EDA - Revisão para P1

- Algoritmo em linguagem C que leia notas.txt contendo as notas de n alunos

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{ FILE *filein, *fileout, *fileaprovados, *filereprovados, *fileporcentagem
```

Variaveis do tipo arquivo.txt

```c
 filein = fopen("Notas.txt", "r");

  if (filein == NULL)
  {   printf("\n\n Erro na abertura de arquivo \n\n");
      return1;
  }
     fscanf(filein, "%f", &nota);
   while (!feof(filein))
   {
        ||
   }
     fscanf(filein, "%f", &Nota);
fclose(filein);
```

lendo e manipulando um.txt

```c
fileout = fopen("Resultados.txt", "w");
printf(fileout, " || ", max, min);
fclose(fileout);
```

criando um.txt

Teorema 1 $\sum d(V) = 2E$ → arestas

↳ graus de cada vértice

GRAFOS SIMPLES

* O número de vértices de grau ímpar, de um grafo simples, é par

$$2E = \sum_{i=1}^{n} d(v_i) = \sum_{\text{grau par}} d(v_i) + \sum_{\text{grau ímpar}} d(v_i)$$

$2E$ ↓ é par; $\sum_{i=1}^{n} d(v_i)$ ↓ é par; grau par ↓ é par; grau ímpar ↓ Precisa ser par para validar o teorema

* O grau máximo de qualquer vértice, em um grafo simples com n vértices é $(n-1)$

- não pode ter arestas paralelas
- não pode ter laço com ele mesmo

* O número máximo de vértices $= n$; o número mínimo $= 1$;

* O número máximo de arestas, em um grafo simples com n vértices é

$$\frac{n(n-1)}{2} \therefore \underbrace{(\text{número máximo do grau de um vértice})}_{(n-1)} . \underbrace{(\text{número máximo de vértices})}_{(n)}$$

(Dupla contagem) → 2

* O número mínimo de arestas é $= 0$.

subgrafo → está contido

subgrafo induzido por vértices → copia vértices; $[V' > 0]$

subgrafo induzido por arestas → copia arestas, $[E' > 0]$

$G - d$ → exclusão

- Independente por arestas → Não compartilha vértices

