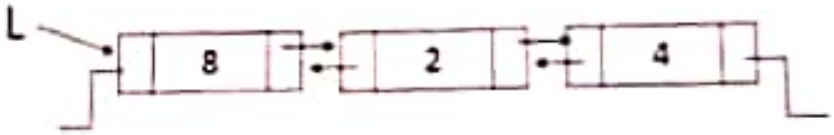
2ª Avaliação Parcial
Curso: Engenharia de Computação
Disciplina: Estruturas de Dados
Prof. Jarbas Joaci de Mesquita Sá Junior
Universidade Federal do Ceará – UFC/Sobral

Nome: Data 29/05/2018

1ª) Considere que um nó de uma lista duplamente encadeada é dado por:

```
typedef struct lista_dupl ListaDupl;
struct lista {
  int info;
  ListaDupl *ant;
  ListaDupl *prox;
};
```

Ex:



Implemente uma função que insere um valor no **fim** de uma lista duplamente encadeada. Essa função deverá receber o endereço do primeiro nodo da lista e o valor a ser inserido e deverá retornar o endereço do primeiro nodo da lista após a inserção. O protótipo da função deve ser: (2,0 pontos)

```
Lista* insere_fim_lista_dupl(ListaDupl* l, int x);
```

2ª) Realize os seguintes procedimentos:

a) Insira a seguinte sequência de chaves em uma árvore binária de busca: 39, 22, 13, 65, 44, 12, 56, 41, 17. (1,0 ponto);

b) Remova os elementos 22, 17 e 13 (1,0 ponto). **Obs**: a remoção deve, obrigatoriamente ocorrer na ordem apresentada.

3ª) Insira a seguinte sequência de chaves em uma árvore AVL: 9, 22, 13, 55, 44, 12, 56, 11, 17. (2,0 pontos)

4ª) Construa uma árvore rubro-negra para a seguinte sequência de chaves: 31, 62, 33, 42, 57, 68, 7, 81, 93. (2,0 pontos)

5ª) Explique como funciona o algoritmo **quicksort**. Quais são suas complexidades no pior e no melhor caso? (2,0 pontos)

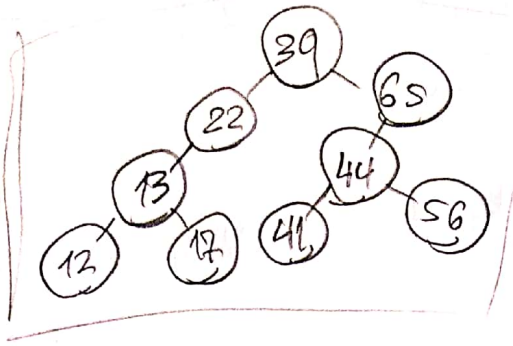
2AP Estrutura 2018.1
1ª Chamada

(1)

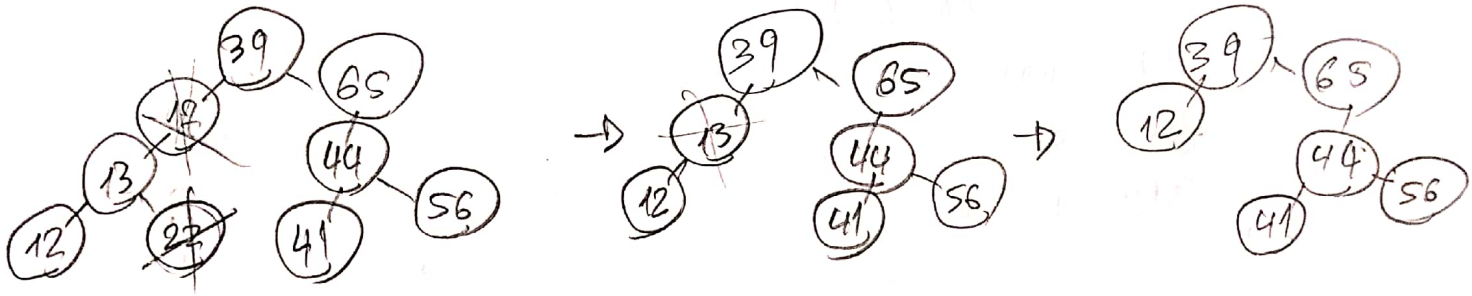
```
ListaDupl* insere_fim_lista_dupl (ListaDupl* l, int x){
    ListaDupl* ln = (ListaDupl*) malloc (sizeof (ListaDupl));
    ln->info = x;   ln -> prox = NULL;
    if (l == NULL){
        ln->ant = NULL;
    // ln->prox = NULL;
    }
    return ln
}

ListaDupl* laux = l;
while (laux->prox != NULL){
    laux = laux->prox;
}

laux->prox = ln;
ln -> ant = laux;
return l;
}
```

