Estruturas de dados para listas
– arrays e listas ligadas –

Fernando Lobo

Algoritmos e Estrutura de Dados

— Algumas figuras retiradas do livro Introduction to Algorithms, 3rd Edition. —

Listas

- Uma lista é um conceito matemático.
- Iremos ver duas estruturas de dados que são usadas para implementar listas.
  - Arrays
  - Listas ligadas

Definições

- Uma lista é uma sequência de zero ou mais elementos, geralmente do mesmo tipo.
  - lista de inteiros, lista de reais, lista de caracteres, lista de strings, lista de alunos, lista de lista de inteiros, etc.
- Há uma ordem implícita estipulada pela posição dos elementos na lista.
- Notação usual: $(a_1, a_2, \ldots, a_n)$
  - a_1 é o primeiro elemento
  - a_2 é o segundo elemento
  - $\ldots$
- Pode ter elementos duplicados. Ex: $(5, 8, 7, 5, 2)$

Definições

- Comprimento: número de elementos da lista.
- ϵ é a lista vazia. O seu comprimento é 0.
- Sublista
  - Seja $L = (a_1, a_2, \ldots, a_n)$. Se $1 \leq i \leq j \leq n$, então $(a_i, \ldots, a_j)$ é uma sublista de L.
  - ϵ é sublista de qualquer lista.
  - Exemplo: seja $L = (1, 2, 4)$
    - ★ Sublistas: ϵ, (1), (2), (4), $(1, 2)$, $(2, 4)$, $(1, 2, 4)$.

Operações habituais

- $First(L) \rightarrow$ retorna o 1º elemento da lista L (dá erro se $L = \epsilon$)
- $Last(L) \rightarrow$ retorna o último elemento da lista L (dá erro se $L = \epsilon$)
- $Length(L) \rightarrow$ retorna o comprimento de L.
- $Retrieve(L, i) \rightarrow$ retorna o elemento que está na posição i.
  - dá erro se $Length(L) < i$
  - $First(L) = Retrieve(L, 1)$
  - $Last(L) = Retrieve(L, Length(L))$
- $isEmpty(L) \rightarrow$ retorna TRUE se $L = \epsilon$, retorna FALSE caso contrário.

Operações habituais

- $Concat(L, M) \rightarrow$ retorna uma nova lista que é o resultado da concatenação de L com M.
  - Se $L = (a_1, a_2, \ldots, a_n)$ e $M = (b_1, b_2, \ldots, b_m)$, $Concat(L, M) = (a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m)$
  - $Concat(L, \epsilon) = Concat(\epsilon, L) = L$

Listas e conjuntos

- Um conjunto não tem elementos repetidos; uma lista pode ter.
- Num conjunto a ordem dos elementos não interessa; numa lista interessa.
- Mas uma lista pode ser usada para implementar um conjunto. Como?
  - ignorando os repetidos e a ordem
  - Ex: $(5, 8, 5, 2, 2, 2)$ representa o conjunto $\{2,5,8\}$.

Listas e dicionários

- Um *dicionário* é um conjunto de elementos sobre o qual podemos executar as seguintes operações.
  - $Search(D, x) \rightarrow$ verifica se x é membro do conjunto D
  - $Insert(D, x) \rightarrow$ acrescenta o elemento x ao conjunto D
  - $Delete(D, x) \rightarrow$ remove o elemento x do conjunto D
- Uma vez que uma lista pode representar um conjunto, também pode ser usada para implementar um dicionário.

Implementação das operações do dicionário usando uma lista L

- *Search*(L, x)
  - Retorna TRUE se L contém o elemento x, retorna FALSE caso contrário.
- *Delete*(L, x)
  - Remove todas as ocorrências de x da lista L.
- *Insert*(L, x)
  - Duas alternativas:
    1. Verificar se x existe na lista (com *Search*). Se sim, não faz nada. Se não, insere x numa qualquer posição da lista.
    2. Insere x numa qualquer posição da lista. (Se já existir, não importa; é inserido uma nova ocorrência de x)

Onde inserir?

- Seja $L = (a_1, a_2, \ldots, a_n)$
- O mais habitual é inserir ao início ou ao fim
  - início: $(x, a_1, \ldots, a_n)$
  - fim: $(a_1, \ldots, a_n, x)$
- mas nada impede que se insira numa qualquer posição i,
  - $(a_1, \ldots, a_{i-1}, x, a_i, \ldots, a_n)$

Outras operações assumindo a existência de uma relação de ordem total entre os elementos do conjunto

- $Minimum(L) \rightarrow$ retorna o elemento mínimo de L.
- $Maximum(L) \rightarrow$ retorna o elemento máximo de L.
- $Predecessor(L, x) \rightarrow$ retorna o elemento de L imediatamente antes de x na relação de ordem estipulada.
- $Successor(L, x) \rightarrow$ retorna o elemento de L imediatamente após x na relação de ordem estipulada.