- Um clique → Seleção de vértices tal que todos os pares de vértices são adjacentes. Toda aresta é um clique, já que depende de 2 vértices

- Vizinho de vértice d → todas as suas adjacências

- subgrafo disjunto → Não possuem arestas e/ou vértices em comum

- $\overline{G}$ → complementar $\therefore$ $[d = n(V) - 1]$ $\therefore$ $d = 7 - 1$

  $d = 6$ → cada vértice deve ter 6 graus

- Grafos isomorfos → { mesmo número de vértices
  mesmo número de arestas
  mesma sequência de graus

- Não existe grafo simples <u>K-regular</u> com K ímpar que possua um número ímpar de vértices → $[n(V) \cdot K = 2E]$ $\therefore$ n é o número de vértices, K é o grau de cada vértice e E o número de arestas. como $2E$ é sempre par e K pode assumir valores pares ou ímpares, n precisa ser sempre par para validar a equação

• Grafos K-Regulares → todos os Vértices tem o mesmo número K de

$[n(v)K = 2E]$

- Percurso simples: Não repete arestas.
- Percurso elementar (caminho): Não repete Vértices nem arestas.
- ciclo: Percurso simples e fechado.
- ciclo elementar: só há repetição do primeiro Vértice.

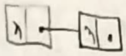
• "Todo percurso elementar é simples, mas nem todo percurso simples é elementar." meu caro Watson!

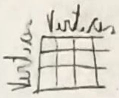
• Grafo conexo → Quando existe caminho entre quaisquer dois Vértices

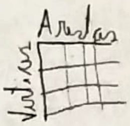
• "Para que um grafo simples e conexo não possua ciclos de comprimento ímpar, o número de Vértices para todo possível ciclo deve ser ímpar, (Valendo também o inverso), desconsiderando o primeiro e último Vértice, que são o mesmo Vértice para todo ciclo.

* Todos os ciclos de comprimento com Valor ímpar de Vertices → O ciclo terá Valor par de arestas.

* Todos os ciclos de comprimento com Valor par de Vértices → O ciclo terá Valor ímpar de arestas

