Trabalho Prático 3

Transformações lineares em Nlogônia

Júlio Guerra Domingues (2022431280)

Estruturas de Dados - DCC/UFMG - 2023-2

Introdução

No Trabalho Prático 3 de Estruturas de Dados, foi proposto o desafio de aplicar transformações lineares em pontos desenhados em uma folha de papel na cidade fictícia de Nlogônia. A complexidade do problema está relacionada à necessidade de computar as posições finais dos pontos após várias transformações lineares ao longo do tempo.

A solução proposta permite a atualização eficiente das transformações e a consulta rápida das posições finais dos pontos. Para tal, empregou-se, conforme sugerido, árvores de segmentação.

Método

O programa foi desenvolvido na linguagem C++, compilado pelo compilador g++, contido na GNU Compiler Collection. A máquina utilizada tem as seguintes especificações:

- Sistema Operacional: MacOS Sonoma 14.0
- Processador: ARM Apple Silicon M2
- Memória RAM: 8,00 GB

O programa é estruturado da seguinte forma:

- main.cpp: Este arquivo gerencia a entrada de dados, processa as operações e controla o fluxo principal do programa. Inicializa uma instância da árvore de segmentação (SegTree) e processa as operações de atualização (u) e consulta (q).
- matriz.hpp e matriz.cpp: Implementam a classe Matriz, que gerencia as operações matriciais. Incluem métodos para definir, somar e acessar os índices das matrizes, além de uma função de multiplicação de matrizes.
- segtree.hpp e segtree.cpp: Definem a classe SegTree, responsável pela estrutura da árvore de segmentação. Contém métodos para atualização e consulta, manipulando os nós da árvore para aplicar as transformações lineares.

Análise de Complexidade

- Atualizações na SegTree (atualiza): A complexidade é O(log n) para cada atualização, onde n é o número de instantes de tempo, pois a árvore de segmentação divide o intervalo de tempo pela metade em cada nível.
- Consultas na SegTree (consulta): De forma semelhante, a complexidade é O(log n) devido à natureza da busca binária na árvore.
- Operações de Matriz: A complexidade da multiplicação de matrizes é O(1), pois as matrizes são de tamanho fixo (2x2).

Estratégias de Robustez

Algumas estratégias de robustez foram implementadas, de modo a garantir que o programa lide com exceções e erros de forma apropriada, evitando falhas inesperadas e tornando-o mais resistente a entradas incorretas ou inesperadas. Destacam-se as seguintes:

- Validação de Entrada em main.cpp: O programa inclui verificações para garantir a correta leitura dos dados de entrada e a validade dos valores recebidos para cada operação específica (atualização e consulta). Em caso de entrada inválida, o programa informa o usuário sobre o erro específico e salta a iteração atual, continuando com a próxima operação.
- Verificação de Índices em matriz.cpp: A classe Matriz inclui verificações de índices em métodos críticos como defineIndice, somaIndice e pegaIndice. Tais verificações garantem que os índices usados estão dentro dos limites da matriz, prevenindo acessos fora dos limites do array e potenciais falhas de segurança.
- Tratamento de Exceções para Aritmética em matriz.cpp: Foi implementado tratamento de exceções para erros de índice, de forma a prevenir falhas inesperadas devido a erros de cálculo ou acessos inválidos.
- Validação de Índices e Tratamento de Exceções em segtree.cpp: A classe SegTree realiza validação de índices nas operações de atualização e consulta, assegurando que os índices usados estejam dentro dos intervalos esperados.
- Tratamento de Exceções para Alocação de Memória em segtree.cpp: Existe uma verificação de argumento válido no construtor de SegTree, que previne a criação de árvores de segmentação com tamanhos inválidos, o que poderia levar a alocações de memória falhas ou comportamento inesperado.

- Testes abrangentes: Alguns testes foram implementados no Makefile, considerando os exemplos fornecidos no documento com as orientações do TP e exemplos gerados com o programa gerador de testes fornecido (podem ser verificados ao rodar com *make test*). Além disso, o programa foi testado nos VPLs no Moodle, o que contribui para sua robustez.

