UNDERGRADUATE LEARNING SERIES / UNIT I

DATA STRUCTURES FUNDAMENTALS

Algorithms से Stack-Queue तक

सरल Hindi-English व्याख्या के साथ

14 अध्यायों में Unit I की व्यवस्थित शिक्षा

ARRAYS | LINKED LISTS | STACK / QUEUE

Hindi + English Edition

Concepts · Diagrams · Numericals · MCQs

पुस्तक का परिचय

Unit I / How to Use This Book

यह पुस्तक आपके दिए syllabus पर आधारित है: introduction, algorithm fundamentals, primitive और non-primitive data types, ADT, linear/nonlinear classification, arrays एवं types, memory allocation/address calculation, sparse array/matrix, linked-list types एवं INSERT/DELETE/TRAVERSE operations, stack और queue का परिचय एवं applications.

Technical terms English में और explanation Hindi-English mix में है. प्रत्येक topic में definition, comparison, example, common traps और practice दिए हैं. 14 अध्याय, conceptual diagrams, solved numericals, pseudocode और 40 MCQs unit को एक साथ cover करते हैं.

Study method
पहले representation draw करें, फिर algorithm dry-run करें. Numericals में base address, bounds और element size अलग लिखें. MCQs answer key देखने से पहले solve करें. 10-mark answer में definition, figure, explanation, example और complexity शामिल करें.

Conventions और Assumptions

Programming language syllabus में specify नहीं है; algorithms language-independent pseudocode हैं, directly runnable code नहीं. Array examples zero-based हैं जब तक bounds explicitly अलग न दिए हों. Address numericals contiguous rectangular storage और supplied element size assume करते हैं. Linked-list diagrams logical references दिखाते हैं; actual addresses नहीं.

Θ tight asymptotic bound है; practical timings hardware और implementation पर depend करते हैं. Allocator costs, runtime headers और alignment simplified memory numericals में omitted हैं जब तक explicitly stated न हों. Empty structures तथा invalid inputs के error policies implementation में define करें.

Syllabus Mapping

Topic	Chapters
Introduction, algorithms, types, ADT, classification	1-4
Arrays, allocation, addresses, sparse matrices	5-8
Linked lists and operations	9-11
Stack and queue	12-13
Revision, MCQs, written practice	14

विषय-सूची

Contents / Chapters 1-14

01. Data Structures का परिचय 4

02. Algorithms के Fundamentals 6

03. Data Types एवं Abstract Data Type 8

04. Linear एवं Nonlinear Classification 10

05. Arrays एवं उनके Types 12

06. Memory Allocation एवं 1-D Address Calculation 14

07. 2-D एवं Multidimensional Address Calculation 15

08. Sparse Array एवं Sparse Matrix 16

09. Linked Lists एवं उनके Types 18

10. Singly Linked List Operations 20

11. Doubly एवं Circular Linked Lists 22

12. Stack का परिचय एवं Applications 24

13. Queue का परिचय एवं Applications 26

14. Revision, Solved Test एवं Exam Practice 28

UNIT I / CHAPTER 01

अध्याय 1: Data Structures का परिचय

Data को व्यवस्थित करके समस्या हल करना

1.1 Data, Information एवं Structure

Data raw values हैं: student names, marks, roll numbers. Meaningful processing के बाद यही information बनता है, जैसे class average या merit list. Data structure values के organization, relationships और operations का तरीका है. केवल values store करना पर्याप्त नहीं; search, update और deletion कैसे होंगे यह भी design का हिस्सा है.

Example: 100 students के marks को 100 अलग variables में रखने के बजाय array में रखें. सभी marks एक loop से traverse हो सकते हैं. यदि records लगातार insert/delete करने हैं और positions known हैं, linked organization useful हो सकती है.

1.2 Basic Operations

Operation	अर्थ	Example
Traverse	प्रत्येक element visit करना	सभी marks print
Search	Required value/key ढूँढना	Roll number खोजें
Insert	नया element जोड़ना	New admission
Delete	Existing element हटाना	Cancelled record
Update	Value बदलना	Correct marks
Sort	निश्चित order बनाना	Marks descending
Merge	Structures combine करना	Two sorted lists

1.3 Logical और Physical View

Logical view बताता है elements आपस में कैसे related हैं. Physical view बताता है representation memory में कैसी है. Linked list logically linear है, लेकिन nodes contiguous addresses पर होना आवश्यक नहीं. Array logical sequence को contiguous equal-size slots से represent करता है.

1.4 Structure कैसे चुनें?

पहले operations और constraints लिखें: क्या index access चाहिए? क्या size पहले पता है? क्या duplicates allowed हैं? Insert/delete कितनी frequent हैं? क्या traversal sequential है? फिर representation चुनें. कोई structure हर problem के लिए best नहीं होती.

Case study: Fixed monthly temperature readings → array. Undo history → stack. Print requests in arrival order → queue. Hierarchical folders → tree. Road connections → graph.

> **Exam point:** Data structure = organized data + relationships + supported operations. Algorithm उसी representation पर काम करने की step-by-step procedure है.

1.5 10-mark Answer Framework

Definition → need → basic operations → classification diagram → two practical examples → choice factors. Array और linked list को केवल “static versus dynamic” कहकर answer समाप्त न करें; access pattern और memory layout भी समझाएँ.

Quick revision: Representation और operation अलग concepts हैं. सही choice time तथा memory दोनों को प्रभावित करती है.

Practice: Library books के लिए तीन required operations लिखें और suitable representation justify करें.

UNIT I / CHAPTER 02

अध्याय 2: Algorithms के Fundamentals

Input से output तक finite procedure

2.1 Algorithm की विशेषताएँ

Algorithm well-defined steps का finite sequence है जो input को required output में बदलता है. Classical algorithm में steps clear हों, terminate करें और effective operations हों. Zero या more inputs और one या more outputs हो सकते हैं. Correctness मतलब हर valid input के लिए specification पूरी होना.

