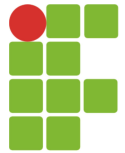
Projeto por Indução - Divisão e Conquista

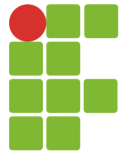
Análise de Algoritmos – Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

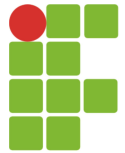
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário



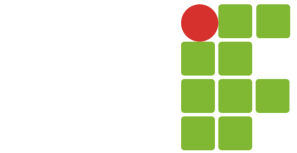
Sumário



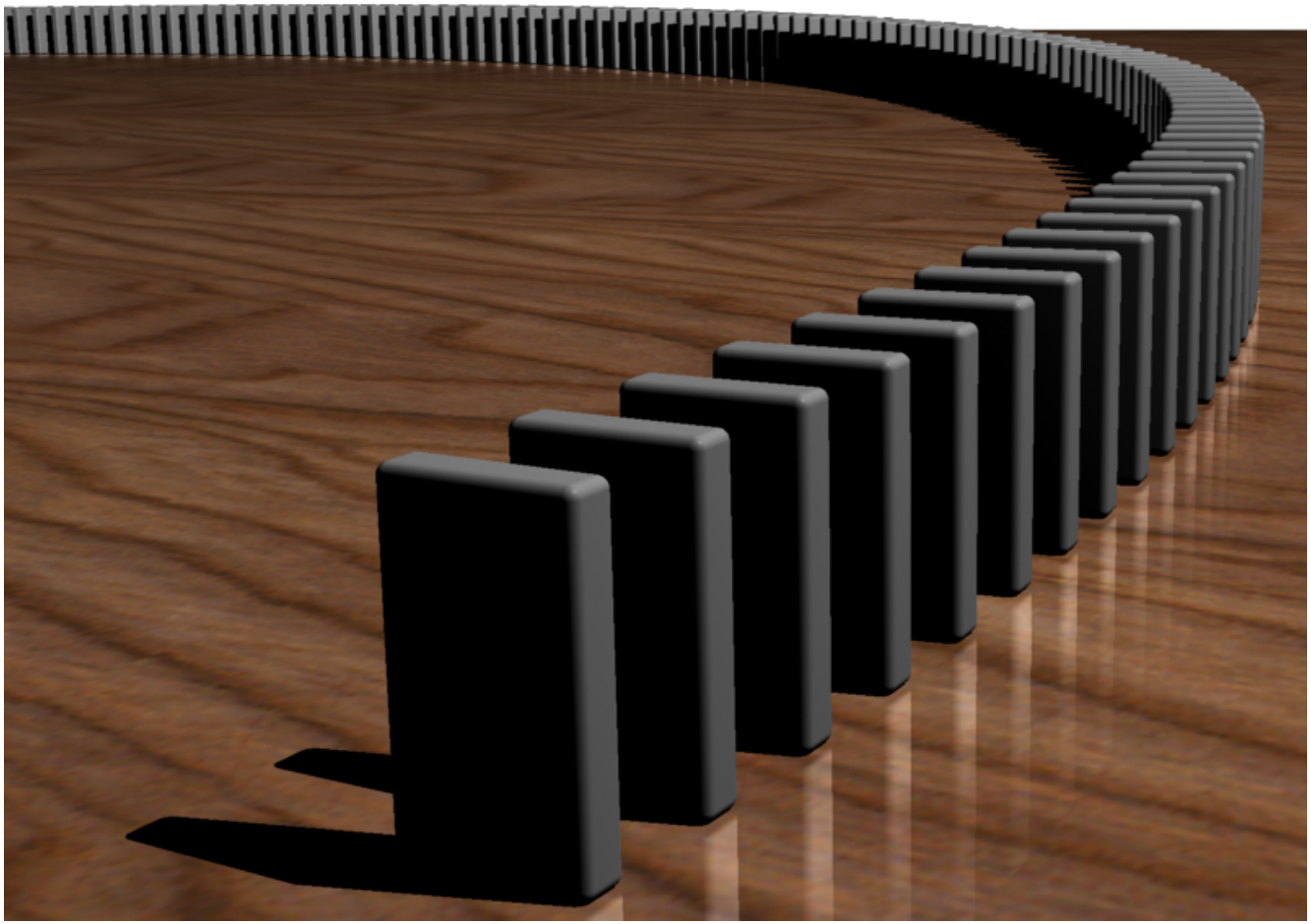
Introdução

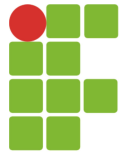
Indução Matemática

- A indução matemática é um método dedutivo de **prova** que se baseia em:
  1. **Caso base:** Verificar que uma determinada assertiva vale para um elemento inteiro k, geralmente $k = 0$ ou $k = 1$.
  2. **Passo de indução:** Consiste em demonstrar que, se uma assertiva vale para todo $m < n$, então ela também vale para n.



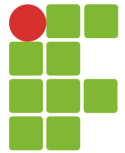
Indução Matemática



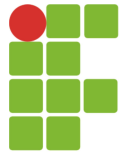


Projeto de Algoritmos por "Indução"

- É possível projetar algoritmos usando uma argumentação análoga à da indução matemática.
- Não é necessário desenvolver uma solução do zero! Basta garantir que:
  1. É possível resolver uma instância pequena do problema (análogo ao **caso base**).
  2. É possível obter uma solução para um problema com tamanho de instância n através das soluções de subproblemas. (análogo ao **passo indutivo**).



