Projeto e Análise de Algoritmos

Pedro Hokama

Segundo Semestre de 2026

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMm1k03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMm1k03Dt5EMI2s2WQBsLsZ17A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende, Lehilton Pedrosa
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

Projeto de algoritmos recursivos

Algoritmos recursivos

Resolvendo um problema com recursão

1. **instâncias pequenas:** resolver diretamente
2. **instâncias grandes:**
   - construir uma instância menor do mesmo problema
   - resolver essa subinstância recursivamente
   - encontrar uma solução para a instância original

Algoritmos recursivos

Resolvendo um problema com recursão

1. **instâncias pequenas:** resolver diretamente
2. **instâncias grandes:**
   - construir uma instância menor do mesmo problema
   - resolver essa subinstância recursivamente
   - encontrar uma solução para a instância original

Algoritmos recursivos

Resolvendo um problema com recursão

1. **instâncias pequenas:** resolver diretamente
2. **instâncias grandes:**
   - construir uma instância menor do mesmo problema
   - resolver essa subinstância recursivamente
   - encontrar uma solução para a instância original

Algoritmos recursivos

Resolvendo um problema com recursão

1. **instâncias pequenas:** resolver diretamente
2. **instâncias grandes:**
   - construir uma instância menor do mesmo problema
   - resolver essa subinstância recursivamente
   - encontrar uma solução para a instância original

Algoritmos recursivos

Resolvendo um problema com recursão

1. **instâncias pequenas:** resolver diretamente
2. **instâncias grandes:**
   - construir uma instância menor do mesmo problema
   - resolver essa subinstância recursivamente
   - encontrar uma solução para a instância original

Algoritmos recursivos

Resolvendo um problema com recursão

1. **instâncias pequenas:** resolver diretamente
2. **instâncias grandes:**
   - construir uma instância menor do mesmo problema
   - resolver essa subinstância recursivamente
   - encontrar uma solução para a instância original

Projeto de algoritmos recursivos

Uma estratégia é:

1. encontrar e definir um subproblema adequado
2. supor que sabemos resolver instâncias menores
3. construir um algoritmo recursivo para o subproblema

O subproblema escolhido:

- não precisar ser o problema original
- costuma ser uma variante ligeiramente mais geral

Projeto de algoritmos recursivos

Uma estratégia é:

1. encontrar e definir um subproblema adequado
2. supor que sabemos resolver instâncias menores
3. construir um algoritmo recursivo para o subproblema

O subproblema escolhido:

- não precisar ser o problema original
- costuma ser uma variante ligeiramente mais geral

Projeto de algoritmos recursivos

Uma estratégia é:

1. encontrar e definir um subproblema adequado
2. supor que sabemos resolver instâncias menores
3. construir um algoritmo recursivo para o subproblema

O subproblema escolhido:

- não precisar ser o problema original
- costuma ser uma variante ligeiramente mais geral

Projeto de algoritmos recursivos

Uma estratégia é:

1. encontrar e definir um subproblema adequado
2. supor que sabemos resolver instâncias menores
3. construir um algoritmo recursivo para o subproblema

O subproblema escolhido:

- não precisar ser o problema original
- costuma ser uma variante ligeiramente mais geral

Projeto de algoritmos recursivos

Uma estratégia é:

1. encontrar e definir um subproblema adequado
2. supor que sabemos resolver instâncias menores
3. construir um algoritmo recursivo para o subproblema

O subproblema escolhido:

- não precisar ser o problema original
- costuma ser uma variante ligeiramente mais geral

Projeto de algoritmos recursivos

Uma estratégia é:

1. encontrar e definir um subproblema adequado
2. supor que sabemos resolver instâncias menores
3. construir um algoritmo recursivo para o subproblema

O subproblema escolhido:

- não precisar ser o problema original
- costuma ser uma variante ligeiramente mais geral

Projeto de algoritmos recursivos

Uma estratégia é:

1. encontrar e definir um subproblema adequado
2. supor que sabemos resolver instâncias menores
3. construir um algoritmo recursivo para o subproblema

O subproblema escolhido:

- não precisar ser o problema original
- costuma ser uma variante ligeiramente mais geral

Recursão e indução

Este processo é análogo a demonstrações por indução

- **caso básico:** instâncias pequenas
- **caso geral:** instâncias grandes
  - a chamada recursiva corresponde à hipótese de indução
  - a construção da solução corresponde ao passo da indução

Algumas vantagens

- A correção vem do próprio processo indutivo
- A complexidade é expressa numa recorrência

Recursão e indução

Este processo é análogo a demonstrações por indução

- **caso básico:** instâncias pequenas
- **caso geral:** instâncias grandes
  - a chamada recursiva corresponde à hipótese de indução
  - a construção da solução corresponde ao passo da indução

Algumas vantagens

- A correção vem do próprio processo indutivo
- A complexidade é expressa numa recorrência

Recursão e indução

Este processo é análogo a demonstrações por indução

- **caso básico:** instâncias pequenas
- **caso geral:** instâncias grandes
  - a chamada recursiva corresponde à hipótese de indução
  - a construção da solução corresponde ao passo da indução

Algumas vantagens

- A correção vem do próprio processo indutivo
- A complexidade é expressa numa recorrência

Recursão e indução

Este processo é análogo a demonstrações por indução

- **caso básico:** instâncias pequenas
- **caso geral:** instâncias grandes
  - a chamada recursiva corresponde à hipótese de indução
  - a construção da solução corresponde ao passo da indução

Algumas vantagens

- A correção vem do próprio processo indutivo
- A complexidade é expressa numa recorrência

Recursão e indução

Este processo é análogo a demonstrações por indução

- **caso básico:** instâncias pequenas
- **caso geral:** instâncias grandes
  - a chamada recursiva corresponde à hipótese de indução
  - a construção da solução corresponde ao passo da indução

Algumas vantagens

- A correção vem do próprio processo indutivo
- A complexidade é expressa numa recorrência

Recursão e indução

Este processo é análogo a demonstrações por indução

- **caso básico:** instâncias pequenas
- **caso geral:** instâncias grandes
  - a chamada recursiva corresponde à hipótese de indução
  - a construção da solução corresponde ao passo da indução

Algumas vantagens

- A correção vem do próprio processo indutivo
- A complexidade é expressa numa recorrência

Recursão e indução

Este processo é análogo a demonstrações por indução

- **caso básico:** instâncias pequenas
- **caso geral:** instâncias grandes
  - a chamada recursiva corresponde à hipótese de indução
  - a construção da solução corresponde ao passo da indução

Algumas vantagens

- A correção vem do próprio processo indutivo
- A complexidade é expressa numa recorrência

Recursão e indução

Este processo é análogo a demonstrações por indução

- **caso básico:** instâncias pequenas
- **caso geral:** instâncias grandes
  - a chamada recursiva corresponde à hipótese de indução
  - a construção da solução corresponde ao passo da indução

Algumas vantagens

- A correção vem do próprio processo indutivo
- A complexidade é expressa numa recorrência

Projeto de algoritmos recursivos

- Cálculo de polinômios

Cálculo de polinômios

Problema

Entrada:

- uma sequência de números reais $a_n, a_{n-1}, \ldots, a_1, a_0$
- um número real x

Saída:

- o valor do polinômio

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- este é um problema bem simples
- mas queremos fazer poucas adições e multiplicações

Cálculo de polinômios

Problema

Entrada:

- uma sequência de números reais $a_n, a_{n-1}, \ldots, a_1, a_0$
- um número real x

Saída:

- o valor do polinômio

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- este é um problema bem simples
- mas queremos fazer poucas adições e multiplicações

Cálculo de polinômios

Problema

Entrada:

- uma sequência de números reais $a_n, a_{n-1}, \ldots, a_1, a_0$
- um número real x

Saída:

- o valor do polinômio

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- este é um problema bem simples
- mas queremos fazer poucas adições e multiplicações

Cálculo de polinômios

Problema

Entrada:

- uma sequência de números reais $a_n, a_{n-1}, \dots, a_1, a_0$
- um número real x

Saída:

- o valor do polinômio

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- este é um problema bem simples
- mas queremos fazer poucas adições e multiplicações

