Projeto e Análise de Algoritmos

Prof. Dr. Ednaldo B. Pizzolato

www.dc.ufscar.br/~ednaldo ednaldo@dc.ufscar.br

Força bruta e Busca exaustiva

- Introdução
- Busca sequencial e String matching
- Os problemas Closest-pair e Convex-hull
- Closest-pair e Convex-hull por força bruta
- Busca exaustiva
  - Traveling Salesman Problem (TSP)
  - Knapsack Problem
  - Casamento
  - Busca em largura e em profundidade

Força bruta e Busca exaustiva

- Introdução
  - Força bruta é uma abordagem direta para resolver um problema, normalmente baseada na definição do problema e dos conceitos envolvidos.

- A força, no caso, é desempenhada pelo computador e não pela inteligência. Seria "baixe a cabeça e siga"!

Força bruta e busca exaustiva

- Como exemplo, vamos abordar o problema da exponenciação (tem também o algoritmo da multiplicação de matrizes...).

$$a^n = \underbrace{a * \ldots * a}_{n}$$

- Força bruta pode ser aplicada em uma vasta gama de problemas e pode ser uma importante estratégia de solução de problemas.

Força bruta e Busca exasutiva

- Em muitas situações o problema deve ser resolvido para algumas instâncias e a estratégia resolve muito bem isso em um tempo aceitável.
- Algoritmos de ordenação (abordagens mais diretas):
  - Selection sort
  - Bubble sort

Força bruta e Busca exasutiva

- Selection sort
  - Busca na lista pelo menor elemento e troca com a primeira posição;
  - Repetir o processo da 2ª posição em diante. (n-1) repetições

Algoritmo

para i ← 0 até n-2 faça

min ← i

para j ← i+1 até n-1 faça

se A[j] < A[min] então min ← j

troca (A[i],A[min])

Força bruta e Busca exaustiva

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$\sum_{i=0}^{n-2} [(n-1)-(i+1)+1]$$

$$\sum_{i=0}^{n-2} (n-1+i) \qquad = \frac{(n-1)n}{2}$$

O algoritmo é $O(n^2)$, mas o número de trocas (swaps) é $O(n)$

Força bruta e Busca exaustiva

- Bubble sort
- Comparar elementos adjacentes na lista e trocá-los se estiverem fora de ordem.
- Repetir o processo n-1 vezes.

Algoritmo

para i ← 0 até n-2 faça

para j ← 0 até n-2-i faça

se A[j+1] < A[j] então

troca (A[j],A[j+1]

Força bruta e Busca exaustiva

- O algoritmo é $O(n^2)$ e, no pior caso, faz n^2 trocas (qual é o pior caso?)
- Como geralmente acontece, a primeira versão do algoritmo de força bruta pode ser melhorada através de um esforço modesto. Qual seria a melhoria neste caso?

Exercícios

- Projete um algoritmo (através da estratégia de força bruta) para resolver

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

Solução 1

- Solução por força bruta

```
soma ← 0
para i←n até 0 faça
  potencia ←1
  para j←1 até i faça
     potencia ← potencia ∗ x
// calcula xⁿ
   soma ← soma + a[i] ∗
potencia
retorna soma
```

Solução 1

- Solução por força bruta

```
soma ← 0
para i←n até 0 faça
   potencia ←1
   para j←1 até i faça
      potencia ← potencia ∗ x // calcula x^n
   soma ← soma + a[i] ∗ potencia
retorna soma
```

- Teria uma solução melhor?

Solução 1

- Solução por força bruta

soma ← a[0]

potencia ←1

para i ← 1 até n faça

    potencia ← potencia ∗ x

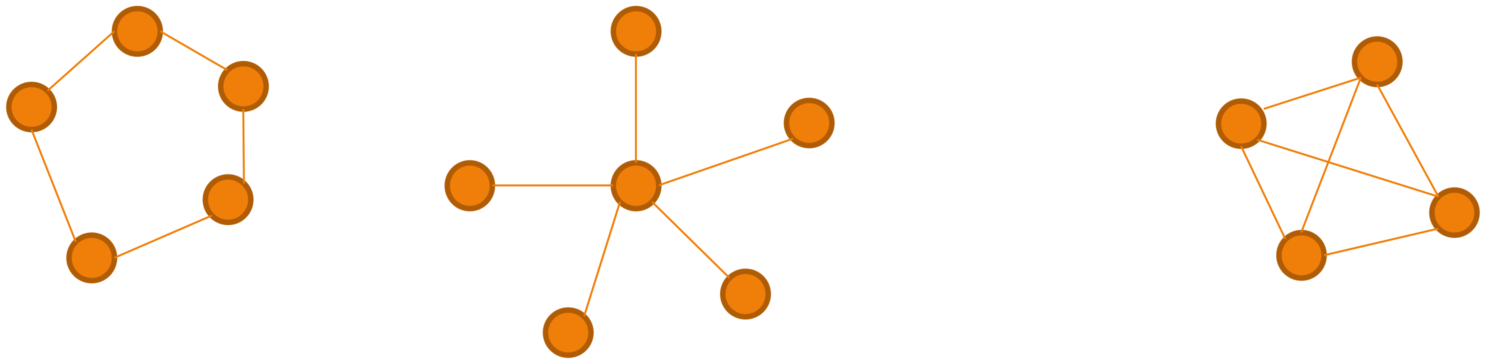
    soma ← soma + a[i] ∗ potencia

retorna soma

- Teria uma solução melhor?
- SIM!!!

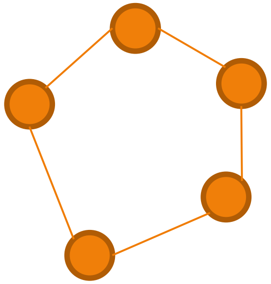
Exercícios

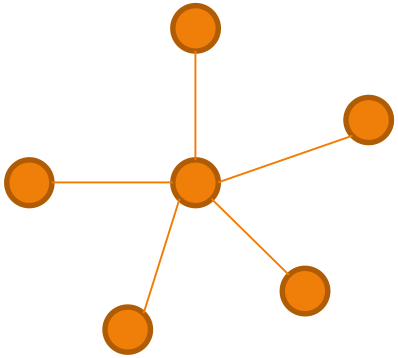
- Uma topologia em rede é um grafo que representa como os computadores ou outros dispositivos se interconectam. Na figura a seguir existem 3 configurações de topologia:

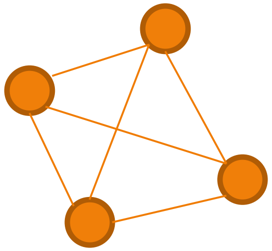


Exercícios

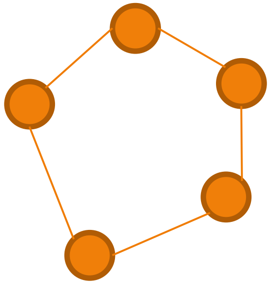
Uma matriz com 0s e 1s, n > 3, é responsável por indicar a topologia da matriz. Faça um algoritmo (força bruta) que identifique, através da matriz, qual topologia representa. Qual é a eficiência do algoritmo?





