SIN110 Algoritmos e Grafos

aula 04

Análise de algoritmos

- recursão e recorrências
- algoritmos recursivos

Técnicas de Projeto de algoritmos

- Projetos de algoritmos por indução

Recursão

Na definição de funções matemáticas, expressões algébricas, tipos de dados, etc. e geralmente são constituídas de duas partes:

- *parte base* e,
- *parte recursiva.*

Recursão - exemplo

```
Busca-binária(x,A,e,d)
se e = d+1
    então devolve -1
    senão  m ← (e+d)/2
           se x = A[m]
               então devolve m
           se x < A[m]
               então devolve Busca-binária(x,A,e,m-1)
               senão devolve Busca-binária(x,A,m+1,d)
```

Para analisar a complexidade e correção de algoritmos que contém uma chamada recursiva, como nos exemplos, precisamos descrever o tempo de execução através de uma *recorrência*.

Recorrências

Recursão

Se o problema é pequeno, resolva-o diretamente, como puder. Se o problema é grande, reduza-o a um problema menor do mesmo tipo.

exemplo:

```
Bbin(x,A,e,d)
se e = d+1
    então devolve -1
    senão m ← ⌊(e+d)/2⌋
            se x = A[m]
           então devolve m
         se x < A[m]
              então devolve Bbin(x,A,e,m-1)
              senão devolve Bbin(x,A,m+1,d)
```

Para analisar a complexidade e correção de algoritmos que contém uma chamada recursiva, precisamos descrever o tempo de execução através de uma *recorrência*.

Recorrência

Equação ou desigualdade que descreve uma função em termos de uma variação dela mesma.

Exemplo:

$$f(n) = \begin{cases} c_1 & \text{se } n = 1 \\ f(n/2) + c_2 & \text{se } n > 1 \end{cases}$$

Na equação, o termo para *n = 1* é chamado de condição inicial e o termo para *n > 1* é denominado termo geral.

Em algumas situações, podem aparecer mais que uma condição inicial e vários termos gerais em uma mesma recorrência.

Recorrência - analise

Precisamos *resolver* a recorrência que define uma função, digamos *f(n) que descreve a complexidade.*

*Ob*ter uma "fórmula fechada" para *f(n).*

"fórmula fechada":

- É uma expressão que envolve um número fixo de operações aritméticas
- Não deve conter expressões da forma "1+2+3+...+n".

Recorrência

Resolver recorrências é uma arte!

Empregamos:

- Método da Substituição,
- Árvore de Recursão ou.
- Teorema Mestre.

Método da Substituição

1. devemos pressupor a forma da solução;
2. usar indução matemática para mostrar que a solução funciona.

Método da Substituição – exemplo 1

Seja a recorrência

$$T(1) = 1$$
$$T(n) = T(n-1)+3n+2 \quad \text{para } n = 2,3,4,\ldots$$

Que define a função T sobre inteiros positivos:

n	1	2	3	4	5	6
T(n)	1	9	20	34	51	71

Nossa hipótese será a função: $T(n) = (3/2)n^2+(7/2)n-4$

Verificando temos:

Se $n=1$ então $T(n) = 1 = 3/2+7/2-4$

Para $n \geq 2$ suponha que a fórmula está certa para $n-1$:

Então: $T(n) = T(n-1) + 3n + 2$

$=^{hip} (3/2)(n-1)^2+(7/2)(n-1)-4+3n+2$

$= (3/2)n^2-3n+3/2+(7/2)n-7/2-4+3n+2$

$=(3/2)n^2 + (7/2)n - 4$ *e está correta!*

Método da Árvore de recursão

Cada nó representa o custo de um único subproblema em algum lugar do conjunto de chamadas recursivas.

No exemplo-1 com a recorrência:

$$T(1) = 1$$
$$T(n) = T(n-1)+3n+2 \quad \text{para } n = 2,3,4,\ldots$$

Expandindo T(n), temos a árvore:

```
                                   T(n)
                                 ↙      ↘
                          T(n-1)   3n+2
                          ↙         ↘
                     T(n-2)           3(n-1)+2
                    ↙        ↘
               T(n-3)          3(n-2)+2
              ↙        ↘
           ...           3(n-3)+2
         ↙
      T(2)
     ↙    ↘
T(1) = 1    8
```

... temos 1 + (n-1) níveis,

Método da Árvore de recursão

exemplo-1 (cont.)

Portanto:

$$T(n) = [3n+2] + [3(n-1) + 2] +$$
$$+ [3(n-2) + 2] + [3(n-3)+2] +$$
$$+ \ldots + 8 + T(1) =$$
$$= 3[n + (n-1) + (n-2) + \ldots + 2] + 2(n-1) + 1 =$$
$$= (3/2)n^2 + (7/2)n - 4$$

... e obtemos a fórmula fechada!

Exercícios

1. Determine as funções que complexidade dos exemplos de algoritmos recursivos apresentados na aula sobre "Recorrências".

2. Resolva a recorrência

$$T(1) = 1$$
$$T(2) = 1$$
$$T(n) = T(n - 2) + 2n + 1 \quad \text{para } n = 3, 4, 5, \ldots$$

3. Resolva a recorrência

$$T(1) = 1$$
$$T(n) = T(\lfloor n/2 \rfloor) + 1 \quad \text{para } n = 2, 3, 4, 5, \ldots$$

Algoritmos Recursivos

Versão iterativa do algoritmo

Ordena – por - Seleção

```
Ordena – por - Seleção(A,n)
para i ← 1 até n-1 faça
        min ← i
        para j ← i + 1 até n faça
                se A[j] <  A[min]
                     então min ← j
        aux  ← A[min]
        A[min]  ←  A[i]
        A[i]  ← aux
```

Algoritmo Ordena – por – Seleção

Versão recursiva

```
Ordena – por – Seleção – rec (A, e, d)
se e < d
    então min ← e
          para k ← e + 1 até d faça
              se A[k] <  A[min]
                  então min ← k
          aux  ←  A[min]
          A[min]  ←  A[e]
          A[e]  ← aux
          Ordena – por – Seleção – rec(A, e+1, d)
```

Análise de algoritmo recursivo: exemplo

```
Ordena – por – Seleção – rec (A, e, d)
1.  se e < d
2.        então min ← e
3.                 para k ← e + 1 até d faça
4.                       se A[k] <  A[min]
5.                                 então min ← k
6.                 aux  ←  A[min]
7.                 A[min]  ←  A[e]
8.                 A[e]  ← aux
9.                 Ordena – por – Seleção – rec(A, e+1, d)
```

1. Correção

O algoritmo **pára**: a condição de parada na linha 1, garante um número finito de chamadas recursivas da função. No laço interno, linhas 3 – 5, o contador k controla a pesquisa do menor elemento.