Sumário



Find One-to-One Mappings

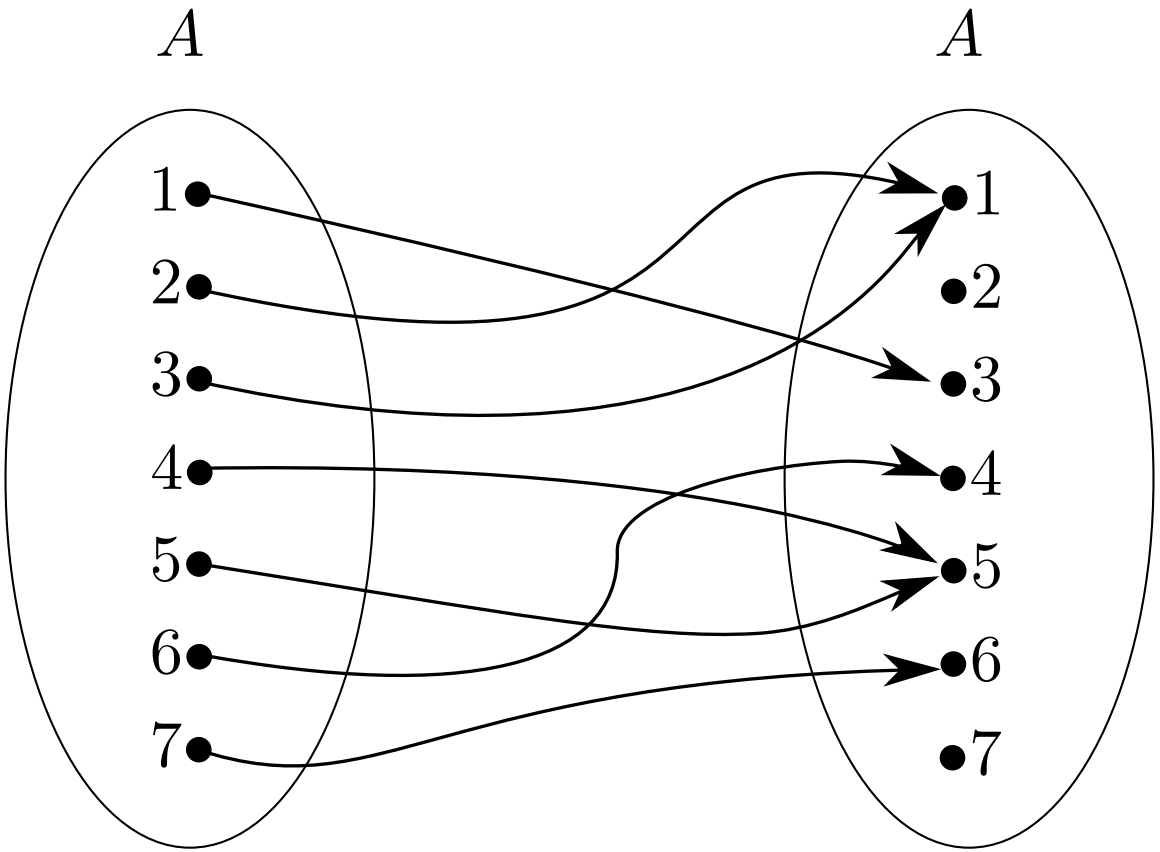
FOTOM Problem

Entrada: Um conjunto finito $A = \{1, \ldots, n\}$ e uma função

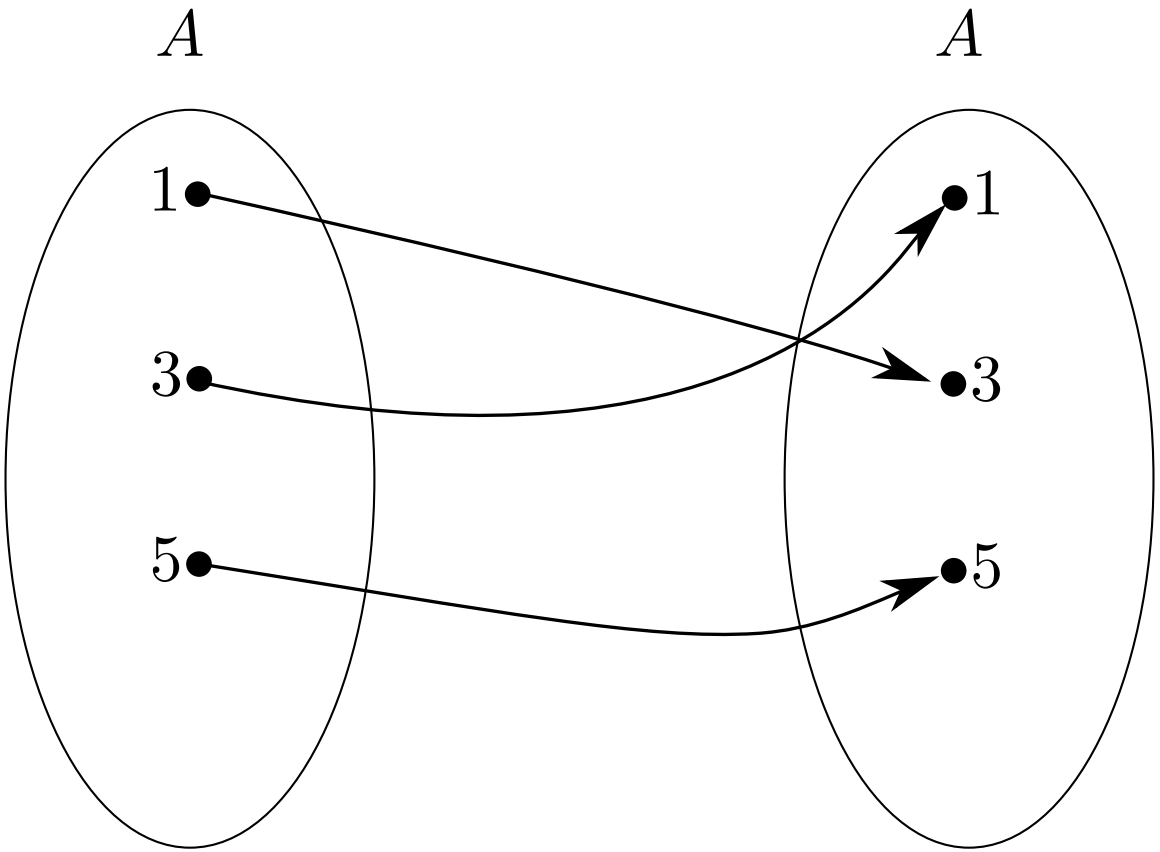
$$f : A \to A$$
$$f : x \mapsto f(x)$$

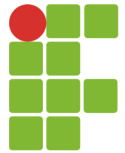
Saída: Determinar o maior conjunto $B \subseteq A$ de modo que $f : B \to B$ seja bijetiva.

Fine One-to-One Mappings



Fine One-to-One Mappings

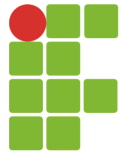




Find One-to-One Mappings

Solução Força-Bruta

- Gerar todos os subconjunto e verificar se é um mapeamento de um para um.
- Escolher o maior dos subconjuntos.

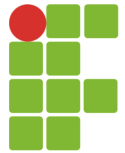


Find One-to-One Mappings

Algorithm 1: FOTOM-BRUTE-FORCE(A, f)

Input: A, f
Output: $B \subseteq A$, maior subconjunto maximal de A que torna f um para um.

```
1  B ← ∅
2  n ← |A|
3  for all( D ∈ P(A) )
4      for( i ← 1; i ≤ n; i++ )  c[i] ← 0
5      for all( i ∈ D )
6          c[f(i)]++
7      one − to − one ← true
8      for all( i ∈ D )
9          if( c[f(i)] ≥ 2 )
10             one − to − one ← false
11     if( one − to − one ∧ |D| > |B| )
12         B ← D
13 return B
```

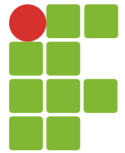


Find One-to-One Mappings

Complexidade do algoritmo FOTOM-BRUTE-FORCE

- No total, temos 2^n subconjuntos. Para checar se a função é um para um, necessitamos de tempo $\Theta(n)$.
- o tempo de pior caso do algoritmo é:

$$\Theta(n2^n)$$



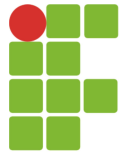
Find One-to-One Mappings

Caso Base

- Conjunto vazio.

Hipótese de Indução

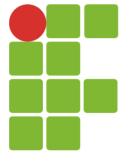
- Sabemos encontrar o subconjunto maximal para o conjunto $A' = A \backslash \{x\}$, onde $x \in A$.



Find One-to-One Mappings

Passo de Indução.

- A redução da entrada consiste em retirar algum elemento que não prejudique o mapeamento um pra um.
- Seja x um elemento que não possua mapeamento. x não pode fazer parte do conjunto maximal, visto que ele não possui um mapeamento para ele, logo a função não pode ser um pra um.
- Caso ele fizesse parte, haveria um nó com mais de uma aresta (princípio das casas dos pombos).
- Basta retirar x.

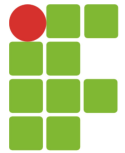


Find One-to-One Mappings

Algorithm 2: FOTOM(A, f)

Input: A, f
Output: $B \subseteq A$,maior conjunto maximal que torna f um pra um.

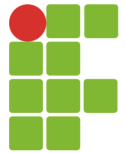
```
1  n ← A.SIZE()
2  B ← A
3  for( i ← 1; i ≤ n; i++ )  c[i] ← 0
4  for( i ← 1; i ≤ n; i++ )  c[f(i)]++
5  for( i ← 1; i ≤ n; i++ )
6      if( c[i] = 0 )
7          Q.PUSH(i) // Inserimos i em uma fila
8  while ¬Q.EMPTY() do
9      i ← Q.FRONT()
10     Q.POP()
11     B ← B\{i}
12     c[f(i)]--
13     if( c[f(i)] = 0 )
14         Q.PUSH(f(i))
15 return B
```



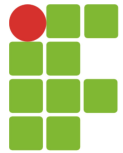
Find One-to-One Mappings

Complexidade do algoritmo FOTOM

- A inicialização leva tempo $\Theta(n)$.
- Além disso, podemos ter no máximo n elementos na fila.
- Logo o **while** não executa mais do que n vezes. Cada iteração do **while** leva tempo constante.
- Portanto, o pior caso do algoritmo tem complexidade $\Theta(n)$.



Sumário

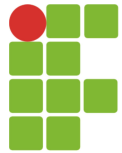


CELEBRITY

Definição (Celebridade)

Uma celebridade é uma pessoa que **somente** conhece a si mesmo, mas que é conhecida por todas as outras pessoas.

