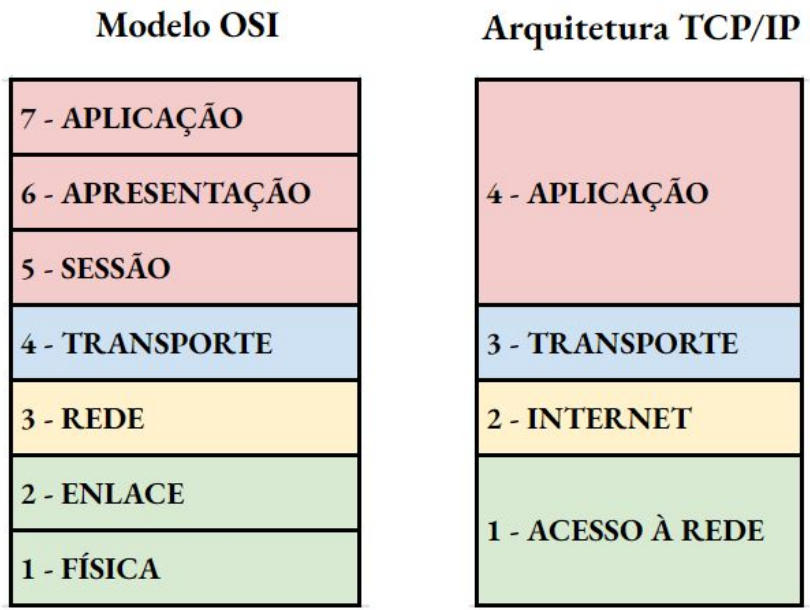
URI | UNIVERSIDADE REGIONAL INTEGRADA DO ALTO URUGUAI E DAS MISSÕES

30-765

Redes de Computadores II

MSc. Fernando Schubert

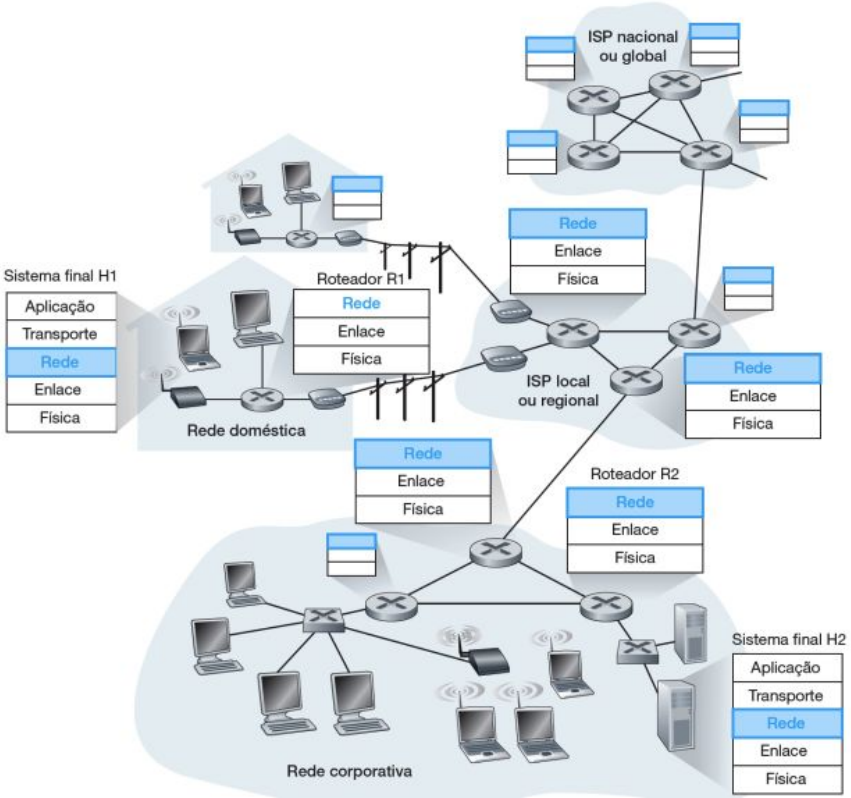
CAMADA DE REDE



CAMADA DE REDE

- Sua função é levar os pacotes da origem até o destino
  - Pode necessitar passar por vários roteadores ao longo do caminho
  - É a camada mais baixa que se preocupa com a transmissão fim a fim
- Necessita conhecer a topologia da rede (roteadores e enlaces) e escolher o caminho apropriado em redes grandes.
- Também necessita fazer a escolha do melhor caminho para evitar sobrecarregar links enquanto outros permanecem ociosos.

CAMADA DE REDE



CAMADA DE REDE NO MODELO OSI

- **Objetivo**: prover funcionalidades para modos de transmissão com e sem conexão para entidades da camada de transporte
- Estabelecer, manter e terminar conexões entre sistemas abertos (isto é, redes)
- Promover troca de unidades de dados de serviços de redes: pacotes
- Manter endereçamento entre hosts de redes distintas
- Qualidade de serviço

CAMADA DE REDE - SERVIÇOS PARA A CAMADA DE TRANSPORTE

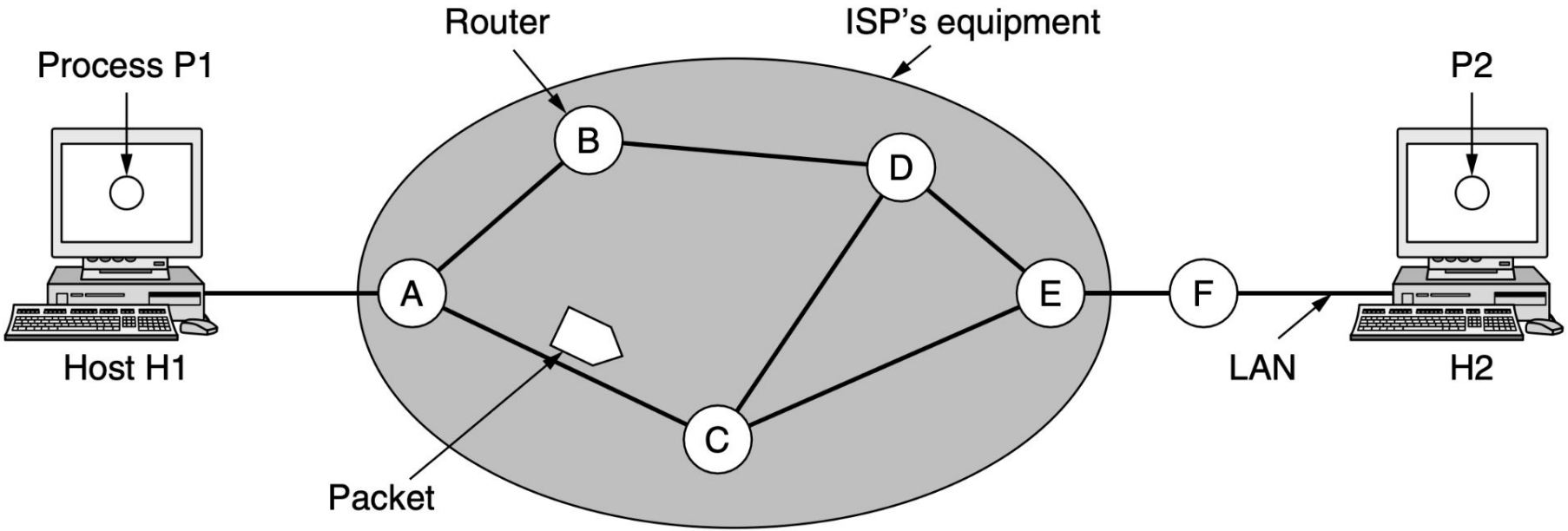
- Serviços devem ser independentes da tecnologia do roteador.
- O transporte deve desconhecer o número, tipo e topologia das redes.
- Endereços de rede devem ser disponibilizados à camada de transporte de maneira uniforme, independente do tipo de rede.

