L.EIC
#2.1

Algoritmos e Estruturas de Dados

Resumos

1. Análise Teórica
2. Estruturas de dados
3. Algoritmos
4. Hash maps
5. Grafos

1. Análise Teórica

Pré-condições Restrições ao input **Pós-condições** Restrições ao output

Correção

- **Parcial** | Se respeitar as pré-condições, então, se terminar, respeita as pós-condições
- **Total** | Se respeitar as pré-condições, então termina e respeita as pós-condições

Invariante (de um loop)

Expressão sobre o estado de variáveis que é verdadeira antes, durante e após o ciclo.

- Prova que o resultado é correto

Variante (de um loop)

Expressão criada a partir de variáveis da iteração que representa um nº inteiro, não negativo e decrescente.

- Prova que o programa acaba.

Ex:

```
long f(int n) {
    long sum = 0;
    int k = 1;
    while(k <= n) {
        sum += k;
        k++;
    }
    return sum;
}
```

- **Invariante de ciclo:**
  sum = 1 + 2 + ... + (k-1)
  ↳ No passo k, a variável "sum" representa a soma dos números 1 a k-1.

- **Variante de ciclo:**
  n + 1 - k

∴ "Total correctness"

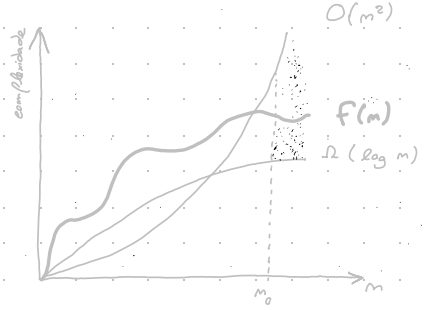
Complexidade

O()
- Pior caso
- Solução majorante

Ω()
- Melhor caso
- Solução minorante

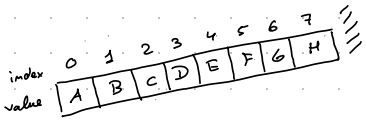
Θ()
- Quando O(g(n)) = Ω(g(n))

Ex:



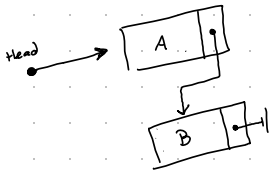
2. Estruturas de Dados

Vetor | Lista baseada em "arrays"



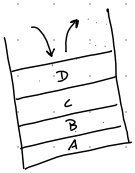
- Pesquisa muito rápida
- Tamanho fixo

Linked List | Lista de "nodes" que incluem o elemento e um apontador para o próximo



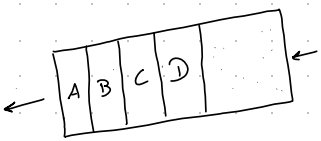
- Tamanho dinâmico
- Usa mais memória
- Pesquisa mais lenta

Stack | Pilha de elementos em que só o topo é acessível



- Mais simples
- FIFO
- Mais limitado

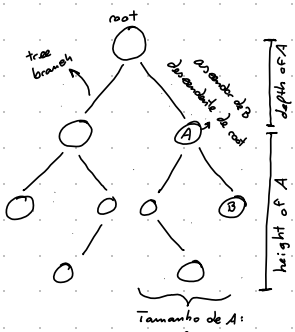
Queue | "Deque" de elementos em que só o fim é acessível.



- Mais simples, mas limitado
- FILO

Priority Queue | Semelhante à "queue", mas ordenada

Binary Tree | Árvore em que cada "node" pai tem até dois "nodes" filhos ligados.



Formas de percorrer:

- Preorder | root → left → right
- InOrder | left → root → right
- Post order | left → right → root
- By Level | Por profundidade (left → right)

- **Binary Search Tree**
  Ordenado pelo operador <
- Mais eficiente em inserir e remover

- **Balanceada**
  Para cada "node" as alturas das subárvores variam até 1
  - AVL Tree
  - Red-Black Tree

- **Binary Heap** | Representação vetorial de uma "Complete Binary Tree"

3. Algoritmos

Pesquisa

Sequential Search

$T(n) = O(n)$ | $S(n) = O(1)$

- Percorrer linearmente o vetor
- Útil em vetores pequenos e sem ordenação

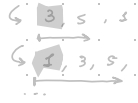
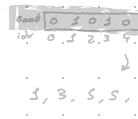
Binary Search

$T(n) = O(\log n)$ | $S(n) = O(1)$

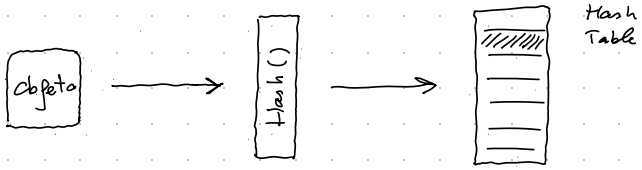
- Necessário um vetor ordenado
- Descartar metade do vetor a cada iteração