O algoritmo ordena corretamente o vetor A[1..n] em ordem crescente ao finalizar a execução do algoritmo.

Em cada processamento, após a execução das linhas 2-8, o subvetor A[1..k] está ordenado.

Vide simulação a seguir.

Análise de algoritmos: outro exemplo

Ordena – por – Seleção - rec(A,n) : simulação

Simulação, em cada iteração **o item de menor valor** é posicionado e destacado em negrito:

15	25	57	13	9	18	48	37	12	92	86	33	Sel(A,e,d)
15	25	57	13	9	18	48	37	12	92	86	33	Sel(A,1,12)
9	25	57	13	15	18	48	37	12	92	86	33	Sel(A,2,12)
9	**12**	57	13	15	18	48	37	25	92	86	33	Sel(A,3,12)
9	**12**	**13**	57	15	18	48	37	25	92	86	33	Sel(A,4,12)
9	**12**	**13**	**15**	57	18	48	37	25	92	86	33	Sel(A,5,12)
9	**12**	**13**	**15**	**18**	57	48	37	25	92	86	33	Sel(A,6,12)
9	**12**	**13**	**15**	**18**	**25**	48	37	57	92	86	33	Sel(A,7,12)
9	**12**	**13**	**15**	**18**	**25**	**33**	37	57	92	86	48	Sel(A,8,12)
9	**12**	**13**	**15**	**18**	**25**	**33**	**37**	57	92	86	48	Sel(A,9,12)
9	**12**	**13**	**15**	**18**	**25**	**33**	**37**	**48**	92	86	57	Sel(A,10,12)
9	**12**	**13**	**15**	**18**	**25**	**33**	**37**	**48**	**57**	86	92	Sel(A,11,12)
9	**12**	**13**	**15**	**18**	**25**	**33**	**37**	**48**	**57**	**86**	92	
9	**12**	**13**	**15**	**18**	**25**	**33**	**37**	**48**	**57**	**86**	**92**	**A[1..12] ordenado**

Análise de algoritmo recursivo: exemplo

Ordena – por – Seleção – rec (A, e, d)	**contagem**
se e < d	**1**
então min ← e	**1**
para k ← e + 1 até d faça	**n**
se A[k] < A[min]	**n-1**
então min ← k	**n-1**
aux ← A[min]	**1**
A[min] ← A[e]	**1**
A[e] ← aux	**1**
Ordena – por – Seleção – rec(A, e+1, d)	**f(n-1)**

2. Complexidade

Relação de recorrência

$$f(1) = 1 \qquad n = 1$$
$$f(n) = f(n-1) + 3n + 3 \qquad n > 1$$

Resolvendo: $f(n) = 3(n+1) + 3(n-1 + 1) + 3(n-2 +1) + ... + 3(2 +1) + f(1)$

$= 3\{[(n)+(n-1)+(n-2)+(2)] + (n-1)*1\}+1$

$= 3\{[(n+2)(n-1)/2]+(n-1)\} + 1 = 3n^2/2 + 9n/2-5$

$$f(n) = 3n^2/2 + 9n/2-5 = O(n^2)$$

Técnicas de projetos de algoritmos

Projeto de algoritmos por indução

Divisão e Conquista

Programação Dinâmica

Método Guloso

Backtraking

Técnicas de projetos de algoritmos

Projeto de algoritmos por indução

Projetos de Algoritmos por Indução

Usamos a *técnica de indução* para desenvolver algoritmos.

A formulação do algoritmo será análoga ao desenvolvimento de uma demonstração por indução matemática.

Para resolver um problema P faremos o projeto de um algoritmo em dois passos:

1. exibir a resolução de uma ou mais instâncias pequenas de P (o caso base);
2. exibir como uma solução de toda a instância de P pode ser obtida a partir da solução de uma ou mais instâncias menores de P.

Projetos de Algoritmos por Indução

O 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 retornadas pelas chamadas recursivas.

Projetos de Algoritmos por Indução

Benefício imediato: o uso (correto) da técnica nos dá uma prova da correção do algoritmo.

A analise da complexidade será expressa em uma recorrência.

O algoritmo é eficiente, embora existam exemplos simples em que isso não acontece.

Projetos de Algoritmos por Indução: exemplo 1

Dada uma seqüência de números reais a_n, a_{n-1}, ..., a_1, a_0, e um número real x, calcular o valor do polinômio:

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

Primeira solução indutiva:

- caso base: n = 0. A solução é a_0.
- Suponha que para um dado *n* saibamos calcular o valor de $P_{n-1}(x) = a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$

Projetos de Algoritmos por Indução: *exemplo 1*

Para calcular $P_n(x)$, basta calcular x^n, multiplicar o resultado por a_n e somar o valor obtido por $P_{n-1}(x)$.

```
Polinômio_1(A,n,x)
se n = 0
    então devolve A[1]
    senão y ← Polinômio_1(A,n-1,x)
          v ← 1
          para i ← 1 até n faça
              v ← v * x
          y ← y + A[n+1] * v
          devolve y
```

Projetos de Algoritmos por Indução: exemplo 1

Observe o algoritmo **Polinomio_1**, que tem como dados de entrada o vetor $A[1..n+1]$ com os coeficientes: $A[1]= a_0$, $A[2]= a_1$, ..., $A[n]= a_{n-1}$, e $A[n+1] = a_n$, n que é o grau do polinômio, o valor de x e, na saída o valor calculado para $P_n(x)$.

