

CI1055: Algoritmos e Estruturas de Dados I

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

Departamento de Informática/UFPR

30 de novembro de 2021

Resumo

Aplicações de vetores: permutações (parte 2)

Objetivos da aula

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

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

Gerando permutações válidas

- O próximo problema é gerar randomicamente um vetor que represente uma permutação válida
- Faremos uso da função `random` da linguagem *Pascal*

Algoritmo 1

- Gera um vetor aleatoriamente e testa usando a função anterior
- Repete este processo até gerar uma permutação válida

Implementação do algoritmo 1

```
1 procedure gerar_permutacao (var v: vetor; n: integer);  
2 var i: integer;  
3 begin  
4     randomize;  
5     repeat  
6         for i:= 1 to n do  
7             v[i]:= random (n) + 1; (* sorteia num entre 1 e n *)  
8         until testa_permutacao_v2 (v, n);  
9 end; (* gera_permutacao *)
```

Análise do algoritmo 1

- Algoritmos com componentes randômicos não são triviais de serem analisados analiticamente
 - Análise de Algoritmos!
- Por isso faremos uma análise empírica

Análise do algoritmo 1

- Muito lento quando n cresce
- É muito pouco provável que os vetores sejam gerados sem repetição
- Experimentalmente, para valores acima de 13 ou 14 o tempo de relógio é inaceitável

- Para cada índice do vetor, sortear o conteúdo
- Em seguida, verifica se algum elemento anteriormente atribuído é igual
- Caso seja, sorteia-se outro, até que a verificação retorne OK
- Isto garante que o vetor final produzido é válido

Implementação do algoritmo 2

```
1 procedure gerar_permutacao_v2 (var v: vetor; n: integer);
2 var i, j: integer;
3 begin
4     randomize;
5     v[1]:= random (n) + 1;
6     for i:= 2 to n do
7         repeat
8             v[i]:= random (n) + 1; (* gera um numero entre 1 e n
9                                     *)
10            j:= 1; (* procura se o elemento ja existe no vetor *)
11            while (j < i) and (v[i]  $\diamond$  v[j]) do
12                j:= j + 1;
13            until j = i; (* descobre que o elemento eh novo *)
end; (* gera_permutacao_v2 *)
```

- Melhor que o anterior
- Executa na casa de 2 segundos para vetores de tamanho até 1000
- Entradas de tamanho 30.000 demora cerca de 20 segundos
- Vetores maiores também demoram muito

- Inicializa-se um vetor de forma ordenada
- Depois faz-se alterações aleatórias de seus elementos um número também aleatório de vezes

Implementação do algoritmo 3

```
1 procedure gerar_permutacao_v3 (var v: vetor; n: integer);  
2 var i, j, k, aux, max: integer;  
3 begin  
4     for i:= 1 to n do  
5         v[i] := i;  
6  
7     randomize;  
8     max:= random (1000); (* troca dois elementos max vezes *)  
9     for i:= 1 to max do  
10        begin  
11            j:= random (n) + 1;  
12            k:= random (n) + 1;  
13            aux:= v[j];  
14            v[j]:= v[k];  
15            v[k]:= aux;  
16        end;  
17 end; (* gera_permutacao_v3 *)
```

Análise do algoritmo 3

- Melhor que o anterior
- Produz corretamente vetores que representam permutações
- É muito rápido e mistura bem os números
- Foi testado com entradas da ordem de um milhão de elementos

Algoritmo 4

- Inicializar um vetor auxiliar com números sequenciais de 1 até n
- Em seguida se realizam n sorteios de índices do vetor auxiliar
- Para um índice j sorteado, copia-se $aux[j]$ no vetor permutação
- Em seguida diminui-se o tamanho do auxiliar, copiando o último sobre $v[j]$

Implementação do algoritmo 4

```
1 procedure gerar_permutacao_v4 (var v: vetor_i; n: longint);
2 var i, j, tam: longint;
3     aux: vetor_i;
4 begin
5     for i := 1 to n do
6         aux[i] := i;
7
8     randomize;
9     tam:= n;
10    for i := 1 to n do
11        begin
12            j := random(tam) + 1;
13            v[i] := aux[j];
14            aux[j] := aux[tam];
15            tam := tam - 1;
16        end;
17 end; (* gera_permutacao_v4 *)
```


Análise do algoritmo 4

- Executa em tempo linear!
- Produz corretamente vetores que representam permutações
- É muito rápido e mistura bem os números
- Foi testado com entradas da ordem de um milhão de elementos

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

- 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/>