Disciplina: INE5609 - Estruturas de Dados
Professor: José Eduardo de Lucca
Aluno: João Paulo Decker Oleinik
Matrícula: 24202528

Perguntas referentes a implementação da “Lista Encadeada Estendida”

1. Essa solução apresenta desempenho geral melhor do que uma lista encadeada comum? Por quê?

Não se pode afirmar que o desempenho geral (em termos de complexidade de tempo Big-O) é melhor, mas sim que ela possui vantagens de desempenho em cenários práticos, especialmente em relação ao uso de memória e cache da CPU.

Busca e Acesso por Posição (O(n)):

Lista Comum: Para acessar a k-ésima posição, são necessários k saltos de ponteiros.

Lista Estendida: Para acessar a k-ésima posição, é preciso percorrer a lista de nós, somando suas contagens internas. No pior caso, o que também é O(n).

Vantagem Prática: A Lista Estendida faz n/8 saltos de ponteiros, em vez de n. Menos saltos de ponteiros significam melhor localidade de cache. Acessar 8 elementos contíguos em um array é muito mais rápido para uma CPU moderna do que acessar 8 elementos espalhados pela memória.

Inserção e Exclusão (O(n)):

Lista Comum: Encontrar o local da inserção/exclusão é O(n). A operação em si (ajuste de ponteiro) é O(1).

Lista Estendida: Encontrar o nó é O(n). A operação de inserir/excluir dentro do array do nó tem um custo de deslocamento de elementos, que é O(k) (onde k é a capacidade do nó, 8). Se o nó estiver cheio, ocorre um "split" (divisão), que também custa O(k). Como k é uma constante pequena, o custo total continua sendo dominado pelo O(n) da busca.

Portanto, a complexidade assintótica (Big-O) para as operações principais permanece O(n), assim como na lista comum. No entanto, a Lista Estendida é praticamente mais rápida para buscas e travessias devido à drástica redução de saltos de ponteiros e melhor uso do cache.

---

2. Uma vantagem dessa estrutura (em relação às listas encadeadas simples) seria economizar memória, uma vez que com essa nova lista só teremos um ponteiro para o próximo a cada 8 elementos (e não a cada 1 elemento). Então, pode-se imaginar que aumentar o tamanho do array (para 20, 30, 50, ou 1000) ajudaria a melhorar ainda mais essa relação (que no exemplo é de 1 para 8). Mas pode haver problemas em aumentar indefinidamente esse tamanho, afetando as operações. Discuta.

A afirmação de que aumentar o tamanho do array melhora a economia de memória está correta. A proporção ponteiros/dados diminui. No entanto, aumentar esse tamanho indefinidamente cria um novo gargalo de desempenho que afeta as operações de modificação.O problema é o custo de manutenção dos arrays internos:

Custo de Inserção/Exclusão: As operações InsereOrdenado e ExcluiDaPosição dentro de um nó exigem o deslocamento de elementos para manter o array ordenado. Esse custo é proporcional ao tamanho do array (O(k)). Se k=8, mover 7 elementos é trivial. Já se k=1000, inserir um elemento na primeira posição exigiria mover 999 elementos, o que é extremamente custoso.

Custo do "Split": A operação de divisão, que ocorre quando um nó cheio precisa receber um elemento, envolve copiar metade dos dados (k/2) para um novo nó. Se k=1000, um split exigiria a cópia de 500 elementos.

Conclusão: Existe um ponto de equilíbrio. Um k muito grande economiza memória, mas torna as operações de inserção, exclusão e divisão proibitivamente lentas. A estrutura perde sua flexibilidade e começa a se comportar como um único grande array, que tem péssimo desempenho para inserções no meio.

---

3. Vimos que em arrays ordenados, a operação de Busca pode ser feita de mais de uma forma: sequencial (argh) ou busca binária. E comente se, por ser uma estrutura ‘híbrida’ (lista encadeada de pequenos arrays), faz sentido usar busca binária ou teríamos que nos contentar com busca sequencial…

Sim, faz total sentido e é uma das principais vantagens da estrutura. A busca não precisa se contentar em ser puramente sequencial. Como cada nó individualmente é um array ordenado, podemos aplicar a busca binária dentro dele. A operação de Busca(valor) ou PosiçãoDe(valor) torna-se híbrida:

Busca Sequencial (entre Nós): O algoritmo percorre a lista de nós (Nó 0, Nó 1, ...).

Otimização: Em cada nó, ele verifica apenas o último elemento. Se valor for maior que o último elemento do nó, o algoritmo pula para o próximo nó.

Busca Binária (dentro do Nó): Assim que o algoritmo encontra um nó onde valor é menor ou igual ao seu último elemento, ele sabe que o valor (se existir) só pode estar naquele nó. Nesse ponto, ele executa uma busca binária (O(log k)) dentro do array daquele nó.

Logo, a estrutura combina a flexibilidade da lista encadeada (para adicionar nós) com a eficiência de busca do array ordenado (busca binária).

---

4. Se, em vez de a lista ser encadeada só num sentido, ela for duplamente encadeada, o desempenho geral ficaria melhor? Por quê? Se quiser implementar dessa forma, pode fazê-lo.

Para as operações especificadas no trabalho (InsereOrdenado, AcessaPosição, Busca, ExcluiDaPosição), o desempenho geral (em Big-O) não ficaria melhor.

Busca e Acesso: Operações como Busca e AcessaPosição exigem uma travessia a partir do início. Um ponteiro "anterior" não oferece nenhuma vantagem para isso.

Inserção: A InsereOrdenado também precisa encontrar o ponto de inserção a partir do início. O custo O(n) da busca permanece.

Exclusão: A operação ExcluiDaPosição é a única que vê um benefício, mas é de implementação, não de desempenho. No código, precisa-se manter uma variável ant (anterior) para "remendar" a lista ao remover um nó vazio. Se fosse duplamente encadeada, o próprio nó a ser removido saberia quem é seu anterior, simplificando a lógica de remoção.

O desempenho O(n) das operações dominantes (busca e acesso) não muda. A lista duplamente encadeada só traria vantagens se precisássemos de operações que percorressem a lista de trás para frente (ex: AcessaUltimaPosicao) ou se a exclusão fosse feita com base no ponteiro do nó (e não na posição).

---

5. Falei de Árvore B no texto. Qual é a relação que existe entre a estrutura Árvore B e essa Lista Encadeada Estendida?

Esta Lista Encadeada Estendida é, essencialmente, uma implementação da camada de folhas (o nível mais baixo) de uma Árvore-B+.As Árvores-B foram mencionadas exatamente porque elas usam os mesmos conceitos:

Nós com Múltiplas Chaves: A ideia central de uma Árvore-B não é ter um nó com 1 dado, mas sim um "bloco" ou "página" que armazena k dados de forma ordenada. O nosso No com seu array interno é exatamente isso.

Política de Divisão (Split): A regra de inserção da Árvore-B é: se um nó está cheio, ele é dividido ao meio, e a chave mediana é "promovida" para o nó pai. A nossa lista faz a primeira parte disso: o nó se divide ao ficar cheio.

Balanceamento: O propósito desse split é "manter um certo 'balanceamento' entre os nós, abrindo espaço para futuras inclusões". Este é o exato mecanismo que as Árvores-B usam para se manterem balanceadas.

Ponteiros Sequenciais (em Árvores-B+): Em uma variante específica, a Árvore-B+, todos os dados residem apenas nas folhas, e as folhas são ligadas umas às outras por ponteiros "próximo" (exatamente como o prox da nossa lista). Isso permite percorrer todos os dados em ordem sem ter que subir e descer pela árvore.

Conclusão: A nossa estrutura pode ser vista como a camada 0 (nível de dados) de uma Árvore-B+, aplicando a mesma lógica de nós baseados em arrays e "split" para garantir o balanceamento.