Elementary Data Structures
Chapter 11

CSCE310: Data Structures and Algorithms
www.cse.unl.edu/~choueiry/S01-310/

Berthe Y. Choueiry (Shu-we-ri)
Ferguson Hall, Room 104
choueiry@cse.unl.edu, Tel: (402)472-5444
Sets are fundamental in CS

Sets are dynamic structures:
  they grow, shrink, change over time

Typical operations:
  • insert element
  • delete element
  • test membership
Objects:

1. key $\rightarrow$ dynamic set as a set of key values
2. satellite data (unused information)

Two types of operations:

1. Queries: \texttt{Search}(S, k), \texttt{Minimum}(S), \texttt{Maximum}(S),
   \texttt{Successor}(S, k), \texttt{Predecessor}(S, k)
2. Modifications: \texttt{Insert}(S, k), \texttt{Delete}(S, k)

Time to execute a set operation generally measured in terms of the size of the set given as one of its parameters.
Dynamic sets as simple data structures using pointers

- Stacks
- Queues
- Linked lists
- Rooted trees
- etc.

Assume: you are familiar with arrays
Stack and queues

Delete(S, k) ≡ Delete(S), deleted element is pre-specified

In stacks: most recent element entered, LIFO
In queues: oldest element entered, FIFO

Implementation: e.g., array
Stack

Insert operation: \textbf{Push} \( (S, x) \)
Delete operation: \textbf{Pop} \( (S) \)

Image: spring-loaded stacks of plates in cafeteria
Order popped is reverse of order pushed (LIFO)

![Diagram of stack operations](image)

\textbf{Figure 11.1} An array implementation of a stack \( S \). Stack elements appear only in the lightly shaded positions. (a) Stack \( S \) has 4 elements. The top element is 9. (b) Stack \( S \) after the calls \textbf{Push}(\( S, 17 \)) and \textbf{Push}(\( S, 3 \)). (c) Stack \( S \) after the call \textbf{Pop}(\( S \)) has returned the element 3, which is the one most recently pushed. Although element 3 still appears in the array, it is no longer in the stack; the top is element 17.

For a stack of \( n \) elements (at most): \( S[1..n] \)
For a stack of $n$ elements (at most): $S[1..n]$

Attributes of $S$: $\text{top}$

$\text{top}[S]$: index of most recently inserted element

Stacks elements: $S[1.. \text{top}[S]]$

Bottom of stack: $S[1]$

Top of stack: $S[\text{top}[S]]$

Empty stack: $\text{top}[S] = 0$
Test for emptiness: Stack-empty(S)

1. if top(S) = 0 then return TRUE
2. else return FALSE

Popping on empty stack: underflow

1. top(S) = top(S) + 1
2. S[top(S)] = x

Popping on full stack: overflow (top[S] > n)

1. if Stack-Empty(S) then error "underflow"
2. else top[S] = top[S] - 1
3. return S[top(S) + 1]

Pseudo-code for Push(Pop) increment (decrement) counter
Queues

Insert operation: Enqueue (S, x)
Delete operation: Dequeue (S)

Image: line of people at the post office
Order popped is reverse of order pushed (LIFO)

(a) \(Q\)

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
15 & 6 & 9 & 8 & 4 \\
\end{array}
\]

\(\text{head}[Q] = 7\)
\(\text{tail}[Q] = 12\)
Attributes of $S$: head and tail

Enqueued: added at the tail, dequeued: always from head

Elements in queue are in positions:

$$\text{head}[Q], \text{head}[Q]+1, \ldots, \text{tail}[Q]-1$$
Wrap around (circular): position 1 directly follows position \( n \)

Empty queue: \( \text{head}[Q] = \text{tail}[Q] \)

Full queue: \( \text{head}[Q] = \text{tail}[Q] + 1 \)

Initially: \( \text{head}[Q] = \text{tail}[Q] = 1 \)
Dequeue empty list: underflow

\texttt{Dequeue}(Q)
1 \ x \leftarrow Q[\text{head}[Q]]
2 \ \textbf{if} \ head[Q] = length[Q]
3 \ \ \textbf{then} \ head[Q] \leftarrow 1
4 \ \ \textbf{else} \ head[Q] \leftarrow head[Q] + 1
5 \ \textbf{return} \ x

Enqueue full queue: overflow

\texttt{Enqueue}(Q, x)
1 \ Q[\text{tail}[Q]] \leftarrow x
2 \ \textbf{if} \ tail[Q] = length[Q]
3 \ \ \textbf{then} \ tail[Q] \leftarrow 1
4 \ \ \textbf{else} \ tail[Q] \leftarrow tail[Q] + 1

Pseudo-code for \texttt{Enqueue(Dequeue)} increment (decrement) tail (head)
Linked list

Objects arranged in a linear order according to a pointer in each object

(a) $head[L]$ → \[ \begin{array}{c} 9 \end{array} \] → \[ \begin{array}{c} 16 \end{array} \] → \[ \begin{array}{c} 4 \end{array} \] → \[ \begin{array}{c} 1 \end{array} \]

(b) $head[L]$ → \[ \begin{array}{c} 25 \end{array} \] → \[ \begin{array}{c} 9 \end{array} \] → \[ \begin{array}{c} 16 \end{array} \] → \[ \begin{array}{c} 4 \end{array} \] → \[ \begin{array}{c} 1 \end{array} \]

(c) $head[L]$ → \[ \begin{array}{c} 25 \end{array} \] → \[ \begin{array}{c} 9 \end{array} \] → \[ \begin{array}{c} 16 \end{array} \] → \[ \begin{array}{c} 1 \end{array} \]

Each element $x$:

- 1 key field
- 2 pointers fields ($next[x], prev[x]$)
First element, head: \( \text{prev}[x] = \text{nil} \)

Last element, tail: \( \text{next}[x] = \text{nil} \)

Attribute \( \text{Head}[L] \) points to first element in list

Empty list: \( \text{Head}[L] = \text{nil} \)
Simply linked list (no prev) vs. doubly linked list
Circular vs. non-circular list
Sorted list (vs. unsorted)

- Linear order of items
- Minimum item is head, maximum item is tail
- Linear order of elements
- Prev of head is tail, and next of tail is head