

Laboratorium 6

Mateusz Cyganek

Algorytm przejścia do postaci macierzy górnej trójkątnej

Algorytm wykonuje 3 operacje:

A

Operacja uzyskania mnożnika dla wiersza komórki 1, do odejmowania od wiersza komórki 2.

```
protected void A(Production production)
{
    Multipliers[production.Pass] =
        Matrix[production.Cell1] / Matrix[production.Cell2];
}
```

B

Operacja mnożenia komórki przez podaną wartość, do odejmowania w operacji C.

```
protected void B(Production production)
{ Matrix[production.Cell1] *= Multipliers[production.Pass]; }
```

C

Operacja odjęcia wartości komórki 2 od komórki 1.

```
protected void C(Production production)
{ Matrix[production.Cell1] -= Matrix[production.Cell2]; }
```

Dzięki operacji A uzyskujemy mnożnik, na całym rzędzie wykonujemy operacje B i C, gdzie B mnoży komórkę przez wartość zwróconą przez operację A, a operacja C odejmuje wartość od komórki w zerowanym rzędzie. Ponieważ $\text{Matrix}[\text{production.Cell1}] * (\text{Matrix}[\text{production.Cell1}] / \text{Matrix}[\text{production.Cell2}]) = \text{Matrix}[\text{production.Cell2}]$, mamy gwarancję, że wyzerujemy odpowiednią komórkę. Algorytm wywołuje te operacje w odpowiedniej kolejności, zerując komórki pod przekątną.

Jedynie zapisując te operacje (`MatrixSolverProductions`), uzyskamy słowo oraz alfabet. Na chi podstawie badając zależności pomiędzy nimi (`Production.IsDependentOn(Production other)`) otrzymamy relacje zależności, niezależności oraz postać Normalną Foaty.

Dzięki postaci normalnej otrzymujemy operacje które można wywołać współbieżnie. Wizualizacja takich grup zadań jest reprezentowana na grafie odpowiednimi kolorami. Kolorem **niebieskim** oznaczono zadania wykonywane wcześniej, a **różowym** – później.

Realizacja zadania

Program zrealizowano w .NET 6.0 i Python 3.10.

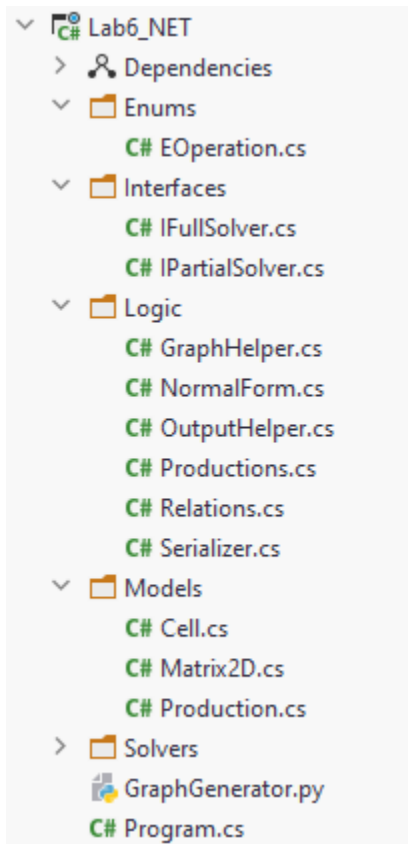
Część .NET dla podanego pliku z danymi wejściowymi tworzy plik z rezultatami w którym znajduje się:

- Alfabet A
- Słowo w
- Relacja zależności D
- Relacja niezależności I
- Postać normalna Foaty
- Rozwiązanie macierzy w postaci górnej trójkątnej i jednostkowej

Część Python generuje obrazek grafu zależności.

Projekt .NET

Projekt .NET zawiera wszystkie klasy. Większość klas i metod zawiera opisy i komentarze.



```
public static Matrix2D SerializeMatrix(string file)
in class Serializer
```

Metoda wczytująca plik i zwracająca macierz

Params: **file** – plik wejściowy

Returns: macierz z wektorem

Exceptions: **Exception** – błąd jeżeli plik nie istnieje

Program

Zawiera metodę main.

