MO417 — Complexidade de Algoritmos I

Cid Carvalho de Souza   Cândida Nunes da Silva
Orlando Lee

26 de setembro de 2011

Revisado por Zanoni Dias

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 um problema P:
    1. mostramos como resolver instâncias pequenas de P (casos base) e
    2. mostramos como obter uma solução de uma instância de P a partir das soluções de 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 problema original a partir daa respostas devolvidas pelas chamadas recursivas.

Projeto de algoritmos por indução

- Um benefício imediato é que o uso (correto) da técnica nos dá uma prova da corretude do algoritmo.
- A complexidade do algoritmo resultante é expressa numa recorrência.
- Muitas vezes é imediato como converter o algoritmo recursivo em um iterativo.
- Frequentemente o algoritmo é eficiente, embora existam exemplos simples em que isso não acontece.

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

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

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

Cálculo de polinômios - Solução 1

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 $P_{n-1}(x)$.

Cálculo de polinômios - Solução 1

CálculoPolinômio(A,n,x)

$\triangleright$ **Entrada:** Coeficientes $A = a_n, a_{n-1}, \ldots, a_1, a_0$ e real x.
$\triangleright$ **Saída:** O valor de $P_n(x)$.

se $n = 0$ **então** $P \leftarrow a_0$
senão
    $A' \leftarrow a_{n-1}, \ldots, a_1, a_0$
    $P' \leftarrow$ CálculoPolinômio($A', n-1, x$)
    $xn \leftarrow x$
    para $i \leftarrow 2$ **até** n **faça** $xn \leftarrow xn * x$
    $P \leftarrow P' + a_n * xn$
devolva (P)

Cálculo de polinômios - Solução 1

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

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

Não é difícil ver que

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

Observação: esta solução desperdiça muito tempo recalculando x^n. É possível fazer melhor!

Cálculo de polinômios - Solução 2

- 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 $(a_0, 1)$.

Cálculo de polinômios - Solução 2

CálculoPolinômio(A,n,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** $P \leftarrow a_0; xn \leftarrow 1$
2. **senão**
3. $A' \leftarrow a_{n-1}, \ldots, a_1, a_0$
4. $(P', x') \leftarrow$ CálculoPolinômio$(A', n-1, x)$
5. $xn \leftarrow x * x'$
6. $P \leftarrow P' + a_n * xn$
7. **devolva** (P, xn)

Cálculo de polinômios - Solução 2

Se $T(n)$ é o número de operações aritméticas realizadas pelo algoritmo, então temos a recorrência:

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

A solução da recorrência é

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

Cálculo de polinômios - Solução 3

- 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, com apenas uma multiplicação e uma adição podemos calcular $P_n(x)$ a partir de $P'_{n-1}(x)$.
- O caso base é trivial pois, para $n = 0$, a solução é a_0.

Cálculo de polinômios - Solução 3

CálculoPolinômio(A,n,x)

▷ **Entrada:** Coeficientes $A = a_n, a_{n-1}, \ldots, a_1, a_0$ e real x.
▷ **Saída:** O valor de $P_n(x)$.

```
se n = 0 então P ← a_0
senão
    A' ← a_n, a_{n-1}, ..., a_1
    P' ← CálculoPolinômio(A', n − 1, x)
    P ← x ∗ P' + a_0
devolva (P)
```