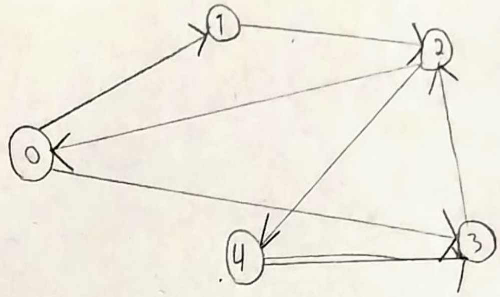
— Lista adjacência → 

— matriz adjacência → 

— matriz incidência → 

— Busca profundidade.

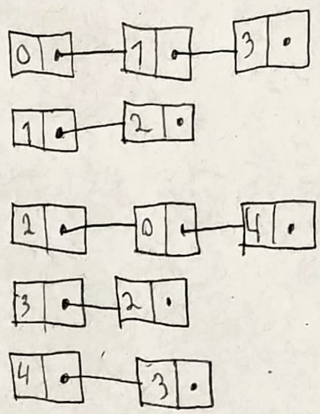
— Busca largura.



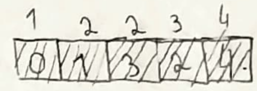
• Busca Profundidade

V	Cont
0	1
1	2
2	3
3	5
4	4

• Lista adjacência



• Busca largura

Fila 

V	cont
0	1
1	2
2	2
3	3
4	4

— Um clique é um subgrafo (K-Regular ou Não) onde $\left(\overset{\text{(grau)}}{K} = n(V) \text{ do clique} - 1\right)$

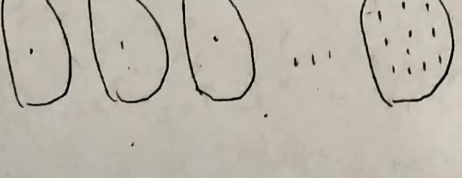
— Um grafo com n vértices e K componentes conexas tem no máximo

$$\frac{(n-k).(n-k+1)}{2}$$ arestas.

$(n-k+1)$ → número máximo de vértices na componente conexa de maior cardinalidade

2 → elimina dupla contagem

$(n-k)$ → grau máximo de cada vértice

k-1  $n-(k-1)$

Caminho Euleriano → Visita todas as arestas exatamente 1 Vez

Caminho Hamiltoniano → Visita todos os Vertices exatamente 1 Vez

EDA - Prova P2 - 18/10/2022

- Heap → Estrutura de dados
  - → Heap de máximo
  - → Heap de mínimo

- Algoritimos:

(arranjar, inserir, subir, descer, remover)

- Árvore B / Árvore AVL

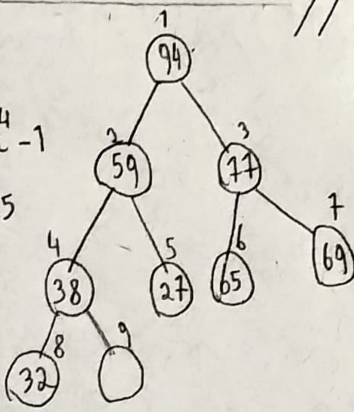
- Função Hashing → (método da divisão, método da multiplicação, e método da dobra)

- Heap

$n = 8$

$m_{max} = 2^4 - 1$

$m_{max} = 15$

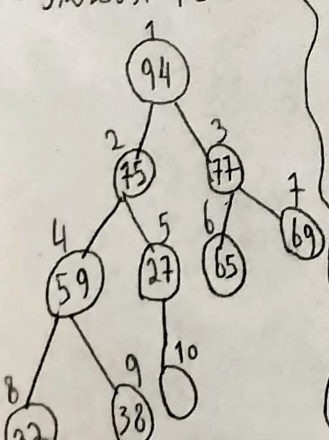


* n → número atual de elementos na lista

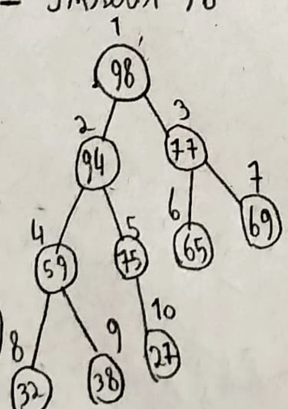
* m → Tamanho da lista

* número máximo de nós em uma árvore binária de altura h → $(2^h - 1)$

- Inserir 75



- Inserir 98

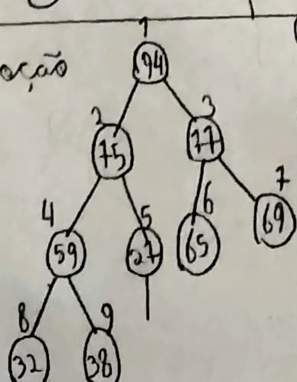


* Inserção começa pelo (n_{max})

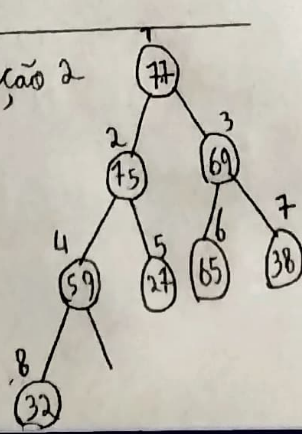
* Remoção começa pelo ($n = 1$)

* camadas não terminais devem ser preenchidas por completo

- Remoção



- Remoção 2



- Algoritimos:

* Inserir

