Ordenação por intercalação

Aula 6

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

Conteúdo da aula

Introdução e motivação

- Operação básica em Computação
- Será que existem métodos mais eficientes de ordenação?
- Ordenação com recursão
- Dividir para conquistar

Introdução e motivação

- Operação básica em Computação
- Será que existem métodos mais eficientes de ordenação?
- Ordenação com recursão
- Dividir para conquistar

Introdução e motivação

- Operação básica em Computação
- Será que existem métodos mais eficientes de ordenação?
- Ordenação com recursão
- Dividir para conquistar

Introdução e motivação

- Operação básica em Computação
- Será que existem métodos mais eficientes de ordenação?
- Ordenação com recursão
- Dividir para conquistar

Dividir para conquistar

Dividir o problema em um número de subproblemas;

Conquistar os subproblemas solucionando-os recursivamente. No entanto, se os tamanhos dos subproblemas são suficientemente pequenos, resolva os subproblemas de uma maneira simples;

Combinar as soluções dos subproblemas na solução do problema original.

Dividir para conquistar

Dividir o problema em um número de subproblemas;

Conquistar os subproblemas solucionando-os recursivamente. No entanto, se os tamanhos dos subproblemas são suficientemente pequenos, resolva os subproblemas de uma maneira simples;

Combinar as soluções dos subproblemas na solução do problema original.

Dividir para conquistar

- **Dividir** o problema em um número de subproblemas;
- **Conquistar** os subproblemas solucionando-os recursivamente. No entanto, se os tamanhos dos subproblemas são suficientemente pequenos, resolva os subproblemas de uma maneira simples;
- **Combinar** as soluções dos subproblemas na solução do problema original.

Problema da intercalação

Problema

Dados dois conjuntos crescentes A e B, com m e n elementos respectivamente, obter um conjunto crescente C a partir de A e B

Problema

Dados dois vetores crescentes $v[p..q-1]$ e $v[q..r-1]$, rearranjar $v[p..r-1]$ em ordem crescente

Problema da intercalação

Problema

Dados dois conjuntos crescentes A e B, com m e n elementos respectivamente, obter um conjunto crescente C a partir de A e B

Problema

Dados dois vetores crescentes $v[p..q-1]$ e $v[q..r-1]$, rearranjar $v[p..r-1]$ em ordem crescente

Problema da intercalação

```
/* Recebe os vetores crescentes v[p..q-1] e v[q..r-1]
   e rearranja v[p..r-1] em ordem crescente */
void intercala(int p, int q, int r, int v[MAX])
{
   int i, j, k, w[MAX];

   i = p; j = q; k = 0;
   while (i < q && j < r) {
      if (v[i] <= v[j]) {
         w[k] = v[i]; i++; }
      else {
         w[k] = v[j]; j++; }
      k++;
   }
   while (i < q) {
      w[k] = v[i]; i++; k++; }
   while (j < r) {
      w[k] = v[j]; j++; k++; }
   for (i = p; i < r; i++)
      v[i] = w[i-p];
}
```

Problema da intercalação

- Tempo de execução de pior caso proporcional ao número de comparações entre os elementos do vetor, isto é, $r - p$
- Isto é, o consumo de tempo no pior caso da função `intercala` é proporcional ao número de elementos do vetor de entrada

Problema da intercalação

- Tempo de execução de pior caso proporcional ao número de comparações entre os elementos do vetor, isto é, $r - p$
- Isto é, o consumo de tempo no pior caso da função `intercala` é proporcional ao número de elementos do vetor de entrada

Ordenação por intercalação

- Dividimos ao meio um vetor v com $r - p$ elementos
- Ordenamos recursivamente essas duas metades de v
- Intercalamos essas metades

Ordenação por intercalação

- Dividimos ao meio um vetor v com $r - p$ elementos
- Ordenamos recursivamente essas duas metades de v
- Intercalamos essas metades

Ordenação por intercalação

- Dividimos ao meio um vetor v com $r - p$ elementos
- Ordenamos recursivamente essas duas metades de v
- Intercalamos essas metades

Ordenação por intercalação

```
void mergesort(int p, int r, int v[MAX])
{
   int q;

   if (p < r - 1) {
      q = (p + r) / 2;
      mergesort(p, q, v);
      mergesort(q, r, v);
      intercala(p, q, r, v);
   }
}
```

Ordenação por intercalação

- Para ordenar um vetor $v[0..n-1]$ basta chamar a função `mergesort` com os seguintes argumentos:

```
mergesort(0, n, v);
```

