Trabalho Prático 0: Operações de Nubby

Algoritmos e Estruturas de Dados III – 2017/2
Entrega: 30/08/2017

1 Introdução

Nubby é um adolescente que possui vários hobbies considerados incomuns, alguns deles são programação e realizar operações matemáticas básicas. Recentemente, Nubby está aprendendo algumas estruturas de dados avançadas como árvores binárias de busca em um curso online. Nubby, esta perdido, ele não consegue observar como estruturas de dados podem influenciar a eficiência dos algoritmos, para ele toda a abstração de armazenar dados estruturalmente é pura ilusão.

O passa tempo favorito de Nubby é realizar operações com elementos alocados sequencialmente em um vetor. Bryan, amigo de Nubby, formado em computação observou a sua frustração com estruturas de dados. Conhecendo os passa tempos de seu amigo, Bryan propôs a Nubby um desafio.

O desafio proposto consiste em dado um vetor $V = [v_1, v_2, .., v_n]$ com n valores inteiros alocados sequencialmente, e duas posições i e j, deve-se realizar buscas para computar o valor $Min_{i,j}$, $Max_{i,j}$ e $Sum_{i,j}$, que são encontrar o mínimo, máximo e soma dos os elementos que estão entre v_i e v_j respectivamente. Outras operações além das buscas definidas acima podem ser realizadas, tais como adicionar ou subtrair um a todos os valores em um intervalo.

Como operações com elementos alocados sequencialmente em um vetor é o passa tempo favorito de Nubby, logo apos receber o desafio, ele afirmou que o problema era muito simples, basta apenas criar uma estrutura T que armazena os valores de mínimo, máximo e soma e pre-computar os valores $Min_{i,j}$, $Max_{i,j}$, $Sum_{i,j}$ para todos os possíveis intervalos e armazenar o resultado em uma matriz $M = [a_{i,j}] \in T^{m \times n}$ onde cada elemento $a_{i,j} = (Min_{i,j}, Max_{i,j}, Sum_{i,j})$ armazena o valor de mínimo, máximo e soma do seu respectivo intervalo.

Para realizar a busca do valor de uma soma em um intervalo que vai de i até j basta apenas acessar as posições $a_{i,j}$ da matriz e retornar o valor da operação desejada. Caso um valor do vetor for alterado, Nubby afirmou

(1,1,1)	(1,3,4)	(1,4,8)	(-1,4,7)	(-1,4,7)
(1,3,4)	(3,3,3)	(3,4,7)	(-1,4,6)	(-1,4,6)
(1,4,8)	(3,4,7)	(4,4,4)	(-1,4,3)	(-1,4,3)
(-1,4,7)	(-1,4,6)	(-1,4,3)	(-1,-1,-1)	(-1,0,-1)
(-1,4,7)	(-1,4,6)	(-1,4,3)	(-1,0,-1)	(0,0,0)

Tabela 1: Resultados pre-computados utilizando a solução proposta por Nubby. Para a busca da soma dos elementos no intervalo $[1, 4]$ basta apenas acessar a primeira linha e quarta coluna e retornar o campo referente a soma (neste caso 7).

que bastava computar os valores da matriz novamente. A tabela 1 mostra como seria uma matriz utilizando a estratégia de Nubby com o vetor $v = [1, 3, 4, -1, 0]$ como entrada.

Bryan, contestou seu amigo, apresentando-o uma nova estrutura de dados semelhante a árvore binária em que consiste em segmentar o vetor em intervalos e representar os intervalos como nós de uma árvore (Árvore de Segmentos/Segment Tree). A estrutura é definida da seguinte forma, as folhas da árvore representam elementos do vetor (intervalos de tamanho 1), os outros nós que não são folhas representam a "união"dos resultados dos nós que são seus filhos ("união"de resultados dos intervalos). A figura 1 mostra uma possível representação da Árvore de Segmentos para o vetor $v = [1, 3, 4, -1, 0]$, cada tupla (a, b, c) que está contido dentro de um nó representa os valores que o nó armazena e o subscrito $[i, j]$ indica que o nó armazena o resultado do intervalo de i até j.

Para buscar o valor $Sum_{1,2}$ basta apenas realizar uma busca na árvore iniciando pelo nó raiz que armazena o resultado do intervalo $[1, 5]$ em seguida mover para seu filho a esquerda que representa o intervalo $[1, 3]$, atualizar o nó atual e finalmente mover para o filho a esquerda que representa o intervalo $[1, 2]$ em seguida retorna o valor do elemento da tupla que representa a soma. Outro detalhe apresentado por Bryan é que se apenas um único elemento do vetor for atualizado não é necessário construir uma nova árvore , basta apenas atualizar o nó folha referente ao elemento e todos os seus nós antecessores até a raiz.

Nubby esta com uma deadline em seu curso online, e para esse desafio ele conta com a sua ajuda, ele pediu para você avaliar se a estrutura de dados apresentada por Bryan realmente possui vantagens sobre a sua ideia inicial de armazenar os resultados em uma matriz.

