Análise de Algoritmos de Caminho Mínimo: Dijkstra, Min-Max e Max-Min para Gargalos

Lucas Gualtieri Firace[1], **Gabriel Martins Rajão**[1], **Luiza Dias Corteletti**[1]

[1]Instituto de Ciências Exatas e Informática - PUC Minas

Resumo. *O artigo explora três algoritmos de otimização de caminho em grafos: Dijkstra, Min-Max e Max-Min. O algoritmo de Dijkstra minimiza o custo total de um caminho, somando os pesos das arestas, enquanto os problemas Min-Max e Max-Min focam na otimização de gargalos. O Min-Max minimiza o maior gargalo, enquanto o Max-Min maximiza o menor gargalo ao longo do caminho. Esses algoritmos têm aplicações práticas em redes de comunicação e tráfego, onde a capacidade de cada aresta é um fator crítico para garantir a eficiência do fluxo.*

1. Definição dos Problemas

Os problemas **Min-Max** e **Max-Min** para gargalos estão relacionados à otimização de caminhos em grafos, com foco nas restrições impostas por gargalos ou limitações de capacidade ao longo das arestas do grafo. Esses problemas diferem do algoritmo de Dijkstra, que visa encontrar o caminho de menor custo total. A seguir, definimos cada um deles e discutimos suas diferenças.

1.1. Problema Min-Max de Gargalos

O objetivo do problema **Min-Max** é encontrar um caminho em um grafo onde o **maior peso (gargalo)** das arestas seja o **menor possível**. Em outras palavras, entre todos os caminhos disponíveis de um nó de origem para um nó de destino, deve-se escolher aquele cujo peso máximo da aresta no caminho seja o menor possível.

1.1.1. Aplicação

Este problema é utilizado em cenários onde o maior limite de uma aresta (como a capacidade de um link) ao longo do caminho é o fator mais crítico, como em redes de fluxo de dados ou logísticas, onde deseja-se minimizar o pior cenário de congestionamento.

1.1.2. Exemplo

Em uma rede de comunicação de dados, o problema Min-Max tentaria escolher o caminho cujo **trecho com o maior atraso (link mais lento)** seja o mais favorável. Por exemplo, se em um caminho temos links com latências de 40ms, 60ms, 80ms e 120ms, o Min-Max tentaria minimizar o caminho em que o maior atraso (120ms) fosse o menor possível, talvez escolhendo um caminho com latências de 40ms, 60ms e 80ms, onde o maior atraso seria 80ms. Dessa forma, Min-Max garante que o pior atraso entre os links no caminho escolhido seria minimizado.

1.2. Problema Max-Min de Gargalos

No problema **Max-Min**, o objetivo é maximizar o **menor peso (gargalo)** em um caminho. Ou seja, entre todos os caminhos disponíveis, escolhe-se o caminho cujo **menor peso da aresta (gargalo)** seja o maior possível. O foco é tentar garantir o maior gargalo mínimo ao longo de um caminho.

1.2.1. Aplicação

Este problema é utilizado em situações onde deseja-se maximizar a menor capacidade possível em um caminho, garantindo que o menor gargalo em um caminho seja o maior possível. Isso é importante em cenários de comunicação ou transporte, onde é essencial garantir que o caminho tenha a maior capacidade mínima possível para evitar congestionamentos.

1.2.2. Exemplo

Imagine uma rede de fluxo de dados em que cada aresta representa a capacidade de um link. O problema Max-Min busca o caminho onde o **link mais restritivo** (o gargalo) tenha a maior capacidade possível. Por exemplo, se em um caminho temos links com capacidades 100, 50, 30 e 70, o gargalo seria 30. O Max-Min tenta encontrar o caminho cujo gargalo seja o maior entre todos os possíveis caminhos.

2. Diferenças em relação ao Algoritmo de Dijkstra

O algoritmo de Dijkstra busca o **caminho de menor custo total** entre dois nós em um grafo, somando os pesos das arestas ao longo de um caminho. Já os problemas de **Min-Max** e **Max-Min** focam em **otimizar os gargalos** ao longo do caminho, não considerando o custo total do caminho, mas sim as capacidades mínimas ou máximas nas arestas.