Cálculo de polinômios

Problema

Entrada:

- uma sequência de números reais $a_n, a_{n-1}, \dots, a_1, a_0$
- um número real x

Saída:

- o valor do polinômio

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- este é um problema bem simples
- mas queremos fazer poucas adições e multiplicações

Cálculo de polinômios

Problema

Entrada:

- uma sequência de números reais $a_n, a_{n-1}, \dots, a_1, a_0$
- um número real x

Saída:

- o valor do polinômio

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- este é um problema bem simples
- mas queremos fazer poucas adições e multiplicações

Cálculo de polinômios

Problema

Entrada:

- uma sequência de números reais $a_n, a_{n-1}, \dots, a_1, a_0$
- um número real x

Saída:

- o valor do polinômio

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- este é um problema bem simples
- mas queremos fazer poucas adições e multiplicações

Cálculo de polinômios

Problema

Entrada:

- uma sequência de números reais $a_n, a_{n-1}, \dots, a_1, a_0$
- um número real x

Saída:

- o valor do polinômio

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- este é um problema bem simples
- mas queremos fazer poucas adições e multiplicações

Primeira tentativa

Hipótese de indução

Dada uma sequência de números reais $a_{n-1}, \dots, a_1, a_0$ e um número real x, sabemos calcular o valor de $P_{n-1}(x) = a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$.

- **Caso base:** $n = 0$.
  - devolvemos a_0
- **Caso geral:** $n > 0$.
  - calculamos $P_{n-1}(x)$ recursivamente
  - somamos $a_n x^n$ ao resultado obtido

Primeira tentativa

Hipótese de indução

Dada uma sequência de números reais $a_{n-1}, \dots, a_1, a_0$ e um número real x, sabemos calcular o valor de $P_{n-1}(x) = a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$.

- **Caso base:** $n = 0$.
  - devolvemos a_0
- **Caso geral:** $n > 0$.
  - calculamos $P_{n-1}(x)$ recursivamente
  - somamos $a_n x^n$ ao resultado obtido

Primeira tentativa

Hipótese de indução

Dada uma sequência de números reais $a_{n-1}, \dots, a_1, a_0$ e um número real x, sabemos calcular o valor de $P_{n-1}(x) = a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$.

- **Caso base:** $n = 0$.
  - devolvemos a_0
- **Caso geral:** $n > 0$.
  - calculamos $P_{n-1}(x)$ recursivamente
  - somamos $a_n x^n$ ao resultado obtido

Primeira tentativa

Hipótese de indução

Dada uma sequência de números reais $a_{n-1}, \ldots, a_1, a_0$ e um número real x, sabemos calcular o valor de $P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$.

- **Caso base**: $n = 0$.
  - devolvemos a_0
- **Caso geral**: $n > 0$.
  - calculamos $P_{n-1}(x)$ recursivamente
  - somamos $a_n x^n$ ao resultado obtido

Primeira tentativa

Hipótese de indução

Dada uma sequência de números reais $a_{n-1}, \ldots, a_1, a_0$ e um número real x, sabemos calcular o valor de $P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$.

- **Caso base**: $n = 0$.
  - devolvemos a_0
- **Caso geral**: $n > 0$.
  - calculamos $P_{n-1}(x)$ recursivamente
  - somamos $a_n x^n$ ao resultado obtido

Primeira tentativa

Hipótese de indução

Dada uma sequência de números reais $a_{n-1}, \ldots, a_1, a_0$ e um número real x, sabemos calcular o valor de $P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$.

- **Caso base**: $n = 0$.
  - devolvemos a_0
- **Caso geral**: $n > 0$.
  - calculamos $P_{n-1}(x)$ recursivamente
  - somamos $a_n x^n$ ao resultado obtido

Algoritmo

```
CÁLCULO-POLINÔMIO(A, x)
 1  se n = 0
 2        y ← a_0
 3  senão
 4        A' ← a_{n-1}, ..., a_1, a_0
 5        y' ← CÁLCULO-POLINÔMIO(A', x)
 6        xn ← 1
 7        para i ← 1 até n
 8              xn ← xn · x
 9        y ← y' + a_n · xn
10  devolva y
```

Complexidade da primeira tentativa

Seja $T(n)$ o número de operações aritméticas realizadas

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + (n+1) \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

Utilizando o método da iteração

$$T(n) = \sum_{i=1}^{n} [(i+1) \text{ multiplicações} + 1 \text{adição}]$$
$$= (n+3)n/2 \text{ multiplicações} + n \text{ adições}.$$

Observações:

- pode-se diminuir o número de multiplicações
- para isso, calculamos x^n com exponenciação rápida

Complexidade da primeira tentativa

Seja $T(n)$ o número de operações aritméticas realizadas

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + (n+1) \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

Utilizando o método da iteração

$$T(n) = \sum_{i=1}^{n} [(i+1) \text{ multiplicações} + 1 \text{adição}]$$
$$= (n+3)n/2 \text{ multiplicações} + n \text{ adições}.$$

Observações:

- pode-se diminuir o número de multiplicações
- para isso, calculamos x^n com exponenciação rápida

Complexidade da primeira tentativa

Seja $T(n)$ o número de operações aritméticas realizadas

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + (n+1) \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

Utilizando o método da iteração

$$T(n) = \sum_{i=1}^{n} [(i+1) \text{ multiplicações} + 1 \text{adição}]$$
$$= (n+3)n/2 \text{ multiplicações} + n \text{ adições}.$$

Observações:

- pode-se diminuir o número de multiplicações
- para isso, calculamos x^n com exponenciação rápida

Complexidade da primeira tentativa

Seja $T(n)$ o número de operações aritméticas realizadas

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + (n+1) \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

Utilizando o método da iteração

$$T(n) = \sum_{i=1}^{n} [(i+1) \text{ multiplicações} + 1 \text{adição}]$$
$$= (n+3)n/2 \text{ multiplicações} + n \text{ adições}.$$

Observações:

- pode-se diminuir o número de multiplicações
- para isso, calculamos x^n com exponenciação rápida

Complexidade da primeira tentativa

Seja $T(n)$ o número de operações aritméticas realizadas

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + (n+1) \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

Utilizando o método da iteração

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} [(i+1) \text{ multiplicações} + 1 \text{adição}] \\ &= (n+3)n/2 \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Observações:

- pode-se diminuir o número de multiplicações
- para isso, calculamos x^n com exponenciação rápida

Complexidade da primeira tentativa

Seja $T(n)$ o número de operações aritméticas realizadas

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + (n+1) \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

Utilizando o método da iteração

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} [(i+1) \text{ multiplicações} + 1 \text{adição}] \\ &= (n+3)n/2 \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Observações:

- pode-se diminuir o número de multiplicações
- para isso, calculamos x^n com exponenciação rápida

Complexidade da primeira tentativa

Seja $T(n)$ o número de operações aritméticas realizadas

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + (n+1) \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

Utilizando o método da iteração

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} [(i+1) \text{ multiplicações} + 1 \text{adição}] \\ &= (n+3)n/2 \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Observações:

- pode-se diminuir o número de multiplicações
- para isso, calculamos x^n com exponenciação rápida

Segunda solução indutiva

Melhorando

- no primeiro algoritmo, recalculamos potências de x.
- podemos reaproveitar o cálculo de x^{n-1}
- para isso, reforçamos a hipótese de indução

Sabemos calcular $P_{[1..n-1]}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ e também o valor de x^{n-1}.

- podemos fazer hipóteses mais fortes sobre a recursão
- mas agora precisamos também devolver o valor de x^n
- i.e., precisamos resolver um problema mais geral

Segunda solução indutiva

Melhorando

- no primeiro algoritmo, recalculamos potências de x.
- podemos reaproveitar o cálculo de x^{n-1}
- para isso, reforçamos a hipótese de indução

Hipótese de indução reforçada

Sabemos calcular $P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ e também o valor de x^{n-1}.