Extensão para sacos (bags)

- As operações que vimos são facilmente adaptadas para o caso de lidarmos com sacos em vez de conjuntos.
- Um saco pode ter elementos repetidos.
- Tal como nos conjuntos, a ordem não interessa.
- Saco $(1, 1, 3, 2) =$ Saco $(3, 1, 2, 1)$

Modelo vs. Implementação

- Até agora só falamos de listas (e de dicionários) de forma abstracta, como um modelo matemático.
- Veremos agora como podemos implementar este conceito de forma concreta.

Representação dos elementos da lista

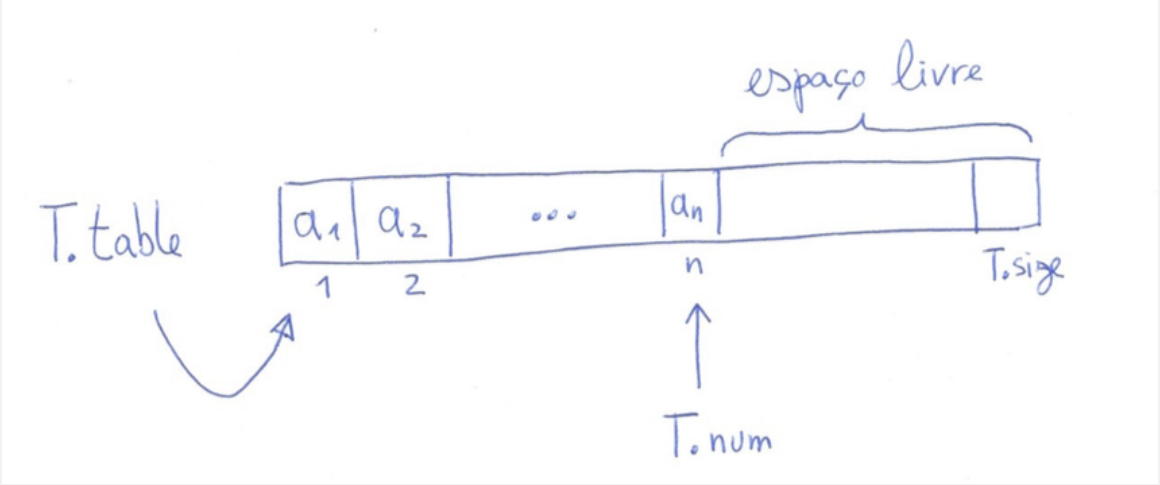
- Vamos assumir que cada elemento é representado por um objecto com vários atributos que podem ser examinados se tivermos um apontador/referência para esse objecto.
- Um desse atributos chama-se **key** e permite-nos identificar o objecto.
- Os restantes atributos do objecto não são usados pela implementação.

Estruturas de dados para implementar listas

- Duas estruturas de dados fundamentais para implementar listas:
  - Arrays
  - Listas ligadas
- Array $\rightarrow$ pedaço de memória contíguo com um determinado número fixo de posições.
- Lista ligada (Linked List) $\rightarrow$ pedaços de memória não contíguos ligados entre si por apontadores/referências.

Implementação baseada num array

- Necessitamos de um objecto T com três atributos:
  - um array $T.table$ de dimensão $T.size$ para guardar os elementos.
  - $T.num$ para sabermos quantos elementos a lista tem.
- O i-ésimo elemento da lista está na posição i do array.

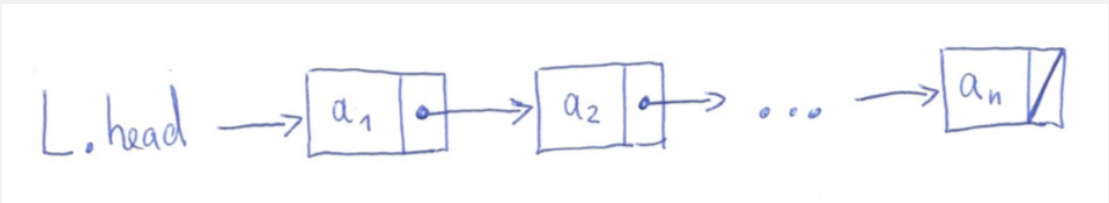


Arrays: vantagens e desvantagens

- Vantagens
  - Acesso ao i-ésimo elemento da lista é feito em O(1).
  - Memória contígua pode ser explorada pela arquitectura do computador (caching).
- Desvantagens
  - Dimensão máxima pré-definida. (Este problema pode ser resolvido, ver próxima aula.)
  - Inserção e remoção são ineficientes, excepto se for ao final da lista. Numa posição arbitrária requer tempo $\Theta(n)$.

Implementação baseada em listas ligadas

- Uma lista ligada é uma sequência de nós.
- Cada nó tem 2 atributos:
  - um elemento da lista.
  - um apontador/referência para o próximo nó da lista ligada. (Não havendo próximo, aponta para NIL)
- A lista ligada tem um atributo *head* que aponta para o primeiro nó.