Ordenação

Ex: {5, 3, 1, 9, 5, 8}

• **Insertion Sort** $T(n) = O(n^2)$ $S(n) = O(1)$	Inserir cada elemento na posição correta.	
• **Selection Sort** $T(n) = O(n^2)$ $S(n) = O(1)$	Colocar o mínimo no início.	
• **Bubble Sort** $T(n) = O(n^2)$ $S(n) = O(1)$	Trocar elementos adjacentes fora de ordem até não ser necessário mais nenhuma troca.	
• **Merge Sort** $T(n) = O(n \log n)$ $S(n) = O(n)$	Dividir em dois sub vetores até só ter um elemento e reconstruir o vetor ordenado.	
• **Quick Sort** $T(n) = O(n \log n)$ $S(n) = O(n)$	Dividir elementos em 2 vetores a partir de um dado "pivot".	
• **Heap Sort** $T(n) = O(n \log n)$ $S(n) = O(1)$	Construir uma "binary heap" e remover ordenadamente os elementos para a estrutura desejada.	
• **Counting Sort** $T(n) = O(n + k)$ $S(n) = O(k)$	Contar o nº de ocorrências de cada elemento e reciar a partir do vetor auxiliar.	
• **Radix Sort** $T(n) = O(d * (n+k))$	Ordenar cada dígito usando um algoritmo estável (p.e, Counting Sort).	

4. Hash maps



$O(1)$
- Inserir
- Procurar
- Remover

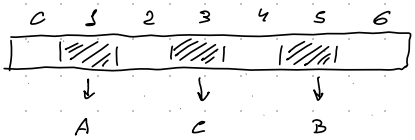
Hash Function

```
int hash(objeet, tableSize)
```

- Custo computacional baixo
- Distribui igualmente os objetos pela tabela

Hash table

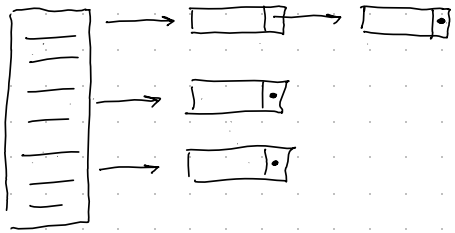
Mapeia índices da hash a objetos



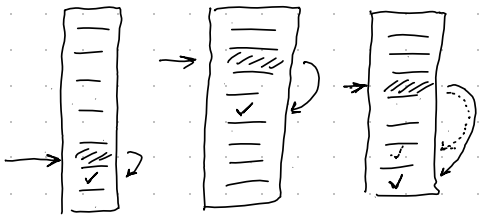
- Tamanho da tabela deve ser primo
- Load Factor (λ) deve ser $\simeq$ < 50%, caso contrário, a procura é demasiado lenta
- Quando vários objetos são mapeados para o mesmo índice, conflitos ocorrem.

Resolução de conflitos

Separate Chaining



Linear / Quadratic Probing



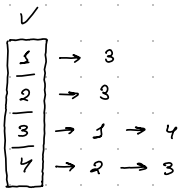
5. Grafos

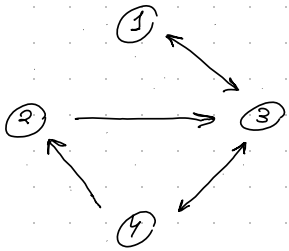
Representação

Matriz

v \ v	1	2	3	4
1			x	
2				x
3	x	x		x
4			x	

Lista





DFS

Algoritmos:

- Componentes conectadas
- "Topological Sorting"
- Deteção de ciclos
- Componentes fortemente ligadas
- Pontos de articulação

Complexidade:

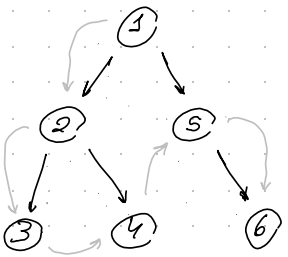
Temporal:

↳ Lista | $O(V+E)$

↳ Matriz | $O(V^2)$

Espacial:

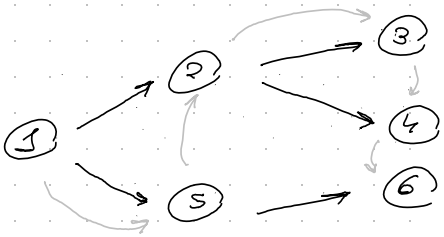
$O(V)$



BFS

Algoritmos:

- Calcular distâncias



A implementação de todos estes algoritmos e suas vantagens, bem como a nomenclatura associada a grafos estão disponíveis nos conteúdos da cadeira.