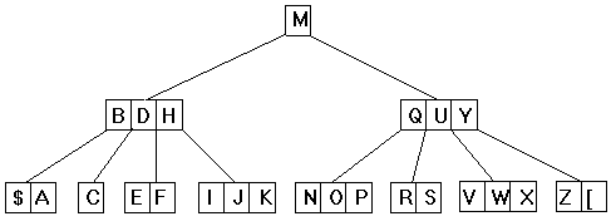
B-Tree

1. Explique a seguinte sentença: "B-Trees são construídas de baixo para cima, enquanto árvores binárias são construídas de cima para baixo".
Quando se insere uma chave numa árvore B, ela é colocada sempre numa folha e por meio de split e promote, a árvore fica sempre balanceada. Numa árvore binária, pode ser que, ao inserir uma chave, a árvore não fique balanceada. Então será necessário fazer algoritmos para fazer as operações que garantam que a árvore fique sempre balanceada. Os nós podem ser inseridos em qualquer posição.
2. Por que B-Trees são consideradas geralmente superiores que as árvores binárias de busca para pesquisa externa, e árvores binárias são comumente usadas para pesquisa interna?
Através da B-Tree, faz-se a pesquisa externa das páginas para se encontrar o possível lugar da chave desejada. Esta pesquisa é muito mais rápida que a binária. Entretanto, dentro de uma página, utiliza-se a pesquisa binária por não ser possível B-Tree e por ser mais rápida que pesquisa sequencial. Pesquisa binária em disco é muito lenta.
3. Como uma folha de uma árvore B difere de um nó interno? Quais são as partes necessárias a uma folha?
Nó folha não possui filhos, seus ponteiros são nulos. Uma B-Tree de ordem n possui n-1 chaves no máximo e deve possuir n/2-1 chaves no mínimo. Os nós folhas são aqueles alocados no nível mais baixo da árvore. Todas as folhas aparecem num mesmo nível.

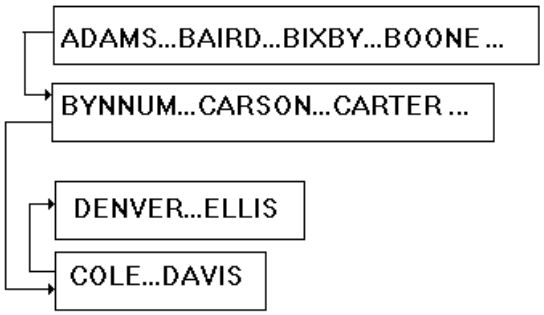
5. Dada a árvore B que contém todas as letras do alfabeto, mostre o que acontece com a árvore com a inserção das chaves **$** (menor que A) e a seguir, da chave **[** (maior que Z).



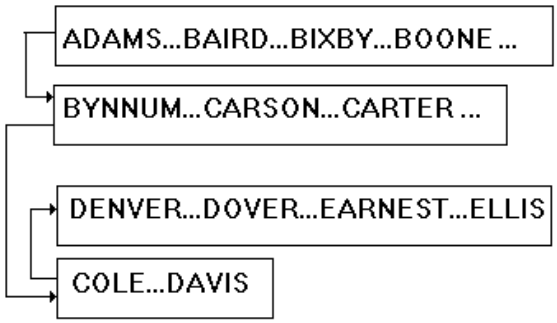
6. Dada uma árvore B de ordem 256, qual o número máximo de descendentes por página? Qual o número mínimo (desconsideradas as folhas e a raiz? E de uma folha? Quantas chaves tem uma página não-folha com 200 descendentes?
máximo de descendentes = 256
mínimo de descendentes = 128 (128=m/2) desconsiderando as folhas e a raiz
raiz tem no mínimo 2 descendentes
folha não tem descendentes
199 chaves
7. Suponha que você vai deletar uma chave em uma árvore B, a qual causa um underflow na página. Se pela página irmã do lado direito é necessária concatenação, e pela página esquerda é possível redistribuição, qual opção você escolheria? Por quê?
Redistribuição, porque se fosse utilizada a concatenação, poderia haver underflow da página pai e teria que ser feita outra concatenação. Com redistribuição isso não ocorre, pois há apenas uma troca de uma folha para outra.
8. Qual a diferença entre uma árvore B e uma B*? Quais as vantagens da árvore B*? Quais as desvantagens? Como se comparam a altura mínima dessas árvores?
Uma árvore B utiliza two-to-three spliting, ou seja, o processo de divisão é adiado até que duas páginas irmãs estejam cheias. Realiza-se então a divisão do conteúdo das duas páginas em 3. Em árvore B, utiliza-se one-to-two spliting onde uma folha cheia é dividida em duas.*
***Vantagem:** cada página tem no mínimo 2/3 das chaves (menos a fragmentação interna).*
***Desvantagem:** deve-se tomar um cuidado especial com o nó raiz e para ele usar one-to-two spliting (algoritmos mais complexos).*
Como B tem no mínimo 2/3 das chaves, ela vai ter uma altura mínima menor que a árvore B.*
9. O que é uma árvore B virtual? Como implementá-la? Quais as vantagens e desvantagens?
Quando se coloca parte da árvore B na memória. Coloca-se a raiz e alguns de seus descendentes na memória, deixando algum espaço para manipular as páginas.
Vantagens: Quando se procura uma chave, pode ser que ela já esteja na memória o que diminui em um acesso ao disco.
Desvantagens: Pode-se deixar na memória páginas que não estão sendo usadas nunca. É necessário substituí-las.