CELEBRITY
Entrada: Um conjunto de pessoas $P = \{0, \ldots, n-1\}$ e uma matriz de adjacências K. $K[i][j] = 1$ se a pessoa i conhece a pessoa j e $K[i][j] = 0$ caso contrário.
Saída: $P' \subseteq P$, conjunto de celebridades.



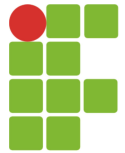
CELEBRITY

Definição (Celebridade)

Uma celebridade é uma pessoa que **somente** conhece a si mesmo, mas que é conhecida por todas as outras pessoas.

CELEBRITY
Entrada: Um conjunto de pessoas $P = \{0, \ldots, n-1\}$ e uma matriz de adjacências K. $K[i][j] = 1$ se a pessoa i conhece a pessoa j e $K[i][j] = 0$ caso contrário.
Saída: $P' \subseteq P$, conjunto de celebridades.

Quantas celebridades podemos ter ao máximo em P?



CELEBRITY

Solução Força-Bruta

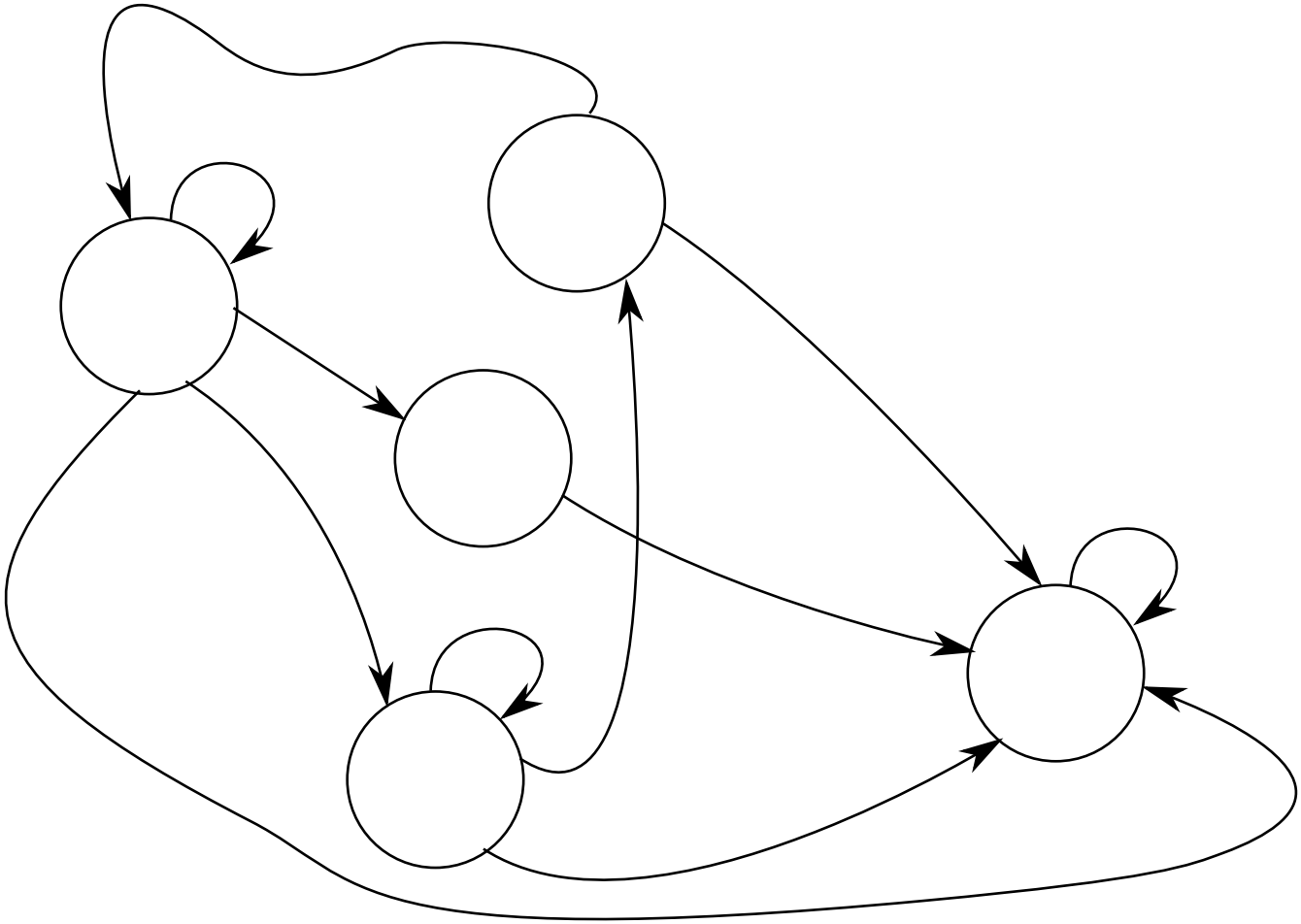
Testar a matriz de adjacências. Se, para um determinado i, temos:

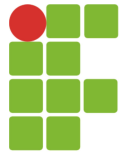
$$K[j][i] = 1, \quad 0 \leq j < n$$

E se:

$$K[i][j] = 0, \quad j \neq i \wedge 0 \leq j < n$$

CELEBRITY



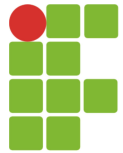


CELEBRITY

Algorithm 3: CELEBRITY-BRUTE-FORCE(K, P)

Input: $K[0, n-1][0, n-1], P$
Output: $P' \subseteq P$, conjunto de celebridades.

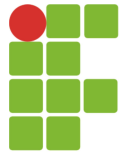
```
1  P' ← ∅
2  for( i ← 0; i < n; i++ )
3      candidate ← K[i][i]
4      for( j ← 0; j < n; j++ )
5          if( j ≠ i ∧ K[j][i] = 0 )
6              candidate ← false
7      for( j ← 0; j < n; j++ )
8          if( i ≠ j ∧ K[i][j] = 1 )
9              candidate ← false
10     if( candidate = true )
11         P' ← {i}
12 return P'
```



CELEBRITY

Complexidade do algoritmo CELEBRITY-BRUTE-FORCE

$\Theta(n^2)$. Para cada $0 \leq i \leq n-1$, fazemos $2(n-1)$ comparações (perguntas).



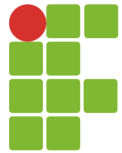
CELEBRITY

Caso Base

Encontrar uma celebridade quando $|P| = 1$ é fácil, basta checar se $K[0][0] = 1$.

Hipótese de Indução

Sabemos encontrar uma celebridade para o conjunto $Q = \{p_0, \ldots, p_{n-2}\}$.

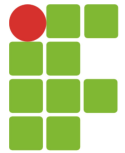


CELEBRITY

Passo de Indução

Para encontrar a celebridade para $P = \{p_0, \ldots, p_{n-1}\}$ a partir de Q, temos 3 possibilidades:

1. A celebridade se encontra em Q.
   - Basta checar se p_{n-1} conhece a celebridade de Q e se a celebridade não conhece p_{n-1}.
2. A celebridade é p_{n-1}.
   - Temos que fazer fazer $2(n-1)$ perguntas para confirmar.
3. A celebridade não se encontra em P.
   - Temos que fazer fazer, no máximo, $2(n-1)$ perguntas para verificar.

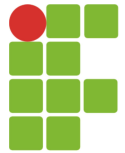


CELEBRITY

No pior caso, temos:

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

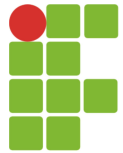
Já sabemos que $T(n) \in \Theta(n^2)$.



CELEBRITY

Projeto por Indução

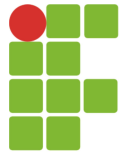
- O algoritmo resultante tem a mesma complexidade do algoritmo força-bruta. Muito esforço para nada!
- Mas não utilizamos uma informação valiosa em nosso favor. É muito mais difícil identificar uma celebridade do que uma não celebridade.
- Vamos tentar em outra direção, começar com um conjunto de n elementos e reduzir o problema.