Cálculo de polinômios - Solução 3

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

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

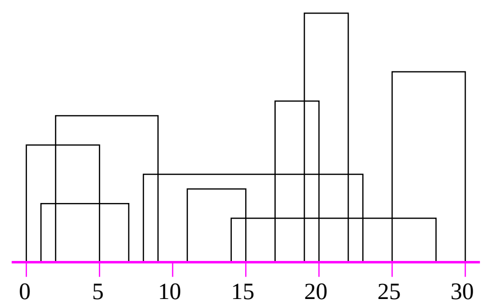
A solução é

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

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

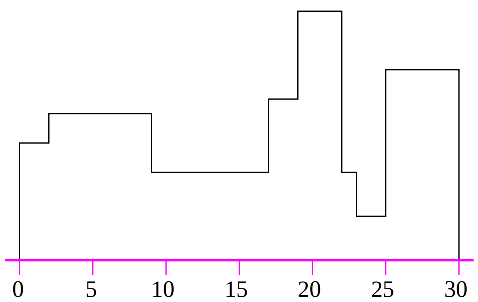
Exemplo 2: o problema do *skyline*

Problema:
Dada uma sequência de triplas (l_i, h_i, r_i) para $i = 1, 2, \ldots, n$ que representam prédios retangulares, determinar a silhueta dos prédios (skyline).

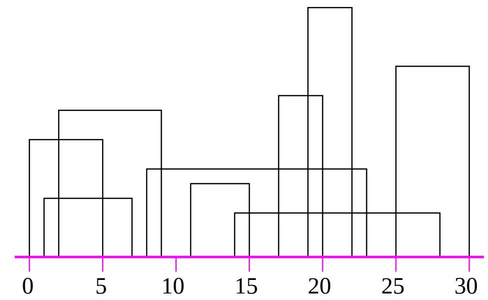


Exemplo 2: o problema do *skyline*

Problema:
Dada uma sequência de triplas (l_i, h_i, r_i) para $i = 1, 2, \ldots, n$ que representam prédios retangulares, determinar a silhueta dos prédios (skyline).



Exemplo 2: o problema do *skyline*

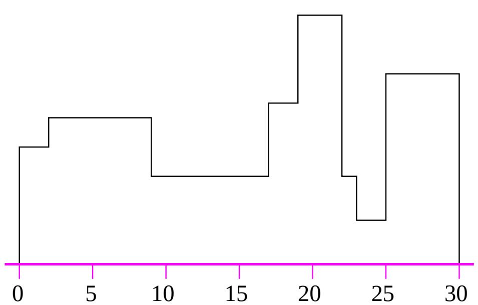


Cada prédio é descrito por uma tripla (l_i, h_i, r_i) onde l_i e r_i são as coordenadas x do prédio e h_i é a altura do prédio.

$(0, 8, 5), (2, 10, 9), (1, 4, 7), (11, 5, 15), (17, 11, 20), (19, 17, 22), (14, 3, 28), (25, 13, 30), (8, 6, 23)$.

Exemplo 2: o problema do *skyline*

A solução do problema (skyline) é uma sequência de coordenadas e alturas ligando os prédios arranjadas da esquerda para a direita.



Skyline:
$(0, \mathbf{8}, 2, \mathbf{10}, 9, \mathbf{6}, 17, \mathbf{11}, 17, \mathbf{11}, 19, \mathbf{17}, 22, \mathbf{6}, 23, \mathbf{3}, 25, \mathbf{13}, 30, \mathbf{0})$.
Os números em **negrito** indicam as alturas.

Skyline - Solução 1

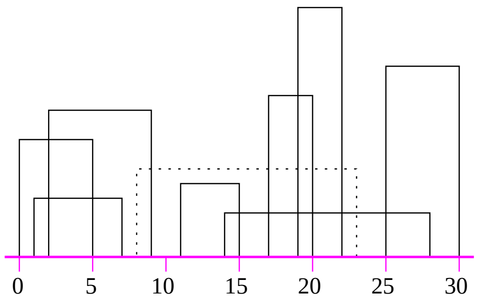
Vamos tentar usar o método de projeto por indução.

Hipótese de indução: (primeira tentativa)
Dada uma sequência de $n - 1$ prédios, sabemos determinar seu skyline.

- O caso base é trivial ($n = 1$).
- Resta saber como obter o skyline dos n prédios a partir do skyline do subproblema.

