Caminhos mínimos

Algoritmo de Floyd-Warshall

Prof. Edson Alves

2019

Faculdade UnB Gama

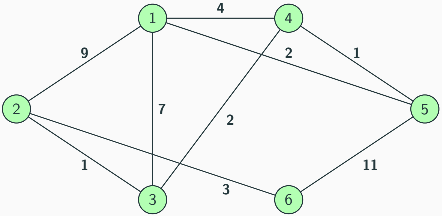
Sumário

Algoritmo de Floyd-Warshall

Algoritmo de Floyd-Warshall

- Assim como os algoritmos de Bellman-Ford e de Dijkstra, o algoritmo de Floyd-Warshall também resolve o problema do caminho mínimo
- Ao contrário dos dois citados, ele computa as distâncias mínimos entre todos os pares de vértices conectados em uma só execução
- O vetor bidimensional que mantém as distância entre todos os pares é inicializado com os pesos das arestas entre os nós, e infinito quando não houver uma aresta entre os dois vértices
- Naturalmente, a distância de um nó a si mesmo é zero
- A cada iteração, o algoritmo tenta melhorar as distâncias por meio do uso de um vértice intermediário
- A complexidade é $O(V^3)$, de modo que é mais eficiente do que V execuções do algoritmo de Dijkstra

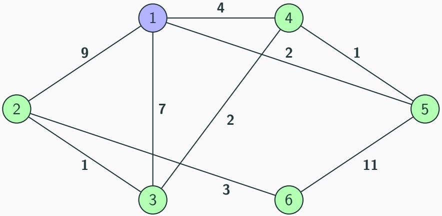
Visualização do algoritmo de Floyd-Warshall



Distâncias

	1	2	3	4	5	6
1	0	9	7	4	2	∞
2	9	0	1	∞	∞	3
3	7	1	0	2	∞	∞
4	4	∞	2	0	1	∞
5	2	∞	∞	1	0	11
6	∞	3	∞	∞	11	0

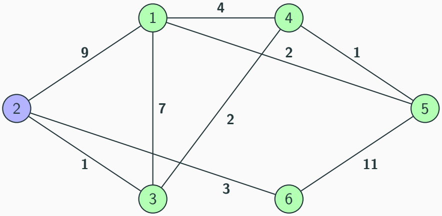
Visualização do algoritmo de Floyd-Warshall



Distâncias

	1	2	3	4	5	6
1	0	9	7	4	2	∞
2	9	0	1	**13**	**11**	3
3	7	1	0	2	**9**	∞
4	4	**13**	2	0	1	∞
5	2	**11**	**9**	1	0	11
6	∞	3	∞	∞	11	0

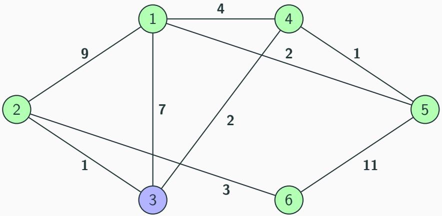
Visualização do algoritmo de Floyd-Warshall



Distâncias

	1	2	3	4	5	6
1	0	9	7	4	2	**12**
2	9	0	1	13	11	3
3	7	1	0	2	9	**4**
4	4	13	2	0	1	**16**
5	2	11	9	1	0	11
6	**12**	3	**4**	**16**	11	0

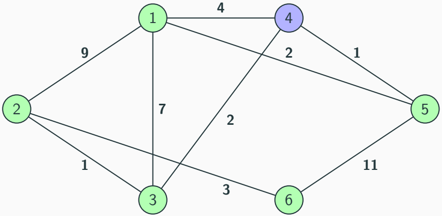
Visualização do algoritmo de Floyd-Warshall



Distâncias

	1	2	3	4	5	6
1	0	**8**	7	4	2	**11**
2	**8**	0	1	**3**	**10**	3
3	7	1	0	2	9	4
4	4	**3**	2	0	1	**6**
5	2	**10**	9	1	0	11
6	**11**	3	4	**6**	11	0

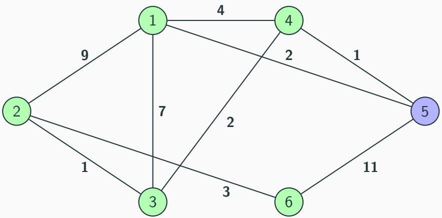
Visualização do algoritmo de Floyd-Warshall



Distâncias

	1	2	3	4	5	6
1	0	**7**	**6**	4	2	**10**
2	**7**	0	1	3	**4**	3
3	**6**	1	0	2	**3**	4
4	4	3	2	0	1	6
5	2	**4**	**3**	1	0	**7**
6	**10**	3	4	6	**7**	0

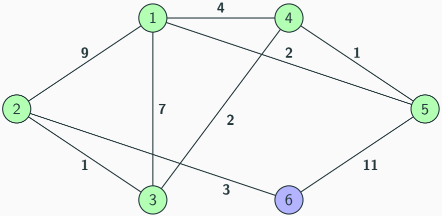
Visualização do algoritmo de Floyd-Warshall



Distâncias

	1	2	3	4	5	6
1	0	**6**	**5**	**3**	2	**9**
2	**6**	0	1	3	4	3
3	**5**	1	0	2	3	4
4	**3**	3	2	0	1	6
5	2	4	3	1	0	7
6	**9**	3	4	6	7	0