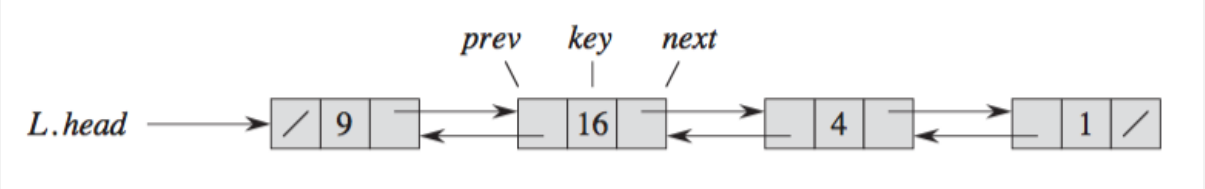


Implementação baseada em listas ligadas

- Ao invés da implementação com array, a posição de um elemento da lista não é dada por um índice.
- A posição é dada pelo número de apontadores que é necessário percorrer desde o inicio da lista ligada.

Variante: lista duplamente ligada

- A definição que vimos atrás é para listas *simplesmente ligadas*.
- Uma variante comum são as listas *duplamente ligadas*.
  - cada nó passa a ter 2 apontadores, um para o nó seguinte (*next*), outro para o nó anterior (*prev*)
  - permite percorrer a lista nos dois sentidos.



Pseudocódigo para as operações de dicionário: Search

- LIST-SEARCH(L, k) faz uma pesquisa sequencial desde o início e retorna um apontador para o primeiro elemento de L que contenha a chave k. Se não encontrar, retorna NIL.
- Numa lista com n elementos, a complexidade no pior caso é $\Theta(n)$.

LIST-SEARCH(L, k)

```
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3        x = x.next
4  return x
```


Pseudocódigo para as operações de dicionário: Insert

- LIST-INSERT(L, x) insere x à cabeça da lista ligada. Assume que o objecto x já tem o atributo *key* inicializado.
- A complexidade é O(1).

LIST-INSERT(L, x)

```
1  x.next = L.head
2  if L.head ≠ NIL
3        L.head.prev = x
4  L.head = x
5  x.prev = NIL
```

Pseudocódigo para as operações de dicionário: Delete

- LIST-DELETE(L, x) remove o elemento x da lista ligada. Assume que a operação recebe um apontador para x.
- A complexidade é O(1).
- Contudo, se tivéssemos de remover um elemento com uma determinada *key*, teríamos primeiro de chamar LIST-SEARCH e a complexidade já seria $\Theta(n)$.

LIST-DELETE(L, x)

```
1  if x.prev ≠ NIL
2        x.prev.next = x.next
3  else L.head = x.next
4  if x.next ≠ NIL
5        x.next.prev = x.prev
```


Mais variantes

- O código pode ficar mais simples se usarmos listas ligadas com *sentinelas*
  - Podem ver detalhes no livro recomendado, cap. 10.
- Outra variante: listas circulares
  - o último nó está ligado com o primeiro nó da lista ligada.
  - a lista ligada passa a ser um anel de nós.
- Se existir uma relação de ordem total entre os elementos, podemos querer manter a lista sempre ordenada.
  - a operação LIST-INSERT(L, x) passaria a ter complexidade $\Theta(n)$.

Outras operações

- Se existir uma relação de ordem total entre os elementos, pode-se implementar as operações:
  - $\textsc{Minimum}(L)$
  - $\textsc{Maximum}(L)$
  - $\textsc{Predecessor}(L, x)$
  - $\textsc{Successor}(L, x)$
- Deixa-se como exercício.

Listas ligadas: vantagens e desvantagens

- Vantagens
  - Não há dimensão máxima pré-definida; o limite de espaço é o limite da memória do computador.
  - Inserir e remover elementos são operações que podem ser feitas de forma mais eficiente.
- Desvantagens
  - Acesso directo não é eficiente.
  - Requer espaço adicional para os apontadores.
  - Não tira partido da localidade da memória.

Vamos resolver este exercício (CLRS, problem 10.1)

10-1 Comparisons among lists

For each of the four types of lists in the following table, what is the asymptotic worst-case running time for each dynamic-set operation listed?

	unsorted, singly linked	sorted, singly linked	unsorted, doubly linked	sorted, doubly linked
SEARCH(L, k)				
INSERT(L, x)				
DELETE(L, x)				
SUCCESSOR(L, x)				
PREDECESSOR(L, x)				
MINIMUM(L)				
MAXIMUM(L)				

O mesmo com arrays

- Repetir o exercício anterior no caso da lista ser implementada à custa de um array.
- Considerar 2 variantes: array desordenado, array ordenado.

Java

- O Java vem como uma implementação de listas duplamente ligadas (java.util.LinkedList)
- O livro "*Algorithms in Java*" de R. Sedgewick (há cópia na biblioteca), ou o mais recente "*Algorithms*" do mesmo autor juntamente com K. Wayne, são referências úteis no que diz respeito a implementação em Java.
  - Ver `https://algs4.cs.princeton.edu/home/`