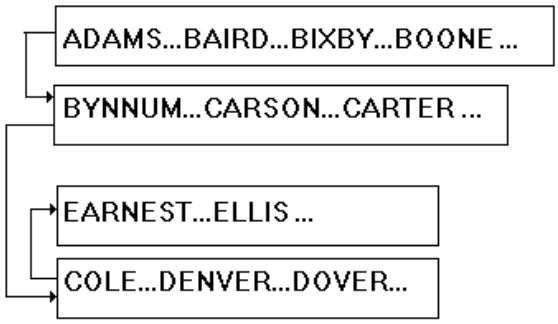
Arvore B+

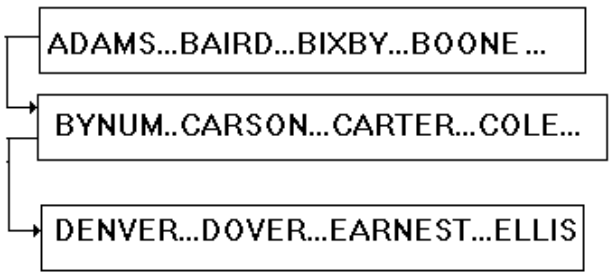
1. O que é uma árvore B+? Quando ela é necessária?
É uma estrutura híbrida formada pelo uso combinado de Sequence Set e B-Tree. Ela é usada quando um mesmo arquivo puder ser pesquisado sequencialmente e por índice. Para pesquisar sequencialmente usa-se o sequence set e por índice usa-se a árvore B que conterá nas suas chaves os índices do arquivo.
2. Dada a sequência da figura abaixo



mostre o que ocorre com a adição das chaves DOVER e EARNEST; e então quando DAVIS é eliminado. Você usou concatenação ou redistribuição?







3. Que condições afetam a escolha do tamanho do bloco usado no sequence set?

- *blocos podem acessar grandes quantidades de informação sequencialmente*
- *precisamos ser capazes de manter vários blocos na memória*
- *não queremos gastar um seek durante a leitura de um único bloco*
- *tamanho de memória disponível*
- *tamanho de um cluster*
- *tamanho de uma trilha*

4. Como poderia-se obter indexed sequenced file sem usar uma árvore?

Utilizando-se um índice simples para os blocos, o que permite pesquisa binária. Índice precisa ser mantido corretamente conforme os blocos são atualizados. Fácil e cabe em memória.

5. Por que os separadores do índice em uma árvore B+ não precisam ser as chaves?

Podem ser simplesmente o início das palavras com duas ou três letras. Assim, na pesquisa compara-se o início da palavra procurada com um índice e este indicará para qual lado a palavra deve estar.

