Національний авіаційний університет

Факультет кібербезпеки, комп’ютерної та програмної інженерії

**ЗВІТ**

**по лабораторній роботі No 4**

**Динамічне програмування**

Дисципліна: «Методи оптимізацій та дослідження операцій»

Кафедра: прикладної математики

ОС: бакалавр

Спеціальність: 113 «Прикладна математика»

ОПП: «Прикладне програмне забезпечення»

Виконав: здобувач вищої освіти 4 курсу. 451 групи

Архіпов Олексій Тімурович

Перевірив: Хребет Валерій Григорович

Київ 2024

**Тема:** Динамічне програмування

**Завдання:**

Реалізувати алгоритм динамічного програмування.

**Порядок обчислень**

Динамічне програмування є потужним і ефективним методом вирішення складних обчислювальних задач. Воно базується на розбитті великої задачі на менші незалежні підзадачі, розв’язок яких може бути запам’ятаний і використаний для вирішення більш складних задач.

До класу задач динамічного програмування можуть бути віднесені задачі, вирішені шляхом їх розкладання на частини, тобто невеликі за обсягом та менш складні завдання. У завданнях динамічного програмування залежності між змінними та цільовою функцією можуть мати і нелінійний характер. Найбільш доцільно динамічне програмування застосовувати для розв’язання таких практичних задач, в яких пошук оптимального рішення вимагає покрокового підходу. Планування кожного кроку повинне проводитися з урахуванням загальної вигоди, одержуваної на завершення всього процесу, що дозволяє оптимізувати кінцевий результат управління за обраним критерієм. Принцип динамічного програмування передбачає, що управління на кожному окремому кроці повинно обиратися з урахуванням наслідків від його впровадження в майбутньому, тобто виходячи з інтересів операції в цілому. На практиці цей підхід став поширеним під час довгострокового планування багатоетапних операцій.

Динамічне програмування базується на кількох ключових принципах:

1. Незалежні підзадачі: велика задача розбивається на менші незалежні підзадачі, які можуть бути вирішені окремо. Результати цих підзадач запам’ятовуються і використовуються для подальших обчислень.
2. Запам’ятовування результатів: динамічне програмування запам’ятовує результати підзадач для уникнення повторних обчислень. Це досягається за допомогою таблиць або кешування результатів обчислень.
3. Оптимальна структура: для застосування динамічного програмування задача повинна мати оптимальну структуру, що означає, що розв’язок задачі може бути представлений як комбінація розв’язків менших підзадач.

Алгоритм динамічного програмування може бути реалізований за допомогою наступних кроків:

1. Визначення базових випадків: визначення базових випадків, для яких відомий результат безпосередньо.
2. Рекурсивне визначення значень: визначення значень задачі за допомогою рекурсивного вираховування менших підзадач.
3. Запам’ятовування результатів: запам’ятовування результатів підзадач для уникнення повторних обчислень.
4. Побудова оптимального розв’язку: побудова оптимального розв’язку задачі на основі результатів підзадач.
5. Обчислення результату: обчислення та повернення результату задачі.

Таблиці заповнення використовуються для збереження результатів підзадач та їх використання для вирішення більш складних задач. Такі таблиці зазвичай заповнюються у порядку, що дозволяє обчислити значення кожної підзадачі до обчислення значення вихідної задачі.

Динамічне програмування має ряд переваг, які роблять його корисним та ефективним методом вирішення певних класів завдань. У табличці, що наведена знизу, описані деяки переваги.

|  |  |
| --- | --- |
| ПЕРЕВАГИ | ОПИС |
| Оптимальна підструктура | Однією з основних властивостей динамічного програмування є наявність оптимальної підструктури. Це означає, що оптимальне розв'язання задачі може бути виражене через оптимальні розв'язування її підзавдань. Таким чином, завдання можна розбити на більш маленькі підзавдання, вирішити кожну з них і потім об'єднати їх рішення, щоб отримати оптимальне рішення. |
| Уникнення повторних обчислень | Динамічне програмування використовує запам'ятовування результатів. Рішення для підзавдання зберігаються в пам'яті, щоб уникнути повторних обчислень. Це особливо корисно, коли в задачі виникають підзавдання, що перекриваються. |
| Ефективність часу виконання | Шляхом використання запам'ятовування результатів та перевикористання вже розрахованих значень динамічне програмування може значно скоротити кількість повторних обчислень. Це дозволяє знизити тимчасову складність алгоритму та прискорити його виконання. |
| Універсальність | Динамічне програмування є загальним підходом, який може застосовуватися до широкого спектру завдань, таких як знаходження найбільшої загальної підпослідовності, завдання рюкзака, знаходження найкоротшого шляху та інших. |

Динамічне програмування має певні обмеження :

* Залежність від структури задачі: для застосування динамічного програмування задача повинна мати оптимальну структуру, що не завжди може бути досягнута.
* Потреба у великій кількості пам’яті: використання запам’ятовування результатів може вимагати значної кількості пам’яті, особливо для великих задач.

