"Melhoras com gasto adicional de memória"

- Bit visitado

- gasto um bit a mais para cada posição da tabela. Todos os bits são inicializados com zero e é atualizado para 1 quando na inserção de uma chave a ela está colidir com alguma posição já ocupada.

Toda vez que uma busca é executada e o bit adicional do n-ésimo re-hash é igual a zero significa que ninguém colidiu nele, inclusive a chave que está sendo buscada, logo ele não se encontra na tabela.

- Método preditor

- gasto um inteiro a mais para cada posição da tabela. Definimos uma função $prb(j, k)$ que computa diretamente a posição do j-ésimo re-hash da chave k,

Exemplo com re-hash linear

$$prb(0, k) = h(k)$$
$$prb(1, k) = (h(k) + c)\ \%\ T.S$$
$$prb(2, k) = (h(k) + 2c)\ \%\ T.S$$
$$\vdots$$
$$prb(j, k) = (h(k) + jc)\ \%\ T.S$$

O campo adicional será chamado de preditor, $pred[T]$, inicializado com 0. Suponha que a chave k_1 está sendo inserida e j é o menor inteiro tal que a operação $pred[prb(j, k)] = 0$ pois de mais alguns re-hash k_1 é inserido na posição $prb(j, k_1)$. $pred[prb(j, k_1)]$ é utilizado para $p = j$.

Numa busca, quando a posição $prb(j, k_1)$ for diferente de k_1 a próxima posição examinada é $prb(j + pred[prb(j, k_1)])$ ou $prb(p, k_1)$.

"Hashing Encadeado"

Gasto um inteiro a mais para cada posição de memória. Um inteiro para marcar a próxima posição disponível na tabela. Não há função de rehash, a busca é orientada por uma lista encadeada onde o campo auxiliar serve de indicador de próximo. No caso de colisão a lista é seguida até o fim. A chave é colocada na posição de marcador de disponível, a última posição da lista é atualizada para o lugar que a chave foi inserida. Analogamente o marcador de disponível é atualizado.

"Escolhendo funções hash"

- método da divisão: $h(k) = k \% T_s$

Quais as características que uma "boa" função hash deve ter?
* Depender de todos os bits da chave
* Sensível a permutações

- método da multiplicação: Consiste em escolher um número real entre 0 e 1 e definir $h(k)$ como $floor(m * frac(c \cdot k))$ onde m é o tamanho da tabela, frac() é a parte fracionária de um número real e floor() é a parte inteira.

Resultados teóricos mostram que valores de c com boas propriedades são $c = 0,6180339887$ ou $c = 0,3819660113$

- método do quadrado médio: Consiste em elevar a chave ao quadrado e selecionar alguns dígitos do "meio" do resultado

- método de dobra. →

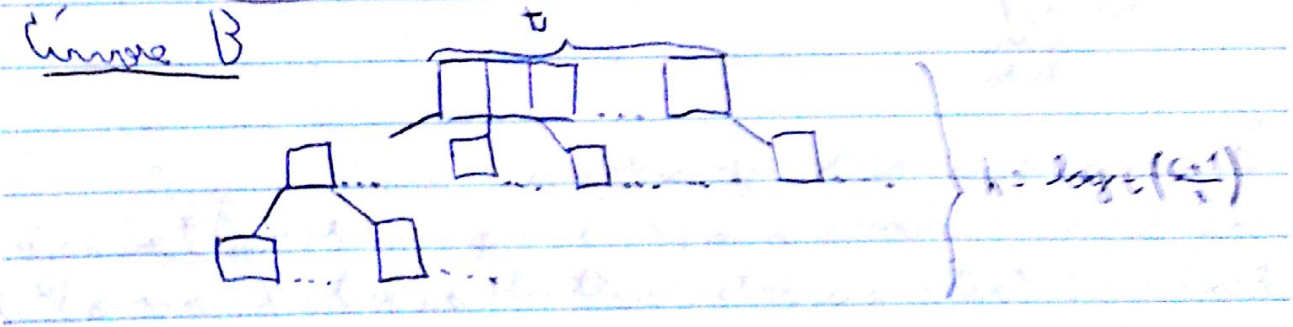
obs: funciona bem para $T_s = 2^k$

→ Completo-se o chave na sua representação binária. Digamos que o resultado foi 01011 10010 10110 e que $k = 5$. Em seguida fazemos uma operação de "ou" exclusivo com as linhas.

$$\begin{array}{r} 01011 \\ 10010 \\ \underline{10110} \\ 01111 \end{array} \quad = 1+2+4+8 = 15$$