CAMADA DE REDE - FUNÇÕES

- Roteamento e repasse de pacotes
- Multiplexação de conexões de rede em um mesmo enlace de dados.
- Detecção e recuperação de erros.
- Sequenciamento de pacotes e controle de fluxo.
- Seleção de serviços.
- Gerenciamento da camada de rede.
- Mapeamento de endereços e transmissões da camada de enlace em endereços e transmissões da camada de rede.

CAMADA DE REDE - FUNÇÕES

- Store-and-forward packet switching



CAMADA DE REDE - FUNÇÕES

Três funções importantes:

- **Determinação de caminhos:** rota escolhida pelos pacotes entre a origem e o destino (algoritmos de roteamento).
- **Comutação:** mover pacotes entre as portas de entrada e de saída dos roteadores.
- **Estabelecimento de conexão:** algumas arquiteturas de rede exigem o estabelecimento de circuitos virtuais antes da transmissão de dados.

CAMADA DE REDE - ROTEAMENTO E REPASSE

- Conexões de rede são providas por entidades de rede em sistemas OSI e por sistemas abertos intermediários que provêem o serviço de repasse (forwarding).
- Sistemas abertos intermediários podem usar:
  - conexões intermediárias de rede;
  - transmissão por conexões de enlace; e
  - meios físicos distintos.
- O caminho de repasses (rota) é importante para a comunicação resultante.
  - O caminho pode considerar a qualidade de transmissão, custo financeiro e outros.

CAMADA DE REDE - SERVIÇOS

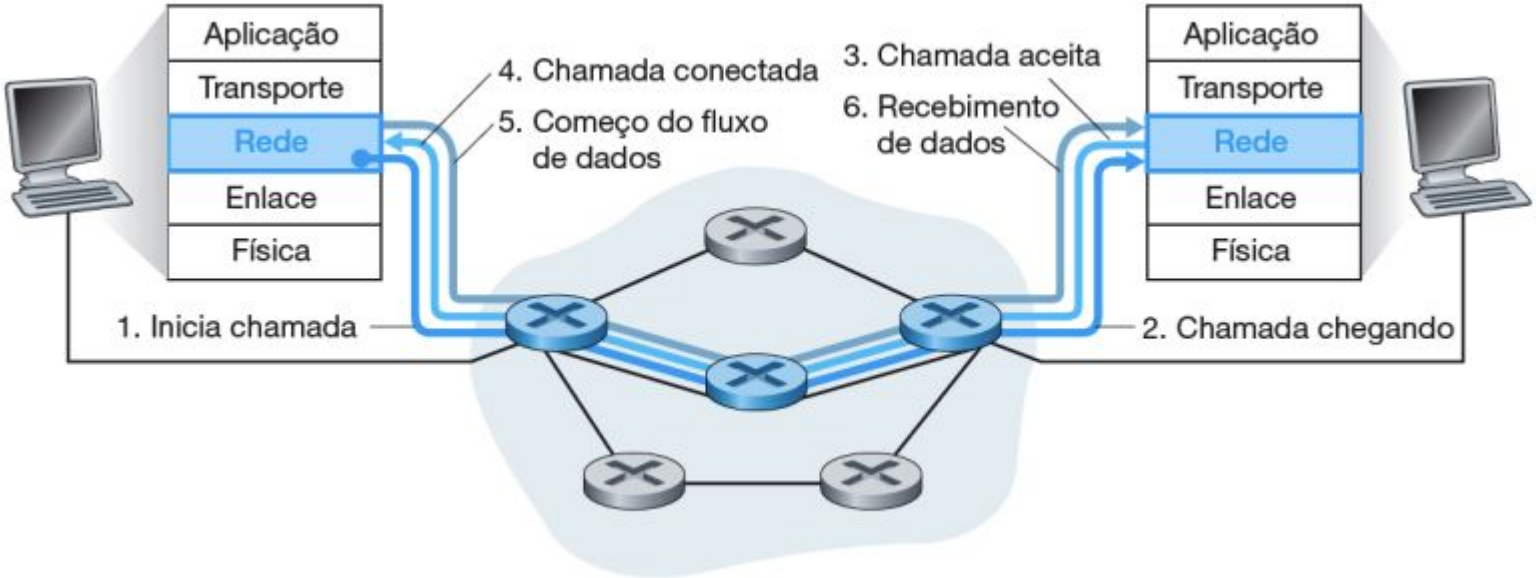
- Existem dois serviços possíveis para entregar pacotes a seus respectivos destinos:
  a. Redes de Circuitos Virtuais;
  b. Redes de Datagramas.

REDES DE CIRCUITOS VIRTUAIS

- Estabelece-se uma conexão antes do envio de dados.
- Libera-se a conexão após a troca de dados.
- Cada pacote transporta um identificador do VC, não transporta o endereço completo do destino.
- Cada roteador na rota mantém informações de estado para a conexão que passa por ele.
- **Vantagens:**
  - Orientado ao desempenho.
  - A banda passante e os recursos do roteador podem ser alocados por VC.
  - Controle de Qualidade de Serviço (QoS) por VC.

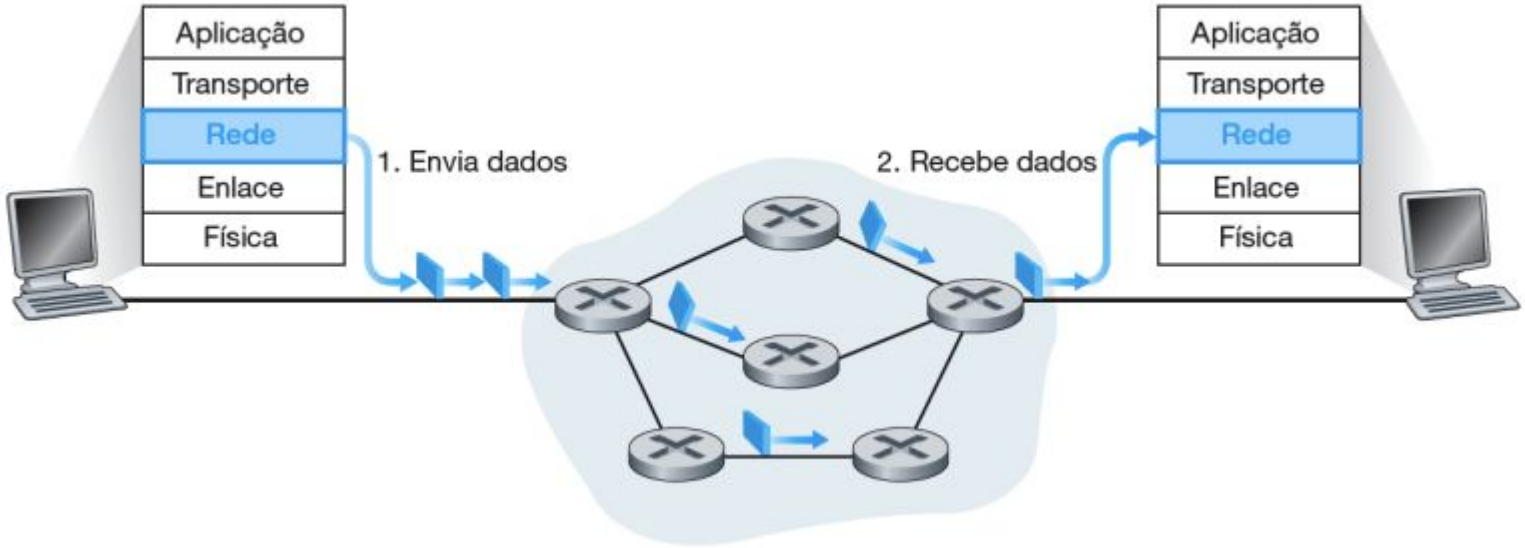
REDES DE CIRCUITOS VIRTUAIS - SINALIZAÇÃO