- Um exemplo de execução da função `mergesort` para um vetor de entrada $v[0..7] = \{4, 6, 7, 3, 5, 1, 2, 8\}$ e chamada

```
mergesort(0, 8, v);
```

- Para ordenar um vetor $v[0..n-1]$ basta chamar a função `mergesort` com os seguintes argumentos:

```
mergesort(0, n, v);
```

- Um exemplo de execução da função `mergesort` para um vetor de entrada $v[0..7] = \{4, 6, 7, 3, 5, 1, 2, 8\}$ e chamada

```
mergesort(0, 8, v);
```

Ordenação por intercalação

- Desempenho da função `mergesort` quando queremos ordenar um vetor $v[0..n-1]$
  - Suponha que n é uma potência de 2
  - O número de elementos do vetor é diminuído a aproximadamente metade a cada chamada da função
  - Logo, o número aproximado de chamadas é proporcional a $\log_2 n$
  - Na primeira vez, o problema original é reduzido a dois subproblemas onde é necessário ordenar os vetores $v[0..\frac{n}{2}-1]$ e $v[\frac{n}{2}..n-1]$
  - Na segunda vez, cada um dos subproblemas são ainda divididos em mais dois subproblemas cada, gerando quatro subproblemas no total, onde é necessário ordenar os vetores $v[0..\frac{n}{4}-1]$, $v[\frac{n}{4}..\frac{n}{2}-1]$, $v[\frac{n}{2}..\frac{3n}{4}-1]$ e $v[\frac{3n}{4}..n-1]$
  - E assim por diante
  - Além disso, o tempo total que a função `intercala` gasta é proporcional ao número de elementos do vetor v, isto é, $r-p$
  - Portanto, a função `mergesort` consome tempo proporcional a $n \log_2 n$

Ordenação por intercalação

- Desempenho da função `mergesort` quando queremos ordenar um vetor $v[0..n-1]$
  - Suponha que n é uma potência de 2
  - O número de elementos do vetor é diminuído a aproximadamente metade a cada chamada da função
  - Logo, o número aproximado de chamadas é proporcional a $\log_2 n$
  - Na primeira vez, o problema original é reduzido a dois subproblemas onde é necessário ordenar os vetores $v[0..\frac{n}{2}-1]$ e $v[\frac{n}{2}..n-1]$
  - Na segunda vez, cada um dos subproblemas são ainda divididos em mais dois subproblemas cada, gerando quatro subproblemas no total, onde é necessário ordenar os vetores $v[0..\frac{n}{4}-1]$, $v[\frac{n}{4}..\frac{n}{2}-1]$, $v[\frac{n}{2}..\frac{3n}{4}-1]$ e $v[\frac{3n}{4}..n-1]$
  - E assim por diante
  - Além disso, o tempo total que a função `intercala` gasta é proporcional ao número de elementos do vetor v, isto é, $r-p$
  - Portanto, a função `mergesort` consome tempo proporcional a $n \log_2 n$

Ordenação por intercalação

- Desempenho da função `mergesort` quando queremos ordenar um vetor $v[0..n-1]$
  - Suponha que n é uma potência de 2
  - O número de elementos do vetor é diminuído a aproximadamente metade a cada chamada da função
  - Logo, o número aproximado de chamadas é proporcional a $\log_2 n$
  - Na primeira vez, o problema original é reduzido a dois subproblemas onde é necessário ordenar os vetores $v[0..\frac{n}{2}-1]$ e $v[\frac{n}{2}..n-1]$
  - Na segunda vez, cada um dos subproblemas são ainda divididos em mais dois subproblemas cada, gerando quatro subproblemas no total, onde é necessário ordenar os vetores $v[0..\frac{n}{4}-1]$, $v[\frac{n}{4}..\frac{n}{2}-1]$, $v[\frac{n}{2}..\frac{3n}{4}-1]$ e $v[\frac{3n}{4}..n-1]$
  - E assim por diante
  - Além disso, o tempo total que a função `intercala` gasta é proporcional ao número de elementos do vetor v, isto é, $r-p$
  - Portanto, a função `mergesort` consome tempo proporcional a $n \log_2 n$