(2) ABB

a)

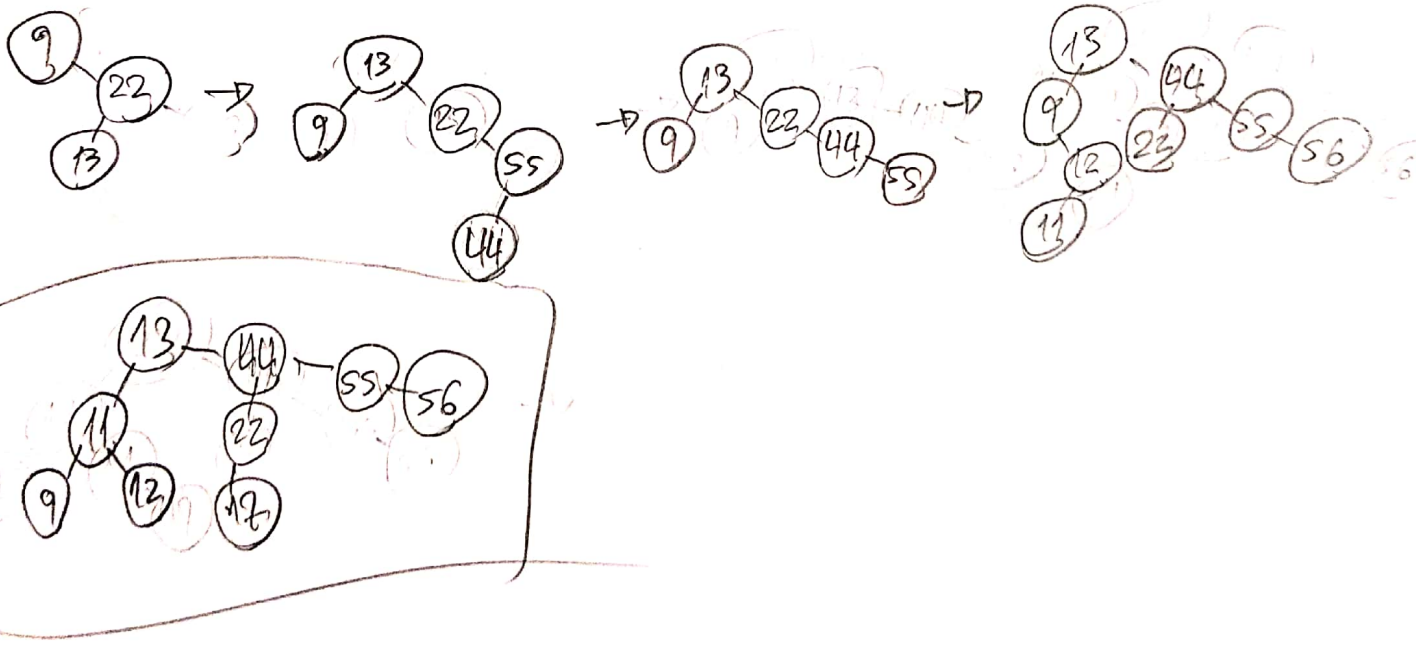


b) remover 22, 17 e 13

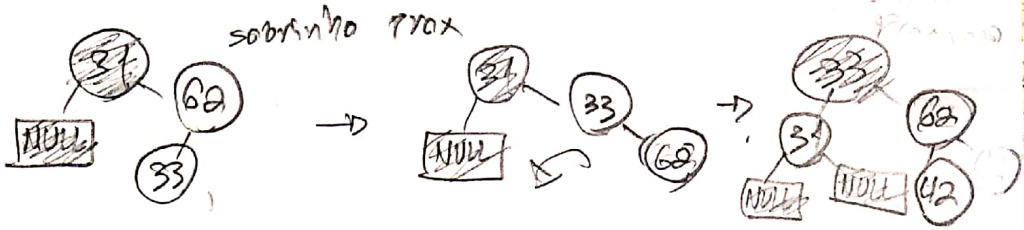


* se não houver o elemento mais a direita da subarvore esquerda nós só removemos

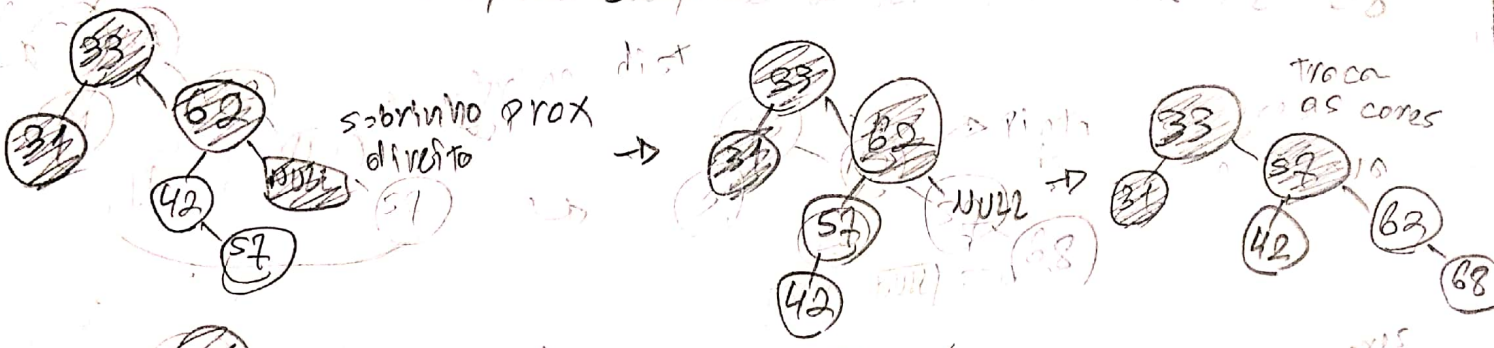
(3) AVL

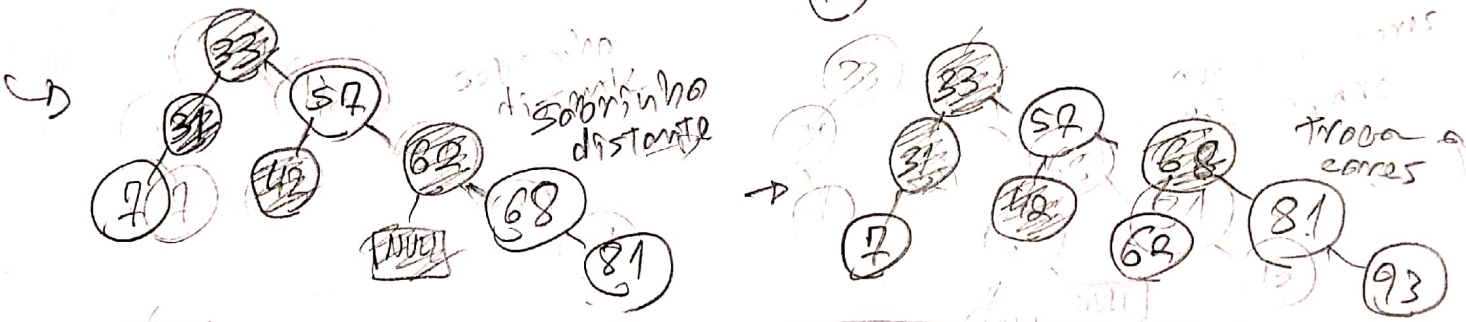


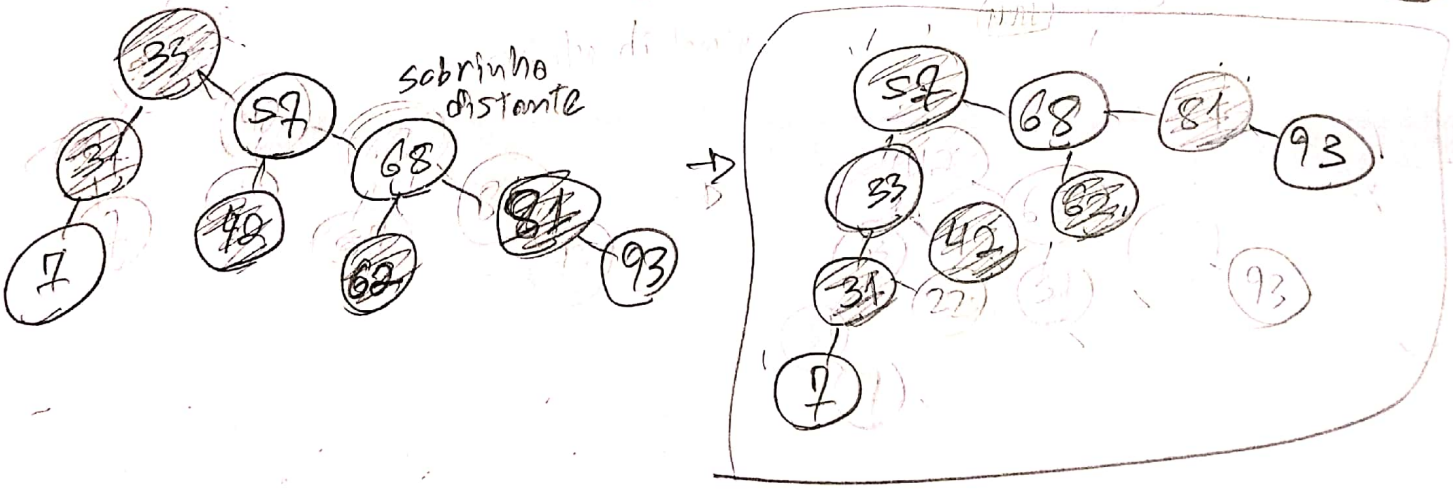
4) 1ª Chamada estruturas 2018.1



troca as cores do pai, tio e avô







(05) QUICKSORT → DIVIDIR PARA CONQUISTAR

O ALGORITIMO É UMA FUNÇÃO QUE RECEBE COMO PARAMETRO O TAMANHO n DO VETOR E UM PONTEIRO PARA UM VETOR V

↳ rapida (n, *V) {...}

INICIAMOS ESCOLHENDO UM PIVÔ X, QUE NESSE CASO SERÁ O PRIMEIRO ELEMENTO DO VETOR V[0]

↳ x = V[0];

