09/10/17
10/10/17

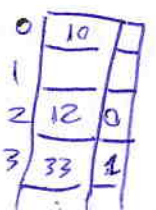
```
bool remove (int k){
    int i, tmp;
    i = h(k);
    while (T[i] > k){
        i = rh(i);
    if( T[i] == k){
        T[i] = -1; // recebe -1 e vai para as próximas, para que sejam ordenadas no lugar certo.
        i = rh(i);
        while (T[i] != -1){
            tmp = T[i];
            T[i] = -1;
            insert (tmp);
            i = rh(i);
        }
        return true;
    }
    return false;
}
```

Melhoria com uso de memória adicional

Bit visitado → não ordenado.

→ Usa um bit a mais pl cada posição da tabela. Inicialmente todas elas valem zero.

Durante a inserção de uma chave K, se ela colide numa posição cujo bit vale 0, ele atualiza para 1



buscando 52. Chega na pos. 2 e verifica valor = 12 ≠ 52 e bit = 0. → não foi feito mais nenhum rehash e portanto 52 não existe.

10/10/12

Na busca por uma determinada chave k, se k não ~~estivesse~~ estiver na Tabela, basta procurar até encontrar um bit == 0.
Pois se k estivesse, ela teria colidido naquela posição em particular e aquele bit valeria 1.

Técnica do Preditor

- gasta um inteiro a mais p/ cada pos. de mem. usando rehash linear

- Definimos uma função chamada Prb(j, k), que calcula diretamente o valor j, que é um rehash da chave k.

Exemplo

$Prb(0, k) = h(k)$

$Prb(1, k) = (h(k) + c) \% T_s$

$Prb(2, k) = (h(k) + 2c) \% T_s$

⋮

$Prb(j, k) = (h(k) + jc) \% T_s$

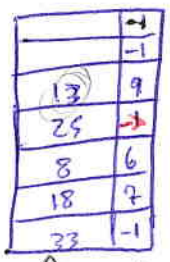
- Definimos um vetor do mesmo tamanho da tabela chamada de Prd[]
- Inicialmente Prd[i] = 0 para todo i.
- Durante a inserção de uma chave K, seja j o menor inteiro, tal que Prd[Prb(j, k)] = 0. Digamos que após mais alguns rehashes K é inserida na pos. prb(p, k);
- O valor de Prd[prb(j, k)] é atualizado p/ p-j
- Ao fazer ~~isto~~ a busca por k e chegar na pos. Prb(j, k) a próx. pos. consultada é $Prb(j + Prd[Prb(j, k)]) = Prb(p, k)$

16/10/12

Melhorias com adição de memória

Tabela Hash encadeada

Gasta um inteiro a mais para cada posição de memória



- Não tem função de rehash;
- Uma variável "global" controla a próxima posição de mem na tabela.
- Os elementos que colidem são organizados numa ~~tabela~~ lista encadeada

T_c → Tchave

T_e → Telementos

Exercício

↳ Escrever o código para inserir e remover elemento de uma tabela hash encadeada

```
int disp = randomNumber; // dentro do Range da tabela
int disp(){
int if(disp != -1){
    ~~if(Te[disp] == -1) return disp;~~
    while(Te[disp] != -1){
        disp = (disp+1) % Tesize
    }
    return disp;
}
}
bool inserir(int k){
i = h(k);
while(Te[i] != -1 && Te[i] != k){
    i = Tc[i];
}
if(Te[i] == k) return false;
if(Te[i] == -1){
    Tc[i] = K;
    return true;
}else{
    disp = disp();
    Tc[i] = disp;
    Te[disp] = K;
    return true;
}
```

ABD2 2

bool ~~int~~ remove (int k){

```
int i = h(k);
if (Te[i] == k){
    Te[i] = -1;
}else{
    while ((Tc[i] != -1) && ( Te[Tc[i]] != k)){
        i = Tc[i];
    }
    if (Te[Tc[i]] != k ) return false;
    Te[Tc[i]] = -1;
    Tc[i] = Tc[Tc[i]];
    return true;
}
return false;
}
```