Análise Experimental

O programa foi avaliado por meio das ferramentas Valgrind (Memcheck, Callgrind e Cachegrind), bibliotecas Memlog (fornecida pelo professor), chrono e gprof. Tais ferramentas forneceram informações importantes sobre vários aspectos do desempenho, incluindo gerenciamento de memória, eficiência de cache e análise de chamadas de funções.

```
● @juliogdomingues → /workspaces/estruturas_de_dados_ufmg/newtp (main) $ valgrind --leak-check=full --show-leak-kinds=all -s ./bin/tp3.out < test1.txt
  ==10679== Memcheck, a memory error detector
  ==10679== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
  ==10679== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
  ==10679== Command: ./bin/tp3.out
  ==10679==
  1 0
  0 1
  ==10679==
  ==10679== HEAP SUMMARY:
  ==10679==     in use at exit: 0 bytes in 0 blocks
  ==10679==   total heap usage: 6 allocs, 6 frees, 86,730 bytes allocated
  ==10679==
  ==10679== All heap blocks were freed -- no leaks are possible
  ==10679==
  ==10679== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Com o Valgrind Memcheck, observa-se que o programa não sofre de vazamentos de memória significativos, indicando um gerenciamento de memória eficiente. Esse aspecto é crucial, especialmente para programas que processam grandes volumes de dados ou que precisam manter a estabilidade durante longos períodos de execução.

```
● @juliogdomingues → /workspaces/estruturas_de_dados_ufmg/newtp (main) $ valgrind --tool=cachegrind ./bin/tp3.out < test1.txt
  ==12838== Cachegrind, a cache and branch-prediction profiler
  ==12838== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
  ==12838== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
  ==12838== Command: ./bin/tp3.out
  ==12838==
  --12838-- warning: L3 cache found, using its data for the LL simulation.
  1 0
  0 1
  ==12838==
  ==12838== I   refs:      2,541,282
  ==12838== I1  misses:        2,139
  ==12838== LLi misses:        2,001
  ==12838== I1  miss rate:      0.08%
  ==12838== LLi miss rate:      0.08%
  ==12838==
  ==12838== D   refs:        807,325  (588,540 rd   + 218,785 wr)
  ==12838== D1  misses:       17,483  ( 14,854 rd   +   2,629 wr)
  ==12838== LLd misses:       10,090  (  8,397 rd   +   1,693 wr)
  ==12838== D1  miss rate:       2.2% (    2.5%     +     1.2%  )
  ==12838== LLd miss rate:       1.2% (    1.4%     +     0.8%  )
  ==12838==
  ==12838== LL refs:          19,622  ( 16,993 rd   +   2,629 wr)
  ==12838== LL misses:        12,091  ( 10,398 rd   +   1,693 wr)
  ==12838== LL miss rate:        0.4% (    0.3%     +     0.8%  )
```

O Cachegrind evidencia que o programa apresenta uma baixa taxa de erro de cache para instruções, sugerindo uma boa localidade de memória neste aspecto. A análise do Callgrind, por sua vez, indica que o programa não possui funções individuais que consomem tempo significativo, o que sugere uma distribuição equilibrada do tempo de execução entre as funções. O gprof também indicou que nenhuma função

acumulou um tempo significativo, mesmo com o caso teste extensivo fornecido pelo monitor.

Conclusões

O TP3 foi focado na implementação de estruturas de dados avançadas para efetuar transformações lineares em pontos, em diferentes instantes de tempo. Este projeto demonstrou a importância de aplicar conceitos de árvores de segmentação e operações matriciais no contexto de algoritmos eficientes.

Assim como nos demais trabalhos práticos, ressalta-se a importância da escolha de estruturas de dados eficientes e bem adaptadas ao domínio em problemas que envolvem manipulação dinâmica de informações. A árvore de segmentação mostrou-se uma ferramenta poderosa para resolver problemas de intervalos dinâmicos, permitindo atualizações e consultas rápidas, fundamentais em cenários de transformações lineares contínuas. Além disso, pude aprofundar o entendimento sobre otimizações de operações matriciais, essenciais para manter a eficiência em cálculos repetitivos.

Bibliografia

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms* (4th ed.). MIT Press. ISBN: 9780262046305.
2. Sedgewick, R., & Wayne, K. (2011). *Algorithms*. Addison-Wesley Professional.
3. Maratona UFMG. (2023, December 1). *Aula 9 - SegTree*. YouTube. https://www.youtube.com/watch?v=OW_nQN-UQhA&ab_channel=MaratonaUFMG