- Sinalização é usada para estabelecer, manter e encerrar Circuitos Virtuais.
- Usados em ATM, Frame Relay e X.25, mas não na Internet.



REDES DE DATAGRAMAS

- Não estabelece conexões.
- Não há informação de estado de conexão nos roteadores.
- Pacotes tipicamente transportam o endereço de destino.
- Pacotes para o mesmo destino podem seguir diferentes rotas.



REDES DE DATAGRAMAS X CIRCUITOS VIRTUAIS

Redes de Datagrama (Internet):

- **Dados trocados entre computadores:**
  - Serviço elástico.
  - Requisitos de atraso não críticos.
- **Sistemas finais inteligentes:**
  - Podem adaptar-se, realizar controle e recuperação de erros.
  - A rede é simples.
  - Complexidade nos sistemas finais.
- **Muitos tipos de enlaces:**
  - Características diferentes.
  - Difícil obter um serviço uniforme.

Redes de Circuito Virtual (ATM):

- **Originário da telefonia:**
  - Ótimo para conversação humana.
  - Tempos estritos, exigências de confiabilidade.
  - Necessário para serviço garantido.
- **Sistemas finais mais simples:**
  - Telefones.
  - Complexidade dentro da rede.

REDES DE DATAGRAMAS X CIRCUITOS VIRTUAIS

- **Circuitos (telefonia):** X.25, Frame Relay, ATM.
- **Datagramas (Arpanet):** Internet Protocol (IP).
- **Década de 1990:** embate entre ATM vs. IP.
  a. Bellheads (ITU) vs. Netheads (IETF).
  b. **Bellheads:** telefonia.
  c. **Netheads:** Internet.
  d. Artigo sobre o embate entre ATM e IP.
- **Fracasso do ATM em manter QoS em ambientes reais.**
- **Domínio do IP.**
- **Porém, atual crescimento de tecnologias orientadas a conexão:**
  - MPLS e VLAN.

REDES DE DATAGRAMAS X CIRCUITOS VIRTUAIS

Função	Com conexão	Sem conexão
Configuração do caminho	sim	desnecessário
Endereçamento	número VC	end. destino
Informação de estado	manutenção de circuitos	sem info. de transmissões
Roteamento	criar circuito	pacotes são independentes
Qualidade de serviço	sim	difícil
Controle de congestionamento	sim	difícil
Efeito de falhas	circuitos falham	datagrama perdido

- **Com conexões ou Rede de Circuitos Virtuais:** X.25, Frame Relay, ATM, MPLS.
- **Sem conexões ou Rede de Datagramas:** IP.

REPASSE (*FORWARDING*) E ROTEAMENTO

Repasse é o envio para a próxima rede.

Roteamento envolve o preenchimento de tabelas para a escolha de qual é a melhor próxima rede para chegar ao destino final.

- Exemplos: RIP, OSPF, BGP.

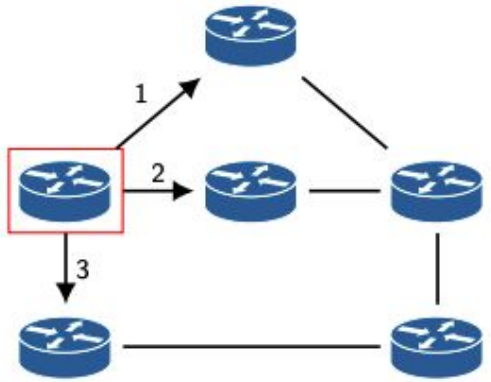
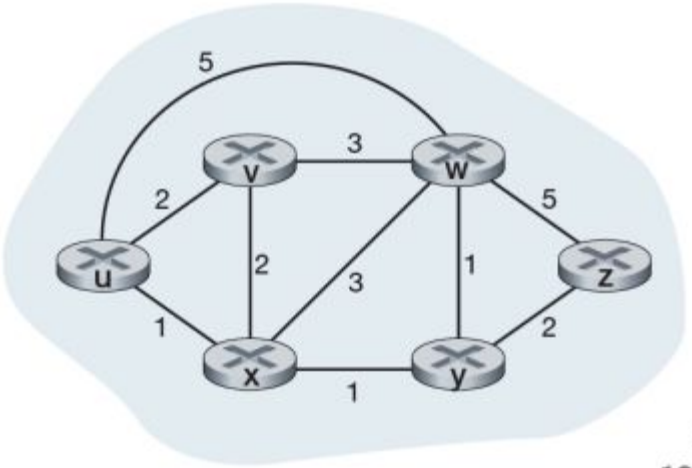


tabela de repasse local

cabeçalho do pacote	enlace saída
0100	3
0101	2
0111	2
1001	1

ROTEAMENTO

- **Determinar “bons” caminhos** (sequência de roteadores) através da rede da fonte até o destino.
- Algoritmos de roteamento são descritos por grafos.
  - Os nós do grafo são roteadores.
  - As arestas do grafo são enlaces.
  - Custo do enlace: atraso, preço ou nível de congestionamento.
- “Bons” caminhos:
  - Caminhos de menor custo.
  - Caminhos redundantes.



ROTEAMENTO - REQUISITOS

- **Requisitos:**
- **Corretude e Simplicidade:** Garantir que o sistema funcione corretamente e seja simples de entender e implementar.
- **Robustez:** Execução por longo tempo, mesmo se houver falhas.
- **Estabilidade:** Escolha rápida de rotas que não mudam com frequência.
- **Equidade (fairness):**
  - **Exemplo:** reduzir atraso médio de pacotes.
- **Eficiência:**
  - **Exemplo:** maximizar transmissão de saída.

CLASSIFICAÇÃO DE ALGORITMOS DE ROTEAMENTO

Informação Global:

- Todos os roteadores têm informações completas da topologia e do custo dos enlaces.
- Algoritmos de estado de enlace (“Link state”).