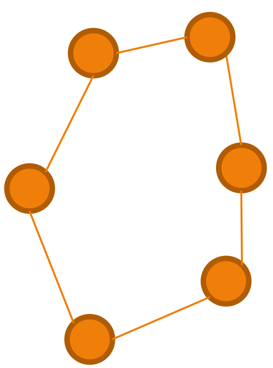


Exercícios



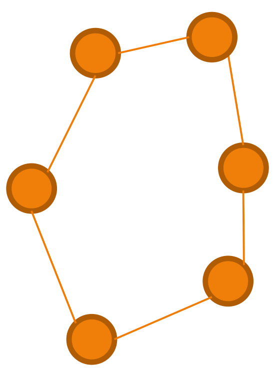
	A_1	A_2	A_3	A_4	A_5
A_1	0	1	0	0	1
A_2	1	0	1	0	0
A_3	0	1	0	1	0
A_4	0	0	1	0	1
A_5	1	0	0	1	0

Exercícios



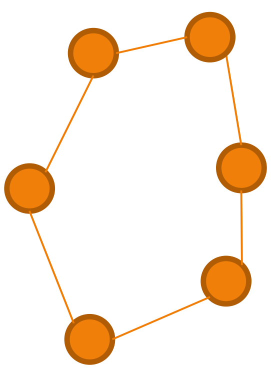
	A_1	A_2	A_3	A_4	A_5	A_6
A_1	0	1	0	0	0	1
A_2	1	0	1	0	0	0
A_3	0	1	0	1	0	0
A_4	0	0	1	0	1	0
A_5	0	0	0	1	0	1
A_6	1	0	0	0	1	0

Exercícios



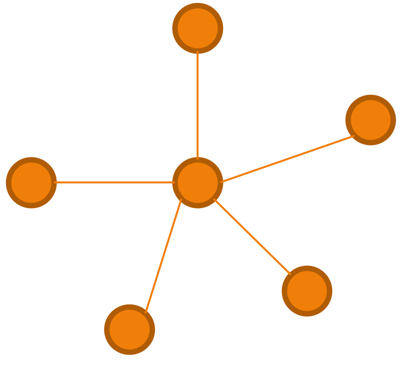
	A_1	A_2	A_3	A_4	A_5	A_6
A_1	0	1	0	0	0	1
A_2	1	0	1	0	0	0
A_3	0	1	0	1	0	0
A_4	0	0	1	0	1	0
A_5	0	0	0	1	0	1
A_6	1	0	0	0	1	0

Exercícios



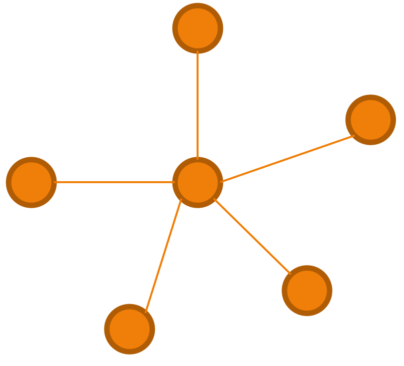
	A_1	A_2	A_3	A_4	A_5	A_6
A_1	0	1	0	0	0	1
A_2	1	0	1	0	0	0
A_3	0	1	0	1	0	0
A_4	0	0	1	0	1	0
A_5	0	0	0	1	0	1
A_6	1	0	0	0	1	0

Exercícios



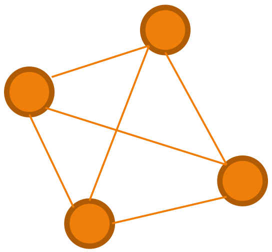
	A_1	A_2	A_3	A_4	A_5	A_6
A_1	0	1	1	1	1	1
A_2	1	0	0	0	0	0
A_3	1	0	0	0	0	0
A_4	1	0	0	0	0	0
A_5	1	0	0	0	0	0
A_6	1	0	0	0	0	0

Exercícios



	A_1	A_2	A_3	A_4	A_5	A_6
A_1	0	1	1	1	1	1
A_2	1	0	0	0	0	0
A_3	1	0	0	0	0	0
A_4	1	0	0	0	0	0
A_5	1	0	0	0	0	0
A_6	1	0	0	0	0	0

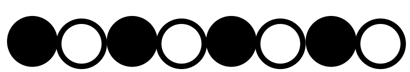
Exercícios



	A_1	A_2	A_3	A_4
A_1	0	1	1	1
A_2	1	0	1	1
A_3	1	1	0	1
A_4	1	1	1	0

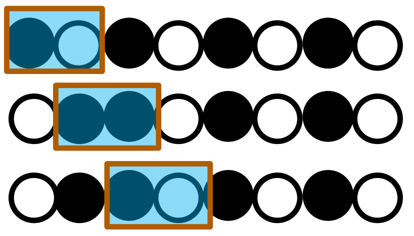
Exercícios

- Existe um número par (2n) de bolas pretas e brancas, sendo n pretas e n brancas. Elas estão intercaladas de forma a ter preta, branca, preta, branca...Sua missão é desenhar um algoritmo que faça com que as pretas fiquem do lado direito e as brancas do lado esquerdo. O único movimento permitido é a troca de bolas vizinhas.



- Qual o número de trocas?

Exercícios



Passo 1

Exercícios

```
int main()
{
    int i, j, k, temp;
    int a[20] = {1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0};
    printf("Discos alternados\n");
    // imprimindo os discos
    for (i=0;i<20;i++)
      printf("%2d ",a[i]);
    printf("\n");

    for (i=0;i<11;i++)
    {
       for (j=0;j<20; j++)
          if (a[j] > a[j+1])
          {
              temp = a[j];
              a[j] = a[j+1];
              a[j+1] = temp;
          }
```

```
...
       printf("==========\n");
       // imprimindo os discos
       for (k=0;k<20;k++)
          printf("%2d ",a[k]);
       printf("\n");
       getchar();
    }
    return 0;
}
```

Trocas = n.(n+1)/2

Força bruta e Busca exaustiva

- Introdução
- Busca sequencial e String matching
- Os problemas Closest-pair e Convex-hull
- Closest-pair e Convex-hull por força bruta
- Busca exaustiva
  - Traveling Salesman Problem (TSP)
  - Knapsack Problem
  - Casamento
  - Busca em largura e em profundidade

Busca sequencial

- Dada uma lista de n elementos e uma chave k, a busca pela chave na lista através da força bruta compreende comparar cada elemento da lista com a chave e:
  - Terminar a busca se encontrou a chave; ou
  - Terminar a busca se encontrou o final da lista.

