Projeto e Análise de Algoritmos*

Projeto de algoritmos por indução e divisão e conquista

Lehilton Pedrosa

Segundo Semestre de 2017

*Criado por C. de Souza, C. da Silva, O. Lee, F. Miyazawa et al.

Projeto de algoritmos por indução

Projeto de algoritmos por indução

- A seguir, usaremos a técnica de indução para desenvolver algoritmos para certos problemas.
- Isto é, a formulação do algoritmo vai ser análoga ao desenvolvimento de uma demonstração por indução.
- Assim, para resolver o problema P falamos do projeto de um algoritmo em dois passos:
  1. Exibir a resolução de uma ou mais instâncias pequenas de P (casos base);
  2. Exibir como a solução de toda instância de P pode ser obtida a partir da solução de uma ou mais instâncias menores de P.

Projeto de algoritmos por indução

Este processo indutivo resulta em algoritmos recursivos, em que:

- a base da indução corresponde à resolução dos casos base da recursão;
- a aplicação da hipótese de indução corresponde a uma ou mais chamadas recursivas; e
- o passo da indução corresponde ao processo de obtenção da resposta para o caso geral a partir daquelas devolvidas pelas chamadas recursivas.

Projeto de algoritmos por indução

- Um benefício imediato é que o uso (correto) da técnica nos dá uma prova da correção do algoritmo.
- A complexidade do algoritmo resultante é expressa numa recorrência
- Frequentemente o algoritmo é eficiente, embora existam exemplos simples em que isso não acontece.
- Iniciaremos com dois exemplos que usam indução:
  1. cálculo do valor de polinômios e
  2. obtenção de subgrafos maximais com restrições de grau.

Exemplo 1 - Cálculo de polinômios

Problema:

Dada uma sequência de números reais $a_n, a_{n-1}, \ldots, a_1, a_0$*, e um número real* x*, calcular o valor do polinômio*

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

Naturalmente este é um problema bem simples.
Estamos interessados em projetar um algoritmo que faça o menor número de operações aritméticas (multiplicações, principalmente).

Exemplo 1 - Cálculo de polinômios

Problema:

Dados uma sequência de números reais $a_n, a_{n-1}, \ldots, a_1, a_0$, e um número real x, calcular o valor do polinômio

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

Hipótese de indução : (primeira tentativa)

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

- **Caso base**: $n = 0$. A solução é a_0.
- Para calcular $P_n(x)$, basta calcular x^n, multiplicar o resultado por a_n e somar o resultado com $P_{n-1}(x)$.

Exemplo 1 - Solução 1 - Algoritmo

```
CÁLCULO-POLINÔMIO(A,x)
▷ Entrada: Coeficientes A = a_n, a_{n-1}, ..., a_1, a_0 e real x.
▷ Saída: O valor de P_n(x).
1   se n = 0 então P ← a_0
2   senão
3       A' ← a_{n-1}, ..., a_1, a_0
4       P' ← CÁLCULO-POLINÔMIO(A', x)
5       xn ← 1
6       para i ← 1 até n faça  xn ← xn * x
7       P ← P' + a_n * xn
8   devolva (P)
```

Exemplo 1 - Solução 1 - Complexidade

Chamando de $T(n)$ o número de operações aritméticas realizadas pelo algoritmo, temos a seguinte recorrência para $T(n)$:

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

Não é difícil ver que

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

Nota: o número de multiplicações pode ser diminuído calculando x^n com o algoritmo de exponenciação rápida que veremos mais tarde.

Segunda solução indutiva

- Desperdício cometido na primeira solução: recálculo de potências de x.
- Alternativa: eliminar essa computação desnecessária trazendo o cálculo de x^{n-1} para dentro da hipótese de indução.

Hipótese de indução reforçada:

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

- Então, no passo de indução, primeiro calculamos x^n multiplicando x por x^{n-1}, conforme exigido na hipótese. Em seguida, calculamos $P_n(x)$ multiplicando x^n por a_n e somando o valor obtido com $P_{n-1}(x)$.
- Note que para o caso base $n = 0$, a solução agora é $(a_0, 1)$.

Exemplo 1 - Solução 2 - Algoritmo

CÁLCULO-POLINÔMIO(A, x)

▷ **Entrada:** Coeficientes $A = a_n, a_{n-1}, \ldots, a_1, a_0$ e real x.

