

## **Ֆլոյդ-Ուորշըլի ալգորիթմի մեքենայացումը**

**Խնդիր:** Կարճագույն ճանապարհի որոնում

**Ուսանող:** ԻԿՄ / ՀԳ / 3-րդ կուրս/ 608 / Սևակ Ամիրխանյան

## Խնդրի դրվածքը

Մեր խնդիրն է գտնել կամայական  $i, j$  գագաթների միջև կարճագույն ճանապարհը, եթե այն գոյություն ունի և զուգահեռ կառուցել նաև այդ ճանապարհը:

Ալգորիթմը մեքենայացված է **C#** լեզվով և աշխատում է **.NET Core** միջավայրում:

Մանրամասն դիտարկենք այն տվյալների կառուցվածքները և մոդելները, որոնք օգտագործվել են ալգորիթմը իրականացնելու ժամանակ:

### Գրաֆի գագաթի մոդելավորումը մեքենայում

```
// Գրաֆի գագաթը ներկայացնող դաս:  
// Դասը հայտնի գաղափար է ՕԿԾ-ում:  
public class Vertex  
{  
    // Գրաֆը բնութագրող համար, որը բնական թիվ է:  
    public int Number { get; set; }  
  
    // Վերոնշյալ ֆունկցիաները / պրոցեդուրաները լեզու-սպեցիֆիկ  
    // կոնստրուկցիաներ են և բուն ալգորիթմի իմպլեմենտացիայի մեջ  
    // մեծ դեր չունեն:  
    // Վերադարձնում է գագաթի հեշ կոդը: Սա անհրաժեշտ է գագաթները  
    // հեշ աղյուսակում պահելու համար:  
    public override int GetHashCode()  
    {  
        return this.Number.GetHashCode();  
    }  
  
    // Համեմատում է գրաֆի 2 գագաթներ ըստ իրենց հեշ կոդերի:  
    // Անհրաժեշտ է հեշավորող ֆունկցիայի համար:  
    public override bool Equals(object obj)  
    {  
        return this.GetHashCode() == obj.GetHashCode();  
    }  
}
```

### Գրաֆի կողի մոդելավորումը մեքենայում

```
// Գրաֆի կողը ներկայացնող կլաս  
public class Edge  
{  
    // Այն գագաթը որտեղից կողը սկսում է
```

```

public Vertex First { get; set; }

// Այն գագաթը որտեղ գնում է կողը
public Vertex Second { get; set; }

// Կողի ուղղությունը ` կարող է ընդունել 3 հնարավոր արժեք`
// 1 -> 2, 2 -> 1 կամ 1 <-> 2
public EdgeDirection Direction { get; set; }

// Կողի երկարությունը
public double Length { get; set; }

// Որոշ ֆունկցիաներ կողերի հետ գործողություններ անելու համար:
// Ստուգում է կողերի հավասար են:
// Տվյալ մոդելավորման մեջ կողերը հավասար են, եթե դրանք
// սկսում են նույն գագաթում, գնում են հապատասխան գագաթ
// և ունեն նույն երկարությունը
public override bool Equals(object obj)
{
    var edge = obj as Edge;

    return this.First == edge.First &&
           this.Second == edge.Second &&
           this.Direction == edge.Direction;
}

// Վերադարձնում է կողի հեշ կոդը
public override int GetHashCode()
{
    return this.First.GetHashCode() ^ this.Second.GetHashCode();
}

// Ստուգում է կողերի նույնությունը տրված ուղղությամբ:
// Հնարավոր է, որը կողի ուղղությունը հայտնի չլինի կանչի ընթացքում
public bool IsEqual(Edge edge, EdgeDirection? d = null)
{
    return this.First == edge.First &&
           this.Second == edge.Second &&
           this.Direction == (d == null ? edge.Direction : d);
}
}

// Ստորուկտուրաներ, որոնց առավել մանրամասն նկարագրությունը կարելի գտնել
// իմ GitHub-յան էջում`
// https://github.com/amirkhaniansev/fw-algorithm/
class VerticesCollection;