```
inserir(nova, n, M){
    if(n<M){
        L[n+1] = nova;
        n = n+1;
        subir(n);
    }
}
```

nova → novo elemento

n → numero atual de elementos

m → tamanho da lista

i → posição do elemento a ser revisto

* Subir

```
subir(j){
    J = [j/2];
    if(J>=1){
        if(L[j]>L[J]){
            trocar(L[j], L[J]);
            subir(J);
        }
    }
}
```

* Remover

```
Remover(n){
    if(n=!0){
        L[1] = L[n];
        n = n-1;
        descer(1, n);
    }
}
```

* Descer

```
descer(i, n){
    J = 2*i;
    if(J<=n){
        if(J<n){
            if(L[J+1]>L[J]){
                J = J+1;
            }
        }
        if(L[J]>L[i]){
            trocar(L[i], L[J]);
            descer(J, n);
        }
    }
}
```

* Arranjar

```
Arranjar(n){
    for(i= n/2; i>=1; i++)
        descer(i, n);
}
```

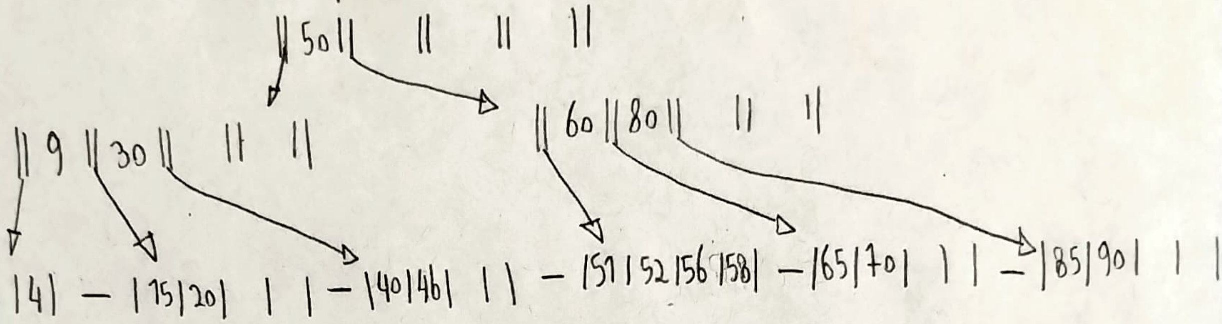
ore B / AVL

|| → chave
|| → ponteiro

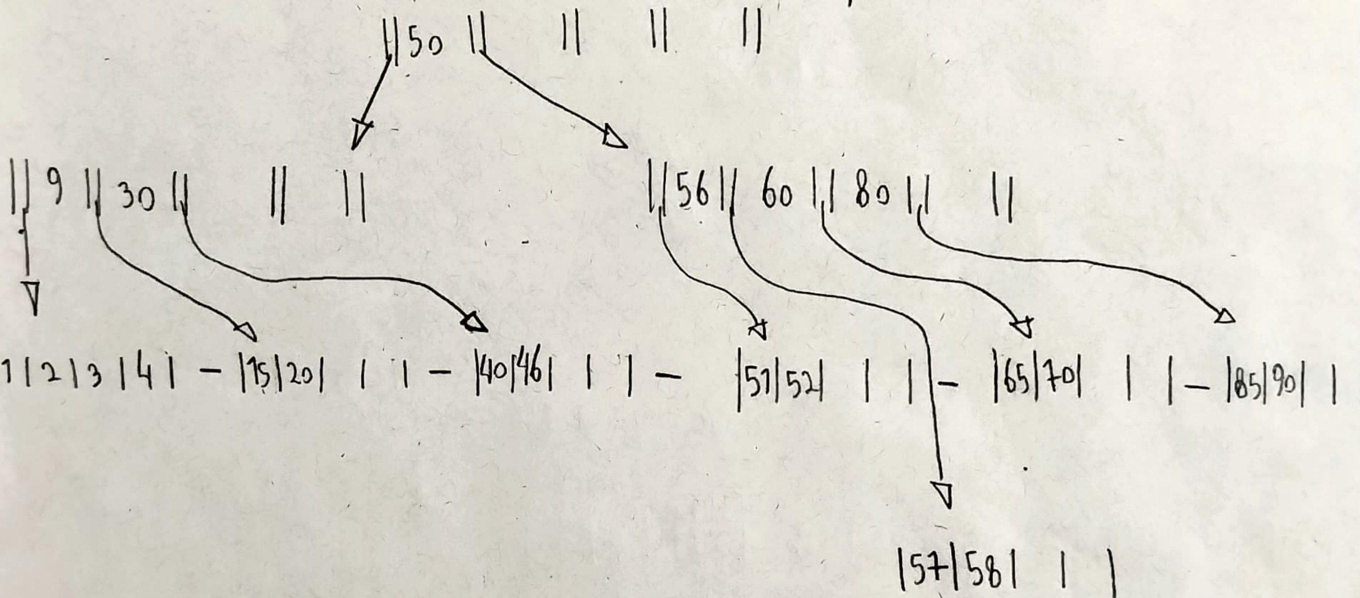
d = número mínimo de chaves em cada nó, exceto a Raiz

2d = número máximo de filhos em cada nó

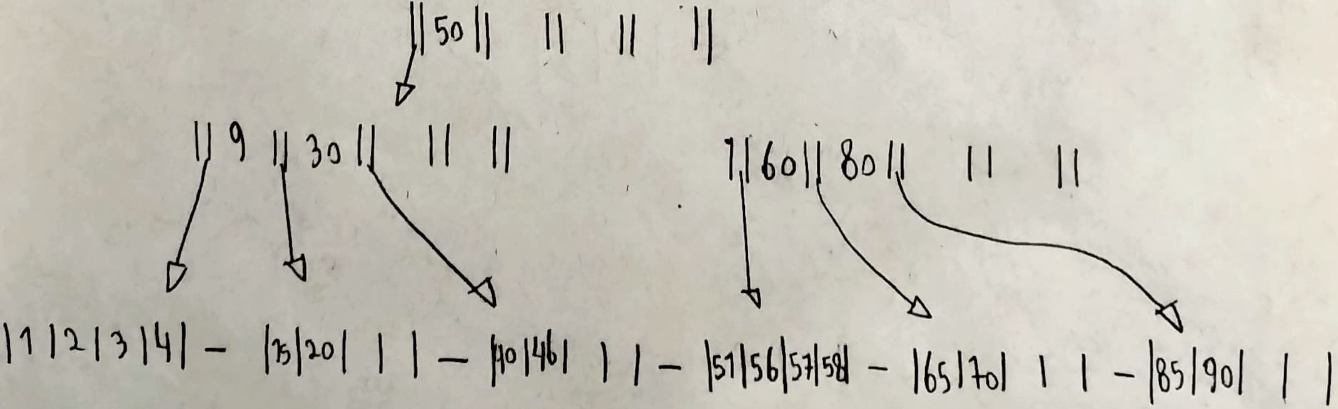
[d = 2]



- Inserir 57 → a pagina foi particionada, pois ficaria com 2d+1 chaves



- Remover 52 → Exclusão de chave com concatenação, uma pagina será excluida e outra será preenchida



- Funções Hashing → chave → $\boxed{f}$ → Posição

- método da multiplicação
  * constante fracionaria A
  * Table size T

→ $f(\text{chave}) = \text{Parte inteira}(T \cdot \text{parte fracionaria}(\text{chave} \cdot A))$

