INSTITUTO FEDERAL DE MATO GROSSO DO SUL, TRÊS LAGOAS

ANÁLISE SOBRE ALGORITMOS DE ORDENAÇÃO

ALUNO: NATHAN KURTHS MENDONÇA CONDE

DISCIPLINA: ESTRUTURA DE DADOS I

TRÊS LAGOAS, 30 DE AGOSTO DE 2022

OBJETIVO

O objetivo do relatório é discorrer sobre as diferenças entre os métodos (SELECTION, MERGE, QUICK, BUBBLE) sort, comparando as suas eficiências de maneiras claras e objetivas, por meio de testes de compilação ou testes de mesa, para melhor entendimento do seu trabalho no computador. Além disso, foi solicitado na disciplina que alterasse o algoritmo Quick Sort afim de incrementar sua eficiência, mas caso o resultado oposto for evidenciado, é de interesse acadêmico justificá-lo.

Atenção: A localização dos códigos se deve ao relatório levar em consideração que o leitor consulta o meu github: https://github.com/NathanKurths/Estrutura-de-Dados-I

INTRODUÇÃO

Entende-se como algoritmos de ordenação um conjunto de funções e estratégias em uma linguagem computacional com o objetivo de organizar um vetor de diversos tamanhos **n**. Neste caso a ordenação foi manipulada em linguagem C, contudo, em algumas exceções se julgou interessante experimentar bibliotecas do C++, com a mesma finalidade de ordenar o vetor. Para melhor compreensão, será definido os quatro algoritmos de forma objetiva e supostamente resumida, todos utilizarão como exemplo o mesmo vetor; imagina-se um vetor com 5 espaços, e preenchidos da seguinte maneira: |67|30|44|11|42|.

Primeiramente, o algoritmo Bubble Sort Entende-se que por padrão, inicializamos o algoritmo no início do vetor, ou seja, V[0], V de vetor. Este V[0] que armazena o valor 67, irá se comparar efetuando a condição de deduzir qual o menor valor com a próxima posição, ou seja, V[1] que armazena o valor 30, e nesse caso o 30 possui menor valor, então o valor troca de posição, logo: V[0] armazena 30 e V[1] armazena 67, em seguida o V[1] se compara com V[2], que no caso V[1] é de fato maior do que V[2], então a mesma ação acontece, e assim sucessivamente, caso o valor a direita seja menor, não ocorre a troca mas ainda prossegue a comparação. Ao atingir o final do vetor ele retorna para o início, pois na maior parte dos casos o vetor está somente menos desordenado em uma verificação, sendo necessária sua execução mais algumas vezes para que aos poucos o vetor se ordene até finalmente não restar dúvidas de que está organizado. Logo, não é um algoritmo muito interessante, e ruim para ordenar vetores grandes.

Em segundo lugar, introduziremos o algoritmo Selection Sort. Para compreendê-lo, também é simples. O V[0] será comparado com todas as casas à sua direita, procurando o menor valor entre todos, quando encontrado o vetor troca de posição com ele, e após feito a troca, será aplicado a mesma lógica para cada elemento, da esquerda pra direita. O nosso vetor comentado

anteriormente trocaria o valor da primeira posição do vetor com o valor 11 que está no V[3] por exemplo. Também é um método não usual, pois para vetores grandes o exercício é extenso.

Em terceiro lugar, o algoritmo Merge Sort que utiliza conquista-dominação. É o mais eficiente dos quatro, pois funciona de forma efetiva para qualquer tipo de vetor, e acompanha uma certa complexidade de funcionamento. O algoritmo define um meio através de uma divisão por 2, que para isso pode-se usar a função floor para arredondar em casos de vetores ímpares, formando sub-vetores popularmente definidos como esquerda e direita, e vão sendo divididos em mais sub-vetores até ficarem sozinhos. Ou seja, um vetor com 8 espaços vai se dividir em dois de 4, depois 4 de 2, e depois os valores ficarão isolados. Quando ficarem isolados o programa para de dividir, e faz o contrário, agora junta os valores de forma ordenada até retornar a um vetor de 8 espaços, ou seja:cada parte isolada vai ser comparada em pares, e se ordenar, depois em quartetos ordenados, depois no vetor completo.

