Treinamento para Competições de Programação

Grafos - Parte II
All-Pairs Shortest Paths:

O Algoritmo de Floyd-Warshall

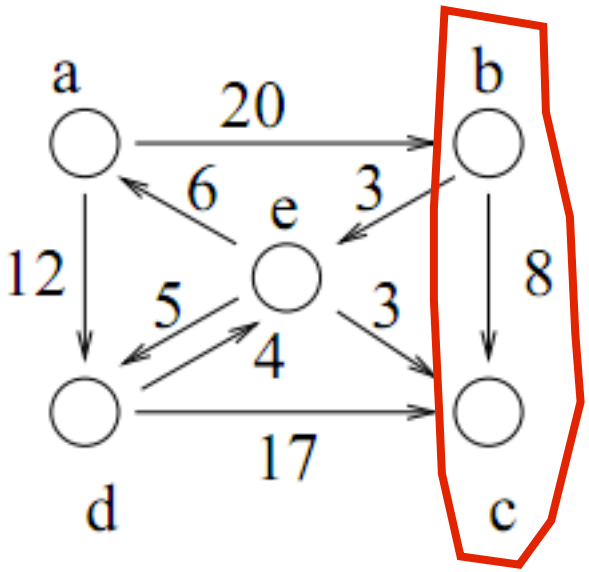
Murilo Adriano Vasconcelos
http://murilo.wordpress.com

O Problema

- Dada a descrição de um grafo ***G(V, E)*** (implícita/explicitamente) sem ciclos negativos, queremos saber quais são as menores distâncias entre cada par de vértices ***vi, vj*** $\in$ ***V***.

Representação do Grafo

- Matriz de adjacência
- Peso da aresta de i para i é 0
- Arestas que não estão no grafo são representadas por ∞ (um valor muito grande)



-	a	b	c	d	e
a	0	20	∞	12	∞
b	∞	0	8	∞	3
c	∞	∞	0	∞	∞
d	∞	∞	17	0	4
e	6	∞	3	5	0

O Algoritmo

```
const int MAXV = 100; // número máximo de vértices
const int INF = 0x3f3f3f3f; // cuidado com esse valor
int grafo[MAXV][MAXV];

for (int i = 0; i < N; ++i) {
    for (int j = i; j < N; ++j)
        grafo[i][j] = grafo[j][i] = INF;

    grafo[i][i] = 0; // distância de i->i = 0
}
...
for (int k = 0; k < N; ++k) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            grafo[i][j] = min(grafo[i][j],
              grafo[i][k] + grafo[k][j]); // ir i->k->j
        }
    }
}
```

O Algoritmo - explicação

A cada passo, o algoritmo tenta diminuir a distância entre os vértices `i` e `j` passando pelo vértice intermediário `k`.

O "`k`" representa o vértice intermediário atual. Assim, quando `k = 3` é finalizado, todos os `grafo[i][j]` estarão os menores caminhos "podendo" passar pelos vértices 0, 1, 2 e 3. Ao final de `k == N-1` temos os menores caminhos podendo passar por todos os vértices. Esse processo é chamado de relaxação.

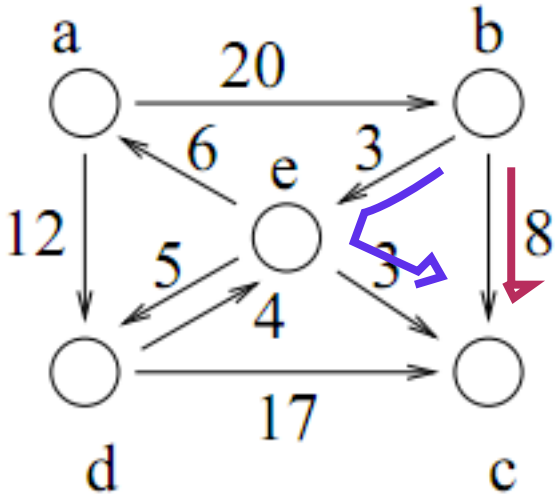
```
// k indica o vértice intermediário que estamos processando
for (int k = 0; k < N; ++k) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            grafo[i][j] = min(grafo[i][j],
              grafo[i][k] + grafo[k][j]); // relaxando
        }
    }
}
```

