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 մոդելավորում 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);
        }
    }

    // Ստրուկտուրաներ, 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 մատրցի
    public SquareMatrix<Cell<double>> A { get; set; }

    // Օժանդակ մատրից 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:
// ՈՒՇԱԴՐՈՒԹՅՈՒՆ՝ Ֆլոյդ-Ուորշըլի ալգորիթմի այս իմպլեմենտացումն ունի
// ուսուցողական, վերլուծական և դեմոնստրատիվ բնույթ, ինչի
// արդյունքում հնարավոր է, որ այն չհապատասխանի արտադրական ստանդարտներին:
```

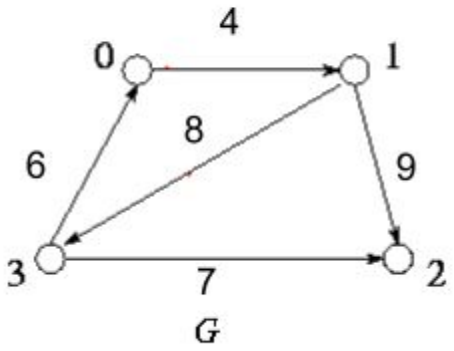
```
// Միանշանակ հնարավոր է ավելի էֆեկտիվ իմպլեմենտացիա՝
// հիշողության և արագագործության տեսանկյունից:
public class FWAlgorithm
{
    // Գրաֆի հարևանևության մատրիցը
    private readonly AdjacencyMatrix initialAdjacencyMatrix;

    // 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 գրաֆի 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 մատրցիցի
        // (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 սկբնարժեքավորում 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 սկբնարժեքավորում 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/amirkhaniansev/fw-algorithm

Վերցնենք հետևյալ գրաֆը և աշխատեցնենք ալգորիթմը դրա վրա



A_0

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

B_0

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

A_1

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

B_1

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

A_2

0	4	13	12
-	0	9	8
-	-	0	-
6	10	7	0

B_2

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

A_3

0	4	13	12
-	0	9	8
-	-	0	-
6	10	7	0

B_3

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

A_4

0	4	13	12
14	0	9	8
-	-	0	-
6	10	7	0

B_4

1	2	2	2
4	2	3	4
1	2	3	4
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 բազության հզորությունն է: