05/08/2019

Prof Chandan Mazumder

Data Structures and Algorithms

Data Structure and Algorithm Analysis

They should be accurate & trustable
→ Develops Intuition and Knowledge
→ Experiments & Obs., Modeling & Analysis.

→ Collection of properties

3 steps to solve a problem
- → Model → Simpler version of the problem.
- → Algorithm
- → Computer representation of data.

Data Processor → Computer

Input → Output

eg. Metal wire (Raw Data) → [ ] → Paper clip (Processed data (Useful form))

Algorithm

→ Finite sequence of steps to solve a problem

⇓

must be feasible & understandable

Some fact about the surrounding

Information & Data → Abstraction of Information

Objective
Interpretation independent
vs
Subjective
Human Bias plays a role

Google → Circular Defn

→ Height of kamal is 6.5 ft.

↕ Data ↕

Data & Operation

Datatype → ① Range
② Type of operation.

int → subset of integer

+, −, *,

add vs +
overflow can happen.

integer → set of all whole numbers

as a mathematical entity → add, subtract, multiply

Diff → 5/3 = 1.
→ 5.0/3.0 = 1.66667.

Program Structures

Program = Algorithm + Data Structure

Programming Language provides facilities to represent algorithms & data.
↓
structured constructs

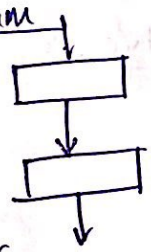
FORTRAN, → Formula Translation.

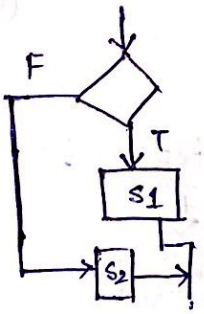
Algol
PASCAL
C

BASIC,
COBOL

1. Sequence
2. Conditional.
3. Subprogram
4. Iteration

Function

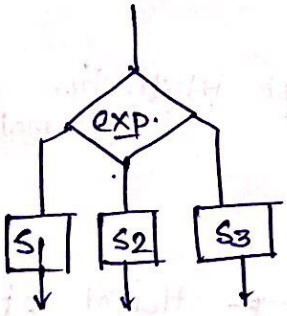




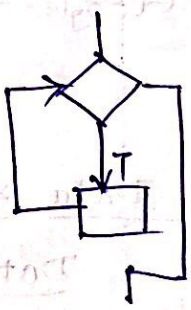
] conditional

```
if (condition)
    s1
else
    s2
```

5. Recursion
6. Control transfer
   ↓
   goto, break, continue, return,
   ↓
   considered harmful.



while (cond).

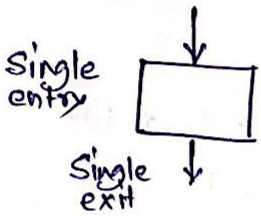


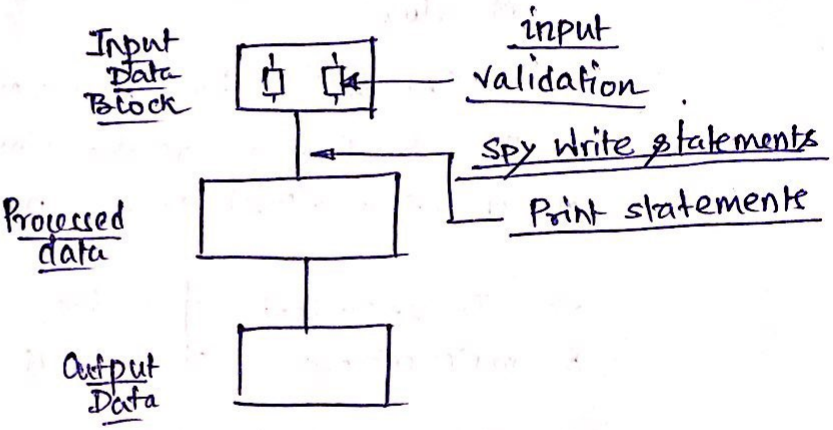

```
do
{
}
while(cond);
```

04/08/2019

Data Structures & Algorithms

Prof. Chandan Mazumder





Data Types

Scalar

Integer Real character ...
Pointer subrange

LISP, Prolog } AI

Data Aggregation Facilities

Arrays Records Sets

Arrays: finite sequence of values of the same type

Records: Record of structs

Structured data type

① Components
② Defined by the set of rules that put them together.
③ Set of operations

```
int x;
```

↑ Memory allocated during compile time (static allocation)

mydata x, y, z;

→ Create
→ Enter the element
→ Read the element
→ take out the element
→ Destroy

Abstract Data Type (ADT).

