Algoritmia Aplicada

Ano lectivo 2011-2012

Aula 12 – Pesquisa

Sumário

- Pesquisa
  - Introdução. Operações associadas à pesquisa
  - Métodos fundamentais
    - Pesquisa sequencial
    - Pesquisa sequencial em listas
    - Pesquisa binária
    - Pesquisa binária interpolada
    - Pesquisa em árvore binária
  - Hashing
    - Funções de *Hash*
    - Encadeamento separado
    - Endereçamento aberto: Exploração linear; *Hash* duplo

Pesquisa

Pesquisa = Obter informação de um conjunto de dados previamente armazenado

- **Objectivo:** Encontrar todos os registos que verifiquem um **dado critério**, ou seja, cujas chaves estejam de algum modo (critério) relacionadas com uma chave de pesquisa

- **Estruturas de dados:**
  - Dicionários
  - Tabelas de símbolos

Pesquisa

- **Estruturas de dados:**
  - Dicionários
  - Tabelas de símbolos
- **Exemplos:**
  - **Dicionário da língua Portuguesa** → as chaves são as palavras e os registos são constituídos por toda a informação associada a cada palavra (definição, pronúncia, etc)
  - **Tabela de símbolos** → é o dicionário para um programa: as chaves são os nomes simbólicos usados no programa e os registos contêm informação que descreve os objectos referidos

Pesquisa

Algumas destas operações podem ser combinadas

- **Operações genéricas na pesquisa:**
  - ✓ **Inicialização da estrutura de dados**
  - ✓ **Pesquisa de um registo (ou registos) com uma dada chave**
  - ✓ **Inserção de um novo registo**
  - ✓ **Supressão de um dado registo**
  - ✓ **Fusão de dois dois dicionários para obtenção de um único**
  - ✓ **Ordenação do dicionário; saída de todos os registos por ordem**

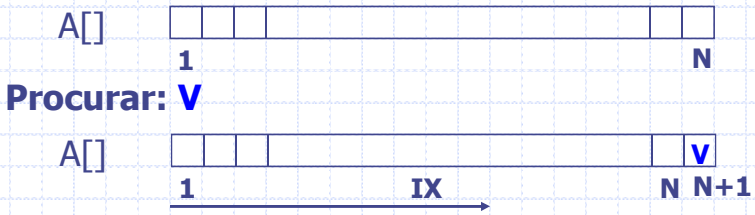
Na pesquisa é geralmente devolvido o local onde está o registo com a chave pedida, ou a informação necessária para inserir um novo registo com essa chave

Pesquisa

- **Registos com chaves duplicadas**
  - **Estrutura primária só pode ter registos com chaves diferentes**
    - Cada registo nessa estrutura pode conter um apontador para uma lista de registos com a mesma chave
    - ⇒ Todos os registos com uma dada chave são obtidos com uma única pesquisa
  - **Deixar os registos com chaves iguais na estrutura primária**
    - ⇒ devolver qualquer registo com a chave pedida na pesquisa (procurar para trás e para a frente...)
  - **Cada registo tem um identificador único (para além da chave)**
    - ⇒ fazer com que a pesquisa encontre um registo com um dado identificador e uma dada chave

Pesquisa sequencial

- **Registos armazenados num array**



Percorrer até encontrar
Se IX > N não encontra
mas termina sempre!

Deve ser possível passar IX como parâmetro para encontrar chaves duplicadas

Pesquisa sequencial

- **Dada a função:**

PESQUISA (A, N, V, POS)

Posição onde foi encontrada a chave [1..N+1]

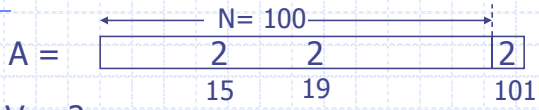
Num array A de N+1 posições
Iniciando-se a pesquisa na posição seguinte a POS

- **Imprimem-se asim, todas as posições em que a chave V ocorre em A:**

```
A[N+1] ← V
POS ← 0
REPEAT
        POS ← PESQUISA(A, N, V, POS)
        WRITE(POS)
UNTIL POS = N +1
```