	Te	Tc
0	~~10~~	1
1	20	~~2~~
2	30	-1
3	33	4
4	~~53~~	~~5~~
5	~~6~~ 3	-1

20 : 2

16/10/12

```
if (Tc[i] == -1) return true;
else{
    Te[i] = Te[Tc[i]];
    Tc[i] = Tc[Tc[i]];
    Tc[Tc[i]] = -1;
    Te[Tc[i]] = -1;
    return true;
}
```

AED2
1

17/10/12

Funções Hash

$h(k) = k \% T_s$

→ método de divisão T_s é nº primo

$29 \% 10 = 9$
$19 \% 10 = 9$
} Não leva em conta todos os algarismos

→ depender de todos os bits da chave.

$139 = h(139) = 1+3+9 = 13 \% 10 = 3$
$193 = h(193) = 1+9+3 = 13 \% 10 = 3$

→ Deve ser sensível a permutações.

Método da Multiplicação

Consiste em escolher um número real c entre 0 e 1 e definir $h(k) =$

→ $h(k) = floor(T_s \cdot frac(k \cdot c))$

Em que floor retorna a parte inteira de um nº real e $frac(x) = x - floor(x)$

{ c → 0 → vai resultar em um nº muito pequeno → pouca alteração
{ c → 1 → não muda nada pois o valor de K não seria ~~valor~~ alterado significativamente

Se o tamanho da palavra é 2^b, C deve ser escolhido de tal forma a parte inteira de $2^b \cdot c$ seja relativamente primo com 2^b.

Teoricamente são valores de C com ~~bloco~~ boas propriedades:

$c = 0,618033\ldots$ ou
$c = 0,381966\ldots$

Métodos de Quadrado Médio

Funciona "melhor" para $T_s = 10^b$

~~Converte~~
Consiste em elevar a chave K ao quadrado e escolher os b dígitos centrais ~~interiores~~ de resultado

$b = 2 \Rightarrow Ts = 100$

$K^2 = 18\boxed{79}56$

Exercício

Escreve um código em "c" que implemente o método quadrado médio

```
int b = 2;
int Ts = 100;
int hash (int k){
    int digitos = 1;
int k2 = k * k;
    int aux = k2;
    while (aux > 1){
        aux = aux / 10;
        digitos ++;

    }
    int meio = (int) digitos / 2;
    int i = pow(10, meio - b/2);
int aux2 = (k2/i) % 100;
    return aux2 % (pow (10, b));
}
```

AED-2

23/10/12

Hash em Disco

- Bucket → cesto → armazena diversos registros num mesmo endereço
  - ↳ tamanho entre cluster e trilha para evitar o movimento da cabeça de leitura.

59 → b → nº de elem. dentro do cesto.

↳ registros dentro do cesto

Hash Extensível

1ª Solução

Número de buckets varia

À medida que os buckets se enchem eles são ~~separados~~ divididos e duplicados.

2 casos
- ~~cada~~ nível global = nível local → divide e duplica diretório e bucket
- nível global ≠ local → divide apenas bucket

Hash Linear

→ qnd um bucket está cheio o bucket do next é dividido.

next → próximo bucket a ser subdividido.

~~Como next já foi dividido então~~

→ como next > h(x) então significa que bucket foi dividido

quando se olha pros n bits significativos → na busca

AED

Árvore B

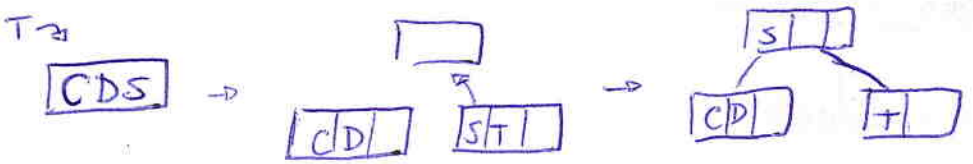
24/10/12

- Construir a árvore de baixo p/ cima
- garante que o tempo todo estará balanceada