Projetos de Algoritmos por Indução: exemplo 1

Na analise da complexidade, chamaremos de T(n) o número de operações aritméticas realizadas pelo algoritmo.

Obtemos a seguinte recorrência para T(n), onde mult = multiplicação e ad = adição:

$$T(n) = 0 \qquad \text{se} \quad n = 0$$

$$T(n) = T(n-1) + n * \text{mult} + 1\ \text{ad} \qquad \text{se} \quad n > 0$$

Resolvendo a recorrência chegamos ao resultado:

$$T(n) = \sum_{i=1}^{n} (i \cdot mult + ad) = \frac{(n+1)n}{2} \cdot mult + n \cdot ad$$

O número de multiplicações pode ser diminuído calculando x^n com um algoritmo que calcula rapidamente a exponenciação, que veremos no próximo exemplo.

Projetos de Algoritmos por Indução: exemplo 2

Um dos desperdícios cometidos nessa primeira solução é o re-cálculo de potências de *x*. Vamos eliminar essa computação desnecessária trazendo o cálculo de x^{n-1} para dentro da hipótese de indução.

Suponha que para um dado *n* saibamos calcular não só o valor de $P_{n-1}(x) = a_{n-1}x^{n-1} + ... + a_1x + a_0$, mas também o valor de x^{n-1}.

Então no passo de indução, primeiramente 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: $(a_0 , 1)$.

Projetos de Algoritmos por Indução: exemplo 2

No algoritmo **Polinomio_2**, temos os mesmos dados de entrada: o vetor A[1..n+1] com os coeficientes, *n* o grau do polinômio, o valor de *x* e agora na saída temos o par de valores ($P_n(x)$, x^n).

```
Polinômio_2(A,n,x)
se n = 0
    então devolve (A[1], 1)
    senão (y,v) ← Polinômio_2(A,n-1,x)
            v ← v * x
            y ← y + A[n+1] * v
            devolve (y, v)
```

Projetos de Algoritmos por Indução: exemplo 2

Na análise desta versão, T(n) novamente representa o número de operações aritméticas. T(n) é dada pela recorrência:

$$T(n) = 0 \qquad \text{se} \quad n = 0$$

$$T(n) = T(n-1) + 2 \text{ mult} + 1 \text{ ad} \qquad \text{se} \quad n > 0$$

Resolvendo a recorrência chegamos ao resultado:

$$T(n) = \sum_{i=1}^{n} (2 \cdot mult + ad) = n \cdot (2 \cdot mult + ad)$$

Com uma grande melhora: T(n) = $\Theta(n)$,

más ainda podemos evoluir.

Projetos de Algoritmos por Indução: exemplo 3

A escolha de considerar o polinômio $P_{n-1}(x)$ na hipótese de indução não é a única possível. Veja uma alternativa que resulta num ganho de complexidade:

- caso base: n = 0. A solução é a_0.
- Suponha que para um dado $n > 0$ saibamos calcular o valor de $P_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} + \ldots + a_1$.

Então $$P_n(x) = xP_{n-1}(x) + a_0$$

Assim, basta uma multiplicação e uma adição para obtermos $P_n(x)$ a partir de $P_{n-1}(x)$.

Observe que o algoritmo implementado nessa solução é a regra de Horner para avaliar polinômios.

Projetos de Algoritmos por Indução: *exemplo 3*

Na entrada do algoritmo **Polinomio_3**, mantivemos os mesmos dados, más invertemos a ordem em que armazenamos os coeficientes: temos a_0 = A[n+1], a_1 = A[n], …, a_{n-1} = A[2], a_n = A[1]), e na saída temos o valor calculado $P_n(x)$:

```
Polinômio_3(A, n, x)
se n = 0
    então devolve A[n+1]
    senão y ← x * Polinômio_3(A, n-1, x)+ A[n+1]
          devolve y
```

Projetos de Algoritmos por Indução: *exemplo*

A análise de complexidade obtemos:

$$T(n) = 0 \qquad \text{se} \quad n = 0$$

$$T(n) = T(n-1) + 1 \text{ mult} + 1 \text{ ad} \qquad \text{se} \quad n > 0$$

Resolvendo a recorrência chegamos ao resultado:

$$T(n) = \sum_{i=1}^{n} (mult + ad) = n \cdot mult + n \cdot ad$$

Embora o resultado tenha a mesma ordem da solução anterior, $T(n) = \Theta(n)$, observamos uma melhora com a redução de uma operação de multiplicação em cada chamada.

Proj Alg por indução: exemplo 5 - Soma máxima

Re-visitando solução de tempo linear:

Dada uma sequência $A = a_1, a_2, \ldots, a_{n-1}, a_n$ *de números inteiros (não necessariamente positivos) encontre uma subsequência consecutiva*

$$A' = a_i, a_{i+1}, \ldots, a_{j-1}, a_j, 1 \le i, j \le n,$$

cuja soma seja máxima dentre todas .

Exemplos:

A = {3,0,-1,2,4,-1,2,-2} temos: A' = {3,0,-1,2,4,-1,2}, soma = 9.

A = {-1, -2, 0} temos: A' = {0}, soma = 0.

A = {-3, -1} temos: A' = { } adotamos como "zero"

Proj Alg por indução: exemplo 5 - Soma máxima

Re-vendo a solução modificada, para mostrar a soma máxima e os índices de início e fim da subseqüência:

```
Somax4(A,n)
max ← aux ← j ← k ← 0
ini ← fim ← 1
para i ← 1 até n faça
      aux ← aux + A[i]
      k ← k + 1
      se n = 0
            então max ← aux
                  fim ← k
            senão se aux < 0
                        então aux ← 0
                              j ← k ← i+1
                              ini ← j
devolve (max, ini, fim)
```

Proj Alg por indução: exemplo 5 - Soma máxima

Inicialmente, examinamos o que obter da hipótese de indução simples:

> *Para um n qualquer, sabemos calcular a soma máxima de seqüências cujo comprimento seja menor que n.*

Seja então a seqüência $A = a_1, a_2, \ldots, ,a_{n-1}, a_n$, uma seqüência de comprimento $n > 1$.