Visualização do algoritmo de Floyd-Warshall



Distâncias

	1	2	3	4	5	6
1	0	6	5	3	2	9
2	6	0	1	3	4	3
3	5	1	0	2	3	4
4	3	3	2	0	1	6
5	2	4	3	1	0	7
6	9	3	4	6	7	0

Implementação do algoritmo de Floyd-Warshall em C++

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;
using edge = tuple<int, int, int>;

const int MAX { 210 }, oo { 1000000010 };
int dist[MAX][MAX];
vector<ii> adj[MAX];

void floyd_warshall(int N) {
    for (int u = 1; u <= N; ++u)
        for (int v = 1; v <= N; ++v)
            dist[u][v] = oo;

    for (int u = 1; u <= N; ++u)
        dist[u][u] = 0;

    for (int u = 1; u <= N; ++u)
        for (const auto& [v, w] : adj[u])
            dist[u][v] = w;
```

Implementação do algoritmo de Floyd-Warshall em C++

```cpp

    for (int k = 1; k <= N; ++k)
        for (int u = 1; u <= N; ++u)
            for (int v = 1; v <= N; ++v)
                dist[u][v] = min(dist[u][v], dist[u][k] + dist[k][v]);
}

int main() {
    vector<edge> edges { edge(1, 2, 9), edge(1, 3, 7), edge(1, 4, 4),
        edge(1, 5, 2), edge(2, 3, 1), edge(2, 6, 3), edge(3, 4, 2),
        edge(4, 5, 1), edge(5, 6, 11) };

    for (const auto& [u, v, w] : edges) {
        adj[u].push_back(ii(v, w));
        adj[v].push_back(ii(u, w));
    }

    floyd_warshall(6);

    return 0;
}
```

Caminhos mínimos

Identificação do caminho mínimo

- Assim como nos algoritmos de Bellman-Ford e Dijkstra, é possível recuperar a sequência de arestas que compõem o caminho mínimo
- Para determinar o caminho, é preciso manter o vetor bidimensional `pred`, onde `pred[u][v]` é o nó que antecede v no caminho mínimo que vai de u a v
- Inicialmente, todos os elementos deste vetor devem ser iguais a um valor sentinela, exceto em dois casos:
    1. `pred[u][u] = u`
    2. `pred[u][v] = u`, se $(u, v) \in E$
- Se o vértice k reduzir a distância `dist[u][v]` (isto é, se `dist[u][k] + dist[k][v] < dist[u][v]`), então o `pred[u][v] = pred[k][v]`
- Deste modo, o caminho mínimo de u a v pode ser recuperado, passando por todos os predecessores até se atingir o nó u

Recuperação do caminho mínimo

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;
using edge = tuple<int, int, int>;

const int MAX { 210 }, oo { 1000000010 };
int dist[MAX][MAX], pred[MAX][MAX];
vector<ii> adj[MAX];

void floyd_warshall(int N)
{
    for (int u = 1; u <= N; ++u)
        for (int v = 1; v <= N; ++v)
            dist[u][v] = oo;

    for (int u = 1; u <= N; ++u)
    {
        dist[u][u] = 0;
        pred[u][u] = u;
    }
```

Recuperação do caminho mínimo

```cpp
    for (int u = 1; u <= N; ++u)
        for (const auto& [v, w] : adj[u]) {
            dist[u][v] = w;
            pred[u][v] = u;
        }

    for (int k = 1; k <= N; ++k)
    {
        for (int u = 1; u <= N; ++u)
        {
            for (int v = 1; v <= N; ++v)
            {
                if (dist[u][v] > dist[u][k] + dist[k][v])
                {
                    dist[u][v] = dist[u][k] + dist[k][v];
                    pred[u][v] = pred[k][v];
                }
            }
        }
    }
}
```

Recuperação do caminho mínimo

```
int main() {
    vector<edge> edges { edge(1, 2, 9), edge(1, 3, 7), edge(1, 4, 4),
        edge(1, 5, 2), edge(2, 3, 1), edge(2, 6, 3), edge(3, 4, 2),
        edge(4, 5, 1), edge(5, 6, 11) };

    int N = 6;

    for (const auto& [u, v, w] : edges) {
        adj[u].push_back(ii(v, w));
        adj[v].push_back(ii(u, w));
    }

    floyd_warshall(N);

    for (int u = 1; u <= N; ++u)
    {
        for (int v = 1; v <= N; ++v)
        {
            vector<int> path;
            auto p = v;
```

Recuperação do caminho mínimo

```
            while (p != u) {
                path.push_back(p);
                p = pred[u][p];
            }

            path.push_back(u);
            reverse(path.begin(), path.end());

            cout << "dist[" << u << "][" << v << "] = " << dist[u][v]
                << '\n';

            for (size_t i = 0; i < path.size(); ++i)
                cout << path[i] << (i + 1 == path.size() ? "\n" : " -> ");
        }
    }

    return 0;
}
```

Referências

1. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
2. **LAAKSONEN**, Antti. *Competitive Programmer's Handbook*, 2018.
3. **SKIENA**, Steven S.; **REVILLA**, Miguel A. *Programming Challenges*, 2003.