Wywołuje wszystkie inne metody i zapisuje wyświetlane informacje do pliku który zostanie utworzony w podanej ścieżce, oraz generuje obrazek grafu zależności.

Interfejsy

Projekt zawiera dwa interfejsy

IPartialSolver

Rozwiązuje macierz do postaci górnej trójkątnej.

IFullSolver

Rozwiązuje macierz postaci górnej trójkątnej do postaci jednostkowej.

Solvers

Folder zawiera klasy rozwiązujące macierze do postaci zależnych od implementowanych interfejsów.

MatrixSolver

Podstawowy solver.

Rozwiązuje synchronicznie macierz do postaci górnej trójkątnej.

Zawiera metody A, B i C, które wykonują operacje na macierzach.

```
public void SolvePartially()
{
    var pass = 0;
    for (var i = 0; i < Matrix.Size - 1; i++)
        for (var k = i + 1; k < Matrix.Size; k++, pass++)
        {
            Invoke(new(EOperation.A, new Cell (i, i), new Cell (k, i), pass));

            for (var j = 0; j < Matrix.Size + 1; j++)
            {
                Invoke(new(EOperation.B, new Cell(k, j), Cell.Empty, pass));
                Invoke(new(EOperation.C, new Cell(k, j), new Cell(i, j), pass));
            }
        }
}
```

MatrixSolverFull

Rozszerzenie podstawowego solver'a.

Rozwiązuje synchronicznie macierz postaci górnej trójkątnej do postaci jednostkowej.

```
public void FinishSolving()
{
    var pass = 0;
    for (var i = Matrix.Size - 1; i >= 0; i--, pass++)
    {
        A(new Production(EOperation.A, new (i, i), Cell.Empty, pass));
        B(new Production(EOperation.B, new (i, Matrix.Size), Cell.Empty,
pass));
        B(new Production(EOperation.B, new (i, i), Cell.Empty, pass));
        for (var k = i - 1; k >= 0; k--)
        {
            C(new Production(EOperation.C, new (k, Matrix.Size), new (k, i),
pass));
            C(new Production(EOperation.C, new (k, i), new (k, i), pass));
        }
    }
}
```

MatrixSolverProductions

Rozszerzenie podstawowego solver'a.

Tworzy słowo – listę produkcji potrzebną do przekształcenia macierzy do postaci górnej trójkątnej.

```
public new void SolvePartially()
{
    var pass = 0;
    for (var i = 0; i < Matrix.Size - 1; i++)
    for (var k = i + 1; k < Matrix.Size; k++, pass++)
    {
        Productions.Add(new (EOperation.A, new (i, i), new (k, i), pass));

        for (var j = 0; j < Matrix.Size + 1; j++)
        {
            Productions.Add(new (EOperation.B, new (k, j), Cell.Empty,
pass));
            Productions.Add(new (EOperation.C, new (k, j), new (i, j), pass));
        }
    }
}
```

MatrixSolverAsync

Rozszerzenie pełnego solver'a.

Rozwiązuje macierz do postaci górnej trójkątnej wielowątkowo na podstawie postaci normalnej Foaty.

Wszystkie operacje danego poziomu wykonywane są współbieżnie.

```
public new void SolvePartially()
{
    // dla każdego poziomu FNF który zawiera operacje od siebie niezależne
    foreach (var level in _fnf)
        // wywołuje i oczekuje wykonania wszystkich operacji danego poziomu
        Task.WaitAll(level.Select(Invoke).ToArray());
}
```

Task tworzony i uruchamiany jest w przeciążeniu metody Invoke()

```
private new async Task Invoke(Production production)
{ await Task.Run(() => base.Invoke(production)); }
```

Models

Folder zawiera klasy opisujące obiekty i metody którymi posługuje się algorytm

- Cell – rekord opakowujący informacje o rzędzie i kolumnie macierzy
- Matrix2D – reprezentacja macierzy, zawiera metody do wypisywania i implementuje interfejs umożliwiający tworzenie kopi macierzy
- Production – rekord w którym zawarte są wszystkie przydatne informacje o produkcji, zawiera metodę IsDependentOn() zwracającą wartość bool informującą o tym czy dana produkcja jest zależna a od innej

Logic

Folder zawiera klasy pomocnicze.