- podemos fazer hipóteses mais fortes sobre a recursão
- mas agora precisamos também devolver o valor de x^n
- i.e., precisamos resolver um problema mais geral

Segunda solução indutiva

Melhorando

- no primeiro algoritmo, recalculamos potências de x.
- podemos reaproveitar o cálculo de x^{n-1}
- para isso, reforçamos a hipótese de indução

Hipótese de indução reforçada

Sabemos calcular $P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ e também o valor de x^{n-1}.

- podemos fazer hipóteses mais fortes sobre a recursão
- mas agora precisamos também devolver o valor de x^n
- i.e., precisamos resolver um problema mais geral

Segunda solução indutiva

Melhorando

- no primeiro algoritmo, recalculamos potências de x.
- podemos reaproveitar o cálculo de x^{n-1}
- para isso, reforçamos a hipótese de indução

Hipótese de indução reforçada

Sabemos calcular $P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ e também o valor de x^{n-1}.

- podemos fazer hipóteses mais fortes sobre a recursão
- mas agora precisamos também devolver o valor de x^n
- i.e., precisamos resolver um problema mais geral

Segunda solução indutiva

Melhorando

- no primeiro algoritmo, recalculamos potências de x.
- podemos reaproveitar o cálculo de x^{n-1}
- para isso, reforçamos a hipótese de indução

Hipótese de indução reforçada

Sabemos calcular $P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ e também o valor de x^{n-1}.

- podemos fazer hipóteses mais fortes sobre a recursão
- mas agora precisamos também devolver o valor de x^n
- i.e., precisamos resolver um problema mais geral

Segunda solução indutiva

Melhorando

- no primeiro algoritmo, recalculamos potências de x.
- podemos reaproveitar o cálculo de x^{n-1}
- para isso, reforçamos a hipótese de indução

Hipótese de indução reforçada

Sabemos calcular $P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ e também o valor de x^{n-1}.

- podemos fazer hipóteses mais fortes sobre a recursão
- mas agora precisamos também devolver o valor de x^n
- i.e., precisamos resolver um problema mais geral

Segunda solução indutiva

Melhorando

- no primeiro algoritmo, recalculamos potências de x.
- podemos reaproveitar o cálculo de x^{n-1}
- para isso, reforçamos a hipótese de indução

Hipótese de indução reforçada

Sabemos calcular $P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ e também o valor de x^{n-1}.

- podemos fazer hipóteses mais fortes sobre a recursão
- mas agora precisamos também devolver o valor de x^n
- i.e., precisamos resolver um problema mais geral

Segunda solução indutiva

Melhorando

- no primeiro algoritmo, recalculamos potências de x.
- podemos reaproveitar o cálculo de x^{n-1}
- para isso, reforçamos a hipótese de indução

Hipótese de indução reforçada

Sabemos calcular $P_{n-1}(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ e também o valor de x^{n-1}.

- podemos fazer hipóteses mais fortes sobre a recursão
- mas agora precisamos também devolver o valor de x^n
- i.e., precisamos resolver um problema mais geral

Segundo algoritmo

```
CÁLCULO-POLINÔMIO(A, x)
1  se n = 0
2        y ← a_0
3        xn ← 1
4  senão
5        A' ← a_{n-1}, ..., a_1, a_0
6        y', x' ← CÁLCULO-POLINÔMIO(A', x)
7        xn ← x * x'
8        y ← y' + a_n * xn
9  devolva y, xn
```

Complexidade do segundo algoritmo

Para essa versão temos

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 2 \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

A solução da recorrência é

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} (2 \text{ multiplicações} + 1 \text{ adição}) \\ &= 2n \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Complexidade do segundo algoritmo

Para essa versão temos

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 2 \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

A solução da recorrência é

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} (2 \text{ multiplicações} + 1 \text{ adição}) \\ &= 2n \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Complexidade do segundo algoritmo

Para essa versão temos

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 2 \text{ multiplicações} + 1 \text{ adição}, & n > 0. \end{cases}$$

A solução da recorrência é

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} (2 \text{ multiplicações} + 1 \text{ adição}) \\ &= 2n \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Terceira solução indutiva

Existem várias escolhas para a definição do subproblema

Outra hipótese de indução

Sabemos calcular $P'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} \cdots + a_1$.

- essa versão é mais simples
- e precisamos de menos operações

Terceira solução indutiva

Existem várias escolhas para a definição do subproblema

Outra hipótese de indução

Sabemos calcular $P'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} \cdots + a_1$.

- essa versão é mais simples
- e precisamos de menos operações

Terceira solução indutiva

Existem várias escolhas para a definição do subproblema

Outra hipótese de indução

Sabemos calcular $P'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} \cdots + a_1$.

- essa versão é mais simples
- e precisamos de menos operações

Terceira solução indutiva

Existem várias escolhas para a definição do subproblema

Outra hipótese de indução

Sabemos calcular $P'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} \cdots + a_1$.

- essa versão é mais simples
- e precisamos de menos operações

Algoritmo melhorado

```
CÁLCULO-POLINÔMIO(A, x)
  se n = 0
      y ← a_0
  senão
      A' ← a_{n-1}, ..., a_1
      y' ← CÁLCULO-POLINÔMIO(A', x)
      y ← x * y' + a_0
  devolva y
```

Complexidade do algoritmo melhorado

Finalmente, temos

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 1 \text{ multiplicação} + 1 \text{ adição}, & n > 0. \end{cases}$$

A solução é

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} (1 \text{ multiplicação} + 1 \text{ adição}) \\ &= n \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Esse algoritmo é chamado de regra de Horner.

Complexidade do algoritmo melhorado

Finalmente, temos

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 1 \text{ multiplicação} + 1 \text{ adição}, & n > 0. \end{cases}$$

A solução é

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} (1 \text{ multiplicação} + 1 \text{ adição}) \\ &= n \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Esse algoritmo é chamado de regra de Horner.

Complexidade do algoritmo melhorado

Finalmente, temos

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 1 \text{ multiplicação} + 1 \text{ adição}, & n > 0. \end{cases}$$

A solução é

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} (1 \text{ multiplicação} + 1 \text{ adição}) \\ &= n \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Esse algoritmo é chamado de regra de Horner.

Complexidade do algoritmo melhorado

Finalmente, temos

$$T(n) = \begin{cases} 0, & n = 0 \\ T(n-1) + 1 \text{ multiplicação} + 1 \text{ adição}, & n > 0. \end{cases}$$

A solução é

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} (1 \text{ multiplicação} + 1 \text{ adição}) \\ &= n \text{ multiplicações} + n \text{ adições}. \end{aligned}$$

Esse algoritmo é chamado de regra de Horner.

Projeto de algoritmos recursivos

- Fator de balanceamento em árvores binárias

Fator de balanceamento em árvores binárias

Vamos relembrar árvore binárias de busca

- cada nó v tem uma chave
- a subárvore esquerda tem chaves menores
- a subárvore direita tem chaves maiores

Definição

O fator de balanceamento (f.b.) de um nó v é a diferença entre as alturas das subárvores esquerda e direita.

- a árvore é balanceada se todo nó tem f.b. limitado
- e.g., em uma árvore AVL, todo nó tem f.b. $-1, 0$ ou $+1$
- uma árvore vazia tem f.b. igual a zero

Fator de balanceamento em árvores binárias

Vamos relembrar árvore binárias de busca

- cada nó v tem uma chave
- a subárvore esquerda tem chaves menores
- a subárvore direita tem chaves maiores

Definição

O fator de balanceamento (f.b.) de um nó v é a diferença entre as alturas das subárvores esquerda e direita.

- a árvore é balanceada se todo nó tem f.b. limitado
- e.g., em uma árvore AVL, todo nó tem f.b. $-1, 0$ ou $+1$
- uma árvore vazia tem f.b. igual a zero

Fator de balanceamento em árvores binárias

Vamos relembrar árvore binárias de busca

- cada nó v tem uma chave
- a subárvore esquerda tem chaves menores
- a subárvore direita tem chaves maiores

Definição

O fator de balanceamento (f.b.) de um nó v é a diferença entre as alturas das subárvores esquerda e direita.