Por fim, Quick Sort. Este método é eficiente na ordenação conquista-dominação, principalmente para vetores pequenos, contudo sua teoria é popularmente intermediária-complexa para vetores grandes. Inicialmente é designado um pivô, uma posição ou um indicador, alguma definição da sua preferência, que será um valor, normalmente, do vetor marcado. Dessa forma, o vetor se organizará comparando esse pivô com todos os demais valores, e caso ele se comparar com um valor menor do que ele, será alocado para a esquerda, e caso um valor maior do que ele, será alocado para a direita. Em vetores menores é possível que se ordene só com a influência do pivô, como por exemplo, nosso pivô sendo o valor 42 do V[4], irá trocar de posição com V[0], 42 como V[0] irá trocar de posição com V[1], 42 como V[2] irá trocar de posição com V[3], e 42 como V[3] troca de posição com V[2]. Ao percorrer esse vetor foi efetuado trocas com o intuito de deixar valores maiores que o pivô a sua direita, e menores na sua esquerda de forma conclusiva por ser pequeno. Em vetores grandes, além de ser feito a mesma descrição, há chamada de recursão na função que faz esta separação, para ordenar o que está no lado esquerdo e no que está no lado direito, pois de fato os menores valores estarão do lado esquerdo, mas possivelmente não estarão de forma crescente que é o que nos interessa ao ordenar, por isso a recursão, serve também para o lado direito.

DESCRIÇÃO DAS ATIVIDADES

Primeiramente foi executado uma versão padrão dos algoritmos afim de estudá-los, havendo algumas alterações de variáveis e outras ferramentas para melhor estudo pessoal, acompanhado de leitura sobre o conteúdo.

Na IDE Visual Studio Code foi criado uma série de arquivos .c com diversas nomeações, sendo relevante os arquivos com a palavra “Estudando” para o conteúdo deste relatório, por exemplo: EstudandoQsort.c, EstudandoBubble.c, e etc. É importante compreender que esses são os algoritmos manipulados de

formas triviais, onde estão na pasta pai do repositório, havendo outras pastas com outros conteúdos.

Na pasta “modificando Qsort” se encontra alterações feita nessa ordenação de certa forma mais trabalhada, uma vez que combina técnicas de recursividade, sendo esse o seu maior diferencial para todos os outros métodos. Neste foi incluído a lógica de estudar se o suposto pivô definido de forma aleatória (em inglês, random), seria mais interessante do que padronizá-lo no início ou no final do vetor. Para isso foi criado uma função com esse objetivo, e incrementado na versão padrão do código, para assim ser compilado e testado. **Há nos códigos funções de calcular o tempo, afim de compará-los em execução e produzir este relatório.**

RESULTADOS

Foi possível concluir que os algoritmos Bubble e Selection são meramente viáveis para fins didáticos, por serem bastante simples. É interessante acentuar que o seu uso em vetores pequenos pode ser viável se o programador preferir um método menos trabalhoso, pois ele percorre o vetor inteiro procurando oportunidades de efetuar trocar. Já o Merge e Quick sort são efetíveis de fato para serem aplicados em trabalhos com vetores, com certos adendos: O Merge Sort é interessante para qualquer tamanho de vetor, trabalha igual, já o Quick Sort é melhor para vetores menores mesmo sendo um bom método para vetores grandes também. O espaço que o Quick Sort usufrui da memória é de fato menor em comparação com o outro algoritmo.

Além disso, foi possível aplicar efetivamente o método Random em escolha de pivô no Quick Sort, que se mostrou mais interessante do que definir um pivô pelas populares variáveis ‘’esquerda’’ ou ‘’direita’’ pelo motivo de que o pivô tem certo pré-controle sobre o custo computacional solicitado nas futuras trocas de posições dos elementos.

Todos esses resultados levam em consideração funções para comparar o tempo e fazer o teste compilando.

CONCLUSÃO

Subentende-se que para fins acadêmicos o Bubble e Selection são indispensáveis para introduzir a ordenação de vetores para estudiosos da área, mas para aplicação futura ou de exame é interessante explorar o Quick e Merge, pois são efetivos para vetores mais complexos. Sendo assim, acrescenta-se que a escolha do pivô no algoritmo Quick Sort é influenciável, uma vez que o valor designado como pivô define quantos elementos vão efetuar troca.

BIBLIOGRAFIA

(foi muito relevante assistir vídeos de programadores para complementar os meus estudos pessoais)

1. https://joaoarthurbm.github.io/eda/posts/merge-sort/#:~:text=Merge%20Sort%20%C3%A9%20um%20algoritmo,que%20n%E2%88%97logn.
2. https://www.youtube.com/watch?v=5prE6Mz8Vh0
3. https://www.youtube.com/watch?v=wx5juM9bbFo