Informação Descentralizada:

- Roteadores só conhecem informações sobre seus vizinhos e os enlaces para chegar até eles.
- Processo de computação interativo, troca de informações com os vizinhos.
- Algoritmos de vetor de distância (“Distance vector”).

CLASSIFICAÇÃO DE ALGORITMOS DE ROTEAMENTO

Estático:

- As rotas mudam lentamente ao longo do tempo.
- Muitas vezes dependem de mudanças feitas por um administrador de rede.

Dinâmico:

- As rotas mudam mais rapidamente.
- Atualizações periódicas.
- Podem responder a mudanças no custo dos enlaces.

TIPOS DE ROTEAMENTO

- **Um sistema autônomo** é uma coleção de redes sob uma administração centralizada.
- Roteamento dentro de um sistema autônomo é realizado por um **“interior gateway routing protocol”**.
  - **Routing Information Protocol (RIP):** baseado em vetor de distâncias.
  - **Open Shortest Path First (OSPF):** baseado no estado de enlaces.
- **Roteamento entre sistemas autônomos** é realizado pelo **Border Gateway Protocol (BGP)**.

TIPOS DE ROTEAMENTO

- **Um sistema autônomo** é uma coleção de redes sob uma administração centralizada.
- Roteamento dentro de um sistema autônomo é realizado por um **“interior gateway routing protocol”**.
  - **Routing Information Protocol (RIP):** baseado em vetor de distâncias.
  - **Open Shortest Path First (OSPF):** baseado no estado de enlaces.
- **Roteamento entre sistemas autônomos** é realizado pelo **Border Gateway Protocol (BGP)**.

PRINCÍPIO DA OTIMALIDADE DE BELLMAN

- Se J está no caminho ótimo entre I e K, então o caminho entre J e K é parte do caminho ótimo entre I e K.
    - Para comprovar considere a parte da tora entre I e J r1 e o restante da rota r2. Se uma rota melhor existir de J até K ela deve ser concatenada a r1 para melhorar o roteamento, contradizendo a afirmação que r1r2 é o caminho ótimo.

PRINCÍPIO DA OTIMALIDADE DE BELLMAN

- O conjunto de rotas ótimas de todas as origens até um destino forma uma árvore cuja raiz é o destino. Conhecida como ***sink tree.***

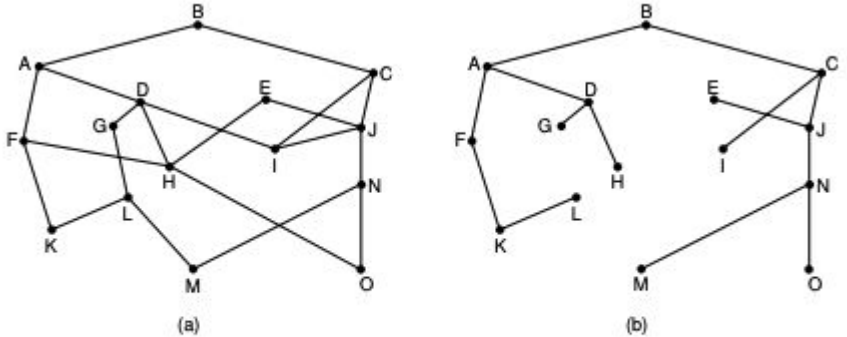
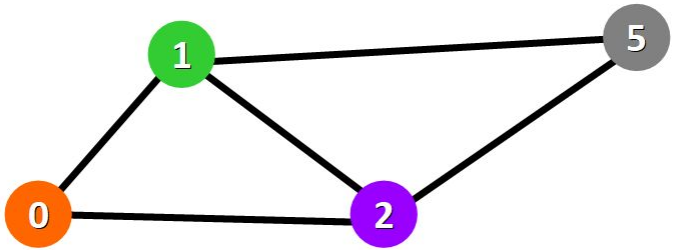


Figura: Uma rede e uma árvore de caminhos ótimos a partir do roteador B (*sink tree*).

REVISÃO: GRAFOS

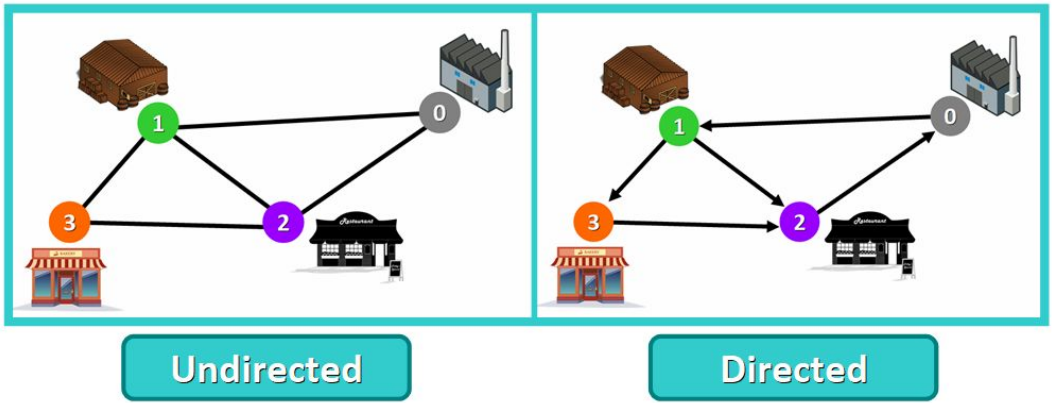
- Grafos são estruturas de dados usadas para representar "conexões" entre pares de elementos.
- Esses elementos são chamados de nós.
  - Eles representam objetos, pessoas ou entidades da vida real.
- As conexões entre nós são chamadas de arestas.
- Os nós são representados por círculos coloridos e as arestas são representadas por linhas que conectam esses círculos.



REVISÃO: TIPOS DE GRAFOS

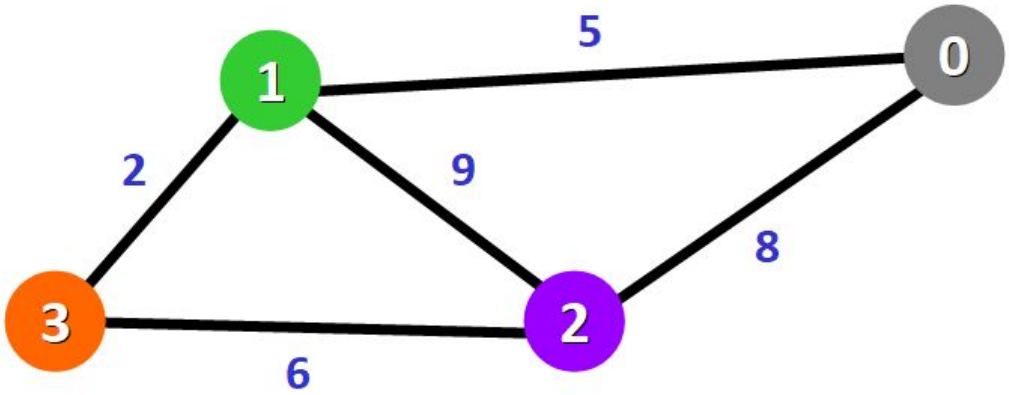
Grafos podem ser:

- **Não direcionados:** se, para cada par de nós conectados, é possível ir de um nó ao outro em ambas as direções.
- **Direcionados:** se, para cada par de nós conectados, é possível ir de um nó a outro apenas em uma direção específica. Usamos setas em vez de linhas simples para representar as arestas direcionadas.