NormalForm

Klasa ta generuje postać normalną Foaty na podstawie słowa, które reprezentowane jest jako lista produkcji

```
// po każdej produkcji
for (var i = 0; i < word.Count; i++)
{
    // pominię jeżeli już została wykorzystana
    if (elements[i].Used)
        continue;

    // wstawiamy element do poziomu FNF
    MarkUsed(elements, i, layer, passesA, passesB);

    // dla wszystkich kolejnych produkcji
    for (var j = i + 1; j < word.Count; j++)
        // jeżeli można wykonać produkcje j współbieżnie z i
        if (IsConcurrent(elements, j, layer, passesA, passesB))
            // dodajemy produkcje j do poziomu FNF
            MarkUsed(elements, j, layer, passesA, passesB);

    // wstawiamy wygenerowany poziom do FNF
    Fnf.Add(layer.Select(x => x.Production).ToList());
    // czyścimy warstwę roboczą
    layer.Clear();
}
```

Productions

Klasa jedynie wypisuje alfabet i słowo w odpowiednim formacie.

Relations

Klasa sprawdza zależności produkcji i przypisuje je do odpowiedniej listy którą zwraca w odpowiednim formacie.

Inne

- GraphHelper – wywołuje skrypt Python który tworzy obrazek grafu zależności
- OutputHelper – metody pomocnicze do wypisywania i zapisywania informacji
- Serializer – wczytuje macierz z podanego pliku

Python

Skrypt pythona'a, wywoływany przez program z ścieżką do pliku z .tmp generuje obrazek z grafem zależności. Plik zawiera komentarze.

Zależności skryptu:

- Os
- Sys
- Matplotlib.pyplot
- Networkx

Wywołanie programu

Program należy wywołać uruchamiając w terminalu program z ścieżką pliku zawierającego dane wejściowe.

```
C:\lab6> .\Lab6_NET_Release\Lab6_NET.exe "C:\lab6\input.txt"
```

Rezultaty

Jeżeli dane podane w pliku były poprawne program zwróci następujące informacje:

```
Wczytana macierz:
[      2.0      1.0      3.0 |      6.0 ]
[      4.0      3.0      8.0 |     15.0 ]
[      6.0      5.0     16.0 |     27.0 ]

Alfabet produkcji:
A = {
A([0, 0], [1, 0]),
B([1, 0]),
C([1, 0], [0, 0]),
.
.
C([2, 2], [1, 2]),
B([2, 3]),
C([2, 3], [1, 3])
}

Słowo:
w =
A([0, 0], [1, 0])
B([1, 0])
C([1, 0], [0, 0])
.
.
C([2, 2], [1, 2])
B([2, 3])
C([2, 3], [1, 3])
```

Relacje zaleznosci:

```
D = sym{
[A([0, 0], [1, 0]), A([0, 0], [1, 0])]
[B([1, 0]), A([0, 0], [1, 0])]
[C([1, 0], [0, 0]), A([0, 0], [1, 0])]
.
.
[B([2, 1]), A([0, 0], [2, 0])]
[C([2, 1], [0, 1]), A([0, 0], [1, 0])]
[C([2, 1], [0, 1]), B([1, 0])]
.
.
[C([2, 3], [1, 3]), B([2, 1])]
[C([2, 3], [1, 3]), B([2, 2])]
[C([2, 3], [1, 3]), B([2, 3])]
}
```

Relacje niezaleznosci:

```
I = sym{
[A([0, 0], [1, 0]), B([1, 0])]
[A([0, 0], [1, 0]), C([1, 0], [0, 0])]
[A([0, 0], [1, 0]), B([1, 1])]
.
.
[C([2, 3], [0, 3]), B([2, 3])]
[C([2, 3], [0, 3]), C([2, 3], [1, 3])]
[A([1, 1], [2, 1]), A([0, 0], [1, 0])]
.
.
[C([2, 3], [1, 3]), C([2, 1], [1, 1])]
[C([2, 3], [1, 3]), C([2, 2], [1, 2])]
[C([2, 3], [1, 3]), C([2, 3], [1, 3])]
}
```

