GA-026 Ciência da Computação: Algoritmos I

Grafos

Programação Dinâmica

- *Programação Dinâmica* é uma técnica na qual um problema maior é solucionado a partir da resolução dos sub-problemas pelos quais ele é constituído, obedecendo-se critérios de precedência.

Programação Dinâmica

- *Programação Dinâmica* é uma técnica na qual um problema maior é solucionado a partir da resolução dos sub-problemas pelos quais ele é constituído, obedecendo-se critérios de precedência.
- É uma técnica algorítmica mais geral para solucionar diversos tipos de problemas.

Programação Dinâmica

- *Programação Dinâmica* é uma técnica na qual um problema maior é solucionado a partir da resolução dos sub-problemas pelos quais ele é constituído, obedecendo-se critérios de precedência.
- É uma técnica algorítmica mais geral para solucionar diversos tipos de problemas.
- Revendo o exemplo de caminho mínimo em um DAG.

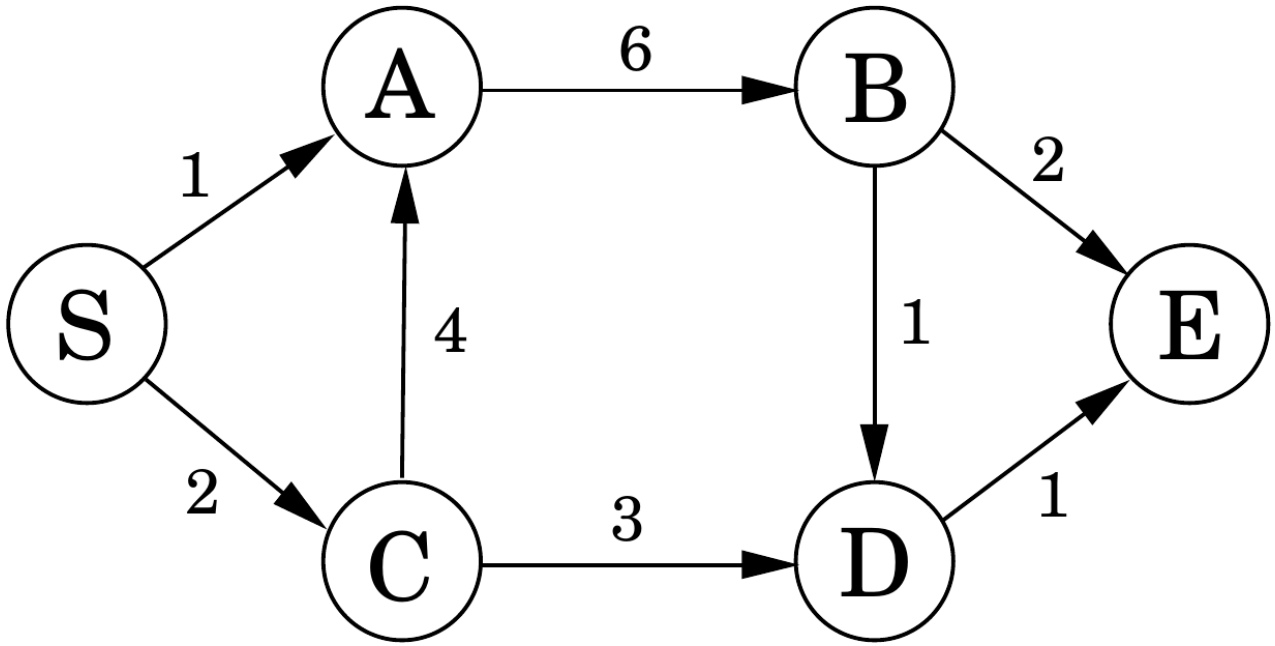
Programação Dinâmica

- *Programação Dinâmica* é uma técnica na qual um problema maior é solucionado a partir da resolução dos sub-problemas pelos quais ele é constituído, obedecendo-se critérios de precedência.
- É uma técnica algorítmica mais geral para solucionar diversos tipos de problemas.
- Revendo o exemplo de caminho mínimo em um DAG.
- A característica especial que distingue um DAG é que seus nós podem ser linearizados.

Programação Dinâmica

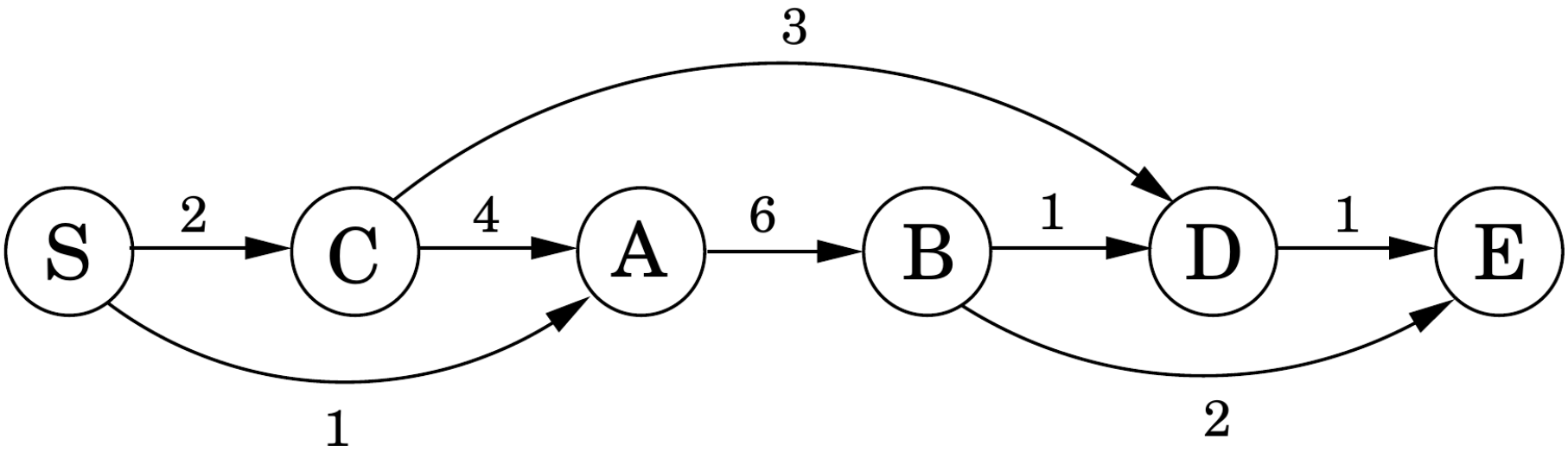
- *Programação Dinâmica* é uma técnica na qual um problema maior é solucionado a partir da resolução dos sub-problemas pelos quais ele é constituído, obedecendo-se critérios de precedência.
- É uma técnica algorítmica mais geral para solucionar diversos tipos de problemas.
- Revendo o exemplo de caminho mínimo em um DAG.
- A característica especial que distingue um DAG é que seus nós podem ser linearizados.
- Os nós podem ser arrumados em uma linha de modo que todas as arestas sigam da esquerda para a direita.

Programação Dinâmica
Caminhos mínimos em DAGs



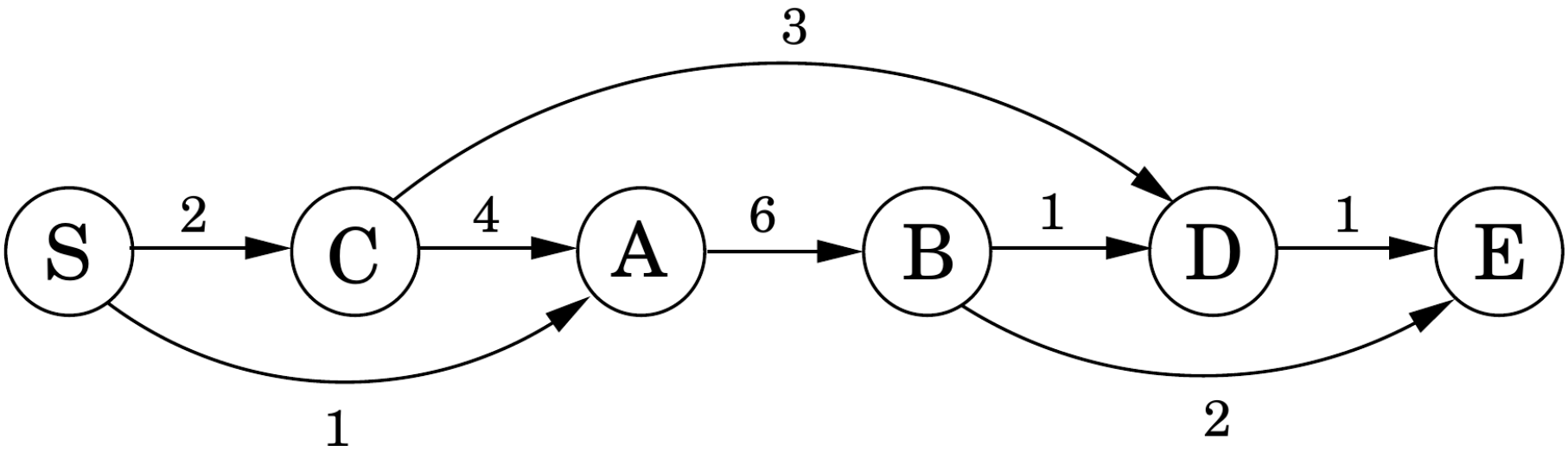
DAG

Programação Dinâmica
Caminhos mínimos em DAGs



DAG linearizado (ordenado topologicamente)

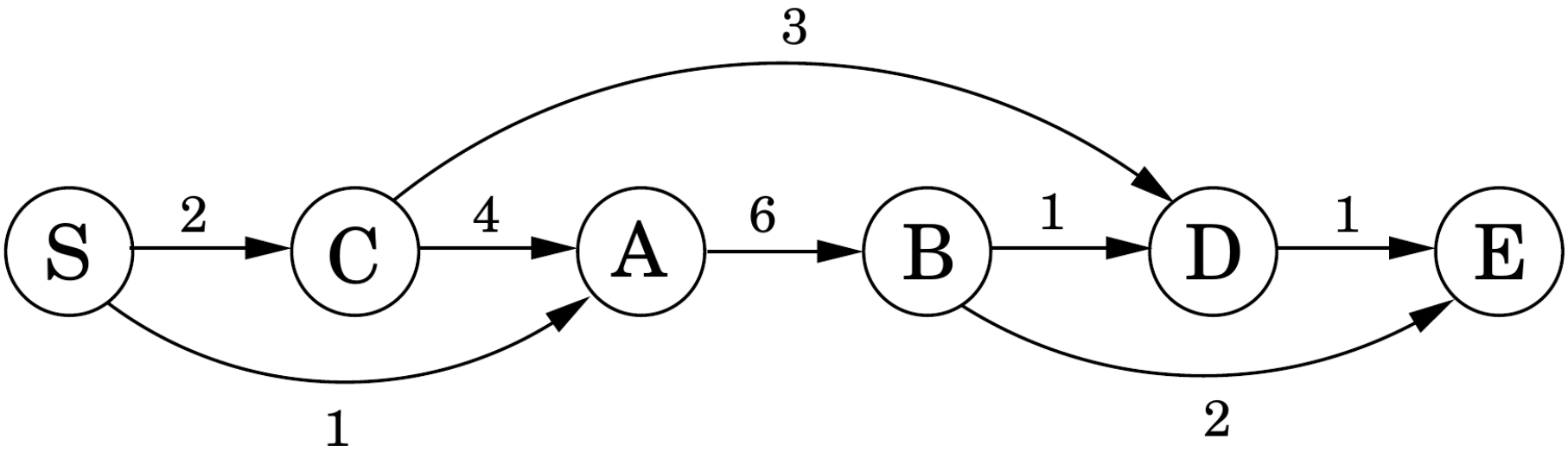
Programação Dinâmica
Caminhos mínimos em DAGs



DAG linearizado (ordenado topologicamente)

Suponha que se queira descobrir as distâncias do nó S para todos os outros nós.