Skyline - Solução 1

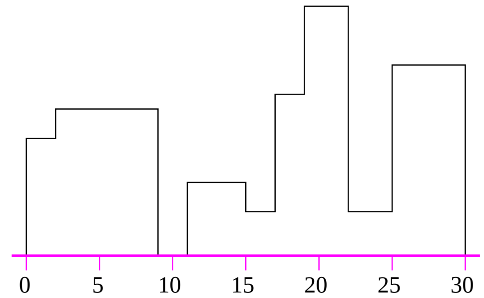
Subproblema obtido removendo-se o prédio $B_n = (8, 6, 23)$:



Prédios do subproblema:
$(0, 8, 5), (2, 10, 9), (1, 4, 7), (11, 5, 15), (17, 11, 20), (19, 17, 22), (14, 3, 28), (25, 13, 30)$.

Skyline - Solução 1

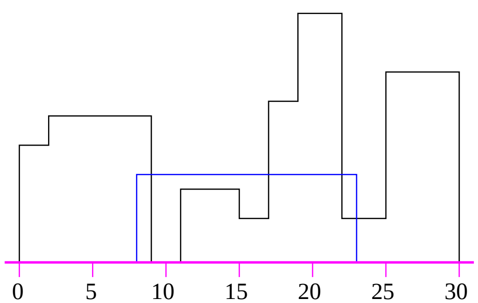
Subproblema obtido removendo-se o prédio $B_n = (8, 6, 23)$:



Skyline do subproblema:
$(0, \mathbf{8}, 2, \mathbf{10}, 9, \mathbf{0}, 11, \mathbf{5}, 15, \mathbf{3}, 17, \mathbf{11}, 19, \mathbf{17}, 22, \mathbf{3}, 25, \mathbf{13}, 30, \mathbf{0})$

Skyline - Solução 1

Para obter a solução do problema original, acrescentamos o prédio $B_n = (8, 6, 23)$ ao skyline do subproblema:

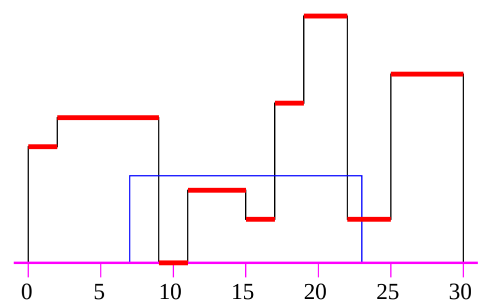


Skyline do subproblema:
$(0, \mathbf{8}, 2, \mathbf{10}, 9, \mathbf{0}, 11, \mathbf{5}, 15, \mathbf{3}, 17, \mathbf{11}, 19, \mathbf{17}, 22, \mathbf{3}, 25, \mathbf{13}, 30, \mathbf{0})$

Skyline - Solução 1

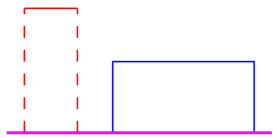
Como atualizar o skyline?

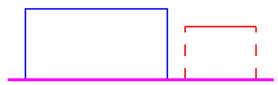
$S = (x_1, h_1, x_2, h_2, \ldots, x_k, h_k)$, $B_n = (l, h, r)$



A idéia é examinar cada segmento $[x_i, x_{i+1}]$ e a cada passo inserir um par **coordenada**, **altura** no skyline final (inicialmente vazio).

Skyline - Solução 1



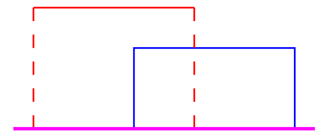


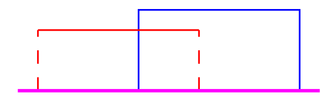
Basta analisar três casos:

(1) $x_{i+1} \leq l$ ou $x_i \geq r$:

- insira x_i, h_i no skyline final.