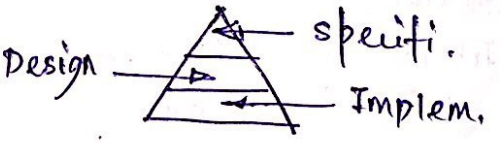
→ Conceptual mode of information structure

→ Specifies the components, their structuring relationships and a list of operations allowed to be performed.

→ Independent of data representation & implementation of operation.

→ Just a specification (what?) & no design / implementation (how?) info.

Components themselves are other ADTs.



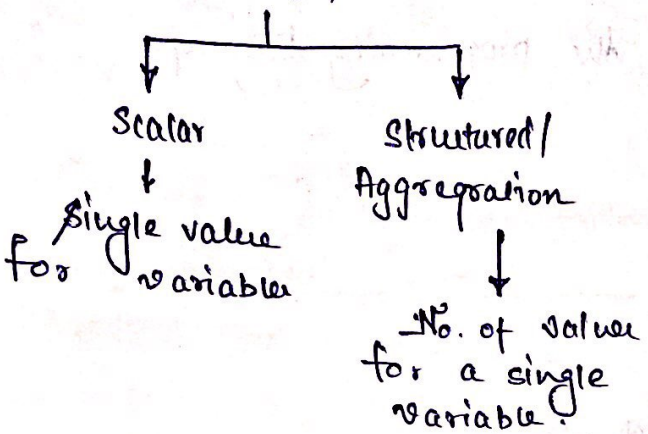
→ No assumption is made of the range.

→ Generalization of primitive data types.

Data Structures and Algorithms

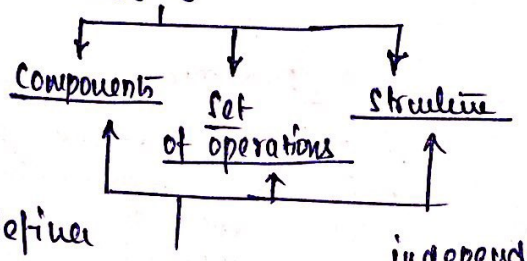
Prof. Chandan Maxumdar

Data Types



Object Orientation
Abstract Data Type
Aggregate
Scalar
0,1
↑ Levels of Abstraction

Aggregate → Arrays, Records, Sets



- ADT defines, independent of data repr. & implementation of operation, just a specification and no design/implementation is included.
  - No assumption is made about the range.
  - Specification involves the 'what' & not 'how'.
  - Generalization of primitive data types.
  - Encapsulate data values.
- Data-structure is the design representation of an ADT. ↑ can be rep. by several data structures
  - eg. 1. (<int>, <int>)
    2. <int>, <int>
  - Many data structures correspond to ADT.
  - operations on data structures are ~~refe~~ referred to as algorithms.

→ Relation among data structure

Data Structure

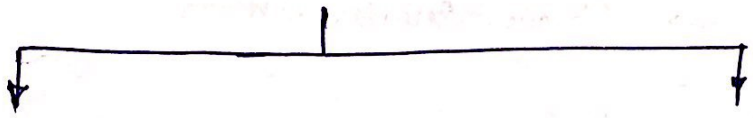
→ Facilitated using the programming language

Data Structure is an ADT

ADT

→ restricted at the levels of design and implementation

Types of Data Structures



Static

→ Memory requirement is fixed at the compile time

'OFF BY 1' ERROR

Dynamic

→ They can shrink & grow at the run-time. Thus, special memory allocation & deallocation functions from the programming system.

Memory

→ Algorithms & ~~Data~~ Usage must be as effient as possible

(Time & memory management is key)

09/09/2019

Data Structures and Algorithms

Prof. Chandan Mazumder

Algorithm Design Methods

- Divide & Conquer
- Greedy Algorithm
  - They may not always provide the best possible result.
  - However they produce good soln with high probability.
  - They can produce a good soln within a short time.

eg. Coin change,
Dijkstra's shortest path algo.

Coin change (exception)

1p 5p 11p 25p 50p

15p → 11p 4×1p (Greedy).
5p×3. ✓

- Dynamic Programming → Mathematicians named table filling & look up as programming
  - Slowly filling up the table
  - Instead of sub-dividing, we find the smaller problems whose soln lead to the original problem.
  - A table is created in which the entries are filled in a particular order.

eg. Fibonacci Series, Factorial

Creating a factorial

eg. n!

0	1	2	3		7
1	1	2	6	24	...

fact(n) = n × fact(n-1).

Backtracking

Eg: n-queens ~~Algorithm~~ Problem

(Please check)

↓

Sir's slides or Google.

Check:- Closed form soln for the n Queen problem.

Branch & Bound

In a chess game, the computer uses Static Evaluation Function

Qualities

→ Memory consumed

→ ~~Execution time~~ Execution time

Factors affecting execution time (check Sir's slides).

Implementation of ADT → Components ← No restriction
→ Structuring Relationship
→ Allowed operations

* Also understand the limitations and explicitly mention them

→ Static datastructure (Memory allocated at compile time)
→ ~~Compl~~ Dynamic ~~datatype~~ datastructure (Memory allocated dynamically).

Operations
1. Create
2. ~~Deletion~~ ~~Destro~~ Dispose (deallocating memory).
3. Enter data elements/components.
4. Access data elements. (eg. Search).
5. Delete data element.
6. Domain dependent operation

Exceptions & Additions are present → 4, 5, 6

Array as Data Structure

Elements of the same type arranged in a sequence.

ORDINAL → finite sequence.

Operations →
(1) store_array (a, i, e).
(2) retrieve_array (a, i) → e.

i → index a → array
e → value

Design
→ Required no. of memory locations are statically allocated consecutively.

Implementation
→ Built into language. (index size).

Error
MAX_INT

Higher Dimensional Arrays

2-design choices
- Row major
- Column major

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Compiler generates code for calculating the physical address of the element.

Row Major

1	2	3	4	5	6	7	8	9

0x1001 → 0x1005

Column Major

1	4	7	2	5	8	3	6	9

Polynomial

$$\sum_{i=0}^{n} a_i \cdot x^i$$

Application of Array

$$P = a_0 x^0 + a_1 x^1 + \dots + a_n x^n$$

floating point coefficients

(1) → 0 1 . . . . n

(2) → 0 1000

(3) → 1000 0

Degree of the polynomial

- → 1000 polynomials
- → 1000 max degree
- → 10,000 total no. of terms.

Termpool → 0 … 10,000

Poly → 0 … 1000

beg — end

Attaching a new term is difficult

Fragile design → An ending index → termpool, Poly

of the term pool is also kept in mind

Factor of Safety

→ (1) 130%. (30% increase)

→ (2) When memory utilisation is close to 70%, add more resources.

Sparse Matrix

Randomly distributed non-zero elements among a pool of zero elements

$$\begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 9 & 0 & 11 & 0 \\ 0 & 0 & 15 & 0 \end{bmatrix}$$

Approach

→ Store the non-zero elements of a sparse matrix in the form of 3 tuples (i, j, val) in an array.

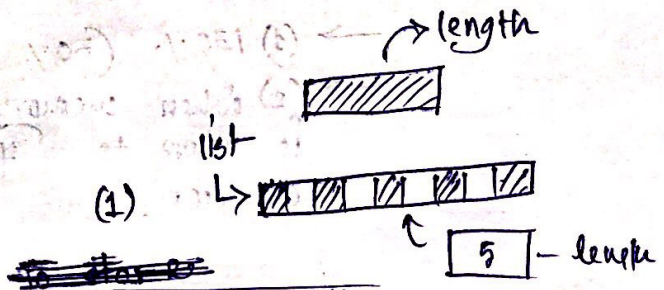
→ The first tuple contains (m, n, t) for a m x n matrix having t non-zero values.

→ Storage ↓ Time ↓

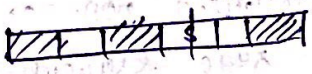
m	n	t
4	4	4

i	j	val
0	1	2
2	0	9
2	2	11
3	2	15

④ list



(2) To store the length, we use a character $.



Pointers (Recap).

Refer to notes SKS sir's

```
i = 20;
```

pointer ptr → address of i

associated with the type of the value, it is pointing to

variable that holds the address of another variable.

Ex. Solve CM sir's pointer question in his slides.

literal string

```
char *s = "abcdefg";
```

→ Heap.

```
s[20] = "...";
```

→ stored in stack.

(1) void pointer
(2) ptr to fn.

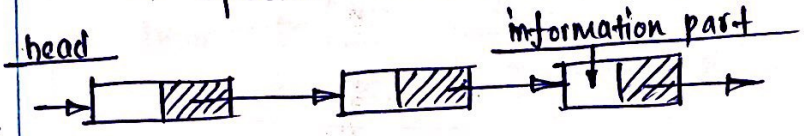
Data Structures and Algorithms

Prof. Chandan Mazumder

Linked List

→ Finite no. of nodes having the same type
→ Special variable that joins them together

Linked List



ADT Single Linked List

Operations (Check CM sir's slides),
init_l, empty_l, atend_l, insert_front, insert_after
delete_front, delete_after

Eg → Read integers from a file & arrange them in a linked list

Note → Check the existence of a file & whether the file is empty or not.

Equality
→ 2 empty lists are equal
→ Each corresponding node will have same information

→ Check whether list is empty or not.

Hint :- Always check (boundary cases or corner) || Very Important

~~Put pen & pad~~

→ C pointer implementation of Single Linked List. (CM sir's slide).
→ Self-referencial Structure

→ Check for memory allocation & exit/return if it fails

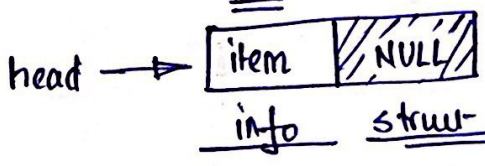
27/8/2019

Data Structures and Algorithms

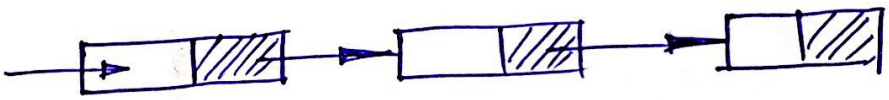
Prof Chandan Mazumder.

Creating Node

I



** Please check the C implementation from CM sir's slides **



Note :-

```
nodetype* delete_front (nodetype **phead)
{                       ...
                        cur->next = NULL;
}                       return (cur);
```

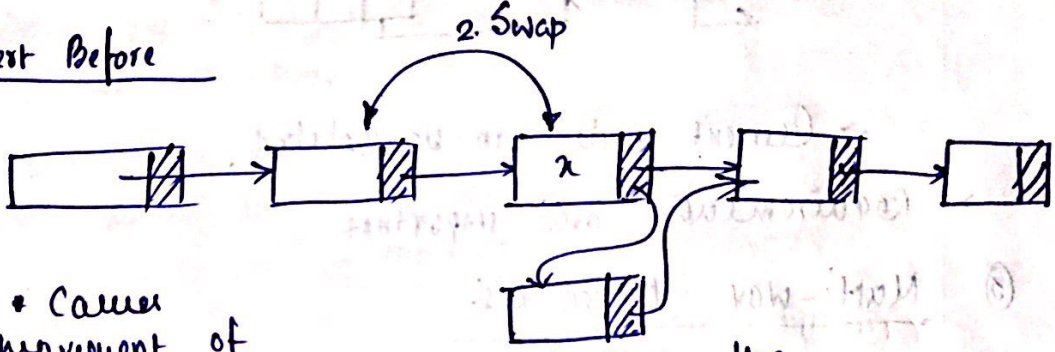
else, we might have a security violation.

Data Structures and Algorithms

Prof. Chandan Mazumdar

Single Linked List

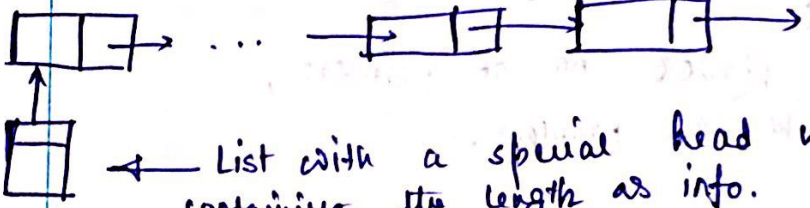
Insert Before



* Causes movement of ~~3x~~ more memory

Variations of Single Linked List

① Head Node



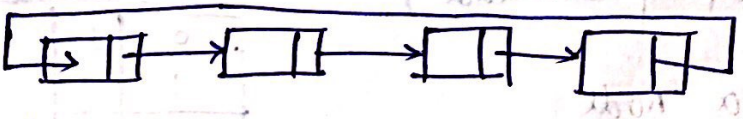
← List with a special head node containing the length as info.

② Dummy first node

→ uniform insertion and deletion operation insert_front & delete_front are not necessary.

→ Depends on the environment.

③ Circular Single Linked List

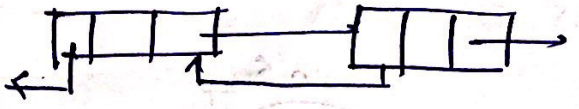


→ Any node to any node traversal is possible

→ Tail pointer can be used. However, when adding elements to the end of tail, remember to traverse tail, else it would be a insert_front operation.

② → Doubly Linked List

→ Frequent to & from movement.



→ Current node can be deleted.

→ Requirements are important

③ Multi-way Linked List

→ More than one sequence of dirⁿ. the list is made.

→ Double linked list can also be implemented inside the DLL

→ Due to higher no. of pointers, security can be a problem.

④ Sparse Matrix using ~~tiny~~ linked list

(Refer to Sir's slide)

→ Single Liked List using array.

→ 'Available nodes' also maintains all the list of unused nodes.

(Array of Structures)

info	cursor
0	1
:	:
:	NIL = -1

Creating a node

→ On node creation, the cursor has to be made null.

03/09/2019

Data Structures and Algorithms

Prof. Chandan Mazumdar

Algorithm Design Methods

→ Algorithms are developed on the basis of the choice of data structure.

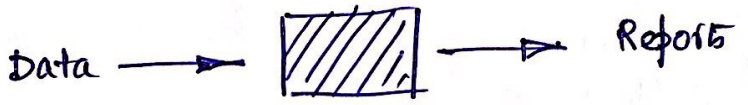
→ The first intuitive algorithm may not be the best one as per memory & time.

→ There are general techniques for development of algorithms.

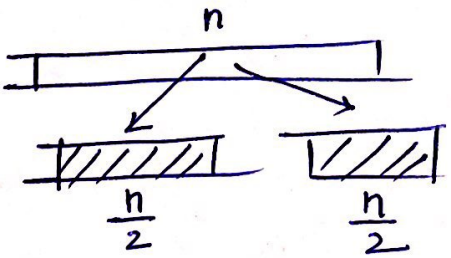
Divide & Conquer

→ Break the problem of size n into smaller problems & combine them.

D&C Problem Solving



1. Data Input
2. Data / Input Validation
3. Data Processing → Results
4. Result Validation
5. Form the Report.



Greedy Algorithm

16/9/19 Algorithm Analysis

Prof. Chandan Mazumder

→ Performance Analysis (Mathematical analysis of memory usage & time of execution)

→ Performance Measurement (~~Mathematical~~ Measurement of actual memory usage & time of execution)

Algorithm complexity

→ Space complexity (Main memory needed)

→ Time complexity (Amount of Computer Time needed)

1. Fixed Space Requirement
2. Variable Space Requirement

Time Complexity

→ Compile time does not depend on the program instance.

→ Execution time depends on the kind of algorithm and the size of the input data.

Time Complexity Analysis Objective

→ Machine Model

Normal single processor sequential computer where all instructions take equal amount of time, & has infinite memory

Time Complexity

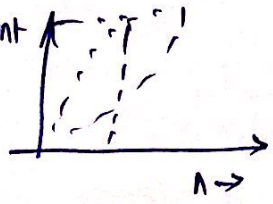
eg. = OR count the syntatically / semantic statement

Ticks of [18 µs] ↑ =

↳ finite accuracy

Time Complexity (step count)

eg. $2n+100$, $30n^2+15n-20$.



(2) Estimation

→ Worst case estimate is taken as it provides an upper bound for all inputs particularly the bad ones

→ Average ~~case analysis~~. case analysis.