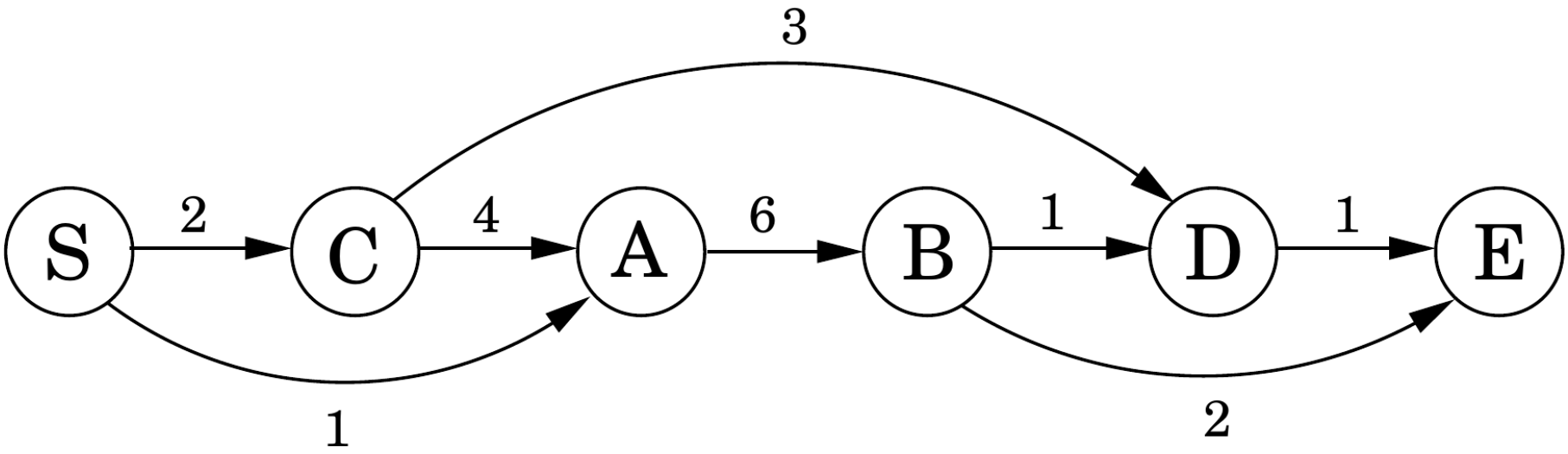
Programação Dinâmica
Caminhos mínimos em DAGs



DAG linearizado (ordenado topologicamente)

Por exemplo, para se chegar ao nó D, a única maneira de chegar a ele é pelos seus predecessores B ou C.

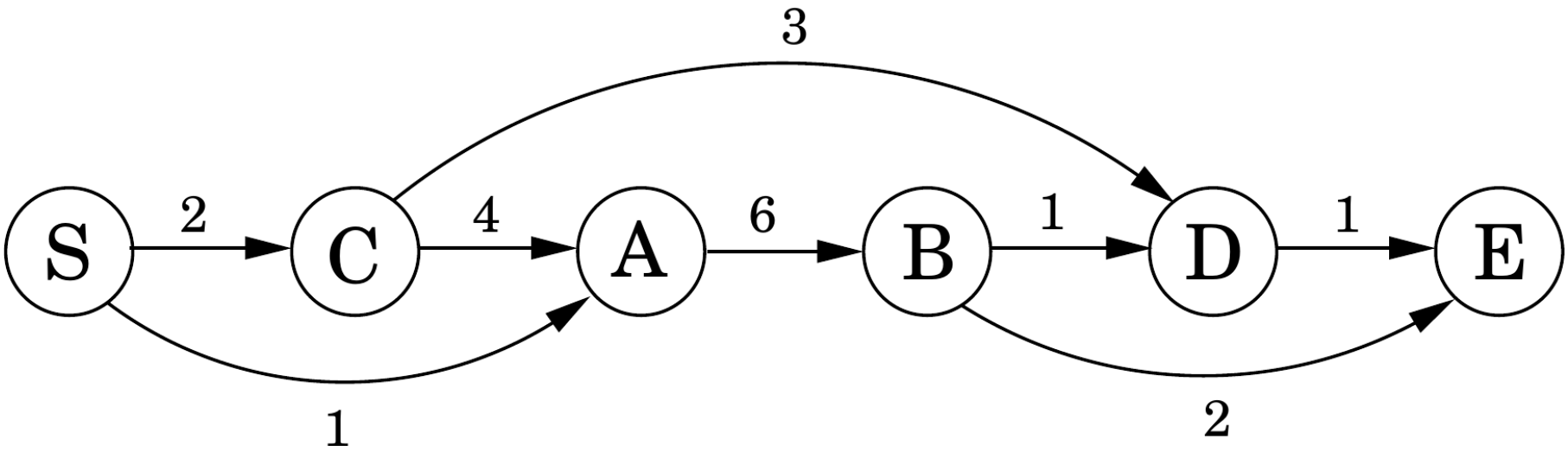
Programação Dinâmica
Caminhos mínimos em DAGs



DAG linearizado (ordenado topologicamente)

Portanto, para encontrarmos o caminho mínimo para D, precisamos apenas comparar estas duas rotas:

Programação Dinâmica
Caminhos mínimos em DAGs



DAG linearizado (ordenado topologicamente)

Portanto, para encontrarmos o caminho mínimo para D, precisamos apenas comparar estas duas rotas:

$dist(D) = \min\{dist(B) + 1, dist(C) + 3\}$

Programação Dinâmica
Caminhos mínimos em DAGs

- Uma relacão similar pode ser escrita para qualquer nó.

Programação Dinâmica
Caminhos mínimos em DAGs

- Uma relacão similar pode ser escrita para qualquer nó.
- Se for computados os valores de $dist$ na ordem da esquerda para a direita, pode-se sempre estar certo de que, quando chegar-se ao nó v, já haverá toda a informação necessária para computar $dist(v)$.

Programação Dinâmica
Caminhos mínimos em DAGs

- Uma relacão similar pode ser escrita para qualquer nó.
- Se for computados os valores de $dist$ na ordem da esquerda para a direita, pode-se sempre estar certo de que, quando chegar-se ao nó v, já haverá toda a informação necessária para computar $dist(v)$.
- Portanto, deve ser capaz de computar todas as distâncias em uma única passada:

Programação Dinâmica
Caminhos mínimos em DAGs

- Uma relacão similar pode ser escrita para qualquer nó.
- Se for computados os valores de *dist* na ordem da esquerda para a direita, pode-se sempre estar certo de que, quando chegar-se ao nó v, já haverá toda a informação necessária para computar $dist(v)$.
- Portanto, deve ser capaz de computar todas as distâncias em uma única passada:

```
inicializa todos os valores dist(·) como ∞
dist(s) = 0
para cada v∈V\{s}, em ordem linearizada:
      dist(v) = min_{(u, v) ∈ E} {dist(u) + l(u, v)}
```

Programação Dinâmica
Caminhos mínimos em DAGs

- Esse algoritmo está resolvendo uma coleção de subproblemas, $\{dist(u) : u \in V\}$

Programação Dinâmica
Caminhos mínimos em DAGs

- Esse algoritmo está resolvendo uma coleção de subproblemas, $\{dist(u) : u \in V\}$
- Começando com o menor deles, $dist(s)$, sabe-se que sua resposta é 0.

Programação Dinâmica
Caminhos mínimos em DAGs

- Esse algoritmo está resolvendo uma coleção de subproblemas, $\{dist(u) : u \in V\}$
- Começando com o menor deles, $dist(s)$, sabe-se que sua resposta é 0.
- Depois procedem-se subproblemas progressivamente "maiores", ou seja, com distâncias para vértices que estão cada vez mais longe na linearização;

Programação Dinâmica
Caminhos mínimos em DAGs

- Esse algoritmo está resolvendo uma coleção de subproblemas, $\{dist(u) : u \in V\}$
- Começando com o menor deles, $dist(s)$, sabe-se que sua resposta é 0.
- Depois procedem-se subproblemas progressivamente "maiores", ou seja, com distâncias para vértices que estão cada vez mais longe na linearização;
- Considera-se um subproblema grande quando temos de resolver muitos outros subproblemas antes de poder chegar a ele.

Programação Dinâmica
Caminhos mínimos em DAGs

- Essa é uma técnica muito geral. Em cada nó, computamos alguma função dos valores dos predecessores do nó.

Programação Dinâmica
Caminhos mínimos em DAGs

- Essa é uma técnica muito geral. Em cada nó, computamos alguma função dos valores dos predecessores do nó.
- No exemplo dado, a função é uma soma de mínimos

Programação Dinâmica
Caminhos mínimos em DAGs

- Essa é uma técnica muito geral. Em cada nó, computamos alguma função dos valores dos predecessores do nó.
- No exemplo dado, a função é uma soma de mínimos
- Programação dinâmica é um paradigma algorítmico muito poderoso no qual um problema é resolvido identificando-se uma coleção de subproblemas e lidando com eles um por um, primeiro os menores.

Programação Dinâmica
Caminhos mínimos em DAGs

- Essa é uma técnica muito geral. Em cada nó, computamos alguma função dos valores dos predecessores do nó.
- No exemplo dado, a função é uma soma de mínimos
- Programação dinâmica é um paradigma algorítmico muito poderoso no qual um problema é resolvido identificando-se uma coleção de subproblemas e lidando com eles um por um, primeiro os menores.
- Usam-se as respostas aos problemas menores para ajudar a descobrir as respostas aos maiores, até que toda a coleção esteja solucionada.

Programação Dinâmica
Caminhos mínimos em DAGs

- Em programação dinâmica o DAG está ***implícito***.

Programação Dinâmica
Caminhos mínimos em DAGs

- Em programação dinâmica o DAG está ***implícito***.
- Seus nós são os subproblemas que definimos e suas arestas são as dependências entre subproblemas.