CELEBRITY

Projeto por Indução

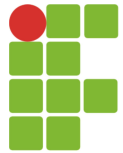
- O algoritmo resultante tem a mesma complexidade do algoritmo força-bruta. Muito esforço para nada!
- Mas não utilizamos uma informação valiosa em nosso favor. É muito mais difícil identificar uma celebridade do que uma não celebridade.
- Vamos tentar em outra direção, começar com um conjunto de n elementos e reduzir o problema.
  - Se p_i diz que conhece p_j, qual a conclusão?
  - Se p_i diz que **não** conhece p_j, qual a conclusão?



CELEBRITY

Passo de Indução

- Para reduzir P em um elemento, perguntamos se $i \in P$ conhece $j \in P$.
- Suponha s.p.g. que i não seja a celebridade. $Q = P \backslash \{i\}$.
- Resolvemos recursivamente o problema para Q.
- Temos os mesmos 3 casos:
  1. A celebridade se encontra em Q.
     - Basta checar se i conhece a celebridade e se a celebridade não conhece i.
  2. A celebridade está em P, mas não em Q.
     - Impossível!
  3. Não existe celebridade.
     - Não precisamos fazer nada, pois já eliminamos i.

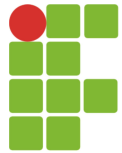


CELEBRITY

Passo de Indução

- Temos os mesmos 3 casos:
  1. A celebridade se encontra em Q.
     - Basta checar se p_i conhece a celebridade e se a celebridade não conhece p_i.
  2. A celebridade está em P, mas não em Q.
     - Impossível!
  3. Não existe celebridade.
     - Não precisamos fazer nada, pois já eliminamos p_i.

A cada redução da instância, fazemos um número constante de comparações!



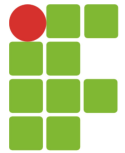
CELEBRITY

Algorithm 4: CELEBRITY(K, P)

Input: K, P

Output: $P' \subseteq P$, conjunto de celebridades.

1 $(n, P', i, j, next) \leftarrow (P.\text{SIZE}(), \emptyset, 0, 1, 2)$
2 **while** $next < n$ **do**
3 $\quad$ **if**($K[i][j] = 1$) $\; i \leftarrow next$
4 $\quad$ **else** $\; j \leftarrow next$
5 $\quad$ $next++$
6 **if**($i = n \wedge n > 1$) $\; candidate \leftarrow j$
7 **else** $\; candidate \leftarrow i$
8 $wrong \leftarrow K[candidate][candidate] = 1 \; ?$ **false** : **true**
9 **for**($k \leftarrow 0; k < n \wedge \neg wrong; k++$)
10 $\quad$ **if**($K[candidate][k] = 1 \wedge k \neq candidate$) $\; wrong \leftarrow true$
11 $\quad$ **if**($K[k][candidate] = 0$) $\; wrong \leftarrow true$
12 **if**($\neg wrong$) $\; P' \leftarrow \{candidate\}$
13 **return** P'



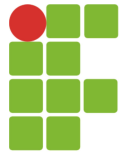
CELEBRITY

Complexidade do algoritmo CELEBRITY

- A ideia do algoritmo é reduzir o problema de uma instância de uma maneira esperta.
- Fazemos uma eliminação a cada redução usando um número constante de perguntas (uma pergunta).
- Após isso, necessitamos de verificar realmente, se o candidato escolhido é realmente uma celebridade.
- Portanto a relação de recorrência é expressa como:

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

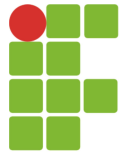
- Sabemos que $T(n) \in \Theta(n)$.



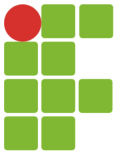
CELEBRITY

Complexidade do algoritmo CELEBRITY

- O procedimento de verificação também leva tempo $\Theta(n)$.
- Logo, o algoritmo CELEBRITY leva tempo $\Theta(n)$.

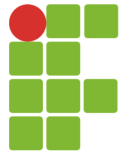


Sumário



SKYLINE

SKYLINE



SKYLINE

SKYLINE

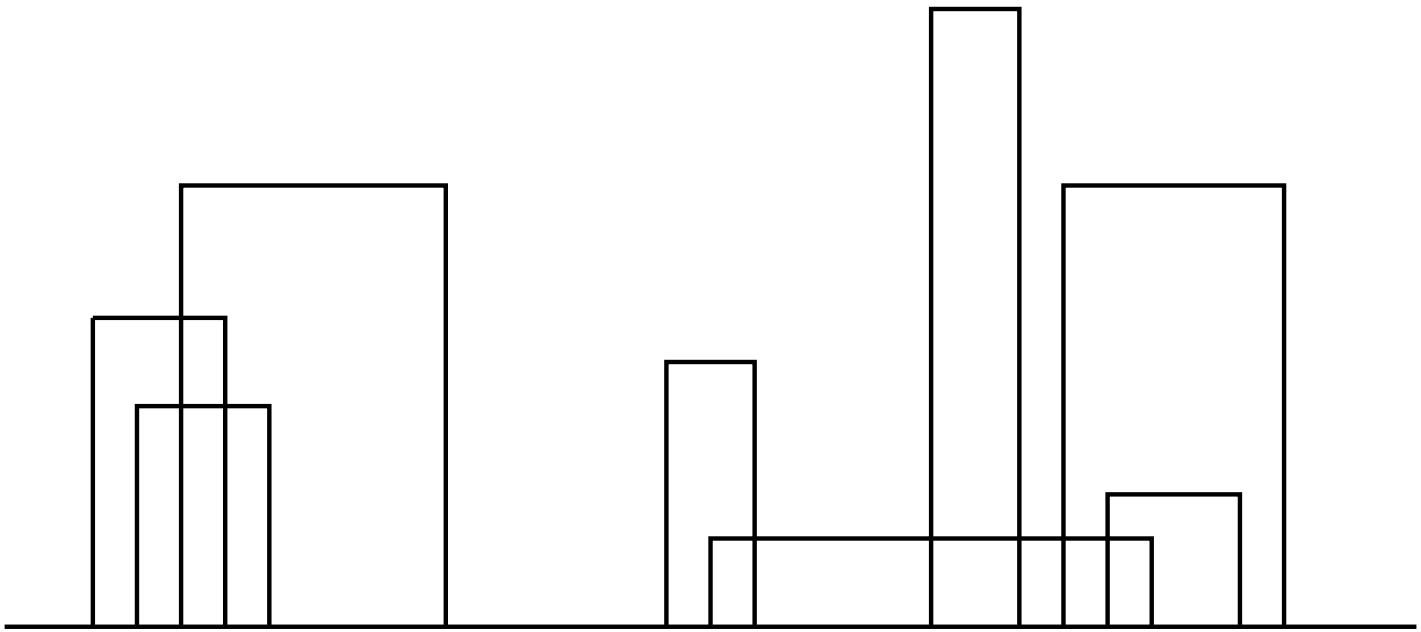
Entrada: Sequência de formas retangulares sobre o eixo x da forma (l, h, r) ordenadas por l.

Saída: Skyline.

