Projeto e Análise de Algoritmos

Hamilton José Brumatto

Bacharelado em Ciência da Computação - UESC

29 de abril de 2016

Projeto de Algoritmos

Projeto de Algoritmos

Busca pela corretude e eficiência.

- Força Bruta: Testa todas as opções até obter o resultado.
- *Backtracking*, um refinamento da força bruta. Nem todas as opções são testadas.
- Indução: A prova por indução representa um algoritmo.
- Divisão e Conquista: Problemas maiores são divididos em problemas menores e a solução combinada no final.
- Otimização: Busca pelo melhor resultado. Temos dois estilos de algoritmos: Programação Dinâmica e Algoritmos Gulosos.

Projeto de Algoritmos pela Força Bruta

- Nem sempre existe uma melhor solução para resolver um problema.
- A única saída é tentar todas as opções.
- A força bruta normalmente apresenta o algoritmo menos eficiente.

Somente é possível dizer que não "dá para fazer melhor" se puder ser provado que a Cota Inferior do Problema apresenta a dificuldade representada pela eficiência do algoritmo.

Resolvendo um problema

Busca do par

Dado um conjunto de números inteiros, buscar ao menos dois números, se houver, que a soma represente um valor dado.

Entrada: Um conjunto de números inteiros, um valor inteiro.

Saída: Se existirem, dois inteiros pertencentes ao conjunto fornecido cuja soma resulte no valor fornecido.

Como resolver este problema?

Força Bruta

- À primeira vista, para encontrar a solução, precisamos para cada número, testar a soma com todos os demais.

Para cada um dos n números, testaremos com $n-1$ outros, ou seja: $T(n) = n(n-1)$ ou $T(n) = n^2 - n$, $T(n) \in O(n^2)$.

Entrada: Uma sequência de números inteiros e um número inteiro
Saída: Se existir, dois números da sequência, cuja soma é igual ao valor numérico

Algoritmo BUSCA_PAR(A,x)
    para $j \leftarrow 1$ **até** $comprimento[A]$ **faça**
        para $i \leftarrow 1$ **até** $comprimento[A]$ **faça**
            se $i \neq j$ **então**
                se $A[i] + A[j] = x$ **então**
                    Retorna $(A[i], A[j])$

Força Bruta

- À primeira vista, para encontrar a solução, precisamos para cada número, testar a soma com todos os demais.

Para cada um dos n números, testaremos com $n-1$ outros, ou seja: $T(n) = n(n-1)$ ou $T(n) = n^2 - n$, $T(n) \in O(n^2)$.

Entrada: Uma sequência de números inteiros e um número inteiro
Saída: Se existir, dois números da sequência, cuja soma é igual ao valor numérico

1: **Algoritmo** BUSCA_PAR(A,x)
2:     **para** $j \leftarrow 1$ **até** $comprimento[A]$ **faça**
3:         **para** $i \leftarrow 1$ **até** $comprimento[A]$ **faça**
4:             **se** $i \neq j$ **então**
5:                 **se** $A[i] + A[j] = x$ **então**
6:                     **Retorna** $(A[i], A[j])$

Dá para fazer melhor?

Refinando a *Força-Bruta*

O que pode ser melhorado:

- A busca pode ser interrompida assim que encontrar um par.
- Se um número já foi testado com outro, ele não precisa repetir este teste.

Entrada: Uma sequência de números inteiros e um número inteiro
Saída: Se existir, dois números da sequência, cuja soma é igual ao valor numérico

Algoritmo BUSCA_PAR(A,x)
  $i \leftarrow 1$, $j \leftarrow i+1$
  enquanto $A[i] + A[j] \neq x$ **e** $i < comprimento[A]$ **faça**
    $j \leftarrow i+1$
    enquanto $A[i] + A[j] \neq x$ **e** $j \leqslant comprimento[A]$ **faça**
      $j \leftarrow j+1$
    se $A[i] + A[j] \neq x$ **então**
      $i \leftarrow i+1$
  se $A[i] + A[j] = x$ **então**
    Retorna $(A[i], A[j])$

Melhorou a eficiência deste algoritmo?

Refinando a *Força-Bruta*

O que pode ser melhorado:

- A busca pode ser interrompida assim que encontrar um par.
- Se um número já foi testado com outro, ele não precisa repetir este teste.

Entrada: Uma sequência de números inteiros e um número inteiro
Saída: Se existir, dois números da sequência, cuja soma é igual ao valor numérico

```
Algoritmo BUSCA_PAR(A,x)
    i ← 1, j ← i + 1
    enquanto A[i] + A[j] ≠ x e i < comprimento[A] faça
        j ← i + 1
        enquanto A[i] + A[j] ≠ x e j ⩽ comprimento[A] faça
            j ← j + 1
        se A[i] + A[j] ≠ x então
            i ← i + 1
    se A[i] + A[j] = x então
        Retorna (A[i], A[j])
```

Melhorou a eficiência deste algoritmo? $T(n) \in O(n^2)$

Evitando a força bruta

- Precisamos organizar a entrada de forma que não seja necessário olhar todas as situações.

Para organizar as informações, devemos utilizar recursos computacionais como Estrutura de Dados:

- Vetores, Matrizes, Conjuntos, Grafos, Árvores, ...

Além disto, os dados podem estar alinhados em algum critério:

- Ordem, Tamanho, Disposição espacial, ...

