])

{

int rc = 0;

if (lx == 0) rc = ly;

else if (ly == 0) rc = lx;

else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0;

else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1;

else rc = min3(

levenshtein\_r(lx-1, x, ly, y)+1,

levenshtein\_r(lx, x, ly-1, y)+1,

levenshtein\_r(lx-1, x, ly-1, y)+(x[lx-1] == y[ly-1]?0:1)

);

return rc;

};

На рисунке 5 представлены дистанции Левенштейна вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

Рис. 2 – проверка работоспособности решений

**Задание 3.** Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . (копии экрана и график вставить в отчет).

**Решение:**

На графике, представленном на рисунке 3, можно заметить, что выполненные с помощью динамического алгоритма, вычисления производятся в разы быстрее, чем с помощью рекурсивного алгоритма.

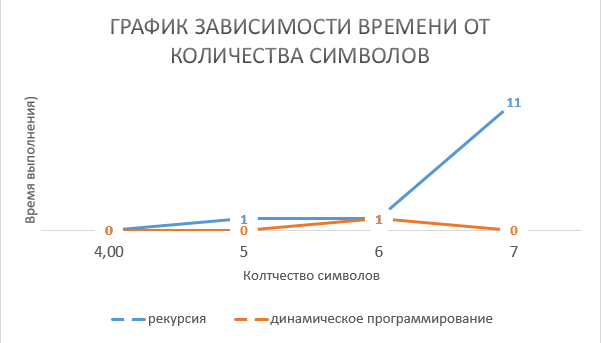


Рис. 3 – сравнительный анализ времени выполнения

**Задание 4.** Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом) (каждый шаг алгоритма по примеру из лекции вставить в отчет).

|  |  |
| --- | --- |
| Задание 4 | |
| Эхо | Хорек |

**Решение:**

1.  
2.  
3.  
4.  
5.  

 = 5.

 = 4.

1.  

 = 4.

 = 3.

1.  
2.  
3.  

 = 3.

 = 2.

1.  
2.  
3.  

 = 2.

 = 1.

1.  

 = 3.

 = 2.

1.  

 = 2.

 = 1.

 = 1.

 = 1.

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 

**Задание 5.**

Выполнить сравнительный анализ времени затраченного на решение задачи об оптимальной расстановке скобок при умножении нескольких матриц для двух методов решения (рекурсивное решение, динамическое программирование). Размерность матриц взять в соответствии с вариантом. Объяснить в отчете принцип расставления скобок по итоговой матрице + код + копии экрана.

|  |
| --- |
| Задание 5 |
| 8\*11, 11\*19, 19\*22, 22\*29, 29\*39, 39\*50 |

**Решение:**

Программный код:

// расстановка скобок (рекурсия)

#define INFINITY 0x7fffffff

#define NINFINITY 0x80000000

int OptimalM(int i, int j, int n, const int c[], int \*s){

#define OPTIMALM\_S(x1,x2) (s[(x1-1)\*n+x2-1])

int o = INFINITY;

int bo = INFINITY;

if (i<j){

for (int k = i; k<j; k++){

bo = OptimalM(i, k, n, c, s) + OptimalM(k + 1, j, n, c, s) + c[i - 1] \* c[k] \* c[j];

if (bo < o){

o = bo;

OPTIMALM\_S(i, j) = k;

}

}

}

else o = 0;

return o;

#undef OPTIMALM\_S

};

// расстановка скобок (динамическое программирование)

int OptimalMD(int n, const int c[], int\* s){

#define OPTIMALM\_S(x1,x2) (s[(x1-1)\*n+x2-1])

#define OPTIMALM\_M(x1,x2) (M[(x1-1)\*n+x2-1])

int \*M = new int[n\*n], j = 0, q = 0;

for (int i = 1; i <= n; i++)

OPTIMALM\_M(i, i) = 0;

for (int l = 2; l <= n; l++){

for (int i = 1; i <= n - l + 1; i++){

j = i + l - 1;