Property	Meaning
Input	Required supplied values
Output	Produced result
Definiteness	हर step unambiguous
Finiteness	Finite number of steps में समाप्त
Effectiveness	Steps practically executable

2.2 Pseudocode Convention

इस पुस्तक में arrays zero-based हैं जब तक lower bound अलग न दिया हो. ← assignment है; = comparison. `NIL` means no node. Ranges inclusive हैं. `for i ← 0 to n-1` में n=0 होने पर zero iterations. Pseudocode किसी specific programming language का runnable program नहीं है.

```
MAXIMUM(A, n)
    require n > 0
    best ← A[0]
    for i ← 1 to n-1
        if A[i] > best
            best ← A[i]
    return best
```

Dry run: A=[12, 7, 19, 4]. best changes 12 → 12 → 19 → 19. Output 19. Exactly n-1 comparisons. Empty input के लिए maximum undefined है; specification में error/optional result देना होगा.

2.3 Time और Space Complexity

Time complexity input size n के साथ operation count का growth है, wall-clock seconds नहीं. Space complexity memory growth है. Auxiliary space input storage के अतिरिक्त working memory है. Maximum algorithm Θ(n) time और Θ(1) auxiliary space लेता है.

Growth	Example	Doubling n का approximate effect
Θ(1)	Valid array index access	लगभग unchanged
Θ(log n)	Binary search	लगभग एक extra level

Growth	Example	Doubling n का approximate effect
Θ(n)	Full traversal	लगभग 2 times
Θ(n²)	Two full nested loops	लगभग 4 times

Big-O asymptotic upper bound, Big-Ω lower bound और Big-Θ tight bound है. O(n) कहना स्वयं “worst case” की definition नहीं है; पहले case specify करें. Linear search best Θ(1), worst Θ(n).

2.4 Correctness की सोच

Maximum loop के बाद processed prefix का largest value best में होना चाहिए. यह loop invariant है. Initialization में first element maximum है; update invariant बनाए रखता है; termination पर prefix पूरा array है.

Practice: Minimum algorithm लिखें. n=5 पर comparisons count करें. O(n²) और Θ(n²) के meanings में अंतर लिखें.

UNIT I / CHAPTER 03

अध्याय 3: Data Types एवं Abstract Data Type

Value, operations और implementation का अंतर

3.1 Data Type

Data type values का domain और permitted operations बताता है. Integer arithmetic, Boolean logical operations और character representation examples हैं. Primitive types language-provided basic types होती हैं; exact names और sizes language/platform के अनुसार बदलती हैं.

Category	Examples	Notes
Primitive/basic	Integer, character, Boolean	Language-dependent
Composite/non-primitive	Array, record, list	Basic values combine कर सकती हैं
User-defined	Student record, custom list	Programmer-defined model

महत्त्वपूर्ण: “int हमेशा 2 bytes” या “हर language में String primitive” जैसे statements सही नहीं हैं. Numerical problem में दिए element size को उपयोग करें.

3.2 Abstract Data Type (ADT)

ADT data values और observable operations को specify करता है, internal representation से independent. Stack ADT में push, pop, peek, isEmpty होते हैं. Stack को array या linked nodes से implement किया जा सकता है; user को same LIFO behavior मिलता है.

3.3 Interface Contract

Operation	Precondition	Result
push(x)	Capacity available if bounded	x नया top
pop()	Stack nonempty	Top remove एवं return
peek()	Stack nonempty	Top return, no removal
isEmpty()	None	Boolean result

Precondition operation से पहले आवश्यक condition है. Postcondition successful operation के बाद guaranteed state है. Implementation contract को satisfy करे.

3.4 Abstraction और Encapsulation

Abstraction essential behavior दिखाती है और representation details छिपाती है. Encapsulation related state और methods को एक unit में रखकर access control देती है. दोनों related हैं पर identical definitions नहीं.

Example: Queue का client enqueue/dequeue call करता है. उसे circular array का front index या node pointer जानने की जरूरत नहीं. Internal implementation बदलने पर observable FIFO contract वही रह सकता है.

3.5 Exam-ready Comparison

Data type: values और operations का general concept. ADT: implementation-independent behavioral specification. Data structure: concrete organization/representation. Algorithm: operations perform करने की procedure.

Practice: List ADT के insertAt, removeAt, getAt contracts लिखें. Empty pop के लिए exception, error status या optional value में से चुना policy स्पष्ट करें.

UNIT I / CHAPTER 04

अध्याय 4: Linear एवं Nonlinear Classification

Sequence, hierarchy और network

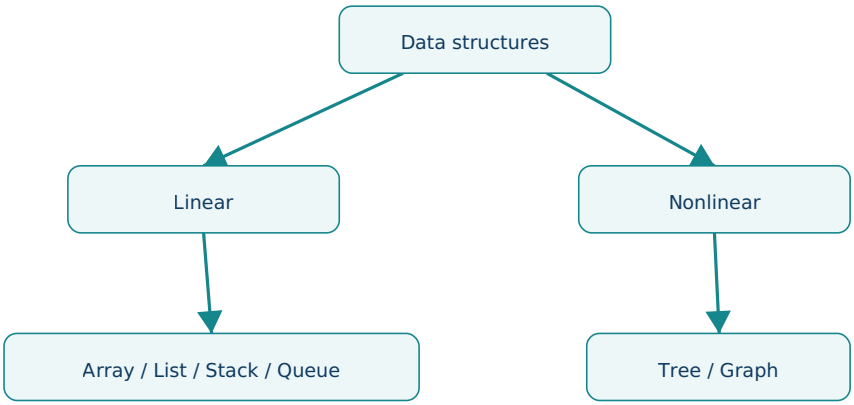

Figure 4.1: Logical classification; representation details may vary.

4.1 Linear Structures

Elements logical sequence में arranged हों. Array, linked list, stack और queue common examples हैं. Middle list element का predecessor और successor होता है; endpoints exceptions हैं. Circular lists में endpoints की definition representation के अनुसार बदलती है.