Skyline - Solução 1

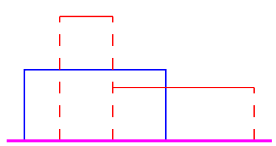


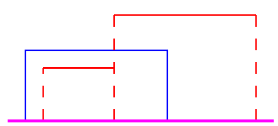


(2) $x_i < l < x_{i+1} \leq r$:

- se $h_i \geq h$ então insira x_i, h_i no skyline final;
- senão, insira x_i, h_i, l, h no skyline final.

Skyline - Solução 1



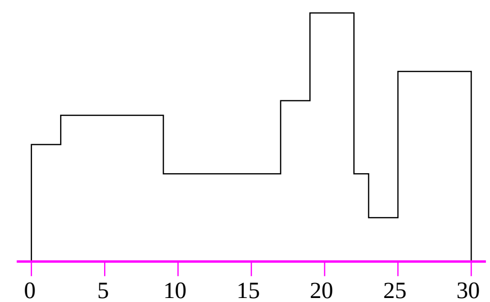


(3) $l \leq x_i < r$:

- se $h_i \geq h$ então insira x_i, h_i no skyline final;
- senão, se $x_{i+1} > r$ então insira r, h_i no skyline final.

Skyline - Solução 1

Usando a idéia descrita podemos obter a solução do problema original.



Skyline:
(0, **8**, 2, **10**, 9, **6**, 17, **11**, 17, **11**, 19, **17**, 22, **6**, 23, **3**, 25, **13**, 30, **0**).

Skyline - Solução 1

Skyline(B,n)

▷ **Entrada:** Prédios $B_i = (l_i, h_i, r_i)$ para $i = 1, 2, \ldots, n$.
▷ **Saída:** O skyline de B.

```
se n = 1 então S ← (l_1, h_1, r_1, 0)
senão
    S ← Skyline(B, n − 1)
    S ← AcrescPredioSkyline(S, B_n)
devolva (S)
```

Exercício. Escreva uma rotina AcrescPredioSkyline que recebe um skyline S e um novo prédio B_n e acrescenta-o a S em **tempo linear**.

Skyline - Solução 1

Chamando de $T(n)$ a complexidade do algoritmo Skyline, temos a seguinte recorrência:

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

A solução da recorrência é $\Theta(n^2)$. Logo, a complexidade do algoritmo Skyline é $\Theta(n^2)$.

É possível fazer melhor que isso?

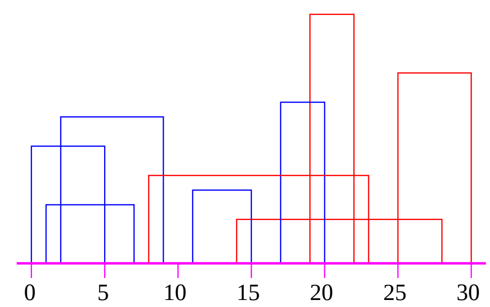
Skyline - Solução 2

Podemos usar **indução forte** e **divisão-e-conquista**.

Hipótese de indução:
Sabemos determinar o skyline de qualquer conjunto com menos que n prédios.

- A base é novamente o caso $n = 1$.
- O **passo de indução** consiste em dividir o problema em dois subproblemas de mesmo tamanho (ou com diferença de 1).
- Resta saber como combinar as soluções dos subproblemas.

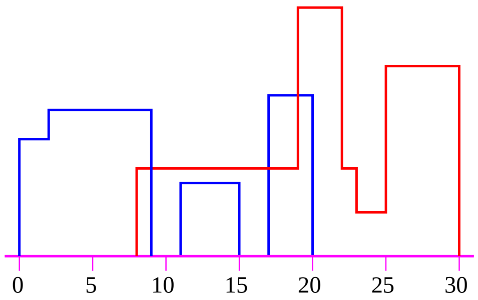
Skyline - Solução 2



Prédios do subproblema 1:
$(0, 8, 5), (2, 10, 9), (1, 4, 7), (11, 5, 15), (17, 11, 20)$.