- ocupação mínima $(m/2) - 1$

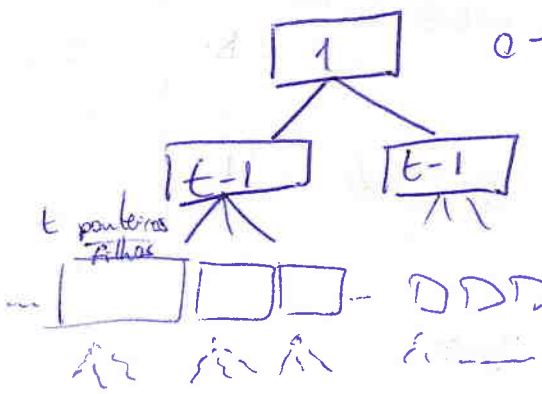
ordem → nº de ponteiros para filhos

- promove la chave da direita



- qnd há subdivisão há promoção
- sempre tenta-se inserir nas folhas.

Altura da árvore



$0 \rightarrow 1$

$1 \rightarrow 2(t-1)$

$2 \rightarrow 2t(t-1)$

$3 \rightarrow 2t^2(t-1)$

$\vdots$

$h \rightarrow 2t^{h-1}(t-1)$

$n \geq 1 + \sum_{i=1}^{h} 2t^{(i-1)}$

17/10/12

Método de dobra

~~Consiste~~ converte a chave na sua representação binária funciona para $T_5 = 2^b$ digamos que $b=5$ e a representação binária de K é 01101 10001 11010

separa o resultado em blocos de b bits e faça uma operação de "ou exclusivo" com os blocos.

```
 01101
 10001
 11010
 -----
 00110  → p/ decimal  6
```

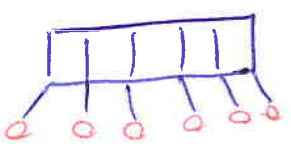

30/10/12

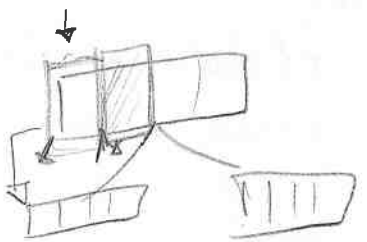
Árvore B

t=3

ordem 2t → nº máximo de ptrs que uma árvore pode ter.

2t-1 chaves, 2t ponteiros





Estrutura do NO

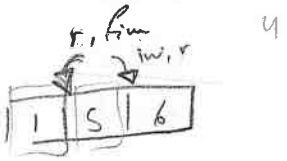
```
const t = 5;
struct no{
    int num_chaves;
    int chaves [2*t-1];
    no filho [2*t];
}
```

30/10/12

Exercício

Escrever um algoritmo de busca numa árvore B.

```
bool buscaArvoreB (int k, no atual){
    bool resp = false;
    if(atual){
        int r = buscaBin (k, atual);
        if (r!=-1) return true;
        else resp = buscaArvoreB (k, atual->filho[r]);
    }
    return resp;
}
```

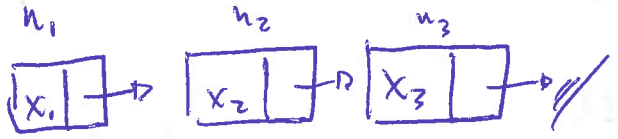



```
int buscaBin (int k, no atual){
    int ini = 0;
    int fim = atual->num_chaves - 1;
    int meio = (ini + fim)/2;
    if(...) return -1;
    while (ini <= fim){
        if (atual->chaves[meio] == k) return meio;
        if (atual->chaves[meio] > k){
            fim = meio - 1;
        }else{
            ini = meio + 1;
        }
    }
    return ini;
}
```