Programação Dinâmica
Problema de subsequência mais longa

- No problema da subsequência crescente mais longa, a entrada é uma seqüência de números $a_1, ..., a_n$.

Programação Dinâmica
Problema de subsequência mais longa

- No problema da subsequência crescente mais longa, a entrada é uma seqüência de números $a_1, ..., a_n$.
- Uma subsequência é qualquer subconjunto desses números tomados em ordem, da forma, $a_{i1}, a_{i2}, ..., a_{ik}$ onde $1 \leq i_1 < i_2 < \cdots < i_k \leq n$.

Programação Dinâmica
Problema de subsequência mais longa

- No problema da subsequência crescente mais longa, a entrada é uma seqüência de números $a_1, ..., a_n$.
- Uma subsequência é qualquer subconjunto desses números tomados em ordem, da forma, $a_{i1}, a_{i2}, ..., a_{ik}$ onde $1 \leq i_1 < i_2 < \cdots < i_k \leq n$.
- Uma subsequência crescente é aquela na qual os números vão ficando estritamente maiores.

Programação Dinâmica
Problema de subsequência mais longa

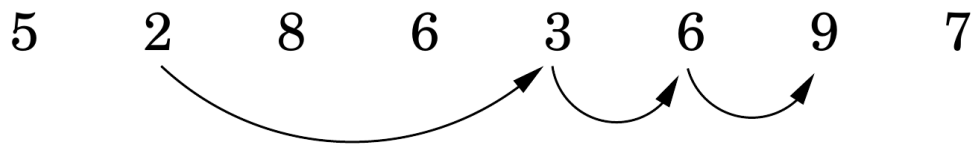
- No problema da subsequência crescente mais longa, a entrada é uma seqüência de números $a_1, ..., a_n$.
- Uma subsequência é qualquer subconjunto desses números tomados em ordem, da forma, $a_{i1}, a_{i2}, ..., a_{ik}$ onde $1 \leq i_1 < i_2 < \cdots < i_k \leq n$.
- Uma subsequência crescente é aquela na qual os números vão ficando estritamente maiores.
- A tarefa é encontrar a subsequência crescente de maior comprimento.

Programação Dinâmica
Problema de subsequência mais longa

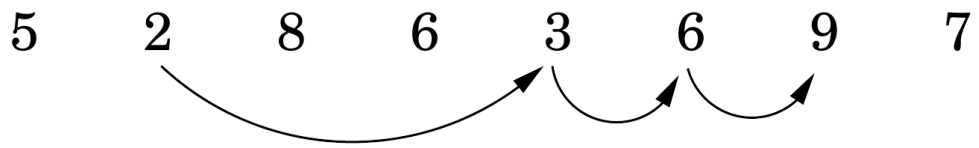
- No problema da subsequência crescente mais longa, a entrada é uma seqüência de números $a_1, ..., a_n$.
- Uma subsequência é qualquer subconjunto desses números tomados em ordem, da forma, $a_{i1}, a_{i2}, ..., a_{ik}$ onde $1 \leq i_1 < i_2 < \cdots < i_k \leq n$.
- Uma subsequência crescente é aquela na qual os números vão ficando estritamente maiores.
- A tarefa é encontrar a subsequência crescente de maior comprimento.
- Por exemplo, a subsequência crescente mais longa de $\{5, 2, 8, 6, 3, 6, 9, 7\}$ é $\{2, 3, 6, 9\}$:

Programação Dinâmica
Problema de subsequência mais longa

- No problema da subsequência crescente mais longa, a entrada é uma seqüência de números $a_1, ..., a_n$.
- Uma subsequência é qualquer subconjunto desses números tomados em ordem, da forma, $a_{i1}, a_{i2}, ..., a_{ik}$ onde $1 \leq i_1 < i_2 < \cdots < i_k \leq n$.
- Uma subsequência crescente é aquela na qual os números vão ficando estritamente maiores.
- A tarefa é encontrar a subsequência crescente de maior comprimento.
- Por exemplo, a subsequência crescente mais longa de $\{5, 2, 8, 6, 3, 6, 9, 7\}$ é $\{2, 3, 6, 9\}$:

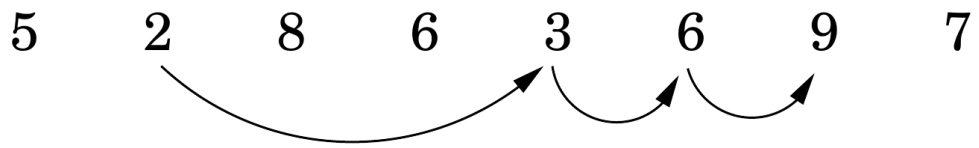


Programação Dinâmica
Problema de subsequência mais longa



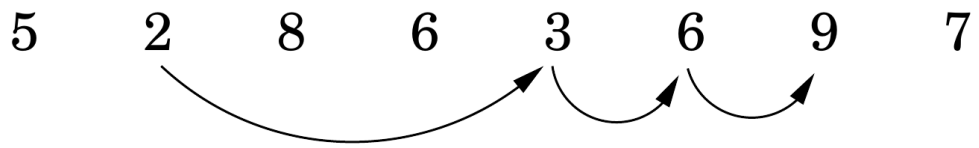
- Nesse exemplo, as setas denotam transições entre elementos consecutivos da solução ótima.

Programação Dinâmica
Problema de subsequência mais longa

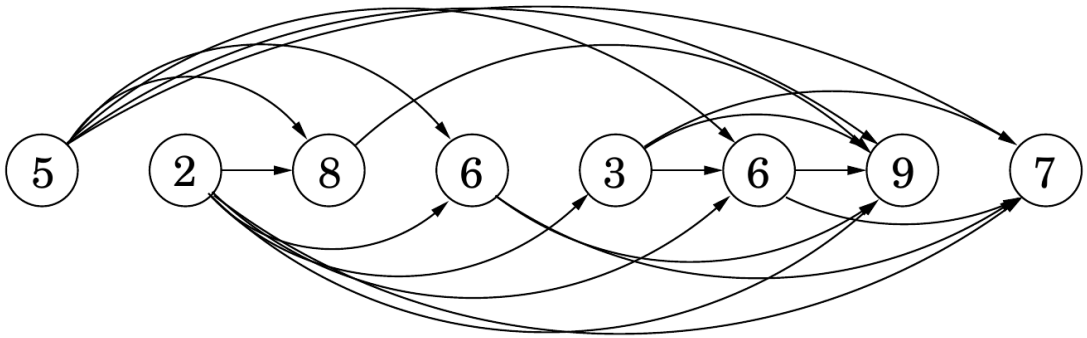


- Nesse exemplo, as setas denotam transições entre elementos consecutivos da solução ótima.
- De forma mais geral, para entender melhor o espaço de soluções, vamos criar um grafo de todas as possíveis transições:

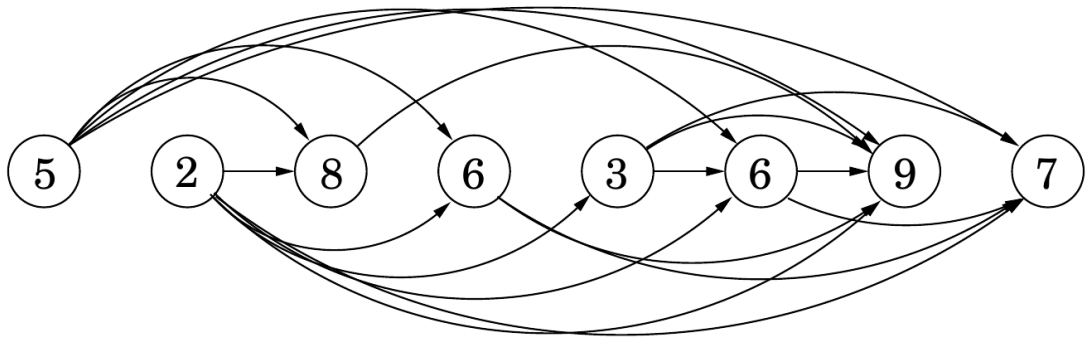
Programação Dinâmica
Problema de subsequência mais longa



- Nesse exemplo, as setas denotam transições entre elementos consecutivos da solução ótima.
- De forma mais geral, para entender melhor o espaço de soluções, vamos criar um grafo de todas as possíveis transições:

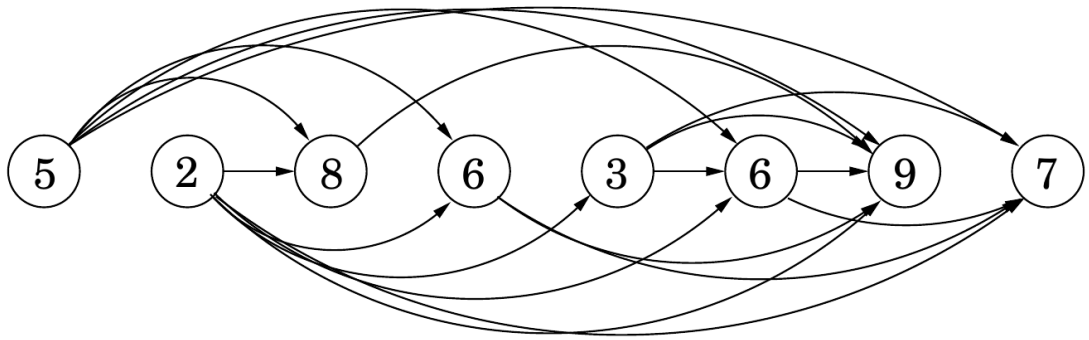


Programação Dinâmica
Problema de subsequência mais longa



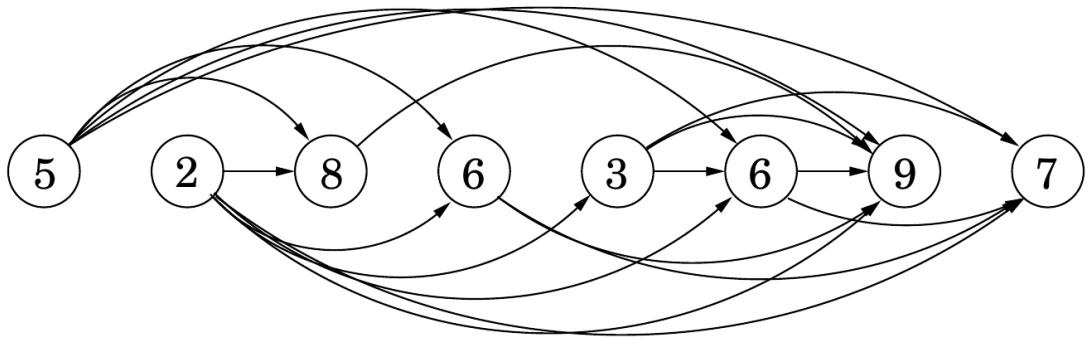
- Este grafo $G = (V, E)$ é um DAG,pois todas as arestas (i, j) têm $i < j$

Programação Dinâmica
Problema de subsequência mais longa



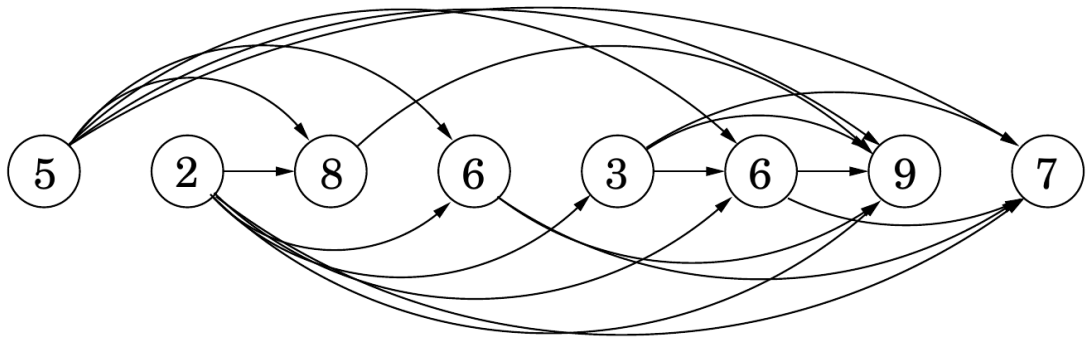
- Este grafo $G = (V, E)$ é um DAG,pois todas as arestas (i, j) têm $i < j$
- Existe uma correspondência um-para-um entre as subsequências crescentes e os caminhos nesse DAG.