HASHING

5. Como um algoritmo hashing poderia tirar vantagens da existência de conjuntos de chaves ordenados?
As chaves seriam criadas e colocadas no arquivo de chaves onde ordenadamente com apenas um seeking. Os endereços serão sequenciais.
6. Explique as técnicas de tratamento de overflow por lista encadeada.
Os sinônimos são encadeados na própria área de endereçamento.
Vantagens: *só sinônimos são acessados em uma busca*
Desvantagens: *necessário garantir que sinônimos sejam encontrados.*
Problema: *tratamento de casas de chaves que não podem ocupar seu home address pois este já é ocupado por outra lista de chaves. Uma solução: two-pass loading.*
Outra opção: *alocar os sinônimos numa área especial de overflow. Neste caso os endereços sem chaves na área primária nunca serão ocupados.*
Vantagens: *resolve problemas de manutenção, área de overflow pode ser simples sequência de chaves, mais rápido que o reajuste depois de cada eliminação.*
Desvantagem: *overflow sempre requer outro acesso*

1. Definição e Propriedade de B-trees

Ordem

A definição atual de B-Tree vincula a ordem de uma árvore B ao número de descententes de um nó (isto é,. de ponteiros). Deste modo, numa árvore B de ordem m o número máximo de chaves é m-1.

Exemplo:

Uma árvore B de ordem 8 tem um máximo de 7 chaves por página.

Número mínimo de chaves por página

Quando uma página é dividida na inserção, os nós são divididos igualmente entre as páginas velha e nova. Deste modo, o número mínimo de chaves em um nó é dado por m/2 - 1 (exceto para a raiz).
Exemplo: Uma árvore B de ordem 8 que tem um máximo de 7 chaves por página tem um mínimo de 3 chaves por página.
Nó folha
Os nós folhas são aqueles alocados no nível mais baixo da árvore.

2. Definição formal das Propriedades de B-trees

Para uma B-tree de ordem m:

1. cada página tem um máximo de m descententes
2. cada página, exceto a raiz e as folhas, tem um mínimo de m/2 descendentes
3. a raiz tem um mínimo de dois descendentes - a menos que seja uma folha
4. todas as folhas aparecem num mesmo nível
5. uma página que não é folha e possui k descendentes contém k-1 chaves
6. uma página folha contém no mínimo m/2 -1 e no máximo m-1 chaves

3. Profundidade de Busca no Pior Caso

É importante entender o relacionamento entre o tamanho de página, o número de chaves armazenados em uma página, e o número de níveis que uma B-tree pode ter.

Exemplo: "Você precisa armazenar 1.000.000 chaves e considera utilizar B-tree de ordem 512. Qual o número máximo de acessos para localizar uma chave ? "
Ou seja: quão profunda pode ser a árvore ?
O pior caso ocorre quando cada página tem apenas o número mínimo de descendentes, e possuem, portanto, altura máxima e largura mínima.
Em geral, para um nível d qualquer de um B-tree, o número mínimo de descendentes é dado por (2 * m/2) ^ (d-1).
Para uma árvore de N chaves, a sua profundidade no nível das páginas folhas, é dado por d onde:

d <= 1 + log(base 256)m/2((N+1)/2)

Então para a B-tree de ordem 512 com 1.000.000 de chaves
d <= 1 + log256(500.000,5) <= 3.37 Essa é a performance que procuramos !

4) Eliminação, Redistribuição e Concatenação

O processo de divisão (split) de páginas garante a manutenção das propriedades da B-tree durante a inserção. Essas propriedades precisam ser mantidas, também, durante a eliminação de chaves. **Caso 1:** eliminação mantém número mínimo de chaves na página.

Solução: Chave é retirada e registros internos à página reorganizados.

Caso 2: eliminação de chave que não está numa folha.
Solução: Sempre eliminamos da folha. Se uma chave deve ser eliminada de uma página que não é folha, trocamos a chave com sua sucessora imediata, a qual, com certeza está numa folha, e então eliminamos a chave da folha.
Caso 3: eliminação causa underflow na página.
O número mínimo de chaves por página nessa árvore é m/2 -1 = 6/2 -1=2.
Solução: Redistribuição. Procura-se uma página irmã (mesmo pai) que contenha mais chaves que o mínimo: se existir redistribui-se as chaves entre essas páginas. A redistribuição causa a colocação de uma nova chave de separação no nó pai.
Caso 4: ocorre underflow e a redistribuição não pode ser aplicada.
Quando não existirem chaves suficientes para dividir entre as duas páginas irmãs.

Solução: Deve-se utilizar concatenação: combina-se o conteúdo das duas páginas e a chave da página pai para formar uma única página. Concatenação é o inverso do Splitting. Como consequência, pode causar o underflow da página pai.
Caso 5: underflow da página pai
Solução: No exemplo, a concatenação das páginas 3 e 4 retira D a página 1, causando underflow na página 1. Redistribuição não pode ser aplicada (por quê?). Deve-se utilizar concatenação novamente.
Caso 6: Diminuição da altura da árvore.
Consequência da concatenação dos filhos do nó.
Solução: A concatenação das páginas 1 e 2 absorve a única chave da raiz.
Este caso mostra o que ocorre quando a concatenação é propagada até a raiz. Note que esse nem sempre é o caso: se a página 2 (Q e W) tivesse mais uma chave, aplicaria-se redistribuição em vez de concatenação.

