Estrutura de Dados

Métodos de Ordenação
Ordenação em tempo linear

Prof. Nilton Luiz Queiroz Jr.

Algoritmos por comparação

- Algoritmos como Quicksort, Mergesort e Heapsort gastam na média um tempo de $O(n \log_2 n)$ para ordenar n elementos;
- Todos esses algoritmos possuem uma característica em comum, eles comparam os elementos de entrada
  - São denominados de algoritmos de ordenação por comparação;
- Esses algoritmos são eficientes, porém, será que existe uma maneira de mostrar que tais algoritmos estão com o melhor tempo de execução em termos de notação assintótica para algoritmos de ordenação por comparação?

Algoritmos por comparação

- Para mostrar o melhor tempo, podemos usar uma árvore de decisão, onde:
  - Cada nó não representa a comparação entre dois elementos;
  - Cada nó folha representa uma permutação;
- A execução de um algoritmo de ordenação consiste em percorrer um caminho de tal árvore;
- Dessa forma, o comprimento do caminho percorrido representa o número de comparações;
  - Ou seja, no pior caso, a altura da árvore;

Algoritmos por comparação

- A quantidade de permutações entre N elementos é dada por N!;
- Uma árvore de altura h tem no máximo 2^h nós;
- Assim, a quantidade de folhas na árvore de comparação é algo maior que N! e menor que 2^h;

$$N! \leq \text{número de folhas} \leq 2^h$$

- Assim, para encontrarmos a altura da árvore com relação a N temos que:

$$\log_2 (N!) \leq \log_2 2^h \rightarrow \log_2 (N!) \leq h$$

- Existe uma prova matemática que diz que $\log_2 (n!)$ é $\Omega (n \log_2 n)$;
- Dessa forma, a altura (número de comparações) é $\Omega (n \log_2 n)$;

Algoritmos de ordenação

- Assim, existem outros algoritmos para ordenar conjuntos de valores que não usam comparação direta entre os elementos da entrada;
  - Countingsort;
  - Radixsort;
  - Bucketsort;

Countingsort

- A ideia do counting sort (ordenação por contagem) é ordenar contando a quantidade de elementos que estão atrás de um determinado valor;
- Suponha que se deseja ordenar elementos inteiros do intervalo 0 até k;
- Utiliza-se vetor auxiliar (contador) com posições de 0 até k, para contar quantas vezes cada elemento aparece, ou seja, contador[i] representa a quantidade de vezes que o elemento i aparece no vetor de entrada;
- Após essa contagem é possível saber qual posição cada elemento do vetor irá ocupar;
    - Basta somar todos os valores que estão no intervalo [0..i-1] do vetor contador;
- Por fim, coloca-se cada valor em sua determinada posição;

Countingsort

- Algoritmo

```
countingsort(v)
        k=maior(v);
        contador[0 .. k+1]
        resposta[1 .. tamanho(v)]
        para i=0 até k faça
                contador[i] = 0
        para j=1 até tamanho(v) faça
                contador[ v[j] ] = contador[ v[j] ]+1
        para i=1 até k faça
                contador[i] = contador[i]+contador[i-1]
        para j = tamanho(v) decrescendo até 1 faça
                resposta[ contador[ v[j] ] ] = v[j]
                contador[ v[j] ] = contador[ v[j] ]-1
        retorne resposta
```

Exemplo

- Ordene os valores a seguir usando o counting sort;

  5, 2, 9, 1, 12, 1, 7, 3, 2

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	0	0	0	0	1	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	0	1	0	0	1	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	0	1	0	0	1	0	0	0	1	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	1	1	0	0	1	0	0	0	1	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	1	1	0	0	1	0	0	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	1	0	0	1	0	0	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	1	0	0	1	0	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	1	1	0	1	0	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	2	1	0	1	0	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	2	1	0	1	0	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	2	1	0	1	0	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	1	0	1	0	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	0	1	0	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	1	0	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	0	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	6	1	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	6	7	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	6	7	7	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	6	7	7	8	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	6	7	7	8	8	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	6	7	7	8	8	8	1
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	6	7	7	8	8	8	9
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	6	7	7	8	8	8	9
0	1	2	3	4	5	6	7	8	9	10	11	12

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	6	7	7	8	8	8	9
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

1	2	3	4	5	6	7	8	9

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	4	5	5	6	6	7	7	8	8	8	9
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

			2					
1	2	3	4	5	6	7	8	9

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	3	5	5	6	6	7	7	8	8	8	9
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

			2	3				
1	2	3	4	5	6	7	8	9

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	3	4	5	6	6	7	7	8	8	8	9
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

			2	3		7		
1	2	3	4	5	6	7	8	9

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	2	3	4	5	6	6	6	7	8	8	8	9
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

