

Projekt: **Floyd - Warshallov algoritam**

Datum: 02.04.2005

Ime i prezime: **Ivica Huk**

Smjer: Računarstvo

## Sadržaj

1. Uvod.....	1
1.1. Teorija grafova.....	1
1.2. Grafovi kao strukture podataka.....	2
2. Floyd–Warshall algoritam.....	2
2.1. Ideja.....	2
2.2. Opis Floyd-Warshall algoritma.....	2
2.3. Primjer .....	3
3. Implementacija.....	5
3.1. Reprezentacija grafa.....	5
3.2. Floyd–Warshall algoritam.....	6
3.3. Ispis najkraćeg puta.....	6
3.4. Složenost.....	7

## 1. Uvod

Ovaj dokument opisuje impementaciju Floyd–Warshall algoritma za traženje najkraćeg puta između vrhova grafa. Floyd–Warshallov algoritam računa najkraći put za sve parove vrhova.

### 1.1. Teorija grafova

U matematici graf je uređeni par  $G=(V, E)$  gdje je

$V$  konačan skup vrhova,  $|V|=v$

$E$  konačan skup bridova,  $|E|=e$

Za neusmjerene grafove skup  $E$  je skup dvočlanih skupova sa elementima is  $V$ , dok je za usmjerene grafove skup  $E$  skup uređenih parova, odnosno  $E \subseteq V \times V$

Primjer:

$$V = \{1, 2, 3, 4\}$$

$$E = \{\{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 4\}\} \text{ za neusmjereni graf, ili}$$

$$E = \{(1, 2), (1, 3), (2, 4), (3, 4)\} \text{ za usmjereni graf}$$

Ako je bridovima grafa  $G$  dodjeljena težina, odnosno ako postoji funkcija  $w : E \rightarrow \mathbb{R}$

kažemo da je graf  $G$  težinski. Netežinski grafovi mogu se smatrati težinskim pri čemu svi bridovi imaju istu težinu, npr. 1.

### 1.2. Grafovi kao strukture podataka

Postoji nekoliko načina reprezentacije grafova. Odabir strukture podataka za reprezentaciju grafova ovisi o strukturi samog grafa i algoritmima koji se koriste za manipulaciju grafom. Općenito možemo razlikovati reprezentaciju grafa uz pomoć matrica i reprezentaciju grafa uz pomoć listi. Prednosti matrice reprezentacije grafova su jednostavnost i konstantno vrijeme potrebno za ispitivanje dali brid  $e(i, j)$  postoji u grafu. Nedostatak matrice reprezentacije je što zauzima  $O(n^2)$  prostora bez obzira na broj bridova što kod grafova koji imaju relativno velik broj vrhova u odnosu na broj brojeva može biti neefikasno. S druge strane reprezentacije uz pomoć listi zauzima manje mjesta ali je implementacija složenija i ispitivanje postoji li određeni brid ima složenost  $O(d_i)$  gdje je  $d_i$  stupanj brida  $i$ .

## 2. Floyd–Warshall algoritam

Floydov-Warshallov algoritam računa najkraći put između svih parova vrhova u grafu. Floyd-Warshallov algoritam radi na matricnoj reprezentaciji grafa, točnije na matrici susjednosti.

### 2.1. Ideja

Neka su vrhovi usmjerenog grafa  $G$ ,  $V = \{1, 2, 3, \dots, n\}$  i neka je  $\{1, 2, \dots, k\}$  podskup od  $V$  za neki  $k$

Za bilo koji par vrhova  $i, j \in V$  razmatramo sve puteve od  $i$  do  $j$  koje možemo povući koristeći vrhove iz  $\{1, 2, \dots, k\}$ , neka je  $p$  najkraći put među njima.