REVISÃO: GRAFOS PONDERADOS

- Um grafo ponderado é um grafo cujas arestas têm um "peso" ou "custo". O peso de uma aresta pode representar distância, tempo ou qualquer outra coisa que modele a "conexão" entre o par de nós que ela conecta.
- Por exemplo, no grafo ponderado abaixo, você pode ver um número azul ao lado de cada aresta. Esse número é usado para representar o peso da aresta correspondente.



ALGORITMO DE DIJKSTRA

- Com o Algoritmo de Dijkstra, o caminho mais curto entre nós em um grafo é encontrado.
  - Em particular, é possível encontrar o caminho mais curto de um nó (chamado de "nó de origem") para todos os outros nós no grafo, produzindo uma árvore de caminhos mais curtos.
- Este algoritmo é usado em dispositivos de GPS para encontrar o caminho mais curto entre a localização atual e o destino. Ele tem amplas aplicações na indústria, especialmente em domínios que requerem modelagem de redes.

ALGORITMO DE DIJKSTRA

- Este algoritmo foi criado e publicado pelo Dr. Edsger W. Dijkstra, um brilhante cientista da computação e engenheiro de software holandês.
- Em 1959, ele publicou um artigo de 3 páginas intitulado "A note on two problems in connexion with graphs", onde explicou seu novo algoritmo.

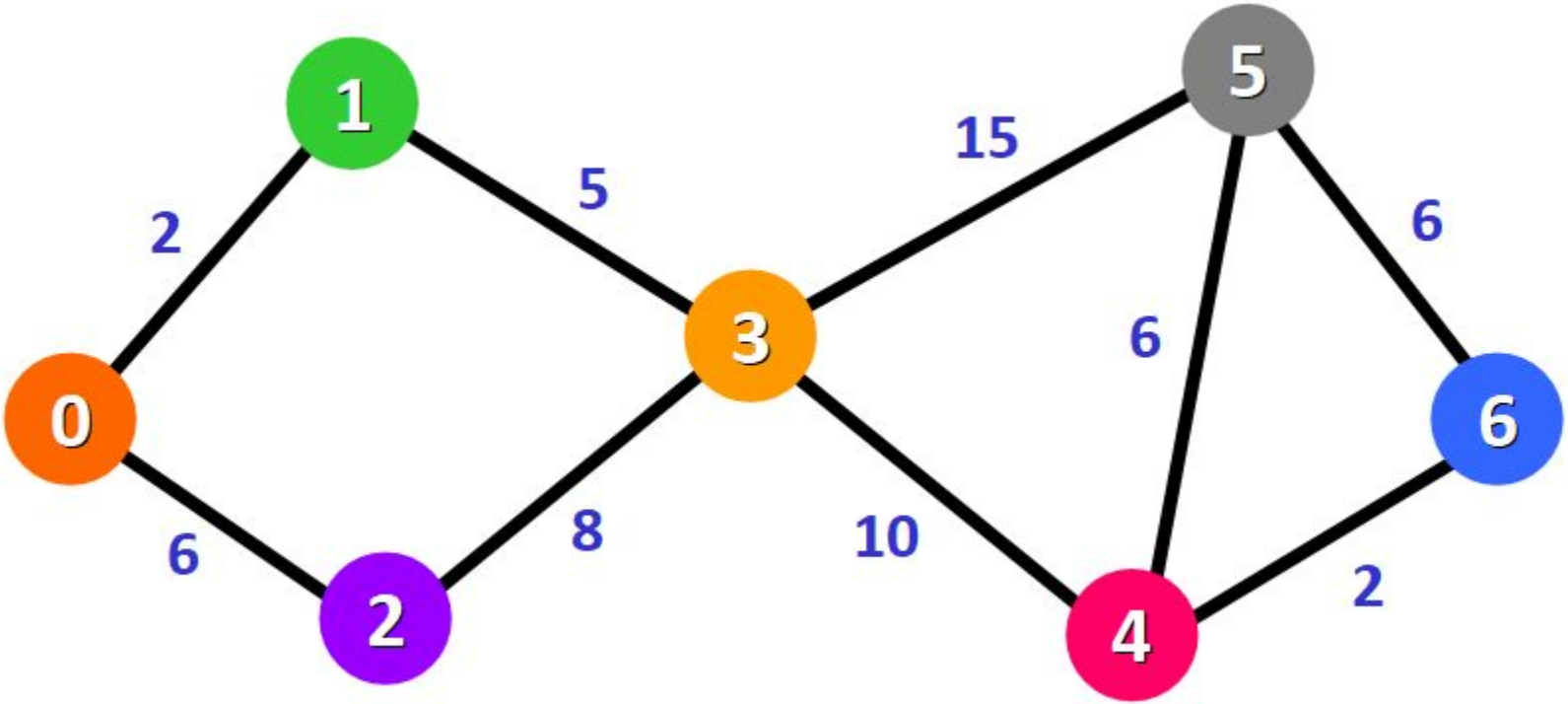
ALGORITMO DE DIJKSTRA - VISÃO GERAL

- O Algoritmo de Dijkstra basicamente começa no nó que você escolhe (o nó de origem) e analisa o grafo para encontrar o caminho mais curto entre esse nó e todos os outros nós no grafo.
- O algoritmo acompanha a distância mais curta conhecida atualmente de cada nó até o nó de origem e atualiza esses valores se encontrar um caminho mais curto.
- Uma vez que o algoritmo encontra o caminho mais curto entre o nó de origem e outro nó, esse nó é marcado como "visitado" e adicionado ao caminho.
- O processo continua até que todos os nós no grafo tenham sido adicionados ao caminho. Desta forma, obtemos um caminho que conecta o nó de origem a todos os outros nós, seguindo o caminho mais curto possível para alcançar cada nó.

ALGORITMO DE DIJKSTRA - REQUISITOS

- O Algoritmo de Dijkstra só pode funcionar com grafos que tenham pesos positivos. Isso porque, durante o processo, os pesos das arestas devem ser somados para encontrar o caminho mais curto.
- Se houver um peso negativo no grafo, o algoritmo não funcionará corretamente.
  - Uma vez que um nó é marcado como "visitado", o caminho atual até esse nó é considerado o caminho mais curto para alcançá-lo.
  - E os pesos negativos podem alterar isso se o peso total puder ser decrementado após essa etapa ter ocorrido.