Prédios do subproblema 2:
$(19, 17, 22), (14, 3, 28), (25, 13, 30), (8, 6, 23)$.

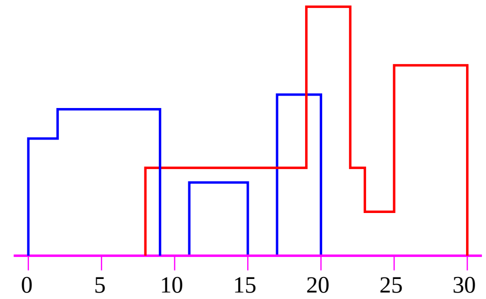
Skyline - Solução 2



Skyline do subproblema 1:
(0, **8**, 2, **10**, 9, **0**, 11, **5**, 15, **0**, 17, **11**, 20, **0**)

Skyline do subproblema 2:
(8, **6**, 19, **17**, 22, **6**, 23, **3**, 25, **13**, 30, **0**)

Skyline - Solução 2



Para obter o skyline do problema original, percorremos os dois skylines da esquerda para a direita e atualizamos a altura sempre que necessário. Isto pode ser feito em tempo $\Theta(n)$ e é similar ao método de intercalação (merge). (Exercício!)

Skyline - Solução 2

Temos a seguinte recorrência para a complexidade de tempo $T(n)$ do algoritmo descrito.

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n), & n > 1. \end{cases}$$

Humm... Já vimos esta recorrência.

A solução da recorrência é $T(n) = \Theta(n \lg n)$ que é assintoticamente melhor que a solução anterior ($\Theta(n^2)$).

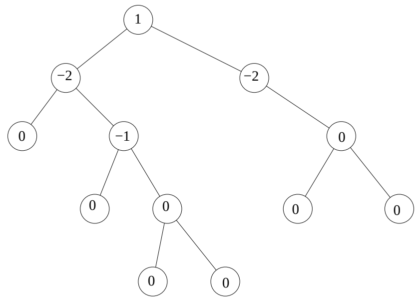
Exemplo 3 - Fatores de balanceamento

Definição:
Considere uma árvore binária. Para todo nó v da árvore, o fator de balanceamento (f.b.) de v é a diferença entre a altura da subárvore esquerda e a altura da subárvore direita de v.

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

Exemplo 3 - Fatores de balanceamento

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



Exemplo 3 - Fatores de balanceamento

Problema:

Dada uma árvore binária A com n nós, calcular os fatores de balanceamento de cada nó de A.

Vamos supor que para cada nó v da árvore há um campo $v.fb$ onde fica armazenado o seu f.b. (a ser calculado).

Fatores de balanceamento

- Vamos projetar o algoritmo indutivamente.

Hipótese de indução:

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

- O caso base ocorre quando $n = 0$. Convencionamos que o f.b. é igual a zero.
- Vamos tentar usar a hipótese de indução para calcular os f.b. dos nós de uma árvore A com exatamente n nós.

Fatores de balanceamento

Aplique a h.i. às subárvores esquerda e direita da raiz e determine os f.b. de seus nós.

Resta apenas 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.

Conclusão: é necessário uma h.i. mais forte!

Hipótese de indução reforçada:

Sabemos como calcular os fatores de balanceamento e alturas dos nós de árvores com menos que n nós.

Fatores de balanceamento

- O caso base $n = 0$ é simples pois, por convenção, o f.b. é igual a 0 e a altura é igual a 0.
- Considere uma árvore binária **A** com $n > 0$ nós. Sejam A_e e A_d as subárvores esquerda e direita de **A**.
- Pela h.i. sabemos calcular (fb_e, h_e) e (fb_d, h_d), os f.b.s e alturas dos nós de A_e e A_d, rspectivamente.

  Então

  f.b. da **raiz** de **A** $= h_e - h_d$ e

  a altura de **A** $= \max(h_e, h_d) + 1$.