- **Dijkstra**: Foca no **custo total** do caminho, somando os pesos de todas as arestas.
- **Min-Max**: Minimiza o maior gargalo ao longo de um caminho.
- **Max-Min**: Maximiza o menor gargalo ao longo de um caminho.

Esses dois problemas são úteis em contextos onde o foco não é necessariamente minimizar a distância ou o custo total do caminho, mas sim garantir que as restrições impostas por gargalos ao longo do caminho sejam tratadas da maneira mais eficiente.

3. Aplicações

Esses problemas são frequentemente aplicados em **redes de comunicação, redes de tráfego ou roteamento de dados**, onde a capacidade limitada de cada enlace entre nós precisa ser considerada para garantir um fluxo eficiente.

4. Algoritmo de Dijkstra

O algoritmo de Dijkstra tem como objetivo encontrar o caminho de **menor custo total** de um vértice de origem para todos os outros vértices. Ele usa uma **fila de prioridade (min-heap)** para garantir que o próximo vértice a ser explorado tenha a menor distância acumulada até o momento. O que é propagado de um vértice para o outro é a **soma das distâncias** (ou pesos das arestas) ao longo do caminho.

- A cada iteração, o vértice com a menor distância acumulada é removido da fila de prioridade.
- Para cada vizinho, a distância total (distância acumulada até o vértice atual + peso da aresta) é calculada.
- Se essa nova distância for menor que a distância registrada anteriormente para o vizinho, essa distância é atualizada, e o vizinho é adicionado ou tem sua prioridade atualizada na fila.
- O algoritmo para assim que todos os vértices tiverem sido visitados.

Algorithm 1 Dijkstra's Algorithm

Require: Grafo $G(V, E)$, vértice de origem x
Ensure: Lista de distâncias mínimas D

Inicialize o vetor de distâncias $D[v] \leftarrow \infty \; \forall v \in V$
$D[x] \leftarrow 0$
Inicialize uma fila de prioridade Q
$Q.push(x, 0)$
while Q não estiver vazia **do**
    $u \leftarrow Q.pop()$
    for cada $v \in \Gamma(u)$ **do**
        if $D[u] + w(u, v) < D[v]$ **then**
            $D[v] \leftarrow D[u] + w(u, v)$
            if $v \notin Q$ **then**
                $Q.push(v, D[v])$
            else
                Atualize a chave de v em Q para $D[v]$
            end if
        end if
    end for
end while
return D

5. Algoritmo Max-Min (Capacidade Máxima)

O algoritmo Max-Min busca encontrar o caminho entre a origem e o destino de modo a **maximizar o menor gargalo (capacidade mínima) ao longo do caminho**. Ele utiliza uma **fila de prioridade (max-heap)** para sempre expandir o vértice que tem a maior capacidade mínima disponível até o momento. Aqui, **o valor propagado entre vértices é a menor capacidade (gargalo)** no caminho até o momento. O objetivo é garantir que o caminho escolhido maximize a menor capacidade entre os vértices.

- A cada iteração, o vértice com a maior capacidade mínima acumulada é removido da fila.
- Para cada vizinho, é calculada a nova capacidade mínima, que é o menor valor entre a capacidade mínima do vértice atual e o peso da aresta que o conecta ao vizinho.
- Se essa nova capacidade mínima for maior que a registrada anteriormente para o vizinho, essa capacidade é atualizada, e o vizinho é adicionado ou tem sua prioridade atualizada na fila.
- O algoritmo para assim que vértice de destino é removido da fila.