ALGORITMO DE DIJKSTRA - EXEMPLO



ALGORITMO DE DIJKSTRA - EXEMPLO

- Precisamos visitar todos os nós
- Iniciamos com um vetor dos nós a visitar:

Unvisited Nodes: {0, 1, 2, 3, 4, 5, 6}

ALGORITMO DE DIJKSTRA - EXEMPLO

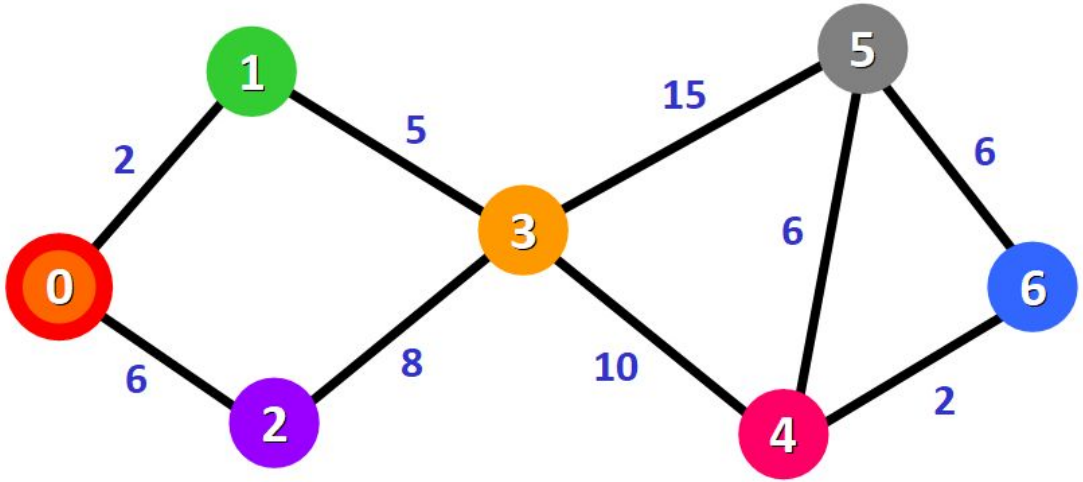
- Como iniciamos do nó 0, marcamos este nó como visitado:

Unvisited Nodes: {~~0~~, 1, 2, 3, 4, 5, 6}

ALGORITMO DE DIJKSTRA - EXEMPLO

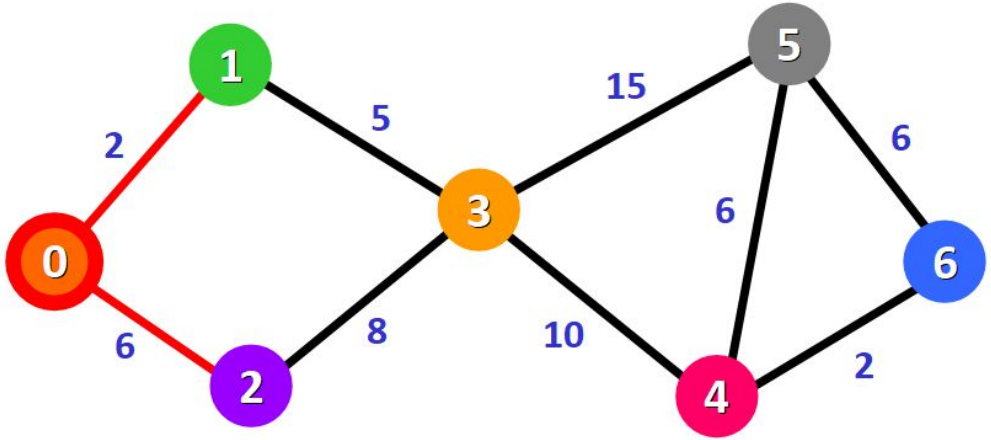
- Como iniciamos do nó 0, marcamos este nó como visitado:

Unvisited Nodes: {~~0~~, 1, 2, 3, 4, 5, 6}



ALGORITMO DE DIJKSTRA - EXEMPLO

- Agora precisamos analisar os nós adjacentes de 0, no caso 1 e 2.



Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ∞
4: ∞
5: ∞
6: ∞

ALGORITMO DE DIJKSTRA - EXEMPLO

Após atualizar as distâncias dos nós adjacentes, precisamos:

- Selecionar o nó que está mais próximo do nó de origem com base nas distâncias atualmente conhecidas.
- Marcá-lo como visitado (quadrado vermelho)
- Adicioná-lo ao caminho.

Distance:

0: 0
1: ~~∞~~ **2**
2: ~~∞~~ **6**
3: ∞
4: ∞
5: ∞
6: ∞

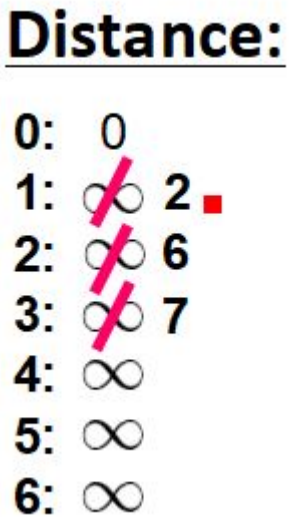
Unvisited Nodes: {~~0~~, ~~1~~, 2, 3, 4, 5, 6}

ALGORITMO DE DIJKSTRA - EXEMPLO

- Agora precisamos analisar os novos nós adjacentes para encontrar o caminho mais curto para alcançá-los. Vamos analisar apenas os nós que são adjacentes aos nós que já fazem parte do caminho mais curto (o caminho marcado com arestas vermelhas).
- O nó 3 e o nó 2 são ambos adjacentes a nós que já estão no caminho, pois estão diretamente conectados ao nó 1 e ao nó 0, respectivamente, como mostrado abaixo. Estes são os nós que vamos analisar na próxima etapa.
- Como já temos a distância do nó de origem para o nó 2 anotada em nossa lista, não precisamos atualizar a distância desta vez. Precisamos apenas atualizar a distância do nó de origem para o novo nó adjacente (nó 3)

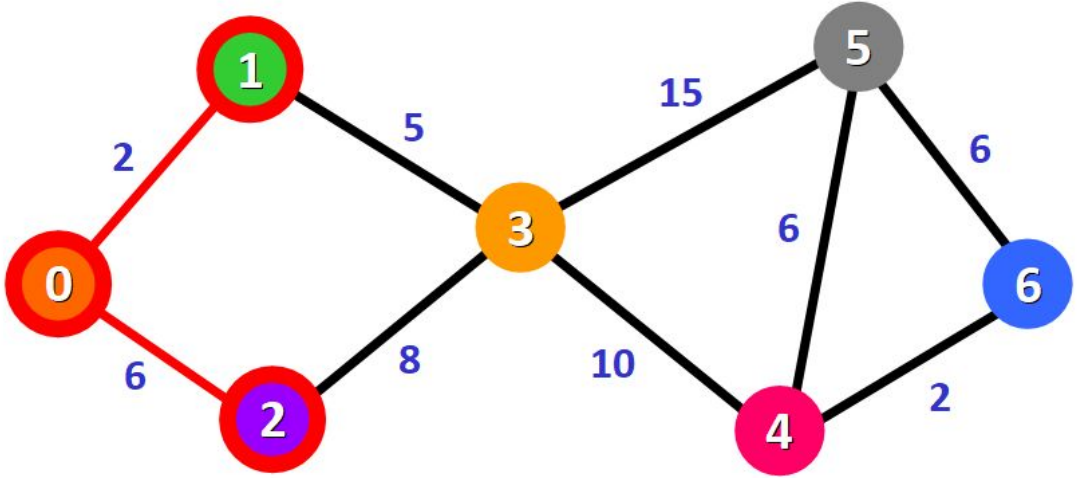
ALGORITMO DE DIJKSTRA - EXEMPLO

- Essa distância é 7. Vamos ver o porquê.
  - Para encontrar a distância do nó de origem para outro nó (neste caso, o nó 3), somamos os pesos de todas as arestas que formam o caminho mais curto para alcançar esse nó:
    - Para o nó 3: a distância total é 7 porque somamos os pesos das arestas que formam o caminho 0 -> 1 -> 3 (2 para a aresta 0 -> 1 e 5 para a aresta 1 -> 3).
-