Eliminação de chave em árvore B

1. Se a chave não estiver numa folha, troque-a com seu sucessor imediato.
2. Elimine a chave da folha.
3. Se a folha continuar com o número mínimo de chaves, fim.
4. A folha tem uma chave a menos que o mínimo. Verifique as páginas irmãs da esquerda e direita:
4.1.se uma delas tiver mais que o número mínimo de chaves, aplique redistribuição.
4.2.senão concatene a página com uma das irmãs e a chave pai.
5. Se ocorreu concatenação, aplique passos de 3 a 6 para a página pai.
6. Se a última chave da raiz for removida, a altura da árvore diminui.

5) Redistribuição durante Inserção

Diferentemente da divisão e da concatenação, o efeito da redistribuiço é local. Não existe propagação.
Outra diferença: não existe regra fixa para o rearranjo das chaves e a eliminação de uma chave pode causar o underflow de apenas uma chave na página, e a redistribuição pode mover apenas uma chave para página com problema.
Exemplo: dada uma B-tree de ordem 101, o número mínimo e máximo de chaves é, respectivamente, 50 e 100. Se ocorre o underflow numa página, e a página irmã tem 100 chaves, qualquer número de chaves entre 1 e 50 pode ser transferido.
Normalmente transfere-se 25, e deixa-se as páginas equilibradas.
Redistribuição é uma idéia nova, a qual não foi explorada no algoritmo de inserção. E é uma opção desejável. Em vez de dividir uma página cheia em duas meia-páginas, pode-se optar por colocar a chave que sobra (ou mais que uma!) em outra página.
Essa estratégia deve resultar numa melhor utilização do espaço.

Redistribuição X Divisão

Depois da divisão de uma nó, cada página está 50% vazia. Portanto a utilização do espaco, no pior caso, em uma ávore B que utiliza splitting é de cerca de 50%. Em média, para árvores grandes, prova-se que o índice é de 69%.
Estudos empíricos indicam que a utilização de redistribuição pode elevar o índice de 67% para 85%. Esses resultados sugerem que qualquer aplicação séria de árvore B utilize, de fato, redistribuição durante a inserção.

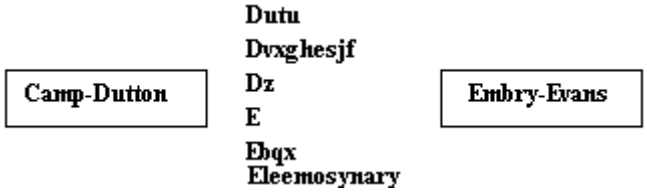
Conceitos B+

Quem pode ser separador ?

Algoritmos especiais para geração e manutenção.

```
find_sep(key1,key2,sep)
char key1[], char key2[], sep[];
{
  while ((*sep++ = *key2++) == *key1++)
    *sep='\0';
}
```

Uma lista de possíveis separadores:



2. Simple Prefix B+ Tree

Quando em vez das chaves inteiras, utilizamos separadores como os índices mantidos na B+Tree, chamamos a estrutura de Simple Prefix B+ Tree

3. Manutenção da Simple Prefix B+ Tree

Quando se fazem necessárias as operações de atualização, inserção e eliminação, no Sequence Set de uma B+ Tree, precisamos manter o índice atualizado.

É importante notar que as operações são realizadas dentro do sequence set, pois é lá que os registros estão. Mudanças no índice são consequências das operações fundamentais no sequence set.
Adicionamos um novo separador no índice só se um novo bloco é formado no sequence set. Um separador é eliminado do índice se e só se um bloco é removido da sequência como consequência de uma concatenação.
Overflow e underflow dos blocos do índice diferem das operações no sequence set uma vez que o índice é potencialmente uma **multilevel**, e é portanto gerenciada como uma árvore B.
O tamanho dos blocos no índice é normalmente o mesmo que o tamanho escolhido para a sequence set. Tamanhos variados para os blocos podem ser suportados e requerem trabalho adicional.