- a árvore é balanceada se todo nó tem f.b. limitado
- e.g., em uma árvore AVL, todo nó tem f.b. $-1, 0$ ou $+1$
- uma árvore vazia tem f.b. igual a zero

Fator de balanceamento em árvores binárias

Vamos relembrar árvore binárias de busca

- cada nó v tem uma chave
- a subárvore esquerda tem chaves menores
- a subárvore direita tem chaves maiores

Definição

O fator de balanceamento (f.b.) de um nó v é a diferença entre as alturas das subárvores esquerda e direita.

- a árvore é balanceada se todo nó tem f.b. limitado
- e.g., em uma árvore AVL, todo nó tem f.b. $-1, 0$ ou $+1$
- uma árvore vazia tem f.b. igual a zero

Fator de balanceamento em árvores binárias

Vamos relembrar árvore binárias de busca

- cada nó v tem uma chave
- a subárvore esquerda tem chaves menores
- a subárvore direita tem chaves maiores

Definição

O fator de balanceamento (f.b.) de um nó v é a diferença entre as alturas das subárvores esquerda e direita.

- a árvore é balanceada se todo nó tem f.b. limitado
- e.g., em uma árvore AVL, todo nó tem f.b. $-1, 0$ ou $+1$
- uma árvore vazia tem f.b. igual a zero

Fator de balanceamento em árvores binárias

Vamos relembrar árvore binárias de busca

- cada nó v tem uma chave
- a subárvore esquerda tem chaves menores
- a subárvore direita tem chaves maiores

Definição

O fator de balanceamento (f.b.) de um nó v é a diferença entre as alturas das subárvores esquerda e direita.

- a árvore é balanceada se todo nó tem f.b. limitado
- e.g., em uma árvore AVL, todo nó tem f.b. $-1, 0$ ou $+1$
- uma árvore vazia tem f.b. igual a zero

Fator de balanceamento em árvores binárias

Vamos relembrar árvore binárias de busca

- cada nó v tem uma chave
- a subárvore esquerda tem chaves menores
- a subárvore direita tem chaves maiores

Definição

O fator de balanceamento (f.b.) de um nó v é a diferença entre as alturas das subárvores esquerda e direita.

- a árvore é balanceada se todo nó tem f.b. limitado
- e.g., em uma árvore AVL, todo nó tem f.b. $-1, 0$ ou $+1$
- uma árvore vazia tem f.b. igual a zero

Fator de balanceamento em árvores binárias

Vamos relembrar árvore binárias de busca

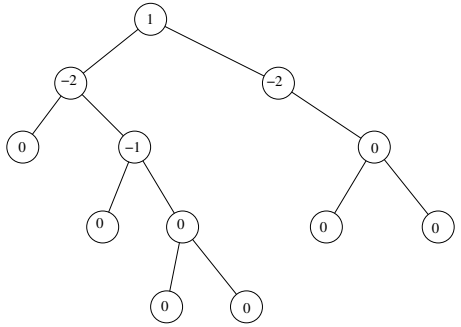
- cada nó v tem uma chave
- a subárvore esquerda tem chaves menores
- a subárvore direita tem chaves maiores

Definição

O fator de balanceamento (f.b.) de um nó v é a diferença entre as alturas das subárvores esquerda e direita.

- a árvore é balanceada se todo nó tem f.b. limitado
- e.g., em uma árvore AVL, todo nó tem f.b. $-1, 0$ ou $+1$
- uma árvore vazia tem f.b. igual a zero

Exemplo



Uma árvore e o f.b. de cada nó.

Calculando o fator de balanceamento

Problema

Entrada:

- uma árvore binária A com n nós

Saída:

- fatores de balanceamento de cada nó de A

Vamos projetar o algoritmo indutivamente.

Sabemos como calcular fatores de balanceamento de árvores com menos que n nós.

Calculando o fator de balanceamento

Problema

Entrada:

- uma árvore binária A com n nós

Saída:

- fatores de balanceamento de cada nó de A

Vamos projetar o algoritmo indutivamente.

Sabemos como calcular fatores de balanceamento de árvores com menos que n nós.

Calculando o fator de balanceamento

Problema

Entrada:

- uma árvore binária A com n nós

Saída:

- fatores de balanceamento de cada nó de A

Vamos projetar o algoritmo indutivamente.

Hipótese de indução

Sabemos como calcular fatores de balanceamento de árvores com menos que n nós.

Calculando o fator de balanceamento

Problema

Entrada:

- uma árvore binária A com n nós

Saída:

- fatores de balanceamento de cada nó de A

Vamos projetar o algoritmo indutivamente.

Hipótese de indução

Sabemos como calcular fatores de balanceamento de árvores com menos que n nós.

Calculando o fator de balanceamento

Problema

Entrada:

- uma árvore binária A com n nós

Saída:

- fatores de balanceamento de cada nó de A

Vamos projetar o algoritmo indutivamente.

Hipótese de indução

Sabemos como calcular fatores de balanceamento de árvores com menos que n nós.

Calculando o fator de balanceamento

Problema

Entrada:

- uma árvore binária A com n nós

Saída:

- fatores de balanceamento de cada nó de A

Vamos projetar o algoritmo indutivamente.

Hipótese de indução

Sabemos como calcular fatores de balanceamento de árvores com menos que n nós.

Calculando o fator de balanceamento

Problema

Entrada:

- uma árvore binária A com n nós

Saída:

- fatores de balanceamento de cada nó de A

Vamos projetar o algoritmo indutivamente.

Hipótese de indução

Sabemos como calcular fatores de balanceamento de árvores com menos que n nós.

Projeto recursivo

Uma dificuldade

- a definição de f.b. não é recursiva!
- o f.b. de um nó depende das alturas das subárvores
- mas sabemos apenas o f.b. das subárvores
- conclusão: é necessária uma h.i. mais forte!

Nova hipótese de indução

Para um $n > 0$ qualquer, sabemos como calcular fatores de balanceamento e alturas de árvores com menos que n nós.

Projeto recursivo

Uma dificuldade

- a definição de f.b. não é recursiva!
- o f.b. de um nó depende das alturas das subárvores
- mas sabemos apenas o f.b. das subárvores
- conclusão: é necessária uma h.i. mais forte!

Nova hipótese de indução

Para um $n > 0$ qualquer, sabemos como calcular fatores de balanceamento e alturas de árvores com menos que n nós.

Projeto recursivo

Uma dificuldade

- a definição de f.b. não é recursiva!
- o f.b. de um nó depende das alturas das subárvores
- mas sabemos apenas o f.b. das subárvores
- conclusão: é necessária uma h.i. mais forte!

Nova hipótese de indução

Para um $n > 0$ qualquer, sabemos como calcular fatores de balanceamento e alturas de árvores com menos que n nós.

Projeto recursivo

Uma dificuldade

- a definição de f.b. não é recursiva!
- o f.b. de um nó depende das alturas das subárvores
- mas sabemos apenas o f.b. das subárvores
- conclusão: é necessária uma h.i. mais forte!

Nova hipótese de indução

Para um $n > 0$ qualquer, sabemos como calcular fatores de balanceamento e alturas de árvores com menos que n nós.

Projeto recursivo

Uma dificuldade

- a definição de f.b. não é recursiva!
- o f.b. de um nó depende das alturas das subárvores
- mas sabemos apenas o f.b. das subárvores
- conclusão: é necessária uma h.i. mais forte!

Nova hipótese de indução

Para um $n > 0$ qualquer, sabemos como calcular fatores de balanceamento e alturas de árvores com menos que n nós.

Projeto recursivo

Uma dificuldade

- a definição de f.b. não é recursiva!
- o f.b. de um nó depende das alturas das subárvores
- mas sabemos apenas o f.b. das subárvores
- conclusão: é necessária uma h.i. mais forte!

Nova hipótese de indução

Para um $n > 0$ qualquer, sabemos como calcular fatores de balanceamento e alturas de árvores com menos que n nós.