OPTIMALM\_M(i, j) = INFINITY;

for (int k = i; k <= j - 1; k++){

q = OPTIMALM\_M(i, k)+OPTIMALM\_M(k + 1, j)+c[i - 1]\*c[k]\* c[j];

if (q < OPTIMALM\_M(i, j)){

OPTIMALM\_M(i, j) = q;

OPTIMALM\_S(i, j) = k;

}

}

}

}

return OPTIMALM\_M(1, n);

#undef OPTIMALM\_M

#undef OPTIMALM\_S

};

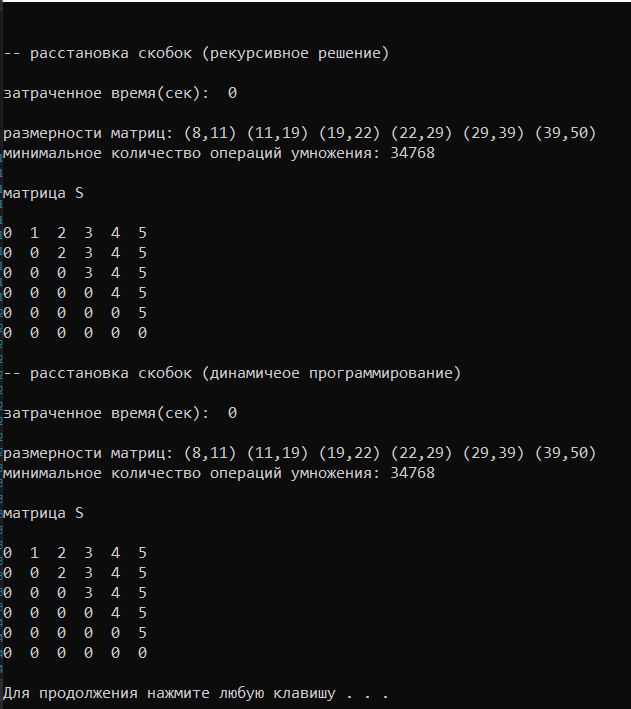


Рис. 4 – результат выполнения программы

Принцип расстановки скобок по итоговой матрице:

Скобки расставляются по принципу «сначала внешние – затем внутренние». Имеется 6 матриц, вот их размерность:

А1=8\*11,

А2=11\*19,

А3=19\*22,

А 4 =22\*29,

А 5 =29\*39,

А 6 =39\*50.

Матрица S:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** |
| **1** | 0 | 1 | 2 | 3 | 4 | 5 |
| **2** | 0 | 0 | 2 | 3 | 4 | 5 |
| **3** | 0 | 0 | 0 | 3 | 4 | 5 |
| **4** | 0 | 0 | 0 | 0 | 4 | 5 |
| **5** | 0 | 0 | 0 | 0 | 0 | 5 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 |

Найдем элемент (1,2) в матрице S, он равен 1. Это означает, что точка разрыва между 1-ой и 6-ой матрицей находится после 5-ой матрицы. Что позволяет расставить скобки следующим образом:

(A1\*A2\*A3\*A4\*A5)\*A6

Точку разрыва между второй и шестой матрицей определяет элемент (1,5). Он равен 4. Следовательно разрыв будет после четвертой матрицы.

((A1\*A2\*A3\*A4)\*A5)\*A6

Далее берем элемент (1,4) и получаем, что он равен 3. Следовательно получаем:

(((A1\*A2\*A3)\*A4)\*A5)\*A6

И на последнем шаге мы возьмем элемент (1,3) и он равен 2:

((((A1\*A2)\*A3)\*A4)\*A5)\*A6

И на последнем шаге мы возьмем элемент (1,2) и он равен 1:

((((A1\*A2)\*A3)\*A4)\*A5)\*A6

Это выражение и есть конечное.

Полученная расстановка скобок позволяет получить минимальное количество операций умножения, равное 34768.