▷ **Saída:** O valor de $P_n(x)$ e o valor de x^n.

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

Exemplo 1 - Solução 2 - Complexidade

Novamente, se $T(n)$ é o número de operações aritméticas realizadas pelo algoritmo, $T(n)$ é dado por:

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

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

- A escolha de considerar o polinômio $P_{n-1}(x)$ na hipótese de indução não é a única possível.
- Podemos **reforçar** ainda mais a h.i. e ter um ganho de complexidade:

Hipótese de indução mais reforçada:

Sabemos calcular o valor do polinômio $P'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} \ldots + a_1$.

- Note que $P_n(x) = xP'_{n-1}(x) + a_0$. Assim, bastam uma multiplicação e uma adição para obtermos $P_n(x)$ a partir de $P'_{n-1}(x)$.
- O caso base é trivial pois, para $n = 0$, a solução é a_0.

Exemplo 1 - Solução 3 - Algoritmo

CÁLCULO-POLINÔMIO(A, x)

▷ **Entrada:** Coeficientes $A = a_n, a_{n-1}, \ldots, a_1, a_0$ e real x.

▷ **Saída:** O valor de $P_n(x)$.

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

Exemplo 1 - Solução 3 - Complexidade

Temos que $T(n)$ é dado por

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

A solução é

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

Essa forma de calcular $P_n(x)$ é chamada de **regra de Horner**.

Projeto por indução - Exemplo 2
Subgrafos Maximais

- Suponha que você esteja planejando uma festa onde queira maximizar as interações entre as pessoas. Uma festa animada, mas sem recursos ilícitos para aumentar a animação.
- Uma das maneiras de conseguir isso é fazer uma lista dos possíveis convidados e, para cada um desses, a lista dos convidados com quem ele(a) interagiria durante a festa.
- Em seguida, é só localizar o maior subgrupo de convidados que interagiriam com, no mínimo, digamos, k outros convidados dentro do subgrupo.

Exemplo 2 - Subgrafos Maximais

Formulação do problema usando grafos:

Dado um grafo simples G (não direcionado) com n vértices e um inteiro $k \leq n$, encontrar em G um subgrafo induzido H com o número máximo de vértices, tal que o grau de cada vértice de H seja $\geq k$. Se um tal subgrafo H não existir, o algoritmo deve identificar esse fato.

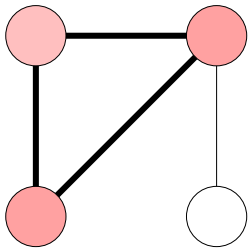
Definições:

H é um subgrafo induzido de um grafo G, uma aresta de G está em H se, e somente se, tem ambos os seus extremos em H.

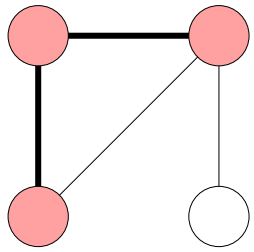
Um subgrafo qualquer G' de G é maximal com relação a uma propriedade P se G' satisfizer P e não existir em G outro subgrafo próprio que contenha G' e satisfaça P.

Exemplo 2 (cont.)

- Exemplo de um subgrafo que é induzido e de outro subgrafo que não é induzido em um grafo G:



INDUZIDO



NÃO INDUZIDO

- Como exemplo de subgrafos maximais (propriedade relativa à continência) que não são máximos (propriedade relativa ao tamanho) em um grafo G, veja definição de clique em grafos.

- No nosso exemplo, o conceito de *maximal* é equivalente ao de *máximo* (número de vértices).

  Você pode provar esta afirmativa?

Exemplo 2 - Indução

Usando o método indutivo:

Hipótese de indução:

Dados inteiros n, k e um grafo G com menos que n vértices, sabemos como encontrar um subgrafo maximal H de G com grau mínimo $\geq k$.

Caso base: o primeiro valor de n para o qual faz sentido buscarmos tais subgrafos H é $n = k + 1$, caso contrário não há como satisfazer a restrição de grau mínimo.

Assim, quando $n = k + 1$, a única forma de existir em G um subgrafo induzido com grau mínimo k é que **todos** os vértices tenham grau igual a k.