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_E
   - do mesmo modo, obtemos a altura da árvore direita h_D
   - por definição, f.b. é $h_E - h_D$
   - e a altura é $\max\{h_E, h_D\} + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Encontrando a solução do subproblema

1. se $n = 0$, a árvore a árvore é vazia, então
   - o f.b. é igual 0.
   - a altura é igual 0.
2. se $n > 0$, a árvore tem pelo menos um nó
   - recursivamente, obtemos a altura da árvore esquerda h_e
   - do mesmo modo, obtemos a altura da árvore direita h_d
   - por definição, f.b. é $h_e - h_d$
   - e a altura é $\max(h_e, h_d) + 1$

Observações

- repare que não precisamos de f.b. para das subárvores
- mas obtemos a altura recursivamente
- de novo, fortalecer a h.i. simplificou o projeto do algoritmo

Pseudocódigo do algoritmo

```
FatorAltura(A)
1  se n = 0
2        f ← 0
3        h ← 0
4  senão
5        f_e, h_e ← FatorAltura(A_e)
6        f_d, h_d ← FatorAltura(A_d)
7        f ← h_e − h_d
8        h ← max(h_e, h_d) + 1
9  devolva f, h
```

Complexidade

Seja $T(n)$ o número de operações executadas no pior caso,

$$T(n) = \begin{cases} \Theta(1), & n = 0 \\ T(n_e) + T(n_d) + \Theta(1), & n > 0, \end{cases}$$

onde n_e, n_d são os números de nós das subárvores.

Estimando o pior caso

- O pior caso parece acontecer se $n_e = 0$ e $n_d = n - 1$
- Nesse caso, temos

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(1) \\ &= T(n-2) + 2T(0) + 2\Theta(1) \\ &\vdots \\ &= T(0) + nT(0) + n\Theta(1) \\ &= (n+1)\Theta(1) + n\Theta(1) = \Theta(n). \end{aligned}$$

Exercício: argumente que este é, de fato, um pior caso.

Estimando o pior caso

- O pior caso parece acontecer se $n_e = 0$ e $n_d = n - 1$
- Nesse caso, temos

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(1) \\ &= T(n-2) + 2T(0) + 2\Theta(1) \\ &\vdots \\ &= T(0) + nT(0) + n\Theta(1) \\ &= (n+1)\Theta(1) + n\Theta(1) = \Theta(n). \end{aligned}$$

Exercício: argumente que este é, de fato, um pior caso.

Estimando o pior caso

- O pior caso parece acontecer se $n_e = 0$ e $n_d = n - 1$
- Nesse caso, temos

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(1) \\ &= T(n-2) + 2T(0) + 2\Theta(1) \\ &\vdots \\ &= T(0) + nT(0) + n\Theta(1) \\ &= (n+1)\Theta(1) + n\Theta(1) = \Theta(n). \end{aligned}$$

Exercício: argumente que este é, de fato, um pior caso.

Estimando o pior caso

- O pior caso parece acontecer se $n_e = 0$ e $n_d = n - 1$
- Nesse caso, temos

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(1) \\ &= T(n-2) + 2T(0) + 2\Theta(1) \\ &\vdots \\ &= T(0) + nT(0) + n\Theta(1) \\ &= (n+1)\Theta(1) + n\Theta(1) = \Theta(n). \end{aligned}$$

Exercício: argumente que este é, de fato, um pior caso.

Projeto de algoritmos recursivos

- Problema da celebridade

Celebridades

Definição

Em um conjunto S de n pessoas, uma celebridade é alguém que é conhecido por todas as pessoas de S mas que não conhece ninguém.

- nem sempre existe uma celebridade
- mas só pode existir uma celebridade em S
- queremos saber se existe uma celebridade em S

Celebridades

Definição

Em um conjunto S de n pessoas, uma celebridade é alguém que é conhecido por todas as pessoas de S mas que não conhece ninguém.

- nem sempre existe uma celebridade
- mas só pode existir uma celebridade em S
- queremos saber se existe uma celebridade em S

Celebridades

Definição

Em um conjunto S de n pessoas, uma celebridade é alguém que é conhecido por todas as pessoas de S mas que não conhece ninguém.

- nem sempre existe uma celebridade
- mas só pode existir uma celebridade em S
- queremos saber se existe uma celebridade em S

Celebridades

Definição

Em um conjunto S de n pessoas, uma celebridade é alguém que é conhecido por todas as pessoas de S mas que não conhece ninguém.

- nem sempre existe uma celebridade
- mas só pode existir uma celebridade em S
- queremos saber se existe uma celebridade em S

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

Saída:

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

Saída:

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

O problema da celebridade

Formalizando o problema:

- queremos descrever o conhecimento entre n pessoas
- representamos isso com uma matriz binária $n \times n$
- $M[i,j] = 1$ se i conhece j, e $M[i,j] = 0$ caso contrário

Problema

Entrada:

- um conjunto de n pessoas e a matriz associada M

Saída:

- uma celebridade, se existir

Observações

- para uma celebridade k, vale $M[i,k] = 1$ para todo i
- analogamente, vale $M[k,i] = 0$ para todo i
- um algoritmo simples verifica se cada pessoa é celebridade, usando $2n(n-1)$ operações

Argumento indutivo

Um argumento indutivo pode ser:

Hipótese de indução

Sabemos encontrar uma celebridade em um conjunto de $n-1$ pessoas ou decidir que não existe.

- se $n = 1$, a única pessoa no conjunto é uma celebridade
- se $n \geq 2$
  - represente o conjunto de pessoas como $S = \{1, 2, \ldots, n\}$
  - podemos encontrar celebridade em $S' = \{1, 2, \ldots, n-1\}$
  - como isso ajuda na instância original?

Argumento indutivo

Um argumento indutivo pode ser:

Hipótese de indução

Sabemos encontrar uma celebridade em um conjunto de $n-1$ pessoas ou decidir que não existe.

- se $n = 1$, a única pessoa no conjunto é uma celebridade
- se $n \geq 2$
  - represente o conjunto de pessoas como $S = \{1, 2, \ldots, n\}$
  - podemos encontrar celebridade em $S' = \{1, 2, \ldots, n-1\}$
  - como isso ajuda na instância original?

Argumento indutivo

Um argumento indutivo pode ser:

Hipótese de indução

Sabemos encontrar uma celebridade em um conjunto de $n-1$ pessoas ou decidir que não existe.

- se $n = 1$, a única pessoa no conjunto é uma celebridade
- se $n \geq 2$
  - represente o conjunto de pessoas como $S = \{1, 2, \ldots, n\}$
  - podemos encontrar celebridade em $S' = \{1, 2, \ldots, n-1\}$
  - como isso ajuda na instância original?

Argumento indutivo

Um argumento indutivo pode ser:

Hipótese de indução

Sabemos encontrar uma celebridade em um conjunto de $n-1$ pessoas ou decidir que não existe.

- se $n = 1$, a única pessoa no conjunto é uma celebridade
- se $n \geq 2$
  - represente o conjunto de pessoas como $S = \{1, 2, \ldots, n\}$
  - podemos encontrar celebridade em $S' = \{1, 2, \ldots, n-1\}$
  - como isso ajuda na instância original?

Argumento indutivo

Um argumento indutivo pode ser:

Hipótese de indução

Sabemos encontrar uma celebridade em um conjunto de $n-1$ pessoas ou decidir que não existe.

- se $n = 1$, a única pessoa no conjunto é uma celebridade
- se $n \geq 2$
  - represente o conjunto de pessoas como $S = \{1, 2, \ldots, n\}$
  - podemos encontrar celebridade em $S' = \{1, 2, \ldots, n-1\}$
  - como isso ajuda na instância original?

Argumento indutivo

Um argumento indutivo pode ser:

Hipótese de indução

Sabemos encontrar uma celebridade em um conjunto de $n-1$ pessoas ou decidir que não existe.

- se $n = 1$, a única pessoa no conjunto é uma celebridade
- se $n \geq 2$
  - represente o conjunto de pessoas como $S = \{1, 2, \ldots, n\}$
  - podemos encontrar celebridade em $S' = \{1, 2, \ldots, n-1\}$
  - como isso ajuda na instância original?

Argumento indutivo

Um argumento indutivo pode ser:

Hipótese de indução

Sabemos encontrar uma celebridade em um conjunto de $n-1$ pessoas ou decidir que não existe.

- se $n = 1$, a única pessoa no conjunto é uma celebridade
- se $n \geq 2$
  - represente o conjunto de pessoas como $S = \{1, 2, \ldots, n\}$
  - podemos encontrar celebridade em $S' = \{1, 2, \ldots, n-1\}$
  - como isso ajuda na instância original?

