Estrutura de Dados

Ordenação em tempo linear

Prof. Dr. Daniel Vecchiato

Agenda

- Introdução
- Bucket sort
- Ordenação por contagem
- Exercícios

Introdução

- Vários métodos de ordenação já foram vistos
  - Todos usando ordenação por comparação
  - Melhor caso: O (n log n)
- É possível existir algoritmos melhores, assumindo que:
  - A entrada possui determinadas características
  - Respeitando algumas restrições
  - Ou seja, para aplicar em casos bem específicos

Introdução

- Nesses casos, pode ser alcançado um desempenho linear
  - O(n)
- Exemplos deste tipo de algoritmo
  - Ordenação por contagem (Couting sort)
  - Bucket sort

Bucket sort

- Pressupõe que:
  - As chaves estão uniformemente distribuídas em um intervalo conhecido
- Casos
  - Bons
    - 1 2 3 4 5
    - -20 -10 0 10 20
  - Ruins
    - 1 1 1 1 1 1 99
    - -900 -800 -700 -9 -8 -7 0 0 0 1 1 1
- Pode-se adotar o intervalo de 0 a 1

Bucket sort

- Ideia
  - Dividir o intervalo total (0 a 1, por exemplo) em **n** subintervalos de mesmo tamanho
    - Chamado aqui de “baldes”
  - Separar cada valor do conjunto de dados nos seus respectivos baldes
  - Caso a entrada seja realmente uniformemente distribuída, não haverá muitos números em cada balde
  - Deve ser ordenado os poucos números em cada balde
  - E, por fim, gerar a saída lendo os números ordenados em cada balde, sequencialmente do 1º balde ao último

Bucket sort

- Algoritmo

```
#define TAM_BUCKET 100
#define NUM_BUCKET 10

typedef struct {
	int chave;
} TItem;

typedef struct {
	int quantidade;
	TItem balde[TAM_BUCKET];
} Bucket;
```

  - As constantes dirão quantos baldes serão criados e quais as suas capacidades
  - Cada balde (Bucket) contém um vetor de TItem e a quantidade de elementos que esse balde possui no momento

Bucket sort

- Algoritmo

```
void bucketSort (TItem *v, int n)
{
      Bucket b[NUM_BUCKET];
      int i, j, k;

      for (i=0; i < NUM_BUCKET; i++)
            b[i].quantidade = 0;
      ...
```

  - O método inicia zerando todos os baldes criados

Bucket sort

- Algoritmo

```
void bucketSort (TItem *v, int n){
        ...
        for( i=0; i < n; i++){
                j = NUM_BUCKET - 1;
                while (1){
                        if (j < 0)
                                break;
                        if (v[i].chave >= j*10) {
                                b[j].balde[b[j].quantidade] = v[i];
                                (b[j].quantidade)++;
                                break;
                        }
                        j--;
                }
        }
        ...
```

- Verificando em qual balde cada elemento será posicionado

Bucket sort

- Algoritmo

```
void bucketSort (TItem *v, int n) {
        ...
        for (i=0; i < NUM_BUCKET; i++)
              if (b[i].quantidade > 0)
                    insertionSort (b[i].balde,
b[i].quantidade);
        ...
```

  - Ordenando com InsertionSort os valores de cada balde

Bucket sort

- Algoritmo

```
void bucketSort (TItem *v, int n) {
        ...
        i = 0;
        for (j=0; j < NUM_BUCKET; j++) {
                for (k=0; k < b[j].quantidade; k++) {
                        v[i] = b[j].balde[k];
                        i++;
                }
        }
 }
```

  - Recuperando os elementos de cada balde no vetor original
  - Obtendo o resultado final

Ordenação por contagem

- Pressupõe que:
  - Cada um dos **n** elementos de entrada é um inteiro no intervalo de 0 a **k-1**
- Ideia
  - Contar, para cada elemento **x** do conjunto de dados, quantos valores são menores que **x**
  - Essa informação servirá para inserir o elemento x diretamente em sua posição no conjunto ordenado
- Exemplo
  - Se é possível saber que 5 elementos são menores que **x**, então **x** deve ser posicionado na 6ª posição do vetor
    - Um cuidado especial deve ser tomado para números repetidos, para que eles não sejam inseridos sempre na mesma posição