Se isso ocorrer, a resposta é $H = G$; caso contrário $H = \emptyset$.

Exemplo 2 - Indução (cont.)

- Seja então G um grafo com n vértices e $k \geq 0$ um inteiro tal que $n > k + 1$.
- Se todos os vértices de G têm grau $\geq k$ não há nada mais a fazer. Devolva $H = G$.
- Caso contrário, seja v um vértice com grau $< k$. É fácil ver que nenhum subgrafo que é solução para o problema conterá v. Assim, v pode ser removido e a hipótese de indução aplicada ao grafo resultante G'.
- Seja H' o subgrafo devolvido pela aplicação da h.i. em G'. Então H' também é resposta para G. ■

Exemplo 2 - Algoritmo

```
SUBGRAFO-MAXIMAL(G, k)
▷ Entrada: Grafo G com n vértices e um inteiro k ≥ 0.
▷ Saída: Subgrafo induzido H de G com grau mínimo k.
1  se (n < k + 1) então H ← ∅
2     senão se todo vértice de G tem grau ≥ k
3        então H ← G
4        senão
5           seja v um vértice de G com grau < k
6           H ← SUBGRAFO-MAXIMAL(G − v, k)
7     devolva H
```

Projeto por indução - Exemplo 3
Fatores de balanceamento em árvores binárias

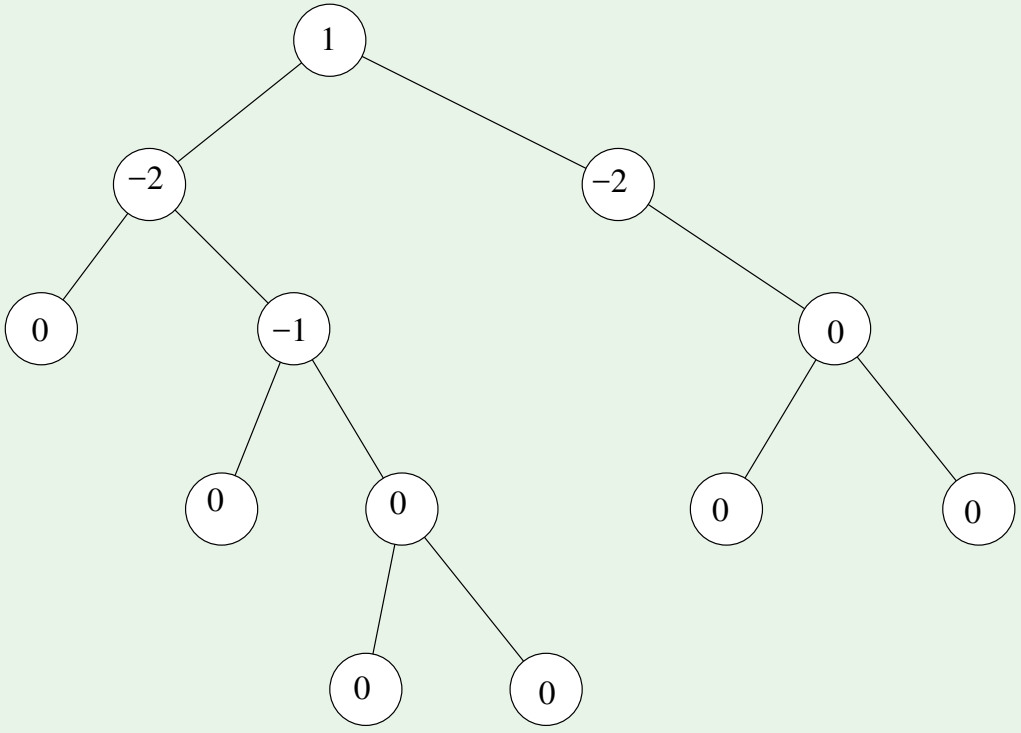
Definição:

Árvores binárias balanceadas são estruturas de dados que minimizam o tempo de busca de informações nela armazenadas. A ideia é que, para todo nó v da árvore, o *fator de balanceamento (f.b.)* de v, isto é, a diferença entre a altura da subárvore esquerda e a altura da subárvore direita de v não desvie muito de zero.