Tada vrijede sljedeće tvrdnje:

Ako  $k$  nije u  $p$ , tada je najkraći put između  $i$  i  $j$  koji prolazi kroz vrhove iz skupa  $\{1, 2, \dots, k-1\}$  ujedno i najkraći put koji prolazi kroz vrhove iz skupa  $\{1, 2, \dots, k\}$

Ako je  $k$  u  $p$  tada  $p$  možmo podijeliti na  $p1$  i  $p2$  gdje je  $p1$  najkraći put od  $i$  do  $k$  koji prolazi vrhovima iz skupa  $\{1, 2, \dots, k-1\}$  i  $p2$  najkraći put od  $k$  do  $j$  koji prolazi vrhovima iz skupa  $\{1, 2, \dots, k-1\}$

Floyd-Warshall algoritam koristi vezu između puta  $p$  i najkraćeg puta od  $i$  do  $j$  koji prolazi kroz vrhove  $\{1, 2, \dots, k-1\}$ . Veza ovisi o tome da li  $p$  prolazi kroz  $k$  ili ne.

### 2.2. Opis Floyd-Warshall algoritma

Neka je  $A_k$  matrica tipa  $n \times n$  gdje je  $A_k[i, j]$  težina najkraćeg puta od  $i$  do  $j$  koja prolazi kroz vrhove  $\leq k$ . Tada definiramo

$$A_0[i, j] = \begin{cases} \text{težina brida od } i \text{ do } j & \text{za } i \neq j \wedge (i, j) \in E \\ \infty & \text{za } i \neq j \wedge (i, j) \notin E \\ 0 & \text{za } i = j \end{cases}$$

Razmatramo najkraći put  $p$  od  $i$  do  $j$  koji prolazi vrhovima  $1 \dots k$ . Tada za put  $p$  vrijedi jedna od tvrdnji:

- put  $p$  ne prolazi kroz  $k$  i u tom slučaju težina puta jednaka je  $A_{k-1}[i, j]$ .

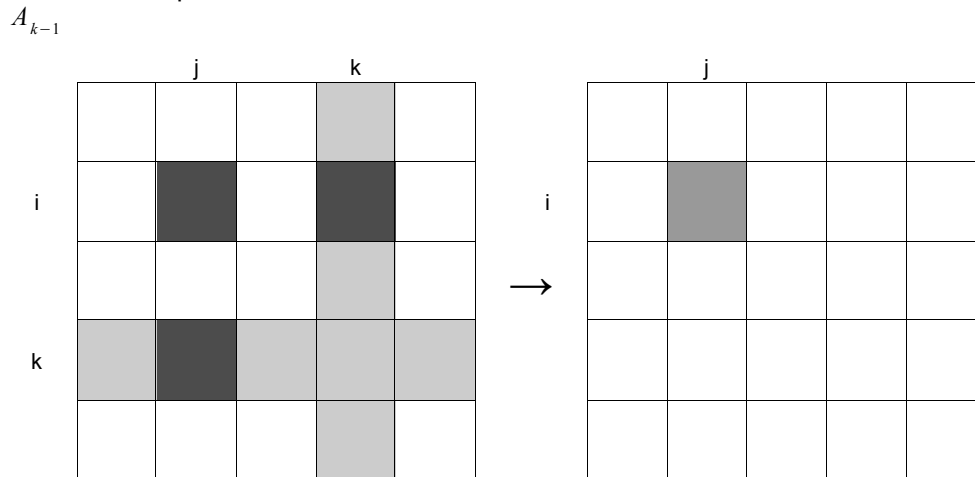
- put  $p$  prolazi kroz  $k$  i u tom slučaju težina puta jednaka je

$$A_{k-1}[i, k] + A_{k-1}[k, j] .$$

Dakle slijedi

$$A_k[i, j] = \min(A_{k-1}[i, j], A_{k-1}[i, k] + A_{k-1}[k, j])$$

Grafički to možemo prikazati kao



Važno je primjetiti da svaki element u  $A_k$  ovisi o vrijednostima u retku  $k$  i stupcu  $k$  u  $A_{k-1}$ . Drugo važno svojstvo je da su vrijednosti elemenata u retku  $k$  i stupcu  $k$  u  $A_k$  jednake jednake onima u  $A_{k-1}$ . Što se može lako pokazati