SKYLINE

Exemplo

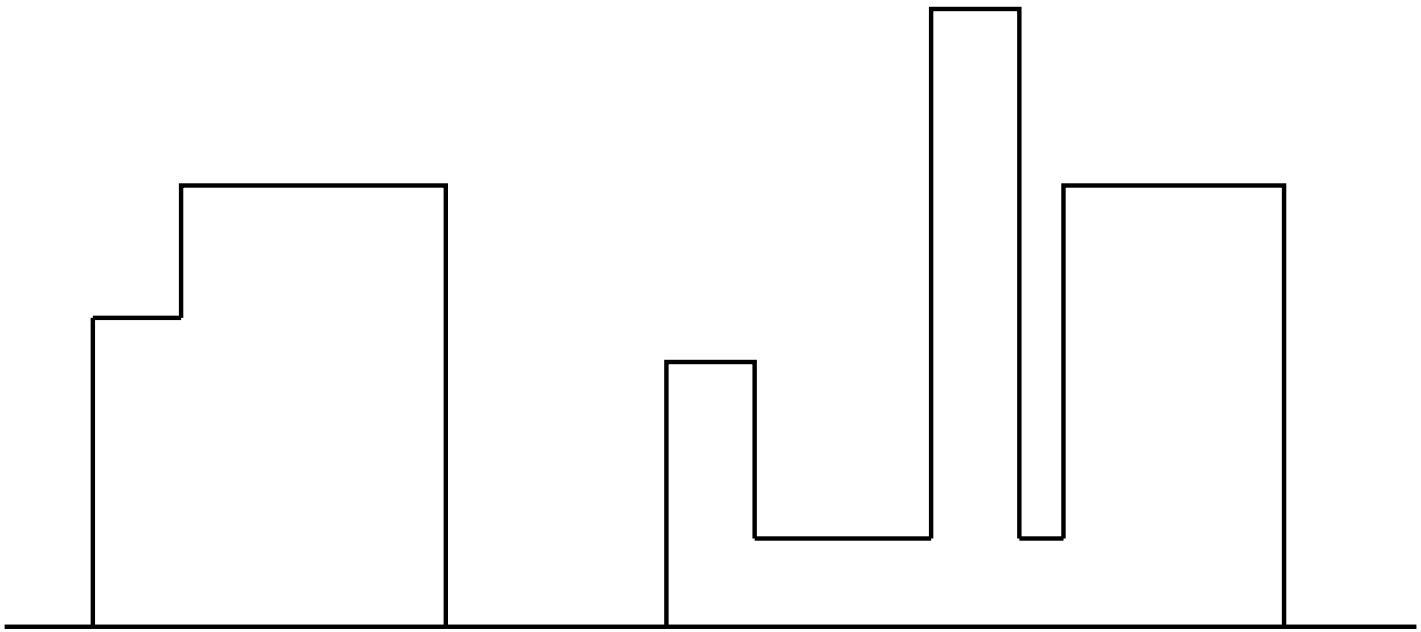
$$(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)$$

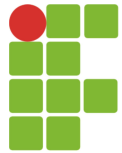


SKYLINE

Exemplo

$$(1, \mathbf{11}, 3, \mathbf{13}, 9, \mathbf{0}, 12, \mathbf{7}, 16, \mathbf{3}, 19, \mathbf{18}, 22, \mathbf{3}, 23, \mathbf{13}, 29)$$





SKYLINE

Caso Base

- Um prédio já é um skyline.

Hipótese de Indução

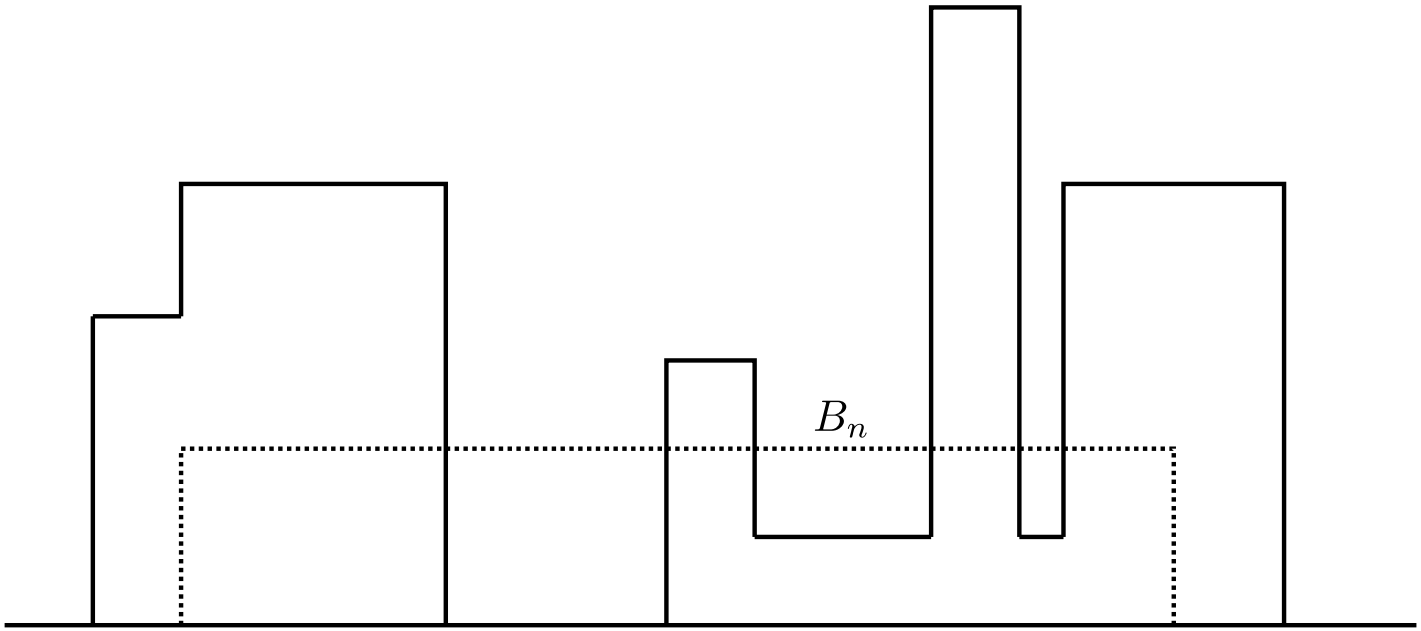
- Já temos o skyline para os $n-1$ primeiros prédios.

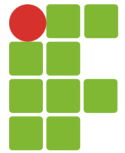
SKYLINE

Passo de Indução

- Incrementalmente, adicionar o prédio n.

$$(1, \mathbf{11}, 3, \mathbf{13}, 9, \mathbf{0}, 12, \mathbf{7}, 16, \mathbf{3}, 19, \mathbf{18}, 22, \mathbf{3}, 23, 13, 29, 0)$$



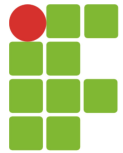


SKYLINE

Complexidade da Adição Incremental do Skyline

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

- $T(n) \in \Theta(n^2)$
- É muito custoso adicionar um prédio a um skyline de cada vez (insertionsort). É melhor juntar dois skylines.



SKYLINE

Caso Base

- Um prédio já é um skyline.

