Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Кафедра радіофізики

курс
Системний аналіз

Звіт
про виконання
лабораторної роботи №1

Динамічне програмування.
Оптимальна розстановка дужок примноження послідовності матриць.

виконала
студентка
групи ФеІ — 31
Литвин Віра

перевірив
доц. Вельгош С.Р.

Львів 2013

Мета: Запрограмувати алгоритм оптимальної розстановки дужок при перемножуванні послідовності матриць.

Теоретичні відомості:

Динамічне програмування так само, як і метод розбиття дозволяє розв'язувати поставлену задачу на основі об'єднання розв'язків знайдених для допоміжних задач. Динамічне програмування, як правило, використовується для *задач оптимізації*, для яких можлива наявність багатьох рішень. Кожному варіанту розв'язку можна співставити деяке значення і нам потрібно знайти серед них той, який є оптимальним (мінімальним або максимальним). Такий розв'язок називають *одним із можливих* оптимальних розв'язків.

Процес розробки алгоритмів динамічного програмування можна розбити на 4 нижченаведені етапи:

1. Опис структури оптимального розв'язку.
2. Рекурсивне визначення значення, що відповідає оптимальному розв'язку.
3. Обчислення значення, що відповідає оптимальному рішенню за допомогою методу висхідного аналізу.
4. Отримання оптимального розв'язку на основі інформації, отриманої на попередніх кроках.

Розглянемо приклад застосування динамічного програмування до алгоритму множення послідовності матриць.

Для заданої послідовності матриць $A_1, A_2, ..., A_n$, де $A_i, i = 1,2,...,n$ має розмірність $p_{i-1} \times p_i$, за допомогою дужок потрібно повністю визначити порядок множення $A_1, A_2, ..., A_n$, для якого б кількість скалярних добутків була мінімальною.

При цьому варто зазначити, що саме множення матриць не входить в задачу. Зазвичай час, що затрачається на знаходження оптимального способу розміщення дужок є меншим ніж довелося б виконувати саме множення.

Підрахунок кількості способів розміщення дужок

Через $P(n)$ позначимо кількість альтернатив розміщення дужок в послідовності із n матриць. Якщо $n = 1,$ то матриця тільки одна і дужки можна розмістити тільки одним способом. Якщо ж $n \geq 2,$ то добуток послідовності матриць, в якій порядок повністю визначається дужками є добутком двох таких підпослідовностей матриць, в яких порядок множення також повністю визначається дужками. Розбиття на підпослідовності можна робити на межі $k - ї$ та $k+1-ї$ матриць для довільного $k = 1,2,...,n-1.$ В результаті можемо записати рекурентне співвідношення

$$P(n) = \begin{cases} 1, n = 1 \\ \sum_{k=1}^{n-1} P(k)(n-k), n \geq 2. \end{cases}$$

Можна показати, що його розв'язок є $\Omega(2^n)$. Таким чином кількість варіантів розміщень дужок експоненційно зростає із збільшенням n і метод прямого перебору усіх варіантів не підійде для визначення оптимальної стратегії розміщень дужок.

Перший етап: структура оптимального розміщення дужок

Для зручності позначимо результат множення матриць $A_i A_{i+1} ... A_j$ через $A_{i..j}$, де $i \leq j.$ Якщо задача не тривіальна, тобто $i < j,$ то довільний спосіб розміщення дужок в добуткові

$A_i A_{i+1}...A_j$ розбиває даний добуток між матрицями A_k та A_{k+1}, де k – ціле значення, $i \le k < j$. Таким чином при деякому k і обчислюються матриці $A_{i..k}$ та $A_{k+1..j}$.

Опишемо оптимальну допоміжну підструктуру для даної задачі. Припустимо, що в результаті оптимального розміщення дужок послідовність $A_i A_{i+1}...A_j$ розбивається на послідовності між матрицями A_k та A_{k+1}. Тоді розміщення дужок в першій з них $A_i A_{i+1}...A_k$ має також бути оптимальним. Якби існував більш економний спосіб розміщення там дужок, то його застосування дозволило б ще швидше перемножити $A_i A_{i+1}...A_k$, що суперечить припущенню про оптимальне розміщення дужок на початку. Аналогічними міркуваннями, приходимо до висновку, що і розміщення дужок для послідовності $A_{k+1} A_{k+2}...A_j$ є також оптимальним, як результат оптимального розміщення дужок в $A_i A_{i+1}...A_j$.

Другий етап: рекурсивний розв'язок

Нехай $m[i, j]$ – це мінімальна кількість скалярних добутків, потрібних для знаходження $A_{i..j}$, а $m[1, n]$ для $A_{1..n}$. Визначимо рекурсивно $m[i, j]$ в такий спосіб

$$m[i, j] = \begin{cases} 0, i = j \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1} p_k p_j\}, i < j. \end{cases} \quad (2)$$

Щоб простіше було прослідкувати за процесом побудови оптимального рішення, позначимо через $s[i, j]$ значення k, при якому послідовність $A_i A_{i+1}...A_j$ розбивається на дві підпослідовності в процесі оптимального розбиття дужок.