Programação Dinâmica
Problema de subsequência mais longa



- Este grafo $G = (V, E)$ é um DAG,pois todas as arestas (i, j) têm $i < j$
- Existe uma correspondência um-para-um entre as subsequências crescentes e os caminhos nesse DAG.
- Portanto, nosso objetivo é simplesmente encontrar o caminho mais longo no DAG:

Programação Dinâmica
Problema de subsequência mais longa



- Este grafo $G = (V, E)$ é um DAG,pois todas as arestas (i, j) têm $i < j$
- Existe uma correspondência um-para-um entre as subsequências crescentes e os caminhos nesse DAG.
- Portanto, nosso objetivo é simplesmente encontrar o caminho mais longo no DAG:

```
para j = 1, 2, ..., n:
        L(j) = 1 + max{L(i):(i, j) ∈ E}
retornar max_j L(j)
```

Programação Dinâmica
Problema de subsequência mais longa

```
para j = 1, 2, ..., n:
        L(j) = 1 + max{L(i):(i, j) ∈ E}
retornar max_j L(j)
```

- $L(j)$ é o tamanho do caminho mais longo, isto é, a subseqüência crescente mais longa terminando em j.

Programação Dinâmica
Problema de subsequência mais longa

```
para j = 1, 2, ..., n:
        L(j) = 1 + max{L(i):(i, j) ∈ E}
retornar max_j L(j)
```

- $L(j)$ é o tamanho do caminho mais longo, isto é, a subseqüência crescente mais longa terminando em j.
- Raciocinando de maneira análoga a caminhos mínimos, qualquer caminho para o nó j tem de passar por um de seus predecessores e, portanto, $L(j)$ é 1 mais o máximo valor $L(\cdot)$ entre esses predecessores.

Programação Dinâmica
Problema de subsequência mais longa

```
para j = 1, 2, ..., n:
        L(j) = 1 + max{L(i):(i, j) ∈ E}
retornar max_j L(j)
```

- Isso é programação dinâmica: para resolvermos nosso problema original, definimos uma coleção de subproblemas $\{L(j) : 1 \leq j \leq n\}$ com a seguinte propriedade-chave que permite que eles sejam resolvidos em uma única passada:

Programação Dinâmica
Problema de subsequência mais longa

```
para j = 1, 2, ..., n:
        L(j) = 1 + max{L(i):(i, j) ∈ E}
retornar max_j L(j)
```

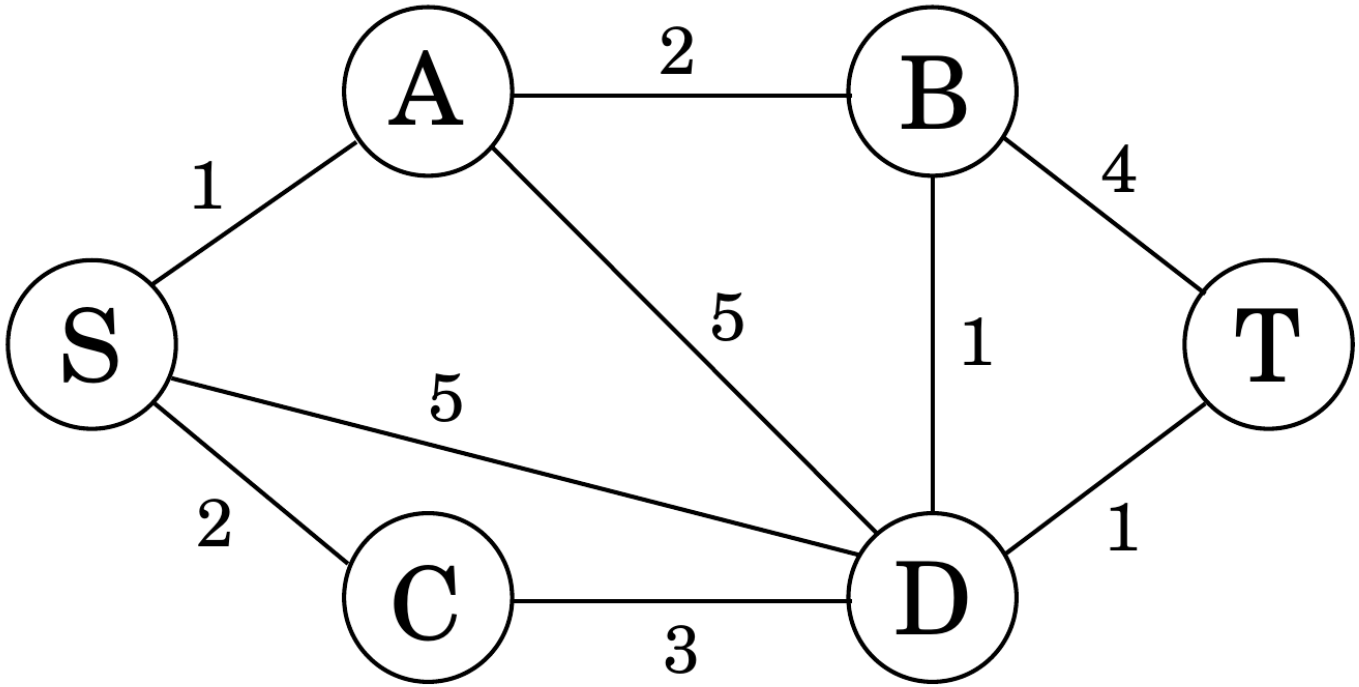
- Isso é programação dinâmica: para resolvermos nosso problema original, definimos uma coleção de subproblemas $\{L(j) : 1 \leq j \leq n\}$ com a seguinte propriedade-chave que permite que eles sejam resolvidos em uma única passada:
  - Existe uma ordenação para os subproblemas e uma relação que mostra como resolver um subproblema dadas as respostas para subproblemas "menores", ou seja, subproblemas que aparecem antes nesta ordenação.

Programação Dinâmica
Problema de subsequência mais longa

```
para j = 1, 2, ..., n:
        L(j) = 1 + max{L(i):(i, j) ∈ E}
retornar max_j L(j)
```

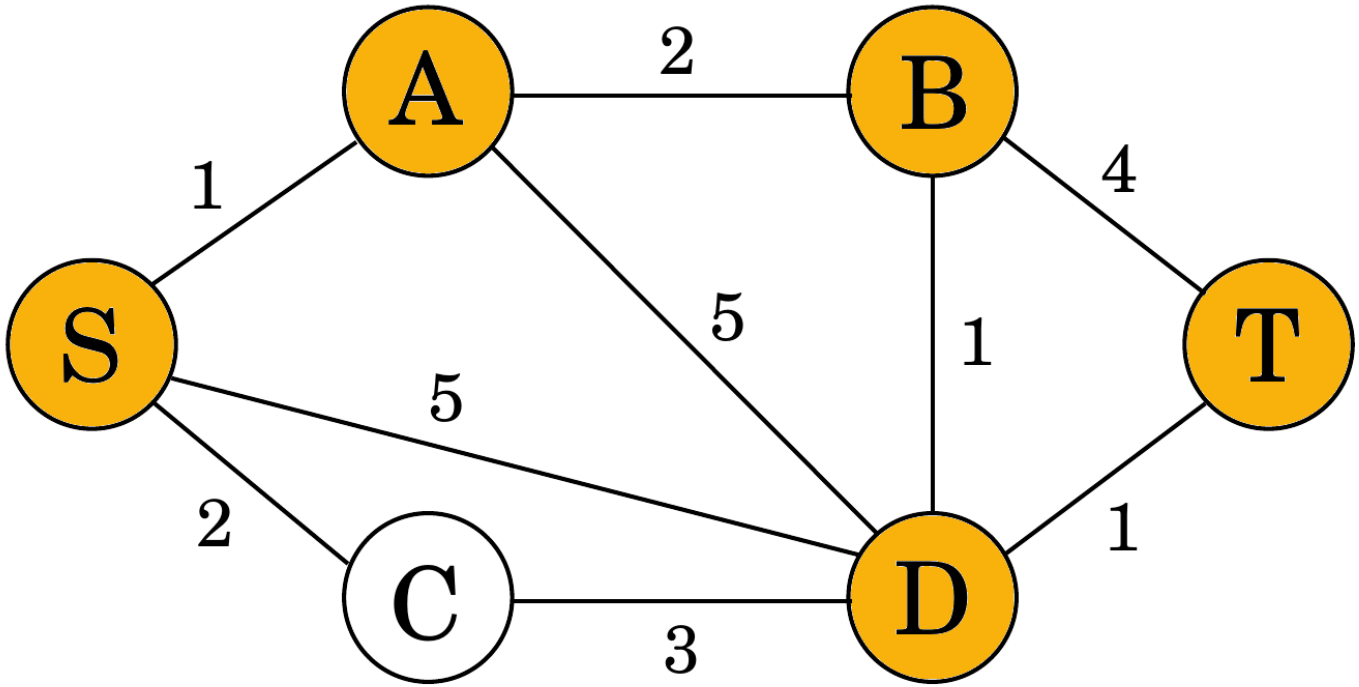
- Isso é programação dinâmica: para resolvermos nosso problema original, definimos uma coleção de subproblemas $\{L(j) : 1 \leq j \leq n\}$ com a seguinte propriedade-chave que permite que eles sejam resolvidos em uma única passada:
  - Existe uma ordenação para os subproblemas e uma relação que mostra como resolver um subproblema dadas as respostas para subproblemas "menores", ou seja, subproblemas que aparecem antes nesta ordenação.
- Neste exemplo, cada subproblema é resolvido usando uma expressão que envolve somente subproblemas menores:
  $L(j) = 1 + \max\{L(i) : (i, j) \in E\}$