Busca sequencial

i ← 0

enquanto i < n E A[i] ≠ k faça

  i ← i + 1

se i < n então

  retorna i

senão

  retorna -1

Busca sequencial

melhoria

i ← *0*

A[n] ← *k*

enquanto A[i] ≠ k faça

    i ← *i + 1*

se i < n então

    retorna i

senão

    retorna -1

Busca sequencial

```
i ← 0
enquanto i < n E A[i] ≠ k faça
        i ← i + 1
se i < n então
   retorna i
senão
   retorna -1
```

```
i ← 0
A[n] ← k
enquanto A[i] ≠ k faça
     i ← i + 1
se i < n então
   retorna i
senão
   retorna -1
```

Qual outra melhoria você faria?

Busca sequencial

```
i ← 0
enquanto i < n E A[i] ≠ k faça
        i ← i + 1
se i < n então
   retorna i
senão
   retorna -1
```

```
i ← 0
A[n] ← k
enquanto A[i] ≠ k faça
        i ← i + 1
se i < n então
   retorna i
senão
   retorna -1
```

Qual outra melhoria você faria?

R.: busca em lista ordenada

String matching

Dado um texto de n caracteres e uma palavra de m caracteres (m ≤ n), o objetivo é encontrar a posição da letra inicial da palavra no texto.

“Conforme a tradição para grandes contratações do Real Madrid, Gareth Bale foi apresentado no gramado do Santiago Bernabéu na manhã...”

→ grandes

String matching

c	o	n	f	o	r	m	e		a		t	r	a	d	i	ç	ã	o

g	r	a	n	d	e	s

String matching

c	o	n	f	o	r	m	e		a		t	r	a	d	i	ç	ã	o
	g	r	a	n	d	e	s											

String matching

c	o	n	f	o	r	m	e		a		t	r	a	d	i	ç	ã	o
		g	r	a	n	d	e	s										

String Matching

Algoritmo

para i ← 0 até n-m faça

j ← 0

enquanto j < m E P[j] = T[i+j] faça

j ← j + 1

if j = m retorna i

retorna -1

P → palavra

T → texto

String Matching

- Para o pior caso teríamos uma palavra no texto quase idêntica à chave ocorrendo várias vezes. Isso ocasionaria n – m + 1 comparações o que faz com que o algoritmo seja O(m.n). Entretanto, para textos comuns o algoritmo é O(n).
- Existem algoritmos mais eficientes como o de Boyer-Moore e Horspool.

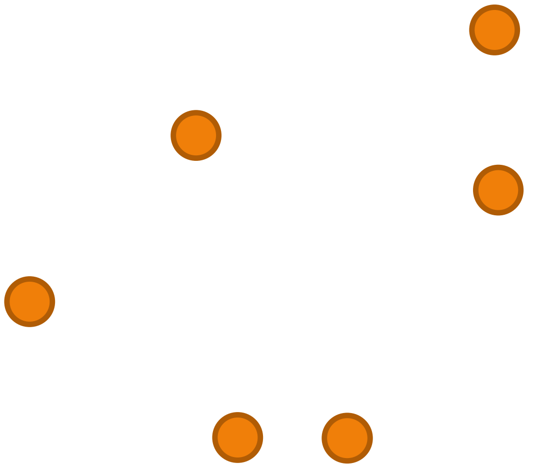
Força bruta e Busca exaustiva

- Introdução
- Busca sequencial e String matching
- Os problemas Closest-pair e Convex-hull
- Closest-pair e Convex-hull por força bruta
- Busca exaustiva
  - Traveling Salesman Problem (TSP)
  - Knapsack Problem
  - Casamento
  - Busca em largura e em profundidade

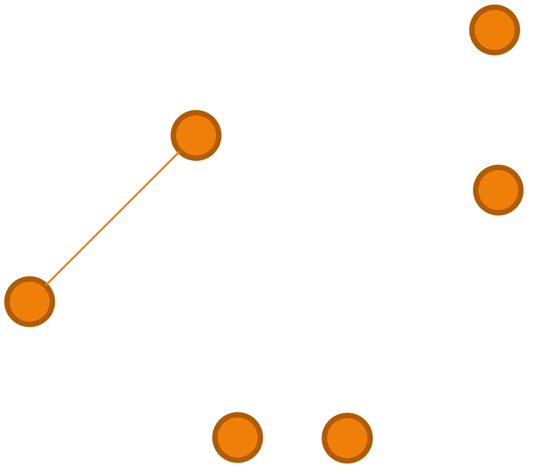
Closest-pair

- O problema se resume a encontrar 2 pontos mais próximos dentro de um conjunto de n pontos.
- Está relacionado com a geometria computacional.
- Pontos podem ser aeronaves, agências de correios, objetos, sequências de DNA...