```

```

class EdgesCollection;
class Graph;
class Matrix<T>;
class SquareMatrix<T>;

// Հարևանության մատրիցը նկարագրող ստրուկտուրա
class AdjacencyMatrix;

// Ալգորիթմի քայլը նկարագրող մոդել
public class Step
{
    public int Number { get; set; }

    // Հարևանության մատրիցի
    public SquareMatrix<Cell<double>> A { get; set; }

    // Օժանդակ մատրիցի ճանապարհը կառուցելու համար
    public SquareMatrix<Cell<int>> B { get; set; }
}

// Մատրիցի բջիջը / վանդակը նկարագրող մոդել
public class Cell<TValue>
{
    // Բջիջի գույնը
    public Color Color { get; set; }

    // Բջիջի կոորդինատները
    public Point Point { get; set; }

    // Բջիջի արժեքը
    public TValue Value { get; set; }

    public Cell(Color color, Point point, TValue value)
    {
        this.Color = color;
        this.Point = point;
        this.Value = value;
    }
}

// Այժմ նկարագրենք ալգորիթմի իրականացումը:
// ՈՒՇԱՂՐՈՒԹՅՈՒՆ՝ Ֆլոյդ-Ուորշլի ալգորիթմի այս իմպլեմենտացումն ունի
// ուսուցողական, վերլուծական և դեմոնստրատիվ բնույթ, ինչի
// արդյունքում հնարավոր է, որ այն չհապատասխանի արտադրական ստանդարտներին:

```

```

// Միանշանակ հնարավոր է ավելի էֆեկտիվ իմպլեմենտացիա`
// հիշողության և արագագործության տեսանկյունից:
public class FWAlgorithm
{
    // զրաֆի հարևանության մատրիցը
    private readonly AdjacencyMatrix initialAdjacencyMatrix;

    // Քայլերի հաջորդականություն
    private readonly List<Step> steps;

    // Քայլերի քանակ
    private readonly int count;

    // Այժմյան քայլ
    private int current;

    // Առաջին քայլ
    public Step FirstStep => this.steps.FirstOrDefault();

    // Վերջին քայլ
    public Step LastStep => this.steps.LastOrDefault();

    // Սկզբնարժեքավորում է ալգորիթմը`
    // ստանալով զրաֆի հարևանության մատրիցը
    public FWAlgorithm(AdjacencyMatrix initialAdjacencyMatrix)
    {
        this.initialAdjacencyMatrix = initialAdjacencyMatrix;
        this.count = initialAdjacencyMatrix.Size;
        this.current = -1;

        this.steps = new List<Step>(initialAdjacencyMatrix.Size);
    }

    // Կատարում է հերթական քայլը
    // Կվերադձնի սխալ բուլյան արժեք, երբ ալգորիթմի աշխատանքն ավարտվի:
    public bool Next()
    {
        if (this.current == -1)
        {
            this.DoFirstStep();
            return true;
        }

        if (this.current == this.count)
            return false;

        var step = this.CreateStep();
    }

```

```

var lastA = this.LastStep.A;
var lastB = this.LastStep.B;
var sum = default(double);
var aValue = default(double);
var bValue = default(int);
var color = default(Color);

for (var i = 0; i < this.count; i++)
{
    for (var j = 0; j < this.count; j++)
    {
        if (j == this.current)
        {
            step.B[i, j] = lastB[i, j];
            // բաց ենք թողում ֆիքսված տողը և սյունը
            if (i == this.current)
            {
                step.A[i, j] = lastA[i, j]; continue;
            }
        }
        // հաշվում ենք հապատասխան հանգույցների արժեքների գումարը
        // և համեմատում ենք իրական հանգույցի արժեքի հետ
        sum = lastA[i, this.current].Value + lastA[this.current, j].Value;

        if (lastA[i, j].Value > sum)
        {
            aValue = sum;
            bValue = lastB[i, this.current].Value;
            color = Color.Red;
        }
        else
        {
            aValue = lastA[i, j].Value;
            bValue = lastB[i, j].Value;
            color = Color.White;
        }
        // սկզբնարժեքավորում ենք հարևանության մատրիցի և
        // օժանդակ մատրիցի
        // (i,j) բջիջը ալգորիթմի հերթական քայլում
        step.A[i, j] = new Cell<double>(color, lastA[i, j].Point, aValue);
        step.B[i, j] = new Cell<int>(color, lastB[i, j].Point, bValue);
    }
}
// 1-ով ավելացնում ենք ալգորիթմի քայլի հաշվիչը
this.current++;
this.steps.Add(step);