CRIAMO UMA VARIAVEL a QUE SERÁ INICIALIZADA COM 1

↳ a = 1;

E TAMBÉM OUTRA VARIAVEL b QUE SERÁ INICIALIZADA COM n-1

↳ b = n-1;

DENTRO DE UM LAÇO INFINITO (while (1) {...}) FAREMOS O SEGUINTE:

- CRIAMO UM LAÇO QUE VAI INCREMENTA O VALOR DE a ENQUANTO V[a] ≤ X E ENQUANTO a < n

↳ while (V[a] ≤ x && a < n) {a++}

- CRIAMO UM LAÇO QUE DECREMENTA O VALOR DE b ENQUANTO V[b] > X

↳ while (V[a] > x) {b--;}

- TESTAMOS A CONDIÇÃO SE (a < b)

SE ESSA CONDIÇÃO FOR VERDADEIRA TROCAMOS OS VALORES V[a] E V[b] DE LUGAR E

E INCREMENTAMOS EM a E DECREMENTAMOS b.
SE A CONDIÇÃO NÃO FOR VERDADEIRA NÓS SAÍMOS DO LAÇO INFINITO

```
if (a<b) {
    temp = V[a];
    V[a] = V[b];
    V[b] = temp;
    a++;
    b--;
} else {
    break;
}
```

* QUANDO A FUNÇÃO SAI DO do *if* OS OUTROS DOIS *while* SÃO TESTADOS NOVAMENTE E O VALOR DE a E DE b PODEM SER ALTERADOS

QUANDO SAIRMOS DO *while(1)* NESSE PONTO A POSIÇÃO CORRETA DO PIVÔ SERÁ A POSIÇÃO b DO VETOR, OU SEJA V[b], ENTÃO TROCAMOS V[0]=x E V[b] DE LUGAR.

```
V[0] = V[b];
V[b] = x;
```

ENTÃO APLICAMOS A FUNÇÃO RECURSIVAMENTE À ESQUERDA DA POSIÇÃO b DO VETOR E À DIREITA

```
rapida(b, v); //esquerda do vetor principal
rapida(n-a, &V[a]); //direita do vetor principal
```

NO FINAL O VETOR ESTARÁ ORDENADO.

CONTINUA...

A COMPLEXIDADE DO QUICKSORT

↳ PIOR CASO: $\Theta(n^2)$

↳ MELHOR CASO: $\Omega(n \log(n))$

↳ CASO MÉDIO: $\Omega(n \log(n))$