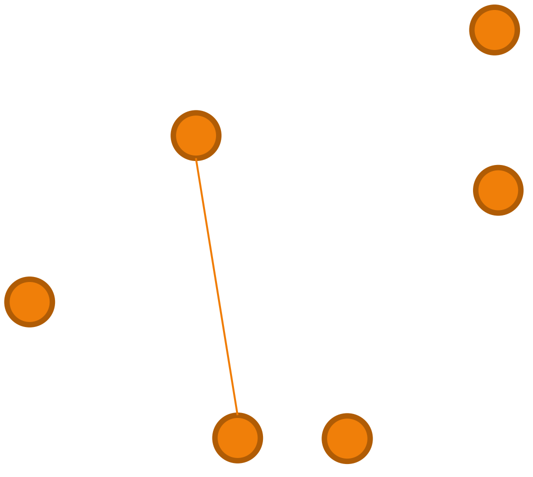
Closest-pair



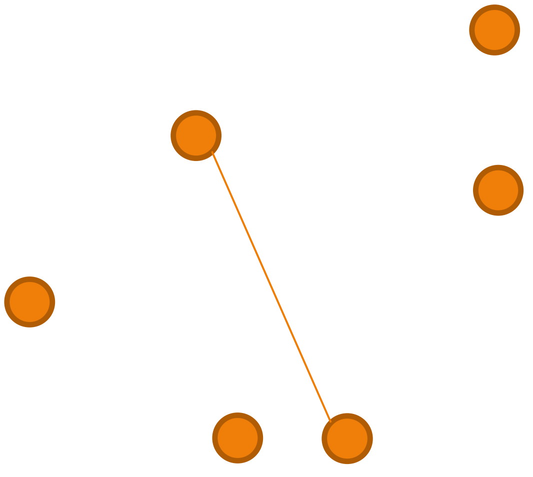
Closest-pair



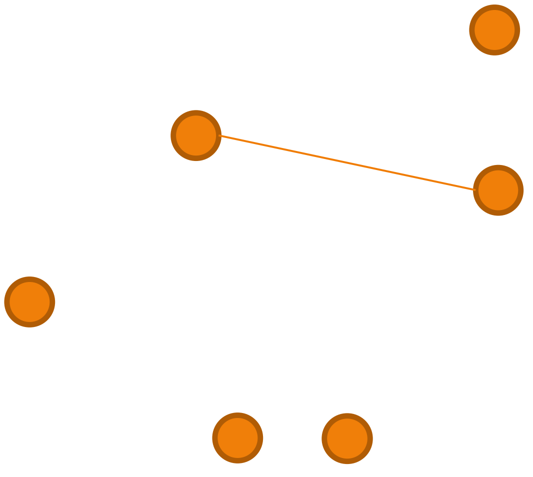
Closest-pair



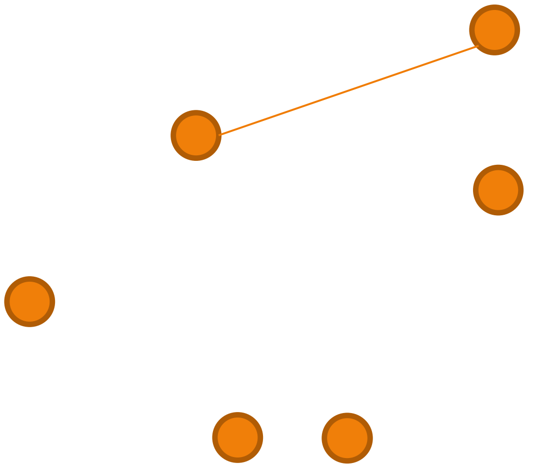
Closest-pair



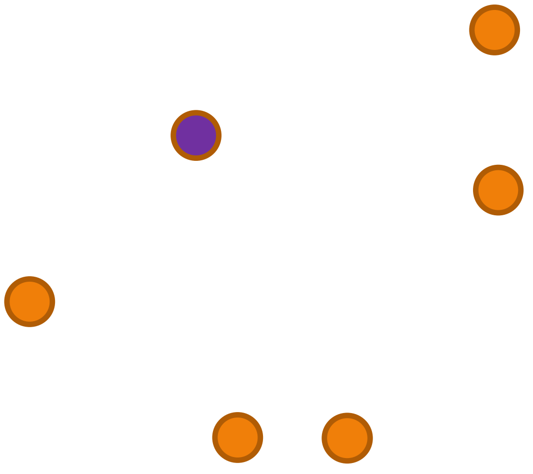
Closest-pair



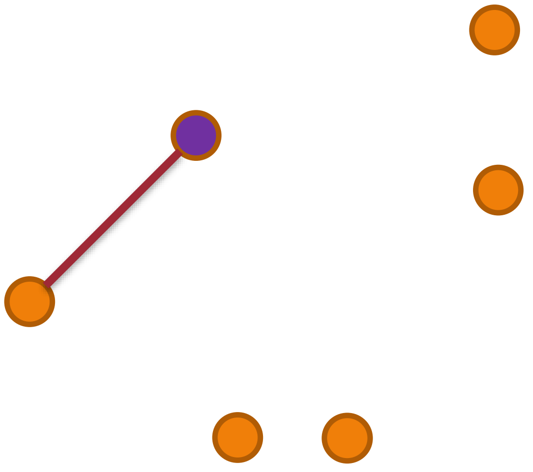
Closest-pair



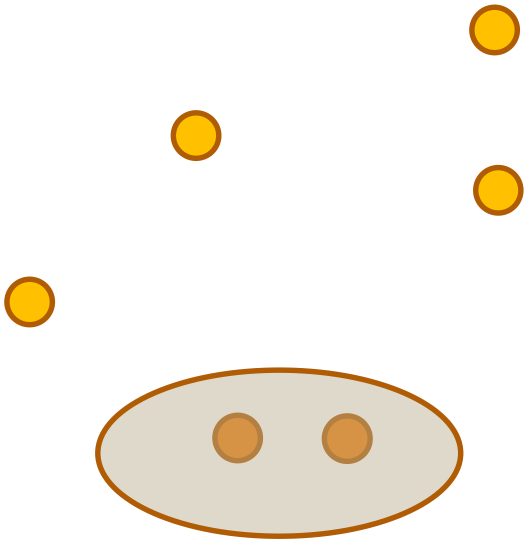
Closest-pair



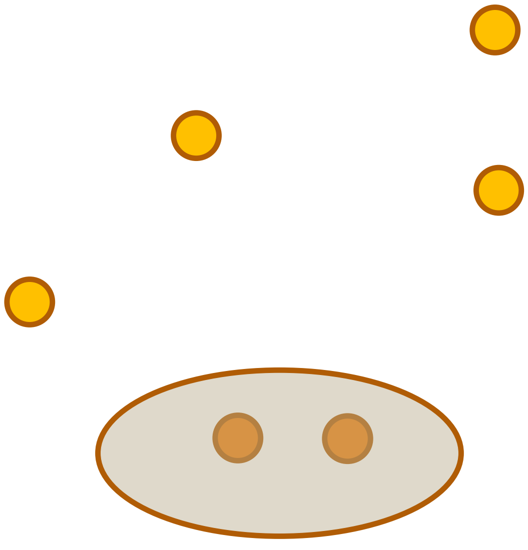
Closest-pair



Closest-pair



Closest-pair