Algorithm 2 Algoritmo Max-Min Path

Require: Grafo $G(V, E)$, vértice de origem $source$, vértice de destino $target$
Ensure: Caminho $P(source, target)$ que maximiza a menor capacidade

Inicialize o vetor de predecessores $predecessors[v] \leftarrow -1 \; \forall v \in V$
Inicialize a fila de prioridade $Q.push(v, -\infty) \; \forall v \in V$
Inicialize o vetor de capacidades minimas $nodeCapacity[v] \leftarrow -\infty \; \forall v \in V$
$nodeCapacity[source] \leftarrow \infty$
$Q.push(source, \infty)$
while Q não estiver vazia **do**
  $u \leftarrow Q.pop()$ {Retira o vértice com maior capacidade disponível}
  if $u = target$ **then**
    interrompemos a busca e reconstruímos o caminho
  end if
  for cada $v \in \Gamma(u)$ **do**
    $newCapacity \leftarrow \min(nodeCapacity[u], w(u, v))$
    if $nodeCapacity[v] < newCapacity$ **then**
      $predecessors[v] \leftarrow u$
      $nodeCapacity[v] \leftarrow newCapacity$
      Atualize a chave de v em Q para $nodeCapacity[v]$
    end if
  end for
end while
return `constructPath(source, target, predecessors)`

6. Algoritmo Min-Max (Capacidade Mínima)

O algoritmo Min-Max tenta minimizar o maior gargalo possível ao longo do caminho, ou seja, **encontrar o caminho em que o maior peso das arestas seja o menor possível**. Ele usa uma **fila de prioridade (min-heap)** para garantir que o próximo vértice a ser explorado tenha o menor gargalo máximo até o momento. Neste caso, **o valor propagado entre vértices é o maior peso (gargalo)** no caminho até o momento. O objetivo é garantir que o caminho escolhido minimize o maior gargalo.

- A cada iteração, o vértice com o menor gargalo máximo acumulado é removido da fila.
- Para cada vizinho, é calculado o novo gargalo máximo, que é o maior valor entre o gargalo máximo do vértice atual e o peso da aresta que o conecta ao vizinho.
- Se essa nova capacidade máxima for menor que a registrada anteriormente para o vizinho, essa capacidade é atualizada, e o vizinho é adicionado ou tem sua prioridade atualizada na fila.
- O algoritmo para assim que vértice de destino é removido da fila.

Algorithm 3 Algoritmo Min-Max Path

Require: Grafo $G(V, E)$, vértice de origem $source$, vértice de destino $target$
Ensure: Caminho $P(source, target)$ que maximiza a menor capacidade

Inicialize o vetor de predecessores $predecessors[v] \leftarrow -1 \; \forall v \in V$
Inicialize a fila de prioridade $Q.push(v, \infty) \; \forall v \in V$
Inicialize o vetor de capacidades maximas $nodeCapacity[v] \leftarrow \infty \; \forall v \in V$
$nodeCapacity[source] \leftarrow \infty$
$Q.push(source, -\infty)$
while Q não estiver vazia **do**
    $u \leftarrow Q.pop()$ {Retira o vértice com menor capacidade disponível}
    if $u = target$ **then**
        interrompemos a busca e reconstruímos o caminho
    end if
    for cada $v \in \Gamma(u)$ **do**
        $newCapacity \leftarrow \max(nodeCapacity[u], w(u, v))$
        if $nodeCapacity[v] > newCapacity$ **then**
            $predecessors[v] \leftarrow u$
            $nodeCapacity[v] \leftarrow newCapacity$
            Atualize a chave de v em Q para $nodeCapacity[v]$
        end if
    end for
end while
return `constructPath(source, target, predecessors)`

6.1. Problemas Teóricos

Nesta subseção, serão discutidos problemas e limitações que devem ser considerados ao implementar o algoritmo de Dijkstra.

6.1.1. Pesos Negativos

O algoritmo de Dijkstra garante encontrar o menor caminho apenas em grafos cujas arestas possuem pesos não negativos. Isso ocorre devido à natureza do algoritmo, que desconsidera caminhos cuja distância já foi superada por uma solução melhor. Quando são utilizados pesos negativos, pode haver situações em que o custo total de um caminho diminui conforme ele é percorrido, levando o algoritmo a ignorar essa rota inicialmente considerada mais cara.

6.1.2. Ciclos

Um grafo que contém ciclos negativos causará uma execução infinita do algoritmo. Se o algoritmo de Dijkstra encontrar um ciclo com pesos negativos, a distância total pode diminuir a cada iteração, resultando em uma repetição interminável do ciclo. Isso acontece porque, a cada passagem pelo ciclo, uma nova distância menor é obtida, o que impede a conclusão do algoritmo.