Za redak  $k$  imamo

$$\begin{aligned} A_k[k, j] &= \min\{A_{k-1}[k, j], A_{k-1}[k, k] + A_{k-1}[k, j]\} \\ &= \min\{A_{k-1}[k, j], 0 + A_{k-1}[k, j]\} \\ &= A_{k-1}[k, j] \end{aligned}$$

Za stupac  $k$  imamo

$$\begin{aligned} A_k[i, k] &= \min\{A_{k-1}[i, k], A_{k-1}[i, k] + A_{k-1}[k, k]\} \\ &= \min\{A_{k-1}[i, k], A_{k-1}[i, k] + 0\} \\ &= A_{k-1}[i, k] \end{aligned}$$

Iz gore navedenog svojstva proizlazi da nam je dovoljna samo jedna matrica odnosno niz.

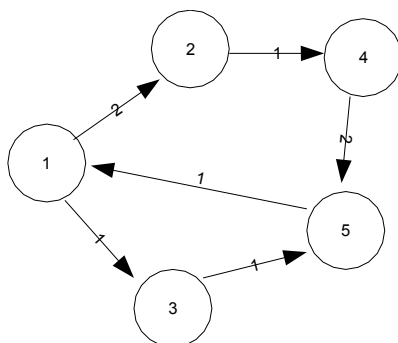
### 2.3. Primjer

$$G = \{V, E\}$$

Neka je  $V = \{1, 2, 3, 4, 5\}$

usmjereni, težinski graf kao na slici

$$E = \{(1, 2), (1, 3), (2, 4), (3, 5), (5, 1)\}$$



Tada bi traženje najkraćeg puta između svih parova vrhova izgledalo kao na slici.

	1	2	3	4	5
1	0	2	1	$\infty$	$\infty$
2	$\infty$	0	$\infty$	1	$\infty$
3	$\infty$	$\infty$	0	$\infty$	1
4	$\infty$	$\infty$	$\infty$	0	2
5	1	$\infty$	$\infty$	$\infty$	0

	1	2	3	4	5
1	0	2	1	$\infty$	$\infty$
2	$\infty$	0	$\infty$	1	$\infty$
3	$\infty$	$\infty$	0	$\infty$	1
4	$\infty$	$\infty$	$\infty$	0	2
5	1	3	2	$\infty$	0

	1	2	3	4	5
1	0	2	1	3	$\infty$
2	$\infty$	0	$\infty$	1	$\infty$
3	$\infty$	$\infty$	0	$\infty$	1
4	$\infty$	$\infty$	$\infty$	0	2
5	1	3	2	4	0

	1	2	3	4	5
1	0	2	1	3	2
2	$\infty$	0	$\infty$	1	$\infty$
3	$\infty$	$\infty$	0	$\infty$	1
4	$\infty$	$\infty$	$\infty$	0	2
5	1	3	2	4	0

	1	2	3	4	5
1	0	2	1	3	2
2	$\infty$	0	$\infty$	1	3
3	$\infty$	$\infty$	0	$\infty$	1
4	$\infty$	$\infty$	$\infty$	0	2
5	1	3	2	4	0

	1	2	3	4	5
1	0	2	1	3	2
2	4	0	5	1	3
3	2	4	0	5	1
4	3	5	4	0	2
5	1	3	2	4	0

## 3. Implementacija

### 3.1. Reprerentacija grafa

Floyd-Warshallov algoritam radi na matricnoj reprezentaciji grafa, točnije na matrici susjednosti. Također potrebna nam je informacija o težini svakog brida. Ako su dva vrha spojena sa više bridova spremamo informaciju o bridu najmanje težine.

```
struct graph_s
{
    int nvertices; /* broj vrhova u grafu */
    int **weights; /* najmanja težina brida između dva vrha */
    int **vertices;
};
```