Exercício → imprimir as chaves de uma Árv. B em ordem crescente

Listas Generalizadas

~~11/06/12~~
06/11/12

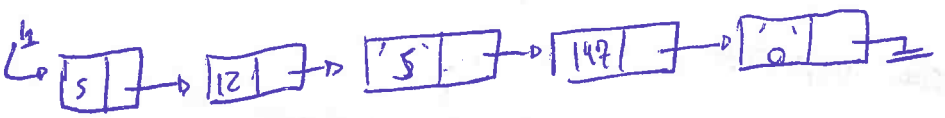


Pode acontecer de $x_1 = x_2 = x_3$

No é uma entidade distinta de informação nele contida

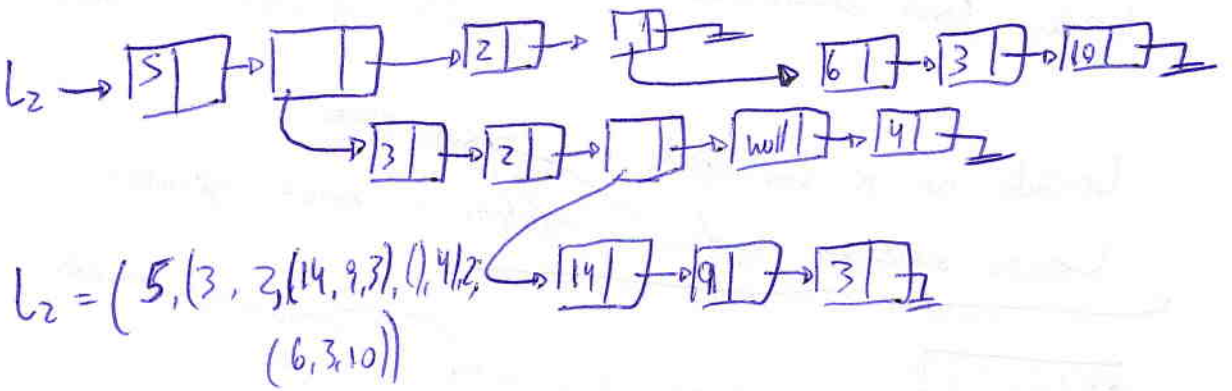
Nesse contexto, chamaremos de "listas generalizadas" uma lista cujos nós podem armazenar informações de diferentes tipos.

Exemplo



Representação

$L_1 = (5, 12, '5', 147, 'a')$



$L_2 = (5, (3, 2, (14, 9, 3), (), 4), 2, (6, 3, 10))$

31/10/12

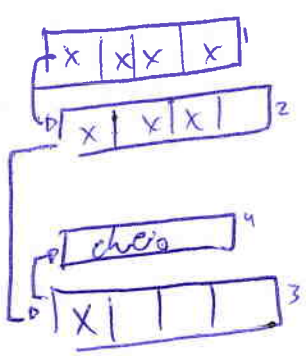
```
void imprime (NO* p){
    if(!p) return;
    int i=0;
    while (i< p->num_elem){
        imprime (p->filho[i]);
        printf("%d", p->chave[i]);
        i++;
    }
    imprime(p->filho[i]);
}
```

nº de chaves por nó → ordem - 1

Árvore B+

- informação repetida
- pode-se obter os registros de forma → sequencial
  - indexada
- bom para → acesso sequencial
- em geral alterações no último nível causarão poucas ou nenhuma alteração (ções) na parte de cima.