Pesquisa sequencial

- **Exemplo:**



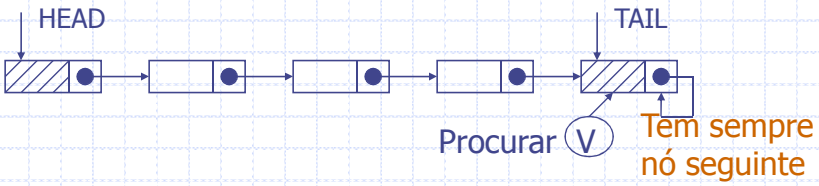
V = 2

Resultado: 15
19

```
A[N+1] ← V
POS ← 0
REPEAT
        POS ← PESQUISA(A, N, V, POS)
        WRITE(POS)
UNTIL POS = N +1
```


Pesquisa sequencial

- **Registos armazenados numa lista**



Pesquisa: Percorrer lista **até** encontrar V
ou enquanto < V

Inserir aqui

- **Vantagem:** **facilidade com que se mantém a lista ordenada**

Pesquisa sequencial

- **Se for conhecida a frequência relativa de acesso aos vários registos, poderá encontrar-se outro tipo de ordenação que facilite a pesquisa:**
  - **Colocar os registos mais frequentemente pedidos no início**
- **Se não for conhecida a frequência:**
  - **Método auto-organizativo: cada vez que um registo é acedido, movê-lo para o início da lista**

  Melhor implementado com listas do que com arrays, uma vez que não se torna necessário mover a informação contida nos registos, apenas se alteram os apontadores

Pesquisa binária

- **Pesquisa binária (array ordenado)**

Se a série de dados é muito extensa:

- **Tempo de pesquisa substancialmente reduzido com utilização do paradigma "dividir para conquistar"**
- **Dividir a série de registos em duas partes, deteminar em que parte é que a chave a encontrar está e analisar do mesmo modo apenas essa parte**
- **Método de dividir a série de dados: mantê-los ordenados e usar índices para o array ordenado de forma a delimitar a parte do array em que se deve trabalhar (delimitar a procura)**

Pesquisa binária

Compara com elemento ao meio, se é menor, está à esquerda, se é maior está à direita

- **Exemplo**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A	A	A	C	E	E	E	G	H	I	L	M	N	P	R	S	X
									I	L	M	N	P	R	S	X
													P	R	S	X
															S	X

Procurar S P=(E+D)div2
9=(1+17)div2
13=(10+17)div2
15=(14+17)div2
16=(16+17)div2

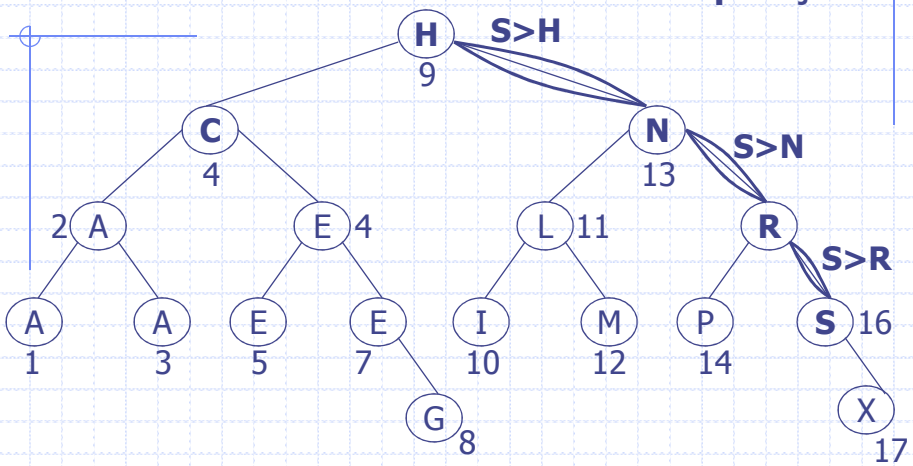
Pesquisa binária

- **Conclusões**
  - **O tamanho do intervalo é reduzido para metade em cada passo**
    - **Nº de vezes que ciclo é executado: lg_2N**
  - **Tempo requerido para a inserção é elevado: o array tem de ser mantido ordenado e para isso os registos têm de ser movidos para se criar espaço para os novos**
  - **Uma inserção requer em média que se movam N/2 registos, pelo que este método não deve ser usado em aplicações que requeiram muitas inserções**