$d \leftarrow \infty$

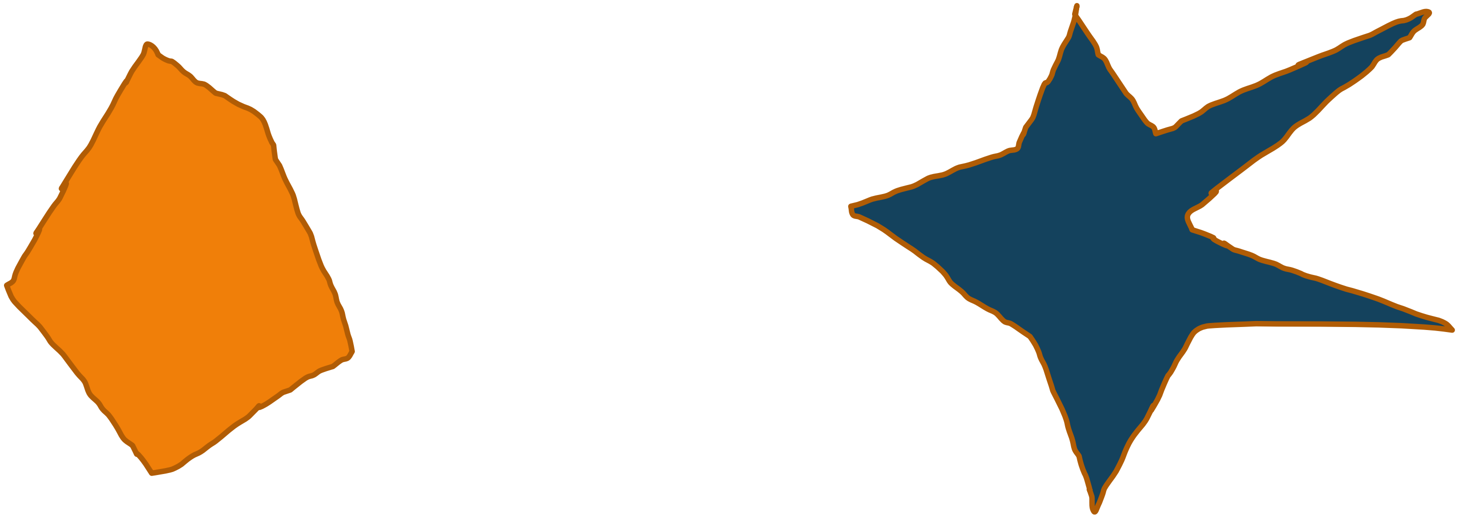
para $i \leftarrow 1$ *até* $n-1$ *faça*

    para $j \leftarrow i+1$ *até* n *faça*

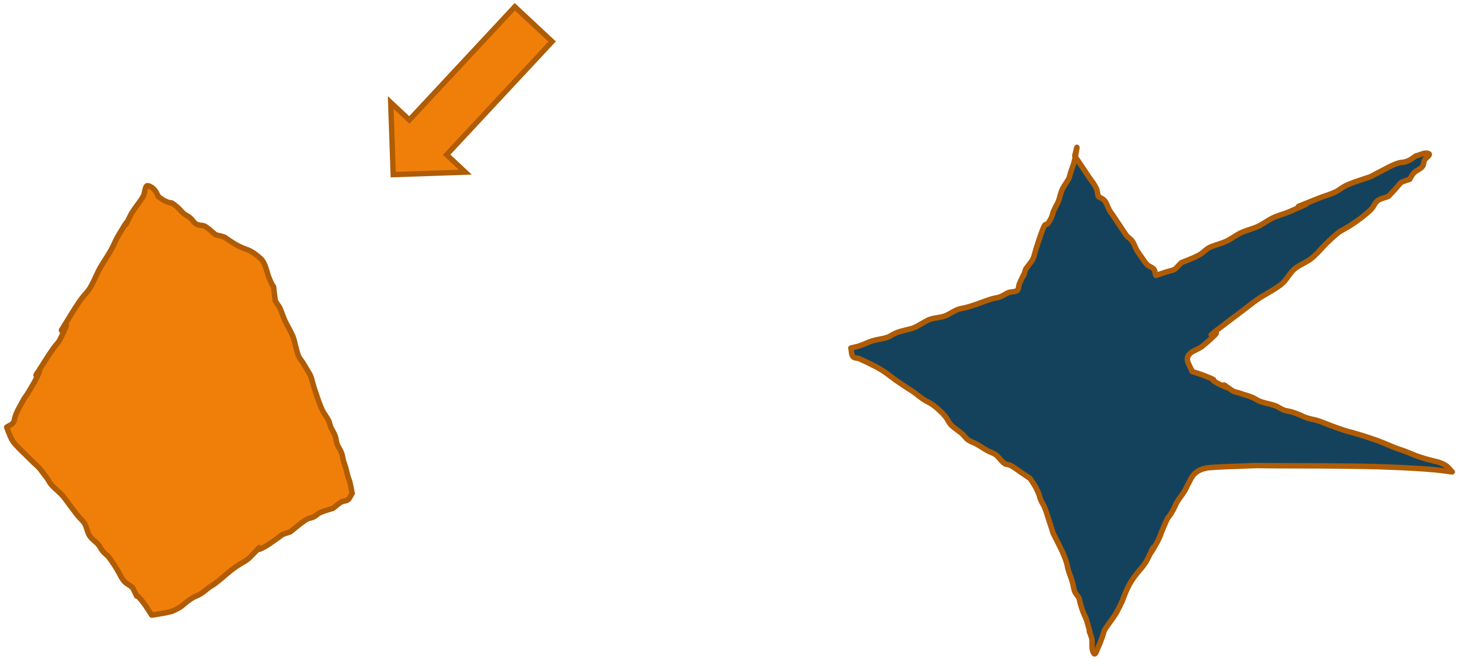
        $d \leftarrow \min(d, \mathrm{dist}(x_i, x_j))$

retorna d

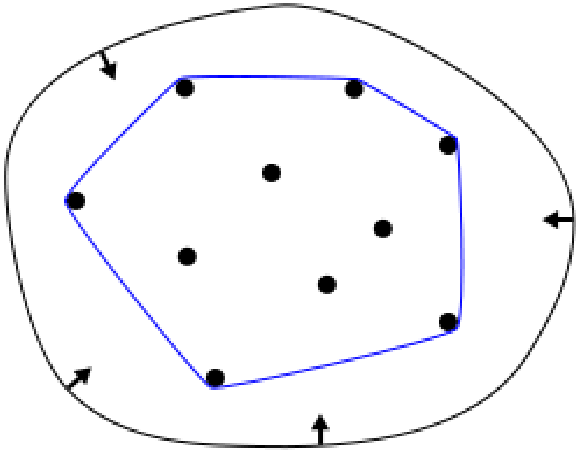
Convex-Hull



Convex-Hull

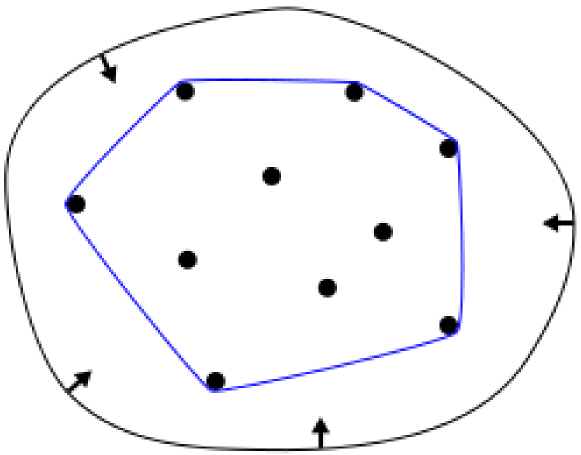


Convex-hull



Definição: Convex hull de um conjunto S de pontos é a menor figura geométrica convexa que contem S.