- cada nó é um seek → separadores
- nós superiores ajudam a efetuar a busca apenas



pode-se reorganizar 3 e 4 → mais simples

" 3 e 2

pode-se fazer merge entre 3 e 2 → transforma em 1 só.
→ bom qnd se tem pouca memória

busca termina na folha

06/11/12

- head(l) → devolve a informação contida no primeiro nó da lista

head(l_2) = 5 → retorna um ponteiro para void → void*
↳ depois deve ser feito um cast

- Tail(l) → lista resultante de l menos o primeiro nó.

exemplos

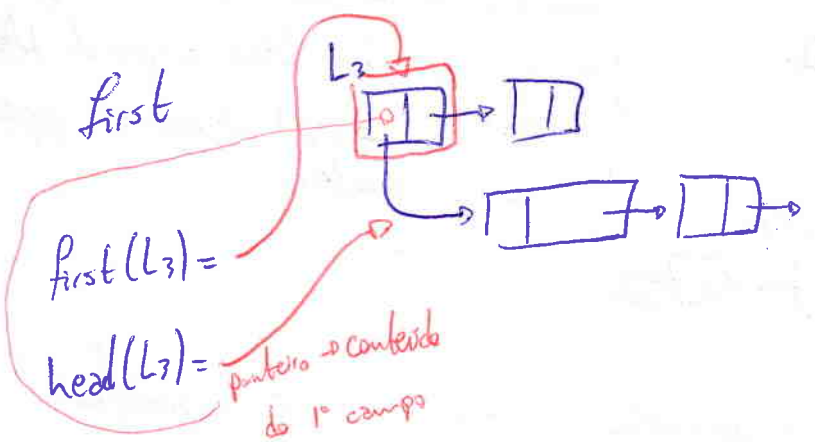
l_1 = (5, 12, 's', 147, 'o')

tail(l_1) = (12, 's', 147, 'o')

head(tail(l_1)) = 12

tail(tail(l_1)) = ('s', 147, 'o')

- first(l) → retorna o primeiro nó da lista
↳ O nó inteiro



- inf(no) → retorna a informação contida no nó.

head(l) = info(first(l))

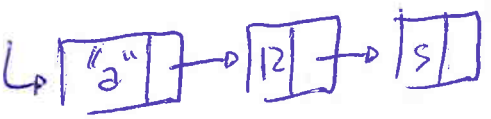
06/11/12

- next(no) → retorna o nó seguinte a um dado nó.
- node type(no) → retorna uma indicação do tipo de elem contém aquele nó.

Operadores que modificam listas

- push(l, x) → adiciona um nó com o valor x na frente da lista

```
l = null;
push(l, 5);
push(l, 12);
push(l, "a");
```



- addon(l, x) → retorna uma lista l1 cujo tail(l1) = l e head(l1) = x

cria uma nova lista com o elem. inicial = ao do parâmetro e o próximo é o resto da lista original. Não é feita cópia! Mexe-se apenas com ponteiro.

l2 = addon(l, 15)

- set info(no, x) → coloca a informação x em um nó específico
- set next(no, next) → atualiza o ponteiro next de um ~~elemento~~ determinado nó.

- set head(l, x) = set info(first(l), x)

exemplo: l = (5, 10, 8) → set head(l, 18) → l = (18, 10, 8) → set head(l, (5, 4, 17))

l = (15, 4, 17, 10, 8);

settail(l1, l2) = set next(first(l1), first(l2))

(tail de l1 recebe ~~a lista~~ l2 inteira)

Exercício

Sem pensar em sublista

Escrever um código para somar uma unidade a toda a variável do tipo inteiro de uma lista generalizada.

```
void somaUmL( (No(cabeca){
    if
    No atual = cabeca;
    while (atual){
        if(nodeType(atual) == 1)// 1 é int
            setInfo(atual, info(atual)+1);
    }
    atual = next(atual);
}
}
```

para recursivo e c/ sublistas

```
if(NodeType(atual) == 3){
    somaUmLinfo(NO)); // passa o conteúdo
                      // ponteiro p/ cabeça
```

AED

04/11/12

Exercício 2

Escrever um código para apagar todos os nós com a informação igual a "w"

```
void apagaW (No cabeca){
    No atual = cabeca;
    No ant = cabeca;
    while (atual){
        if (strcmp(info(atual), "w") == 0){  // se 0 então é igual
            if(atual == ant){
                NO temp = atual;
                cabeca = next(atual);
                free(temp);
            }else{
                No temp = atual;
                setNext(ant, next(atual));
                free(atual);
            }
        }else{
            ant = atual;
        }
        atual = next(atual);
    }
}
```

// ant não pode ser atualizado
// caso seja excluída um nó
// pois

AED 3 #Informação Acessível 04/11/12

Definição: Um nó n ou uma informação x é acessível a partir de uma lista l ou de um ponteiro externo se existir uma sequência de operação.

head e tail que, quando aplicada em l, resultam em uma lista em que n é o primeiro nó.

