CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho, Bruno Müller Jr, Carmem Hara

Departamento de Informática/UFPR

25 de agosto de 2020

Resumo

Aplicações de vetores: permutações (parte 1)

Objetivos da aula

- Resolver problemas que envolvem o uso de vetores
- Discutir eficiência dos algoritmos

Motivação

- O aprendizado de vetores é fortalecido quando procuramos resolver problemas variados que podem ser implementados com seu uso
- O entendimento da diferença entre o índice do vetor e o conteúdo ao qual este índice aponta é importante
- Vamos apresentar um problema matemático conhecido como *permutação* e tentar resolvê-lo usando vetores
- Vamos também discutir a eficiência das diferentes soluções

Definição

- Uma permutação é uma função bijetora de um conjunto nele mesmo
- Intuitivamente, é uma maneira de reordenar os elementos do conjunto (que não tem elementos repetidos)

Exemplo

Seja o conjunto $\{1, 2, 3, 4, 5\}$:

Uma permutação pode ser esta:

$$P(1) = 4,$$
$$P(2) = 1,$$
$$P(3) = 5,$$
$$P(4) = 2,$$
$$P(5) = 3.$$

Esquematicamente:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 1 & 5 & 2 & 3 \end{pmatrix}$$

Outra pode ser esta:

$$P(1) = 2,$$
$$P(2) = 5,$$
$$P(3) = 1,$$
$$P(4) = 3,$$
$$P(5) = 2.$$

Esquematicamente:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 1 & 3 & 2 \end{pmatrix}$$

Número de diferentes permutações

- Existem $n!$ maneiras de se reordenar os n elementos de um conjunto
- Se $n = 5$ então existem 120 permutações

Representação computacional

- Pode-se usar um vetor para modelar uma permutação
- Seja V um vetor de n posições inteiras
- Cada posição é um valor (sem repetição) entre 1 e n.
- Por exemplo:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 1 & 5 & 2 & 3 \end{pmatrix}$$

1	2	3	4	5
4	1	5	2	3

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 1 & 3 & 2 \end{pmatrix}$$

1	2	3	4	5
2	5	1	3	2

Problemas com permutações

- Testar se uma representação corresponde a uma permutação
- Gerar aleatoriamente uma representação correspondente a uma permutação
- Determinar a ordem de uma permutação

Testar se uma representação é uma permutação

- Dado um vetor qualquer, ele pode ser a representação de alguma permutação do conjunto de números entre 1 e n ?
- Computacionalmente, deve-se testar de todos os números entre 1 e n estão presentes no vetor
- Isto é, não pode haver números repetidos nem faltar ninguém

Algoritmo 1

- Para todos os números entre 1 e n
- Testa se este número está presente no vetor
- Se todos estiverem, dado que são n números em um vetor de n elementos, é porque não houve repetição
- Logo, a permutação é válida

Implementação do algoritmo 1

```
1  function testa_permutacao (var v: vetor; n: integer): boolean;
2  var i, j: integer;
3      eh_permutacao: boolean;
4  begin
5      eh_permutacao:= true;
6      i:= 1;
7      while eh_permutacao and (i <= n) do
8      begin
9          j:= 1;                (* procura se i esta no vetor *)
10         while (j <= n) and (v[j] <> i) do
11             j:= j + 1;
12         if v[j] <> i then (* se nao achou nao eh permutacao *)
13             eh_permutacao:= false;
14         i:= i +1;
15     end;
16     testa_permutacao:= eh_permutacao;
17 end; (* testa_permutacao *)
```

Análise do algoritmo 1

- Melhor caso: ele procura o elemento 1 e não acha no término do laço interno. Executou n comparações
- Pior caso: ocorre quando o vetor representa uma permutação, neste caso o algoritmo executou n^2 comparações
- Pode ser mais eficiente?

Algoritmo 2

- Usa um vetor auxiliar, inicialmente zerado
- Percorre-se o vetor de entrada e para cada índice:
  - tenta inserir este elemento no vetor auxiliar
  - a inserção é válida se naquela posição do vetor auxiliar houver um zero
  - caso exista um valor diferente de zero, é porque existe um elemento repetido e a permutação não é válida
- Se conseguiu inserir todos os elementos, é porque a permutação é válida

Implementação do algoritmo 2

```
1  function testa_permutacao_v2 (var v: vetor; n: integer): boolean;
2  var i: integer;
3      aux: vetor;
4      eh_permutacao: boolean;
5  begin
6      zerar_vetor (aux,n);
7      eh_permutacao:= true;
8      i:= 1;
9      while eh_permutacao and (i <= n) do
10     begin
11         if (v[i] >= 1) and (v[i] <= n) and (aux[v[i]] = 0) then
12             aux[v[i]]:= v[i]
13         else
14             eh_permutacao:= false;
15         i:= i + 1;
16     end;
17     testa_permutacao_v2:= eh_permutacao;
18 end; (* testa_permutacao_v2 *)
```

Análise do algoritmo 2

- Melhor caso: executou n atribuições para zerar o vetor auxiliar. Em seguida, inseriu um elemento com sucesso. Logo depois, não conseguiu inserir o segundo. Custou portanto $n + 2$ operações
- Pior caso: ocorre quando o vetor representa uma permutação, neste caso o algoritmo executou n atribuições e n comparações, em um custo total de $n + n = 2n$ operações
- Isto é, o melhor caso é similar, mas o pior caso é bem mais eficiente

Fim do tópico

- este material está no livro no capítulo 9, seção 9.7.1

Licença

- Slides feitos em LaTeX usando beamer
- Licença

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`