Programação Dinâmica
Caminhos confiáveis mínimos



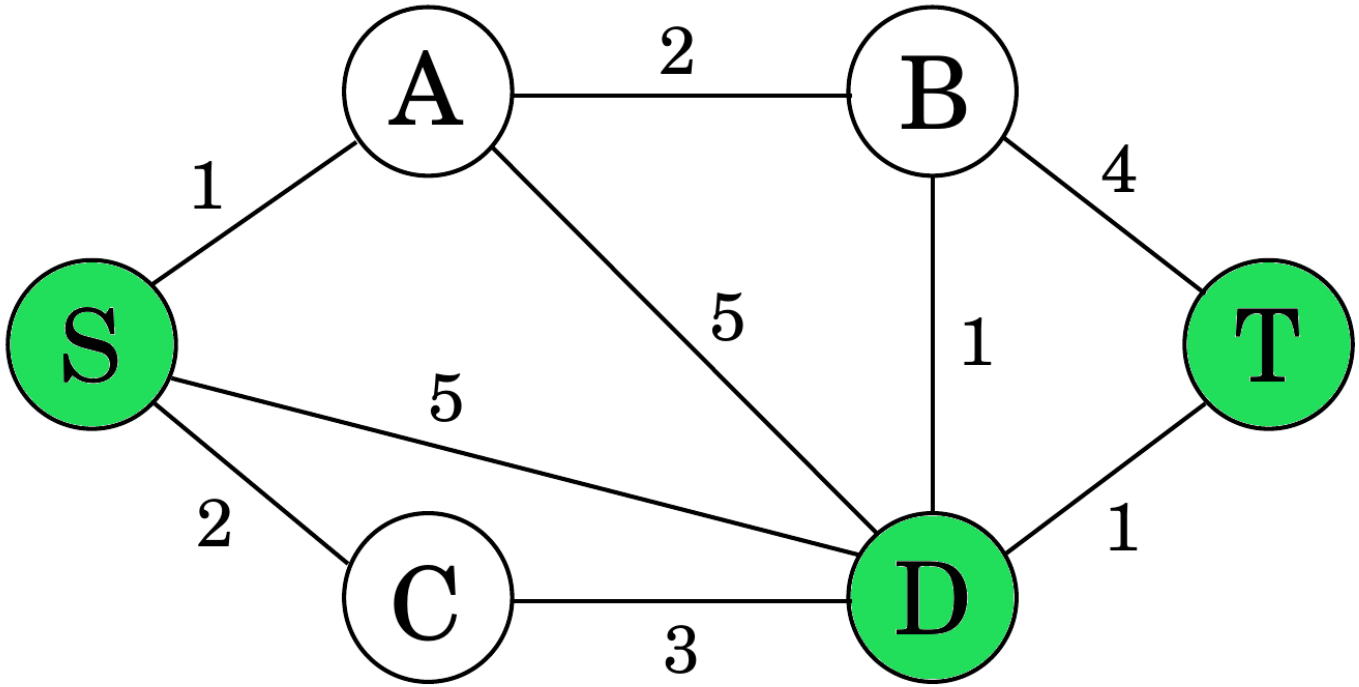
Grafo de uma rede de comunicação

Programação Dinâmica
Caminhos confiáveis mínimos



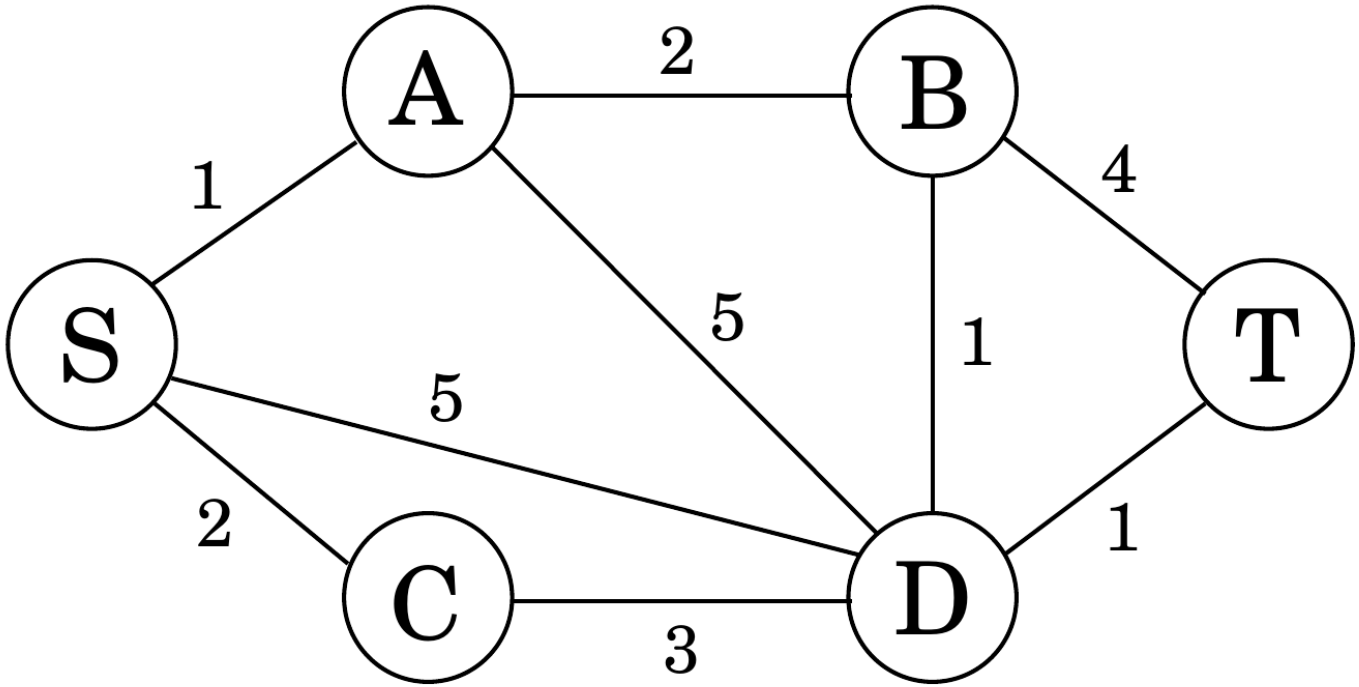
Caminho de menor distância: 4 arestas

Programação Dinâmica
Caminhos confiáveis mínimos



Caminho com menor chance de perda de dados: 2 arestas

Programação Dinâmica
Caminhos confiáveis mínimos



Escolher o caminho mínimo, com restrição no número de arestas

Programação Dinâmica
Caminhos confiáveis mínimos

- Suponha, então, que seja dado um grafo G com comprimentos nas arestas, juntamente com dois nós s e t e um inteiro k.

Programação Dinâmica
Caminhos confiáveis mínimos

- Suponha, então, que seja dado um grafo G com comprimentos nas arestas, juntamente com dois nós s e t e um inteiro k.
- Deseja-se obter o caminho mínimo de s até t que use no máximo k arestas.

Programação Dinâmica
Caminhos confiáveis mínimos

- Suponha, então, que seja dado um grafo G com comprimentos nas arestas, juntamente com dois nós s e t e um inteiro k.
- Deseja-se obter o caminho mínimo de s até t que use no máximo k arestas.
- Defini-se para cada vértice v e cada inteiro $i \leq k$, $dist(v, i)$ como o comprimento do menor caminho de s até v que usa i arestas.

Programação Dinâmica
Caminhos confiáveis mínimos

- Suponha, então, que seja dado um grafo G com comprimentos nas arestas, juntamente com dois nós s e t e um inteiro k.
- Deseja-se obter o caminho mínimo de s até t que use no máximo k arestas.
- Defini-se para cada vértice v e cada inteiro $i \leq k$, $dist(v, i)$ como o comprimento do menor caminho de s até v que usa i arestas.
- Os valores inciais de $dist(v, 0)$ é ∞ para todos os vértices exceto s, para o qual é 0.

Programação Dinâmica
Caminhos confiáveis mínimos

- Suponha, então, que seja dado um grafo G com comprimentos nas arestas, juntamente com dois nós s e t e um inteiro k.
- Deseja-se obter o caminho mínimo de s até t que use no máximo k arestas.
- Defini-se para cada vértice v e cada inteiro $i \leq k$, $dist(v, i)$ como o comprimento do menor caminho de s até v que usa i arestas.
- Os valores inciais de $dist(v, 0)$ é ∞ para todos os vértices exceto s, para o qual é 0.
- A equação geral de atualização é:

$$\texttt{dist}(v, i) = \min_{(u,v) \in E} \{\texttt{dist}(u, i-1) + \ell(u, v)\}$$

Programação Dinâmica
Caminhos confiáveis mínimos para todos os pares

- Encontrar o caminho mínimo não apenas entre s e t, mas entre todos os pares de vértices.

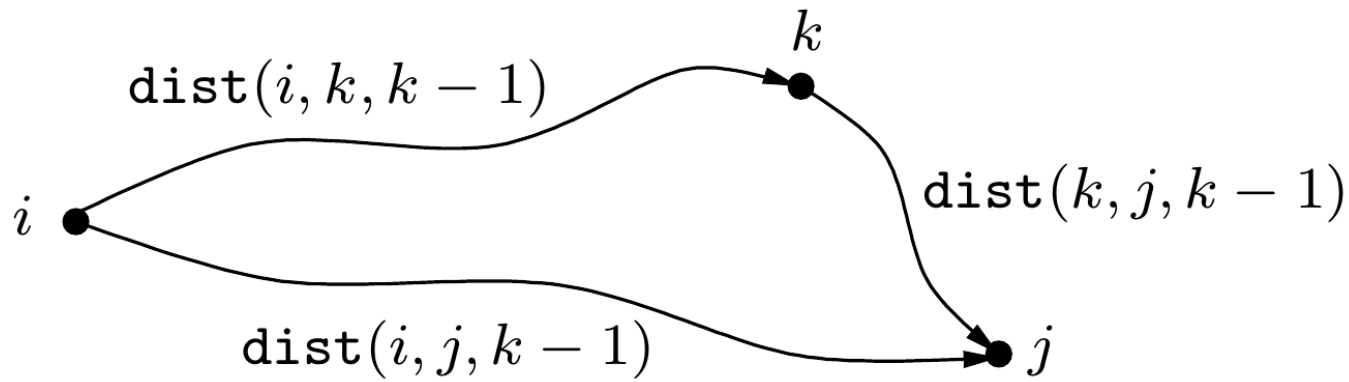
Programação Dinâmica
Caminhos confiáveis mínimos para todos os pares

- Encontrar o caminho mínimo não apenas entre s e t, mas entre todos os pares de vértices.
- Enumeram-se os vértices em V como $\{1, 2, ..., n\}$, e denota por $dist(i, j, k)$ o comprimento do caminho mínimo de i até j no qual apenas os nós $\{1, 2, ..., k\}$ podem ser usados como nós intermediários.