Принцип оптимальності Беллмана є ключовим поняттям динамічного програмування. Він стверджує, що оптимальне розв'язання задачі може бути представлене у вигляді оптимальних рішень для її підзавдань.

Принцип оптимальності Беллмана формулюється так: "Оптимальне вирішення проблеми включає оптимальні рішення підзадач."

Цей принцип забезпечує основу розробки алгоритмів динамічного програмування. Використовуючи принцип оптимальності Беллмана, завдання розбивається більш дрібні підзавдання, котрим є оптимальні рішення. Потім ці рішення комбінуються для отримання оптимального рішення всього завдання.

Основна умова того, щоб цей принцип здійснювався, така: процес управління повинен бути без зворотного зв’язку, тобто керування на даному кроці не повинно впливати на попередні кроки.

Основне функціональне рівняння Беллмана можна записати у такому вигляді [6]:

*,*

де є сукупністю чисел, що визначають стан системи S на k-му кроці і отримані в результаті керування яке забезпечує перехід системи S із стану в ; U = () - оптимальна стратегія керування; - це прибуток, який отримується при переході будь-якого стану в кінцевий стан при реалізації оптимальної стратегії керування на решті (n-k) кроках.

Динамічне програмування має широкі застосування в різних областях. Наведу декілька прикладів :

1. Задача рюкзака: в задачі рюкзака динамічне програмування може бути використане для вирішення проблеми максимізації ваги предметів, які можуть бути поміщені в рюкзак з обмеженою вмістом.
2. Задача найкоротшого шляху: динамічне програмування може бути застосоване для вирішення задачі пошуку найкоротшого шляху в графі між двома точками. Воно дозволяє знайти оптимальний шлях, обчислюючи найкоротші шляхи від кожної проміжної точки до кінцевої точки.
3. Задача розкладання послідовності: динамічне програмування може бути використане для розкладання послідовності на підпослідовності з максимальною сумою. Це застосування знайшло широке застосування в галузі біоінформатики та аналізу геномів.
4. Задача найкращого порядку обчислень: динамічне програмування може використовуватися для знаходження найкращого порядку обчислень для мінімізації загального часу виконання. Наприклад, у задачах оптимізації матричних множень або обчислення послідовності Фібоначчі.
5. Задача найкращого розбиття послідовності: динамічне програмування може бути використане для знаходження найкращого способу розбиття послідовності на підпослідовності з максимальною сумою. Це може бути застосовано, наприклад, для знаходження найбільшого розбиття стрічки на підстрічки з максимальною сумарною вагою.

Відповідно до обчислювальної схеми динамічного програмування розглянемо спочатку випадок n = 1, тобто всі наявні кошти виділяються до одного підприємства. Позначимо максимально можливий приріст випуску продукції на цьому підприємстві, який відповідає виділеній сумі х. Кожному значенню х відповідає певне значення випуску, тому можна записати:

Нехай тепер n = 2, тобто кошти розподіляються між двома підприємствами. Якщо другому підприємству виділена сума х, то приріст на ньому складе . Кошти, що залишилися іншому підприємству, складуть (с - х) і дозволить збільшити приріст продукції на двох підприємствах:

.

Оптимальному значенню приросту продукції при розподілі суми с між двома підприємствами відповідає таке х, при якому сума , буде максимальною. Це можна записати так:

.

Значення можна обчислити, якщо відомі і т.д.

Функціональне рівняння Беллмана для цієї задачі:

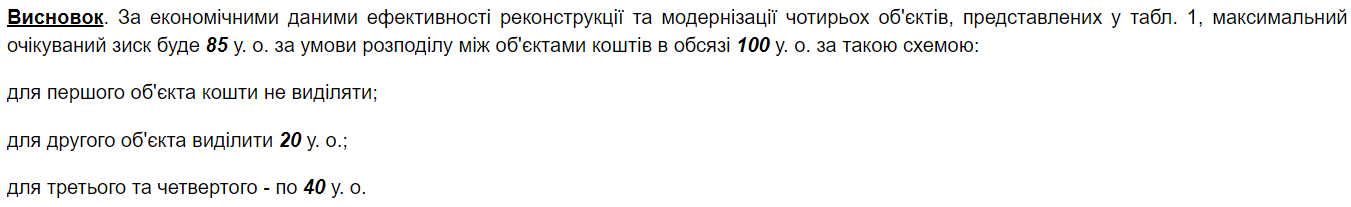
.