Rules for computing running time

→ Sequence

→ Alternative structure

→ Loops

→ Nested Loops

→ Subprograms

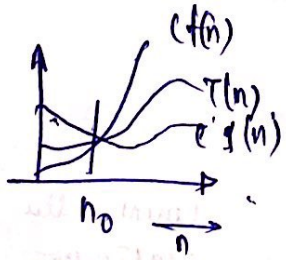
→ Recursive Subprograms

Order of

→ $T(n) = O(f(n))$ if there are positive constants c & n_0 s.t. $T(n) \leq cf(n)$ when $n \geq n_0$.

→ $cf(n)$ is atleast as large as $T(n)$.

→ Growth rate of $T(n)$ ≤ that of $f(n)$. } valid $n \geq n_0$



$T(n) = \Omega(g(n))$

$T(n) \geq c(g(n))$ for $n \geq n_0$.

$T(n) = \Theta(h(n))$

Properties

$T_1(n) = O(f(n))$ $\quad T_2 = O(f(n))$

$T_1(n) + T_2(n) = O(\max(fn), n)$

$T_1(n) * T_2(n) = O(fn) * g(n))$

$T_1(n) \leq c_1 f(n)$ $\quad n \geq n_{01}$

$T_2(n) \leq c_2 g(n)$ $\quad n \geq n_{02}$

$T_1 + T_2 \leq c_1 f(n) + c_2 g(n)$

ton $\quad n \geq n_0 = \max(n_{01}, n_{02})$

$\Rightarrow T_1 + T_2 \leq (c_1 + c_2) \max(fn, gn)$

$\Rightarrow T_1 + T_2 \leq c \max(fn, gn)$

$\Rightarrow T_1 + T_2 \leq O(\max(fn, gn))$

↓

max. time/space complexity that counts

Growth Functions

Computation

$\lim_{n \to \infty} \left(\frac{f(n)}{g(n)} \right)$

27/9/19

Algorithm Analysis

Prof Chandan Mazumder

① ① ①
read n; sum = 0; i = 0;
n [while (n!=0) do
    i = i+1; ——— ②
    sum = sum + x; ——— ②
    read n; ——— ①
avg = sum/i; ——— ②
write avg; ——— ①

$5n + 6 \equiv O(n)$

~~$O(5n+6) = O(n)$~~

'OR' $O(1) + O(n) = O(n)$.

Correctness of an Algorithm

① Partial Correctness
If the algorithm terminates it gives correct result.

② Termination
For all inputs, the algorithm terminates

read A[i] ⟶ ②

O(n) { for (i = 1 to n) do
    read A[i] ⟶ (2+1) .] n .
= 3n + 1

O(n²) { for (i = 1 to n-1) do
    for (i = n to i+1) do, ⟶ +1
    n [if (A[j] > A[j+1]) ③
        then swap(A[j+1], A[i]) ③
    ②

∴ $O(n^2)$

~~$\sum_{i=1}^{n-1} q$~~

$\sum_{i=1}^{n-1} \sum_{j=n}^{i+1} q$

$= q \sum_{i=1}^{n-1} (n-i) = q[(n-1) + (n-2) + \dots (1)]$

Rough Calculation.

$= \frac{q}{2}(n^2 - n)$.

$T(n) = O(n^2)$.

② Logarithmic Time Algorithm

$O(\log n)$

↳ cuts the problem by a fraction (eg. 1/2).

Binary Search

Iteration	No. of element
1	$\frac{n}{2}$
2	$\frac{n}{2^2}$
⋮	
n	$\frac{n}{2^n}$

$2^k = n$

$\Rightarrow \boxed{k = \log_2 n}$

Time Analysis

$n = 1024$.

Linear search = 1023.

Binary search = 10

Euclid's GCD algorithm

No. of iteration

$2 \log n$.

After 2 iteration,

remainder ≤ original value

30/09/2019

Data Structures and Algorithm

Prof. Chandan Mazumder

CT: 21st October, 2019

Logarithmic time Algorithm

(Refer to Sir's slides)

Fibonacci Series

$T(n) = T(n-1) + T(n-2) + 2$

$T(n) \geq fib(n)$.

$\because fib(n) = fib(n-1) + fib(n-2)$.

$Fib(n) \geq \left(\frac{3}{2}\right)^{n-1}$

↳ Proof by induction

$Fib(0) = 1 \geq \left(\frac{3}{2}\right)^{0-1} = \frac{2}{3}$.

$Fib(1) = 1 \geq \left(\frac{3}{2}\right)^{1-1} = 1$.

$Fib(2) = 2 \geq \left(\frac{3}{2}\right)^{1} = \frac{3}{2}$.

Taking induction hypothesis

$Fib(m) \geq \left(\frac{3}{2}\right)^{m-1} \quad [m = 0, 1 \ldots m]$

$Fib(m+1) = Fib(m) + Fib(m-1)$

$\geq \left(\frac{3}{2}\right)^{m-1} + \left(\frac{3}{2}\right)^{m-2}$

$= \left(\frac{3}{2}\right)^{m}\left[\frac{2}{3} + \frac{4}{9}\right]$

$= \left(\frac{3}{2}\right)^{m}\left[\frac{10}{9}\right]$

$\therefore \boxed{Fib(m+1) \geq \left(\frac{3}{2}\right)^{m}}$

Proved by Induction

$\boxed{T(n) = \Omega\left(\left(\frac{3}{2}\right)^{n-1}\right)}$

Even the lower-bound is exponential

~~05/9/2019~~

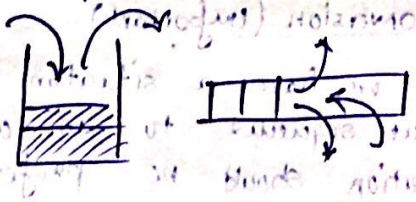
05/11/2019

Data Structures and Algorithms

Prof. Chandan Mazumder

ADT stack

→ finite seq. of elements of same type, where insertion & deletion can happen only at one end.



→ s_create(s)

→ s_dispose(s)

→ s_empty(s)

→ s_full(s)

→ push(s,e) → pop(s) → Stack is LIFO (Last In First Out structure),

Array Implementation of stack

Note: Element might be still accessed by array[(index)] however they cannot be accessed using stack operations.

Linked List Implementation of Stack

Note: check stack Full while pushing

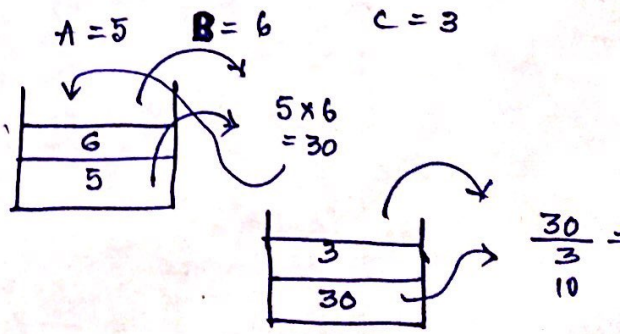
(??) → Physically nodes may not be created, so stack full case may arise.

Logically stack cannot overflow when using linked list.

Infix to Postfix

① A + B / C

↳ A B * C / $



② A / B ** C + D * E − A * C

↳ A B C ** / D E * + A C * − $

Postfix Evaluation

→ Important Example

Observations

→ Refer to sir's slides.

Conversion (Important)

If we have a situation of converting an input sequence to an output sequence, stack notation should be preferred.

→ Refer to sir's slides.

11/011/2019 — Data Structures and Algorithms — Prof. Chandan Mazumder

Recursion

→ fn defined in terms of itself.

→ Power lies in defining a potentially infinite set of objects by a finite set of statements.

→ sub-programs can be defined recursively.

→ $f(x_0) = \langle expression \rangle$

$f(x_i) = G(x_i, f(x_j)) \quad j < i$

Types of recursion

→ **Linear Recursion** — can also be expressed in terms of → Simple iterative soln

```
fnL(...)
{
   if(base condn is satisfied)
      then return some value.
   else
   {
      perform some actions
   }
}
```

→ **Binary Recursion**

Algorithm that makes 2 internal calls to itself.

→ **Non-linear Recursion** (Algorithm using a number of internal recursive calls within the body of procedure).

eg. Sample Generation, Combination Generation, Permutation Generation.

→ Mutual Recursion

①

Rules for Writing correct & eff. algorithms.

1) Base Case
2) Making progress
3) Design Rule
4) Compound Interest Rule.

Recursive calls are particularly appropriate when the underlying data is defined in recursive terms.
and NOT ALL RECURSIVE ALGORITHMS ARE EFFECIENT.

Bad example of Recursion

→ Large No. of calls. (If we have 1M calls, then 1M calls will be active).

In Linear Recursion, we can remove recursion.

