Caminho Mínimo entre todos os pares de vértices
Professor Wladimir

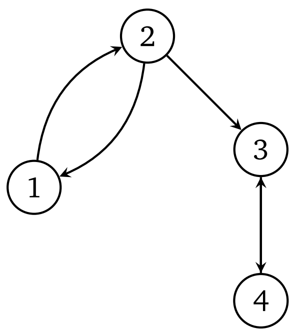
1 Fecho Transitivo

Dado uma relação R sobre um conjunto A, a relação R é dita transitiva se $(a, b) \in R$ e $(b, c) \in R$ então $(a, c) \in R$ para todo a,b e c.

Na matemática, a relação $<$ no conjunto $\mathbb{R}$ é uma relação transitiva, a relação $|$ (divide) no conjunto $\mathbb{Z}$ também é uma relação transitiva. Por exemplo, 2 divide 4 e 4 divide 16, logo 2 divide 16.

Dado uma relação R sobre um conjunto A, queremos encontrar a menor relação R^* tal que $R \subseteq R^*$ e R^* é transitiva.

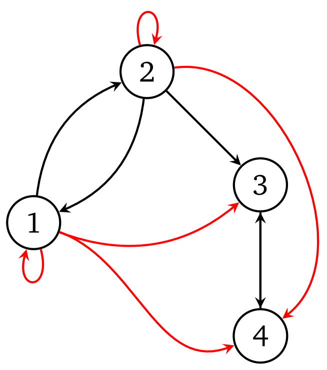
Exemplo: Sejam $A = \{1, 2, 3, 4\}$ e $R = \{(1, 2), (2, 3), (3, 4), (2, 1)\}$. Ache o fecho transitivo de R.



Achar o fecho transitivo equivale encontrar a relação de conectividade. Precisamos computar todos os caminhos:

- a partir do vértice 1, temos caminhos para: 2, 3, 4 e 1
- a partir do vértice 2, temos caminhos para: 2, 1, 3 e 4
- o único outro caminho é aquele que vai do vértice 3 para o 4

O fecho transitivo é:



Matriz de Adjacência de uma relação

Seja M_R a matriz de adjacência de uma relação R (ou de um grafo dirigido), onde:

- Cada entrada $M_R(i, j)$ é igual a 1 se existe uma aresta (ou par na relação) de i para j;

- $M_R(i,j) = 0$ caso contrário.

A matriz $M_{R\odot}^k$ representa a existência de caminhos de comprimento exatamente k. Para construir o **fecho transitivo**, basta calcular:

$$M^T = M_R \vee (M_R)_\odot^2 \vee (M_R)_\odot^3 \vee \cdots \vee (M_R)_\odot^k,$$

até que uma potência não adicione mais nenhuma nova conexão, ou seja:

$$(M_R)_\odot^k = (M_R)_\odot^{k+1}.$$

Esse processo sempre estabiliza após no máximo n passos, onde n é o número de vértices da matriz.

Assim, o **fecho transitivo de** R pode ser obtido computando as potências booleanas de M_R até que nenhuma nova conexão seja descoberta.

Algebricamente, o fecho transitivo poderia ser calculado utilizando as potências de matrizes:

$$M_R = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad (M_R)_\odot^2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$(M_R)_\odot^3 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad (M_R)_\odot^4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Algoritmo de Floyd-Warshall

Seja um grafo direcionado com n vértices. Podemos representar a existência de caminhos entre pares de vértices usando uma sequência de matrizes booleanas W^k, definidas da seguinte forma:

- $W^k[i][j] = 1$ se existe um caminho de i para j cujos **vértices internos** (se existirem) pertencem ao conjunto $\{1, 2, \ldots, k\}$.
- W^0 é a **matriz de adjacência** do grafo, incluindo os laços (se houver), ou seja, $W^0[i][j] = 1$ se existe uma aresta direta de i para j.

A construção das matrizes W^k é feita de forma iterativa, a partir da seguinte recorrência:

$$W^k[i][j] = W^{k-1}[i][j] \vee \left(W^{k-1}[i][k] \wedge W^{k-1}[k][j]\right)$$

Em palavras: existe um caminho de i para j usando vértices internos no conjunto $\{1, \ldots, k\}$ se:

- Já existia esse caminho usando apenas vértices internos de $\{1, \ldots, k-1\}$, ou
- Existe um caminho de i para k **e** um caminho de k para j, ambos usando apenas vértices internos de $\{1, \ldots, k-1\}$.

Ao final da execução, a matriz W^n indica se existe **algum caminho (de qualquer comprimento)** entre cada par de vértices, ou seja, ela representa o **fecho transitivo** da relação de adjacência original do grafo.

Esse algoritmo tem complexidade $O(n^3)$ e é uma forma elegante de computar acessibilidade em grafos via programação dinâmica.