Ordenação por intercalação

- Desempenho da função `mergesort` quando queremos ordenar um vetor $v[0..n-1]$
  - Suponha que n é uma potência de 2
  - O número de elementos do vetor é diminuído a aproximadamente metade a cada chamada da função
  - Logo, o número aproximado de chamadas é proporcional a $\log_2 n$
  - Na primeira vez, o problema original é reduzido a dois subproblemas onde é necessário ordenar os vetores $v[0..\frac{n}{2}-1]$ e $v[\frac{n}{2}..n-1]$
  - Na segunda vez, cada um dos subproblemas são ainda divididos em mais dois subproblemas cada, gerando quatro subproblemas no total, onde é necessário ordenar os vetores $v[0..\frac{n}{4}-1]$, $v[\frac{n}{4}..\frac{n}{2}-1]$, $v[\frac{n}{2}..\frac{3n}{4}-1]$ e $v[\frac{3n}{4}..n-1]$
  - E assim por diante
  - Além disso, o tempo total que a função `intercala` gasta é proporcional ao número de elementos do vetor v, isto é, $r-p$
  - Portanto, a função `mergesort` consome tempo proporcional a $n \log_2 n$

Ordenação por intercalação

- Desempenho da função `mergesort` quando queremos ordenar um vetor $v[0..n-1]$
  - Suponha que n é uma potência de 2
  - O número de elementos do vetor é diminuído a aproximadamente metade a cada chamada da função
  - Logo, o número aproximado de chamadas é proporcional a $\log_2 n$
  - Na primeira vez, o problema original é reduzido a dois subproblemas onde é necessário ordenar os vetores $v[0..\frac{n}{2}-1]$ e $v[\frac{n}{2}..n-1]$
  - Na segunda vez, cada um dos subproblemas são ainda divididos em mais dois subproblemas cada, gerando quatro subproblemas no total, onde é necessário ordenar os vetores $v[0..\frac{n}{4}-1]$, $v[\frac{n}{4}..\frac{n}{2}-1]$, $v[\frac{n}{2}..\frac{3n}{4}-1]$ e $v[\frac{3n}{4}..n-1]$
  - E assim por diante
  - Além disso, o tempo total que a função `intercala` gasta é proporcional ao número de elementos do vetor v, isto é, $r-p$
  - Portanto, a função `mergesort` consome tempo proporcional a $n \log_2 n$

Ordenação por intercalação

- Desempenho da função `mergesort` quando queremos ordenar um vetor $v[0..n-1]$
  - Suponha que n é uma potência de 2
  - O número de elementos do vetor é diminuído a aproximadamente metade a cada chamada da função
  - Logo, o número aproximado de chamadas é proporcional a $\log_2 n$
  - Na primeira vez, o problema original é reduzido a dois subproblemas onde é necessário ordenar os vetores $v[0..\frac{n}{2}-1]$ e $v[\frac{n}{2}..n-1]$
  - Na segunda vez, cada um dos subproblemas são ainda divididos em mais dois subproblemas cada, gerando quatro subproblemas no total, onde é necessário ordenar os vetores $v[0..\frac{n}{4}-1]$, $v[\frac{n}{4}..\frac{n}{2}-1]$, $v[\frac{n}{2}..\frac{3n}{4}-1]$ e $v[\frac{3n}{4}..n-1]$
  - E assim por diante
  - Além disso, o tempo total que a função `intercala` gasta é proporcional ao número de elementos do vetor v, isto é, $r-p$
  - Portanto, a função `mergesort` consome tempo proporcional a $n\log_2 n$

Ordenação por intercalação

- Desempenho da função `mergesort` quando queremos ordenar um vetor $v[0..n-1]$
  - Suponha que n é uma potência de 2
  - O número de elementos do vetor é diminuído a aproximadamente metade a cada chamada da função
  - Logo, o número aproximado de chamadas é proporcional a $\log_2 n$
  - Na primeira vez, o problema original é reduzido a dois subproblemas onde é necessário ordenar os vetores $v[0..\frac{n}{2}-1]$ e $v[\frac{n}{2}..n-1]$
  - Na segunda vez, cada um dos subproblemas são ainda divididos em mais dois subproblemas cada, gerando quatro subproblemas no total, onde é necessário ordenar os vetores $v[0..\frac{n}{4}-1]$, $v[\frac{n}{4}..\frac{n}{2}-1]$, $v[\frac{n}{2}..\frac{3n}{4}-1]$ e $v[\frac{3n}{4}..n-1]$
  - E assim por diante
  - Além disso, o tempo total que a função `intercala` gasta é proporcional ao número de elementos do vetor v, isto é, $r-p$
  - Portanto, a função `mergesort` consome tempo proporcional a $n\log_2 n$