	1		2	3		7		
1	2	3	4	5	6	7	8	9

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	1	3	4	5	6	6	6	7	8	8	8	9
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

	1		2	3		7		12
1	2	3	4	5	6	7	8	9

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	1	3	4	5	6	6	6	7	8	8	8	8
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

1	1		2	3		7		12
1	2	3	4	5	6	7	8	9

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	0	3	4	5	6	6	6	7	8	8	8	8
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

1	1		2	3		7	9	12
1	2	3	4	5	6	7	8	9

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	0	3	4	5	6	6	6	7	7	8	8	8
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

1	1	2	2	3		7	9	12
1	2	3	4	5	6	7	8	9

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	0	2	4	5	6	6	6	7	7	8	8	8
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

1	1	2	2	3	5	7	9	12
1	2	3	4	5	6	7	8	9

Exemplo

V:

5	2	9	1	12	1	7	3	2
1	2	3	4	5	6	7	8	9

contador

0	0	2	4	5	5	6	6	7	7	8	8	8
0	1	2	3	4	5	6	7	8	9	10	11	12

resposta

1	1	2	2	3	5	7	9	12
1	2	3	4	5	6	7	8	9

Implementação

- Para implementar o counting sort será utilizada a função maior, que obtém o maior elemento de um vetor:

```
int maior(int v[],int n){
        int m,i;
        m=v[0];
        for(i=1;i<n;i++){
                if(v[i]>m){
                        m=v[i];
                }
        }
        return m;
}
```

Countingsort

- Implementação:
  - Lembre-se que em C os vetores sempre começam no índice 0, isso quer dizer que o vetor resposta e o vetor v índices diferentes dos mostrados no pseudo-código;

```
void counting_sort(int v[],int n){
        /*algoritmo*/
}
```

Countingsort

```
void counting_sort(int v[],int n){
        int i,k, *resposta, *contador;
        if(n>0){
                k=maior(v,n);
                resposta=(int*) malloc (n*sizeof(int));
                contador=(int*) malloc ((k+1)*sizeof(int));
                for(i=0;i<=k;i++)
                        contador[i]=0;
                for(i=0;i<n;i++)
                        contador[ v[i] ] ++;
                for(i=1;i<=k;i++){
                        contador[i]+=contador[i-1];
                for(i=n-1;i>=0;i--){
                        resposta[ contador[ v[i] ] - 1 ] = v[i];
                        contador[v [i] ]--;
                }
                for(i=0;i<n;i++)
                        v[i] = resposta[i];
                free(resposta);
                free(contador);
        }
}
```

Countingsort

- Vantagens:
  - Ordena em tempo linear com relação ao tamanho do vetor;
    - Complexidade O(n+k) onde n é o tamanho do vetor e k é o maior elemento do vetor
  - Não realiza comparações diretas entre os elementos;
  - É um algoritmo estável;
- Desvantagens:
  - Necessita de vetores auxiliares;
  - O tempo e o espaço usado depende também do “tamanho” do maior elemento dentre os que serão ordenados;
    - Por exemplo, para ordenar os vetor [1, 3, 100000000] serão necessárias dois laços de repetição que fazem 100000001 iterações, além de um vetor com 100000001 posições;

Radixsort

- O radixsort (ou ordenação digital) realiza o processo de ordenação para cada dígito dos números, começando pelo dígito menos significativo;
    1. Inicialmente os números são ordenados levando em conta somente o dígito da unidade;
    2. Depois, mantendo a ordem obtida no passo 1, os números são ordenados pelo dígito da dezena;
    3. Mantendo a ordem obtida no passo 2, os números são ordenados pelo dígito da centena;
    4. E assim até que sejam alcançados todos os dígitos do maior número;
- A eficiência do radixsort depende do método usado para ordenar os dígitos;
    - Por esse motivo, geralmente usa-se o counting sort para esse processo de ordenação entre os dígitos;

Radixsort

- Algoritmo do radixsort;

```
radixsort(v)
        m=maior(v)
        d=digitos(m);
        para i=1 até d faça
                ordene os elementos de v pelo i-ésimo dígito menos significativo
                usando um método estável
```

- Nesse caso, o método estável será o countingsort;

Radixsort

- Considerando os valores a seguir aplique o radixsort, mostrando o vetor resultante após cada ordenação;

329, 457, 657, 839, 436, 720, 355

Radixsort

Inicial: 329, 457, 657, 839, 436, 720, 355

Radixsort

Inicial: 329, 457, 657, 839, 436, 720, 355

d=1: 720, 355, 436, 457, 657, 329, 839

Radixsort

Inicial:	329, 457, 657, 839, 436, 720, 355
d=1:	720, 355, 436, 457, 657, 329, 839
d=2:	720, 329, 436, 839, 355, 457, 657