Evitando a força bruta

- Precisamos organizar a entrada de forma que não seja necessário olhar todas as situações.

Para organizar as informações, devemos utilizar recursos computacionais como Estrutura de Dados:

- Vetores, Matrizes, Conjuntos, Grafos, Árvores, ...

Além disto, os dados podem estar alinhados em algum critério:

- Ordem, Tamanho, Disposição espacial, ...

O que poderia melhorar no algoritmo que fizéssemos alguma organização nos dados?

Revendo o algoritmo *BUSCA_PAR*

Vamos supor que o conjunto fornecido na entrada contém números inteiros em ordem crescente.

Revendo o algoritmo *BUSCA_PAR*

Vamos supor que o conjunto fornecido na entrada contém números inteiros em ordem crescente.

Entrada: Uma sequência ordenada de números inteiros e um número inteiro
Saída: Se existir, dois números da sequência, cuja soma é igual ao valor numérico

Algoritmo BUSCA_PAR(A,x)
  $i \leftarrow 1; j \leftarrow comprimento[A]$
  enquanto $i \neq j$ **e** $A[i] + A[j] \neq x$ **faça**
    se $A[i] + A[j] > x$ **então**
      $j \leftarrow j - 1$
    senão
      $i \leftarrow i + 1$
  se $A[i] + A[j] = x$ **então**
    Retorna $(A[i], A[j])$

Qual a assintocidade deste algoritmo?

Revendo o algoritmo *BUSCA_PAR*

Vamos supor que o conjunto fornecido na entrada contém números inteiros em ordem crescente.

Entrada: Uma sequência ordenada de números inteiros e um número inteiro
Saída: Se existir, dois números da sequência, cuja soma é igual ao valor numérico

Algoritmo BUSCA_PAR(A,x)
  $i \leftarrow 1; j \leftarrow comprimento[A]$
  enquanto $i \neq j$ **e** $A[i] + A[j] \neq x$ **faça**
    se $A[i] + A[j] > x$ **então**
      $j \leftarrow j - 1$
    senão
      $i \leftarrow i + 1$
  se $A[i] + A[j] = x$ **então**
    Retorna $(A[i], A[j])$

Qual a assintocidade deste algoritmo? $T(n) \in O(n)$

Revendo o algoritmo *BUSCA_PAR*

Aplicando a nova solução no problema original:

Entrada: Uma sequência de números inteiros e um número inteiro
Saída: Se existir, dois números da sequência, cuja soma é igual ao valor numérico

Algoritmo $\mathrm{BUSCA\_PAR}$(A,x)
  ORDENAR(A)
  $i \leftarrow 1; j \leftarrow comprimento[A]$
  enquanto $i \neq j$ **e** $A[i] + A[j] \neq x$ **faça**
    se $A[i] + A[j] > x$ **então**
      $j \leftarrow j - 1$
    senão
      $i \leftarrow i + 1$
  se $A[i] + A[j] = x$ **então**
    Retorna $(A[i], A[j])$

E agora, qual a assintocidade?

Algoritmo de Ordenação

Os algoritmos de ordenação representam um capítulo importante no estudo de projeto e análise de algoritmos. No entanto, não vamos abordar os vários algoritmos e questões mais profundas como a análise da COTA INFERIOR

Eventualmente usaremos algum algoritmo de ordenação como exemplo, e este poderá ser melhor detalhado no momento.

Já estudamos um algoritmo de inserção (INSERT_SORT), de seleção (SELECT_SORT), um baseado em trocas (BUBBLE_SORT), vamos agora ver outro algoritmo baseado na troca: MERGE_SORT

O algoritmo MERGE_SORT

- Dividimos a seqüência em duas partes (Divisão e Conquista),
- Recursivamente aplicamos o algoritmo em cada parte, e cada parte estará ordenada (Indução).
- Intercalam-se os valores resultantes formando uma seqüência final ordenada.

Calculando o tempo do algoritmo: $T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + n$
$T(n) \in O(n \log n)$

Qual será então o tempo do algoritmo BUSCA_PAR ?

Algoritmo MERGE_SORT

Entrada: Uma sequência de números
Saída: A mesma sequência de números, ordenada

Algoritmo MERGE_SORT(A,p,r)
  se $p < r$ **então**
    $q \leftarrow (p + r)/2$
    MERGE_SORT(A, p, q)
    MERGE_SORT($A, q + 1, r$)
    INTERCALA(A, p, q, r)

Algoritmo MERGE_SORT: Rotina INTERCALA

Entrada: Uma sequência onde no intervalo [p,r] a sequência está ordenada no subarranjo [p,q] e no subarranjo [q,r].
Saída: A mesma sequência, ordenada no intervalo [p,r].

Algoritmo INTERCALA(A,p,q,r)
  para $i \leftarrow p$ **até** q **faça**
    $B[i] \leftarrow A[i]$
  para $j \leftarrow q+1$ **até** r **faça**
    $B[r+q+1-j] \leftarrow A[j]$
  $i \leftarrow p$
  $j \leftarrow r$
  para $k \leftarrow p$ **até** r **faça**
    se $B[i] \leqslant B[j]$ **então**
      $A[k] \leftarrow B[i]$
      $i \leftarrow i+1$
    senão
      $A[k] \leftarrow B[j]$
      $j \leftarrow j-1$

Atividades para projeto de algoritmos

- Resolver a 5ª Lista de exercícios.