Postac normalna Foaty:

```
FNF([w]) =
[A([0, 0], [1, 0]) A([0, 0], [2, 0])]

[B([1, 0]) B([1, 1]) B([1, 2]) B([1, 3]) B([2, 0]) B([2, 1]) B([2, 2]) B([2, 3])]

[C([1, 0], [0, 0]) C([1, 1], [0, 1]) C([1, 2], [0, 2]) C([1, 3], [0, 3]) C([2, 0], [0,
0]) C([2, 1], [0, 1]) C([2, 2],
[0, 2]) C([2, 3], [0, 3])]

[A([1, 1], [2, 1])]

[B([2, 0]) B([2, 1]) B([2, 2]) B([2, 3])]

[C([2, 0], [1, 0]) C([2, 1], [1, 1]) C([2, 2], [1, 2]) C([2, 3], [1, 3])]
```

Oczekiwany wynik:

[2.0	1.0	3.0	6.0]
[.0	.5	1.0	1.5]
[.0	.0	.75	.75]
[1.0	.0	.0	1.0]
[.0	1.0	.0	1.0]
[.0	.0	1.0	1.0]

Otrzymany (współbieżnie) wynik:

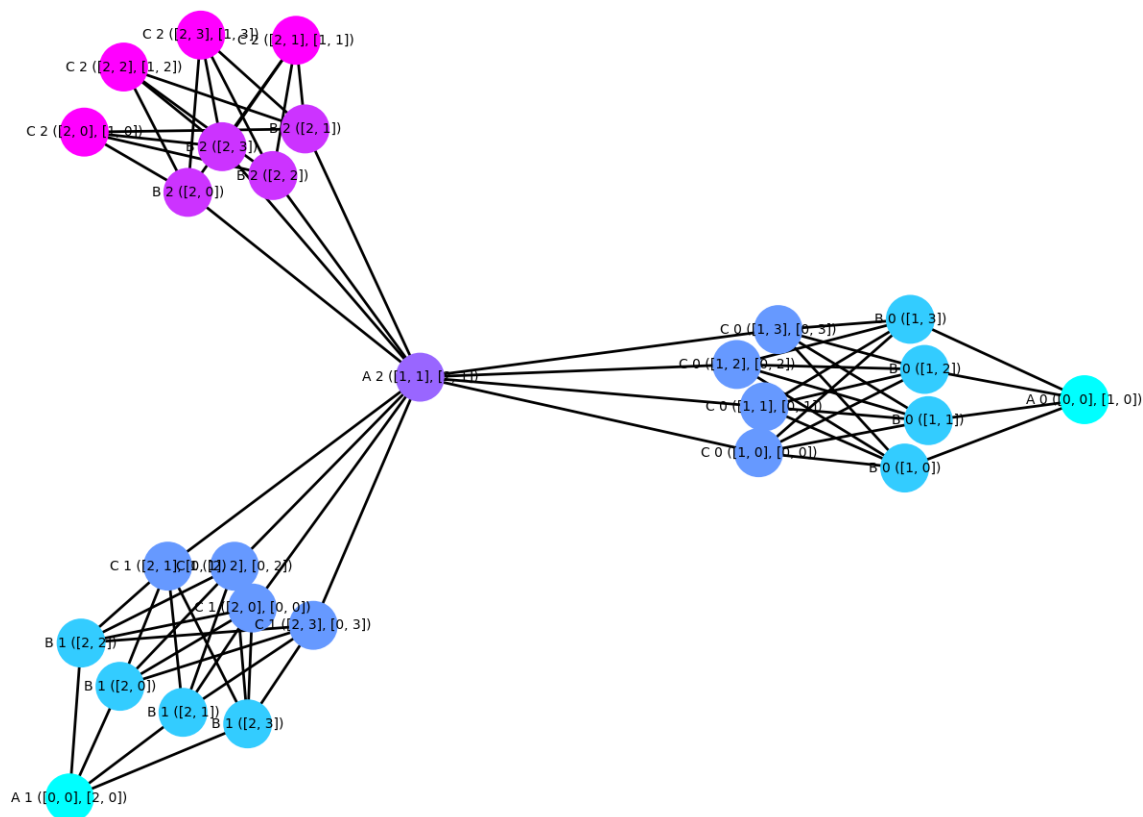
[2.0	1.0	3.0		6.0]
[.0	.5	1.0		1.5]
[.0	.0	.75		.75]