#### 3.1.1. Inicijalizacija grafa

Funkcija za inicijalizaciju grafamogla bi izgledati ovako

```
int
graph_initialize(graph_t *graph, int size)
{
    int i;

    assert(graph);
    if (NULL == graph)
        return (GRAPH_ERROR_INVALID_PARAM);

    if ((NULL == (graph->vertices = malloc(sizeof(int *) * size))) ||
        (NULL == (graph->weights = malloc(sizeof(int *) * size))))
        return (GRAPH_ERROR_ALLOCATION_ERROR);

    for (i = 0; i < size; i++)
    {
        int j;

        if (NULL == (graph->vertices[i] = malloc(sizeof(int) * size)))
            return (GRAPH_ERROR_ALLOCATION_ERROR);

        if (NULL == (graph->weights[i] = malloc(sizeof(int) * size)))
            return (GRAPH_ERROR_ALLOCATION_ERROR);

        for (j = 0; j < size; j++)
        {
            graph->vertices[i][j] = 0;
            graph->weights[i][j] = INF;
        }
    }

    graph->nvertices = size;

    return (0);
}
```

#### 3.1.2. Dodavanje bridova

Dodavanje bridova kod reprezentacije grafa matricom susjednosti je vrlo jednostavno. Funkcija za dodavanje brodova mogla bi se implementirati na sljedeći način

int

```
graph_add_edge(graph_t * graph, int start, int end, int weight)
{
    assert(graph);
    if (NULL == graph)
        return (GRAPH_ERROR_INVALID_PARAM);
}
```

```

    assert(start < graph->nvertices && end < graph->nvertices);
    if (start >= graph->nvertices || end >= graph->nvertices)
        return (GRAPH_ERROR_INVALID_PARAM);

    graph->vertices[start][end] += 1;
    if (weight < graph->weights[start][end])
        graph->weights[start][end] = weight;

    return (0);
}

```

### 3.2. Floyd–Warshall algoritam

Uz gore pokazano svojstvo da su vrijednosti elemenata u retku  $k$  i stupcu  $k$  u  $A_k$  jednake jednake onima u  $A_{k-1}$  Floyd-Warshall algoritam možemo vrlo jednostavno implementirati.

#### 3.2.1. Inicijalizacija $A_0$

Matricu  $A_0$  moramo inicijalizirati tako da je  $A_0[i, j]$  jednak težini brida, ili nekom ekvivalentu beskonačnosti ako takav brid ne postoji. Kod za inicijalizaciju matrice  $A_0$  mogao bi izgledati ovako

```

for (i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
        weights[i][j] = (i == j) ? 0 : g->weights[i][j];
}

```

#### 3.2.2. Računanje $A_k$

```

for (k = 0; k < n; k++)
{
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (weights[i][k] + weights[k][j] < weights[i][j])
                weights[i][j] = weights[i][k] + weights[k][j];
        }
    }
}

```

### 3.3. Ispis najkraćeg puta

Kao i Dijkstrin algoritam, Floyd–Warshallov algoritam ne ispisuje najkraći put već samo računa dužinu puta za sve parove vrhova. Da bismo mogli ispisati najkraći put uvodimo još jednu matricu u koju upisujemo:

```

for (k = 0; k < n; k++)
{
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (weights[i][k] + weights[k][j] < weights[i][j])
            {
                weights[i][j] = weights[i][k] + weights[k][j];
            }
        }
    }
}

```

```
        paths[i][j] = k;
    }
}
}
```

Nakon izvršavanja  $paths[i, j]$  sadrži vrh na najkraćem putu od  $i$  do  $j$ . Najkraći put tada možemo ispisati

```
void print_path(int **paths, int start, int end)
{
    int k = paths[start][end];

    if (k > 0)
    {
        print_path(paths, start, k);
        printf("%d ", k);
        print_path(paths, k, end);
    }
}
```

```
for (i = 0; i < graph.nvertices; i++)
{
    for (j = 0; j < graph.nvertices; j++)
    {
        if (weights[i][j] == INF || i == j)
            continue;

        printf("%d->%d: ", i, j);
        printf("%d ", i);
        print_path(paths, i, j);
        printf("%d\n", j);
    }
}
```

### 3.4. Složenost

Floyd-Warshallov algoritam ima složenost  $O(n^3)$ . Striktno gledano takva složenost nije bolja od složenosti Dijkstrainog algoritma izvršenog  $n$  puta ali su unutarnje petlje vrlo brze tako da je u praksi ovaj algoritam puno brži.