Considere a seqüência $A_{n-1} = a_1, a_2, \ldots, a_{n-1}$ obtida removendo-se o número a_n que apresenta a soma máxima na subseqüência $A_{n-1}' = a_i, a_{i+1}, \ldots, ,a_{j-1}, a_j$, obtida aplicando hipótese.

Proj Alg por indução: exemplo 5 - Soma máxima

Más, o que falta à hipótese?...

É evidente que quando encontramos $A' = a_k , a_{k+1} , ..., ,a_{n-1}, a_n$, na verdade encontramos um **sufixo** dentre as subseqüências de A.

Assim se conhecemos o sufixo que nos dá a maior soma, teremos resolvido o problema completamente.

Parece então natural enunciar a seguinte hipótese (agora fortalecida):

Para um n qualquer, sabemos calcular a soma máxima de seqüências cujo comprimento seja menor que n e o sufixo máximo de seqüências cujo comprimento seja menor que n.

É clara desta discussão também, a base da indução: para $n = 1$, a seqüência de $A = a_1$ é a_1 se $a_1 \geq 0$, e vazia (com valor nulo) caso contrário.

Proj Alg por indução: exemplo 5 - Soma máxima

Escrevemos então o algoritmo:

```
Somax4-ind(A,n)
se n = 1
    então se A[1]< 0
            então ini ← fim ← i-suf ← max ← suf ← 0
            senão ini ← fim ← i-suf ← 1
                  max ← suf ← 0
    senão (ini, fim, i-suf, max, suf) ← Somax4-ind(A,n-1)
          se i-suf = 0
             então i-suf ← n
          suf ← suf + A[n]
          se suf > max
             então ini ← i-suf
                   fim ← n
                   Max ← suf
             senão se suf < 0
                      então suf ← i-suf ← 0
devolve (ini, fim, i-suf, max, suf)
```

Proj Alg por indução: exemplo 5 - Soma máxima

Analisando a complexidade *f(n)* do algoritmo obtemos:

$$f(n) = \Theta(1) \qquad se \quad n = 1$$

$$f(n) = f(n-1) + \Theta(1) \qquad se \quad n > 1$$

Com solução:

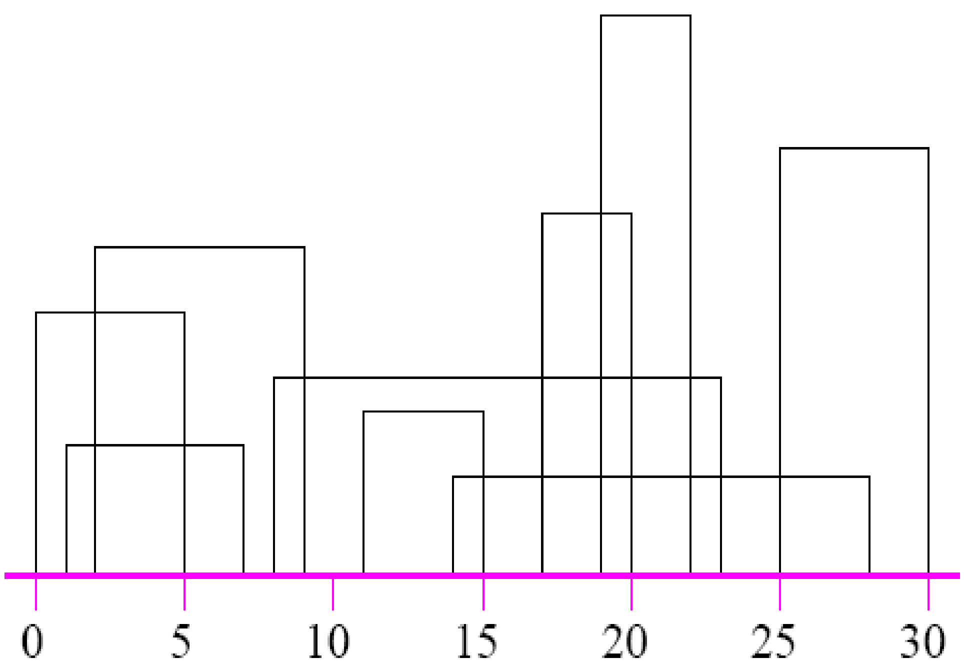
$$\sum_{1}^{n} \Theta(1) = n\Theta(1) = \Theta(n)$$

mesma ordem que a solução anterior.

Exemplo 2: o problema do *skyline*

Problema:

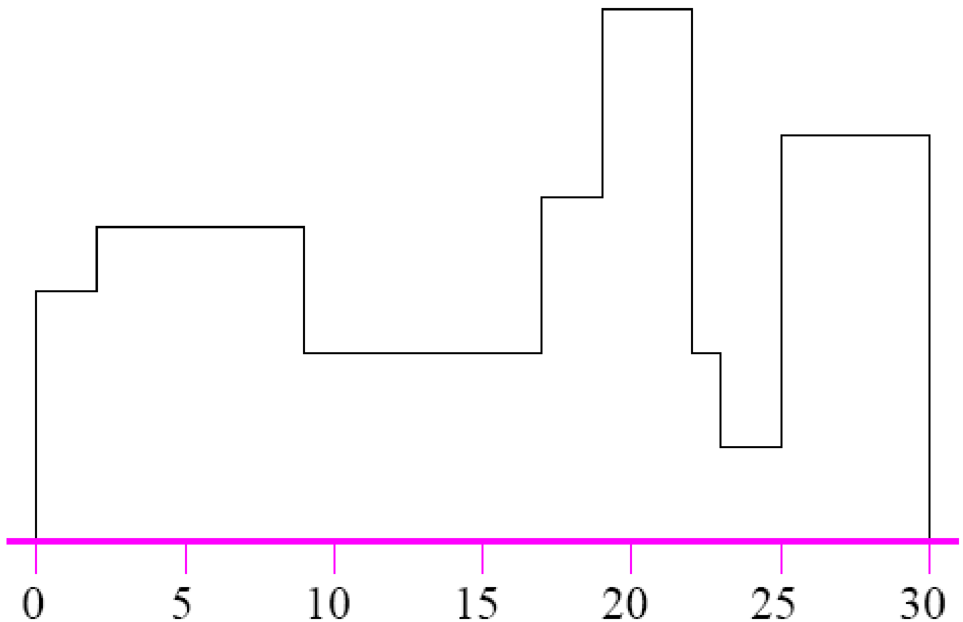
Dada uma seqüência de triplas (l_i, h_i, r_i) para $i = 1, 2, \ldots, n$ que representam prédios retangulares, determinar a silhueta dos prédios (skyline).