```

```

        return true;
    }

    // Կատարում է առաջին զրոյական քայլը`
    // այսինքն սկսնաթեքավորում է հարևանության մատրիցը
    // և կառուցում է օժանդակ B մատրիցը
    // օրինակ`
    //      |V| = 3 ` |1 2 3|
    //                  |1 2 3|
    //                  |1 2 3|
    private void DoFirstStep()
    {
        var firstStep = this.CreateStep();

        for (var i = 0; i < this.count; i++)
        {
            for (var j = 0; j < this.count; j++)
            {
                firstStep.A[i, j] = this.initialAdjacencyMatrix[i, j];
                firstStep.B[i, j] = new Cell<int>(
                    Color.White, new Point(i, j), j + 1);
            }
        }

        this.current++;
        this.steps.Add(firstStep);
    }

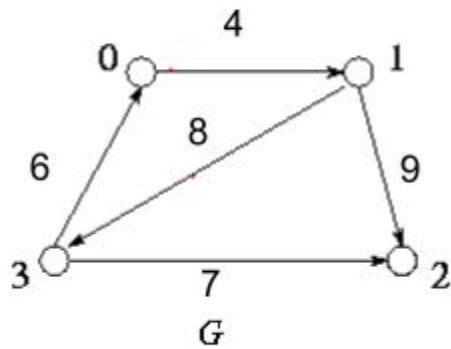
    // Ստեղծում և սկսնաթեքավորում է ալգորիթմի հերթական քայլը
    private Step CreateStep()
    {
        return new Step
        {
            Number = 0,
            A = new SquareMatrix<Cell<double>>(this.count),
            B = new SquareMatrix<Cell<int>>(this.count)
        };
    }
}

```

Ալգորիթմի ամբողջական իմպլեմենտացիան օրինակներով հանդերձ հասանելի է իմ GitHub-յան էջում հետևյալ հղումով`

<https://github.com/amirkhaneansev/fw-algorithm>

Վերցնենք հետևյալ գրաֆը և աշխատեցնենք ալգորիթը դրա վրա



A<sub>0</sub>

0	4	-	-
-	0	9	8
-	-	0	-
6	-	7	0

B<sub>0</sub>

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4

A<sub>1</sub>

0	4	-	-
-	0	9	8
-	-	0	-
6	10	7	0

B<sub>1</sub>

1	2	3	4
1	2	3	4
1	2	3	4
1	1	3	4



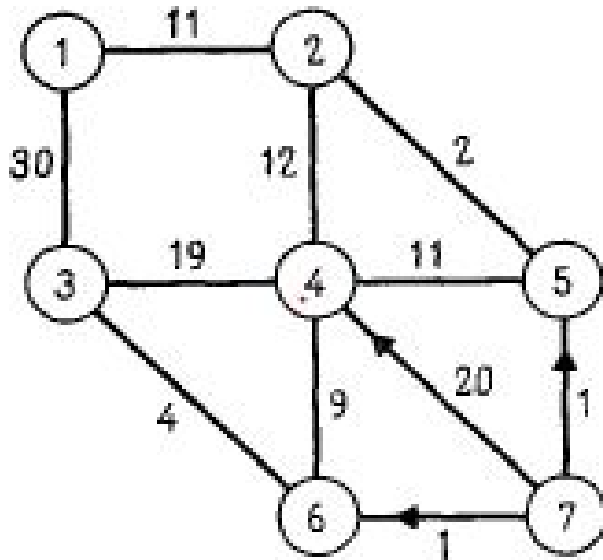
$A_2$					$B_2$			
0	4	13	12		1	2	2	2
-	0	9	8		1	2	3	4
-	-	0	-		1	2	3	4
6	10	7	0		1	1	3	4

$A_3$					$B_3$			
0	4	13	12		1	2	2	2
-	0	9	8		1	2	3	4
-	-	0	-		1	2	3	4
6	10	7	0		1	1	3	4

$A_4$					$B_4$			
0	4	13	12		1	2	2	2
14	0	9	8		4	2	3	4
-	-	0	-		1	2	3	4
6	10	7	0		1	1	3	4

$A_4$  մատրիցի  $(i, j)$  վանդակում գրված է  $i$  գագաթից  $j$  գագաթ ճանապարհի երկարությունը:  $B_4$  մատրիցում գրված է  $(i, j)$  ճանապարհը:  
 Ակնայտ է որ ալգորիթմի բարդությունը կլինի  $O(|V|^3)$ , որտեղ  $|V|$  -ն  $G$  գրաֆի գագաթների բազմության հզորությունն է:

Ստորև դիտարկվում է **T.C. Hu-ի Integer Programming and Network Flows** գրքի օրինակը



