# O algoritmo de Bellman-Ford com ordenação de vértices

Neste trabalho, consideraremos uma solução para o Problema do Caminho de Custo Mínimo (PCCM). Este problema consiste em encontrar todos os caminhos de custo mínimo de um dado vértice de origem a todos os demais vértices de um grafo orientado com custos nos arcos. O custo de um caminho é a soma dos custos dos seus arcos.

Observe que pode não existir um caminho partindo da origem para alguns vértices do grafo. Neste caso, basta identificar tais vértices como inatingíveis.

Outro problema ocorre quando o grafo possuir um ciclo orientado de custo total negativo. Neste caso, o PCCM se torna NP-Difícil e não será resolvido neste trabalho.

Você deverá usar uma adaptação do algoritmo de Bellman-Ford para resolver o PCCM e identificar vértices inatingíveis e ciclos negativos.

# O Algoritmo

O Algoritmo 1 é conhecido como Algoritmo de Bellman-Ford e consulta repetidamente todos os arcos do grafo e atualiza os caminhos conhecidos quando houver uma melhora. Ao final do processo, uma última iteração pode ser feita para identificar possíveis ciclos negativos.

**Algorithm 1** Bellman-Ford (1956)

```
1:  procedure BELLMAND-FORD(G,s,Anterior,Distancia)
2:      Inicializa vetores Anterior e Distancia para todo v ∈ V(G).
3:      Distancia[s] ← 0
4:      repeat
5:          for cada arco (u, v) ∈ A(G) do
6:              if Distancia[u] +c((u, v)) < Distancia[v] then
7:                  Distancia[v] ← Distancia[u] +c((u, v))
8:                  Anterior[v] ← u
9:              end if
10:         end for
11:     until execute por n(G) − 1 vezes
12: end procedure
```

A ordem de visitação aos arcos na linha 5 é indiferente para o algoritmo e, neste trabalho, escolheremos uma ordem específica. Mantem-se uma ordenação O dos vértices e, a cada iteração, eles serão considerados na ordem O e todos os arcos de saída de cada vértice serão consultados na ordem que estiverem na estrutura de dados.

Para a primeira iteração, a ordem deve iniciar com o vértice de origem e seguir com os demais em ordem numérica.

Enquanto se executa o algoritmo, marcam-se como processados os vértices da ordem, cujos arcos de saída serão considerados. Ao considerar os vizinhos do vértice sendo processado, marca-se os que foram reduzidos e identifica-se os que foram reduzidos após serem processados. Após processar todos os vértices, adicionam-se à sequência da próxima iteração aqueles que foram reduzidos após serem processados, depois os que foram reduzidos, mas posteriormente

processados e, por fim, os que não foram reduzidos, sempre na mesma ordem relativa (uns aos outros) que estavam na ordem da iteração atual.

Ao final do algoritmo, deve-se implementar a etapa de verificação da existência de ciclo negativo. Ou seja, executar mais uma iteração e verificar se nenhuma atualização é realizada.

Um resumo do que deve ser implementado é apresentado no Algoritmo 2.

**Algorithm 2** Algoritmo a Implementar

```
procedure PCCM(G,s,Anterior,Distancia)
    Inicializa vetores Anterior e Distancia para todo v ∈ V(G).
    Distancia[s] ← 0
    Prepare a ordem inicial dos vértices O
    repeat
        Imprima a ordem de visitação dos vértices O
        Inicialize a ordem da próxima iteração O'
        Desmarque todos os vértices
        for cada u ∈ O do
            Marque u como processado
            for cada arco (u, v) ∈ A(G) do
                if Distancia[u] +c((u, v)) < Distancia[v] then
                    Distancia[v] ← Distancia[u] +c((u, v))
                    Anterior[v] ← u
                    Marque v como reduzido ou reduzido após processado, conforme o caso
                end if
            end for
        end for
        Crie uma ordem O', adicionando todos os vértices de O da forma descrita no trabalho
        Substitua O por O'.
    until execute por n(G) − 1 vezes
    Indentifique ciclos negativos
    Imprima as informações solicitadas
end procedure
```

### Extra

Caso exista um ciclo negativo, identifique os vértices pertencentes ao ciclo, que devem ser impressos pelo algoritmo.

## Implementação

Para armazenar o grafo, você pode usar uma lista de adjacências ou até mesmo um vetor de adjacências (para simplificar o percurso). Não utilize matrizes de adjacência ou incidência pois ocupam muito espaço em memória e tornam o algoritmo muito lento. Também não utilize estruturas de dados prontas (em pacotes) para armazenar o grafo.

O programa deve ser inteiramente codificado pelo aluno sem auxílio de nenhuma ferramenta de geração de código.

Seu programa deverá processar a linha de comando, executar o que se pede, imprimindo a saída esperada e terminar.