4.2 Nonlinear Structures

Hierarchy या general network connections होने पर structure nonlinear कहलाता है. Tree parent-child organization देता है. Graph vertices और edges से general relationships model करता है. इस unit में इनका परिचय classification के लिए है, detailed algorithms नहीं.

Aspect	Linear	Nonlinear
Relationship	Sequence	Hierarchy/network
Examples	Array, list, stack, queue	Tree, graph
Traversal	अक्सर sequential	Multiple traversal strategies
Application	Marks, jobs, undo	Folders, routes, dependencies

4.3 Static एवं Dynamic

Fixed-size array की capacity creation पर तय होती है; logical length capacity से छोटी हो सकती है. Linked list nodes runtime पर allocate/release हो सकते हैं. Dynamic arrays भी मौजूद हैं; resizing में नया larger contiguous block allocate और elements copy हो सकते हैं. इसलिए “array कभी dynamic नहीं हो सकता” गलत generalization है.

4.4 Homogeneous एवं Heterogeneous

Elementary typed array में same element type होता है. Record में अलग field types हो सकती हैं: name, roll और marks. कुछ languages flexible containers देती हैं; syllabus classification को selected language context से अलग न करें.

4.5 Choice Exercise

Online helpdesk tickets FIFO queue से process हों. Music playlist linked list या dynamic array से represent हो सकती है, operations पर निर्भर. Undo के लिए stack; organizational chart tree; social connections graph. Application clue से structure identify करें, फिर operations justify करें.

> **Common trap:** Memory addresses का sequence और logical relationship का sequence अलग बातें हैं. Linked list linear है, चाहे nodes scattered हों.

Practice: पाँच real-life examples classify करें. “Stack is always an array” statement सुधारें.

UNIT I / CHAPTER 05

अध्याय 5: Arrays एवं उनके Types

Index-based contiguous representation

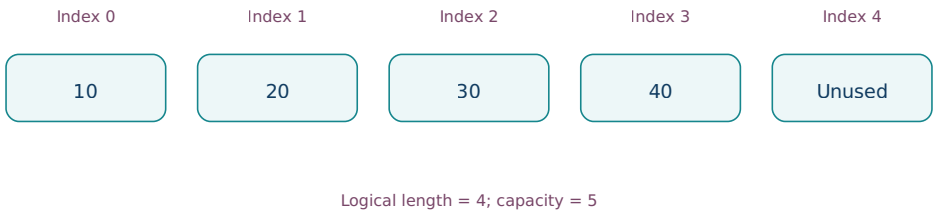

Figure 5.1: Adjacent equal-sized array slots; logical length excludes unused capacity.

5.1 Array Model

Array same-sized elements के contiguous slots की indexed collection है. Physical capacity और logical length अलग हैं. Capacity 8 के array में केवल 5 occupied elements हों तो n=5 है, capacity=8. Unused slots logical sequence का हिस्सा नहीं हैं.

5.2 Types

Type	Logical example	Usage
One-dimensional	A[i]	Marks list
Two-dimensional	A[i,j]	Matrix/table
Multidimensional	A[i,j,k]	Time × row × column
Jagged array	Rows with different lengths	Variable-length row data

Rectangular contiguous matrix की address formulas arbitrary jagged arrays पर लागू नहीं होतीं. कुछ languages multidimensional-looking structures को arrays of references के रूप में रखती हैं. इस पुस्तक के numericals explicitly contiguous rectangular storage assume करते हैं.

5.3 Access एवं Traversal

```
TRAVERSE(A, n)
    for i ← 0 to n-1
        visit A[i]
```

Valid index access Θ(1) है; full traversal Θ(n). Search unsorted array में worst Θ(n). Sorted array पर binary search possible है; sorting और random access assumptions आवश्यक हैं.

5.4 Insertion

```
INSERT(A, n, capacity, pos, value)
    require 0 ≤ pos ≤ n
    if n = capacity
        report overflow
    else
        for i ← n-1 downto pos
            A[i+1] ← A[i]
        A[pos] ← value
        n ← n+1
    return n
```

Dry run: [10,20,30,40], pos=1, value=15. Shift 40,30,20 one slot right. Result [10,15,20,30,40]. n-pos shifts; worst Θ(n), append Θ(1) when capacity available.

5.5 Deletion

```
DELETE(A, n, pos)
    require 0 ≤ pos < n
    removed ← A[pos]
    for i ← pos to n-2
        A[i] ← A[i+1]
    n ← n-1
    return (removed, n)
```

Delete index 2 from [10,15,20,30,40] → [10,15,30,40]. Logical length decreases; physical allocation need not shrink. Order-preserving middle deletion worst Θ(n).

Practice: Capacity 6, n=4: insert at 0 and at n. Count shifts. Delete last element; explain why no shifting occurs.

UNIT I / CHAPTER 06

अध्याय 6: Memory Allocation एवं 1-D Address Calculation

Base address, lower bound और element size

6.1 Contiguous Allocation

If each element occupies w bytes, consecutive element starts differ by w. Base address B is address of first logical element A[L], where L is lower bound. Index distance i-L counts slots, not bytes.

> **Formula:** LOC(A[i]) = B + (i-L) × w. Valid index: L ≤ i ≤ U. Number of elements N=U-L+1. Storage=Nw bytes, excluding allocator metadata/alignment outside this simplified model.

6.2 Solved Example 1

A[0..9], B=1000, w=4. LOC(A[6]) =1000+(6-0)×4 = **1024**. Last element starts at1036; its four bytes occupy1036-1039. Total storage40bytes.

6.3 Solved Example 2: Nonzero Lower Bound

A[-2..7], B=2000, w=2. Find A[3]. Offset=3-(-2)=5 elements. Byte offset10. Address=**2010**. Do not multiply raw index3 when base refers to A[-2].

6.4 Reverse Calculation

B=500, L=1, w=8, given element start address556. i=L+(address-B)/w =1+56/8=**8**. Check divisibility and declared range; unaligned address might not be an element start.