**Результати тестування**

Приклад 1:



Результат:



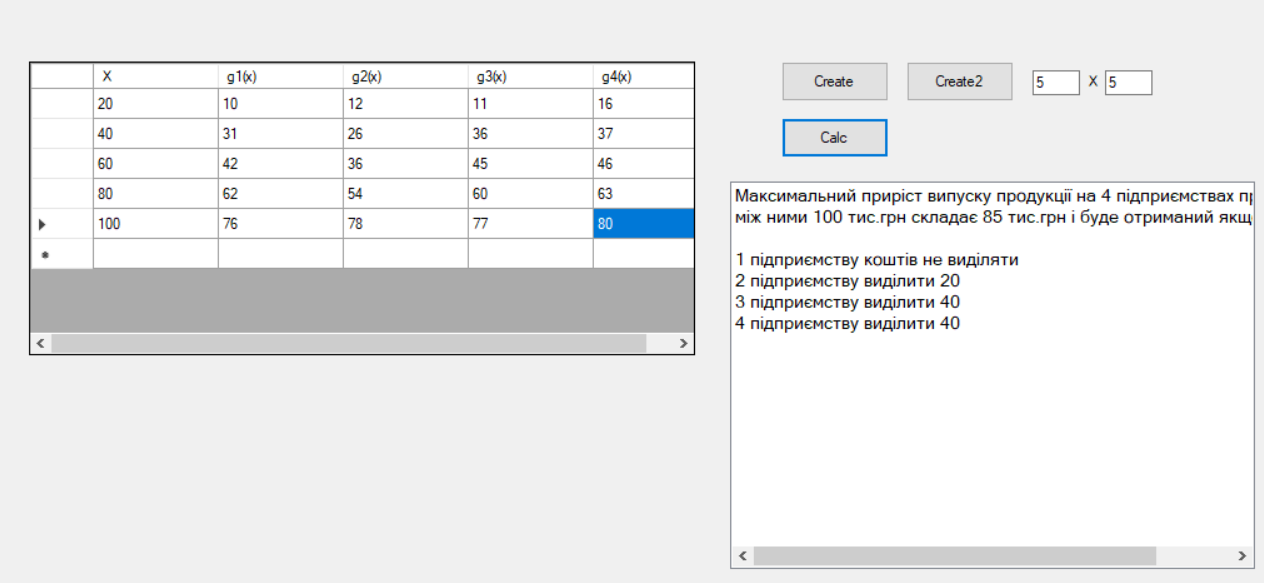
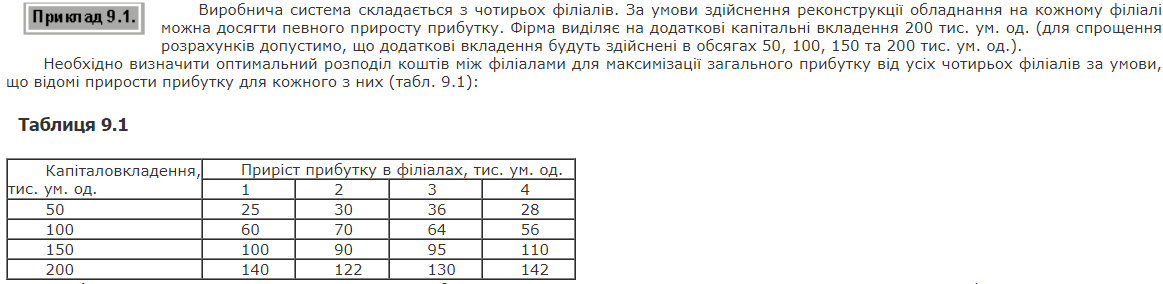


Рис.1. Приклад 1

Приклад 2:



Результат:



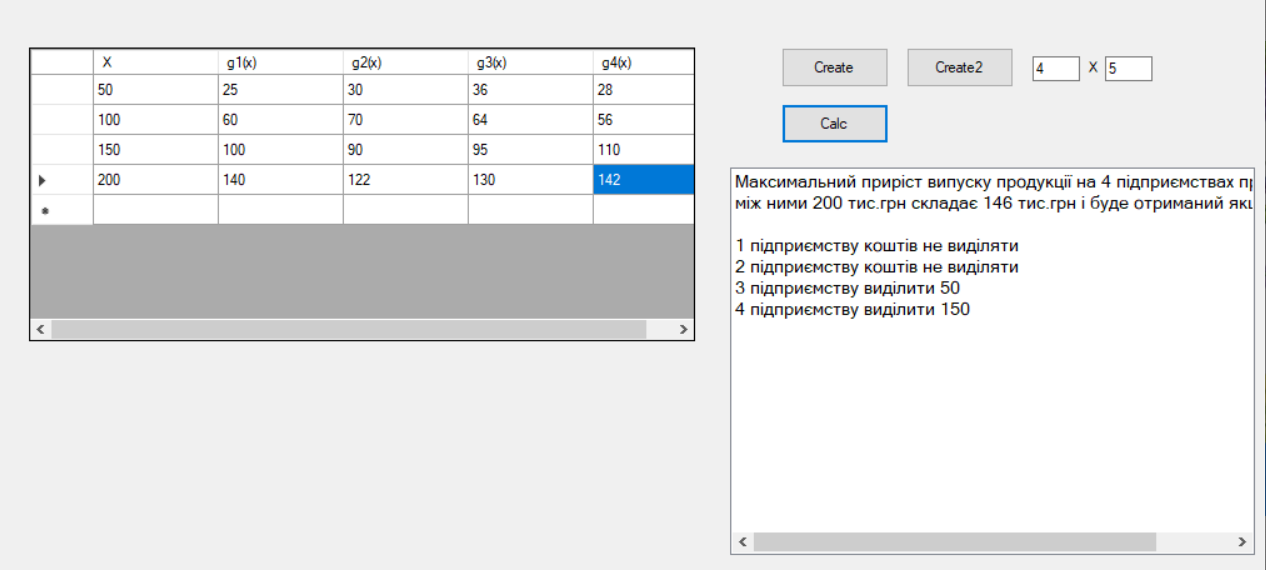


Рис.2. Приклад 2

Всі результати роботи програми співпадали з розв’язками задач у джерелах інтернету.

**Висновоки**

* Мною була створена програма для вирішення задачі про оптимальний розподіл коштів.
* За допомогою динамічного програмування визначив оптимальний розподіл капіталовкладень, що дозволить максимізувати загальний приріст випуску продукції.
* Порівняв результати роботи програми з вирішеними прикладами з джерел інтернету.
* Результат перевірки показав, що програма працює коректно.

**Код програми**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

int n, m;

double[,] ReadMatrixFromGrid()

{

n = Convert.ToInt32(tbn.Text);

m = Convert.ToInt32(tbm.Text);

double[,]matrix = new double[n, m];

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

matrix[i, j] = Convert.ToDouble(dgMatrix.Rows[i].Cells[j].Value);

}

}

return matrix;

}

void Create()

{

int MatrixSize = 5;

tbn.Text = "5";

tbm.Text = "5";

dgMatrix.Rows.Clear();

dgMatrix.Columns.Clear();

int i = 0;

for (i = 0; i < MatrixSize; i++)

{

if (i == 0) dgMatrix.Columns.Add("X", "X");

else dgMatrix.Columns.Add("g" + i.ToString() + "(x)", "g" + i.ToString() + "(x)");

}

dgMatrix.Rows.Add(MatrixSize);

i = 0;

dgMatrix.Rows[i].Cells[0].Value = "20";

dgMatrix.Rows[i].Cells[1].Value = "19";

dgMatrix.Rows[i].Cells[2].Value = "14";

dgMatrix.Rows[i].Cells[3].Value = "20";

dgMatrix.Rows[i].Cells[4].Value = "25";

i = 1;

dgMatrix.Rows[i].Cells[0].Value = "40";

dgMatrix.Rows[i].Cells[1].Value = "36";

dgMatrix.Rows[i].Cells[2].Value = "32";

dgMatrix.Rows[i].Cells[3].Value = "36";

dgMatrix.Rows[i].Cells[4].Value = "53";

i = 2;

dgMatrix.Rows[i].Cells[0].Value = "60";

dgMatrix.Rows[i].Cells[1].Value = "51";

dgMatrix.Rows[i].Cells[2].Value = "52";

dgMatrix.Rows[i].Cells[3].Value = "47";

dgMatrix.Rows[i].Cells[4].Value = "66";

i = 3;

dgMatrix.Rows[i].Cells[0].Value = "80";

dgMatrix.Rows[i].Cells[1].Value = "72";

dgMatrix.Rows[i].Cells[2].Value = "61";

dgMatrix.Rows[i].Cells[3].Value = "72";

dgMatrix.Rows[i].Cells[4].Value = "70";

i = 4;

dgMatrix.Rows[i].Cells[0].Value = "100";

dgMatrix.Rows[i].Cells[1].Value = "81";

dgMatrix.Rows[i].Cells[2].Value = "79";

dgMatrix.Rows[i].Cells[3].Value = "80";

dgMatrix.Rows[i].Cells[4].Value = "84";

