DCC 012 – Estrutura de Dados II – Trabalho 2 - 2019.3
Prof. Bárbara de Melo Quintela

Busca e Compressão de Informações sobre Jogos de Tabuleiro

Na primeira parte do trabalho, foi implementada uma solução para exibir as *categorias mais recorrentes* e *os usuários mais ativos* do `BoardGameGeek`. Porém, necessita-se agora de uma funcionalidade para buscar uma determinada avaliação em seu conjunto de dados e realizar a compressão da descrição do jogo de tabuleiro associado. Uma vez que são publicadas milhares de avaliações por dia no mundo todo, a sua solução deve ter um bom desempenho. Para que isso seja garantido, seu grupo deverá analisar o desempenho de algoritmos de busca em estruturas balanceadas conforme descrito a seguir. Esta análise consiste em comparar os algoritmos considerando três métricas de desempenho: número de comparações de chaves, número de cópias de registros realizadas, e tempo total gasto para busca (tempo de processamento e não o tempo de relógio). Além disso, você deverá avaliar o desempenho de algoritmos de compressão para as descrições dos jogos associados às avaliações. Para este caso, as métricas de desempenho são: taxa de compressão, armazenamento em disco e tempo de execução (tempo de processamento),

Você deverá utilizar um conjunto real de registros para os experimentos. No link `https://www.kaggle.com/jvanelteren/boardgamegeek-reviews#bgg-13m-reviews.csv`, há 13 milhões de avaliações de jogos até Maio de 2019, armazenadas em um arquivo CSV: "`bgg-13m-reviews.csv`", contendo identificação do jogo (ID), nome do usuário (user), avaliação (rating - variando de 0 a 10). O arquivo `bgg-13m-reviews.csv` se relaciona com os demais arquivos da base de dados pelos campos identificadores. Por exemplo, o campo `ID` pode ser usado para obter as informações sobre o jogo no arquivo `games_detailed_info.csv`, que contém a descrição (definida no campo `description`) que será utilizada para a compressão.

1 – Análise dos Algoritmos Utilizando Memória Interna

Você deverá avaliar o desempenho de estruturas balanceadas ao inserir uma avaliação usando como chave a combinação de `ID` e `user` (pode aproveitar a combinação que foi realizada na parte 1 para armazenar a chave como um número inteiro ou armazenar a chave como texto). Você também deverá analisar o desempenho dessas estruturas ao realizar a busca pela avaliação usando (`ID, user`). As estruturas que devem ser avaliadas são:

- Árvore Vermelho-Preto
- Árvore B (d = 2)
- Árvore B (d = 20)

Considere que a ordem d da árvore B representa o número mínimo de chaves em cada nó (exceto a raiz), conforme visto em sala de aula. Não há necessidade de se manter preso aos valores de d especificados acima, o grupo pode adotar valores diferentes (ou

mais que dois valores) para os testes. Tenha o cuidado, no entanto, de escolher valores de d que permitam avaliar a diferença no desempenho para árvores de ordem baixa e árvores de ordem alta. Você ainda deverá implementar funções/métodos para importar os conjuntos de elementos aleatórios. Estes métodos/funções devem ser chamados uma vez para cada um dos N elementos a serem ordenados.

Análise

Os algoritmos deverão ser aplicados a entradas com diferentes tamanhos (parâmetro N). Para cada valor de N, você deve gerar 5 (cinco) conjuntos de elementos diferentes, utilizando sementes diferentes para o gerador de números aleatórios. Você pode gerar um número aleatório com valores entre 1 e o número de linhas do seu arquivo de dados e importar o dado correspondente ao número da linha gerado. Experimente, no mínimo, com valores de N = 1000, 5000, 10000, 50000, 100000 e 500000. Os algoritmos serão avaliados comparando os valores médios das 5 execuções para cada valor de N testado.
O seu programa principal deve receber um arquivo de entrada com o seguinte formato:

6 → número de valores de N que se seguem, um por linha
1000
5000
10000
50000
100000
500000

Para cada valor de N lido do arquivo de entrada:

- Gera cada um dos conjuntos de elementos, constrói a árvore, contabiliza estatísticas de desempenho para inserção na árvore analisada
- Armazena estatísticas de desempenho em arquivo de saída (`saidaInsercao.txt`).
- Realiza a busca na árvore gerada pelas entradas da inserção, contabiliza estatísticas de desempenho para busca na árvore analisada.
- Armazena estatísticas de desempenho em arquivo de saída (`saidaBusca.txt`).

Ao final, basta processar os arquivos de saída referentes a cada uma das sementes, calculando as médias de cada estatística, para cada valor de N e para cada estrutura de dados considerados.

Resultados

Apresente gráficos e tabelas para as três métricas pedidas, número de comparações, número de cópias e tempo de execução (tempo de processamento), comparando o desempenho das árvores e diferentes valores de N. Discuta seus resultados. Quais são os compromissos de desempenho observados?
Qual o impacto das variações nos valores da ordem para as Árvores B?