Convex-hull



Teorema: Uma convex-hull de qualquer conjunto S de n pontos (n > 2) sendo que nem todos estejam em uma mesma linha é um polígono convexo com vértices em alguns dos pontos de S.

Exercícios

O problema de closest-pair pode ser estendido para k dimensões, em que a distância Euclidiana entre 2 pontos p' $(x'_1, x'_2, \ldots x'_k)$ e p" $(x''_1, x''_2, \ldots x''_k)$ é definida da seguinte forma:

$$d(p', p'') = \sqrt{\sum_{s=1}^{k} (x'_s - x''_s)^2}$$

Qual é a eficiência do algoritmo de força bruta para este problema?

Exercícios

Como é um espaço k-dimensional, haverá um loop para computar as distâncias de cada uma das dimensões:

$$d(p', p'') = \sqrt{\sum_{s=1}^{k} (x'_s - x''_s)^2}$$

Exercícios

Além disso, temos n pontos que devem ser comparados com os outros. Precisamos ter um cuidado para não fazer comparações repetidas...

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{s=1}^{k} 1 = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} k = k.\sum_{i=1}^{n-1} (n-i) = \frac{k.(n-1).n}{2} \in \Theta(kn^2)$$

Força bruta e Busca exaustiva

- Introdução
- Busca sequencial e String matching
- Os problemas Closest-pair e Convex-hull
- Closest-pair e Convex-hull por força bruta
- Busca exaustiva
  - Traveling Salesman Problem (TSP)
  - Knapsack Problem
  - Casamento
  - Busca em largura e em profundidade

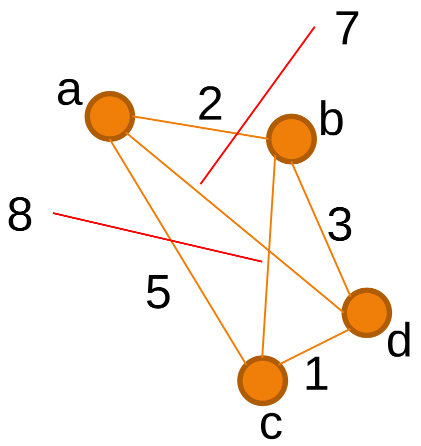
Busca exaustiva

- Muitos problemas importantes requerem encontrar um elemento com uma propriedade especial em um domínio que cresce exponencialmente (ou mais rápido) dada uma instância com seu respectivo tamanho.
- Tipicamente, tais problemas aparecem/ocorrem em situações que envolvem objetos combinatoriais tais como permutações, combinações e subconjuntos de um dado conjunto.

Busca exaustiva

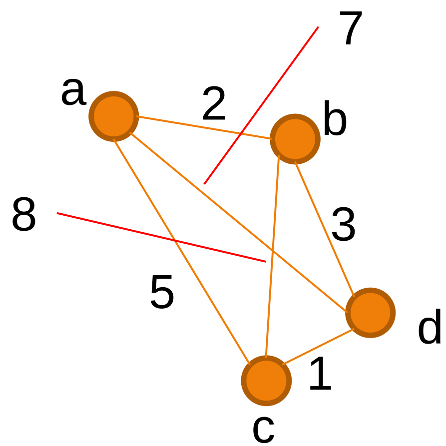
Definição: busca exaustiva é simplesmente a aplicação da estratégia de força bruta a problemas de natureza combinatorial. Isso significa que, na prática, o algoritmo deve gerar cada elemento do domínio do problema, selecionar aqueles que satisfazem alguma condição e achar o(s) elemento(s) desejado(s).

Traveling Salesman Problem



- O problema se resume a encontrar o percurso mais curto (menor custo) que percorra n cidades exatamente uma vez antes de retornar ao ponto inicial.

Traveling Salesman Problem



a→b→c→d→a => 18

a→b→d→c→a => 11 ok!

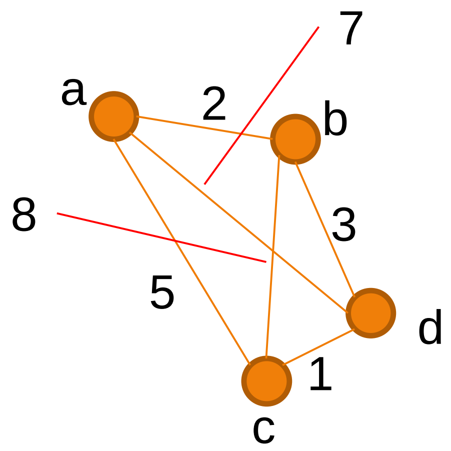
a→c→b→d→a => 23

a→c→d→b→a => 11 ok!

...

Este é um problema de grafos chamado de circuito Hamiltoniano.

Traveling Salesman Problem



Para estes casos, pode-se:

1.) gerar todas as permutações de n-1 cidades intermediárias;

2.) obter o tamanho dos percursos de cada permutação;

3.) encontrar o menor percurso.

Traveling Salesman Problem

No exemplo apresentado, ficou claro que algumas permutações são simétricas. Mesmo fazendo alguma melhoria no algoritmo de força bruta para evitar recálculos desnecessários, tem-se que a eficiência do algoritmo é $\frac{1}{2}(n-1)!$

Isso significa que o algoritmo de força bruta resolve problemas de tamanho n pequeno...