⇒ **Se existirem registos com chaves iguais, têm de existir ciclos para ambos os lados para obter todos os registos**

Pesquisa binária

- **Sequência de comparações**



- **Baseia-se no valor V a procurar**
- **Descreve-se através de uma árvore binária**

Pesquisa binária interpolada

- **Melhoria à pesquisa binária**
  - **Tentar adivinhar mais concretamente onde a chave a procurar está, em vez de se tentar o meio**
  - **Processo semelhante ao utilizado para consulta numa lista telefónica**
    - **Se nome procurado começa por 'B', procuramos junto ao início da lista**
    - **Se começa por 'V', procuramos mais para o fim**
  - **Em função da chave e da dimensão, estimar a posição**

Pesquisa binária interpolada

- **Melhoria à pesquisa binária**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A	A	A	C	E	E	E	G	H	I	L	M	N	P	R	S	X
									I	L	M	N	P	R	S	X
													P	R	S	X

Em função da chave e da dimensão, estimar a posição
Exemplo: chave é 'S'

1		24		19	Alfabeto
'A'	..	'X'	----	'S'	
1	..	17	----	13	

A pesquisa termina em apenas 3 passos

24	-----	17
19	-----	x

⟹ **x = 13,45**

Pesquisa binária interpolada

- **Conclusões**
  - **Outras chaves são encontradas ainda mais facilmente: 'X' e 'A' são encontradas no 1º passo**
  - **A pesquisa interpolada consegue diminuir o número de elementos examinados para lg lg N**
    Se N =1000 000 000, então lg lg N <5
  - **Qualquer registo pode ser descoberto com muito poucos acessos, o que é um melhoramento considerável face à pesquisa binária**
  - **O método assume que as chaves estão bem distribuídas ao longo de todo o intervalo**
  - **Requer alguns cálculos adicionais**

Pesquisa em árvore binária

- **Procedimento**
  - **Construir uma estrutura composta por nós (árvore binária)**
  - **Cada nó é um registo com uma chave e dois elos (esq e dir)**
  - **Todos os nós com chave inferior (<) à de um dado nó, situam-se na sub-árvore esquerda**
  - **Todos os nós com chave superior ou igual (>=) à de um dado nó, situam-se na sub-árvore direita**

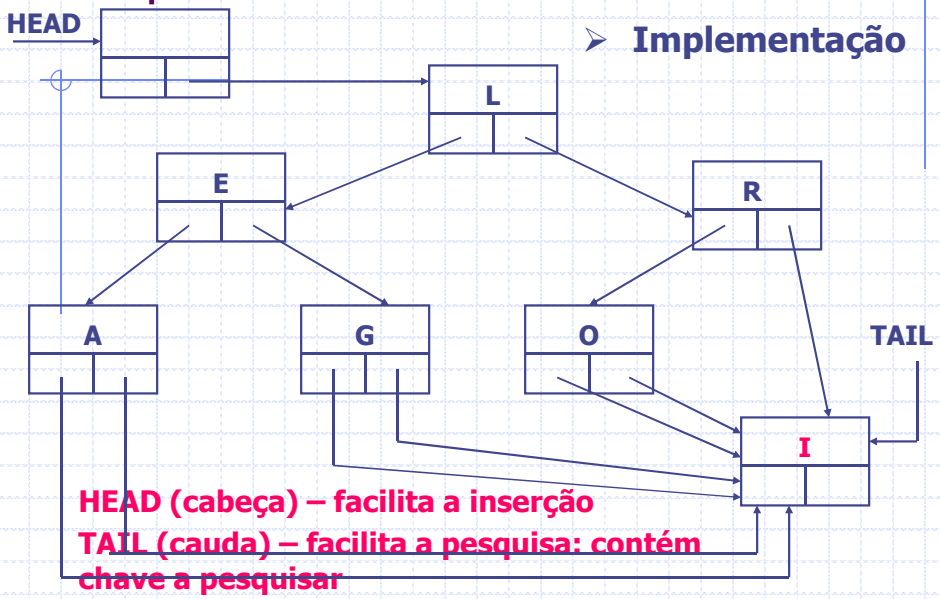
Pesquisa em árvore binária

- **Encontrar um registo com uma dada chave V**
  - **Começar pela raiz: compara-se com a chave da raiz**
  - **Se for igual a V, pára-se a pesquisa**
  - **Se for menor, procura-se pela sub-árvore esquerda**
  - **Se for maior, procura-se pela sub-árvore direita**
  - **Aplica-se o método recursivamente**