Третій етап: визначення оптимальної вартості

Замість того, щоб рекурсивно вирішувати співвідношення (2) виконаємо третій крок парадигми динамічного програмування і знайдемо оптимальну вартість, побудувавши таблицю у висхідному напрямкові. В нижче наведеній процедурі будемо вважати, що розміри матриць A_i рівні $p_{i-1} \times p_i (i = 1,2,...,n)$. Вхідні дані представляють собою послідовність $p = \{p_0, p_1,..., p_n\}$, де її довжина $length[p] = n+1$. Щоб коректно реалізувати висхідний підхід, потрібно визначити за допомогою яких записів таблиці будуть вираховуватись значення $m[i, j]$. Із рівняння (2) видно, що вартість $m[i, j]$ знаходження добутку $j - i + 1$ матриць залежить тільки від вартості обчислення послідовностей матриць, що містять менше від $j - i + 1$ матриць.

Спершу в цьому алгоритмові обчислюються величини $m[i, i] \leftarrow 0, \ i = 1,2,...,n,$ які представляють мінімальні вартості для послідовностей одиничної довжини. Потім в першій ітерації циклу за допомогою рекурентного співвідношення (2) обчислюються величини $m[i, i+1]$ при $i = 1,2,...,n-1$ (мінімальні вартості для послідовності $l = 2$). При другому проході цього циклу обчислюються величини $m[i, i+2]$ при $i = 1,2,...,n-2$ (мінімальні вартості для послідовностей довжини $l = 3$) і т.д. На кожному етапі в рядках 9-12 величини $m[i, j]$ залежать тільки від уже обчислених і занесених до таблиці значень $m[i, k]$ і $m[k+1, j]$.

Для прикладу взято 6 матриць, розміри яких рівні $A_1 - 30 \times 35, \ A_2 - 35 \times 15, \ A_3 - 15 \times 5, \ A_4 - 5 \times 10, \ A_5 - 10 \times 20, \ A_6 - 20 \times 25.$

Мінімальна кількість скалярних множень для добутку 6 матриць дорівнює m[1,6]=15125 k=3. Для пояснення цього наведемо приклад. При обрахуванні елемента

$m[5,2]$ використовувались пари елементів, що ідуть від матриць A_2 та A_5.

$$m[2,5] = \min\begin{Bmatrix} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{Bmatrix} = 7125.$$

Четвертий етап: побудова оптимального розв'язку

Оптимальний розв'язок нескладно побудувати за допомогою інформації, що міститься в матриці s.

В кожному елементі $s[i, j]$ зберігається значення індекса k, де при оптимальному розміщенні дужок в послідовності $A_i A_{i+1} ... A_j$ виконується розбиття. Таким чином нам відомо, що оптимальне обрахування добутку матриці $A_{1...n}$ виглядає як $A_{1..s[1,n]} A_{s[1,n]+1..n}$. Часткові добутки матриці можна обчислити рекурсивно, оскільки елемент $s[1, s[1, n]]$ визначає матричний добуток, що виконується останнім при обчислені $A_{s[1,n]+1..n}$.

В результаті до вищенаведеного прикладу буде виведено рядок $((A_1(A_2A_3))((A_4A_5)A_6))$.