O fortalecimento da hipótese de indução tornou a resolução do problema mais fácil.

Fatores de balanceamento

FatorAltura (A)

▷ **Entrada:** Uma árvore binária A com $n \geq 0$ nós
▷ **Saída:** A árvore A os f.b.s nos seus nós e a altura de A

se A for vazia **então** $h \leftarrow 0$
senão
    seja r a raiz de A
    $h_e \leftarrow$ FatorAltura(A_e)
    $h_d \leftarrow$ FatorAltura(A_d)
    ▷ armazena o f.b. na raiz em $r.f$
    $r.fb \leftarrow h_e - h_d$
    $h \leftarrow \max(h_e, h_d) + 1$
devolva h

Fatores de balanceamento

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.

A solução da recorrência é $T(n) = \Theta(n)$. Ela não depende dos valores exatos de n_e e n_d.

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:

Determinar se existe uma celebridade em um conjunto S de n pessoas.

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.
Por convenção, $M[i,i] = 0$ para todo i.

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 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 $2(n-1)n$.

O problema da celebridade

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.

O problema da celebridade

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 três 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. A pessoa n é celebridade em S se $M[n,j] = 0$ e $M[j,n] = 1$ para todo $j < n$.
3. Não existe uma celebridade em S.

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

O problema da celebridade

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 de S uma pessoa que sabemos não ser celebridade.

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

O problema da celebridade

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

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

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 celebridade em S', então não existe uma celebridade em S.

O problema da celebridade

Celebridade(i, j, M)

▷ **Entrada:** dois inteiros i e j e $M_{n \times n}$, a matriz que define quem conhece quem em S.

▷ **Saída:** Um inteiro k, tal que $1 \leq k \leq n$, que identifica a celebridade em S ou $k = 0$, caso ela não exista

1. **se** $i = j$ **então** $k \leftarrow$ i
2. **senão**
3. **se** $M[i,j] = 1$ **então** $i \leftarrow i + 1$; $s \leftarrow i$
4. **senão** $j \leftarrow j - 1$; $s \leftarrow j$
5. $k \leftarrow$ Celebridade(i, j, M)
6. **se** $k > 0$ **então**
7. **se** $(M[s,k] \neq 1)$ **ou** $(M[k,s] \neq 0)$ **então** $k \leftarrow 0$
9. **devolva** k

O problema da celebridade

O algoritmo deve ser chamado da seguinte forma: **Celebridade(1,n,M)**.

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

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

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: $Y = [\,]$

Subsequência consecutiva máxima

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.

Subsequência consecutiva máxima

Há três casos a examinar:

1. $Y' = [\ \ ]$. Neste caso, $Y = x_n$ se $x_n \geq 0$ ou $Y = [\ \ ]$ 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:
   1. Y' também é SCM de X; isto é, $Y = Y'$.
   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 $k \leq n-1$.

Subsequência consecutiva máxima

- A hipótese de indução nos permite resolver diretamente todos os casos anteriores, exceto o último, onde precisamos obter o sufixo de soma máxima de X, que pode ser calculado, no pior caso, em tempo $\Theta(n)$. Sendo assim, é possível resolver o problema em $\Theta(n^2)$ (Exercício).
- No entanto, podemos reforçar a hipótese de indução para permitir resolver diretamente todos os casos.
- **Neste caso, o que podemos acrescentar 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 de soma máxima de X', além da SCM, teremos resolvido o problema completamente para X.

Subsequência consecutiva máxima

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.

Subsequência consecutiva máxima

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 $MaxSeq$.

– x_k é o primeiro elemento do sufixo de soma máxima de X, cujo valor é $MaxSuf$.

– O valor $j = 0$ significa que X é composta somente de negativos. Neste caso, convencionamos $MaxSeq = 0$.

– O valor $k = n + 1$ significa que o sufixo de soma máxima de X é vazio. Neste caso, $MaxSuf = 0$.