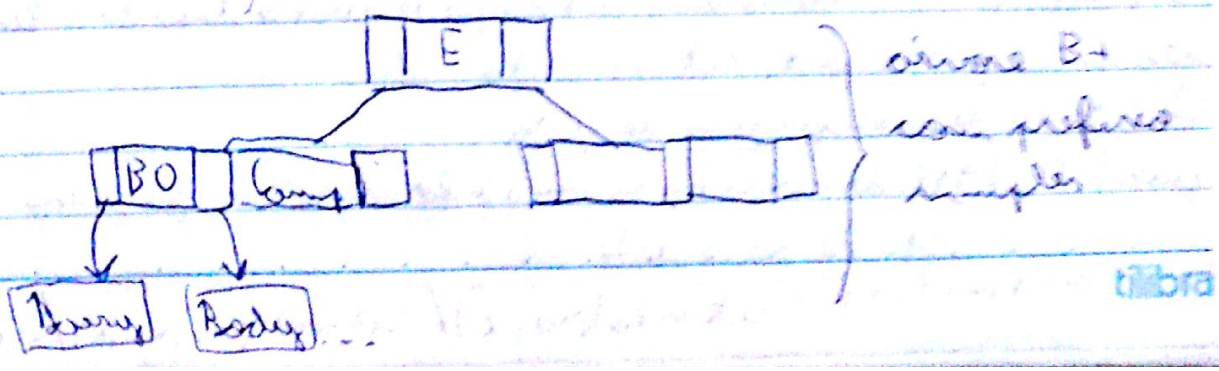
- Chaves alfa-numéricas: O mais comum é transformar cada letra em um número via tabela ASCII e escolher uma função qualquer.
Outra possibilidade é interpretar cada letra como um dígito na base 26.
Exemplo: bola = b o l a
$1 \cdot 26^3 + 14 \cdot 26^2 + 12 \cdot 26^1 + 0 \cdot 26^0 = 23316$

Árvore B



Árvore B+

- Usada para melhorar o percurso sequencial de chaves
- Lista separada dos dados ordenados via nós



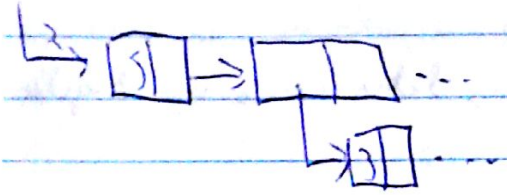
11

Listas Generalizadas

Uma forma de lista generalizada listas cujos nós podem armazenar elementos de diferentes tipos.

Exemplos

Representação: $l_1 = (5, '5', 12, 'A', 147)$



Representação

$l_2 = (5, (3, 2, (14, 9, 'a'), (1), 4), 2, (6, 3, 10))$

↓ nó ↓ nó / outra lista

Operadores para lista generalizada

head (l) → retorna a informação que existe no 1º nó da lista.

tail (l) → retorna a lista resultante da lista l menos o 1º elemento.

first (l) → retorna o primeiro nó da lista

Operadores sobre o nó de uma lista

info (no) → retorna a informação armazenada em um nó particular

next (no) → retorna o nó seguinte a um nó específico

nodeType (no) → retorna uma indicação de qual o tipo de informação está armazenada num determinado nó

Operadores de modificação da lista

push (l, x) → ele deve criar um nó com a informação x e fazer com que l aponte para este nó

addon (l, x) / setInfo (no, x) / setNext (no, next) / setHead (l, x)

Definição

Uma informação ou um nó é acessível a partir de uma lista ou de um ponteiro externo se existe uma sequência de operações head e tail que, se aplicada em l resultam numa lista em que é o primeiro elemento.

Liberação de memória

- Critério de exclusão: apagar todos os nós tal que não existam mais ponteiros para o mesmo

Estrutura para o nó

```
struct node{
    int type;
    union{int int_info;
          char char_info;
          node node_info
         }info;
    node* next;
    int refs;
}
```

Técnica não recursiva de exclusão, em 2 passos
Passo 1 - marcação
Passo 2 - remoção

- Supondo que os nós estão em um vetor Nodes[Num_Nodes] e os ponteiros externos estão em um outro vetor [num]

```
for(i=0, i<max, i++)
   nodes[arc[i]].mark=true;
int j;
i=0;
while(i<numnodes){
   j=i+1;
   if(nodes[i].mark){
      if(nodes[i].tufo==i && &nodes[node[i].info].mark!=true){
         nodes[nodes[i].info].mark=true;
         if(nodes[i].info<j)
            j=nodes[i].info;
      }
      if((node[i].next>0)&&(node[node[i].next].mark!=true){
         nodes[nodes[i].next].mark=true;
         if(nodes[i].next<j)
            j=nodes[i].next;
      }
   }
   i=j;
}
```