Օ-ական քայլ

A <sub>0</sub>							B <sub>0</sub>						
0	11	30	-	-	-	-	1	2	3	4	5	6	7
11	0	-	12	2	-	-	1	2	3	4	5	6	7
30	-	0	19	-	4	-	1	2	3	4	5	6	7
-	12	19	0	11	9	-	1	2	3	4	5	6	7
-	2	-	11	0	-	-	1	2	3	4	5	6	7
-	-	4	9	-	0	-	1	2	3	4	5	6	7
-	-	-	20	1	1	0	1	2	3	4	5	6	7

1-ին քայլ

A <sub>1</sub>							B <sub>1</sub>						
0	11	30	-	-	-	-	1	2	3	4	5	6	7
11	0	41	12	2	-	-	1	2	1	4	5	6	7
30	41	0	19	-	4	-	1	1	3	4	5	6	7
-	12	19	0	11	9	-	1	2	3	4	5	6	7
-	2	-	11	0	-	-	1	2	3	4	5	6	7
-	-	4	9	-	0	-	1	2	3	4	5	6	7
-	-	-	20	1	1	0	1	2	3	4	5	6	7

2-րդ քայլ

A <sub>2</sub>							B <sub>2</sub>						
0	11	30	23	13	-	-	1	2	3	2	2	6	7
11	0	41	12	2	-	-	1	2	1	4	5	6	7
30	41	0	19	43	4	-	1	1	3	4	1	6	7
23	12	19	0	11	9	-	2	2	3	4	5	6	7
13	2	43	11	0	-	-	2	2	2	4	5	6	7
-	-	4	9	-	0	-	1	2	3	4	5	6	7
-	-	-	20	1	1	0	1	2	3	4	5	6	7

### 3-րդ քայլ

$A_3$							$B_3$						
0	11	30	23	13	34	-	1	2	3	2	2	3	7
11	0	41	12	2	45	-	1	2	1	4	5	1	7
30	41	0	19	43	4	-	1	1	3	4	1	6	7
23	12	19	0	11	9	-	2	2	3	4	5	6	7
13	2	43	11	0	47	-	2	2	2	4	5	2	7
34	45	4	9	47	0	-	3	3	3	4	3	6	7
-	-	-	20	1	1	0	1	2	3	4	5	6	7

### 4-րդ քայլ

$A_4$							$B_4$						
0	11	30	23	13	32	-	1	2	3	2	2	2	7
11	0	31	12	2	21	-	1	2	4	4	5	4	7
30	31	0	19	30	4	-	1	4	3	4	4	6	7
23	12	19	0	11	9	-	2	2	3	4	5	6	7
13	2	30	11	0	20	-	2	2	4	4	5	4	7
32	21	4	9	20	0	-	4	4	3	4	4	6	7
43	32	39	20	1	1	0	4	4	4	4	5	6	7

5-րդ քայլ

$A_5$							$B_5$						
0	11	30	23	13	32	-	1	2	3	2	2	2	7
11	0	31	12	2	21	-	1	2	4	4	5	4	7
30	31	0	19	30	4	-	1	4	3	4	4	6	7
23	12	19	0	11	9	-	2	2	3	4	5	6	7
13	2	30	11	0	20	-	2	2	4	4	5	4	7
32	21	4	9	20	0	-	4	4	3	4	4	6	7
14	3	39	12	1	1	0	5	5	5	5	5	6	7

6-րդ քայլ

$A_6$							$B_6$						
0	11	30	23	13	32	-	1	2	3	2	2	2	7
11	0	25	12	2	21	-	1	2	4	4	5	4	7
30	25	0	13	24	4	-	1	6	3	6	6	6	7
23	12	13	0	11	9	-	2	2	6	4	5	6	7
13	2	24	11	0	20	-	2	2	4	4	5	4	7
32	21	4	9	20	0	-	4	4	3	4	4	6	7
14	3	5	10	1	1	0	5	5	6	6	5	6	7

7-րդ քայլ

A <sub>7</sub>							B <sub>7</sub>						
0	11	30	23	13	32	-	1	2	3	2	2	2	7
11	0	25	12	2	21	-	1	2	4	4	5	4	7
30	25	0	13	24	4	-	1	6	3	6	6	6	7
23	12	13	0	11	9	-	2	2	6	4	5	6	7
13	2	24	11	0	20	-	2	2	4	4	5	4	7
32	21	4	9	20	0	-	4	4	3	4	4	6	7
14	3	5	10	1	1	0	5	5	6	6	5	6	7