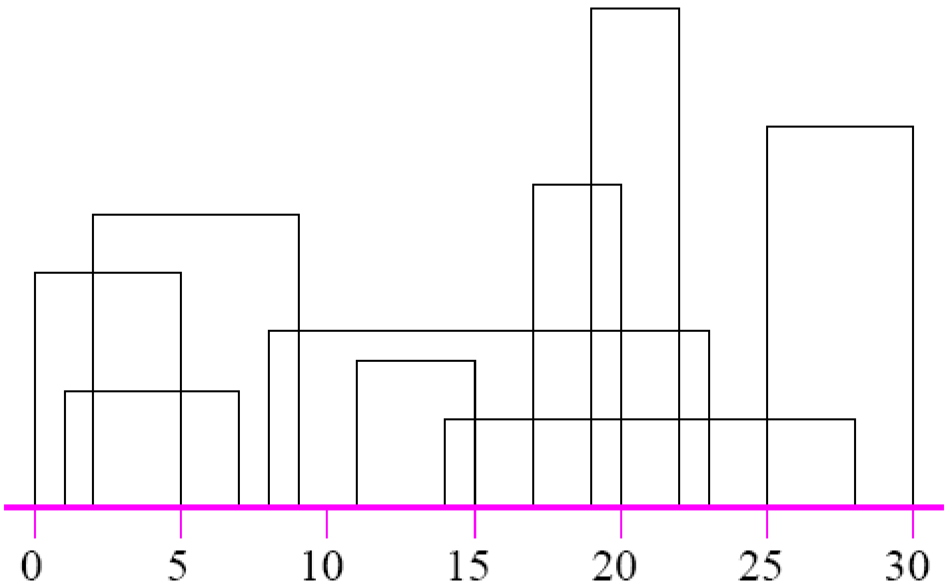
Exemplo 2: o problema do *skyline*

Problema:

Dada uma seqüência 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*

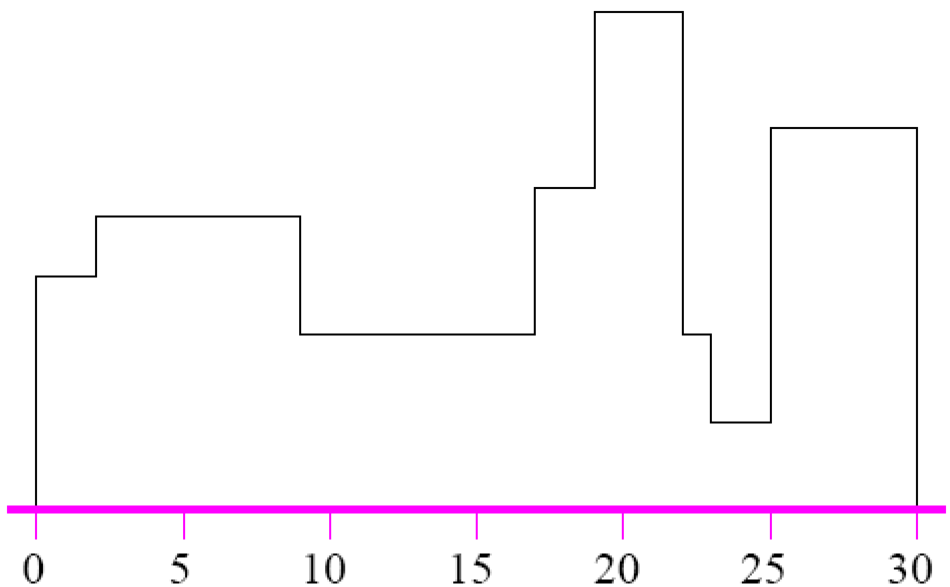


Cada prédio é descrito por uma tripla (l_i, h_i, r_i) onde l_i e r_i são as coordenadas x do prédio e h_i é a altura do prédio.

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

Exemplo 2: o problema do *skyline*

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



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

Skyline - Solução 1

Vamos tentar usar o método de projeto por indução.