Pára-se quando o registo é encontrado ou quando a sub-árvore corrente estiver vazia

Pesquisa em árvore binária

- Implementação



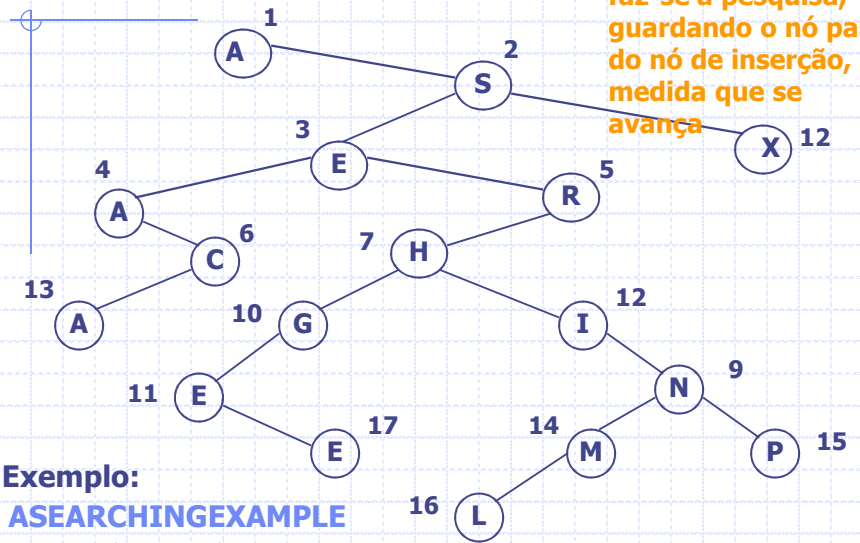
HEAD (cabeça) – facilita a inserção

TAIL (cauda) – facilita a pesquisa: contém chave a pesquisar

Pesquisa em árvore binária

- Inserção de um nodo na árvore

Para inserir um nó, faz-se a pesquisa, guardando o nó pai do nó de inserção, à medida que se avança



Exemplo:

ASEARCHINGEXAMPLE

Pesquisa em árvore binária

- **Conclusões**
  - **Tempos de execução dos algoritmos em árvores binárias dependem da forma da árvore**
  - **Se a árvore está equilibrada, temos $\log_2 N$ nós entre a raiz e o fundo $\Rightarrow$ tempos logarítmicos para a pesquisa**
  - **Se os N valores a colocar tiverem uma distribuição aleatória, então o que fica na raiz dividirá estatisticamente as chaves ao meio – árvore balanceada**

    Na prática e podendo na raiz ficar qualquer elemento, uma árvore balanceada não deverá ocorrer e a média de passos para a pesquisa é da ordem $2 \log_2 N$
  - **Se os as chaves aparecerem por ordem (ou ordem inversa), então o método resulta igual à pesquisa sequencial**

    Existem algoritmos para balancear as árvores

Hashing

- **Método *Hashing:*** Abordagem completamente diferente da pesquisa
- Referenciar os registos de uma tabela efectuando operações aritméticas sobre as chaves de modo a obter endereços numa tabela

Hashing

- Se as chaves fossem inteiros únicos entre 1 e N, então seria possível guardar o registo com chave I na posição I da tabela - chave inteira: endereço pode ser igual (acesso imediato)
- *Hashing:* generalização deste método trivial para aplicações de pesquisa onde não temos um conhecimento específico sobre os valores das chaves

Hashing

➢ **Utilização do *hashing***

1) Cálculo de uma função de *hash*
   - Função de hash: transforma a chave de pesquisa num endereço de uma tabela
   - Não existem funções perfeitas → chaves diferentes podem originar o mesmo endereço (colisões)