- Convenciona-se que a árvore vazia tem fator de balanceamento *zero* e altura igual a -1.
- Árvores AVL são exemplos de árvores binárias balanceadas, em que o f.b. de cada nó é $-1, 0$ ou $+1$.

Exemplo 3 (cont.)

Exemplo de uma árvore binária e os fatores de balanceamento de seus nós.



Exemplo 3 (cont.)

Problema:

Dada uma árvore binária A com n nós, calcular os 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.

- **Caso base**: quando $n = 0$, convencionamos que o f.b. é igual a zero.
- Vamos mostrar agora como usar a hipótese para calcular f.b.s de uma árvore A com exatamente n nós.

Exemplo 3 - indução (cont.)

A ideia é aplicar a h.i. às subárvores esquerda e direita da raiz e, em seguida, calcular o f.b. da raiz.

Dificuldade: o f.b. da raiz depende das alturas das subárvores esquerda e direita e não dos seus f.b.s.

Conclusão: é necessário 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.

Exemplo 3 - indução (cont.)

- Novamente, o caso base $n = 0$ é fácil pois, por convenção, o f.b. é nulo e a altura igual a -1.
- Seja A uma árvore binária com n nós, para um $n > 0$ qualquer.

  Sejam (f_e, h_e) e (f_d, h_d) os f.b.s e alturas das subárvores esquerda (A_e) e direita (A_d) de A que, por h.i., sabemos como calcular.

  Então, o f.b. da raiz de A é $h_e - h_d$ e a altura da árvore é $\max(h_e, h_d) + 1$.

  Isto completa o cálculo dos f.b.s e da altura de A. ■

De novo, o fortalecimento da hipótese tornou a resolução do problema mais fácil.

Exemplo 3 - Algoritmo

```
FatorAltura (A)
▷ Entrada: Uma árvore binária A com n ≥ 0 nós
▷ Saída: A árvore A os f.b.s nos seus nós e a altura de A
1   se n = 0 então h ← −1
2   senão
3       seja r a raiz de A
3       h_e ← FatorAltura(A_e)
4       h_d ← FatorAltura(A_d)
5       ▷ armazena o f.b. na raiz em r.f
6       r.f ← h_e − h_d
7       h ← max(h_e, h_d) + 1
8   devolva h
```

Exemplo 3 - Complexidade

Seja $T(n)$ o número de operações executadas pelo algoritmo para calcular os f.b.s e a altura de uma árvore A de n nós. Então

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

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

Exemplo 3 - Complexidade

O pior caso da recorrência parece ser quando, ao longo da recursão, um de n_e, n_d é sempre igual a zero (e o outro é sempre igual a $n-1$). Isto é, quando $T(n) = T(n-1) + T(0) + \Theta(1)$.

Chega-se então a:

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

Exercício:

Há algum ganho em complexidade quando ambos n_e e n_d são aproximadamente $n/2$ ao longo da recursão?

Projeto por Indução - Exemplo 4: o problema da celebridade

Definição

Num conjunto S de n pessoas, uma *celebridade* é alguém que é conhecido por todas as pessoas de S mas que não conhece ninguém. (Celebridades são pessoas de difícil convívio!).

Note que pode existir no máximo uma celebridade em S!

Problema:

Queremos saber se existe uma celebridade em S.

Projeto por Indução - Exemplo 4: o problema da celebridade

Vamos formalizar melhor: para um conjunto de n pessoas, associamos uma matriz $n \times n$ M tal que $M[i,j] = 1$ se a pessoa i conhece a pessoa j e $M[i,j] = 0$ caso contrário.

Problema:

Dado um conjunto de n pessoas e a matriz associada M encontrar (se existir) uma celebridade no conjunto.
Isto é, determinar um k tal que todos os elementos da coluna k (exceto $M[k,k]$) são 1s e todos os elementos da linha k (exceto $M[k,k]$) são 0s.

Existe uma solução simples mas laboriosa: para cada pessoa i, verifique todos os outros elementos da linha i e da coluna i. O custo dessa solução é $2n(n-1)$.

Exemplo 4 - Indução

Um argumento indutivo que parece ser mais eficiente é o seguinte:

Hipótese de Indução:

Sabemos encontrar uma celebridade (se existir) em um conjunto de $n-1$ pessoas.