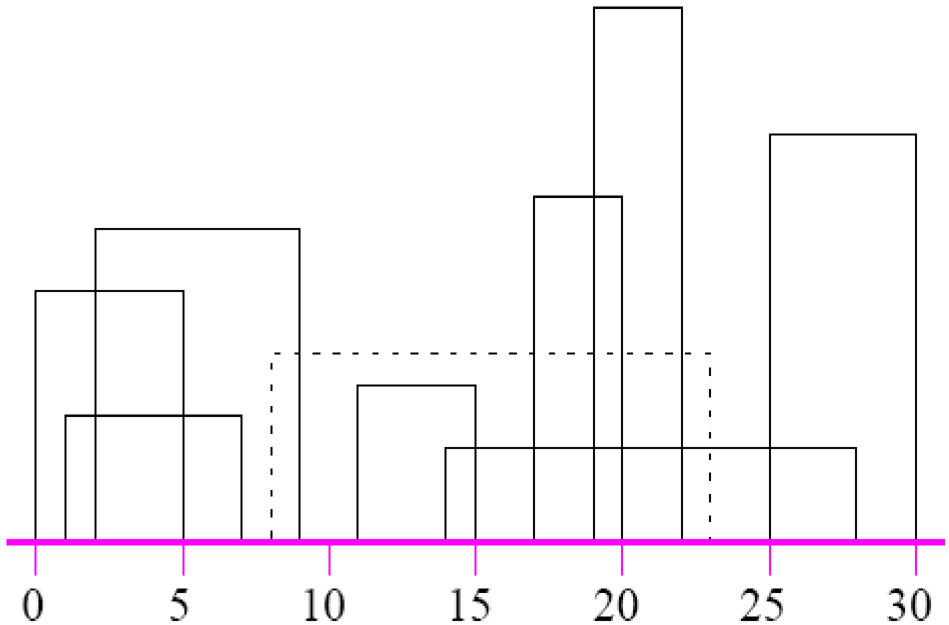
Hipótese de indução: (primeira tentativa)

Dada uma seqüência 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

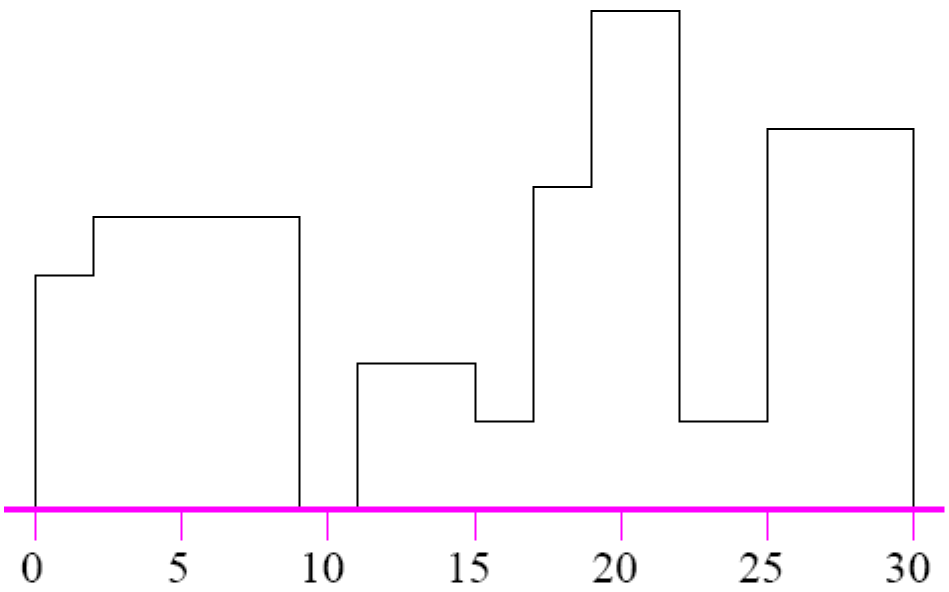
Subproblema obtido removendo-se o prédio $B_n = (8, 6, 23)$:



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

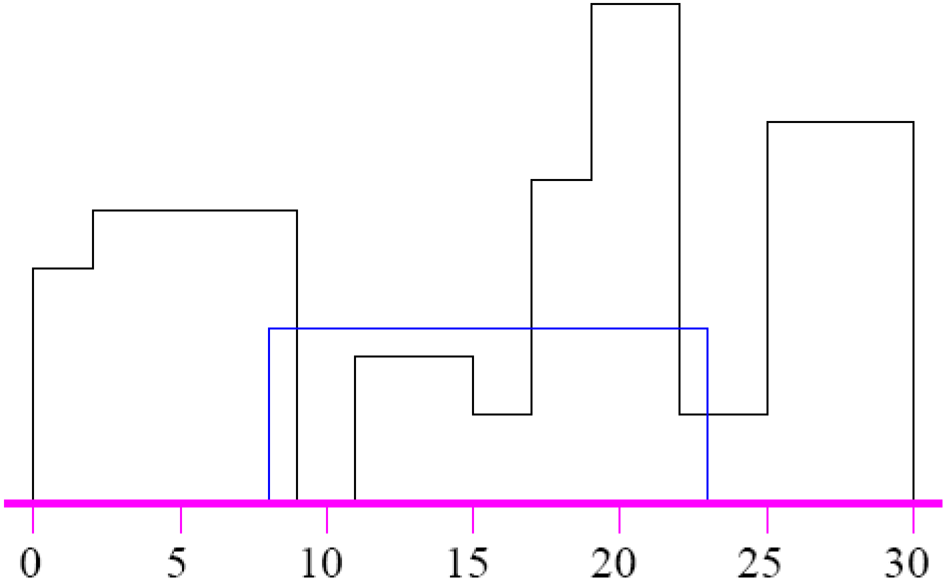
Subproblema obtido removendo-se o prédio $B_n = (8, 6, 23)$:



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:

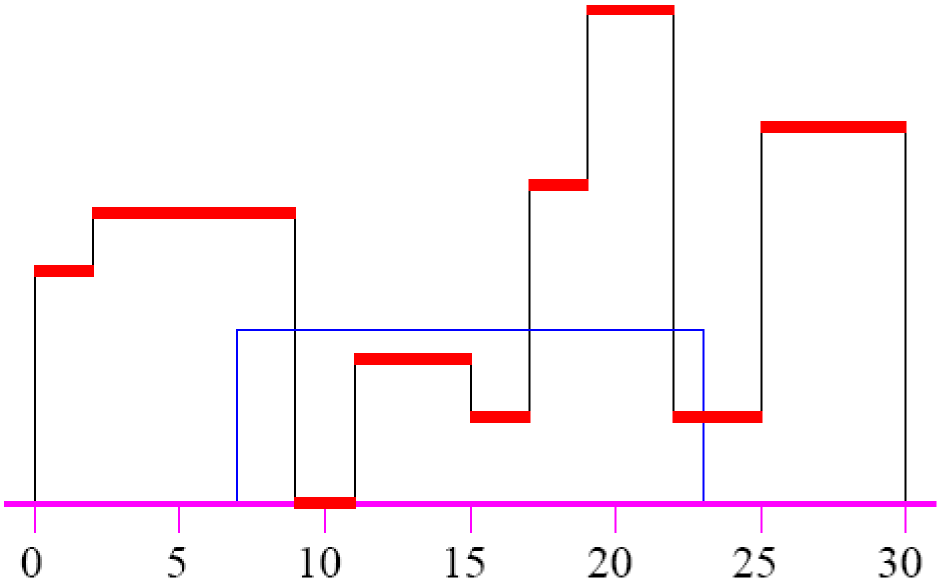


Skyline do subproblema:
$(0, \mathbf{8}, 2, \mathbf{10}, 9, \mathbf{0}, 11, \mathbf{5}, 15, \mathbf{3}, 17, \mathbf{11}, 19, \mathbf{17}, 22, \mathbf{3}, 25, \mathbf{13}, 30, \mathbf{0})$

Skyline - Solução 1

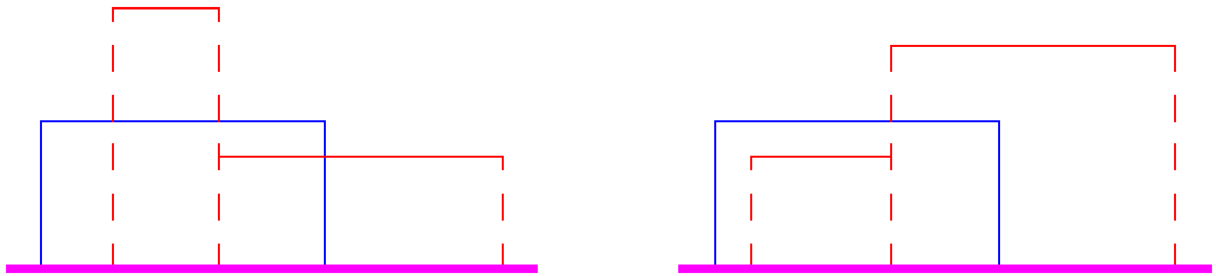
Como atualizar o skyline?

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



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

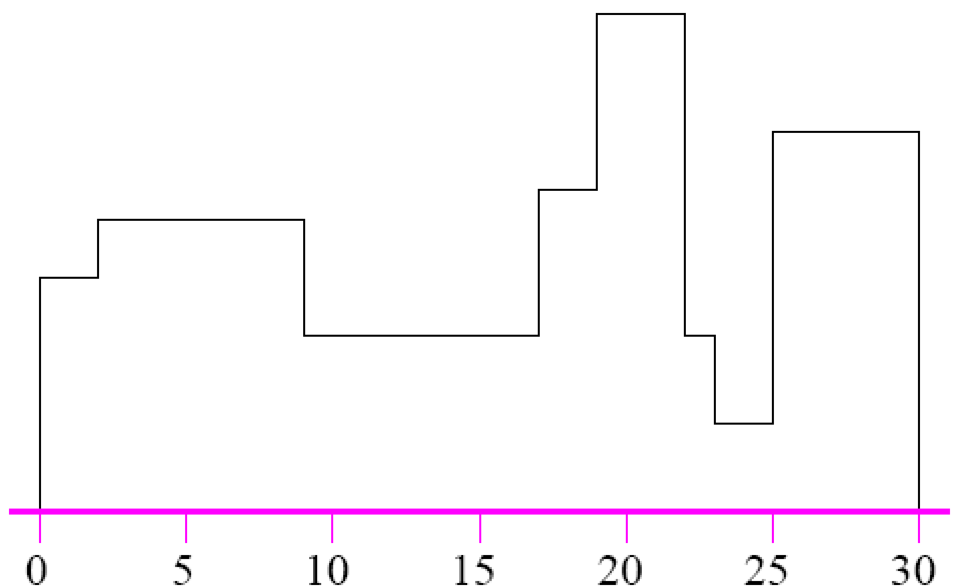


(3) $l \leq x_i < r$:

- 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.
1. **se** $n = 1$ **então** $S \leftarrow (l_1, h_1, r_1, 0)$
2. **senão**
3. $S \leftarrow$ Skyline($B, n - 1$)
4. $S \leftarrow$ AcrescPredioSkyline(S, B_n)
5. **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:

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

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

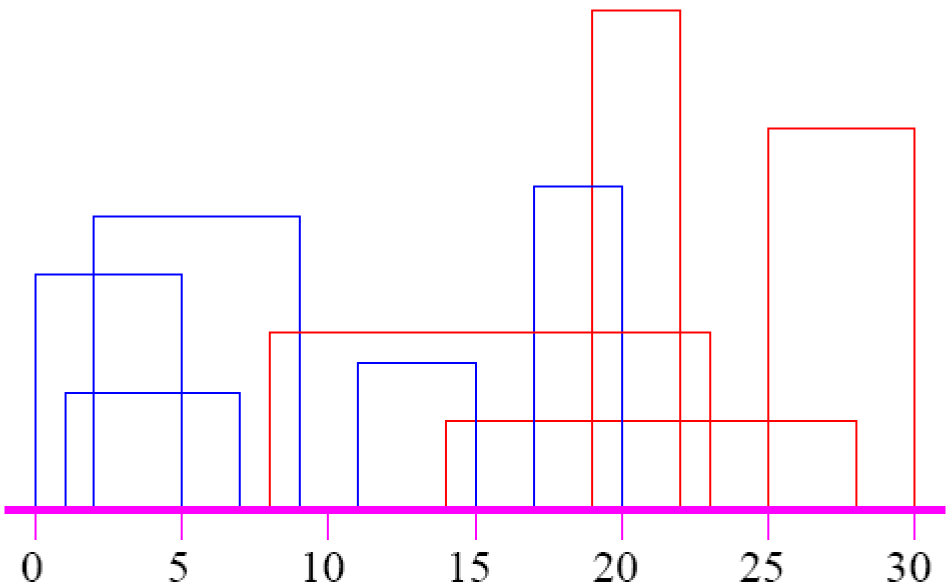
Podemos usar **indução forte** e **divisão-e-conquista**.