Traveling Salesman Problem - Exercício

Considere um computador que faça 10 bilhões de adições por segundo. Estime o número máximo de cidades em que seria possível resolver o problema TSP em:

a) 1 hora
b) 1 dia
c) 1 ano

Traveling Salesman Problem - Exercício

para 1 hora temos:

$$\frac{1}{2}n!10^{-9} \leq t$$

$$\frac{1}{2}n!10^{-9} \leq 1h = 3.6.10^{3}s$$

Traveling Salesman Problem - Exercício

para 1 hora temos:

$$n!10^{-9} \leq 1h = 7.2.10^{3}s$$

$$n! \leq 7.2.10^{12}$$

com n = 15 temos aprox. $1.3 \cdot 10^{12}$

Traveling Salesman Problem - Exercício

Assim, o número de cidades cujo problema TSP poderia ser resolvido em :

a) 1 hora 15
b) 1 dia 16
c) 1 ano 18

20 cidades só seriam resolvidas em 1 século!

Knapsack problem

Dados n itens de pesos conhecidos $w_1, w_2, ..., w_n$ e valores $v_1, v_2, ..., v_n$ e um saco com capacidade W, encontre o subconjunto mais valioso de itens que caibam no saco.

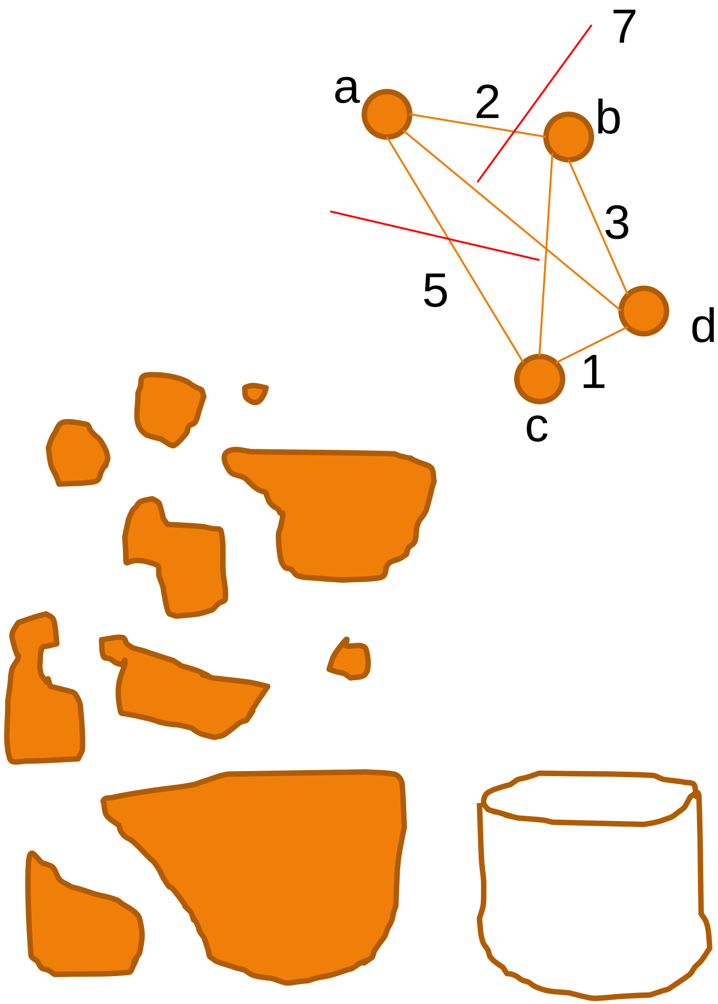
Knapsack problem

A busca exaustiva para este problema faz com que se gerem todos os subconjuntos possíveis que satisfaçam à condição. A partir dos subconjuntos, pode-se encontrar aquele mais valioso.

Knapsack problem

Mas, como o número de subconjuntos de um conjunto de n elementos é 2^n, a conclusão prática é que a busca exaustiva produz um algoritmo $\Omega(2^n)$ independentemente da forma mais ou menos eficiente que subconjuntos individuais são gerados.

Knapsack problem



Assim, o TSP e o KP, com o uso de busca exaustiva produz algoritmos extremamente ineficientes e são considerados bons exemplos da classe de problemas chamada de NP-hard, em que não existe algoritmo com tempo polinomial que os resolva.

Força bruta e Busca exaustiva

- Introdução
- Busca sequencial e String matching
- Os problemas Closest-pair e Convex-hull
- Closest-pair e Convex-hull por força bruta
- Busca exaustiva
  - Traveling Salesman Problem (TSP)
  - Knapsack Problem
  - Casamento
  - Busca em largura e em profundidade

Casamento/pareamento

	T_1	T_2	T_3	T_4
P_1	9	2	7	8
P_2	6	4	3	7
P_3	5	8	1	8
P_4	7	6	9	4

Imagine a situação de n pessoas procurando n empregos; ou n homens procurando casar com n mulheres...

Há uma matriz de custo que indica os valores para cada pareamento.

Casamento/pareamento

	T_1	T_2	T_3	T_4
P_1	9	2	7	8
P_2	6	4	3	7
P_3	5	8	1	8
P_4	7	6	9	4

(<1,1>, <2,2>, <3,3>,<4,4>) = 18

(<1,1>, <2,2>, <3,4>,<4,3>) = 30

(<1,1>, <2,3>, <3,2>,<4,4>) = 24

(<1,1>, <2,3>, <3,4>,<4,2>) = 26

(<1,1>, <2,4>, <3,2>,<4,3>) = 33

(<1,1>, <2,4>, <3,3>,<4,2>) = 23

...

Casamento/pareamento

	T_1	T_2	T_3	T_4
P_1	9	2	7	8
P_2	6	4	3	7
P_3	5	8	1	8
P_4	7	6	9	4

Desde que o número de permutações a serem consideradas é n!, a busca exaustiva é impraticável (serve apenas para valores bem pequenos de n).

Existe um algoritmo mais eficiente chamado Hungarian method.

Isso é uma exceção... Em geral não há soluções polinomiais para problemas que crescem exponencialmente.

Casamento/pareamento

Para o problema ao lado, identifique o casamento que resulta em menor custo.

	T_1	T_2	T_3	T_4
P_1	9	2	7	8
P_2	6	4	3	7
P_3	5	8	1	8
P_4	7	6	9	4

Casamento/pareamento

	T_1	T_2	T_3	T_4
P_1	9	2	7	8
P_2	6	4	3	7
P_3	5	8	1	8
P_4	7	6	9	4

Para o problema ao lado, identifique o casamento que resulta em menor custo.