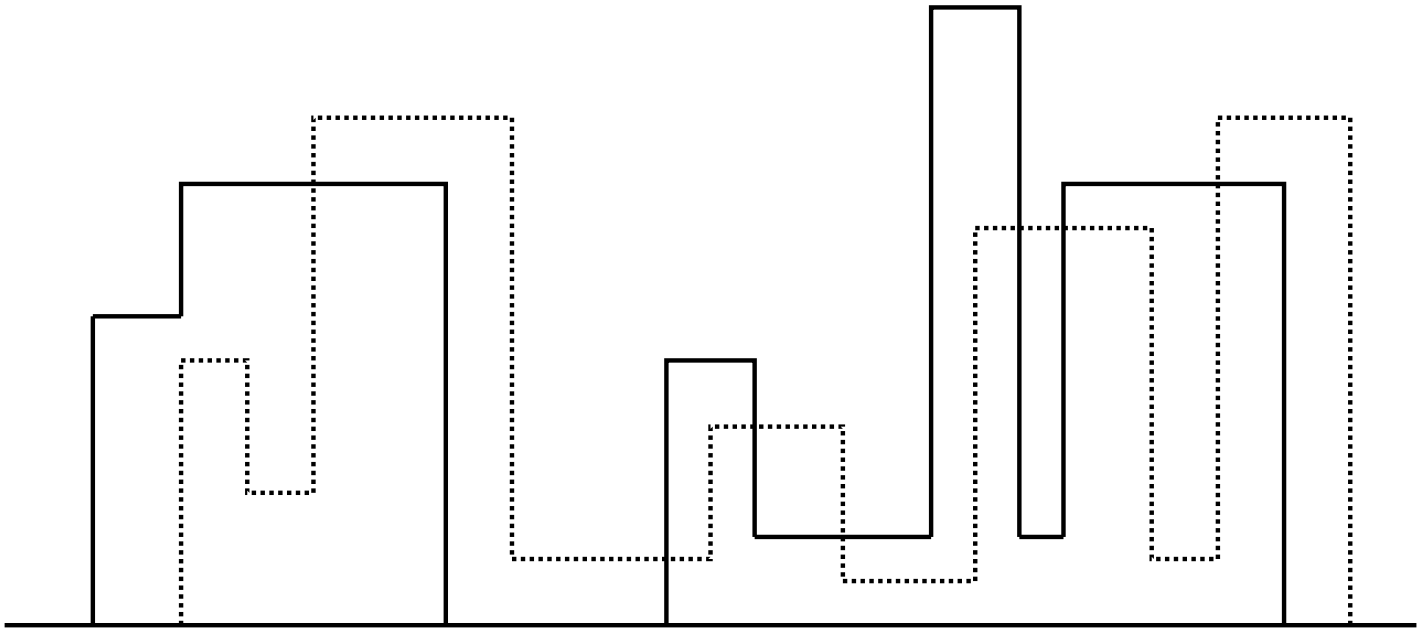
Hipótese de Indução

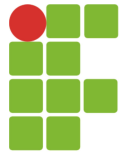
- Já temos dois skylines. Um para cada partição de $n/2$ prédios.

SKYLINE

Passo de Indução

- Fazer o merge dos dois skylines.



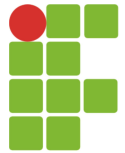


SKYLINE

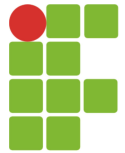
Complexidade da Adição de Dois Skylines

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

- $T(n) \in \Theta(n \lg n)$.



Sumário



CPPP

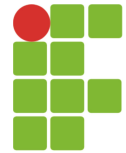
CPPP

Entrada: Conjunto de pontos

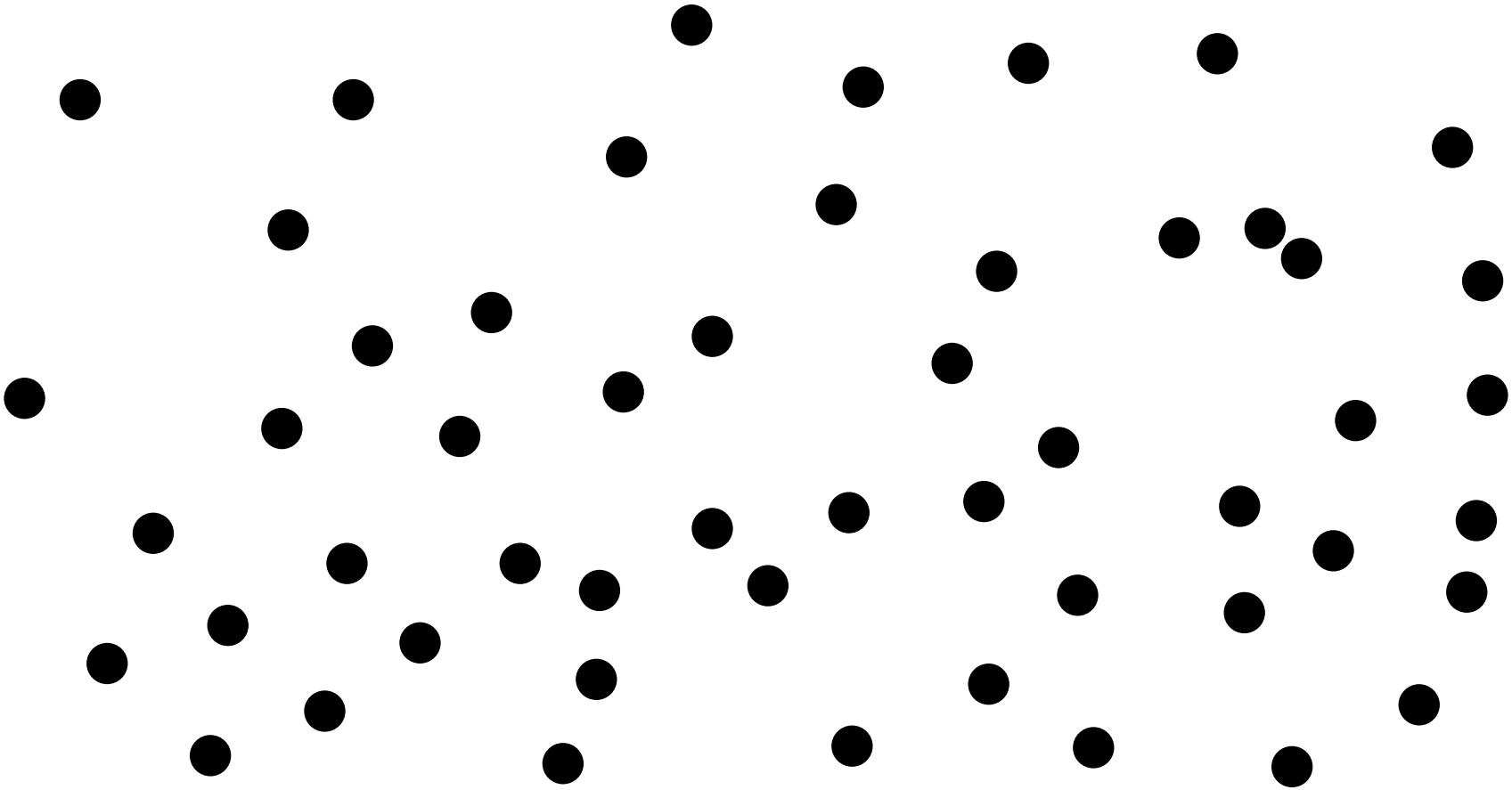
$$P = \{p_0, \ldots, p_{n-1}\} \subseteq \mathbb{R}^2$$

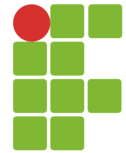
Saída:

$$(a, b) \in \arg\min_{x,y} \{d(x, y) | x, y \in P\}$$

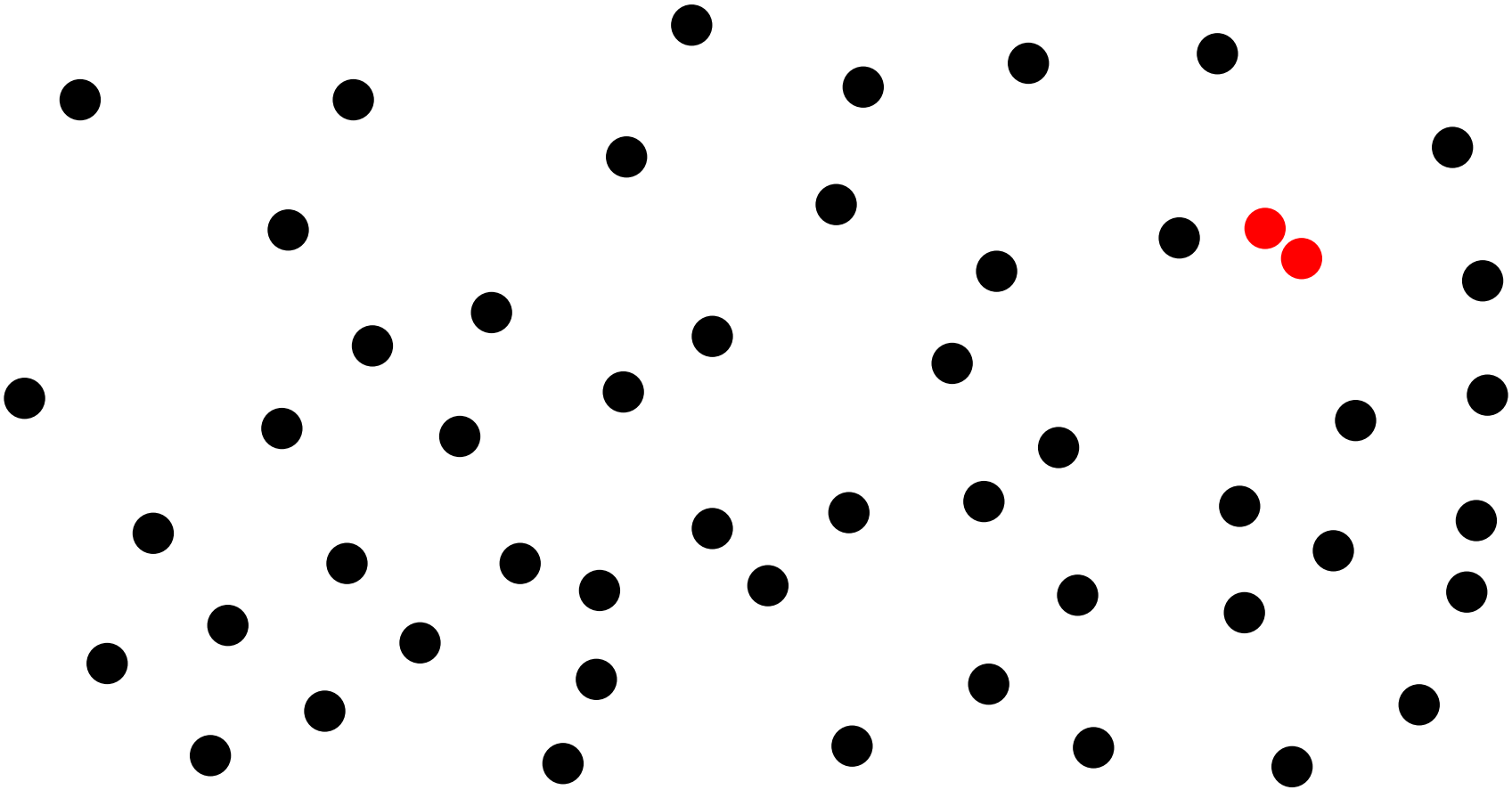


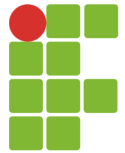
CPPP





CPPP

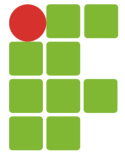




CPPP

Algoritmo Força-Bruta

- Devemos comparar os $\binom{n}{2}$ pares de pontos e selecionar os que possuem menor distância.



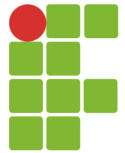
CPPP

Algorithm 5: CPPP-BRUTE-FORCE(P)

Input: P

Output: $(a, b) \in \arg\min_{x,y}\{d(x, y)|x, y \in P\}$

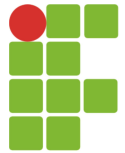
```
1 n ← P.SIZE()
2 dist ← ∞
3 for( i ← 0; i < n − 1; i++ )
4 |   for( j ← i + 1; j < n; j++ )
5 |   |   if( d(P[i], P[j]) < dist )
6 |   |   |   (x, y) ← (P[i], P[j])
7 |   |   |   dist ← d(P[i], P[j])
8 return (x, y)
```



CPPP

Complexidade do Algoritmo CPPP-BruteForce

Como devemos comparar a distância de $\binom{n}{2}$ pares, o algoritmo força-bruta leva tempo quadrático $\Theta(n^2)$.

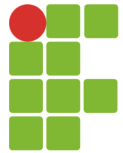


CPPP

Complexidade do Algoritmo CPPP-BruteForce

Como devemos comparar a distância de $\binom{n}{2}$ pares, o algoritmo força-bruta leva tempo quadrático $\Theta(n^2)$.

Podemos fazer melhor usando projeto por indução.



CPPP

Caso Base

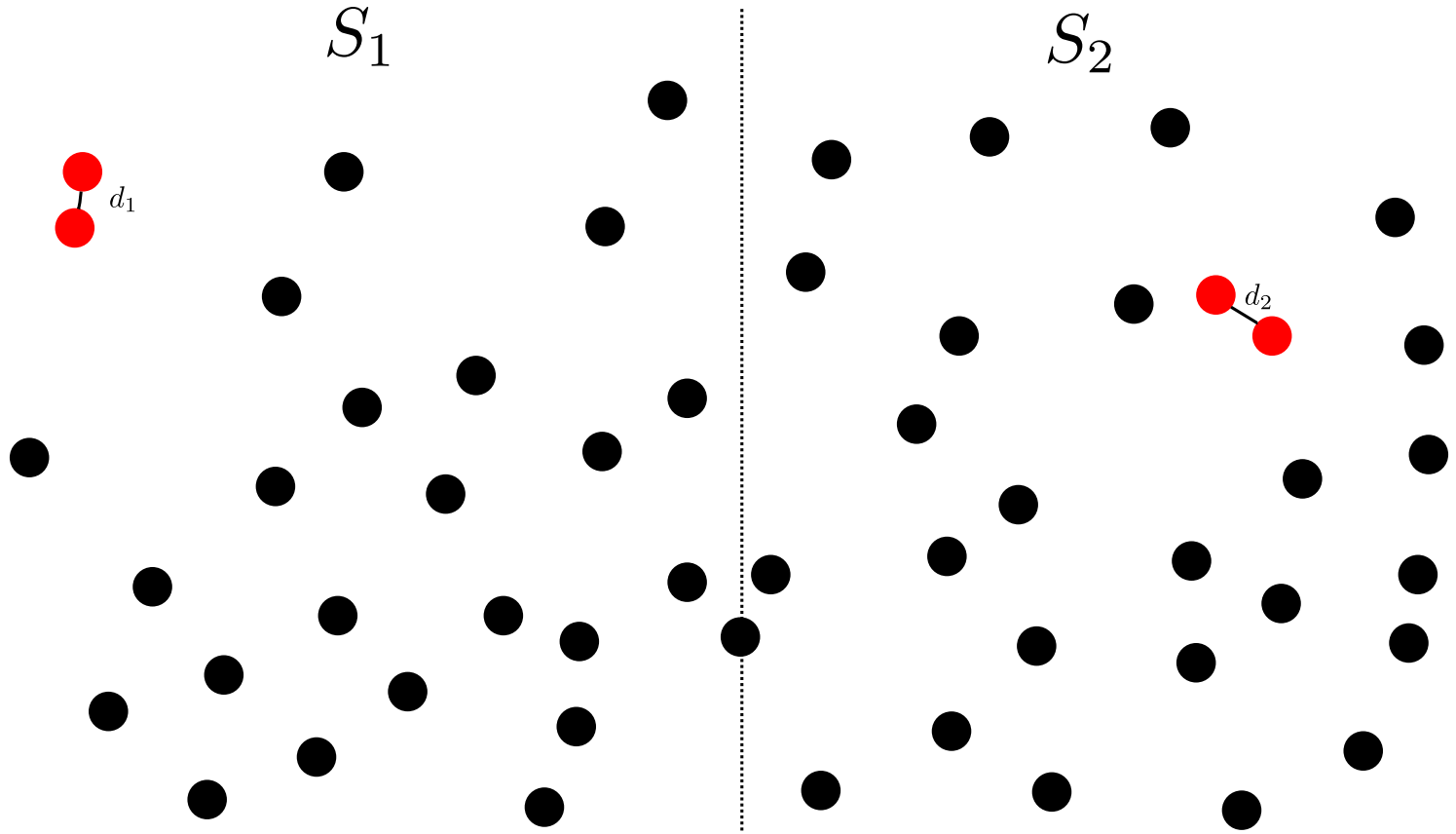
Estimar a menor distância entre dois ou três pontos é trivial.

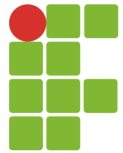
Hipótese de Indução

Sabemos a distância entre dois pontos para uma partição de tamanho $n/2$.