Ordenação por contagem

- Algoritmo

```
void contagem (TItem *A, TItem *B, int n, int k) {
      int i, C[k];
```

  - A[0, ..., n-1]: Vetor de entrada
  - B[0, ..., n-1]: Vetor para armazenar a saída ordenada
  - C[0, ..., k]: Vetor utilizado para armazenamento temporário
  - n: Tamanho do vetor
  - 0 ... k-1: Faixa de valores existente no vetor A

  - Ex:
  - Vetor a ser ordenado:
  - N = 5
  - K = 9

Ordenação por contagem

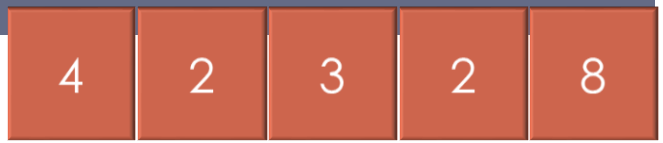
- Algoritmo

```
void contagem (TItem *A, TItem *B, int n, int k) {
        int i, C[k];
        for (i = 0; i < k; i++)
                C[i] = 0;
```

  - Iniciando o vetor C (vetor de contagem) com zero
  - Vetor A:

  - Vetor C:

Ordenação por contagem



- Algoritmo

```
void contagem (TItem *A, TItem *B, int n, int k) {
        int i, C[k];
        for (i = 0; i < k; i++)
                C[i] = 0;

        for (i = 0; i < n; i++)
                C[A[i].chave] = C[A[i].chave] + 1;
```

- Cada índice do vetor C irá armazenar quantos elementos existem no conjunto de entradas com o respectivo valor
- Se depois deste loop, houver o valor 2 na posição 3, quer dizer que há dois elementos com o valor 3 no vetor A (conjunto de entrada)

Ordenação por contagem

- Algoritmo

```
void contagem (TItem *A, TItem *B, int n, int k) {
        ...

        for (i = 1; i < k; i++)
                C[i] = C[i] + C[i-1];
```

- Agora em cada posição de C[i] contém o número de elementos menores ou iguais a i
- Se houver o valor 4 na posição 5, quer dizer que há quatro elementos menores ou iguais a 5

- Antes:

- Depois:

Ordenação por contagem

- Algoritmo

```
void contagem (TItem *A, TItem *B, int n, int k) {
        ...
        for (i = n-1; i >= 0; i--) {
                B[C[A[i].chave]-1] = A[i];
                C[A[i].chave] = C[A[i].chave] - 1;
        }
}
```

- O último loop posiciona cada elemento no seu lugar ordenado
- A cada iteração, o vetor C decrementa a chave utilizada
- Vetor A:

4	2	3	2	8

- Vetor C:

0	0	0	2	3	4	4	4	4

- Vetor B:

2	2	3	4	8

Ordenação por contagem

- Algoritmo

```
void contagem (TItem *A, TItem *B, int n, int k) {
      int i, C[k];
      for (i = 0; i < k; i++)
            C[i] = 0;

      for (i = 0; i < n; i++)
            C[A[i].chave] = C[A[i].chave] + 1;

      for (i = 1; i < k; i++)
            C[i] = C[i] + C[i-1];

      for (i = n-1; i >= 0; i--) {
            B[C[A[i].chave]-1] = A[i];
            C[A[i].chave] = C[A[i].chave] - 1;
      }
}
```

Exercícios

- Implemente o método de ordenação por contagem
- Implemente o método bucket sort

Estrutura de Dados

Material elaborado por:
Thiago Meirelles Ventura

Baseado em:

- Ascencio, A. F. G; Araújo, G. S. Estruturas de Dados. Pearson, 2011.
- Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. Algoritmos: teoria e prática. Elsevier, 2002.
- Aulas do Prof. Reinaldo Silva Fortes (http://www.decom.ufop.br/reinaldo/)