DCC 012 – Estrutura de Dados II – Trabalho 2 - 2019.3
Prof. Bárbara de Melo Quintela

2 – Análise dos Algoritmos de Huffman e LZW para compressão

Você deverá avaliar o desempenho e a taxa de compressão ao comprimir o corpo de uma descrição de jogo utilizando os algoritmos de Huffman e LZW.

Análise:
Os algoritmos deverão ser aplicados a entradas com diferentes tamanhos (parâmetro N). Para cada valor de N, você deve gerar 5 (cinco) conjuntos de elementos diferentes, utilizando sementes diferentes para o gerador de números aleatórios. Você pode gerar um número aleatório com valores entre 1 e o número de linhas do seu arquivo de dados e importar o dado correspondente ao número da linha gerado. Experimente, no mínimo, com valores de N = 1000, 5000, 10000, 50000, 100000 e 50000. Os algoritmos serão avaliados comparando os valores médios das 5 execuções para cada valor de N testado.
O seu programa principal deve receber um arquivo de entrada (`entrada.txt`) com o seguinte formato:

6 → número de valores de N que se seguem, um por linha
1000
5000
10000
50000
100000
500000

Para cada valor de N, lido do arquivo de entrada `entrada.txt`:

- Gera cada um dos conjuntos de elementos e salva apenas esse conjunto em disco.
- Comprime cada descrição e contabiliza estatísticas de desempenho para o algoritmo analisado.
- Salva o conjunto de descrições comprimidas em disco.
- Armazena estatísticas de desempenho em arquivo de saída (`saida.txt`)

Ao final, basta processar os arquivos de saída referentes a cada uma das sementes, calculando as médias de cada estatística, para cada valor de N e para cada algoritmo considerado.

Resultados
Apresente gráficos e tabelas para as três métricas pedidas, taxa de compressão, armazenamento em disco e tempo de execução (tempo de processamento), comparando o desempenho dos algoritmos e diferentes valores de N. Discuta seus resultados. Quais são os compromissos de desempenho observados?

Considerações

DCC 012 – Estrutura de Dados II – Trabalho 2 - 2019.3
Prof. Bárbara de Melo Quintela

1) Todo código fonte deve ser documentado. A documentação inclui, dentre outros, a documentação de procedimentos, de funções, de variáveis, de partes do código fonte que realizam tarefas específicas. Ou seja, o código fonte deve ser documentado tanto em nível de rotinas quanto em nível de variáveis e blocos funcionais.
2) A interface pode ser feita em modo texto (terminal) ou modo gráfico e deve ser funcional.
3) A implementação deve ser realizada usando a linguagem de programação C, C++ ou Java.

Entrega

O grupo deverá ser formado por até 4 alunos e as responsabilidades de cada aluno deve ser documentada e registrada. O prazo final para entrega do código fonte é dia **27/11 junto com o** relatório. Deverá ser agendada uma data para apresentação do trabalho para os professores **até o dia 27/11**.

Deve ser entregue os códigos implementados e um relatório com os seguintes itens:
1) Descrição das atividades realizadas por cada membro do grupo
2) Análises e explicação sobre as estruturas de dados implementadas

Critérios de avaliação

Você não fechará o trabalho só tendo um "sistema que funciona". O sistema deve funcionar bem e o quão bem ele funcionar será refletido na sua nota. A nota poderá ser comparativa, então se esforce para ter uma solução melhor que a dos outros colegas. O objetivo do trabalho é testar a sua capacidade de fazer boas escolhas (e boas adaptações) de estruturas para resolver problemas. **Então usar classes prontas ou métodos prontos não são permitidos aqui.** Você poderá, se quiser, comparar sua solução com outras prontas. Mas deve perseguir o seu melhor sem usar recursos de terceiros.

Os membros da equipe serão avaliados pelo produto final do trabalho e pelos resultados individuais alcançados. Assim, numa mesma equipe, um membro pode ficar com nota 90 e outro com nota 50, por exemplo. Dentre os pontos que serão avaliados, estão:

- Execução do programa (caso o programa não funcione, a nota será zero)
- Código documentado e boa prática de programação (o mínimo necessário de variáveis globais, variáveis e funções com nomes de fácil compreensão, soluções elegantes de programação, código bem modularizado, etc)
- Testes: procure fazer testes relevantes como, por exemplo, aqueles que verificam casos extremos e casos de exceções
- Relatório bem redigido

Note que o grande desafio deste trabalho está na avaliação dos vários algoritmos nos diferentes cenários, e não na implementação de código. Logo, na divisão de pontos, a documentação receberá, no mínimo, 50% dos pontos totais.

Uma boa documentação deverá apresentar não somente resultados brutos mas também uma discussão dos mesmos, levando a conclusões sobre a superioridade de um ou outro algoritmo, para cada métrica avaliada.