ALGORITMO DE DIJKSTRA - EXEMPLO

- Agora que temos a distância para os nós adjacentes, precisamos escolher qual nó será adicionado ao caminho. Devemos selecionar o nó não visitado com a distância mais curta (atualmente conhecida) para o nó de origem.
  - A partir da lista de distâncias, podemos identificar imediatamente que este é o nó 2, com distância 6

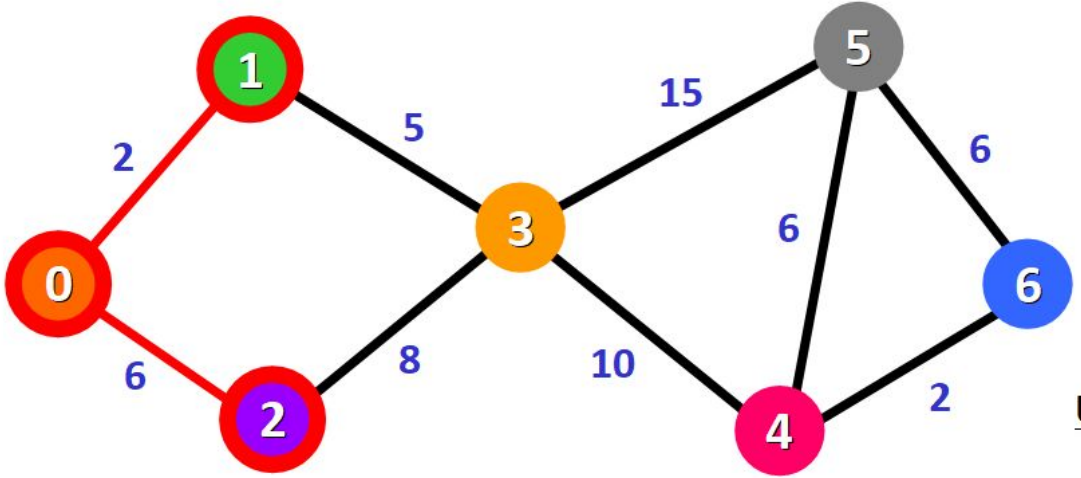


Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ∞
5: ∞
6: ∞

ALGORITMO DE DIJKSTRA - EXEMPLO

- Agora que temos a distância para os nós adjacentes, precisamos escolher qual nó será adicionado ao caminho. Devemos selecionar o nó não visitado com a distância mais curta (atualmente conhecida) para o nó de origem.
  - A partir da lista de distâncias, podemos identificar imediatamente que este é o nó 2, com distância 6



Distance:

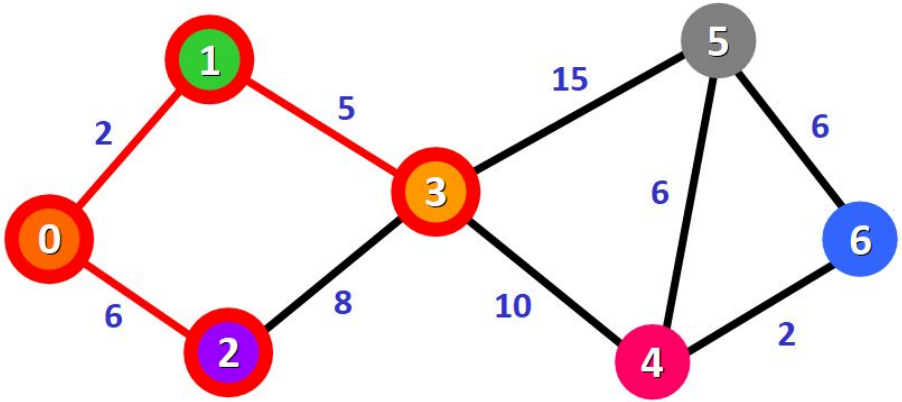
0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ∞
5: ∞
6: ∞

Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, 3, 4, 5, 6}

ALGORITMO DE DIJKSTRA - EXEMPLO

- Agora precisamos repetir o processo para encontrar o caminho mais curto do nó de origem para o novo nó adjacente, que é o nó 3.
- Você pode ver que temos dois caminhos possíveis: 0 -> 1 -> 3 ou 0 -> 2 -> 3. Vamos ver como podemos decidir qual é o caminho mais curto.

Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, 4, 5, 6}

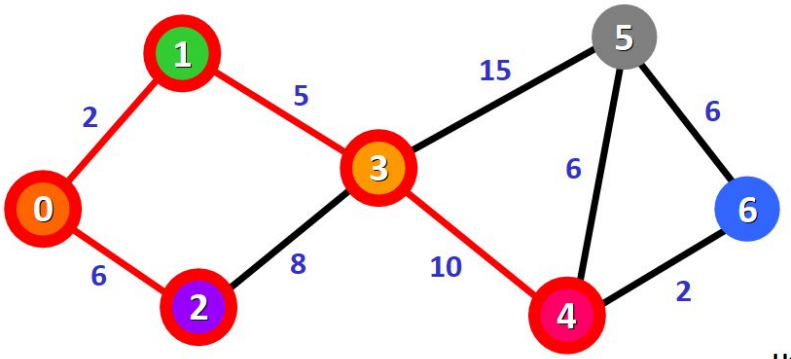


Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7 from (5 + 2) vs. 14 from (6 + 8)
4: ∞
5: ∞
6: ∞

ALGORITMO DE DIJKSTRA - EXEMPLO

- Agora repetimos o processo novamente.
- Precisamos verificar os novos nós adjacentes que ainda não visitamos. Desta vez, esses nós são o nó 4 e o nó 5, pois são adjacentes ao nó 3.
- Atualizamos as distâncias desses nós até o nó de origem, sempre tentando encontrar um caminho mais curto, se possível:
  - Para o nó 4: a distância é 17, proveniente do caminho 0 -> 1 -> 3 -> 4.
  - Para o nó 5: a distância é 22, proveniente do caminho 0 -> 1 -> 3 -> 5.



Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, 6}

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ~~∞~~ 17 from (2 + 5 + 10)
5: ~~∞~~ 22 from (2 + 5 + 15)
6: ∞

ALGORITMO DE DIJKSTRA - EXEMPLO

- E repetimos o processo novamente. Verificamos os nós adjacentes: nó 5 e nó 6. Precisamos analisar cada caminho possível que podemos seguir para alcançá-los a partir dos nós que já foram marcados como visitados e adicionados ao caminho.
- Para o nó 5:
  - A primeira opção é seguir o caminho 0 -> 1 -> 3 -> 5, que tem uma distância de 22 do nó de origem (2 + 5 + 15). Essa distância já foi registrada na lista de distâncias em uma etapa anterior.
    A segunda opção seria seguir o caminho 0 -> 1 -> 3 -> 4 -> 5, que tem uma distância de 23 do nó de origem (2 + 5 + 10 + 6).
    Claramente, o primeiro caminho é mais curto, então o escolhemos para o nó 5.
- Para o nó 6:
  - O caminho disponível é 0 -> 1 -> 3 -> 4 -> 6, que tem uma distância de 19 do nó de origem (2 + 5 + 10 + 2).