Código C++

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> ii;
int main(){
    int N = 4;
    vector <ii> edges;
    edges.push_back({1,2});
    edges.push_back({2,1});
    edges.push_back({2,3});
    edges.push_back({3,4});
    vector<vector<vector<int>>> W(N+1,
        vector<vector<int>>(N+1, vector<int>(N+1, 0))
    );
    for(auto & [a,b] : edges){
        printf("edge %d %d\n", a, b);
        W[a][b][0] = 1;
    }
    printf("W(%d) = \n", 0);
    for(int i = 1; i <= N; i++){
        for(int j = 1; j <= N; j++){
            printf("%2d ", W[i][j][0]);
        }
        printf("\n");
    }
    for(int k = 1; k <= N; k++){
        printf("W(%d) = \n", k);
        for(int i = 1; i <= N; i++){
            for(int j = 1; j <= N; j++){
                W[i][j][k] = W[i][j][k-1] || (W[i][k][k-1] && W[k][j][k-1]);
                printf("%2d ", W[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }
}
/*
edge 1 2
edge 2 1
edge 2 3
edge 3 4
W(0) =
 0  1  0  0
 1  0  1  0
 0  0  0  1
 0  0  0  0
W(1) =
 0  1  0  0
 1  1  1  0
 0  0  0  1
 0  0  0  0

W(2) =
 1  1  1  0
 1  1  1  0
 0  0  0  1
 0  0  0  0

W(3) =
 1  1  1  1
 1  1  1  1
 0  0  0  1
 0  0  0  0
```

```

W(4) =
 1  1  1  1
 1  1  1  1
 0  0  0  1
 0  0  0  0
*/
```

Final Mundial 2008

O diretor regional do ICPC no Brasil deseja planejar rotas aéreas para as finais mundiais em Banff (2008), considerando:

- Uma malha aérea com voos diretos (sem escalas) entre cidades, cada um com um custo associado.
- Cidades ordenadas por preferência de escala (cidade 1 é a mais preferível, cidade 2 a segunda, etc.).
- Consultas que definem origem (`o`), destino (`d`) e um limite `t` de cidades que podem ser usadas como escalas (apenas cidades de 1 a `t`).

2 Objetivo

Para cada consulta, calcular o custo mínimo do voo entre `o` e `d`, permitindo no máximo escalas nas cidades `1, 2, ..., t`.

3 Entrada

Cada instância contém:

- Dois inteiros `n` (nº de cidades) e `m` (nº de voos).
- `m` linhas com `u`, `v`, `w` (origem, destino, custo do voo direto).
- Um inteiro `c` (nº de consultas).
- `c` linhas com `o`, `d`, `t` (origem, destino, limite de cidades para escalas).

4 Restrições

- $1 \leq n \leq 100$ (cidades).
- $1 \leq m \leq 10^5$ (voos).
- $0 \leq w \leq 100$ (custo por voo).
- $1 \leq c \leq 10^4$ (consultas).
- $0 \leq t \leq n$ (limite de cidades para escalas).

5 Exemplo de Consulta

Para `t = 1`, o voo pode ser:

- Direto (`o` $\to$ `d`).
- Com 1 escala na cidade `1` (`o` $\to$ `1` $\to$ `d`).

https://br.spoj.com/problems/MINIMO/

```
#include <stdio.h>
#define INF 1000000001 // INF+INF não dá overflow

#define MAXN 101
#define MAXC 10001
int d[MAXN][MAXN][MAXN];
int n,m,maxt,u,v,w,i,j,k,c,teste=1;
int o[MAXC],dt[MAXC],t[MAXC];
int min(int a,int b){
    return a<b?a:b;
}
int main(){
    while( scanf("%d %d",&n,&m) > 0 ){
           if(teste>1)printf("\n");
              for(i=1;i<=n;i++)
            for(j=1;j<=n;j++){
             d[i][j][0]=INF;
            }
           for(i=1;i<=n;i++) d[i][i][0]=0;
           for(i=1;i<=m;i++){
            scanf("%d %d %d",&u,&v,&w);
            d[u][v][0]= min ( d[u][v][0], w );
           }
           scanf("%d",&c);
           for(i=1;i<=c;i++){
             scanf("%d %d %d",&o[i],&dt[i],&t[i]);
           }
           for(k=1;k<=n;k++){
             for(i=1;i<=n;i++){
              for(j=1;j<=n;j++){
               d[i][j][k] = min( d[i][j][k-1] , d[i][k][k-1] + d[k][j][k-1])  ;
             }
            }
           }
          printf("Instancia %d\n",teste++);
          for(i=1;i<=c;i++){
              if(d[o[i]][dt[i]][t[i]]!=INF)
                printf("%d\n",d[o[i]][dt[i]][t[i]]);
              else
                printf("-1\n");
          }
          printf("\n");
    }
    return 0;
}
```

Quantas dependências

https://br.spoj.com/problems/DEPENDEN/

```
#include <stdio.h>
#include <string.h>

int g[101][101];

int main(){
  int n,i,j,k,d,t;

  while(1){

        scanf("%d",&n);

        if(n==0)
```

```
            return 0;

        //tava errado aqui
        //memset(g,0,sizeof(int));
        memset(g,0,sizeof(g));

        for(i=1;i<=n;i++){
            scanf("%d",&t);
            for(j=1;j<=t;j++){
                scanf("%d",&d);
                g[i][d]=1;
            }
        }

        for(k=1;k<=n;k++)
            for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                    if(g[i][k]==1 && g[k][j]==1){
                        //printf("encontrando dependencia indireta %d %d\n",i,j);
                        g[i][j]=1;
                    }

        int max=0;
        int maxi=0;
        int atual;

        for(i=1;i<=n;i++){
            atual = 0;
            for(j=1;j<=n;j++){
                if(g[i][j]==1)
                    atual++;
            }
            if(atual > max){
                max = atual;
                maxi=i;
            }
        }
        printf("%d\n",maxi);
    }
    return 0;
}
```

Exercícios

1. https://judge.beecrowd.com/pt/problems/view/1384
2. https://judge.beecrowd.com/pt/problems/view/1148
3. https://br.spoj.com/problems/DENGUE/
4. https://www.spoj.com/problems/CHICAGO/
5. https://www.spoj.com/problems/ARBITRAG/
6. https://br.spoj.com/problems/REUNIAO2/