Белорусский государственный технологический университет

Кафедра Программной инженерии

**“Математическое программирование”**

**Отчет по лабораторной работе №4**

**Решение задач методом**

**Динамическое программирование**

**Вариант 4**

Выполнила: студентка 2 курса 1 группы

Самсоник Анастасия Ивановна

Минск 2023

**Лабораторная работа 4**

**ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ**

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

**Ход работы**

***Задание 1.***

**На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной 200.**

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

Код программы:

|  |
| --- |
| #include <iostream>  #include <ctime>  using std::cout;  using std::endl;  char\* GenerateRandomString(int size)  {  char\* str = (char\*)malloc(sizeof(char) \* size);  for (int i = 0; i < size; i++) {  str[i] = rand() % 26 + 'a';  }  return str;  }  int main()  {  clock\_t t1 = 0, t2 = 0, t3 = 0, t4 = 0;  Auxil\* Rand = new Auxil();  unsigned int lenS1 = 300, lenS2 = 200;  float k[7] = { 0.02F, 0.025F, 0.03F, 0.035F, 0.04F, 0.045F, 0.05F };  char\* S1 = new char[lenS1],  \* S2 = new char[lenS2];  cout << endl << "-- Случайно сгенерированные строки -----" << endl;  for (unsigned int i = 0; i < (lenS1 > lenS2 ? lenS1 : lenS2); i++)  {  if(i<lenS1)  S1[i]=(int)Rand->get\_int('a','z');  if(i<lenS2)  S2[i]=(int)Rand->get\_int('a','z');  }  cout << endl << "--строка S1 -----" << endl<< S1 << endl;  cout << endl << "--строка S2 -----" << endl << S2 << endl;} |

Листинг 1. Функция генерации случайной строки

Результат:

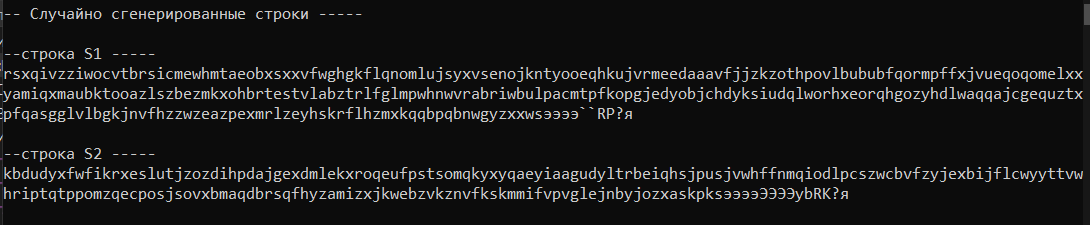
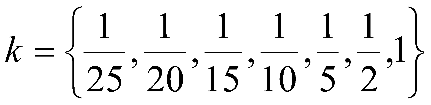


Рисунок 1. Результат генерации строк

***Задание 2.***

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

Реализация через динамическое программирование:

|  |
| --- |
| int min3(int x1, int x2, int x3)  {  return std::min(std::min(x1, x2), x3); //минимальное из трех  }  int levenshtein(int lx, const char x[], int ly, const char y[])  //длина слова х, слово длиной lx  //длина слова y, слово длиной ly  {  int\* d = new int[(lx + 1) \* (ly + 1)];  for (int i = 0; i <= lx; i++) DD(i, 0) = i;  //длина строки х=i (кол-во символов в слове x), а DD возвращ. саму строку  for (int j = 0; j <= ly; j++) DD(0, j) = j;  //длина строки y=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); //возвращает длину слова и строки  } |

Листинг 2.1. Реализация через динамическое программирование

Реализация через рекурсию:

|  |
| --- |
| int levenshtein\_r( //рекурсия  int lx, const char x[],  int ly, const char y[]  )  {  int rc = 0; //результат, дистанция  if (lx == 0) rc = ly; //если длина x=0, то дистанция = длине слова y  else if (ly == 0) rc = lx;  else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0;  //если длина x = длина y = 1 и первые буквы совапали, дистанция = 0  else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1;  //если длина x = длина y = 1 и первые буквы не совапали, дистанция = 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;  }; |

Листинг 2.2. Реализация через рекурсию

Результат:

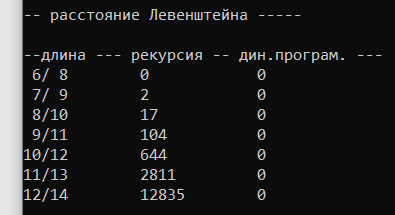


Рисунок 2. Результаты динамического программирования и рекурсии

***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).**

Метод динамического программирования значительно эффективнее, чем рекурсивный метод так, как выполняется значительно быстрее. График построен на результате из рисунка 2.

|  |  |  |
| --- | --- | --- |
| Кол. Символов = k | Время вычисления для динамического программирования | Время вычисления методом рекурсии |
| 0,04 | 0 | 0.00 |
| 0,05 | 0 | 1 |
| 0,0625 | 0 | 17 |
| 0,1 | 0 | 104 |
| 0,2 | 0 | 644 |
| 0,5 | 0 | 2811 |
| 1 | 0 | 12835 |

Рисунок 3. График времени выполнения алгоритмов

***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). Вариант 8.**

|  |  |
| --- | --- |
| Задание 4 | |
| Вар | Баран |

Найти расстояние Левенштейна между словами «Вар» и «Баран».