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



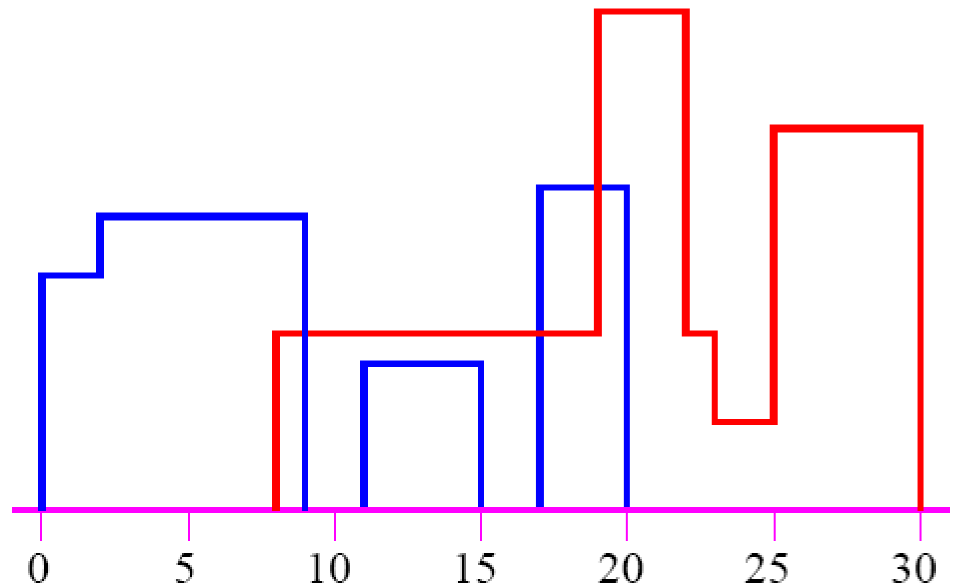
Prédios do subproblema 1:

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

Prédios do subproblema 2:

$(19, 17, 22), (14, 3, 28), (25, 13, 30), (8, 6, 23)$.

Skyline - Solução 2



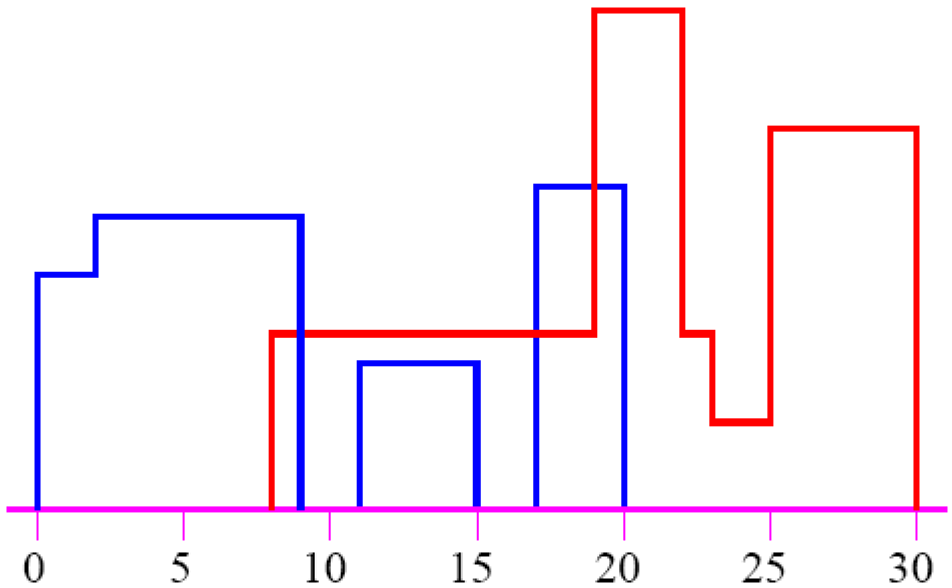
Skyline do subproblema 1:

(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.

$$T(n) = \begin{cases} 0, & n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n), & n > 1. \end{cases}$$

Humm. . . Já vimos esta recorrência.

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

Projeto por Indução : 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.

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 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 k não existe celebridade em S', então a pessoa n é celebridade em S se $M[n, j] = 0$ e $M[j, n] = 1$ para todo $j < n$; caso contrário não há 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(S, M)

$\triangleright$ **Entrada:** conjunto de pessoas $S = \{1, 2, \ldots, n\}$;
M, a matriz que define quem conhece quem em S.

$\triangleright$ **Saída:** Um inteiro $k \leq n$ que é celebridade em S ou $k = 0$

se $|S| = 1$ **então** $k \leftarrow$ elemento em S
senão
    sejam i, j quaisquer duas pessoas em S
    se $M[i, j] = 1$ **então** $s \leftarrow i$ **senão** $s \leftarrow j$
    $S' \leftarrow S \setminus \{s\}$
    $k \leftarrow$ Celebridade(S', M)
    se $k > 0$ **então**
        se $(M[s, k] \neq 1)$ **ou** $(M[k, s] \neq 0)$ **então** $k \leftarrow 0$
devolva k

O problema da celebridade

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

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

Exercício

1. Para o problema de *verificar se um número x faz parte de uma seqüência de n números armazenados em um vetor A[1..n]*, um algoritmo simples nos leva a uma solução de complexidade *O(n)* no pior caso. Projete um algoritmo incremental por indução, analisando sua correção e complexidade nessa nova versão.

Exercício

2. Projete, por indução, um algoritmo que recebe um vetor A[e..d] e devolve o valor da soma dos elementos A[i] tais que $e \leq i \leq d$ e i é *par*.