2) Resolução das colisões

   Utilização de métodos de resolução de colisões
   - Listas
   - arrays

Hashing

- **Hashing: exemplo do binómio espaço-tempo**

BINÓMIO	ESPAÇO	TEMPO
	Endereço = Chave	Pesquisa sequencial

- Se não existissem **limitações de memória**, poderíamos fazer apenas um acesso usando a chave como endereço
- Se não existisse a **limitação no tempo**, então poderíamos usar um espaço mínimo e fazer a pesquisa sequencial

É um método razoável se usarmos alguma memória e tempo e podermos balancear entre um extremo e outro

Hashing - Funções de *Hash*

- **Cálculo de um endereço a partir de uma chave**

- Problema aritmético semelhante à geração de números aleatórios
- A chave é numérica ou alfanumérica e temos de a transformar num inteiro entre os limites 0..M-1, onde M é o tamanho da memória disponível
- Função ideal deverá ser fácil de calcular e para cada entrada deverá gerar uma entrada igualmente provável (aleatória)
- Transformar a chave num número que possa ser operado e tão grande quanto possível

Hashing - Funções de *Hash*

- **Cálculo de um endereço a partir de uma chave**

CHAVE ⟶ NÚMERO ⟶ LIMITE de INDÍCES ARRAY

- **Método usual *(1)***
  - Escolher M como primo
  - Para cada chave K calcular

    H(K) = K mod M

Método linear, fácil de implementar e que espalha os valores de um modo razoável

Hashing - Funções de *Hash*

- **Cálculo de um endereço a partir de uma chave**
- **Gerador congruente linear de aleatórios *(2)***
  - Para cada chave K calcular

    H(K) = m *bits* iniciais de (bK mod w)
    - w é o tamanho da palavra do computador
    - b é um número escolhido

Vantagem: espalha melhor valores de chave muito próximos

Hashing

➢ **Binómio Espaço-Tempo**

➢ **Exemplo**

- 1000 chaves
- cada chave ∈ [1...20 000]

⇒ Se não há limitações de espaço

Usar a própria chave como endereço

Basta 1 comparação! → Tempo

Mas há 19 000 posições vazias! → Espaço

Hashing

(chaves são números inteiros > 0)

➢ **Por exemplo:** **A =**

.....	500	....	**-1**	.....

1..... 500..... 1000.....20000

sentinela = -1

Pesquisar a chave = 500

Endereço = chave = 500

A[500] = 500 → Pesquisa c/ sucesso ⇒ 1 comparação!

Pesquisar a chave = 1000

Endereço = chave = 1000

A[1000] = -1 → Pesquisa s/ sucesso ⇒ 1 comparação!

Hashing

- **Binómio Espaço-Tempo (cont.)**

⇒ Se não há limitações de tempo

↓

Usar uma Função de Hash para obter o endereço na tabela, a qual será mais pequena

↓

Há o risco de colisões (função não é perfeita)

↓

Pesquisa sequencial → Tempo ↗↗

Hashing

(chaves são números inteiros > 0)

- **Por exemplo:** **A =**

.....	500	2001	-1	.....
1.....	501	502	503	1500

H (chave) = (chave mod 1500) +1

⇒ Pesquisar a chave = 500

↓

Endereço = H(500) = 500 mod 1500 + 1 = 501

↓

A[501] = 500 → Pesquisa c/ sucesso ⇒ 1 comparação!

⇒ Pesquisar a chave = 3500

↓

Endereço = H(3500) = 3500 mod 1500 + 1 = 501

↓

A[501] = 500 (ocupado)

Tentar a seguinte (método exploração linear)

↓

A[502] = 2001 (ocupado)

↓

A[503] = -1 → Pesquisa s/ sucesso ⇒ 3 comparações!