Passo indutivo

Dois casos:

1. Existe celebridade k em S'
   - se $M[n,k] = 1$ e $M[k,n] = 0$, então k é celebridade em S
   - senão, n ainda poderia ser celebridade
2. Não existe celebridade em S'
   - nesse caso, apenas n poderia ser celebridade

- em qualquer caso, precisamos verificar se n é celebridade
- temos que verificar se $M[n,j] = 0$ e $M[j,n] = 1$ para $j < n$
- também obtemos um algoritmo quadrático

Passo indutivo

Dois casos:

1. Existe celebridade k em S'
   - se $M[n,k] = 1$ e $M[k,n] = 0$, então k é celebridade em S
   - senão, n ainda poderia ser celebridade
2. Não existe celebridade em S'
   - nesse caso, apenas n poderia ser celebridade

- em qualquer caso, precisamos verificar se n é celebridade
- temos que verificar se $M[n,j] = 0$ e $M[j,n] = 1$ para $j < n$
- também obtemos um algoritmo quadrático

Passo indutivo

Dois casos:

1. Existe celebridade k em S'
   - se $M[n,k] = 1$ e $M[k,n] = 0$, então k é celebridade em S
   - senão, n ainda poderia ser celebridade
2. Não existe celebridade em S'
   - nesse caso, apenas n poderia ser celebridade

- em qualquer caso, precisamos verificar se n é celebridade
- temos que verificar se $M[n,j] = 0$ e $M[j,n] = 1$ para $j < n$
- também obtemos um algoritmo quadrático

Passo indutivo

Dois casos:

1. Existe celebridade k em S'
   - se $M[n,k] = 1$ e $M[k,n] = 0$, então k é celebridade em S
   - senão, n ainda poderia ser celebridade
2. Não existe celebridade em S'
   - nesse caso, apenas n poderia ser celebridade

- em qualquer caso, precisamos verificar se n é celebridade
- temos que verificar se $M[n,j] = 0$ e $M[j,n] = 1$ para $j < n$
- também obtemos um algoritmo quadrático

Passo indutivo

Dois casos:

1. Existe celebridade k em S'
   - se $M[n,k] = 1$ e $M[k,n] = 0$, então k é celebridade em S
   - senão, n ainda poderia ser celebridade
2. Não existe celebridade em S'
   - nesse caso, apenas n poderia ser celebridade

- em qualquer caso, precisamos verificar se n é celebridade
- temos que verificar se $M[n,j] = 0$ e $M[j,n] = 1$ para $j < n$
- também obtemos um algoritmo quadrático

Passo indutivo

Dois casos:

1. Existe celebridade k em S'
   - se $M[n,k] = 1$ e $M[k,n] = 0$, então k é celebridade em S
   - senão, n ainda poderia ser celebridade
2. Não existe celebridade em S'
   - nesse caso, apenas n poderia ser celebridade

- em qualquer caso, precisamos verificar se n é celebridade
- temos que verificar se $M[n,j] = 0$ e $M[j,n] = 1$ para $j < n$
- também obtemos um algoritmo quadrático

Passo indutivo

Dois casos:

1. Existe celebridade k em S'
   - se $M[n,k] = 1$ e $M[k,n] = 0$, então k é celebridade em S
   - senão, n ainda poderia ser celebridade
2. Não existe celebridade em S'
   - nesse caso, apenas n poderia ser celebridade

- em qualquer caso, precisamos verificar se n é celebridade
- temos que verificar se $M[n,j] = 0$ e $M[j,n] = 1$ para $j < n$
- também obtemos um algoritmo quadrático

Passo indutivo

Dois casos:

1. Existe celebridade k em S'
   - se $M[n,k] = 1$ e $M[k,n] = 0$, então k é celebridade em S
   - senão, n ainda poderia ser celebridade
2. Não existe celebridade em S'
   - nesse caso, apenas n poderia ser celebridade

- em qualquer caso, precisamos verificar se n é celebridade
- temos que verificar se $M[n,j] = 0$ e $M[j,n] = 1$ para $j < n$
- também obtemos um algoritmo quadrático

Passo indutivo

Dois casos:

1. Existe celebridade k em S'
   - se $M[n,k] = 1$ e $M[k,n] = 0$, então k é celebridade em S
   - senão, n ainda poderia ser celebridade
2. Não existe celebridade em S'
   - nesse caso, apenas n poderia ser celebridade

- em qualquer caso, precisamos verificar se n é celebridade
- temos que verificar se $M[n,j] = 0$ e $M[j,n] = 1$ para $j < n$
- também obtemos um algoritmo quadrático

Segunda tentativa

Suponha que s não é celebridade

- podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$
- e não precisaríamos verificar s depois

Como encontrar alguém que não é celebridade?

- considere duas pessoas i e j
- se $M[i,j] = 1$, então i não é celebridade
- mas se $M[i,j] = 0$, então j não é celebridade

Vamos usar a mesma hipótese anterior

- i.e., consideramos o mesmo subproblema
- mas escolhemos a subinstância mais cuidadosamente

Segunda tentativa

Suponha que s não é celebridade

- podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$
- e não precisaríamos verificar s depois

Como encontrar alguém que não é celebridade?

- considere duas pessoas i e j
- se $M[i, j] = 1$, então i não é celebridade
- mas se $M[i, j] = 0$, então j não é celebridade

Vamos usar a mesma hipótese anterior

- i.e., consideramos o mesmo subproblema
- mas escolhemos a subinstância mais cuidadosamente

Segunda tentativa

Suponha que s não é celebridade

- podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$
- e não precisaríamos verificar s depois

Como encontrar alguém que não é celebridade?

- considere duas pessoas i e j
- se $M[i, j] = 1$, então i não é celebridade
- mas se $M[i, j] = 0$, então j não é celebridade

Vamos usar a mesma hipótese anterior

- i.e., consideramos o mesmo subproblema
- mas escolhemos a subinstância mais cuidadosamente

Segunda tentativa

Suponha que s não é celebridade

- podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$
- e não precisaríamos verificar s depois

Como encontrar alguém que não é celebridade?

- considere duas pessoas i e j
- se $M[i, j] = 1$, então i não é celebridade
- mas se $M[i, j] = 0$, então j não é celebridade

Vamos usar a mesma hipótese anterior

- i.e., consideramos o mesmo subproblema
- mas escolhemos a subinstância mais cuidadosamente

Segunda tentativa

Suponha que s não é celebridade

- podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$
- e não precisaríamos verificar s depois

Como encontrar alguém que não é celebridade?

- considere duas pessoas i e j
- se $M[i, j] = 1$, então i não é celebridade
- mas se $M[i, j] = 0$, então j não é celebridade

Vamos usar a mesma hipótese anterior

- i.e., consideramos o mesmo subproblema
- mas escolhemos a subinstância mais cuidadosamente

Segunda tentativa

Suponha que s não é celebridade

- podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$
- e não precisaríamos verificar s depois

Como encontrar alguém que não é celebridade?

- considere duas pessoas i e j
- se $M[i,j] = 1$, então i não é celebridade
- mas se $M[i,j] = 0$, então j não é celebridade

Vamos usar a mesma hipótese anterior

- i.e., consideramos o mesmo subproblema
- mas escolhemos a subinstância mais cuidadosamente

Segunda tentativa

Suponha que s não é celebridade

- podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$
- e não precisaríamos verificar s depois

Como encontrar alguém que não é celebridade?

- considere duas pessoas i e j
- se $M[i,j] = 1$, então i não é celebridade
- mas se $M[i,j] = 0$, então j não é celebridade

Vamos usar a mesma hipótese anterior

- i.e., consideramos o mesmo subproblema
- mas escolhemos a subinstância mais cuidadosamente

Segunda tentativa

Suponha que s não é celebridade

- podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$
- e não precisaríamos verificar s depois

Como encontrar alguém que não é celebridade?