- Se $n = 1$, podemos considerar que o único elemento é uma celebridade.
- Outra opção seria considerarmos o caso base como $n = 2$, o primeiro caso interessante.
  A solução é simples: existe uma celebridade se, e somente se, $M[1,2] \oplus M[2,1] = 1$. Mais uma comparação define a celebridade: se $M[1,2] = 0$, então a celebridade é a pessoa 1; se não, é a pessoa 2.

Exemplo 4 - Indução (cont.)

Tome então um conjunto $S = \{1, 2, \ldots, n\}, n > 2$, de pessoas e a matriz M associada. Considere o conjunto $S' = S \setminus \{n\}$;

Há dois casos possíveis:

1. Existe uma celebridade em S', digamos a pessoa k; então, k é celebridade em S se, e somente se, $M[n, k] = 1$ e $M[k, n] = 0$.
2. Se não existir celebridade em S', então a pessoa n é celebridade em S se $M[n, j] = 0$ e $M[j, n] = 1, \forall j < n$; caso contrário não há celebridade em S.

Essa primeira tentativa, infelizmente, também conduz a um algoritmo quadrático. Por quê?

Exemplo 4 - Segunda tentativa

A segunda tentativa baseia-se em um fato muito simples:

Dadas duas pessoas i e j, é possível determinar se uma delas ***não*** *é uma celebridade com apenas uma comparação: se $M[i,j] = 1$, então i não é celebridade; caso contrário j não é celebridade.*

Vamos usar esse argumento aplicando a hipótese de indução sobre o conjunto de $n-1$ pessoas obtidas removendo dentre as n uma pessoa que sabemos não ser celebridade.

- O caso base e a hipótese de indução são os mesmos que anteriormente.

Exemplo 4 - Segunda tentativa

Tome então um conjunto arbitrário de $n > 2$ pessoas e a matriz M associada.

Sejam i e j quaisquer duas pessoas e suponha que, usando o argumento acima, determinemos que j não é celebridade.

Seja $S' = S \setminus \{j\}$ e considere os dois casos possíveis:

1. Existe uma celebridade em S', digamos a pessoa k. Se $M[j,k] = 1$ e $M[k,j] = 0$, então k é celebridade em S; caso contrário não há uma celebridade em S.
2. Não existe uma celebridade em S'; então não existe uma celebridade em S.

Exemplo 4 - Algoritmo

```
CELEBRIDADE(S, M)
▷ Entrada: conjunto de pessoas S = {1, 2, ..., n};
           M, a matriz que define quem conhece quem em S.
▷ Saída: Um inteiro k ≤ n que é celebridade em S ou k = 0
1   se |S| = 1 então k ← elemento em S
2   senão
3       sejam i, j quaisquer duas pessoas em S
4       se M[i, j] = 1 então s ← i senão s ← j
5       S' ← S \ {s}
6       k ← CELEBRIDADE(S', M)
7       se k > 0 então
8           se (M[s, k] ≠ 1) ou (M[k, s] ≠ 0) então k ← 0
9   devolva k
```

Exemplo 4 - Complexidade

O algoritmo resultante tem complexidade linear em n.

A recorrência $T(n)$ para o número de operações executadas pelo algoritmo é:

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

A solução desta recorrência é

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

Projeto por indução - Exemplo 5
Subsequência consecutiva máxima (SCM)

Problema:

Dada uma sequência $X = x_1, x_2, \ldots, x_n$ de números reais (não necessariamente positivos), encontrar uma ***subsequência consecutiva*** $Y = x_i, x_{i+1}, \ldots, x_j$ *de* X*, onde* $1 \leq i, j \leq n$*, cuja soma seja máxima dentre todas as subsequências consecutivas.*

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: $Y = [\,]$

Exemplo 5 - Indução

Como antes, vamos examinar o que podemos obter de uma **hipótese de indução** simples:

Hipótese de indução:

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

- Seja então $X = x_1, x_2, \ldots, x_n$ uma sequência qualquer de comprimento $n > 1$.
- Considere a sequência X' obtida de X removendo-se x_n.
- Seja $Y' = x_i, x_{i+1}, \ldots, x_j$ a SCM de X', obtida aplicando-se a h.i.