Hashing - Colisões

➢ **Encadeamento separado**

- Construir uma lista para cada endereço da tabela com os registos que provocaram colisões (originaram o mesmo endereço)
  - se as listas estiverem ordenadas a pesquisa é uma generalização

H(K) = K mod 11

CHAVE:	A	S	E	A	R	C	H	I	N	G	E	X	A	M	P	L	E
HASH:	1	8	5	1	7	3	8	9	3	7	5	2	1	2	5	1	5

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	A	M	C		E		G	H	I	
	A	X	N		E		R	S		
	A				E					
	L				P					

Hashing - Colisões

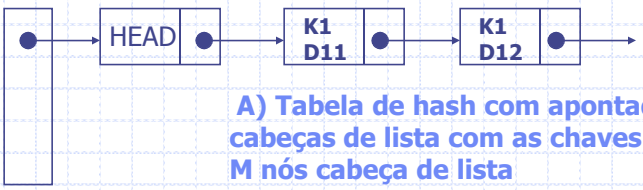
➢ **Encadeamento separado**

- **Tempo de pesquisa depende do tamanho das listas**
- **Se listas forem curtas podem estar desordenadas**
- **Se N>>M, o comprimento médio das listas é N/M**

N = número de chaves da tabela
M = tamanho da tabela

O Hashing reduz de um factor de M o tempo da pesquisa, face à pesquisa sequencial

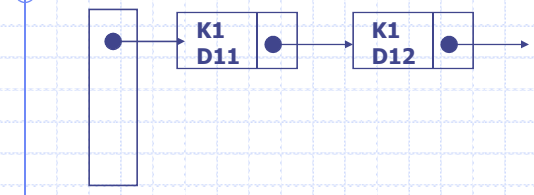
➢ **Implementação**

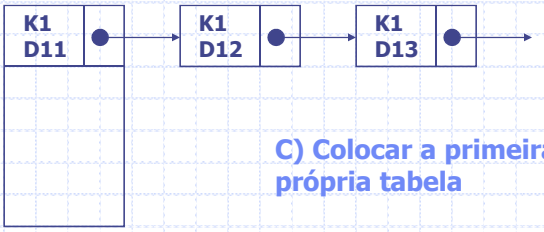


A) Tabela de hash com apontadores para cabeças de lista com as chaves reais
M nós cabeça de lista

Hashing - Colisões

B) Tabela de apontadores para as primeiras chaves da lista





C) Colocar a primeira chave na própria tabela

Hashing - Colisões

➢ **Endereçamento aberto**

- **Se M (tamanho tabela) > N (nº de elementos) usam-se os espaços não ocupados na tabela para resolver as colisões**
- **Método mais simples endereçamento aberto → Exploração Linear**

Quando ocorre colisão
Explora-se a posição seguinte da tabela

a) **Se CHAVE PESQUISA = CHAVE REGISTO**
   Então termina com sucesso – ENCONTROU
b) **Se não existe registo (posição vazia)**
   Então termina sem sucesso – NÃO ENCONTROU
c) **Explora-se a posição seguinte até que chave seja encontrada, ou apareça uma posição vazia**

Hashing - Colisões

➢ **Endereçamento aberto**

- **Se o registo que contém a chave de pesquisa deve ser colocado após uma pesquisa sem sucesso, então pode simplesmente ocupar o espaço vazio que fez terminar a pesquisa**
- **O endereçamento aberto não é indicado quando a tabela começa a ficar cheia (N ≈ M), pois é necessário efectuar muitas comparações por os registos estarem aglomerados – CLUSTERING**

Solução: HASH-DUPLO

Em vez de se examinar as entradas seguintes, há outra função de Hash H2(K) para as obter

Hashing - Colisões

➢ **Supressão de chaves**

Se ocorrerem colisões com a chave que se quer retirar, futuras pesquisas vão terminar no espaço vazio

Solução:

Chave especial que faça a pesquisa continuar, mas que seja lembrada como disponível

Hashing - Colisões

- **Supressão de chaves**
  - **Sem problemas no encadeamento separado**
  - **Com problemas no endereçamento aberto**

Exemplo: A =

H(T) = 6
H(A) = 8
H(K) = 7
H(Z) = 8
H(I) = 6

Hashing - Colisões

- **Eliminar T**

- **Pesquisar I**

H(I) = 6

Vazio → Não existe

ERRADO!

- **Solução: chave especial**

Vazio para inserções

Ocupado para pesquisas

- Eliminar T