Subsequência consecutiva máxima

SCM(X, n): (cont.)

se $n = 1$ **então**
  se $x_1 < 0$
    então $i, j \leftarrow 0; k \leftarrow 2; MaxSeq, MaxSuf \leftarrow 0$
    senão $i, j, k \leftarrow 1; MaxSeq, MaxSuf \leftarrow x_1$
senão
  $(i, j, k, MaxSeq, MaxSuf) \leftarrow$ SCM$(X, n-1)$
  $MaxSuf \leftarrow MaxSuf + x_n$
  se $MaxSuf < 0$ **então** $MaxSuf \leftarrow 0; k \leftarrow n+1$
  se $MaxSuf > MaxSeq$
    então $i \leftarrow k; j \leftarrow n; MaxSeq \leftarrow MaxSuf$
 devolva $(i, j, k, MaxSeq, MaxSuf)$

Subsequência consecutiva máxima

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

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

A solução desta recorrência é

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

Exemplo 6 - o problema da ordenação

Este já é um problema conhecido.

Problema:

Dado um vetor $A[1 \ldots n]$ rearranjar o vetor em ordem crescente.

Vamos rever alguns dos algoritmos vistos para resolver este problema dentro do paradigma de projeto de algoritmos por indução.

Exemplo 6 - o problema da ordenação

Vamos aplicar o método de projeto por indução.

Hipótese de indução:

Sabemos ordenar um vetor com $n-1$ elementos.

- O caso base $n=1$ (ou $n=0$) é trivial.
- Para fazer o passo de indução, aplique a h.i. a $A[1\dots n-1]$. Agora para obter o vetor inteiro ordenado, basta inserir $A[n]$ neste subvetor.
- Este é o algoritmo Ordena-Por-Inserção (InsertionSort) visto anteriormente, mas com uma formulação recursiva.

Exemplo 6 - o problema da ordenação

- A complexidade de tempo $T(n)$ do InsertionSort no pior caso é dada por

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

- A solução da recorrência é $\Theta(n^2)$.
- Este algoritmo é essencialmente equivalente à versão iterativa vista anteriormente. Ela usa o que chamamos de recursão de cauda que pode traduzida facilmente em um algoritmo iterativo.

Exemplo 6 - o problema da ordenação

Outra idéia típica em projeto de indução é tentar remover um elemento apropriado para usar a hipótese de indução.

Hipótese de indução:

Sabemos ordenar um vetor com $n-1$ elementos.

- O caso base $n=1$ é trivial.
- Encontre a posição do máximo de $A[1\dots n]$ e troque-o de lugar com $A[n]$. Isto pode ser feito em tempo $O(n)$. Aplique a h.i a $A[1\dots n-1]$. Agora o vetor $A[1\dots n]$ está ordenado!
- Este é o algoritmo SelectionSort.

Exemplo 6 - o problema da ordenação

- A complexidade de tempo $T(n)$ do SelectionSort é dada por

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

- A solução da recorrência é $\Theta(n^2)$.
- É fácil escrever uma versão iterativa deste algoritmo.
- Não é difícil perceber que a complexidade deste algoritmo poderia ser reduzida se soubéssemos encontrar o máximo mais rapidamente.
- O algoritmo Heapsort que veremos mais tarde faz isso.

Exemplo 6 - o problema da ordenação

- Outro algoritmo de ordenação visto foi o Mergesort que tem complexidade de tempo $O(n \lg n)$.
- O Mergesort é um algoritmo de divisão-e-conquista e ela baseia-se na rotina Intercala.
- Nos algoritmos InsertionSort e Mergesort, o trabalho mais pesado consiste em obter uma solução do problema a partir da solução parcial.
  No algoritmo SelectionSort e no Heapsort, o trabalho mais pesado consiste em encontrar o elemento a ser removido (qual subproblema resolver). Após feito isto, a recursão imediatamente resolve o problema.

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.