[1.0	.0	.0		1.0]
[.0	1.0	.0		1.0]
[.0	.0	1.0		1.0]

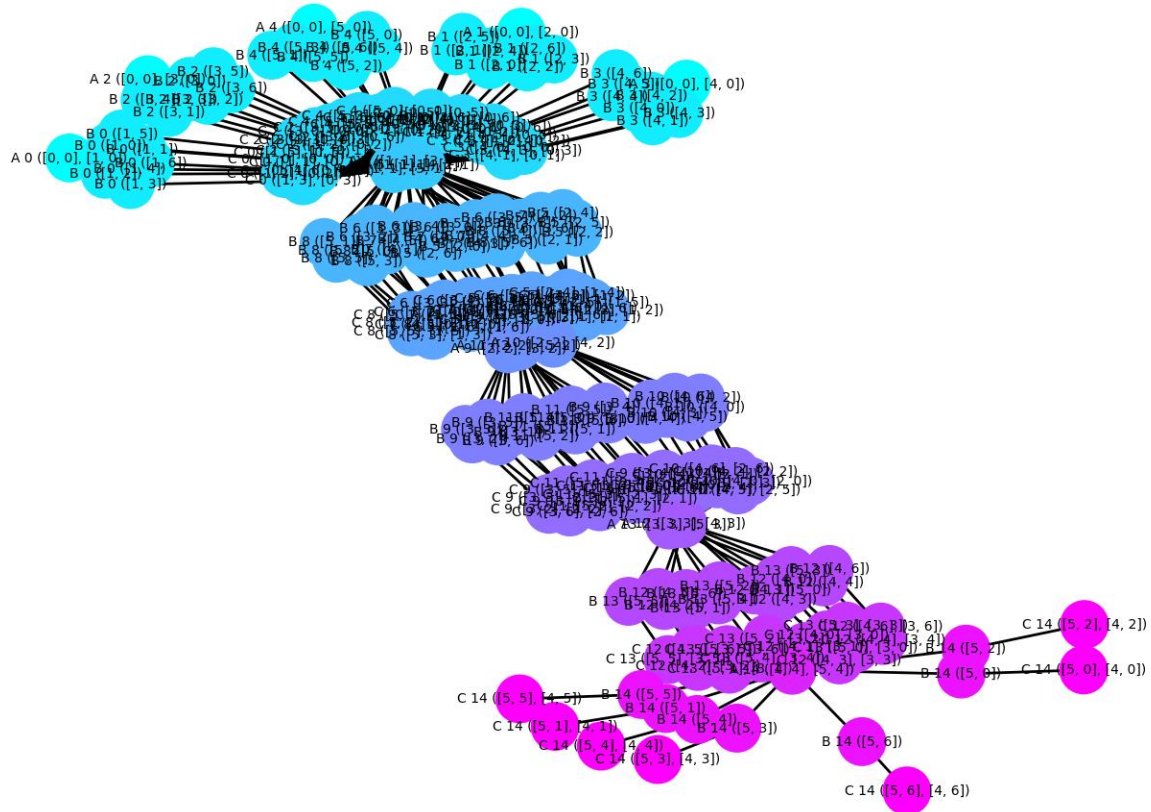
Wynik zapisano do: 'C:\lab6\input_results.txt'

Rozwiązanie zapisano do: 'C:\lab6\input_solution.txt'

Graf zapisano do: 'C:\lab6\input_graph.png'



Graf macierzy 6x6



Przyjęty format

1. <rozmiar macierzy>
2. <1-szy rząd macierzy>
3. <2-gi rząd macierzy>

.

.

.

n. <wektor>