Universidade Federal de Roraima - UFRR

Docente: Hebert Oliveira Rocha

Discentes: Glisbel, Thiago Vieira e Shelly Leal

Disciplina: Sistemas Operacionais I 2023.1

Data de Entrega: 17 de Abril de 2023

Documentação do Trabalho (Uso do Grupo para a Apresentação)

Tema 07: Busca em Profundidade usando Multithreading

**Roteiro de Apresentação**

**Tempo de Apresentação - 08 MINUTOS**

Ínicio - PARTE 1 - Apresentação do grupo (Shelly)

* Apresentar o tema do trabalho
* Apresentar os integrantes do grupo
* Definir “Quais são os Algoritmos de Busca ou Varredura?”

Os algoritmos de busca ou varredura são algoritmos que permitem a busca de informações em uma estrutura de dados, como uma lista, árvore, grafo ou matriz, em busca de um valor específico. Alguns dos algoritmos de busca mais comuns incluem:

…

Existem muitos outros algoritmos de busca e varredura que são usados em diferentes situações e estruturas de dados. Cada algoritmo tem suas próprias vantagens e desvantagens, e a escolha do algoritmo certo depende do problema em questão.

Ínicio - PARTE 2 - Definição do Tema (Glisbel)

* O que é Busca em Profundidade? ESTUDAR O SITE DE REFERÊNCIA

<https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/dfs.html>

* Onde usamos? (TRAZER EXEMPLOS)
* Uso das Threads na DFS (Depth-First Search) (LEIA OS SLIDES BASE DO TRABALHO)

(Slide 5) - Explicação

O padrão de design simultâneo que será implementado, mostra o benefício de ter vários núcleos no processador. O programa não apenas pode processar outros nós enquanto um está dormindo, mas também pode processar vários nós praticamente ao mesmo tempo.

* (Slide 7) - Explicação
* - Threads Encadeada: As funções são executadas em série de maneira pré-ordenada. como funciona de forma sequencial, é necessário esperar o término de uma função para a outra iniciar.
* Threads Simultânea: As funções estão sendo executadas quase independentemente e praticamente de forma paralela, tendo em vista que ainda há um pequeno atraso para o início de cada função.

**Codificação e Implementação - PARTE 3 - Aplicação (Shelly) (Thiago Vieira)**

* Evidencie o código DFS usando apenas um Thread (Single Thread) (Arquivo .c) **(Shelly)**

**Explicação**

O programa começa definindo algumas constantes e bibliotecas necessárias, incluindo a biblioteca de pilha personalizada "stack.h". Em seguida, o programa cria um grafo aleatório não direcionado, com o número de nós definido pela constante NUM\_NODES.

Depois que o grafo é criado, o programa exibe a matriz de adjacência do grafo para fins de depuração.

Em seguida, o programa começa a busca em profundidade com um único thread na função principal main(). A função DFS() é chamada com um parâmetro inicial (nó inicial) e começa a explorar o grafo usando o algoritmo DFS.

A função DFS() usa uma pilha para implementar o algoritmo DFS. Ela inicializa uma pilha, adiciona o nó inicial na pilha e, em seguida, executa o loop principal da busca em profundidade. O loop continua até que a pilha esteja vazia. Em cada iteração do loop, um nó é removido da pilha e é verificado se ele já foi visitado ou não. Se ele ainda não tiver sido visitado, é marcado como visitado e imprimido na tela em vermelho.

Em seguida, o algoritmo itera sobre os vizinhos do nó visitado e os adiciona à pilha, se ainda não foram visitados.

Quando a busca em profundidade termina, o programa exibe o tempo total decorrido pela execução do algoritmo e encerra.

Em resumo, o programa gera um grafo aleatório não direcionado e executa a busca em profundidade nesse grafo usando um único thread, usando uma pilha para rastrear os nós visitados.

* Evidencie o Codificação do DFS com Várias Threads (Multithreading) **(Thiago Vieira)**

Explicação:

O programa começa lendo o número de nós e arestas do grafo e o número de threads que serão usadas na busca em profundidade. Em seguida, lê a matriz de adjacência que representa o grafo e o nó de início da busca.

Cada thread executa a função dfs, que realizará a busca em profundidade. A thread continuará executando enquanto houver nós na pilha a serem visitados. A pilha é protegida por um mutex, para garantir que apenas uma thread acessa a pilha de cada vez.

Cada vez que uma thread retira um nó da pilha, verifica se ele já foi visitado. Se não foi, a thread o marca como visitado, imprime uma mensagem indicando que o visitou e adiciona todos os seus vizinhos não visitados na pilha.

No final, o programa espera todas as threads terminarem com a função pthread\_join.

Observe que, para permitir que cada thread saiba qual é seu ID, é criado um ponteiro thread\_id e passado como argumento para cada thread, que deve ser liberado com free dentro da função dfs.