Programação Dinâmica
Caminhos confiáveis mínimos para todos os pares

- Encontrar o caminho mínimo não apenas entre s e t, mas entre todos os pares de vértices.
- Enumeram-se os vértices em V como $\{1, 2, ..., n\}$, e denota por $dist(i, j, k)$ o comprimento do caminho mínimo de i até j no qual apenas os nós $\{1, 2, ..., k\}$ podem ser usados como nós intermediários.
- Inicialmente, $dist(i, j, 0)$ é o comprimento entre a aresta direcionada entre i e j, se ela existe, e é ∞ , caso contrário.

Programação Dinâmica
Caminhos confiáveis mínimos para todos os pares



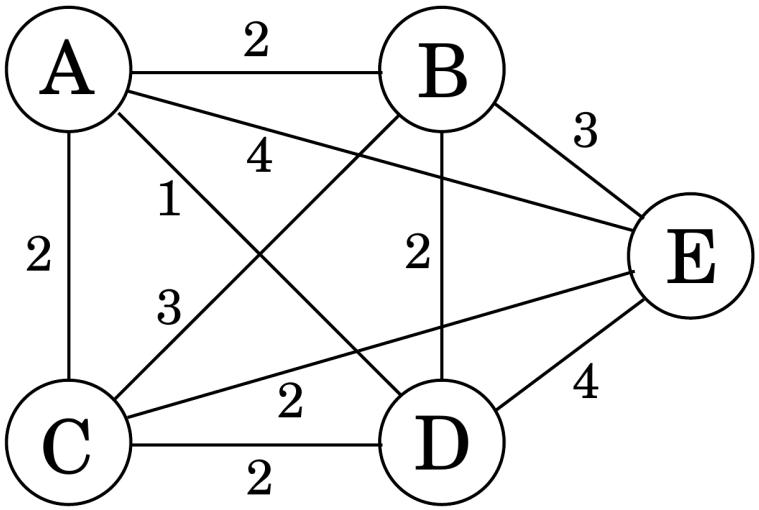
- Usar k nos dá um caminho menor de i até j se e somente se

$$\texttt{dist}(i,k,k-1) + \texttt{dist}(k,j,k-1) \;<\; \texttt{dist}(i,j,k-1)$$

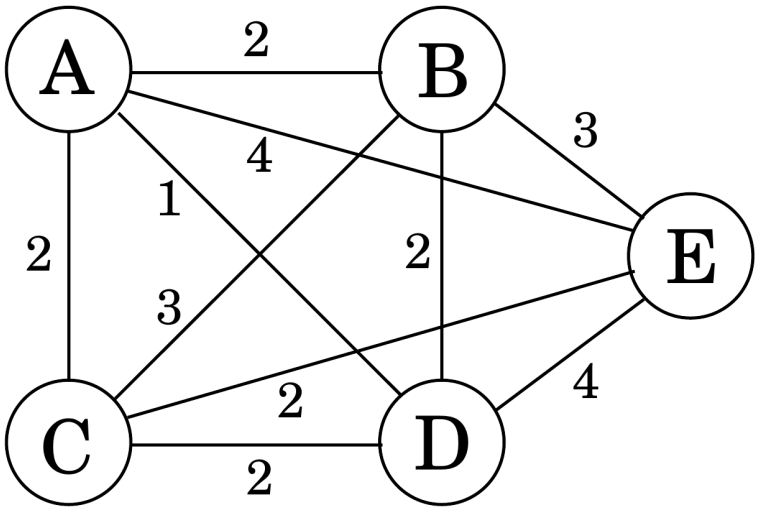
Programação Dinâmica
Algoritmo de Floyd-Warshall

```
para i = 1 até n:
  para j = 1 até n:
    dist(i, j, 0) = ∞
para todo (i, j) ∈ E:
  dist (i, j, 0) = ℓ(i, j)
para k = 1 até n:
  para i = 1 até n:
    para j = 1 até n:
      dist(i, j, k) = min{dist(i, k, k − 1) + dist(k, j, k − 1), dist(i, j, k − 1)}
```

Programação Dinâmica
O Problema do Caixeiro-Viajante

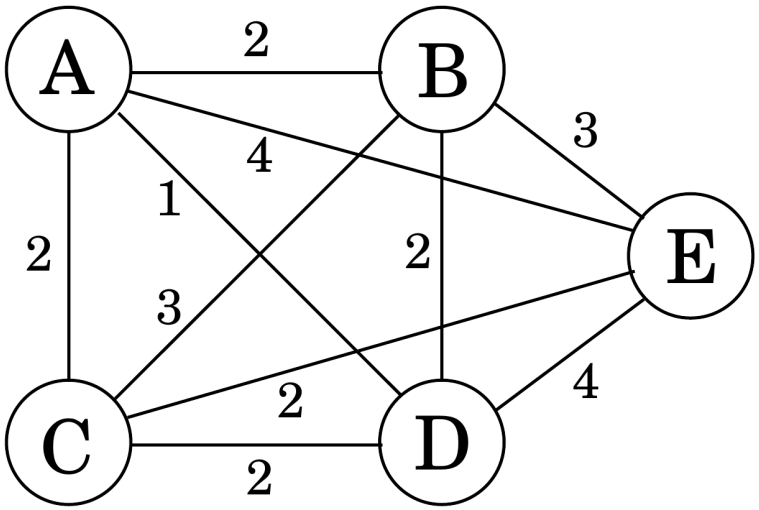


Programação Dinâmica
O Problema do Caixeiro-Viajante



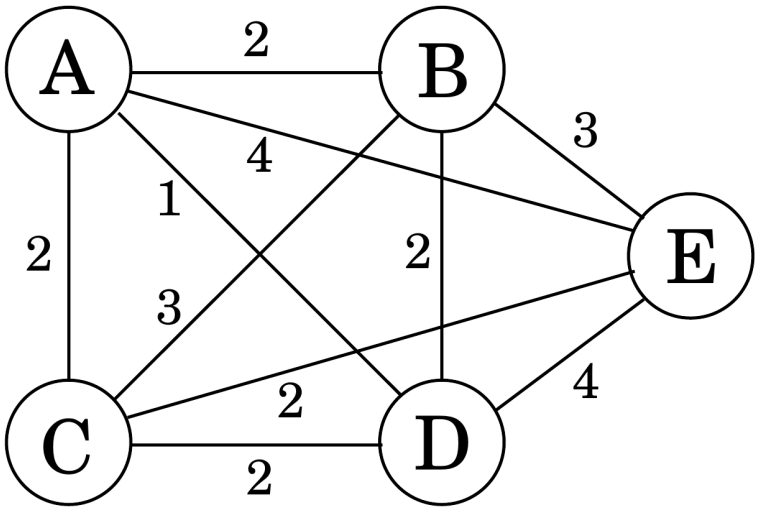
- Um caixeiro-viajante está se preparando para uma grande jornada de vendas.

Programação Dinâmica
O Problema do Caixeiro-Viajante



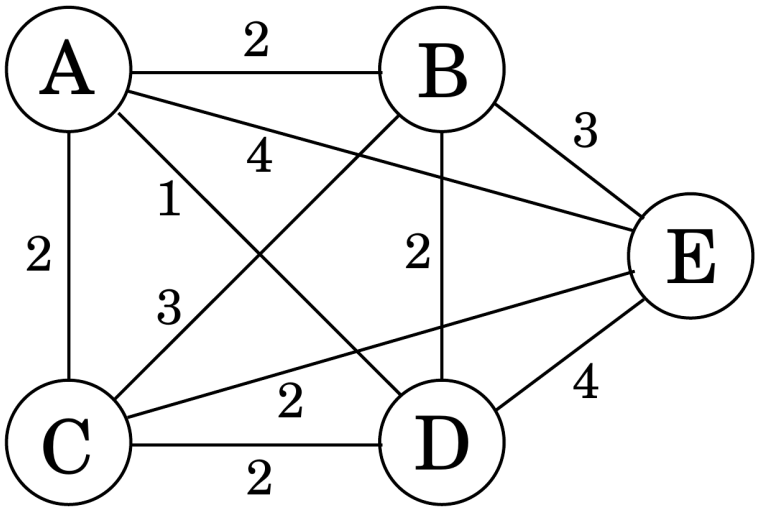
- Um caixeiro-viajante está se preparando para uma grande jornada de vendas.
- Começando em sua cidade natal, maleta em mãos, ele vai conduzir uma jornada na qual cada uma das suas cidades-alvo será visitada exatamente uma vez antes que retorne para casa.

Programação Dinâmica
O Problema do Caixeiro-Viajante



- Um caixeiro-viajante está se preparando para uma grande jornada de vendas.
- Começando em sua cidade natal, maleta em mãos, ele vai conduzir uma jornada na qual cada uma das suas cidades-alvo será visitada exatamente uma vez antes que retorne para casa.
- Dadas as distâncias entre os pares de cidades, qual é a melhor ordem na qual visitá-las, para minimizar a distância total viajada?

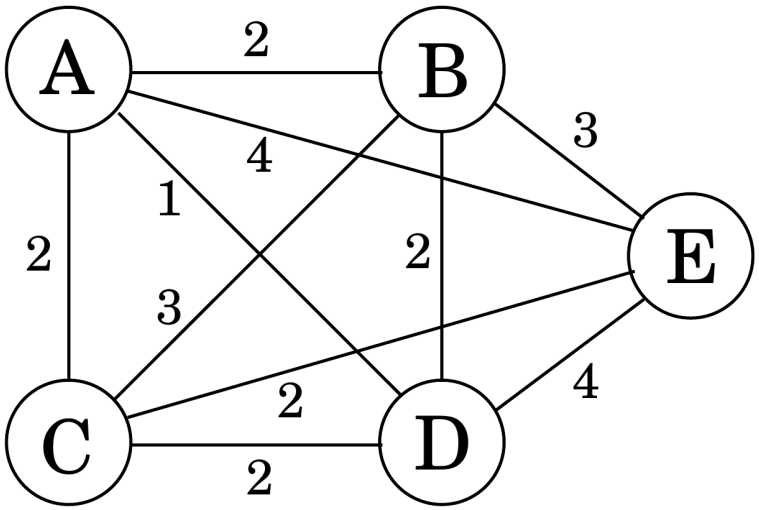
Programação Dinâmica
O Problema do Caixeiro-Viajante



- Subproblema associado:

  Para um subconjunto de cidades $S \subseteq 1, 2, ..., n$ que inclui 1 e $j \in S$, seja $C(S, j)$ o comprimento do caminho, mínimo visitando cada nó em S exatamente uma vez, começaando em 1 e terminando em j.

