Elementary Data Structures

Pramesh Kumar

IIT Delhi

January 4, 2024

Introduction

A data structure is a way of storing and manipulating data within computer memory.

Outline

Array

Stack

Queue

Linked list

Conclusions

Array

- Used for storing an ordered set.
- Each element of the set is identified using a key or index.
- Each element occupy same memory size.

2	8	4	5	1	·	·	·	·	·	·	·	·	·

- Accessing element can be performed in $O(1)$ time. For example, for k^{th} element we just use $array[k]$.
- Deleting an element may require as many operations as the length of the array.

Matrix

- A matrix is a two-dimensional array where data is stored in form of rows and columns.

Stack

- A special kind of ordered list in which all insertions and deletions take place at one end called the top.
- It follows LIFO order for operations.
- Checking if Stack is empty can be performed in $O(1)$ time.

```
procedure STACK_EMPTY($S$)
    if $top(S) == 0$ then:
        return TRUE
    else
        return FALSE
    end if
end procedure
```

Outline

- Array
- Stack
- Queue
- Linked list
- Conclusions

Stack

- Inserting or pushing an element e can be performed in $O(1)$ time.

```
procedure STACK_PUSH($S, e$)
    if $top(S) == size(S)$ then:
        error "overflow"
    else
        $top(S) = top(S) + 1$
        $A[top(S)] = e$
    end if
end procedure
```

- Deleting or popping can also be performed in $O(1)$ time.

```
procedure STACK_POP($S$)
    if STACK_EMPTY($S$) ==TRUE then:
        error "underflow"
    else
        $top(S) = top(S) - 1$
        return $S[top(S) + 1]$
    end if
end procedure
```

Outline

Array

Stack

Queue

Linked list

Conclusions

Queue

- A special kind of list in which elements are inserted at one end called rear and deleted from the other end front.
- It follows FIFO order for operations.
- Checking if the queue is empty can be performed in $O(1)$ time.

```
procedure QUEUE_EMPTY($Q$)
    if $rear(Q) == size(Q)$ then:
        return TRUE
    else
        return FALSE
    end if
end procedure
```

Queue

- Inserting an element or enqueue x can be performed in $O(1)$ time.

```
procedure Enqueue($Q, x$)
    $Q[rear(Q)] = x$
    if $rear(Q) == size(Q)$ then
        $rear(Q) = 1$
    else
        $rear(Q) = rear(Q) + 1$
    end if
end procedure
```

- Deleting an element or dequeue can be performed in $O(1)$ time.

```
procedure Dequeue($Q$)
    $x = Q[front(Q)]$
    if $front(Q) == size(Q)$ then
        $front(Q) = 1$
    else
        $front(Q) = front(Q) + 1$
    end if
end procedure
```

Outline

Array

Stack

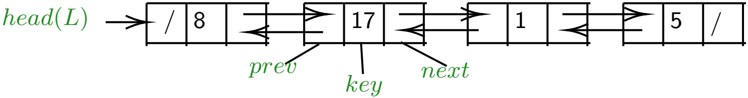
Queue

Linked list

Conclusions

Linked list

- A linked list is a data structure in which the objects are arranged in linear order which is determined by a pointer in each object.
- Two types
  - Doubly linked list: Each element is an object with an attribute key and two pointer attributes next and prev. Given an element x, $next(x)$ points to its successor in the linked list whereas $prev(x)$ points to its predecessor in the linked list. If $prev(x) = \texttt{NIL}$, then x is the first element. Similarly, if $next(x) = \texttt{NIL}$, then x is the last element of the linked list. $head(L)$ points to the first element of linked list L.



  - Singly linked list: Unlike doubly linked list, it has only one pointer attribute $next$.

Doubly linked list

- Searching an element with key k in doubly linked list L of size n can be performed in $\Theta(n)$ time.

```
procedure List_Search($L, k$)
    $x = head(L)$
    while $x \neq$ NIL and $key(x) \neq k$ do
        $x = next(x)$
    end while
end procedure
```

- Inserting an element x to the front of the linked list can be done in $O(1)$ time.

```
procedure List_Prepend($L, x$)
    $next(x) = head(L)$
    $prev(x) =$ NIL
    if $head(L) \neq$ NIL then
        $prev(head(L)) = x$
    end if
    $head(L) = x$
end procedure
```

Doubly linked list

- Inserting an element x immediately following y can be performed in $O(1)$ time.

```
procedure LIST_INSERT($x, y$)
    $next(x) = next(y)$
    $prev(x) = y$
    if $next(y) \neq \texttt{NIL}$ then
        $prev(next(y)) = x$
    end if
    $next(y) = x$
end procedure
```

- Deleting an element x can be done in $O(1)$ time.

```
procedure LIST_DELETE($L, x$)
    if $prev(x) \neq \texttt{NIL}$ then
        $next(prev(x)) = next(x)$
    else
        $head(L) = next(x)$
    end if
    if $next(x) \neq \texttt{NIL}$ then
        $prev(next(x)) = prev(x)$
    end if
end procedure
```

Outline

Array

Stack

Queue

Linked list

Conclusions

Conclusions

- Implementing an algorithm using efficient data structures can make a lot of difference in the running time of the algorithm.
- We studied a few elementary data structures.
- I encourage you to study other data structures such as binary search tress, red-black trees, hash tables, different types of heaps, etc.

Suggested reading

1. CLRS Chapter 10
2. AMO Appendix A

Thank you!