6.5 Allocation Concepts

Compile-time fixed extent and runtime allocated arrays are both possible depending on language. Stack storage, heap storage and static storage are language/implementation matters. Array's core address model is contiguous equal-size slots; it does not dictate one universal allocation region.

Given	Calculation	Answer
B=800, L=0, w=2, i=9	800+9×2	818
B=1200, L=5, w=4, i=8	1200+3×4	1212
Bounds 3..12, w=8	(12-3+1)×8	80 bytes

6.6 Numerical Answer Template

Write B,L,w,index → check range → element offset → byte offset → final address with units. Index itself unitless है; byte address और byte offset अलग values हैं.

Practice: A[4..15], B=3000, w=4: A[11] address? **Answer:** 3028. Total bytes? **Answer:**48.

UNIT I / CHAPTER 07

अध्याय 7: 2-D एवं Multidimensional Address Calculation

Row-major और column-major layouts

7.1 Rectangular Matrix Assumptions

A[Lr..Ur, Lc..Uc] has R=Ur-Lr+1 rows and C=Uc-Lc+1 columns. B=LOC(A[Lr,Lc]); every element w bytes. Row-major stores one full row, then next row. Column-major stores one full column, then next column.

Layout	Formula
Row-major	B + w[(i-Lr)C + (j-Lc)]
Column-major	B + w[(j-Lc)R + (i-Lr)]

7.2 Solved Numerical

A[1..3, 2..5], B=1000, w=4. R=3,C=4. Find A[3,4].

Row-major: offset=(3-1)×4+(4-2)=10. Address=1000+4×10=**1040**.

Column-major: offset=(4-2)×3+(3-1)=8. Address=1000+4×8=**1032**.

Same logical element का address storage layout के अनुसार बदलता है. Total storage=3×4×4=48bytes.

7.3 Zero-based Example

A[4][5], B=200,w=2; zero-based A[2,3]. Row-major offset2×5+3=13 → **226**. Column-major offset3×4+2=14 → **228**.

7.4 3-D Row-major

A[L1..U1,L2..U2,L3..U3], dimensions N1,N2,N3. Last index varies fastest.

```
LOC(A[i,j,k]) = B + w[
    (i-L1) × N2 × N3
    + (j-L2) × N3
    + (k-L3)]
```

Example: zero-based2×3×4, B=100,w=2, element[1,2,3]. Offset=1×12+2×4+3=23 → **146**. Total24elements; last element occupies146-147.

7.5 Common Traps

Use number of columns C in row-major and number of rows R in column-major. Subtract each respective lower bound. Never add element offset directly to byte address unless w=1. Check bounds before calculation. Matrix dimensions denote counts, not always highest indices.

Practice: A[0..2,0..3], B=500,w=8; A[1,2] row-major **548**, column-major **556**. Derive each offset before checking answer.

UNIT I / CHAPTER 08

अध्याय 8: Sparse Array एवं Sparse Matrix

Mostly-zero data का compact storage

8.1 Sparse का Meaning

जब अधिकांश elements zero हों, dense representation unnecessary zeros store करती है. Sparse representation nonzero values और positions store करती है. “Mostly zero” qualitative description है; एक universal percentage threshold नहीं. Representation useful है या नहीं, value/index sizes और access operations पर निर्भर है.

8.2 Example Matrix

Row/Col	0	1	2	3
0	0	0	7	0
1	0	0	0	0
2	5	0	0	9
3	0	0	0	0

16 positions में only3 nonzeros. Zero-based triplets(row,column,value): (0,2,7),(2,0,5),(2,3,9).

8.3 Triplet/COO Representation

Row	Column	Value
4	4	3
0	2	7
2	0	5
2	3	9

इस textbook convention में first row metadata है: total rows, total columns, nonzero count. यह matrix element नहीं है. Other COO formats metadata separately रख सकती हैं; first header row universally mandatory नहीं.

8.4 Space Numerical

Assume हर triplet field4bytes और dense value4bytes. Dense storage=16×4=64bytes. Header+3triplets=(3+1)×3×4=48bytes. Savings16bytes. Real implementation में object/array overhead अलग हो सकता है.

General dense mn values: mnw bytes. Header convention triplets: (t+1)(2s+w) bytes, index size s और value size w. Sparse saves memory when this expression is less than mnw.

8.5 Conversion एवं Transpose

```
TO_TRIPLETS(A, rows, cols)
    T ← empty list
    for r ← 0 to rows-1
        for c ← 0 to cols-1
            if A[r,c] ≠ 0
                append (r,c,A[r,c]) to T
    return (rows, cols, T)
```

Dense-to-triplet scan Θ(rows×cols), output storage Θ(t). Transpose में each(r,c,value) becomes(c,r,value), और dimensions swap होती हैं. Sorted-row format चाहिए तो output reorder करना पड़ सकता है.

8.6 Other Formats एवं Applications

CSR compressed sparse row uses row pointers, column indices and values. COO easy construction देता है; CSR efficient row traversal देती है. Scientific matrices, graph adjacency और document-term data common applications हैं. One-dimensional sparse array भी(index,value) pairs से represent हो सकती है.

Practice: 10×10 matrix with5 nonzeros; all fields4bytes. Dense400, header-triplet72bytes. Explain why sparse access is not automatically Θ(1).

UNIT I / CHAPTER 09

अध्याय 9: Linked Lists एवं उनके Types

Nodes, links और dynamic organization

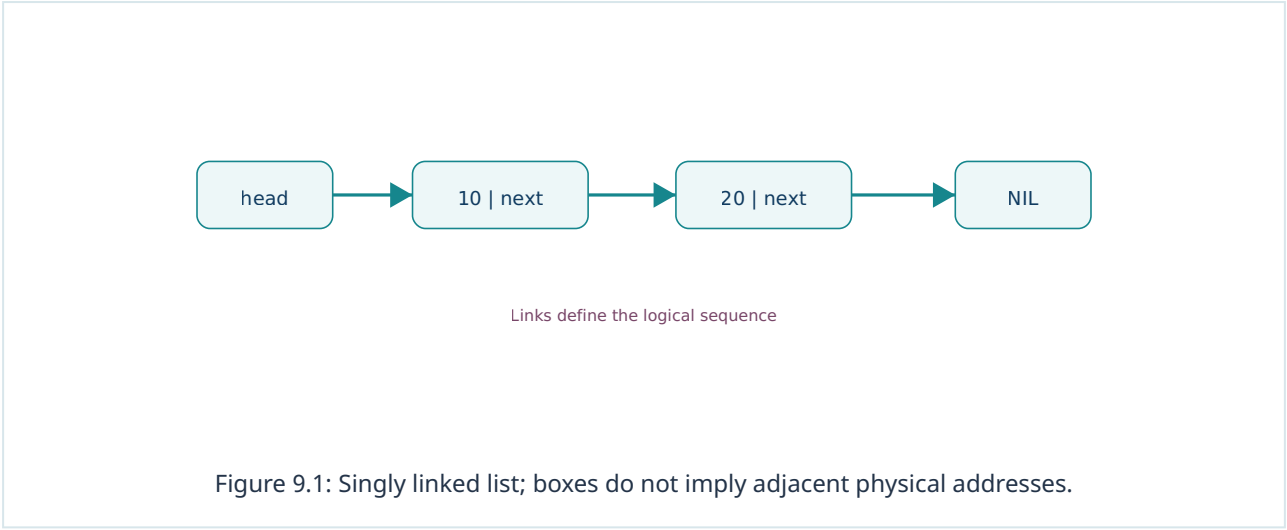

Figure 9.1: Singly linked list; boxes do not imply adjacent physical addresses.

9.1 Node Model

Singly linked node में data और next link होता है. head first node reference है; empty list में head=NIL. Last node का next=NIL. Nodes logically ordered हैं; physical addresses contiguous होने की जरूरत नहीं.

9.2 Types

Type	Links	End condition	Useful property
Singly linked	next	Last next=NIL	Forward traversal
Doubly linked	prev,next	Boundary links=NIL	Bidirectional navigation
Circular singly	next	Last next=head	Repeated cyclic visits
Circular doubly	prev,next	Head/tail links wrap	Both directions cyclic

Header/sentinel lists extra dummy node रख सकती हैं जिससे boundary handling सरल होती है. Header list representation technique है; singly/doubly और circular/noncircular properties के साथ combine हो सकती है.

9.3 Array बनाम Linked List

Aspect	Array	Linked list
Storage	Contiguous slots	Linked nodes
Index access	Θ(1)	Θ(n) traversal
Link overhead	No per-element next link	Pointer/reference overhead
Insert after known node	May require shifting	Θ(1) pointer updates

Aspect	Array	Linked list
Find insertion position	Index arithmetic possible	May require Θ(n) traversal
Locality	Usually good	Often weaker

“Linked-list insertion हमेशा O(1)” अधूरा है. Correct position/node पहले search करने पर total time Θ(n) हो सकती है. Allocation cost और memory failure भी practical matters हैं.

9.4 Space Calculation

Assume data4bytes,next reference8bytes, no padding/header overhead. Singly node simplified12bytes; n nodes12n. Doubly with two references20bytes/node. Actual sizes platform और runtime depend करती हैं.

9.5 Traversal

```
TRAVERSE(head)
    current ← head
    while current ≠ NIL
        visit current.data
        current ← current.next
```

Time Θ(n), auxiliary Θ(1). Empty list loop zero iterations. Accidental cycle होने पर NIL-based traversal terminate नहीं होगी; list invariant important है.

Practice: Draw 10→20→30 with head and NIL. Remove middle node logically; which reference changes? Explain data versus link fields.

UNIT I / CHAPTER 10

अध्याय 10: Singly Linked List Operations

INSERT, DELETE और TRAVERSE

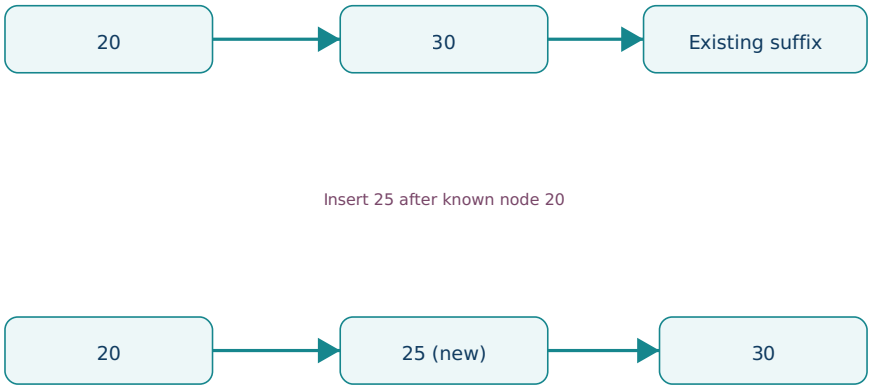

Figure 10.1: Preserve the successor link before reconnecting the predecessor.

10.1 Insert at Beginning

```
INSERT_FRONT(head, value)
    node ← allocate new node
    node.data ← value
    node.next ← head
    head ← node
    return head
```

Works for empty and nonempty lists. Link new node to old head before replacing head. Pointer manipulation Θ(1), assuming successful node allocation.

10.2 Insert After Known Node

```
INSERT_AFTER(p, value)
    require p ≠ NIL
    node ← allocate new node
    node.data ← value
    node.next ← p.next
    p.next ← node
```

Dry run: 10→20→30, p points20, value25. New.next=30, then20.next=new. Result10→20→25→30. Reverse assignment order can lose suffix or form incorrect links.

10.3 Append at End

Without tail pointer, traverse until current.next=NIL: Θ(n) for nonempty list. Empty list is special: new node becomes head. With maintained tail pointer append can be Θ(1); after insertion tail=new, and new.next=NIL. Every deletion must preserve tail invariant too.

10.4 Delete First Node

```
DELETE_FRONT(head)
    if head = NIL
        report underflow
    else
        victim ← head
        value ← victim.data
        head ← victim.next
        release victim if manually managed
        return (head, value)
```

Automatic-memory languages do not require explicit free; discard obsolete references. Pointer update Θ(1). Singleton list becomes empty.

10.5 Delete First Matching Value

```
DELETE_VALUE(head, key)
    prev ← NIL
    current ← head
    while current ≠ NIL and current.data ≠ key
        prev ← current
        current ← current.next
    if current = NIL
        return (head, false)
    if prev = NIL
        head ← current.next
    else
        prev.next ← current.next
    release current if manually managed
    return (head, true)
```

Deletes only first match, duplicates may remain. Search+delete worst Θ(n), auxiliaryΘ(1). Key absent leaves list unchanged. If tail tracked and current=tail, set tail=prev; after deleting sole node both head and tail=NIL.

10.6 Search एवं Position

Zero-based traversal index increments per visited node. getAt(k) Θ(k+1) for valid k, worst Θ(n). Negative/out-of-range positions must be rejected, not silently dereferenced.

Practice: Start empty; insertFront20, insertFront10, insertAfter(first,15), deleteValue20 →10→15. Draw references after every operation.

UNIT I / CHAPTER 11

अध्याय 11: Doubly एवं Circular Linked Lists

Additional links और boundary rules

11.1 Doubly Linked Invariants

For adjacent x and y: x.next=y implies y.prev=x. In noncircular list head.prev=NIL, tail.next=NIL. Empty list has both head and tail=NIL.

11.2 Insert After Known Node p

```
INSERT_AFTER_DOUBLY(p, value)
    require p ≠ NIL
    node ← allocate new node
    node.data ← value
    q ← p.next
    node.prev ← p
    node.next ← q
    p.next ← node
    if q ≠ NIL
        q.prev ← node
    else
        tail ← node
```

Assumes head/tail are maintained in list state. At end q=NIL; no dereference of q. Between20and30, both20.next and30.prev must change.

11.3 Delete Known Node x

Set x.prev.next=x.next if predecessor exists, else head=x.next. Set x.next.prev=x.prev if successor exists, else tail=x.prev. Release/discard x after reading required links. Θ(1) pointer updates when valid node is already known. Searching x still may be Θ(n).

11.4 Circular Singly Traversal

```
TRAVERSE_CIRCULAR(head)
    if head = NIL
        return
    current ← head
    repeat
        visit current.data
        current ← current.next
    until current = head
```

This assumes well-formed cycle that returns to head. Nonempty circular list has no NIL endpoint. Singleton's next points to itself. NIL-terminated loop is wrong for a valid circular list.

11.5 Circular Tail Representation

Store tail, derive head=tail.next. Empty tail=NIL. To append: new.next=tail.next, tail.next=new, tail=new. Empty case: new.next=new,tail=new. Θ(1) link updates. To delete head: for singleton set tail=NIL; otherwise tail.next=head.next.

11.6 Applications एवं Tradeoffs

Doubly list: forward/backward navigation and known-node deletion. Circular list: round-robin scheduling, cyclic playlists. Extra links consume memory and create more consistency requirements. Repeated traversal does not imply “infinite algorithm”: explicit stopping condition is needed.

Practice: Draw circular3-node list and singleton. List four pointer changes needed for middle doubly insertion. Compare circular last node with linear last node.

UNIT I / CHAPTER 12

अध्याय 12: Stack का परिचय एवं Applications

Last In, First Out (LIFO)

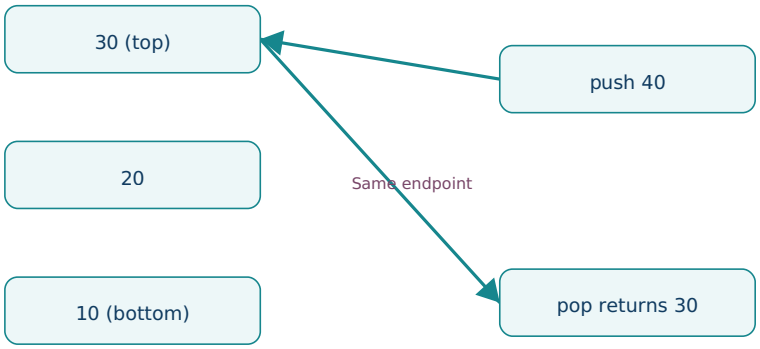

Figure 12.1: Stack snapshot. Push and pop arrows show alternative next operations.

12.1 Stack ADT

Insertion और deletion same endpoint top पर होती हैं. push inserts; pop removes/returns; peek reads top without removal. Stack size and isEmpty observations भी useful हैं. Bounded stack capacity full होने पर overflow; empty pop/peek पर underflow/error policy.

12.2 Array Representation

Use capacity C, array S[0..C-1], top=-1 initially. Nonempty valid top0..C-1. Size=top+1.

```
PUSH(S, x)
    if top = C-1
        report overflow
    else
        top ← top+1
        S[top] ← x

POP(S)
    if top = -1
        report underflow
    else
        value ← S[top]
        top ← top-1
        return value
```

Fixed-capacity array push/pop Θ(1) auxiliary space. Dynamic-array push amortizedΘ(1), occasional resizeΘ(n) rather than every push worstΘ(1). Linked implementation front acts as top; allocation may fail.

12.3 Dry Run

Operation	Bottom→Top	Returned value
push10	[10]	-
push20	[10,20]	-
push30	[10,20,30]	-
pop	[10,20]	30
peek	[10,20]	20
pop	[10]	20

12.4 Applications

Function calls/recursion use call-stack concepts: frames preserve return address, local state and parameters as applicable. Undo operations reverse most recent action. Balanced delimiters push opening symbols and match closing symbols. Expression conversion/evaluation uses stacks. DFS may use explicit stack or recursion.

12.5 Postfix Evaluation

For tokens `2 3 + 4 *`: push2,push3; + pops right3,left2 →push5; push4; * →push20. Result20. For subtraction/division operand order matters: first pop is right operand.

```
On operator op:
    right ← pop()
    left ← pop()
    push(left op right)
```

Malformed expression, insufficient operands and division by zero need validation. Final stack must contain exactly one result.

Practice: Evaluate `8 2 / 3 -` →1. Trace balanced delimiters for `{[()]}` and mismatch `{[(])}`.

UNIT I / CHAPTER 13

अध्याय 13: Queue का परिचय एवं Applications

First In, First Out (FIFO)

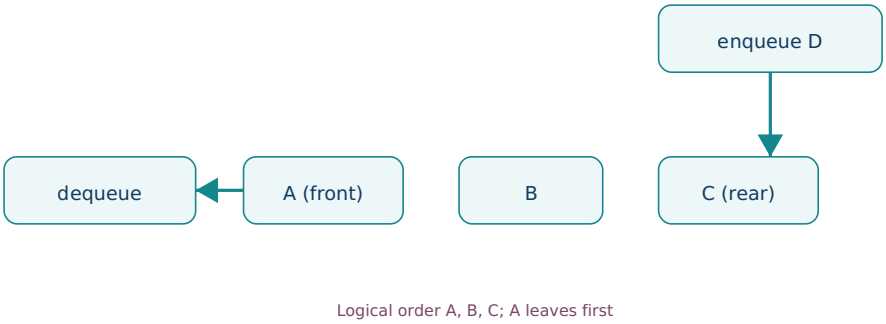

Figure 13.1: FIFO queue; physical circular-array indices may wrap.

13.1 Queue ADT

enqueue rear पर insert करती है; dequeue front से remove करती है. peek/front item read करती है. FIFO oldest still-present item पहले return करता है. Stack and queue both linear ADTs, ordering rules अलग हैं.

13.2 Linear Array Limitation

If front increases after deletion but rear only moves right, free slots at beginning reuse न होने पर false overflow दिख सकती है. Shifting every dequeue fixes reuse but Θ(n) work है. Circular array modulo arithmetic से slots reuse करती है.

13.3 Circular Queue: Count Convention

Use capacity C>0; front=0,rear=0,count=0 initially. rear is next insertion slot, front next removal slot. count disambiguates empty versus full; front=rear alone sufficient नहीं.

```
ENQUEUE(Q, x)
    if count = C
        report overflow
    else
        Q[rear] ← x
        rear ← (rear+1) mod C
        count ← count+1

DEQUEUE(Q)
    if count = 0
        report underflow
```

```
    else
        value ← Q[front]
        front ← (front+1) mod C
        count ← count-1
        return value
```

Both Θ(1). Other conventions reserve one unused slot; उनके full condition `(rear+1) mod C=front` को count convention से mix न करें.

13.4 Dry Run: Capacity 3

Operation	front	rear	count	Logical queue
Initial	0	0	0	[]
enqueue A	0	1	1	[A]
enqueue B	0	2	2	[A,B]
dequeue	1	2	1	[B]
enqueue C	1	0	2	[B,C]
enqueue D	1	1	3	[B,C,D]

Full state front=rear=1,count3; empty initial front=rear=0,count0.

13.5 Linked Queue

Maintain head/front and tail/rear. Enqueue after tail; dequeue head. First insertion sets both references. Removing sole node resets both=NIL. Operations Θ(1) link changes if pointers maintained.

13.6 Applications एवं Variants

Print spool, service requests, buffers और BFS FIFO queues उपयोग कर सकते हैं. Circular queue bounded buffers में useful. Deque permits both endpoints. Priority queue serves by priority, so ordinary FIFO queue के समान contract नहीं है; equal-priority ties may follow FIFO policy.

Practice: Continue dry run with dequeue twice, enqueue E. Result logical queue[D,E], front0,rear2,count2.

UNIT I / CHAPTER 14

अध्याय 14: Revision, Solved Test एवं Exam Practice

पूरे Unit की तैयारी

14.1 Master Comparison

Structure	Access rule	Insert/delete focus	Key application
Array	Index	Shifting may be needed	Fixed indexed readings
Singly list	Follow next	Known-node links	Sequential dynamic list
Doubly list	Both directions	More links update	Navigation
Circular list	Wrap around	Maintain cycle	Round robin
Stack	LIFO	One top endpoint	Undo, expressions
Queue	FIFO	Rear/front	Arrival-order service

14.2 Complexity Revision

Operation	Conditions	Time
Array index read	Valid index, direct array	Θ(1)
Array traversal	n logical elements	Θ(n)
Middle array insert	Preserve order	Worst Θ(n)
List traversal/search	n nodes	Worst Θ(n)
Singly insert after node	Node already known	Θ(1) link changes
Doubly delete node	Node already known	Θ(1) link changes
Stack push/pop	Fixed array or links	Θ(1), ignoring allocation
Circular queue operations	Count-based bounded model	Θ(1)

14.3 Solved Mixed Questions

Q1: Array bounds[-3..6], B=400,w=4; index2? Offset5, address**420**. Total10elements,40bytes.

Q2: A[2..5,1..3], B=100,w=2; A[4,2]? R4,C3. Row offset2×3+1=7,address**114**. Column offset1×4+2=6,address**112**.

Q3: Stack push5,push8,pop,push2,pop,pop returns? **8,2,5**. Final empty. Another pop underflow.

Q4: Queue enqueue5,enqueue8,dequeue,enqueue2,dequeue,dequeue returns? **5,8,2**. Final empty.

Q5: n=100 nodes: insert after already-known node versus insert after searching key? Link update$\Theta(1)$; search+insert worst$\Theta(n)$.

14.4 Unit Test: 40 MCQs

Choose one best answer; solve before checking key.