Programação Dinâmica
O Problema do Caixeiro-Viajante

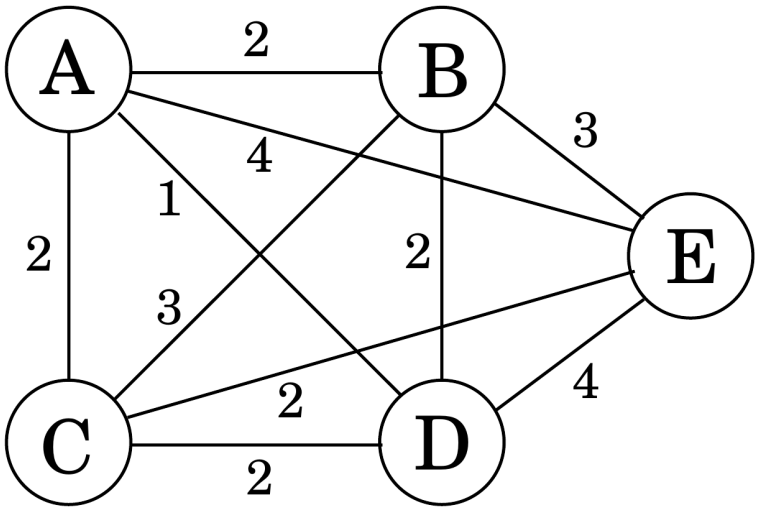


- Subproblema associado:

  Para um subconjunto de cidades $S \subseteq 1, 2, ..., n$ que inclui 1 e $j \in S$, seja $C(S, j)$ o comprimento do caminho, mínimo visitando cada nó em S exatamente uma vez, começaando em 1 e terminando em j.

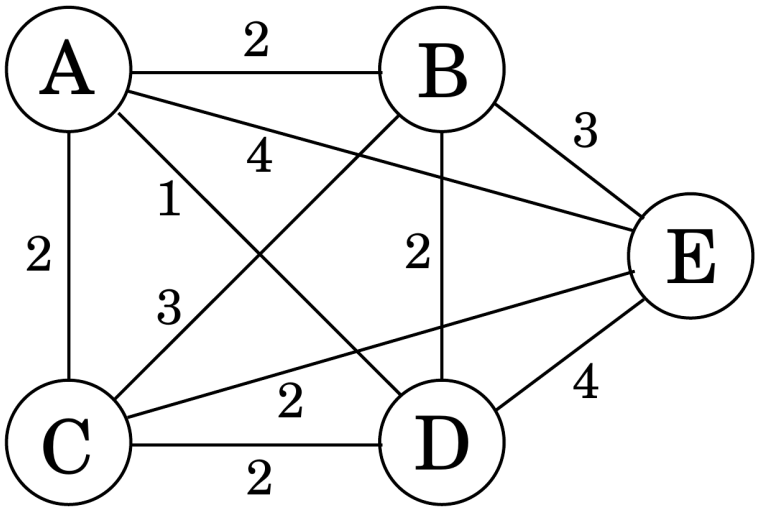
- Quando $|S| > 1$, definimos $C(S, 1) = \infty$, pois o caminho não pode começar e terminar em 1.

Programação Dinâmica
O Problema do Caixeiro-Viajante



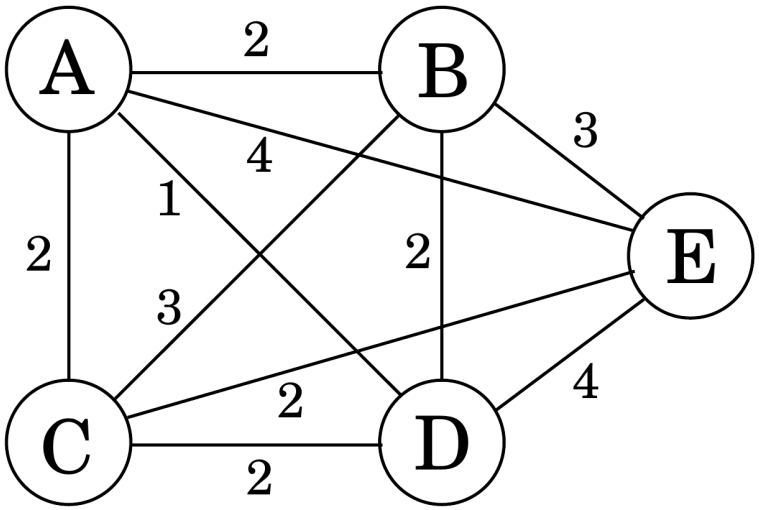
- Expressando $C(S, j)$ em termos de subproblemas menores, precisamos começar em 1 e terminar em j.

Programação Dinâmica
O Problema do Caixeiro-Viajante



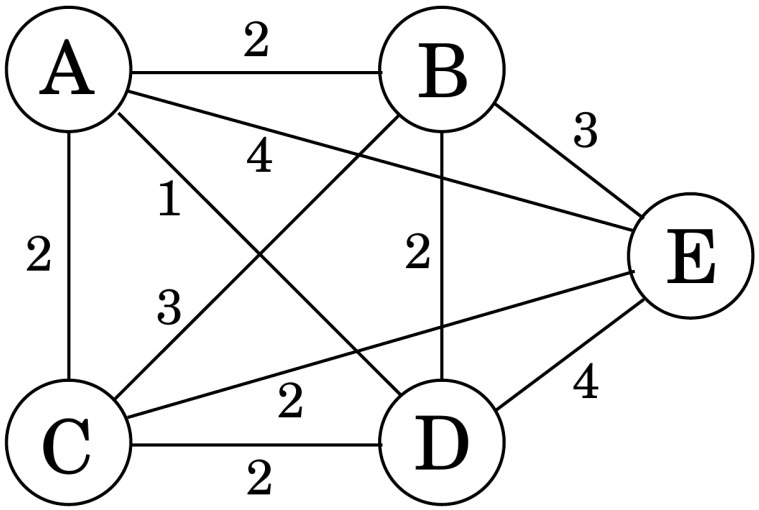
- Expressando $C(S, j)$ em termos de subproblemas menores, precisamos começar em 1 e terminar em j.
- A penúltima cidade a ser visitada tem de ser algum $i \in S$, tal que o comprimento total seja a distância de 1 até i, ou seja, $C(S - j, i)$, mais o comprimento da aresta final, d_{ij}.

Programação Dinâmica
O Problema do Caixeiro-Viajante



- Expressando $C(S, j)$ em termos de subproblemas menores, precisamos começar em 1 e terminar em j.
- A penúltima cidade a ser visitada tem de ser algum $i \in S$, tal que o comprimento total seja a distância de 1 até i, ou seja, $C(S - j, i)$, mais o comprimento da aresta final, d_{ij}.
- Temos de selecionar o melhor i tal que:

Programação Dinâmica
O Problema do Caixeiro-Viajante



- Expressando $C(S, j)$ em termos de subproblemas menores, precisamos começar em 1 e terminar em j.
- A penúltima cidade a ser visitada tem de ser algum $i \in S$, tal que o comprimento total seja a distância de 1 até i, ou seja, $C(S - j, i)$, mais o comprimento da aresta final, d_{ij}.
- Temos de selecionar o melhor i tal que:

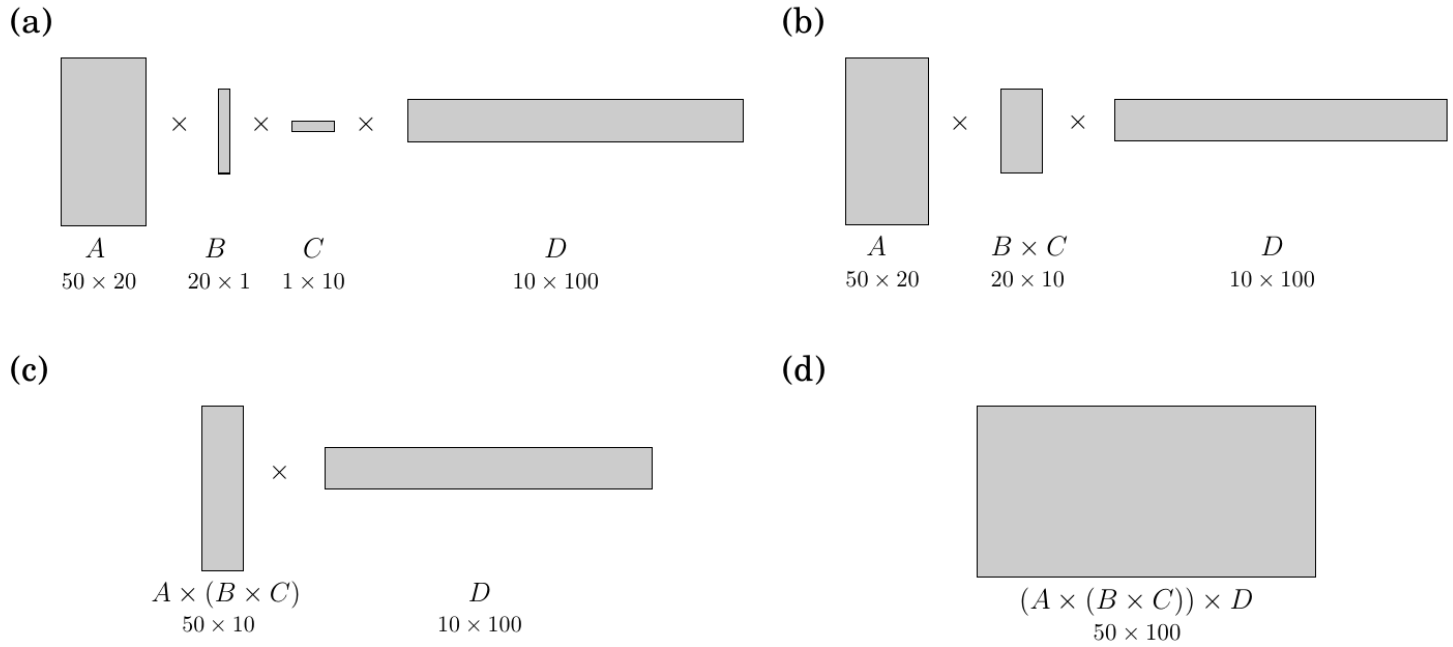
$$C(S, j) = \min_{i \in S : i \neq j} C(S - \{j\}, i) + d_{ij}$$

Programação Dinâmica
O Problema do Caixeiro-Viajante

$C(\{1\}, 1) = 0$
para $s = 2$ até n:
  para todos os subconjuntos $S \subseteq \{1, 2, \ldots, n\}$ de tamanho s e contendo 1:
    $C(S,1) = \infty$
    para todo $j \in S,\ j \neq 1$:
      $C(S, j) = \min\{C(S - \{j\}, i) + d_{ij} : i \in S, i \neq j\}$
retornar $\min_j C(\{1, \ldots, n\}, j) + d_{j1}$

Programação Dinâmica
Multiplicação de cadeias de matrizes

Figure 6.6 $A \times B \times C \times D = (A \times (B \times C)) \times D$.