- considere duas pessoas i e j
- se $M[i,j] = 1$, então i não é celebridade
- mas se $M[i,j] = 0$, então j não é celebridade

Vamos usar a mesma hipótese anterior

- i.e., consideramos o mesmo subproblema
- mas escolhemos a subinstância mais cuidadosamente

Segunda tentativa

Suponha que s não é celebridade

- podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$
- e não precisaríamos verificar s depois

Como encontrar alguém que não é celebridade?

- considere duas pessoas i e j
- se $M[i,j] = 1$, então i não é celebridade
- mas se $M[i,j] = 0$, então j não é celebridade

Vamos usar a mesma hipótese anterior

- i.e., consideramos o mesmo subproblema
- mas escolhemos a subinstância mais cuidadosamente

Segunda tentativa

Suponha que s não é celebridade

- podemos fazer a chamada recursiva para $S' = S \setminus \{s\}$
- e não precisaríamos verificar s depois

Como encontrar alguém que não é celebridade?

- considere duas pessoas i e j
- se $M[i,j] = 1$, então i não é celebridade
- mas se $M[i,j] = 0$, então j não é celebridade

Vamos usar a mesma hipótese anterior

- i.e., consideramos o mesmo subproblema
- mas escolhemos a subinstância mais cuidadosamente

Encontrando uma celebridade

O algoritmo devolve NIL se não houver celebridade.

```
CELEBRIDADE(S, M)
 1  se |S| = 1
 2     seja k a única pessoa em S
 3  senão
 4     sejam i, j duas pessoas em S
 5     se M[i,j] = 1
 6        s ← i
 7     senão
 8        s ← j
 9     S' ← S \ {s}
10     k ← CELEBRIDADE(S', M)
11     se k = NIL ou M[s,k] = 0 ou M[k,s] = 1
12        k ← NIL
13  devolva k
```

Exemplo 4 - Complexidade

Seja $T(n)$ o número de operações executadas

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução da recorrência é

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$

Exemplo 4 - Complexidade

Seja $T(n)$ o número de operações executadas

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução da recorrência é

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$

Exemplo 4 - Complexidade

Seja $T(n)$ o número de operações executadas

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução da recorrência é

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$

Exemplo 4 - Complexidade

Seja $T(n)$ o número de operações executadas

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução da recorrência é

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$

Projeto de algoritmos recursivos

- Subsequência consecutiva máxima

Subsequência consecutiva máxima (SCM)

Problema

Entrada:

- uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais

Saída:

- uma subsequência consecutiva $Y = x_i, x_{i+1}, \ldots, x_j$, onde $1 \leq i, j \leq n$, cuja soma seja máxima.

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$	Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$	Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$	Resp: $Y = [\,]$

Subsequência consecutiva máxima (SCM)

Problema

Entrada:

- uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais

Saída:

- uma subsequência consecutiva $Y = x_i, x_{i+1}, \ldots, x_j$, onde $1 \leq i, j \leq n$, cuja soma seja máxima.

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$	Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$	Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$	Resp: $Y = [\,]$

Subsequência consecutiva máxima (SCM)

Problema

Entrada:

- uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais

Saída:

- uma subsequência consecutiva $Y = x_i, x_{i+1}, \ldots, x_j$, onde $1 \leq i, j \leq n$, cuja soma seja máxima.

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$	Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$	Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$	Resp: $Y = [\,]$

Subsequência consecutiva máxima (SCM)

Problema

Entrada:

- uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais

Saída:

- uma subsequência consecutiva $Y = x_i, x_{i+1}, \ldots, x_j$, onde $1 \leq i, j \leq n$, cuja soma seja máxima.

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$	Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$	Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$	Resp: $Y = [\,]$

Subsequência consecutiva máxima (SCM)

Problema

Entrada:

- uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais

Saída:

- uma subsequência consecutiva $Y = x_i, x_{i+1}, \ldots, x_j$, onde $1 \leq i, j \leq n$, cuja soma seja máxima.

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$	Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$	Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$	Resp: $Y = [\,]$

Subsequência consecutiva máxima (SCM)

Problema

Entrada:

- uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais

Saída:

- uma subsequência consecutiva $Y = x_i, x_{i+1}, \ldots, x_j$, onde $1 \leq i, j \leq n$, cuja soma seja máxima.

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$ Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$ Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$ Resp: $Y = [\,]$

Subsequência consecutiva máxima (SCM)

Problema

Entrada:

- uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais

Saída:

- uma subsequência consecutiva $Y = x_i, x_{i+1}, \ldots, x_j$, onde $1 \leq i, j \leq n$, cuja soma seja máxima.

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$ Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$ Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$ Resp: $Y = [\,]$

Subsequência consecutiva máxima (SCM)

Problema

Entrada:

- uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais

Saída:

- uma subsequência consecutiva $Y = x_i, x_{i+1}, \ldots, x_j$, onde $1 \leq i, j \leq n$, cuja soma seja máxima.

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$ Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$ Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$ Resp: $Y = [\,]$

Subsequência consecutiva máxima (SCM)

Problema

Entrada:

- uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais

Saída:

- uma subsequência consecutiva $Y = x_i, x_{i+1}, \ldots, x_j$, onde $1 \leq i, j \leq n$, cuja soma seja máxima.

Exemplos:

$X = [4, 2, -7, 3, 0, -2, 1, 5, -2]$ Resp: $Y = [3, 0, -2, 1, 5]$
$X = [-1, -2, 0]$ Resp: $Y = [0]$ ou $Y = [\,]$
$X = [-3, -1]$ Resp: $Y = [\,]$

Hipótese

Hipótese de indução

Sabemos calcular a SCM de sequências de comprimento $n-1$.

- considere uma sequência $X = x_1, x_2, \ldots, x_n$
- removendo x_n, obtemos uma sequência menor X'
- usando a h.i., obtemos uma SCM $Y' = x_i, x_{i+1}, \ldots, x_j$ de X'

Hipótese

Hipótese de indução

Sabemos calcular a SCM de sequências de comprimento $n-1$.

- considere uma sequência $X = x_1, x_2, \ldots, x_n$
- removendo x_n, obtemos uma sequência menor X'
- usando a h.i., obtemos uma SCM $Y' = x_i, x_{i+1}, \ldots, x_j$ de X'

Hipótese

Hipótese de indução

Sabemos calcular a SCM de sequências de comprimento $n-1$.

- considere uma sequência $X = x_1, x_2, \ldots, x_n$
- removendo x_n, obtemos uma sequência menor X'
- usando a h.i., obtemos uma SCM $Y' = x_i, x_{i+1}, \ldots, x_j$ de X'

Hipótese

Hipótese de indução

Sabemos calcular a SCM de sequências de comprimento $n-1$.

- considere uma sequência $X = x_1, x_2, \ldots, x_n$
- removendo x_n, obtemos uma sequência menor X'
- usando a h.i., obtemos uma SCM $Y' = x_i, x_{i+1}, \ldots, x_j$ de X'

Passo indutivo

Há três casos a analisar

1. $Y' = \emptyset$: se $x_n \geq 0$, faça $Y = x_n$, senão faça $Y = \emptyset$
2. $j = n - 1$: se $x_n \geq 0$, faça $Y = Y' \| x_n$, senão faça $Y = Y'$
3. $j < n - 1$: consideramos dois subcasos
   - se Y' for uma SCM de X, então basta fazer $Y = Y'$
   - caso contrário, x_n faz parte de uma SCM de X e fazemos $Y = x_k, x_{k+1}, \ldots, x_n$, para algum $k \leq n - 1$.

Passo indutivo

Há três casos a analisar

1. $Y' = \emptyset$: se $x_n \geq 0$, faça $Y = x_n$, senão faça $Y = \emptyset$
2. $j = n - 1$: se $x_n \geq 0$, faça $Y = Y' \| x_n$, senão faça $Y = Y'$
3. $j < n - 1$: consideramos dois subcasos
   - se Y' for uma SCM de X, então basta fazer $Y = Y'$
   - caso contrário, x_n faz parte de uma SCM de X e fazemos $Y = x_k, x_{k+1}, \ldots, x_n$, para algum $k \leq n - 1$.

