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 a cada repetição do laço. Para economizar memória, ao invés de manter uma lista de arcos na ordem desejada, usaremos uma ordem dos vértices. Assim, se os vértices estão na ordem O, a cada iteração, cada um deles será considerados na ordem O e todos os arcos de saída de cada vértice serão consultados.

Para a primeira iteração, a ordem deve iniciar com o vértice de origem e seguir com os demais em ordem numérica *crescente*. Essa ordem será usada em toda iteração ímpar e será denominada O_I. Na segunda iteração e nas demais iterações pares, será usada a ordem O_P que inicia com o vértice de origem e em seguida contém os demais vértices em ordem numérica *decrescente*. Assim, com a exceção do primeiro vértice (que é a origem) as ordens O_I e O_P tem os vértices em ordens inversas.

Enquanto se executa o algoritmo, se em alguma iteração nenhum vértice tiver sua distância reduzida, então o algoritmo deve ser parado prematuramente, pois novas iterações não vão alterar a situação 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 as ordens de visitação dos vértices O_I e O_P
    Imprima as ordens O_I e O_P
    repeat
        Escolha a ordem da rodada O entre O_I e O_P
        for cada u ∈ O do
            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 que houve atualização nesta rodada.
                end if
            end for
        end for
        if Não houve atualização nesta rodada then
            Pare a execução do laço principal
        end if
    until execute por n(G) − 1 vezes
    Imprima as informações de rodadas e os vetores Distancia e Anterior
    Indentifique ciclos negativos
    if Há ciclos negativos then
        Imprima informações sobre o ciclo negativo
    else
        Imprima os caminhos de menor custo para todos os vértices
    end if
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.

Para escolher um valor para representar o infinito, na inicialização das distâncias, você pode usar $2 \cdot 10^9$. É um número que cabe na representação numérica do tipo inteiro em várias linguagens. Note que, durante o processamento e no final do algoritmo, os valores em Distancia podem ser reduzidos por causa das arestas negativas e, ainda assim, o vértice ser inatingível. Portanto, considere qualquer valor maior que 10^9 como sendo infinito.

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 12.11, compatível 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

No caso de Python, será executado com a versão 3.11.2 .

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.

Antes de iniciar as iterações do algoritmo, imprima as duas ordens de vértices usadas (O_I e O_P), uma por linha, no formato abaixo.

O I v_1 v_2 $\ldots$ v_n

O P v_1 v_2 $\ldots$ v_n

Ao final da última rodada, imprima o estado final do algoritmo, ou seja, o número da última rodada completa (k, iniciando em 1) e o conteúdo dos vetores Distancia e Anterior no formato abaixo. O valor d_i = Distancia[i] e a_i = Anterior[i].

F k

D d_1 d_2 $\ldots$ d_n

A a_1 a_2 $\ldots$ a_n

Então, identifique se existe um ciclo negativo no grafo. Caso exista um ciclo negativo imprima a linha abaixo e termine o programa.

CN

Se você tiver implementado a atividade extra de impressão do ciclo, imprima:

C v n $v_1 v_2 \ldots v_n v_1$

Nesta linha v é o custo do ciclo, n é número de vértices e $v_1 v_2 \ldots v_n v_1$ são os vértices do ciclo na ordem dos arcos (incluindo a repetição do vértice v_1), iniciando (v_1) 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 identificando o caminho encontrado, se houver.

P t v c v_1 v_2 ... 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: $v_1 v_2 \ldots v_n$, sendo $s = v_1$ e $v_n = t$).

Ou apenas

P t v

Caso você não tenha implementado a impressão dos vértices do caminho.

Caso não exista caminho de s a t, imprima:

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 -100 e +100, 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 as ordens corretas dos vértices	0,5
Para assim que a rodada não tiver atualizações e imprime o valor de k correto	0,5
Imprime os vetores Anterior e Distancia com os valores corretos	0,5
Computa e imprime o valor correto dos caminhos mínimos existentes	4,0
Identifica e imprime os vértices inatingíveis	1,0
Imprime os caminhos mínimos existentes e encontrados corretamente	1,5
Identifica a existência de 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