**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS**

**CAMINHO MÍNIMO APLICADO À CLUSTERS B-WOULF**

**HIGOR FERREIRA ALVES SANTOS**

**Goiânia**

**2019**

# Prólogo

Há diversas aplicações em muitas áreas que podem ser resolvidas com uma ferramenta matemática denominada Grafo.

Segundo Rosen “Um grafo G=(V, E) consiste em V, um conjunto não vazio de vértices (ou nós), e E, um conjunto de arestas. Cada aresta tem um ou dois vértices associados a ela, chamados de suas extremidades. Dizemos que uma aresta liga ou conecta suas extremidades.”, (ROSEN KENNETH H., Matemática Discreta e Suas Aplicações, 6ª ed).

Neste artigo, abordaremos um uso abstrato e genérico de grafos aplicados a dois ou mais nós de processamento, em que determinado nó requisita o menor caminho para um grafo . Esta requisição deverá ser processada por um nó servo disponível em nossa rede de Clusters (Figura 1). Abordaremos também o problema de garantir a integridade do grafo nos diversos nós, e isto será feito através de uma cópia do grafo para cada nó, sendo que, uma mudança no grafo causada pelo nó , refletirá nos nós , com .

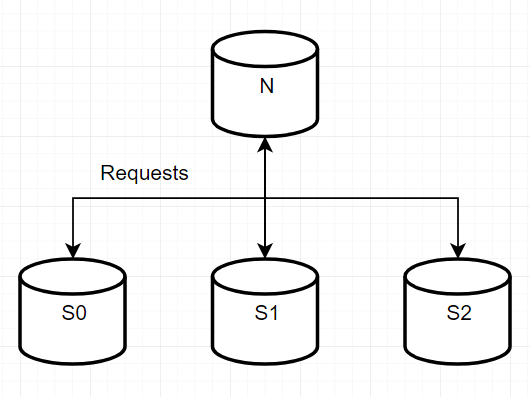


Figura 1: Exemplo de Clusters com n = 3 servos

Basicamente, haverá um nó comandante que se comunicará com seus servos, usaremos o MPI em nossa modelagem, de forma que a comunicação entre os Clusters por meio de mensagens possa ser facilitada.

# Condições iniciais

O grafo escolhido para este trabalho, será um grafo que poderá conter até 65 mil vértices diferentes, sendo que 5 mil arestas serão criadas aletoriamente, resultando em um número não maior que 10 mil vértices úteis. Será um grafo valorado, (com pesos nas arestas), com todas as arestas iniciando com valor 100. Aplicaremos o algoritmo de Bellman Ford, responsável por encontrar o caminho mínimo, que decrementará os pesos das arestas onde ele passar por 1 a cada iteração.

# Representação

Há, basicamente, duas maneiras mais utilizadas para se representar um grafo computacionalmente. Uma forma é através de uma matriz de adjacência, e a outra é a lista de adjacência.

## Matriz de adjacência

A representação na forma de matriz de adjacência, basicamente consiste em uma matriz , sendo o número de vértices do grafo. Esta matriz é formada por 0s e 1s, em que no caso de um grafo não valorado uma aresta qualquer é marcada com 1 na matriz, sendo , e , o 0 indica a ausência de arestas. Também em um grafo valorado, pode-se representar o peso do da aresta colocando-o como valor do par ordenado , a ausência de arestas pode ainda ser o próprio 0 ou qualquer outro valor fora do intervalo de interesse.

### Desvantagem da matriz de adjacência

Embora muito intuitiva para visualizar o Grafo, a matriz de adjacência (figura 2) tem a desvantagem de ocupar de espaço. No nosso caso, dada uma máquina com inteiros de 4 bytes, nosso grafo ocuparia   
Vide: <<https://convertlive.com/u/convert/bytes/to/gigabytes#67600000000>>

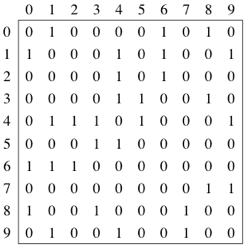
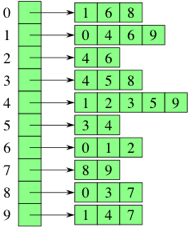


Figura 2: Matriz de adjacência de um grafo com dez vértices

## Lista de adjacência

A representação de um grafo por lista de adjacência pode ser entendida como um grande vetor, em que cada índice é um determinado vértice, que contém um ponteiro para uma lista de vértices adjacentes a , é clado que num grafo finito, . A figura 3 mostra uma representação básica de um grafo por lista de adjacência. Podemos ver claramente que cada vértice do grafo, contém uma lista com seus vértices adjacentes, em outras palavras, temos uma lista de vértices. Por exemplo: é uma aresta no grafo da figura 3, aresta essa que parte de 0 para 1. Se queremos um grafo não direcional, basta inserir os dados como . É claro que, num nível abstrato, vamos lidar com ponteiros e nós.

Figura 3: Grafo representado em lista de adjacência



### Espaço ocupado pela lista de adjacência

Na representação por lista de adjacência, teremos um total de ponteiros para conjuntos de até  
 vértices, estas são as representações de nossas arestas. Com respeito ao espaço ocupado, para ser mais preciso, nossa lista ocupará elementos, pois, devido ao fato de nosso grafo não ser direcionado, a aresta tem de ser representada com , e . Devemos levar também em consideração, que os ponteiros possuem um tamanho fixo dada a arquitetura da máquina.

### Estimando o tamanho do grafo com base em lista de adjacência

Dada uma arquitetura de 64bits, sabemos que o endereçamento desta máquina necessita de 8bytes. Sendo assim, nosso grafo inicial consistiria em um buffer de ponteiros com  
 (PB = Pointer Buffer). Esses ponteiros deverão apontar para outros buffers que são os vértices adjacentes a cada vértice do nosso grafo, assumindo que estes buffers serão buffers simples de inteiros, e que os inteiros ocupam 4bytes, dizemos que os buffers de adjacência irão nos custar (AB = Adjacency Buffer).

Sendo assim, o tamanho total de nosso grafo representado como lista de adjacência, seria, num primeiro momento , que é centenas de vezes menor que a representação anterior.

É claro que nosso grafo ocupará um espaço muito maior, pois não usaremos simples valores inteiros, mas uma lista de um tipo abstrato de dados chamado Vertex, que conterá o valor do nosso vértice, bem como um campo para cor, outro para o peso da aresta, e também um ponteiro para o próximo elemento da lista. Isto veremos mais adiante.

# Modelagem do grafo