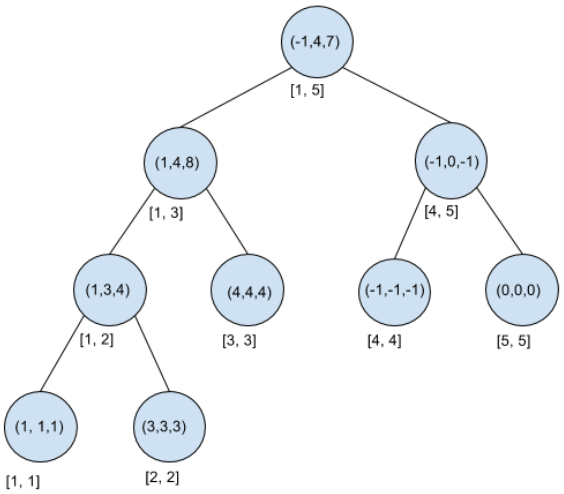

Figura 1: Árvore de segmentos: cada nó não folha da árvore armazena a união dos resultados de seus filhos, cada nó folha armazena os resultados de um intervalo unitário. O nó raiz armazena os resultados do intervalo $[1, 5]$, em seguida o intervalo $[1, 5]$ é particionado ao meio criando dois novos nós um representando o intervalo $[1, 3]$ e outro o intervalo $[4, 5]$, como o tamanho do intervalo não é par o nó da esquerda representa um intervalo com um elemento a mais, em seguida os intervalos são partidos ao meio até se tornarem intervalos que possuem apenas um único elemento e são representados pelas folhas.

2 Entrada e saída

Entrada A primeira linha contêm 2 inteiros positivos **N** ($1 \leq N \leq 10^6$) o número de elementos no vetor e **M**($1 \leq M \leq 10^7$) a quantidade de consultas a serem realizadas.

A segunda linha contém **N** números inteiros. Os elementos estarão na ordem em que eles devem ser armazenados no vetor o qual o primeiro elemento possui índice 1 e não 0 como na linguagem C.

Cada uma das **M** linhas em seguida inicia-se com uma string **Add**, **Sub**, **Min**, **Max** ou **Sum** e dois inteiros positivos i e j ($1 \leq i \leq j \leq N$) os quais indicam as seguintes operações:

- **Add** adiciona 1 a todos os elementos no intervalo $[i, j]$
- **Sub** subtrai 1 a todos os elementos no intervalo $[i, j]$

- **Min** consulta o valor mínimo dos elementos intervalo $[i, j]$
- **Max** consulta o valor máximo dos elementos no intervalo $[i, j]$
- **Sum** consulta a soma dos elementos no intervalo $[i, j]$

Exemplo de entrada

```
5 7
1 3 4 −1 0
Sum 1 5
Max 4 5
Min 4 5
Add 4 4
Sum 4 5
Sub 1 3
Sum 1 3
```

Saída Para cada uma das operações **Min**, **Max** ou **Sum** imprima uma linha com o resultado da operação. Os resultados das operações estarão no intervalo $[-2^{31} + 2, 2^{31} - 2]$.

Exemplo de saída

```
7
0
−1
0
5
```

3 O que deve ser entregue

Deverá ser submetido um arquivo **.zip** contendo apenas uma pasta chamada tp0, esta pasta deverá conter: (i) Documentação **em formato PDF** e (ii) Implementação.

Documentação Deverá ter no máximo 10 páginas e seguir tanto os critérios de avaliação discutidos na Seção 4.1, bem como as diretrizes sobre a elaboração de documentações disponibilizadas no *moodle*. Além disso, a documentação deverá conter análise experimental validando as complexidades de tempo e espaço e comparação entre os dois métodos propostos (método utilizando matrizes e árvore de segmentos).

Implementação Código fonte do seu TP (*.c* e *.h*), **a implementação de cada um dos métodos deve estar explicitamente separadas**, por exemplo crie um arquivo *matriz.*{*c/h*} e outro *arvore.*{*c/h*}. Para os dois métodos implemente as operações o mais eficiente possível.

Makefile Inclua um *makefile* na submissão que permita compilar o trabalho. **Deverá ser produzido dois executáveis** um chamado *matriz* e outro chamado *arvore*. O executável *matriz* deverá executar a entrada utilizando o método de armazenar os resultados em uma matriz enquanto o executável chamado *arvore* deverá executar o método utilizando a árvore de segmentação.

É obrigatório o uso das *flags*: **-Wall -Wextra -Werror -std=c99 -pedantic -O2**.

Execução A entrada e saída devem ser lida e impressa utilizando a entrada padrão (stdin). Um exemplo de execução utilizando o executável *arvore* para um arquivo de entrada *in_1.in* e imprimindo o resultado em *out_1.out* será da seguinte forma:

$$./arvore < in\_1.in > out\_1.out$$

4 Avaliação

Eis uma lista **não exaustiva** dos critérios de avaliação que serão utilizados.