ALGORITMO DE DIJKSTRA - EXEMPLO

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ~~∞~~ 17
5: ~~∞~~ 22 vs. 23 (2 + 5 + 10 + 6)
6: ~~∞~~ 19 from (2 + 5 + 10 + 2)

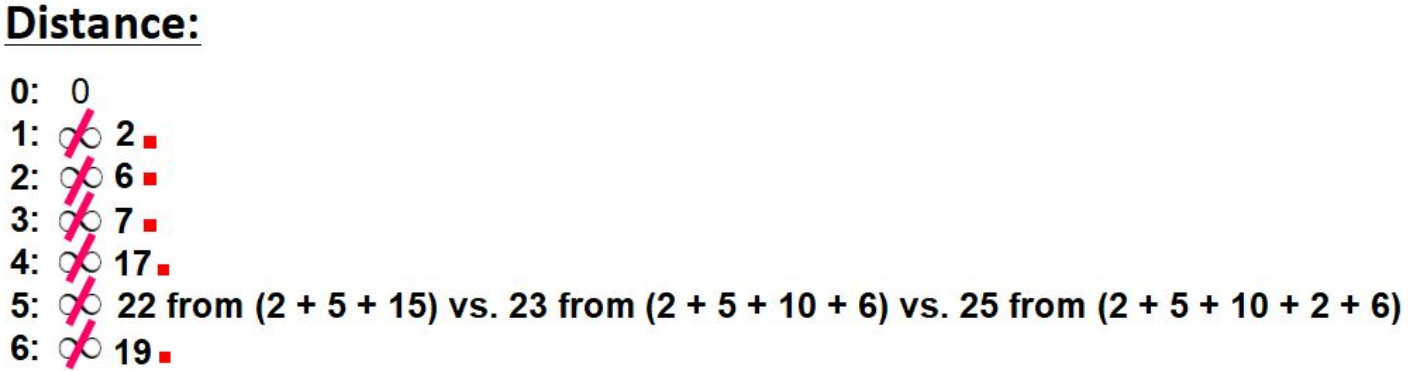
Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ~~∞~~ 17
5: ~~∞~~ 22
6: ~~∞~~ 19

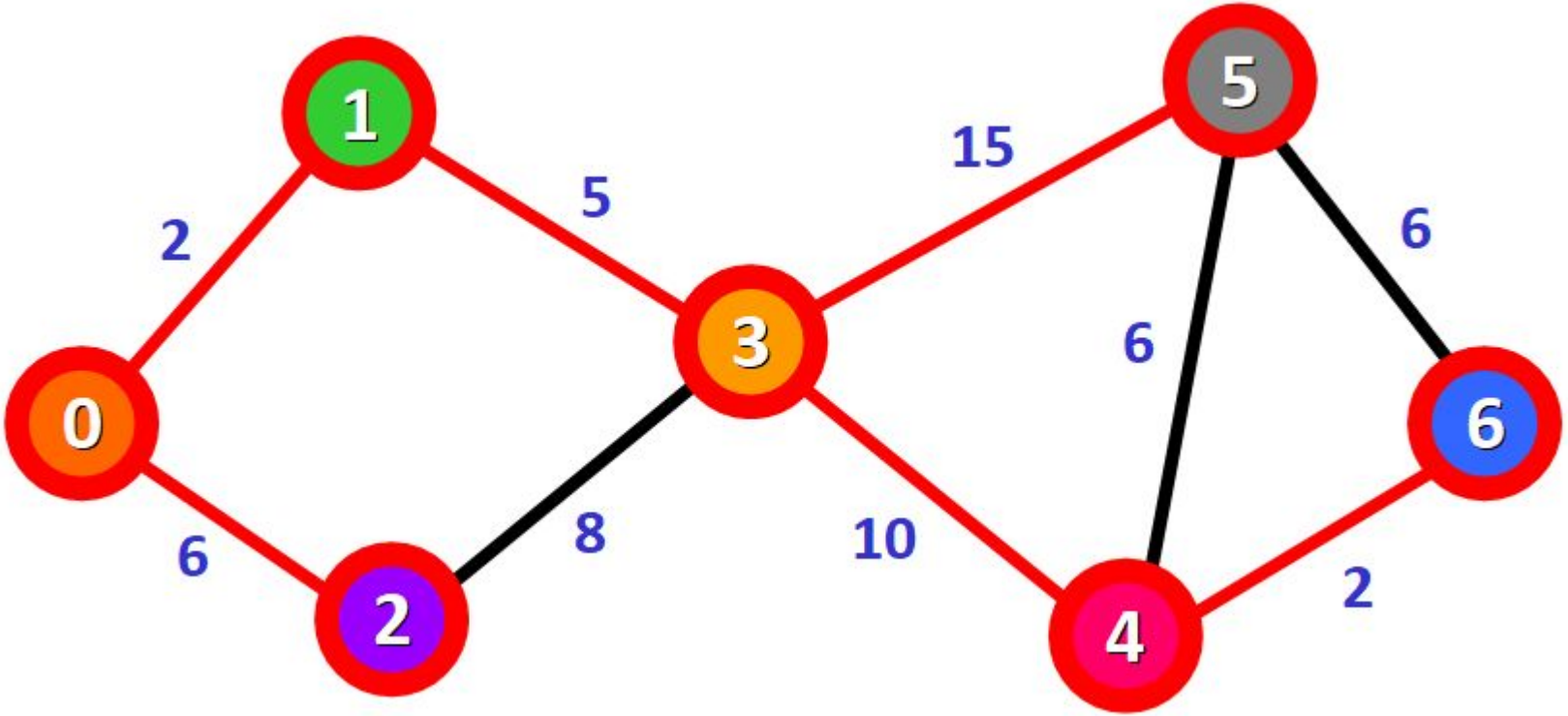
Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, ~~6~~}

ALGORITMO DE DIJKSTRA - EXEMPLO

- Apenas um nó ainda não foi visitado, o nó 5. Vamos ver como podemos incluí-lo no caminho.
- Existem três caminhos diferentes que podemos seguir para alcançar o nó 5 a partir dos nós que já foram adicionados ao caminho:
  - Opção 1: 0 → 1 → 3 → 5 com uma distância de 22 (2 + 5 + 15).
    Opção 2: 0 → 1 → 3 → 4 → 5 com uma distância de 23 (2 + 5 + 10 + 6).
    Opção 3: 0 → 1 → 3 → 4 → 6 → 5 com uma distância de 25 (2 + 5 + 10 + 2 + 6).



ALGORITMO DE DIJKSTRA - EXEMPLO



ALGORITMO DE DIJKSTRA - EXEMPLO

https://opendsa-server.cs.vt.edu/embed/DijkstraPE