Exemplo

$A = 0,852$

$T = 997$

$f(578946)$

$\therefore f(578946) = 578946 \cdot 0,852 = 493261,992$

$f(578946) = 997 \cdot 0,992 = 989,024$

$f(578946) = 989$ //

- método da dobra

Exemplo

chegar a 2 digitos com a chave 75 39 57 82

$|7|5|3|9|5|7|8|2| \therefore$

$$\begin{array}{r} |5|7|8|2| \\ + \; |9|3|5|7| \\ \hline \end{array}$$

→ sem o vai 1 quando passar de 10

$|4|4|3|9| \therefore$

$$\begin{array}{r} |3|9| \\ + \; |4|4| \\ \hline \end{array}$$

$\therefore |7|3|$ //

...odo da Divisão → Funciona bem com table size de valor primo

...xemplo
$T = 13$

$f(44) = 44$ resto da divisão (13) ∴ $\begin{array}{r|l} 44 & 13 \\ \hline 39 & 3 \\ \underline{5} & \end{array}$ ∴ $f(44) = 44 \bmod 13 = 5$

$\begin{cases} T = 29 \\ f(6345) \end{cases}$ ∴ $f(6345) \bmod 29 = \begin{array}{r|l} 6345 & 29 \\ \hline 6322 & 218 \\ \underline{23} & \end{array}$ ∴ $f(6345) = 6345 \bmod 29 = 23$

- Propriedade que as chaves de um Heap têm que satisfazer.

$S_i \leq \frac{S_i}{2}$; sendo $1 < i < n$

- Função de árvore AVL que retorna o número mínimo de nós

```
int avl (int h){
    if(h == 0 || h == 1)
    return h;
    return(1 + avl(h-1) + avl(h-2));
}
```

- Árvore AVL → (Adelson - Velskii e Landis - 1962)

$* \; h = \log_2(n+1)$, onde $\begin{cases} h \text{ é altura} \\ n \text{ é número de nós} \end{cases}$

Uma árvore AVL é uma árvore binária balanceada, na qual as alturas das duas subárvores de qualquer nó nunca difere em mais de 1.