4.1 Documentação

Introdução Inclua uma breve explicação do problema que está sendo resolvido no seu trabalho.

Solução do Problema Você deve descrever cada solução do problema de maneira clara e precisa, detalhando e justificando os algoritmos e estruturas de dados utilizados. Para tal, artifícios como pseudo-códigos, exemplos ou diagramas podem ser úteis. Note que documentar uma solução não é o mesmo que documentar seu código. **Não** é necessário incluir trechos de código em sua documentação nem mostrar detalhes de sua implementação, exceto quando estes influenciem o seu algoritmo principal, o que se torna interessante.

Análise de Complexidade Inclua uma análise de complexidade de tempo e espaço dos principais algoritmos e estrutura de dados utilizados. Cada complexidade apresentada deverá ser devidamente **justificada** para que seja aceita. Faça uma análise para cada tipo de algoritmo implementado. Mostre vantagens e desvantagens de cada um dos modelos utilizados.

Avaliação Experimental Sua documentação deve incluir os resultados de experimentos que avaliem o tempo de execução de seu código em função de características da entrada. Cabe a você gerar entradas para esses experimentos. Para tal, um gráfico mostrando o tempo de execução em função do tamanho da entrada pode ser interessante. Você também deve interpretar os resultados obtidos. Comente sobre cada gráfico ou tabela que você apresentar mostrando o que é possível concluir a partir dele.

4.2 Implementação

Linguagem & Ambiente O seu programa deverá ser implementado na linguagem **C** e poderá fazer uso de funções da biblioteca padrão da linguagem. Trabalhos que utilizem qualquer outra linguagem de programação ou utilizem outras bibliotecas que não seja padrão ou variáveis globais serão zerados. Além disso, certifique-se que seu código compile e funcione corretamente nas máquinas **Linux** dos laboratórios do DCC.

Casos de teste A sua implementação passará por um processo de correção automatizado, portanto, o formato da saída do seu programa deve ser idêntico aquele descrito nessa especificação. Saídas com qualquer divergência serão consideradas erradas, mesmo que as divergências sejam *whitespaces*. e.g. espaços, *tabs*, quebras de linha, etc. Para auxilia-lo na depuração do seu código, será fornecido um pequeno, **não-exaustivo**, conjunto de entradas e suas respectivas saídas. É seu dever certificar-se que seu código funciona corretamente para qualquer entrada válida.

Alocação Dinâmica Algoritmos e estruturas de dados deverão fazer uso de memória alocada dinamicamente (`malloc()` ou `calloc()`). Certifique-se que seu programa utiliza essas regiões de memória corretamente, pois os monitores penalizarão implementações que realizam *out-of-bounds access* e que tenham vazamento de memória (não desalocar memoria dinâmica).
A alocação dinâmica deverá fazer uso das funções malloc() ou calloc() da biblioteca padrão C, bem como liberar tudo o que for alocado utilizando

free(), para gerenciar o uso da memória. **DICA:** Utilize `valgrind` antes de submeter o seu TP.

Qualidade do código Seu código também será avaliado no quesito de legibilidade, dando atenção, porém não limitando-se, aos seguintes items: (i) **INDENTAÇÃO**; (ii) nomes de variável e função descritivos e claros; (iii) Modularização adequada; (iv) Comentários dentro de funções, explicando o que certos trechos mais complicados fazem; (v) Comentários fora de funções, explicando, em alto-nível, o que as funções mais importantes fazem; (vi) funções concisas que desempenham somente uma tarefa; (vii) **Proibido uso de variáveis globais**.

Atrasos Trabalhos poderão ser entregues após o prazo estabelecido, porém sujeitos a uma penalização regida pela seguinte fórmula:

$$\Delta_p = \frac{2^{d-1}}{0.32}\,\%$$

Por exemplo, se a nota dada pelo corretor for 70 e você entregou o TP com 4 dias corridos de atraso, sua penalização será de $\Delta_p = 25\%$ e, portanto, a sua nota final será: $N_f = 70 \cdot (1 - \Delta_p) = 52.2$. Note que a penalização é exponencial e 6 dias de atraso resultam em uma penalização de 100%.

5 Considerações Finais

Como Nubby gosta apenas de memória alocada sequencialmente a sua árvore deverá ser armazenada em um vetor alocado sequencialmente, caso contrário ele considerará que você trapaceou. Existem vários materiais sobre árvore de segmentação de ótima qualidade online .

Assim como em todos os trabalhos dessa disciplina é estritamente proibida a copia parcial ou integra de códigos, seja da internet ou de colegas. Utilizaremos o algoritmo *MOSS* para detecção de plágio em trabalhos, seja honesto. Você não aprende nada copiando código de terceiros nem pedindo a outra pessoa que faça o trabalho por você. Se a cópia for detectada, sua nota será zerada e os professores serão informados para que as devidas providências sejam tomadas.