1. Слова имеют общую часть «ар», поэтому можно строить слово вокруг нее
2. Заменим в начале букву «в» на «б», получив слово «бар». Мы совершили **1** шаг
3. После «бар» добавим букву «а». **2 шага**
4. После «бара» добавим букву «н». **3 шага**
5. Таким образом, мы совершили **3** шага, значит =**3**

1. L(«Вар», «Баран») = min

2. L(«Ва», «Баран») = min

3. L(«Вар», «Бара») = min

4. L(«Ва», «Бара») = min

5. L(«В», «Баран») = min

L(«», «Баран») = 5,

L(«», «Бара») = 4

6. L(«В», «Бара») = min

L(«», «Бара») = 4,

L(«», «Бар») = 3

7. L(«Вар», «Бар») = min

8. L(«Ва», «Бар») = min

9. L(«Вар», «Ба») = min

10. L(«Вар», «Б») = min

L(«Вар», «») = 3,

L(«Ва», «») = 2,

11. L(«Ва», «Ба») = min

12. L(«В», «Ба») = min

L(«», «Ба») = 2,

L(«», «Б») = 1,

13. L(«Ва», «Б») = min

L(«Ва», «») = 2,

L(«В», «») = 1

14. L(«В», «Б») = min

L(«», «Б») = 1,

L(«В», «») = 1,

15. L(«Ва», «Б») = min (2, 3, 2) = 2

16. L(«В», «Ба») = min (3, 2, 2) = 2

17. L(«Ва», «Ба») = min (3, 3, 1) = 1

18. L(«В», «Бар») = min (4, 3, 3) = 3

19. L(«Вар», «Б») = min (3, 4, 3) = 3

20. L(«Вар», «Ба») = min (2, 5, 4) = 2

21. L(«Ва», «Бар») = min (4, 2, 3) = 2

22. L(«Вар», «Бар») = min (3, 3, 2) = 2

23. L(«В», «Бара») = min (5, 4, 4) = 4

24. L(«В», «Баран») = min (6, 5, 5) = 5

25. L(«Ва», «Бара») = min (5, 3, 4) = 4

26. L(«Вар», «Бара») = min (4, 3, 2) = 2

27. L(«Ва», «Баран») = min (6, 4, 5) = 4

28. L(«Вар», «Баран») = min (5, 3, 4) = 3

Дистанция Левенштейна для слов «Вар» и «Баран»: 3.

**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.**

Реализация через рекурсию:

|  |
| --- |
| 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, 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  }; |

Листинг 5.1. Реализация через рекурсию

Реализация через динамическое программирование:

|  |
| --- |
| // 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  }; |

Листинг 5.2. Реализация через динамическое программирование

Результат:

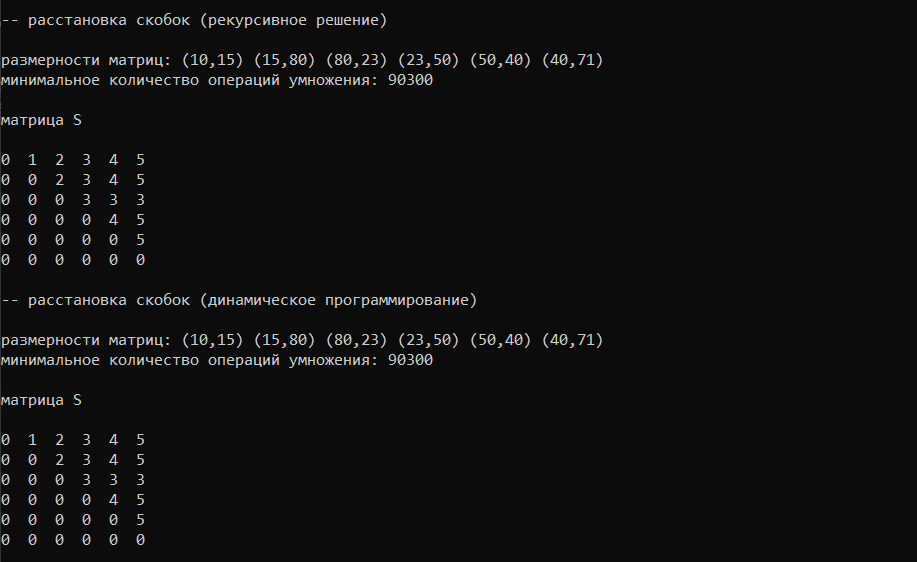


Рисунок 3. Результаты динамического программирования и рекурсии

**Принцип расстановки скобок по итоговой матрице:**

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

А1=10\*15,

А2=15\*80,

А3=80\*23,

А4 =23\*50,

А5 =50\*40,

А6 =40\*71.

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 | 3 | 3 |
| **4** | 0 | 0 | 0 | 0 | 4 | 5 |
| **5** | 0 | 0 | 0 | 0 | 0 | 5 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 |

Найдем элемент (1,6) в матрице S, он равен 5. 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

Найдем элемент (1,5) в матрице S, он равен 4. Это означает, что точка разрыва между 1-ой и 5-ой матрицей находится после 4-ой матрицы. Что позволяет расставить скобки следующим образом:

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

Найдем элемент (1,4) в матрице S, он равен 3. Это означает, что точка разрыва между 1-ой и 4-ой матрицей находится после 3-ой матрицы. Что позволяет расставить скобки следующим образом:

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

Найдем элемент (1,3) в матрице S, он равен 2. Это означает, что точка разрыва между 1-ой и 3-ой матрицей находится после 2-ой матрицы. Что позволяет расставить скобки следующим образом:

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

Это выражение и есть конечное.

Полученная расстановка скобок позволяет получить минимальное количество операций умножения, равное 90300.

**Вывод:** В ходе лабораторной работы были освоены общие принципы решения задач методом динамического программирования, а также проблемы решения задач с рекурсивным методом.