Exemplo 5 - Indução

Há três casos a examinar:

1. $Y' = [\quad]$. Neste caso, $Y = x_n$ se $x_n \geq 0$ ou $Y = [\quad]$ se $x_n < 0$.
2. $j = n - 1$. Como no caso anterior, temos $Y = Y' \| x_n$ se $x_n \geq 0$ ou $Y = Y'$ se $x_n < 0$.
3. $j < n - 1$. Aqui há dois subcasos a considerar:

   3.1 Y' também é SCM de X; isto é, $Y = Y'$.

   3.2 Y' não é a SCM de X. Isto significa que x_n é parte de uma SCM Y de X. Esta tem que ser da forma $x_k, x_{k+1}, \ldots, x_{n-1}, x_n$, para algum $k \leq n - 1$.

Exemplo 5 - Indução

- A hipótese de indução nos permite resolver todos os casos anteriores, exceto o último.
  Não há informação suficiente na h.i. para permitir a resolução deste caso.

  O que falta na h.i.?
- É evidente que, quando

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

  então $x_k, x_{k+1}, \ldots, x_{n-1}$ é um *sufixo* de X' de soma máxima entre os *sufixos* de X'.
- Assim, se conhecermos o sufixo máximo de X', além da SCM, teremos resolvido o problema completamente para X.

Exemplo 5 - Indução

Parece então natural enunciar a seguinte h.i. fortalecida:

Hipótese de indução reforçada:

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

É clara desta discussão também, a base da indução: para $n = 1$, a SCM de $X = x_1$ é x_1 caso $x_1 \geq 0$, e a sequência vazia caso contrário. Nesse caso, o sufixo máximo é igual a SCM.

Exemplo 5 - Algoritmo

SCM(X, n)

▷ **Entrada:** um inteiro n e uma sequência de n números reais $X = [x_1, x_2, \ldots, x_n]$.

▷ **Saída:** Inteiros i, j, k e reais $MaxSeq, MaxSuf$ tais que:

- x_i, x_j são o primeiro e último elementos da SCM de X, cujo valor é $MaxSeq$; e
- x_k é o primeiro elemento do sufixo máximo de X, cujo valor é $MaxSuf$.
- O valor $j = 0$ significa que X é composta de negativos somente. Neste caso, convencionamos $MaxSeq = 0$.
- O valor $k = 0$ significa que o sufixo máximo de X é vazio. Neste caso, $MaxSuf = 0$.

Exemplo 5 - Algoritmo (cont.)

```
SCM(X, n): (cont.)
1   se n = 1
2   então
3       se x_1 < 0
4           então i, j, k ← 0; MaxSeq, MaxSuf ← 0
5           senão i, j, k ← 1; MaxSeq, MaxSuf ← x_1
6   senão
7       (i, j, k, MaxSeq, MaxSuf) ← SCM(X, n − 1)
8       se MaxSuf = 0 então k ← n
9       MaxSuf ← MaxSuf + x_n
10        se MaxSuf > MaxSeq
11            então i ← k; j ← n; MaxSeq ← MaxSuf
12            senão se MaxSuf < 0 então MaxSuf ← 0; k ← 0
13    devolva (i, j, k, MaxSeq, MaxSuf)
```

Exemplo 5 - Complexidade

A complexidade $T(n)$ de SCM é simples de ser calculada.

Como no último exemplo,

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

A solução desta recorrência é

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

Reforçar a hipótese de indução: preciso lembrar disso

Projeto por indução - Erros comuns

Os erros discutidos nas provas por indução, naturalmente, traduzem-se em erros no projeto de um algoritmo por indução.

Exemplo:

Problema: *Dado um grafo conexo G, verificar se G é bipartido ou não. Caso seja, devolver a partição dos vértices.*

Definição:

Um grafo é *bipartido* se seu conjunto de vértices pode ser particionado em dois conjuntos, de forma que toda aresta de G tenha extremos em conjuntos diferentes.

Teorema:

Se G é conexo e bipartido, então a bipartição é única.

Projeto por indução - Erros comuns

O que há de **errado** com o seguinte (esboço de um) algoritmo recursivo para verificar se um grafo conexo é bipartido?