CPPP

Passo de Indução





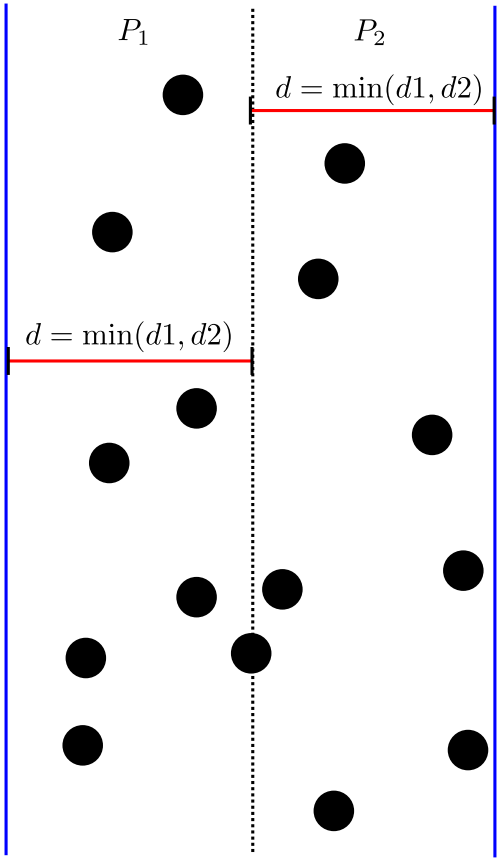
CPPP

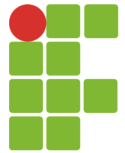
Passo de Indução

- Sabemos que S_1 dá como resposta um par de pontos (a_1, b_1) com distância d_1 e que S_2 dá como resposta um par de pontos (a_2, b_2) com distância d_2.
- Falta descobrir o par de pontos com menor distância que estejam nas partições S_1 e S_2.
- O que podemos dizer sobre os pontos (a_3, b_3) entre as partições S_1 e S_2?

CPPP

Passo de Indução





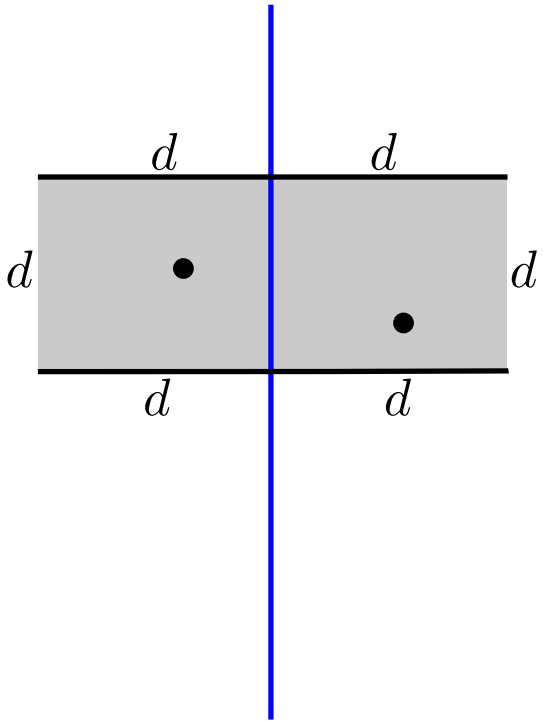
CPPP

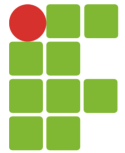
Passo de Indução

- Ainda temos um problema. Podemos ter os $n/2$ pontos em P_1 e os outros $n/2$ pontos em P_2.
- No entanto, existe um fator crucial. Quantos pontos podemos ter com distância $d = \min(d_1, d_2)$ de um determinado ponto de uma partição?

CPPP

Passo de Indução

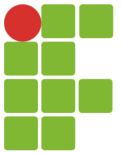




CPPP

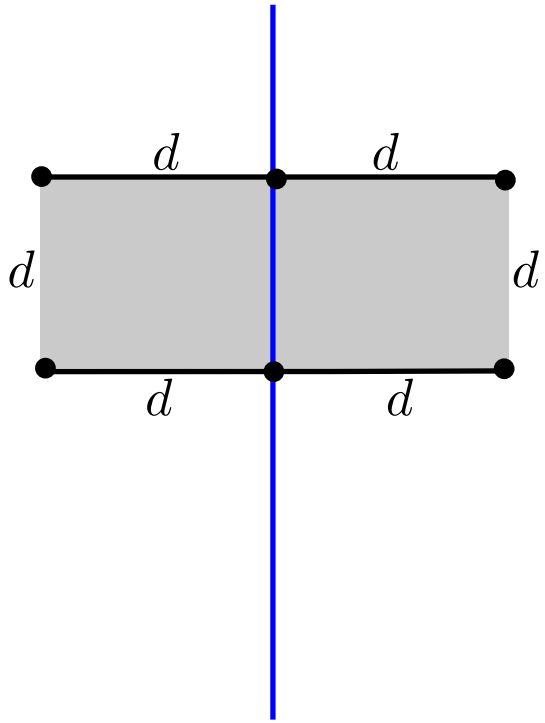
Passo de Indução

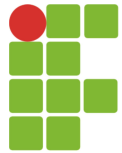
- Se existem dois pontos em duas partições diferentes com distância $\leq d$, eles tem que estar dentro de um retângulo $d \times 2d$.
- Quantos pontos podemos ter em um retângulo $d \times 2d$.



CPPP

Passo de Indução



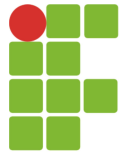


CPPP

Algorithm 6: CPPP(P)

Input: P
Output: $(a, b) \in \arg\min_{x,y}\{d(x, y)|x, y \in P\}$

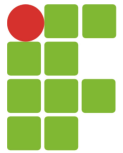
```
1  n ← P.SIZE()
2  if( n ≤ 3 )
3      return CPPP-BRUTE-FORCE(P)
4  P_x = SORT-BY-X(P)
5  P_y = SORT-BY-Y(P)
6  x_m ← ⌊n/2⌋
7  (a_1, b_1) ← CPPP(P_x[0, x_m])
8  (a_2, b_2) ← CPPP(P_x[x_m + 1, n − 1])
9  d_3 ← min{d(a_1, b_1), d(a_2, b_2)}
10 (a_3, b_3) ← arg min{d(a_1, b_1), d(a_2, b_2)}
11 D_y ← {p ∈ P_y | |p.x − x_m.x| < d_3}
12 for( i = 0; i < D_y.SIZE(); i++ )
13     for( j = i + 1; j < D_y.SIZE() ∧ |D[i].y − D[j].y| < d; j++ ) Máximo de 5 vizinhos
14         if( d(P[i], P[j]) < d_3 )
15             d_3 ← d(P[i], P[j])
16             (a_3, b_3) ← (P[i], P[j])
17 return (a_3, b_3)
```



CPPP

Complexidade do Algoritmo CPPP

- Dividimos as partições em duas de tamanho $n/2$ e realizamos um esforço da ordem de $\Theta(n \lg n)$ pelos seguintes motivos:
  - Ordenar pelas abcissas e ordenadas leva tempo $\Theta(n \lg n)$
  - Selecionar os pontos ordenados pela ordenada que estão à esquerda ou à direita da partição leva tempo $\Theta(n)$.
  - Selecionar os pontos que caem dentro da faixa da distância x para a divisa da partição leva tempo $\Theta(n)$.
  - Precisamos checar no máximo 5 vizinhos.



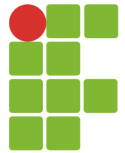
CPPP

Complexidade do Algoritmo CPPP

- Portanto, a relação de recorrência é:

$$T(n) = \begin{cases} \Theta(1), & n \leq 3 \\ 2T(n/2) + \Theta(n \lg n) \end{cases}$$

- Pelo método da substituição é possível provar que o tempo de pior caso do algoritmo tem custo $T(n) \in \Theta(n \lg^2 n)$.



CPPP

Complexidade do Algoritmo CPPP

- É possível reduzir para $\Theta(n \lg n)$?
- Sim. Basta reforçar a hipótese de indução.
  - Sabemos achar os pontos mais próximos das duas partições **E** temos os pontos dessas partições ordenados pelas abcissas e ordenadas.
- Realizamos o **merge** em tempo $\Theta(n)$.

$$T(n) = \begin{cases} \Theta(1), & n \leq 3 \\ 2T(n/2) + \Theta(n) \end{cases}$$

- $T(n) \in \Theta(n \lg n)$.