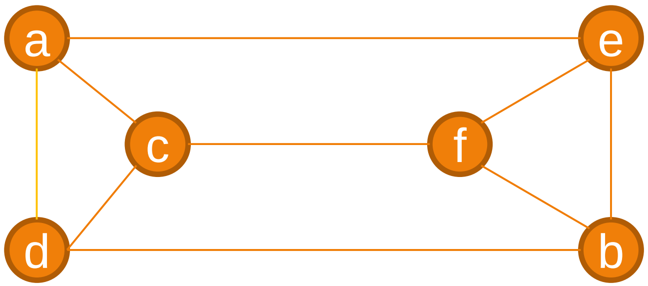
R.: 2 + 6 + 1 + 4 = 13

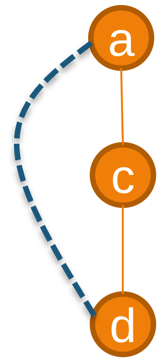
$P_1 – T_2$
$P_2 – T_1$
$P_3 – T_3$
$P_4 – T_4$

Força bruta e Busca exaustiva

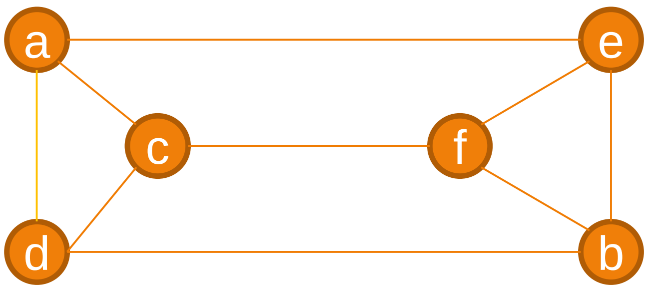
- Introdução
- Busca sequencial e String matching
- Os problemas Closest-pair e Convex-hull
- Closest-pair e Convex-hull por força bruta
- Busca exaustiva
  - Traveling Salesman Problem (TSP)
  - Knapsack Problem
  - Casamento
  - Busca em largura e em profundidade

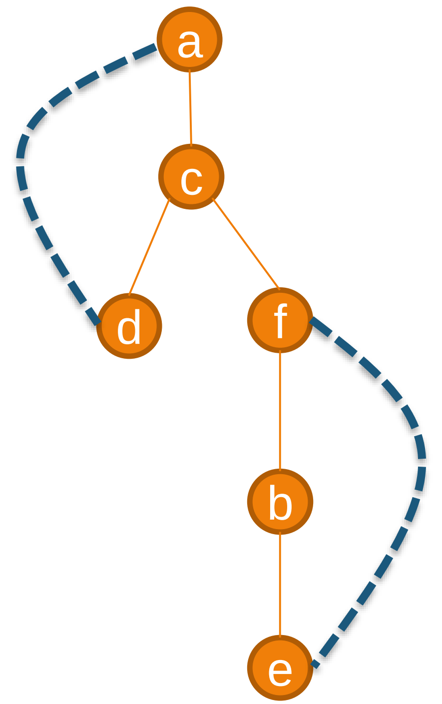
Busca em profundidade



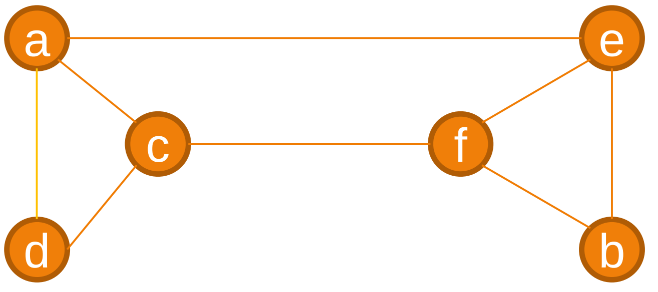


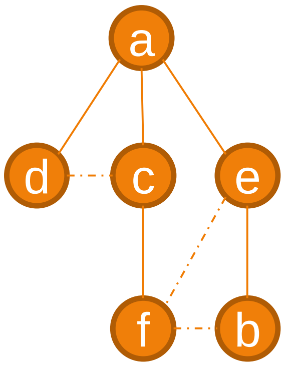
Busca em profundidade





Busca em profundidade





Tarefa

- Estudar busca em largura
- Estudar busca em profundidade
- Estudar a eficiência dos dois algoritmos

Resumo

- Força bruta é uma estratégia direta de resolver problemas, normalmente baseada nas definições dos conceitos envolvidos;
- As principais características de tais algoritmos é a simplicidade e a vasta gama de aplicações;
- Normalmente algoritmos de força bruta podem ser melhorados com modestos esforços;
- Exemplos de tais algoritmos são: multiplicação de matrizes, selection sort, busca sequencial...

Resumo

- Busca exaustiva é a aplicação de força bruta em problemas combinatoriais;
- TSP, KP e Assignment Problem são exemplos típicos de busca exaustiva;
- Busca exaustiva não é uma solução viável para problemas médios ou grandes;
- DFS e BSF são exemplos de travessia de grafos.

Exercícios

- Uma empresa tem 2 aparelhos eletrônicos idênticos para fazer um teste: até qual altura (em andares) o aparelho pode cair sem quebrar? Imagine que a empresa tenha um prédio de n andares para fazer o teste e considere que uma vez que o aparelho se quebrou, o teste só poderá ser completado com o segundo aparelho. Faça um algoritmo para resolver este problema.

Solução 1

i ← 1

enquanto i ≤ n E aparelho ok faça

  jogue o aparelho do andar i

se aparelho ok então

  retorne n

senão

  retorn i-1

fim-se

Complexidade O(n)

Dá para fazer um algoritmo mais eficiente?

Solução 1

i ← 1

enquanto i ≤ n E aparelho ok faça

jogue o aparelho do andar i

se aparelho ok então

retorne n

senão

retorne i-1

fim-se

Complexidade O(n)

Dá para fazer um algoritmo mais eficiente?

R.: SIM!

Solução 1

i ← 1

enquanto aparelho-1 ok e i < $\sqrt{n}$ faça

  jogue o aparelho-1 do andar $i.\sqrt{n}$

  i ← i + 1

se aparelho-1 ok então

  retorne n

senão

  ajuste fino com o aparelho-2

Solução 1

senão

j ← i – 1

k ← 1

enquanto aparelho-2 ok faça

$j.\sqrt{n} + k$

jogue o aparelho-2 do andar $j.\sqrt{n} + k - 1$

k ← k + 1

retorne

fim-se

Complexidade?

Solução 1

senão

j ← i – 1

k ← 1

enquanto aparelho-2 ok faça

$j.\sqrt{n}+k$

jogue o aparelho-2 do andar $j.\sqrt{n}+k-1$

k ← k + 1

retorne

fim-se

Complexidade:

$$\in O(\sqrt{n}+\sqrt{n}) = O(2\sqrt{n}) = O(\sqrt{n})$$

THE END