1. Data structure primarily specifies: A. Data organization B. Only screen color C. Only compiler name D. Only file extension
2. Finite algorithm property: A. No output B. Finiteness C. Infinite looping D. Random ambiguity
3. ADT focuses on: A. Only pointer width B. CPU brand C. Observable behavior D. Fixed machine address
4. Linked list is logically: A. Always nonlinear B. A matrix only C. A compiler D. Linear
5. Tree is commonly: A. Nonlinear B. Stack only C. Queue only D. Primitive integer
6. Full traversal of n array elements: A. $\Theta(\log n)$ always B. $\Theta(n)$ C. $\Theta(1)$ D. $\Theta(n^3)$ necessarily
7. Direct valid array-index access: A. $\Theta(n^2)$ B. Impossible C. $\Theta(1)$ D. $\Theta(n)$ always
8. Array middle insertion preserving order may require: A. No movement ever B. Tree rotation C. Hash removal D. Shifting
9. LOC(A[i]) uses: A. B+(i-L)w B. B+iw always C. B-i-L D. B/w always
10. Bounds 2..8 contain: A. 10 B. 7 elements C. 6 D. 8
11. Row-major formula multiplies row offset by: A. Highest raw index always B. Zero C. Number of columns D. Number of rows
12. Column-major formula multiplies column offset by: A. Number of columns B. Element value C. Stack top D. Number of rows
13. Sparse matrix has: A. Mostly zero entries B. Only ones C. No dimensions D. All entries nonzero necessarily
14. Triplet stores a nonzero as: A. Input,output,time B. Row,column,value C. Head,tail,top D. File,color,width
15. Header in this book's triplet format stores: A. First element always B. CPU address C. Rows,columns,nonzeros D. Only zeros
16. Singly linked node typically contains: A. Only two counters B. Only an index C. No links D. Data,next
17. Empty singly list head is: A. NIL B. Always node zero C. Tail.next always D. Full
18. Last node of noncircular singly list next: A. Last data B. NIL C. Head necessarily D. First value
19. Index access in linked list worst case: A. No traversal B. $\Theta(\log n)$ always C. $\Theta(n)$ D. $\Theta(1)$ always
20. Known-node singly insertion requires: A. All data shifted B. Sorting C. Queue reversal D. Constant link updates
21. Delete first node of singleton list leaves: A. Empty list B. Two nodes C. Full stack D. A tree
22. Doubly node has: A. Four compulsory values B. prev,next links C. Only next D. No references
23. Circular singly last.next points to: A. Stack top always B. A value field C. Head D. NIL always
24. Circular singleton next points to: A. NIL necessarily B. Second node C. Unknown CPU D. Itself
25. Circular traversal should stop when: A. Returns to head B. Always sees NIL C. Visits no node D. Array index negative
26. Stack ordering rule: A. Random-only B. LIFO C. FIFO D. Sorted order
27. Stack insertion operation: A. traverse only B. rotate only C. push D. dequeue
28. Stack pop removes: A. Bottom always B. Every element C. Middle necessarily D. Top
29. Empty pop is: A. Underflow B. Successful insert C. Queue wrap D. Row-major
30. Array stack initial top convention here: A. n+1 B. -1 C. C D. 1 always
31. Undo commonly uses: A. Graph only B. Column-major C. Stack D. Sparse matrix necessarily
32. In postfix subtraction first popped operand is: A. Left always B. Ignored C. Operator D. Right operand
33. Queue ordering: A. FIFO B. LIFO C. Max-first always D. Sorted-only
34. Queue insertion occurs at: A. Any random slot logically B. Rear C. Front always D. Top
35. Queue removal occurs at: A. Top B. Last occupied index always C. Front D. Rear always
36. Circular queue next index uses: A. Data value B. Matrix transpose C. String length only D. Modulo capacity

37. Count-based queue full condition: A. count=C B. count=0 C. front=rear alone D. rear=-1 always
38. front=rear in count-based queue can mean: A. Invalid always B. Empty or full C. Only empty D. Only full
39. Linked queue deleting sole node should: A. Insert two nodes B. Rotate array C. Reset front and rear D. Keep rear dangling
40. Priority queue differs from ordinary queue because: A. Always LIFO B. No items allowed C. It is only a matrix D. Priority determines service

उत्तर-कुंजी:

1-A, 2-B, 3-C, 4-D, 5-A, 6-B, 7-C, 8-D, 9-A, 10-B, 11-C, 12-D, 13-A, 14-B, 15-C, 16-D, 17-A, 18-B, 19-C, 20-D, 21-A, 22-B, 23-C, 24-D, 25-A, 26-B, 27-C, 28-D, 29-A, 30-B, 31-C, 32-D, 33-A, 34-B, 35-C, 36-D, 37-A, 38-B, 39-C, 40-D.

14.5 5-mark Questions

Define ADT with example. Compare primitive and composite types. Explain linear/nonlinear structures. Explain sparse triplets. List stack applications. Explain circular queue wraparound.

14.6 10-mark Questions

1. Data structures: need, operations, classification और examples.
2. Arrays: types, memory allocation, insertion/deletion and address numericals.
3. Derive row-major/column-major formula; solve a nonzero-lower-bound example.
4. Singly linked list insert/delete/traverse with diagrams and edge cases.
5. Doubly and circular linked lists: representation, operations and applications.
6. Stack versus queue with pseudocode, dry run and complexity.

14.7 Mini Practical: Student Marks List

Implement the List operations in a language of your choice. Maintain array capacity10 and logical lengthn. Menu: append mark, insert at position, delete at position, display, maximum. Marks0..100 only; reject invalid inputs. Maximum on empty list should report “No marks”.

Test sequence: append70,append80,insertAt(1,75)→[70,75,80]; deleteAt(0)→[75,80]; maximum80. Then delete all and check empty display. Fill tocapacity10; eleventh insertion reports overflow without corrupting data.

Linked version: Repeat with singly nodes. Compare number of shifts versus link updates, including traversal required to locate a position. Use same zero-based interface so results can be compared.

14.8 Final Checklist

Can you draw each representation? Explain why formulas subtract lower bounds? State complexity assumptions? Handle empty, singleton, full, missing key and invalid position? Trace LIFO versus FIFO? If yes, unit's core concepts are ready for written and MCQ exams.