Passo indutivo

Há três casos a analisar

1. $Y' = \emptyset$: se $x_n \geq 0$, faça $Y = x_n$, senão faça $Y = \emptyset$
2. $j = n - 1$: se $x_n \geq 0$, faça $Y = Y' \| x_n$, senão faça $Y = Y'$
3. $j < n - 1$: consideramos dois subcasos
   - se Y' for uma SCM de X, então basta fazer $Y = Y'$
   - caso contrário, x_n faz parte de uma SCM de X e fazemos $Y = x_k, x_{k+1}, \ldots, x_n$, para algum $k \leq n - 1$.

Passo indutivo

Há três casos a analisar

1. $Y' = \emptyset$: se $x_n \geq 0$, faça $Y = x_n$, senão faça $Y = \emptyset$
2. $j = n - 1$: se $x_n \geq 0$, faça $Y = Y' \| x_n$, senão faça $Y = Y'$
3. $j < n - 1$: consideramos dois subcasos
   - se Y' for uma SCM de X, então basta fazer $Y = Y'$
   - caso contrário, x_n faz parte de uma SCM de X e fazemos $Y = x_k, x_{k+1}, \ldots, x_n$, para algum $k \leq n - 1$.

Passo indutivo

Há três casos a analisar

1. $Y' = \emptyset$: se $x_n \geq 0$, faça $Y = x_n$, senão faça $Y = \emptyset$
2. $j = n - 1$: se $x_n \geq 0$, faça $Y = Y' \| x_n$, senão faça $Y = Y'$
3. $j < n - 1$: consideramos dois subcasos
   - se Y' for uma SCM de X, então basta fazer $Y = Y'$
   - caso contrário, x_n faz parte de uma SCM de X e fazemos $Y = x_k, x_{k+1}, \ldots, x_n$, para algum $k \leq n - 1$.

Passo indutivo

Há três casos a analisar

1. $Y' = \emptyset$: se $x_n \geq 0$, faça $Y = x_n$, senão faça $Y = \emptyset$
2. $j = n - 1$: se $x_n \geq 0$, faça $Y = Y' \| x_n$, senão faça $Y = Y'$
3. $j < n - 1$: consideramos dois subcasos
   - se Y' for uma SCM de X, então basta fazer $Y = Y'$
   - caso contrário, x_n faz parte de uma SCM de X e fazemos $Y = x_k, x_{k+1}, \ldots, x_n$, para algum $k \leq n - 1$.

Refletindo

- podemos executar os dois primeiros casos rapidamente
- mas para o último caso, a h.i. não fornece o valor de k
- mas, nesse caso, sabemos que uma SCM é

$$Y = x_k, x_{k+1}, \ldots, x_{n-1}, x_n,$$

- ou seja, Y é um sufixo de X
- basta conhecer também o sufixo de maior soma de X'

Refletindo

- podemos executar os dois primeiros casos rapidamente
- mas para o último caso, a h.i. não fornece o valor de k
- mas, nesse caso, sabemos que uma SCM é

$$Y = x_k, x_{k+1}, \ldots, x_{n-1}, x_n,$$

- ou seja, Y é um sufixo de X
- basta conhecer também o sufixo de maior soma de X'

Refletindo

- podemos executar os dois primeiros casos rapidamente
- mas para o último caso, a h.i. não fornece o valor de k
- mas, nesse caso, sabemos que uma SCM é

$$Y = x_k, x_{k+1}, \ldots, x_{n-1}, x_n,$$

- ou seja, Y é um sufixo de X
- basta conhecer também o sufixo de maior soma de X'

Refletindo

- podemos executar os dois primeiros casos rapidamente
- mas para o último caso, a h.i. não fornece o valor de k
- mas, nesse caso, sabemos que uma SCM é

$$Y = x_k, x_{k+1}, \ldots, x_{n-1}, x_n,$$

- ou seja, Y é um sufixo de X
- basta conhecer também o sufixo de maior soma de X'

Refletindo

- podemos executar os dois primeiros casos rapidamente
- mas para o último caso, a h.i. não fornece o valor de k
- mas, nesse caso, sabemos que uma SCM é

$$Y = x_k, x_{k+1}, \ldots, x_{n-1}, x_n,$$

- ou seja, Y é um sufixo de X
- basta conhecer também o sufixo de maior soma de X'

Hipótese de indução reforçada

Hipótese de indução reforçada

Sabemos calcular a SCM e o sufixo de maior soma de sequências de comprimento $n - 1$.

- consideramos também sequência de tamanho $n = 0$
- nesse caso, o sufixo de maior soma e a SCM são vazias

Hipótese de indução reforçada

Hipótese de indução reforçada

Sabemos calcular a SCM e o sufixo de maior soma de sequências de comprimento $n-1$.

- consideramos também sequência de tamanho $n = 0$
- nesse caso, o sufixo de maior soma e a SCM são vazias

Hipótese de indução reforçada

Hipótese de indução reforçada

Sabemos calcular a SCM e o sufixo de maior soma de sequências de comprimento $n-1$.

- consideramos também sequência de tamanho $n = 0$
- nesse caso, o sufixo de maior soma e a SCM são vazias

Entrada e saída do subproblema

Subproblema

Entrada:

- um inteiro n
- n números reais $X = x_1, x_2, \ldots, x_n$

Saída:

- inteiros i, j, k tais que
  - $Y = x_i, x_{i+1}, \ldots, x_j$ é uma SCM de X
  - $S = x_k, x_{k+1}, \ldots, x_n$ é um sufixo de soma máxima de X
- números reais $MaxSeq, MaxSuf$ tais que
  - a soma de Y é $MaxSeq$
  - a soma de S é $MaxSuf$

Entrada e saída do subproblema

Subproblema

Entrada:

- um inteiro n
- n números reais $X = x_1, x_2, \ldots, x_n$

Saída:

- inteiros i, j, k tais que
  - $Y = x_i, x_{i+1}, \ldots, x_j$ é uma SCM de X
  - $S = x_k, x_{k+1}, \ldots, x_n$ é um sufixo de soma máxima de X
- números reais $MaxSeq, MaxSuf$ tais que
  - a soma de Y é $MaxSeq$
  - a soma de S é $MaxSuf$

Entrada e saída do subproblema

Subproblema

Entrada:

- um inteiro n
- n números reais $X = x_1, x_2, \ldots, x_n$

Saída:

- inteiros i, j, k tais que
  - $Y = x_i, x_{i+1}, \ldots, x_j$ é uma SCM de X
  - $S = x_k, x_{k+1}, \ldots, x_n$ é um sufixo de soma máxima de X
- números reais $MaxSeq, MaxSuf$ tais que
  - a soma de Y é $MaxSeq$
  - a soma de S é $MaxSuf$

Entrada e saída do subproblema

Subproblema

Entrada:

- um inteiro n
- n números reais $X = x_1, x_2, \ldots, x_n$

Saída:

- inteiros i, j, k tais que
  - $Y = x_i, x_{i+1}, \ldots, x_j$ é uma SCM de X
  - $S = x_k, x_{k+1}, \ldots, x_n$ é um sufixo de soma máxima de X
- números reais $MaxSeq, MaxSuf$ tais que
  - a soma de Y é $MaxSeq$
  - a soma de S é $MaxSuf$

Algoritmo

```
SCM(X, n):
 1  se n = 0
 2      i, j, k ← 1, MaxSeq, MaxSuf ← 0
 3  senão
 4      i, j, k, MaxSeq, MaxSuf ← SCM(X, n − 1)
 5      MaxSuf ← MaxSuf + x_n
 6      se MaxSuf < 0
 7          k ← n + 1, MaxSuf ← 0
 8      se MaxSuf > MaxSeq
 9          i ← k, j ← n, MaxSeq ← MaxSuf
10  devolva i, j, k, MaxSeq, MaxSuf
```

Complexidade

O tempo de execução $T(n)$ de SCM é

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução desta recorrência é

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$

Complexidade

O tempo de execução $T(n)$ de SCM é

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução desta recorrência é

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$

Complexidade

O tempo de execução $T(n)$ de SCM é

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução desta recorrência é

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$

Complexidade

O tempo de execução $T(n)$ de SCM é

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n > 1. \end{cases}$$

A solução desta recorrência é

$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n).$$