- Sejam G um grafo conexo, v um vértice de G e considere o grafo $G' = G - \{v\}$.
- Se G' não for bipartido, então G também não é. Caso contrário, sejam A e B os dois conjuntos da bipartição de G' obtidos recursivamente.
- Considere agora o vértice v e sua relação com os vértices de A e B.
- Se v tiver um vizinho em A e outro em B, então G não é bipartido (já que a bipartição, se existir, deve ser única).
- Caso contrário, adicione v a A ou B, o conjunto no qual v não tem vizinhos. A bipartição está completa.

Divisão e Conquista

Projeto de Algoritmos por Divisão e Conquista

- **Revisando:** Um algoritmo de divisão e conquista resolve um problema:
  - obtendo soluções para subproblemas recursivamente;
  - combinando as soluções parciais de (um ou mais)
- É mais um paradigma de projeto de algoritmos baseado no princípio da indução.
- Informalmente podemos dizer que:
  - o paradigma incremental utiliza indução fraca
  - o paradigma de divisão e conquista utiliza indução forte
- É natural, portanto, demonstrar a correção de algoritmos de divisão e conquista por indução.

Algoritmo Genérico

```
DIVISAO-CONQUISTA(x)
▷ Entrada: A instância x
▷ Saída: Solução y do problema em questão para x
1   se x é suficientemente pequeno então
      ▷ Solucao(x) algoritmo para pequenas instâncias
2       retorne Solucao(x)
3   senão
        ▷ divisão
4       decomponha x em instâncias menores x_1, x_2, ..., x_k
5       para i de 1 até k faça y_i := DIVISAO-CONQUISTA(x_i)
        ▷ conquista
6       combine as soluções y_i para obter a solução y de x.
7   retorne (y)
```

Projeto por Divisão e Conquista - Exemplo 1
Exponenciação

Problema:

Calcular a^n, para todo real a e inteiro $n \geq 0$.

Primeira solução, por indução fraca:

- **Caso base:** $n = 0$; $a^0 = 1$.
- **Hipótese de indução:** *Suponha que, para qualquer inteiro* $n > 0$ *e real* a*, sei calcular* a^{n-1}.
- **Passo da indução:** Queremos provar que conseguimos calcular a^n, para $n > 0$. Por hipótese de indução, sei calcular a^{n-1}. Então, calculo a^n multiplicando a^{n-1} por a.

Exemplo 1 - Solução 1 - Algoritmo

```
EXPONENCIACAO(a, n)
▷ Entrada: A base a e o expoente n.
▷ Saída: O valor de a^n.
1   se n = 0 então
2       retorne (1) {caso base}
3   senão
4       an' := EXPONENCIACAO(a, n − 1)
5       an := an' * a
6   retorne (an)
```

Exemplo 1 - Solução 1 - Complexidade

Seja $T(n)$ o número de operações executadas pelo algoritmo para calcular a^n.

Então a relação de recorrência deste algoritmo é:

$$T(n) = \begin{cases} c_1, & n = 0 \\ T(n-1) + c_2, & n > 0, \end{cases}$$

onde c_1 e c_2 representam, respectivamente, o tempo (constante) executado na atribuição da base e multiplicação do passo.

Neste caso, não é difícil ver que

$$T(n) = c_1 + \sum_{i=1}^{n} c_2 = c_1 + nc_2 = \Theta(n).$$

Este algoritmo é linear no tamanho da entrada?

Exemplo 1 - Solução 2 - Divisão e Conquista

Vamos agora projetar um algoritmo para o problema usando indução forte de forma a obter um algoritmo de divisão e conquista.

Segunda solução, por indução forte:

- **Caso base:** $n = 0$; $a^0 = 1$.
- **Hipótese de indução:** *Suponha que, para qualquer inteiro $n > 0$ e real a, sei calcular a^k, para todo $k < n$.*
- **Passo da indução:** Queremos provar que conseguimos calcular a^n, para $n > 0$. Por hipótese de indução sei calcular $a^{\lfloor \frac{n}{2} \rfloor}$. Então, calculo a^n da seguinte forma:

$$a^n = \begin{cases} \left(a^{\lfloor \frac{n}{2} \rfloor}\right)^2, & \text{se } n \text{ par;} \\ a \cdot \left(a^{\lfloor \frac{n}{2} \rfloor}\right)^2, & \text{se } n \text{ ímpar.} \end{cases}$$