}

void Calc()

{

double[,] matrix = ReadMatrixFromGrid();

List<double> invest = new List<double>();

List<double> invest1 = new List<double>();

invest1.Add(0);

List<List<double>> P\_n = new List<List<double>>();

int c = 1;

for(int i = 0; i < n; i++)

{

invest.Add(matrix[i, 0]);

invest1.Add(matrix[i, 0]);

List<double> P\_i = new List<double>();

if(c < m)

{

for (int j = 0; j < n; j++)

{

P\_i.Add(matrix[j, c]);

}

P\_n.Add(P\_i);

c++;

}

}

List<Tuple<List<double>, List<double>>> tuples = new List<Tuple<List<double>, List<double>>>();

Tuple<List<double>, List<double>> t = new Tuple<List<double>, List<double>>(invest, P\_n[0]);

tuples.Add(t);

c = 1;

int c2 = 0;

for(int k = 0; k < P\_n.Count - 1; k++)

{

double[,] m = new double[invest.Count, invest.Count + 1];

for (int i = 0; i < invest.Count; i++)

{

for (int j = 0; j < invest1.Count; j++)

{

if (j == 0) m[i, j] = tuples[k].Item2[i];

else

{

double res = invest[i] - invest1[j];

if (res < 0) break;

else

{

if (res == 0) m[i, j] = P\_n[c][i];

else

{

m[i, j] = P\_n[c][j - 1] + tuples[k].Item2[c2];

c2--;

}

}

}

}

c2 = i;

}

c++;

List<double> temp1 = new List<double>(), temp2 = new List<double>();

for (int i = 0; i < invest.Count; i++)

{

List<double> temp = new List<double>();

for (int j = 0; j < invest1.Count; j++)

{

temp.Add(m[i, j]);

}

temp1.Add(temp.Max());

temp2.Add(invest1[temp.IndexOf(temp.Max())]);

}

t = new Tuple<List<double>, List<double>>(temp2, temp1);

tuples.Add(t);

}

double max\_number = tuples[tuples.Count - 1].Item2.Max();

int index = tuples[tuples.Count - 1].Item2.IndexOf(max\_number);

double max\_index = tuples[tuples.Count - 1].Item1[index];

double ostatok = invest.Max() - max\_index;//60

List<double> number = new List<double>();

List<double> ind = new List<double>();

ind.Add(max\_index);

c = 0;

for (int i = tuples.Count - 2; i >= 0; i--)

{

int k = invest.IndexOf(ostatok);

double n = tuples[i].Item2.GetRange(0, k+1).Max();

index = tuples[i].Item2.IndexOf(n);

double inde = tuples[i].Item1[index];

ostatok = ostatok - inde;

number.Add(n); ind.Add(inde);

if(ostatok == 0 && i != 0)

{

c = i;

do

{

ind.Add(0);

c--;

} while (c != 0);

break;

}

}

listBox1.Items.Clear();

listBox1.Items.Add("Максимальний приріст випуску продукції на " + tuples.Count + " підприємствах при розподілі");

listBox1.Items.Add("між ними " + invest.Max() + " тис.грн складає " + max\_number + " тис.грн і буде отриманий якщо:");

listBox1.Items.Add("");

c = 1;

for(int i = ind.Count - 1; i >= 0; i--)

{

if (ind[i] == 0) listBox1.Items.Add(c + " підприємству коштів не виділяти");

else listBox1.Items.Add(c + " підприємству виділити " + ind[i]);

c++;

}

}

private void btnCreate\_Click(object sender, EventArgs e)

{

Create();

}

private void btnCalc\_Click(object sender, EventArgs e)

{

Calc();

}

private void btnCreate2\_Click(object sender, EventArgs e)

{

n = Convert.ToInt32(tbn.Text);

m = Convert.ToInt32(tbm.Text);

dgMatrix.Rows.Clear();

dgMatrix.Columns.Clear();

int i = 0;

for (i = 0; i < m; i++)

{

if (i == 0) dgMatrix.Columns.Add("X", "X");

else dgMatrix.Columns.Add("g" + i.ToString() + "(x)", "g" + i.ToString() + "(x)");

}

dgMatrix.Rows.Add(n);

}

}

}