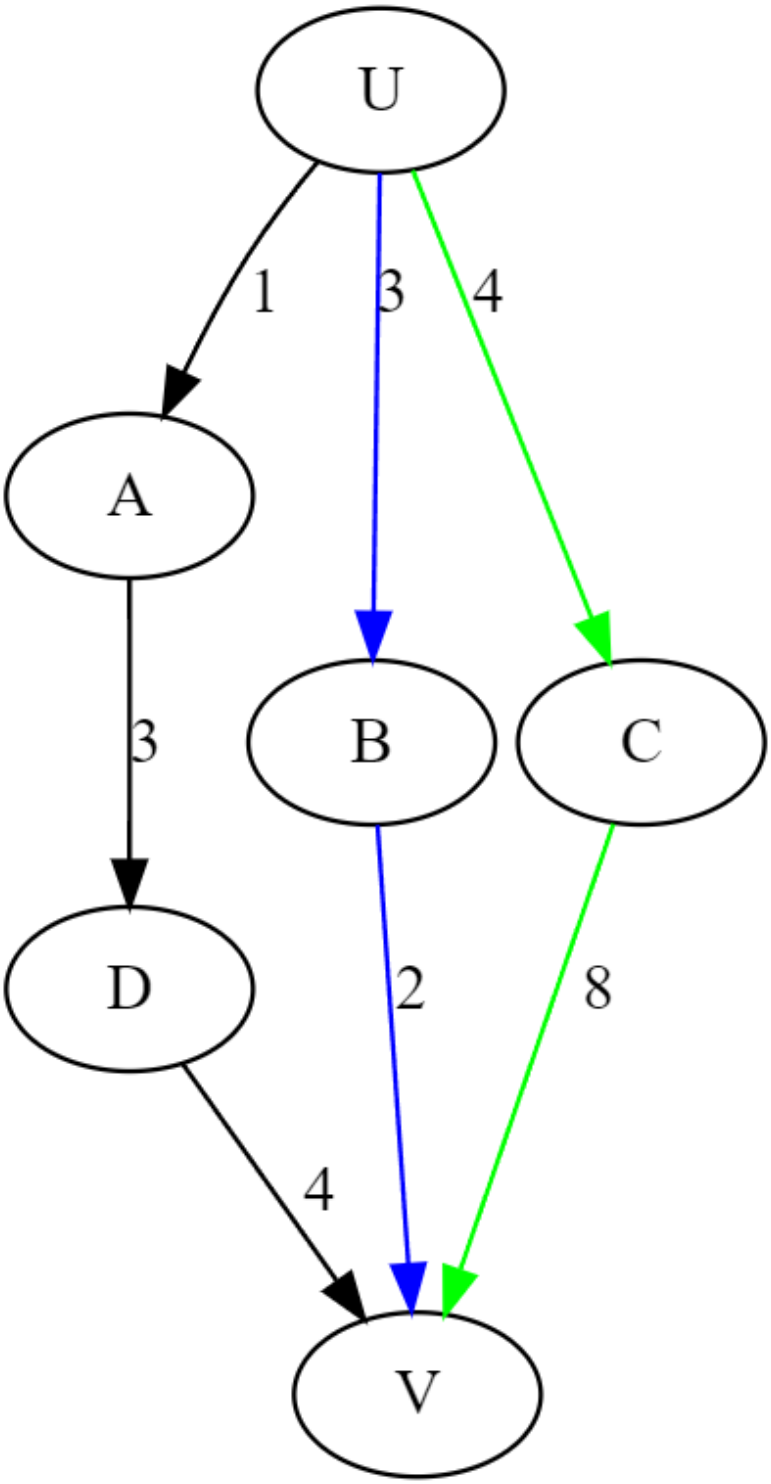

Figure 1. Caminho em azul representa o caminho encontrado pelo algoritmo Min-Max, onde o objetivo é minimizar o maior gargalo (capacidade máxima) ao longo do caminho. Caminho em verde representa o caminho encontrado pelo algoritmo Max-Min, que busca maximizar a menor capacidade (gargalo) ao longo do caminho.