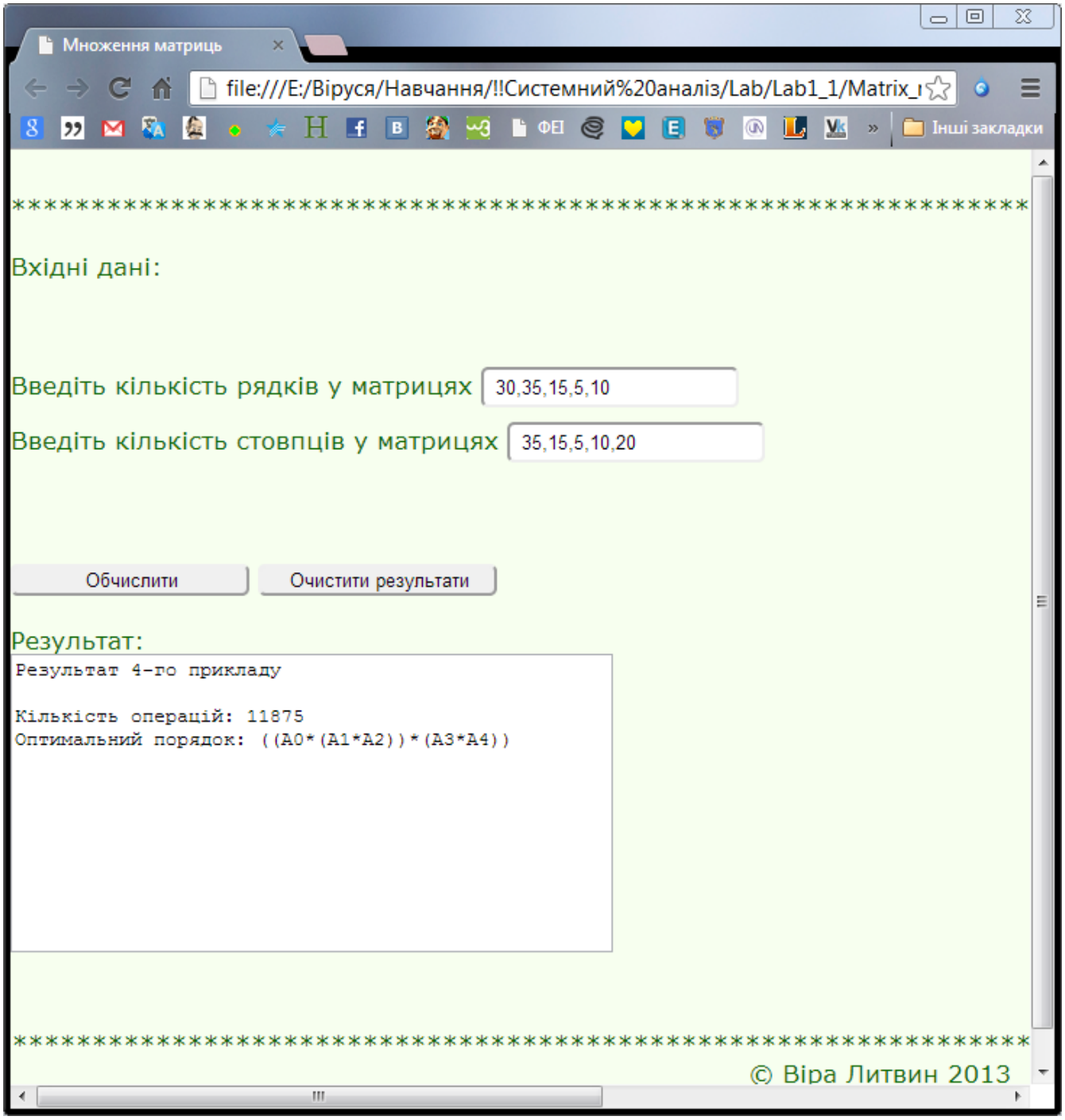
Хід роботи:

В середовищі JetBrains WebStorm створюємо макет веб-сторінки, передбачаємо:

- два поля для вводу розмірностей матриць
- текстове поле для виводу результату
- дві кнопки : виконати обчислення та очистити результат.

Підключаємо до сторінки скрип з функціями для пошуку оптимальної розстановки дужок.

На рисунку нище подано макет сторінки із розвязаним тестовим прикладом.



Код програми:

```
var counter = 0;
function matrixMult()
{
   counter++; // лічильник кількості обрахунків
   var rowsInput = document.getElementById("rows").value;
   var colsInput = document.getElementById("columns").value;
   var output    = document.getElementById("output");
   output.value  += "Результат " + counter + "-го прикладу\n\n";
   rows = rowsInput.split(','); // розділяє введені через кому розмірності
   cols = colsInput.split(',');
   n = rows.length;
   /* Перевірка на можливість множення */
   var isM = true;
   for( i = 0; i < n-1; i ++ )
   {
```

```
      if ( cols[i] != rows[i+1] )
      {
         alert("Множення неможливе! Розмірності не співпадають");
         isM = false;
         break;
      }
   }
   /* оголошуємо масиви */
   matrix = _2DArray(n,n);
   sequence = _2DArray(n,n);

   /* починаємо обчислення */
   for( d = 1; d <= n ; d++)
   {
      for ( i = 0; i < n - d; i++)
      {
         j = i + d;
         matrix[i][j] = 2147483647;
         for( k = i; k <= j - 1;k++)
         {         m = matrix[i][k] + matrix[k+1][j] + rows[i]*cols[k]*cols[j];
            if ( m < matrix[i][j] )
            {
               matrix[i][j] = m;
               sequence[i][j] = k;
            }
         }
      }
   }
   output.value += "Кількість операцій: " + matrix[0][n-1] + "\n";
   output.value += "Оптимальний порядок: " + parenthesize(sequence, 0, n-1) + "\n";
}
//Функція для формування стрічки із записом найкращого розв'язку
function parenthesize(array, i, j)
{
   if ( j > i )
      s = "(" + parenthesize(array, i, array[i][j]) + "*" + parenthesize(array,array[i][j]+1,j) + ")";
   else
      s = characterMap(i);
   return s;
}
//Масив індексів
function characterMap(i)
{
   letters = new
Array("A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","
Y","Z");
   return "A" + i;
//    return letters[i];
}
//Функція для обнулення матриць
function _2DArray(rows,cols)
{
   var i;
   var j;
   var a = new Array(rows);
   for (i=0; i < rows; i++)
```

```
    {
        a[i] = new Array(cols);
        for (j=0; j < cols; j++)
        {
            a[i][j] = 0;
        }
    }
    return(a);
}
function clearOutput()
{
    document.getElementById("output").value = "";
}
```

Висновок:

під час виконання цієї лабораторної роботи було здійснено програмну реалізацію алгоритму оптимального перемноження послідовності матриць, як ми переконались з прикладду порядок у якому ми це робимо справді є суттєвим.