Estrutura de Dados

Merge sort

Prof. Dr. Daniel Vecchiato

Agenda

- Introdução
- Merge sort
  - Funcionamento
  - Comportamento
  - Implementação recursiva
  - Implementação iterativa
- Comparação
- Exercícios

Introdução

- Os algoritmos básicos de ordenação possuem custo muito alto quando a entrada é grande
- Alguns métodos podem ser eficazes, mesmo com grandes conjuntos de dados
- Uma abordagem para trabalhar com grandes conjuntos de dados é a divisão e conquista
  - Divisão: a entrada é quebrada em partes menores
  - Conquista: cada pedaço é processado
  - E depois os resultados parciais são combinados para gerar o resultado final

Introdução

- Normalmente, algoritmos de divisão e conquista são recursivos
- Vantagens:
  - Uso eficiente de memória cache
  - Possibilita o paralelismo
    - entre processadores ou entre máquinas

Introdução

- Algoritmo básico de divisão e conquista

```
divisaoConquista (x)
    se x é pequeno
        return resolve (x)
    se não
        decompor x em conjuntos pequenos
        para cada conjunto gerado
            y(i) = divisaoConquista (subconjuntos(i))
        return combinar (y)
```

Merge sort

- Também chamado de ordenação por intercalação
- Aproveita do conceito de divisão e conquista
- Sempre divide a entrada de forma balanceada
  - Tenta ter subconjuntos de mesmo tamanho

Merge sort

- Funcionamento
  - Dividir o conjunto de dados em duas partes
  - Para cada parte, divida novamente em duas partes
    - Faça isso recursivamente
  - Depois, intercale as duas partes de modo ordenado

Merge sort

- Funcionamento

https://en.wikipedia.org/wiki/File:Merge-sort-example-300px.gif

Merge sort

- Comportamento
  - https://www.toptal.com/developers/sorting-algorithms/merge-sort
  - Custo
    - O(n log n)
  - Estável
  - Desvantagem:
    - Utiliza memória auxiliar O(n)

Merge sort

- Implementação recursiva

```
void mergeSort (TItem *v, int n) {
   mergeSortOrdena (v, 0, n-1);
}

void mergeSortOrdena (TItem *v, int esq, int dir) {
   if (esq < dir) {
      int meio = (esq + dir) / 2;
      mergeSortOrdena (v, esq, meio);
      mergeSortOrdena (v, meio+1, dir);
      mergeSortIntercala (v, esq, meio, dir);
   }
}
```

Merge sort

```
void mergeSortIntercala (TItem *v, int esq, int meio, int dir) {
   int i, j, k;
   int a_tam = meio - esq + 1;
   int b_tam = dir - meio;
   TItem *a = (TItem *) malloc (sizeof (TItem) * a_tam);
   TItem *b = (TItem *) malloc (sizeof (TItem) * b_tam);

   for (i = 0; i < a_tam; i++)
      a[i] = v[i+esq];
   for (i = 0; i < b_tam; i++)
      b[i] = v[i+meio+1];

   for (i = 0, j = 0, k = esq; k <= dir; k++) {
      if (i == a_tam)
         v[k] = b[j++];
      else if (j == b_tam)
         v[k] = a[i++];
      else if (a[i].chave < b[j].chave)
         v[k] = a[i++];
      else
         v[k] = b[j++];
   }

   free (a);
   free (b);
}
```

Merge sort

- Implementação iterativa

```
void mergeSortIterativo (TItem *v, int n) {
   int esq, dir;
   int b = 1;
   while (b < n) {
      esq = 0;
      while (esq + b < n) {
         dir = esq + 2 * b;
         if (dir > n)
            dir = n;
         mergeSortIntercala (v, esq, esq+b-1, dir-1);
         esq = esq + 2 * b;
      }
      b *= 2;
   }
}
```

Comparação

Algoritmo	Comparações			Movimentações		
	Melhor	**Médio**	**Pior**	**Melhor**	**Médio**	**Pior**
Bubble	$O(n^2)$			$O(n^2)$		
Selection	$O(n^2)$			$O(n)$		
Insertion	$O(n)$	$O(n^2)$		$O(n)$	$O(n^2)$	
Merge	$O(n \log n)$			-		

Exercícios

- Implemente o método de ordenação Merge sort
- Imprima em cada intercalação
  - Qual índice inicial (esq) e final (dir) estão sendo processados
  - Como ficou o vetor depois de cada intercalação

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/)