* Explicar o Uso do Código - COLOQUE TÓPICOS IMPORTANTES PARA EXPLICAÇÃO
* Usa Semáforo (Binário - 0 e 1) (Explicar a razão de uso)

O que é o Mutex?

O Mutex (abreviação de "mutual exclusion") é um objeto de sincronização usado para garantir a exclusão mútua no acesso a recursos compartilhados em um ambiente multi-thread. Em outras palavras, o Mutex é uma ferramenta para evitar que duas ou mais threads acessem o mesmo recurso simultaneamente, o que pode levar a resultados indesejados ou imprevisíveis.

O Mutex funciona como uma chave, onde uma thread que deseja acessar um recurso compartilhado deve primeiro adquirir o Mutex, antes de acessar o recurso. Enquanto a thread estiver usando o recurso, o Mutex permanece bloqueado, impedindo que outras threads o acessem. Quando a thread terminar de usar o recurso, ela deve liberar o Mutex, permitindo que outras threads o acessem.

O Mutex é implementado usando um semáforo binário, que é um semáforo que só pode ter dois valores: 0 e 1. Quando o Mutex está bloqueado, o semáforo é 0 e nenhuma outra thread pode adquirir o Mutex. Quando o Mutex está disponível, o semáforo é 1 e uma thread pode adquiri-lo. As threads que tentam adquirir o Mutex enquanto ele estiver bloqueado ficam em espera até que o Mutex seja liberado.

* Evidencie o código DFS com Multithreading com Semáforo (Arquivo .c) (Thiago Vieira)

**Explicação**

Este código implementa um algoritmo de busca em profundidade (DFS) em um grafo usando programação multithread. O grafo é gerado aleatoriamente e armazenado em uma matriz de adjacência. O número de nós do grafo é definido pela constante NUM\_NODES.

A implementação usa uma pilha compartilhada (globalStack) e várias threads (THREAD\_COUNT) para executar a busca em profundidade. Cada thread executa o algoritmo DFS em paralelo, processando os nós que foram desempilhados da pilha global.

O algoritmo DFS usa semáforos para controlar o acesso concorrente à pilha global e às variáveis de controle, como visited e parent. Além disso, a implementação usa semáforos para garantir que a saída impressa na tela seja sincronizada corretamente.

Ao finalizar a busca, o tempo de execução é medido e impresso na tela. A implementação também limpa a memória alocada para a pilha global e os semáforos, além de terminar as threads criadas.

aleatoriamente e pode conter até 800 nós. O programa cria um stack global e inicialmente coloca o nó 0 nele. Em seguida, cria 4 threads que executam a função DFS de forma concorrente, removendo nós do stack global até que este esteja vazio. Cada thread utiliza um mutex para acessar o stack global e imprimir na tela os nós visitados. O programa também utiliza semáforos para garantir que cada thread processe nós diferentes e para evitar problemas de concorrência.

O programa inicia medindo o tempo de execução da DFS multithreaded e, ao final, libera o stack global e o mutex utilizado, finalizando o programa.

Este é um programa C que executa um algoritmo Depth-First Search (DFS) em um gráfico gerado aleatoriamente usando multithreading. O programa cria um gráfico com um tamanho de 800 nós e, em seguida, executa o DFS no gráfico usando quatro threads. A função DFS usa uma estrutura de dados de pilha para acompanhar os nós que estão sendo visitados. O programa usa as seguintes bibliotecas: time.h, stdio.h, stdlib.h, pthread.he um arquivo de cabeçalho customizado stack.h. O programa define uma estrutura chamada Thread que possui dois campos: number e tid (ID do thread). Ele também declara uma matriz de estruturas Thread chamadas threads com um tamanho de THREAD\_COUNT (que é definido como 4). O programa define uma variável pthread\_mutex\_t chamada mutex para ser usada para exclusão mútua ao acessar variáveis compartilhadas. **O programa define algumas variáveis globais a serem usadas pela função DFS: parent é uma matriz de valores longos que armazena o pai de cada nó visitado na árvore DFS. visited é uma matriz de valores longos que rastreia os nós visitados.**

GRAFO é uma matriz bidimensional de valores longos que representa o gráfico. O programa também define algumas funções: createGraph() gera um gráfico aleatório de tamanho NUM\_NODES e preenche a matriz do gráfico com seus valores. printGraph() imprime o gráfico gerado. DFS() executa o algoritmo DFS usando multithreading. main() é a função principal do programa que inicializa as variáveis globais, cria os threads e chama a função DFS(). O programa usa uma estrutura de dados de pilha para implementar o algoritmo DFS.

A pilha é implementada usando um arquivo de cabeçalho personalizado stack.h que define uma estrutura Stack e funções para criar, empurrar, remover e destruir uma pilha.

