INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO - CAMPUS GUARULHOS

FRANCISCO DAVI SILVA
GUILHERME VIEIRA DA SILVA
HENRIQUE FERREIRA CAMELO
PEDRO HENRIQUE DIAS BRAGANÇA DOMINGOS

ALGORITMOS DE ORDENAÇÃO

ShellSort

GUARULHOS
2024

ORIGEM

A informática se baseia em dados. Dados que são inseridos, armazenados, lidos e buscados, entre outras operações, nas máquinas. No entanto, nem sempre esses dados são organizados da melhor forma, causando problemas e excesso de trabalho computacional.

Para se resolver esse problema, criaram-se os algoritmos de ordenação, códigos que rearranjam vetores (sequências de variáveis) e tornam mais facilitada sua busca.

Um desses algoritmos é o ShellSort, desenvolvido em 1959 por Donald Shell. Se trata de uma extensão do método de Inserção, onde diferente daquele ordena os itens a uma distância h, que é um intervalo pré determinado, e não aqueles adjacentes (um ao lado do outro).

De acordo com VIANA (2017):

> Os itens separados de h posições (itens distantes) são ordenados: o elemento na posição x é comparado e trocado (caso satisfaça a condição de ordenação) com o elemento na posição x-h. Este processo se repete até h=1, quando esta condição é satisfeita o algoritmo é equivalente ao método de inserção.

DESEMPENHO E SEQUÊNCIA DE KNUTH

Seu melhor caso de desempenho acontece quando a coleção de dados está parcialmente ordenada, tendo um custo super-linear de comparações em função de seus n elementos:

$O(n \log_2 n)$ comparações

Seu desempenho no pior caso acontece quando os dados estão ordenados de forma inversa, tendo um custo quadrático de comparações em função da quantidade de elementos n:

$O(n^2)$ comparações

Ainda assim, seu desempenho depende da sequência de intervalos h escolhida para as comparações, que é definida de forma arbitrária. Em seu início, Shell a definiu como h = n/2, fórmula que mais tarde se provou não tão eficaz.

A formulação mais eficiente foi defendida por Donald Knuth em 1973, onde mostrou de forma experimental que tal sequência dificilmente teria uma concorrência pelo menos 20% mais eficiente. Segue a fórmula:

Para S = 1:

h(S) = 1

Para S > 1:

h(S) = 3h(S - 1) + 1

A sequência para h corresponde a **1, 4, 13, 40, 121, 364, 1.093, 3.280, ...**

Vantagens e Desvantagens

Como vantagens, podemos citar sua eficiência para arquivos de tamanho moderado e sua fácil implementação em poucas linhas de código. Como desvantagens, temos seu tempo de execução que depende da ordem de ordenação inicial do arquivo e da sequência do intervalo escolhido.

CÓDIGO EM LINGUAGEM C

```
void shellsort(struct item *vetor, int tamanho) {
int i, j, intervalo;
struct item auxiliar;

/* calcula o intervalo inicial. */
for(intervalo = 1; intervalo < tamanho; intervalo = 3*intervalo+1);

while(intervalo > 0) {
      /* atualiza o valor do intervalo. */
      intervalo = (intervalo-1)/3;

      for(i = intervalo; i < tamanho; i++) {
      auxiliar = vetor[i];
      j = i;

/* efetua comparações entre elementos com distância do intervalo: */
            while(vetor[j - intervalo].chave > auxiliar.chave) {
                  vetor[j] = vetor[j - intervalo];
                  j -= intervalo;
```

```
                    vetor[j] = auxiliar;
                }
        }
}
```

PASSO A PASSO

for(intervalo = 1; intervalo < tamanho; intervalo = 3*intervalo+1);

- Define o valor inicial do intervalo dependendo do tamanho do vetor.
- O intervalo só pode assumir os valores 1, 4, 13, 40, 121, 364, 1.093, 3.280, ... (no caso da sequência de Knuth),
- O laço termina em si mesmo e acaba quando o intervalo assume o valor mais próximo e menor do que o tamanho do array seguindo os valores acima.
- Exemplo: Se o tamanho do array for 15, seu valor inicial será 13.

while(intervalo > 0) {
intervalo = (intervalo-1)/3;

- Diminui o valor do intervalo para cada ciclo efetuado enquanto o intervalo for maior que zero.
- Note que a operação (intervalo-1)/3 é o inverso da operação apresentada no for anterior (3*intervalo+1).
- Portanto, o intervalo ainda segue a mesma sequência de valores: 1, 4, 13, 40, 121, 364, 1.093, 3.280, ...

for(i = intervalo; i < tamanho; i++) {
auxiliar = vetor[i];
j = i;

- Laço para atribuir valor às variáveis.
- Coloca a variável i na mesma posição do intervalo. Enquanto i for menor que o tamanho do vetor, ou seja, ainda estiver dentro do vetor, o valor de i é incrementado.
- A variável auxiliar recebe os valores dentro da posição indicada por i

- j fica na mesma posição de i.

while(vetor[j – intervalo].chave > auxiliar.chave) {
vetor[j] = vetor[j – intervalo];
j -= intervalo;

- Faz uma verificação: executa o código enquanto o valor da chave contido na posição (j - intervalo) for maior que o valor da chave na posição da variável auxiliar.

- Se a verificação acima for verdadeira, os valores do vetor na posição j recebem os valores da posição j - intervalo.

- Após isso, ocorre a atribuição j-= intervalo, que nada mais é que j = j - intervalo, o que faz com que j vá para a posição que ele estava anteriormente verificando.

vetor[j] = auxiliar;

- Por último, o vetor na posição j agora recebe os valores contidos na variável auxiliar

Em resumo, o código verificou se um determinado valor atrás da variável i era maior que o valor contido em i. Então, caso fosse, o valor contido na posição da variável i é passado para trás e o valor de trás é passado para frente, assumindo o valor na posição do i.

BIBLIOGRAFIA

TOFFOLO, Túlio. Ordenação: ShellSort. Disponível em: http://www3.decom.ufop.br/toffolo/site_media/uploads/2013-1/bcc202/slides/16._shellsort.pdf. Acesso em: 15 mar. 2024.

CAROLINA, Ana. Funcionamento do Shell Sort. **Youtube,** 2012. Disponível em: https://www.youtube.com/watch?v=M3bS6w1R434. Acesso em: 15 mar. 2024.

UDIPROD. Shell sort vs Insertion sort. **Youtube,** 2022. Disponível em: https://www.youtube.com/watch?v=g06hNBhoS1k. Acesso em: 15 mar. 2024.

VIANA, Daniel. Conheça os principais algoritmos de ordenação. **TreinaWeb,** 2017. Disponível em: https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao#:~:text=Selection%20Sort,-A%20ordena%C3%A7%C3%A3o%20por&text=Para%20todos%20os%20casos%20oS1k. Acesso em: 15 mar. 2024.

SHELL SORT. **Desenvolvendo Software,** 2023. Disponível em: http://desenvolvendosoftware.com.br/algoritmos/ordenacao/shell-sort.html. Acesso em: 15 mar. 2024.

SHELL SORT: Gap sequences. **Wikipedia,** 2023. Disponível em: https://en.wikipedia.org/wiki/Shellsort#Gap_sequences. Acesso em: 15 mar. 2024.

SIMIC, Milos. The Complexity of Shellsort. **Baeldung,** 2023. Disponível em: https://www.baeldung.com/cs/shellsort-complexity. Acesso em: 15 mar. 2024.