Radixsort

Inicial:	329, 457, 657, 839, 436, 720, 355
d=1:	720, 355, 436, 457, 657, 329, 839
d=2:	720, 329, 436, 839, 355, 457, 657
d=3:	329, 355, 436, 457, 657, 720, 839

Implementação

- Para a implementação do radixsort serão usadas algumas funções auxiliares:
  - Obter o maior elemento de um vetor de tamanho n;
    - Já implementada
  - Contar a quantidade de dígitos de um número;
    - conta_digitos(int numero);
  - Obter o i-ésimo dígito menos significativo de um número;
    - obtem_digito(int numero, int i);
- Além das funções será necessária uma adaptação do algoritmo countingsort, para ordenar os elementos pelo i-ésimo digito;

Implementação

```
int conta_digitos(int n){
        /*algoritmo*/
}
```

```
#include <math.h>
int obtem_digito(int num, int i){
        /*algoritmo*/
}
```

Dica: use a função pow(x,y) que retorna x^y

Implementação

```
int conta_digitos(int n){
        int c=0;
        while(n>0){
                c++;
                n/=10;
        }
        return c;
}
```

```
#include <math.h>
int obtem_digito(int num, int i){
        int mod,div;
        mod = (int)pow(10,i);
        div = (int)pow(10,i-1);
        return (num % mod) / div;
}
```

Implementação

```
void counting_sort(int v[],int n){
        int i,k, *resposta, *contador;
        if(n>0){
                k=maior(v,n);
                resposta=(int*) malloc (n*sizeof(int));
                contador=(int*) malloc ((k+1)*sizeof(int));
                for(i=0;i<=k;i++)
                        contador[i]=0;
                for(i=0;i<n;i++)
                        contador[ v[i] ] ++;
                for(i=1;i<=k;i++)
                        contador[i]+=contador[i-1];
                for(i=n-1;i>=0;i--){
                        resposta[ contador[ v[i] ] - 1 ] = v[i];
                        contador[v [i] ]--;
                }
                for(i=0;i<n;i++)
                        v[i] = resposta[i];
                free(resposta);
                free(contador);
        }
}
```

```
void counting_sort_digito(int v[], int digito, int n){
        /*algoritmo*/
}
```

Implementação

```
void counting_sort(int v[],int n){
        int i,k, *resposta, *contador;
        if(n>0){
                k=maior(v,n);
                resposta=(int*) malloc (n*sizeof(int));
                contador=(int*) malloc ((k+1)*sizeof(int));
                for(i=0;i<=k;i++)
                        contador[i]=0;
                for(i=0;i<n;i++)
                        contador[ v[i] ] ++;
                for(i=1;i<=k;i++)
                        contador[i]+=contador[i-1];
                for(i=n-1;i>=0;i--){
                        resposta[ contador[ v[i] ] - 1 ] = v[i];
                        contador[v [i] ]--;
                }
                for(i=0;i<n;i++)
                        v[i] = resposta[i];
                free(resposta);
                free(contador);
        }
}
```

```
void counting_sort_digito(int v[], int digito, int n){
        int i,k, *resposta, *contador;
        if(n>0){
                k=9;
                resposta=(int*) malloc (n*sizeof(int));
                contador=(int*) malloc ((k+1)*sizeof(int));
                for(i=0;i<=k;i++)
                        contador[i]=0;
                for(i=0;i<n;i++)
                        contador[ obtem_digito(v[i],digito) ] ++;
                for(i=1;i<=k;i++)
                        contador[i]+=contador[i-1];
                for(i=n-1;i>=0;i--){
                        resposta[ contador[ obtem_digito(v[i],digito) ] - 1 ]=v[i];
                        contador[ obtem_digito(v[i],digito) ]--;
                }
                for(i=0;i<n;i++)
                        v[i] = resposta[i];
                free(resposta);
                free(contador);
        }
}
```

Implementação

```
void radixsort(int v[], int n){
    /*algoritmo*/
}
```

Implementação

```
void radixsort(int v[], int n){
    int m,i,num_dig;
    m=maior(v,n);
    num_dig=conta_digitos(m);
    for(i=1;i<=num_dig;i++){
        counting_sort_digito(v,i,n);
    }
}
```

Exercícios

1. Aplique o countingsort no seguinte conjunto de valores:

   1, 4, 1, 3, 6, 1, 4, 8, 2, 3, 7, 2, 3

2. Aplique o radixsort no seguinte conjunto de valores:

   234, 934, 112, 23, 191, 918, 3, 17, 587

Referências

CORMEN, T.H.; LEISERSON,C.E.; STEIN, C.; RIVEST, R.L. Algoritmos: Teoria e Prática. Terceira Edição. Editora Campus, 2011.