Na função DFS(), cada thread pega um nó da pilha e o processa. Se o nó não foi visitado antes, ele o marca como visitado e define seu pai. Em seguida, ele verifica todos os nós vizinhos e os coloca na pilha, caso ainda não tenham sido visitados. Esse processo continua até que a pilha esteja vazia. O programa usa exclusão mútua para proteger as variáveis compartilhadas de condições de corrida. As funções pthread\_mutex\_lock() e pthread\_mutex\_unlock() são usadas para bloquear e desbloquear a variável mutex antes e depois de acessar variáveis compartilhadas. No final do programa, a pilha e a variável mutex são destruídas usando as funções destroyStack() e pthread\_mutex\_destroy(), respectivamente. A função pthread\_exit() é chamada para finalizar todas as threads.

* Evidenciar TESTES - DFS em Multithreading em funcionalidade (Shelly)

**Explicação complementar**

Ele cria um certo número de threads (THREAD\_COUNT) e cada uma delas executa a função DFS simultaneamente. As threads compartilham uma pilha global (globalStack) e algumas variáveis (parent, visited e graph) que são acessadas simultaneamente pelas threads.

A função DFS é a função que cada thread executa, e ela implementa o algoritmo Depth-First Search (DFS) em um grafo. Cada thread pega um nó da pilha global (globalStack), explora todos os seus filhos e adiciona todos os filhos inexplorados na pilha global. Isso continua até que todos os nós tenham sido visitados. As threads se comunicam entre si usando semáforos (mutex e see\_mutex) para evitar condições de corrida.

Conclusão - PARTE FINAL - Considerações Finais (Shelly)

* Evidenciar referências (De onde foi pesquisado, textos, artigos ou trabalhos anteriores)
* Evidenciar dificuldades (Entender a implementação do Código)
* Ressaltar a explicação (O que é DFS de fato?)
* Complementar assunto se houver necessidade (Em aberto)

Outras ANOTAÇÕES sobre o Código DFS:

Anotações do Trabalho - Busca em Profundidade usando Multithreading

Primeiramente, criamos uma pilha Dinâmica, pois podemos definir a quantidade de NODES que serão implementados.

Depois, é criado um STRUCT para cada Thread

Biblioteca do tipo pthread.h (Guarda os números) (Biblioteca específica do LINUX) (Cria as Threads)

LONG = Abreviação de Unsigned Long que equivale a 32 bits (Tipo INT)

Para COMPILAR O Algoritmo é necessário que usemos o Pront de Comando do LINUX (WSL)

Para termos uma INTERFACE para o nossa PILHA foi criado um arquivo STACK.c (Ao compilar, devemos passar os DOIS CÓDIGOS ao mesmo tempo.)

Variável do CLOCK -> Evidenciar o tempo de execução do GRAFO

Os 0 e 1 ao compilar são o resultado do GRAFO criado.

0 = NÃO TEM LIGAÇÃO

1 = TEM LIGAÇÃO

Ao compilar, os números Coloridos -> É o Percurso do GRAFO

VOID = Semente Aleatória

Para criar uma MATRIZ é preciso usar um FOR dentro de um FOR

// Código do DFS - Comentários

Primeiramnente, o que é?

Existem vários Algoritmos de BUSCA, o que iremos falar é sobre o:

DFS (Deep First Search) - Busca em Profundidade (Usa Pilha) LIFO

BFS (*breadth-first search*) - Busca em Largura (Usa Fila) FIFO

Usa DUAS PILHAS PRINCIPAIS:

LOCAL - Threads percorridas, as que não foram acessadas são mandadas para Pilha GLOBAL

GLOBAL (Região Compartilhada = REGIÃO CRÍTICA) (Todos os Nodes são definidos em GLOBAL) - Recebe as NODES que não foram acessadas. No momento em que essas NODES têm acesso a essa Pilha Global, os threads assumem seu Nó (Como Pai daquele Node) a Thread e dão continuidades a elas, até que todos os NODES sejam percorridos.

Vetor - VARIÁVEL -> VISITOR 'S = Para dizer se o NODE foi VISITADO OU NÃO, ou seja, é uma VARIÁVEL para marcar qual NODE foi visitado e não repetir a saída do NODE.

Vetor -> PARENT =

Como é definido o NODES no GLOBAL, seu NÓ RAIZ inicia sempre na GLOBAL

Na leitura do código, ao percorrer o GRAFO, o que fica evidente é o próximo NODE que ele irá apontar.

// Na apresentação, podemos fazer o exemplo do GRAFO ao VIVO para ajudar na Lógica dos GRAFOS

GRAFOS ONLINE - site

// As threads aguardam que apareça alguma coisa na Pilha Global para percorrer

// As Threads querem acessar quase que concorrentemente a Região Crítica (Pilha Global), com isso usamos o MUTEX (Exclusão Mútua) para que as Threads não acessem ao mesmo tempo.

- Usando 0 e 1, Bloqueia e Desbloqueia, Libera ou Prende. (Semaphore Binary) 0 e 1

/\* Código faz

Ver a LIGAÇÃO ENTRE DOIS NODES.

// Apresentação

Comentar sobre as dificuldades encontradas para entender a Lógica do Algoritmo.