Programação Dinâmica
Multiplicação de cadeias de matrizes

- Suponha que queiramos multiplicar quatro matrizes, $A \times B \times C \times D$, de dimensões 50×20, 20×1, 1×10 e 10×100, respectivamente.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- Suponha que queiramos multiplicar quatro matrizes, $A \times B \times C \times D$, de dimensões 50×20, 20×1, 1×10 e 10×100, respectivamente.
- Isso envolve multiplicar iterativamente duas matrizes por vez.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- Suponha que queiramos multiplicar quatro matrizes, $A \times B \times C \times D$, de dimensões 50×20, 20×1, 1×10 e 10×100, respectivamente.
- Isso envolve multiplicar iterativamente duas matrizes por vez.
- Multiplicação de matrizes não é comutativa (em geral, $A \times B \neq B \times A$), mas associativa, o que significa, por exemplo, que:

Programação Dinâmica
Multiplicação de cadeias de matrizes

- Suponha que queiramos multiplicar quatro matrizes, $A \times B \times C \times D$, de dimensões 50×20, 20×1, 1×10 e 10×100, respectivamente.
- Isso envolve multiplicar iterativamente duas matrizes por vez.
- Multiplicação de matrizes não é comutativa (em geral, $A \times B \neq B \times A$), mas associativa, o que significa, por exemplo, que:

  $A \times (B \times C) = (A \times B) \times C)$

Programação Dinâmica
Multiplicação de cadeias de matrizes

- Suponha que queiramos multiplicar quatro matrizes, $A \times B \times C \times D$, de dimensões 50×20, 20×1, 1×10 e 10×100, respectivamente.
- Isso envolve multiplicar iterativamente duas matrizes por vez.
- Multiplicação de matrizes não é comutativa (em geral, $A \times B \neq B \times A$), mas associativa, o que significa, por exemplo, que:

  $A \times (B \times C) = (A \times B) \times C)$

- Portanto, podemos computar o nosso produto de quatro matrizes de muitas maneiras diferentes, dependendo de como organizamos os parênteses.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- Suponha que queiramos multiplicar quatro matrizes, $A \times B \times C \times D$, de dimensões 50×20, 20×1, 1×10 e 10×100, respectivamente.
- Isso envolve multiplicar iterativamente duas matrizes por vez.
- Multiplicação de matrizes não é comutativa (em geral, $A \times B \neq B \times A$), mas associativa, o que significa, por exemplo, que:

  $A \times (B \times C) = (A \times B) \times C)$

- Portanto, podemos computar o nosso produto de quatro matrizes de muitas maneiras diferentes, dependendo de como organizamos os parênteses.
- Será que algumas dessas maneiras são melhores do que outras?

Programação Dinâmica
Multiplicação de cadeias de matrizes

- Multiplicar uma matriz $m \times n$ por uma matriz $n \times p$ toma mnp multiplicações, em uma aproximação suficiente.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- Multiplicar uma matriz $m \times n$ por uma matriz $n \times p$ toma mnp multiplicações, em uma aproximação suficiente.
- Usando essa fórmula, vamos comparar várias maneiras diferentes de avaliar $A \times B \times C \times D$:

Programação Dinâmica
Multiplicação de cadeias de matrizes

- Multiplicar uma matriz $m \times n$ por uma matriz $n \times p$ toma mnp multiplicações, em uma aproximação suficiente.
- Usando essa fórmula, vamos comparar várias maneiras diferentes de avaliar $A \times B \times C \times D$:

Organização de parênteses	Computação do custo	Custo
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	120.200
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	60.200
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	7.000

Programação Dinâmica
Multiplicação de cadeias de matrizes

- Multiplicar uma matriz $m \times n$ por uma matriz $n \times p$ toma mnp multiplicações, em uma aproximação suficiente.
- Usando essa fórmula, vamos comparar várias maneiras diferentes de avaliar $A \times B \times C \times D$:

Organização de parênteses	Computação do custo	Custo
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	120.200
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	60.200
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	7.000

- Como determinar a ordem ótima para computar $A_1 \times A_2 \times \cdots A_n$, onde os A_i são matrizes com dimensões $m_0 \times m_1, m_1 \times m_2, ..., m_{n-1} \times m_n$, respectivamente?

Programação Dinâmica
Multiplicação de cadeias de matrizes

- As possíveis ordens, nas quais fazer a multiplicaçãoo, correspondem às várias árvores binárias cheias com n folhas.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- As possíveis ordens, nas quais fazer a multiplicaçãoo, correspondem às várias árvores binárias cheias com n folhas.
- Para uma árvore ser ótima, suas sub-árvores também têm de ser ótimas.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- As possíveis ordens, nas quais fazer a multiplicaçãoo, correspondem às várias árvores binárias cheias com n folhas.
- Para uma árvore ser ótima, suas sub-árvores também têm de ser ótimas.
- Os sub-problemas correspondentes às sub-árvores são produtos da forma $A_i \times A_{i+1} \times \cdots \times A_j$

Programação Dinâmica
Multiplicação de cadeias de matrizes

- As possíveis ordens, nas quais fazer a multiplicaçãoo, correspondem às várias árvores binárias cheias com n folhas.
- Para uma árvore ser ótima, suas sub-árvores também têm de ser ótimas.
- Os sub-problemas correspondentes às sub-árvores são produtos da forma $A_i \times A_{i+1} \times \cdots \times A_j$
- Defini-se:

Programação Dinâmica
Multiplicação de cadeias de matrizes

- As possíveis ordens, nas quais fazer a multiplicaçãoo, correspondem às várias árvores binárias cheias com n folhas.
- Para uma árvore ser ótima, suas sub-árvores também têm de ser ótimas.
- Os sub-problemas correspondentes às sub-árvores são produtos da forma $A_i \times A_{i+1} \times \cdots \times A_j$
- Defini-se:

  $C(i, j) =$ custo mínimo de multiplicar $A_i \times A_{i+1} \times \cdots \times A_j$

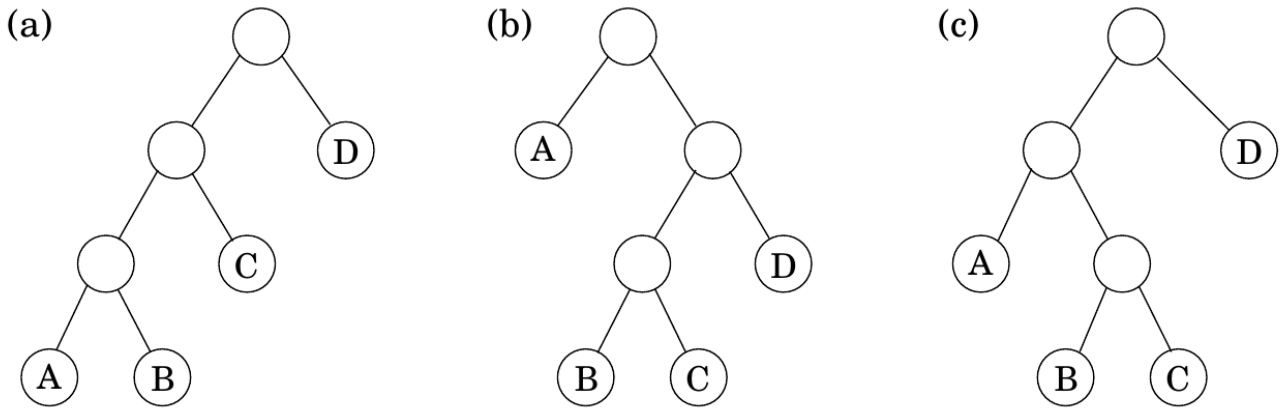
Programação Dinâmica
Multiplicação de cadeias de matrizes

- A primeira coisa a notar é que uma particular organização de parênteses pode ser representada muito naturalmente por uma árvore binária na qual as matrizes individuais correspondem a folhas, a raiz é o produto final e nós interiores são produtos intermediários:

Programação Dinâmica
Multiplicação de cadeias de matrizes

- A primeira coisa a notar é que uma particular organização de parênteses pode ser representada muito naturalmente por uma árvore binária na qual as matrizes individuais correspondem a folhas, a raiz é o produto final e nós interiores são produtos intermediários:

Figure 6.7 (a) $((A \times B) \times C) \times D$; (b) $A \times ((B \times C) \times D)$; (c) $(A \times (B \times C)) \times D$.



Programação Dinâmica
Multiplicação de cadeias de matrizes

- O tamanho desse subproblema é o número de multiplicações de matrizes, $|j - i|$.
- O menor subproblema é quando $i = j$, caso em que não há nada a multiplicar, portanto $C(i, j) = 0$.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- O tamanho desse subproblema é o número de multiplicações de matrizes, $|j - i|$.
- O menor subproblema é quando $i = j$, caso em que não há nada a multiplicar, portanto $C(i, j) = 0$.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- O tamanho desse subproblema é o número de multiplicações de matrizes, $|j - i|$.
- O menor subproblema é quando $i = j$, caso em que não há nada a multiplicar, portanto $C(i, j) = 0$.
- Para $j > i$, considere a subárvore ótima para $C(i, j)$.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- O tamanho desse subproblema é o número de multiplicações de matrizes, $|j - i|$.
- O menor subproblema é quando $i = j$, caso em que não há nada a multiplicar, portanto $C(i, j) = 0$.
- Para $j > i$, considere a subárvore ótima para $C(i, j)$.
- A primeira ramificação nessa subárvore, aquela no topo, vai dividir o produto em duas partes:
- da forma $A_i \cdots \times A_k$ e $A_{k+1} \cdots \times A_j$, para algum k entre i e j.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- O tamanho desse subproblema é o número de multiplicações de matrizes, $|j - i|$.
- O menor subproblema é quando $i = j$, caso em que não há nada a multiplicar, portanto $C(i, j) = 0$.
- Para $j > i$, considere a subárvore ótima para $C(i, j)$.
- A primeira ramificação nessa subárvore, aquela no topo, vai dividir o produto em duas partes:
- da forma $A_i \cdots \times A_k$ e $A_{k+1} \cdots \times A_j$, para algum k entre i e j.
- O custo da subárvore é, então, o custo destes dois produtos parciais, mais o custo de combiná-los:
- $C(i, k) + C(k + 1, j) + m_i \cdot m_k \cdot m_j$.

Programação Dinâmica
Multiplicação de cadeias de matrizes

- E precisamos apenas encontrar o ponto de divisão k para o qual isso é mínimo:

Programação Dinâmica
Multiplicação de cadeias de matrizes

- E precisamos apenas encontrar o ponto de divisão k para o qual isso é mínimo:

$$C(i,j) = \min_{i \leq k < j} \{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j\}$$

Programação Dinâmica
Multiplicação de cadeias de matrizes

- E precisamos apenas encontrar o ponto de divisão k para o qual isso é mínimo:

$$C(i,j) = \min_{i \leq k < j} \{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j\}$$

- Algorimo:

```
para i = 1 até n: C(i,i) = 0
para s = 1 até n − 1:
    para i = 1 até n − s:
        j = i + s
        C(i,j) = min{C(i,k) + C(k+1,j) + m_{i−1} · m_k · m_j : i ≤ k < j}
retornar C(1,n)
```