## Linguagem e Execução

Seu programa deve ser implementado em C, C++ ou Python. Deve se chamar pccm.c, pccm.cpp ou pccm.py.

Ele será compilado e executado em uma máquina x86_64 (amd64) em Debian Linux 12.5,como as máquinas dos laboratórios da Facom.

Para os dois primeiros casos, ele será compilado com gcc versão 12.2.0 usando o comando:

gcc -std=c11 -Wall -o pccm *.c

ou

g++ -std=c++20 -Wall -o pccm *.cpp

Para o terceiro caso, será executado com Python 3.11.2 com:

## Entrada e Saída

A linha de comando será como:

./pccm grafo.txt s

ou

./python3 pccm.py grafo.txt s

O programa deve ler o grafo do arquivo grafo.txt e executar o algoritmo solicitado com origem em s até todos os destinos.

No início de cada iteração (de 1 a no máximo m(G)), imprima:

O k $v_1 v_2 \dots v_n$

A ordem de análise dos vértices na k-ésima iteração será $v_1, v_2, \dots, v_n$.

Ao final da última análise, avalie o estado final e determine se existe um ciclo negativo no grafo.

Caso exista um ciclo negativo imprima a linha abaixo e termine o programa.

C

Se você tiver implementado a identificação do ciclo, imprima:

C v n $v_1 v_2 \dots v_n v_1$

Nesta linha v é o custo do ciclo, n é número de vértices e $v_1 v_2 \dots v_n v_1$ são os vértices do ciclo na ordem dos arcos (incluindo a repetição do vértice v1), iniciando pelo vértice de menor número.

Caso não haja um ciclo negativo, imprima, para todos os vértices (incluindo o próprio s) em ordem numérica, uma linha contendo:

Caso tenha encontrado o caminho de custo mínimo de s até t:

P t v c $v_1 v_2 \dots v_n$

A linha apresenta as informações do menor caminho de s para t: o valor (v), o comprimento (c) e o caminho em si (sequência de vértices: $v_1 v_2 \dots v_n$, sendo $s = v_1$ e $v_n = t$).

Ou apenas

P t v

Caso você não tenha implementado a identificação do caminho.

Caso não exista caminho de s a t:

U t

Caso ocorra algum problema na especificação do grafo ou do vértice de origem, imprima uma linha contendo:

E

### Sobre o grafo

O grafo é simples, ou seja, não possui laços ou arestas múltiplas, não terá mais de $10^6$ vértices nem mais de $10^7$ arcos. O custo de cada arco será entre -1.000 e +1.000, podendo ser 0.

O arquivo que representa o grafo está no formado descrito abaixo:

I n m

N i g- g+

E i j c

T

O propósito das linhas são identificadas pelo caractere inicial.

A primeira linha é do tipo I e contém informações gerais do grafo: número de vértices e número de arcos. Então há uma linha do tipo N para cada vértice contendo o número do vértice de 0 a n-1 com o grau de entrada e o grau de saída. Por fim, há uma linha do tipo E para cada arco do grafo, com o vértice de origem, vértice de destino e custo. O custo será sempre um inteiro.

## Avaliação

Sua nota será zerada e seu trabalho desconsiderado caso falte com ética (não seja autor do trabalho em todo ou em parte), implemente um algoritmo diferente do solicitado, use uma linguagem diferente das especificadas ou utilize pacotes ou funções disponíveis em bibliotecas para executar parte das atividade solicitadas.

Seu programa será analisado contra um conjunto de grafos e vértices de origem e sua nota dependerá da execução correta nestes testes. Por isso, não imprima na saída nenhuma informação extra.

O primeiro grupo de testes contém grafos pequenos e moderados (até 100 vértices) e o segundo contém grafos maiores. Serão avaliados as situações da Tabela 1. Apenas os itens indicados usam grafos do segundo grupo.

| Atividade | Nota |
|---|---|
| Imprime e utiliza a sequência de vértices como solicitado | 1,5 |
| Computa o valor correto dos caminhos mínimos em grafos com todos os vértices atingíveis e custos não-negativos. | 2,5 |
| Computa o valor correto dos caminhos e identifica vértices inatingíveis em grafos com custos não-negativos | 1,0 |
| Computa o valor correto dos caminhos mínimos em grafos com todos os vértices atingíveis mas com arcos negativos (sem ciclos negativos) | 1,0 |
| Computa o valor correto dos caminhos mínimos e identifica vértices inatingíveis em grafos com arcos negativos (sem ciclos negativos) | 1,0 |
| Imprime os caminhos encontrados corretamente | 1,5 |
| Identifica um ciclo negativo | 1,0 |
| Executa no tempo esperado para grafos grandes | 1,0 |
| Imprime um ciclo negativo encontrado | 1,0 |

Tabela 1: Tabela de pontuação