Ordenação por intercalação

- Desempenho da função `mergesort` quando queremos ordenar um vetor $v[0..n-1]$
  - Suponha que n é uma potência de 2
  - O número de elementos do vetor é diminuído a aproximadamente metade a cada chamada da função
  - Logo, o número aproximado de chamadas é proporcional a $\log_2 n$
  - Na primeira vez, o problema original é reduzido a dois subproblemas onde é necessário ordenar os vetores $v[0..\frac{n}{2}-1]$ e $v[\frac{n}{2}..n-1]$
  - Na segunda vez, cada um dos subproblemas são ainda divididos em mais dois subproblemas cada, gerando quatro subproblemas no total, onde é necessário ordenar os vetores $v[0..\frac{n}{4}-1]$, $v[\frac{n}{4}..\frac{n}{2}-1]$, $v[\frac{n}{2}..\frac{3n}{4}-1]$ e $v[\frac{3n}{4}..n-1]$
  - E assim por diante
  - Além disso, o tempo total que a função `intercala` gasta é proporcional ao número de elementos do vetor v, isto é, $r-p$
  - Portanto, a função `mergesort` consome tempo proporcional a $n \log_2 n$

Ordenação por intercalação

- Desempenho da função `mergesort` quando queremos ordenar um vetor $v[0..n-1]$
  - Suponha que n é uma potência de 2
  - O número de elementos do vetor é diminuído a aproximadamente metade a cada chamada da função
  - Logo, o número aproximado de chamadas é proporcional a $\log_2 n$
  - Na primeira vez, o problema original é reduzido a dois subproblemas onde é necessário ordenar os vetores $v[0..\frac{n}{2}-1]$ e $v[\frac{n}{2}..n-1]$
  - Na segunda vez, cada um dos subproblemas são ainda divididos em mais dois subproblemas cada, gerando quatro subproblemas no total, onde é necessário ordenar os vetores $v[0..\frac{n}{4}-1]$, $v[\frac{n}{4}..\frac{n}{2}-1]$, $v[\frac{n}{2}..\frac{3n}{4}-1]$ e $v[\frac{3n}{4}..n-1]$
  - E assim por diante
  - Além disso, o tempo total que a função `intercala` gasta é proporcional ao número de elementos do vetor v, isto é, $r-p$
  - Portanto, a função `mergesort` consome tempo proporcional a $n \log_2 n$

Exercícios

1. Simule detalhadamente a execução da função `mergesort` sobre o vetor de entrada $v[0..7] = \{3, 41, 52, 26, 38, 57, 9, 49\}$.
2. A função `intercala` está correta nos casos extremos $p = q$ e $q = r$?
3. Um algoritmo de intercalação é **estável** se não altera a posição relativa dos elementos que têm um mesmo valor. Por exemplo, se o vetor tiver dois elementos de valor 222, um algoritmo de intercalação estável manterá o primeiro 222 antes do segundo. A função `intercala` é estável? Se a comparação `v[i] <= v[j]` for trocada por `v[i] < v[j]` a função fica estável?

Exercícios

4. O que acontece se trocarmos `(p + r)/2` por `(p + r - 1)/2` no código da função `mergesort`? Que acontece se trocarmos `(p + r)/2` por `(p + r + 1)/2`?
5. Escreva uma versão da ordenação por intercalação que rearranje um vetor $v[p..r-1]$ em ordem decrescente.
6. Escreva uma função eficiente que receba um conjunto S de n números reais e um número real x e determine se existe um par de elementos em S cuja soma é exatamente X.

Exercícios

7. Agora que você aprendeu o método da ordenação por intercalação, o problema a seguir, que já vimos na aula 2, exercício 2.10, fica bem mais fácil de ser resolvido. Seja A um vetor de n números inteiros distintos. Se $i < j$ e $A[i] > A[j]$ então o par (i, j) é chamado uma **inversão** de A.
   (a) Liste as cinco inversões do vetor $\{2, 3, 8, 6, 1\}$.
   (b) Qual vetor com elementos do conjunto $\{1, 2, \ldots, n\}$ tem o maior número de inversões? Quantas são?
   (c) Qual a relação entre o tempo de execução da ordenação por inserção e o número de inversões em um vetor de entrada? Justifique sua resposta.
   (d) Modificando a ordenação por intercalação, escreva uma função eficiente, com tempo de execução $O(n \log n)$, que determine o número de inversões em uma permutação de n elementos.

Exercícios

8. Escreva um programa para comparar experimentalmente o desempenho da função `mergesort` com o das funções `trocas_sucessivas`, `selecao` e `insercao` da aula 5. Use um vetor com números (pseudo-)aleatórios para fazer os testes.
9. Veja animações dos métodos de ordenação que já vimos nas seguintes páginas:
   - Sort Animation de R. Mohammadi;
   - Sorting Algorithms de J. Harrison;
   - Sorting Algorithms de P. Morin;
   - Sorting Algorithms Animations de D. R. Martin.