Menor distância vs. distância direta

- Qual a menor distância entre b e c?
  - Distância direta = 8
  - Distância mínima = 6

8 6

```
g[b][c] = min(g[b][c], g[b][e] + g[e][c]);
```



-	a	b	c	d	e
a	0	20	∞	12	∞
b	∞	0	8	∞	3
c	∞	∞	0	∞	∞
d	∞	∞	17	0	4
e	6	∞	3	5	0

Considerações

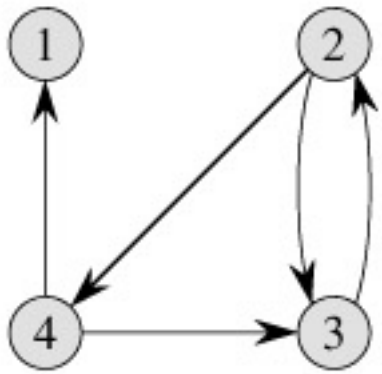
- Utiliza ***O(N^2)*** espaço para guardar a matriz de adjacência
- Complexidade de tempo ***O(N^3)***
- Ou seja, funciona bem para ***N <= 200***
  - ***200^2 = 40.000 x 4 = 160.000bytes***
  - ***200^3 = 8.000.000 operações***
- Se 200 < N <= 500 e você não tiver outra ideia, tente, talvez passa!!!
  - ***500^3 = 125.000.000 operações!***

Aplicações - Fecho Transitivo

- Dada a descrição de um grafo direcionado, responder, para cada par de vértices, se existe pelo menos um caminho entre eles

Aplicações - Fecho Transitivo

- Matriz de adjacência binária
  - 1 - se existe uma aresta de i para j
  - 0 - caso contrário



-	1	2	3	4
1	1	0	0	0
2	0	1	1	1
3	0	1	1	0
4	1	0	1	1

O Algoritmo

```
bool grafo[MAXV][MAXV]; // 1 existe uma aresta, 0 cc

for (int i = 0; i < N; ++i) {
    for (int j = i; j < N; ++j)
        grafo[i][j] = grafo[j][i] = false;

    grafo[i][i] = true; // existe um caminho de i para i
}
... // substitui o min() por um OR (||)
for (int k = 0; k < N; ++k) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            grafo[i][j] = grafo[i][j] ||
                (grafo[i][k] && grafo[k][j]);
        }
    }
}
```

Reconstrução do caminho

Alguns problemas podem pedir que você informe além dos menores caminhos, o caminho em si.

Para isso, podemos armazenar facilmente as informações necessárias para a reconstrução do caminho utilizando outra matriz NxN e ir preenchendo na medida que uma relaxação é feita.

Pra imprimir fazer um backtracking nessa matriz.

Reconstrução do caminho

```
int g[MAXV][MAXV]; // pesos das arestas
int path[MAXV][MAXV]; // armazenamento do caminho

// Inicializa o grafo e lê o grafo
// Inicializa path[i][j] com -1

for (int k = 0; k < N; ++k) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            if (g[i][k] + g[k][j] < g[i][j]) {
                g[i][j] = g[i][k] + g[k][j];
                path[i][j] = k; // de i pra j, passei por k
            }
        }
    }
}
```

Reconstrução do caminho

```
void print_path(int i, int j)
{
    if (g[i][j] == INF) {
        cout << “Sem caminho\n”;
        return;
    }

    // Se não há alguém entre i e j (menor caminho é
    direto)
    if (path[i][j] == -1) {
        cout << “ “;
        return;
    }

    print_path(i, path[i][j]);
    cout << path[i][j] << “ “;
    print_path(path[i][j], j);
}
```

Alguns Problemas

SpojBR - MINIMO

Codeforces Round #33 Probl. B - String Problem

UVa 821 - Page Hopping
UVa 10171 - Meeting Prof. Miguel...
UVa 10724 - Road Construction
UVa 10793 - The Orc Attack
UVa 10803 - Thunder Mountain
UVa 10987 - AntiFloyd
UVa 11015 - Rendezvous

Dúvidas?