Exemplo 1 - Solução 2 - Algoritmo

```
EXPONENCIACAO-DC(a, n)
▷ Entrada: A base a e o expoente n.
▷ Saída: O valor de a^n.
1   se n = 0 então
2       retorne (1) {caso base}
3   senão
            ▷ divisão
4       an' := EXPONENCIACAO-DC(a, n div 2)
            ▷ conquista
5       an := an' * an'
6       se (n mod 2) = 1
7           an := an * a
8   retorne (an)
```

Exemplo 1 - Solução 2 - Complexidade

- Seja $T(n)$ o número de operações executadas pelo algoritmo de divisão e conquista para calcular a^n.
- Então a relação de recorrência deste algoritmo é:

$$T(n) = \begin{cases} c_1, & n = 0 \\ T(\lfloor \frac{n}{2} \rfloor) + c_2, & n > 0, \end{cases}$$

- Não é difícil ver que $T(n) \in \Theta(\log n)$. Por quê?

Projeto por Divisão e Conquista - Exemplo 2
Busca Binária

Problema:

Dado um vetor ordenado A com n números reais e um real x, determinar a posição $1 \leq i \leq n$ tal que $A[i] = x$, ou que não existe tal i.

- O projeto de um algoritmo para este problema usando indução simples, nos leva a um algoritmo incremental de complexidade de pior caso $\Theta(n)$. Pense em como seria a indução!
- Se utilizarmos indução forte para projetar o algoritmo, podemos obter um algoritmo de divisão e conquista que nos leva ao algoritmo de busca binária. Pense na indução!
- Como o vetor está ordenado, conseguimos determinar, com apenas uma comparação, que *metade* das posições do vetor não pode conter o valor x.

Exemplo 2 - Algoritmo

```
BUSCA-BINARIA(A, e, d, x)
▷ Entrada: Vetor A, delimitadores e e d do subvetor e x.
▷ Saída: Índice 1 ≤ i ≤ n tal que A[i] = x ou i = 0.
 1   se e = d então se A[e] = x então retorne (e)
 2           senão retorne (0)
 3   senão
 4       i := (e + d) div 2
 5       se A[i] = x retorne (i)
 6       senão se A[i] > x
 7           i := BUSCA-BINARIA(A, e, i − 1, x)
 8           senão {A[i] < x}
 9           i := BUSCA-BINARIA(A, i + 1, d, x)
10   retorne (i)
```

Exemplo 2 - Complexidade

- O número de operações $T(n)$ executadas na busca binária no pior caso é:

$$T(n) = \begin{cases} c_1, & n = 1 \\ T(\lceil \frac{n}{2} \rceil) + c_2, & n > 1, \end{cases}$$

- Não é difícil ver que $T(n) \in \Theta(\log n)$. Por quê?
- O algoritmo de busca binária (divisão e conquista) tem complexidade de pior caso $\Theta(\log n)$, que é assintoticamente melhor que o algoritmo de busca linear (incremental).
- E se o vetor não estivesse ordenado, qual paradigma nos levaria a um algoritmo assintoticamente melhor?

Um exemplo real

De um aluno de mestrado do IC

Entrada:

- m listas L_l tem n_l inteiros ordenados do menor para maior
- número k

Saída:

- k tuplas $(a_1, a_2, \ldots, a_m)$ com menores somas

OBS: soma de uma tupla $(a_1, a_2, \ldots, a_k)$ é $a_1 + a_2 + \cdots + a_m$.

Um exemplo real - Algoritmo

Algoritmo trivial

1. Enumerar todas as tuplas
2. Ordenar elementos
3. Selecionar k primeiros

- **Complexidade:** $\tilde{O}(n_1 \times n_2 \times \cdots \times n_m)$
- Como fazer melhor? divisão e conquista

Um exemplo real - Divisão e conquista

Estratégia

- Caso base:
  - $m = 1$
  - $m = 2$ também!
- Caso geral:
  - Divisão: dividimos em sub-somas (associativa e comutativa!)
  - Conquista: é o mesmo problema, para $m = 2$!

Algoritmo e análise: no quadro