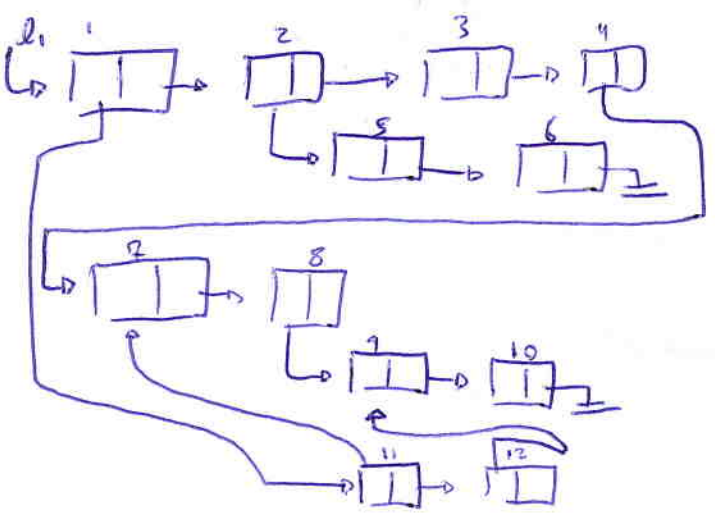
Pergunta

O nó com a informação [14] acessível a partir de lista l?

↳ head (tail (tail (head (tail (l)))))

AED 2 13/11/12

Liberação de Memória



A ideia geral é somente apagar os nós que não estão acessíveis a partir de ponteiros externos.

13/11/12

Uma solução possível é manter dentro do próprio nó um contador de ponteiros

Estrutura do nó

```
struct Node {
    int type;
    union {
        int intinfo;
        char charinfo;
        Node * lstinfo;
    } info;
    int ponteiro;
    Node * next;
};
```

Exercício

Implementar uma função que gerencia o número de ponteiros que chegam em um nó.

```
void gerenciaLista (Node no){
    if(no){
        no.ponteiro += ... ;
    }
    if(no.ponteiro <= 0){
        if (Nodetype(No) == 3) {
            gerenciaLista (no->info);
        }
        gerenciaLista (no->next);
        free (no);
    }
}
```

→ tipo lista

muitas varreduras caso haja uma grande lista para frente (descendo bar)

Algoritmos de Duas Fases

13/11/12

```
struct Node {
    int mark;
    int type;
    union {
        int itinfo;
        char charinfo;
        int lstinfo;
    } info
    int next;
}
```

Vamos supor que todas as ponteiras externas estão armazenadas em um vetor `acc[max]`

→ Apenas os nós acessíveis por ptrs externas contida em Nodes

Fase de marcação

```
for (i=0; i<max; i++){
    Nodes[acc[i]].mark=1;
}
```

// Mark = 1 não pode apagar

```
i=1;
while (i < NumNodes){
```

→ todos os nós

```
    j= i+1;
    if (Nodes[i].mark){// 1 representa true
        if((Nodes[i].type == 3) && (Nodes[Nodes[i].lstinfo].mark != 1)){
            Nodes[Nodes[i].lstinfo].mark = 1;
        }
        if (Nodes[i].lstinfo < j)
            j= Nodes[i].lstinfo;
    }
    if(Nodes[Nodes[i].next].mark != 1){
        Nodes[Nodes[i].next].mark = 1;
        if(Nodes[i].next < j)
            j= Nodes[i].next;
    }
    i= j;
}
```