Non-terminating Recursive Program

→ $f(x) = f(x+1) + f(x+2)$

AWAY

Application of Stack

→ LIFO Data Structure

```
main()
{
  f1()
  ...
}
```

// a1 //

```
f1()
{
  a1 → f2()
}
```

// a2 //

```
a2 → f2()
{
  f3()
}
```

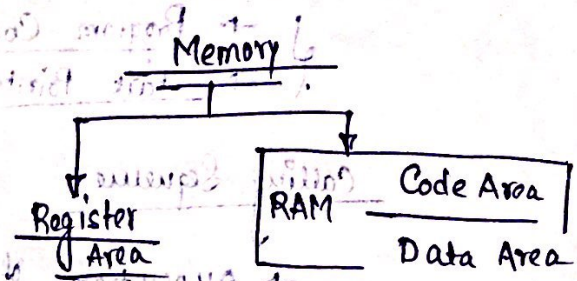
// a3 //

→ Fn calls (A Top Level overview)

→ Fn Calls & The Runtime Stack

① Runtime Environment

② C Language uses a stack based runtime environment



Code Area

Entry point for procedure 1 → Code for Procedure 1

Entry point for procedure n → Code for Procedure n

→ Data Area

→ Dynamic Memory

Stack Area | Heap Area

Memory Organisation

code/area
global/static area
stack
free space
Heap.

Procedure activation Record

→ Concept of stack frames

When activation records are kept on stack, they are called stack frames.

arguments
bookkeeping info
local data
local temporaries

Registers

→ When the processor has many registers, the entire static area & stack activation records may be kept in the registers.

- → Program Counter (PC)
- → Stack Pointer (SP).

Calling Sequence

- → Allocation of memory for the activation record
- → Computation & storing the arguments
- → ...

Return Sequence

Stack-based Runtime Environment

Global Procedures

A view of the Runtime Stack

Access to Variables

Calling sequence

Prologue

Object code

Epilogue

Compiler works on KERNEL

18/11/2019

Data Structures and Algorithms

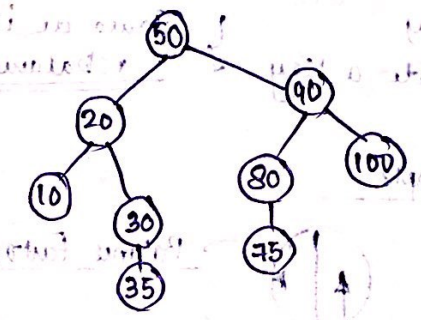
Prof. Chandan Mazumder

Binary Search Tree

a) Key of every node in the right sub-tree of T is greater than key at root

b) " " left " " "

c) All keys are distinct.



BST operations (Refer to slides)

→ Recursive Search (Linear Recursion) → Iterative

→ Non-Recursive Search

→ Insertion Example

→ Deletion Example

Deleting the root node (eg. 50). [using Inorder successor/Inorder predecessor, and checking whether the node has become a leaf, if it does then delete it].

Problem of BST

Height depends on insertion & deletion of keys.

Tree may degenerate into a linked list.

Remedy:- Balanced Tree

Height Balanced Tree (AVL Tree)

Adelson – Velskii, Landis.

$|h_L - h_R| <= 1$.

Left sub tree — Right sub tree

Balance Factor.

$BF = h_L - h_R$

BF = 1 → left high

= −1 → right high

= 0 → equal. High.

AVL Tree Operations

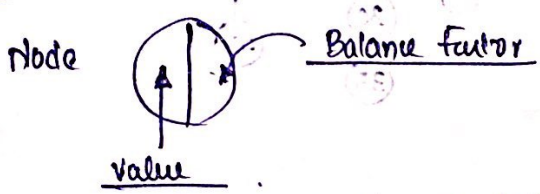
① Search
② Max & Min
③ K^{th} max & K^{th} min

} Same as BST.

④ Insert a Key
⑤ ~~Delete~~ Delete a Key

} Same as in BST, rebalance it necessary.

AVL Tree Example



Rebalancing needs Rotation (Refer to sir's slides)

Right Rotation
Left – Rotation

} Constant time algorithm.

Insertion Examples

Tree remains unbalanced even after rotation.

↳ Right child has left high / Left child has Right high.

Remedy :- Double rotation.

Deletion Example

Data Structures and Algorithms

Prof. Chandan Mazumdar

2-3 Tree

→ All leaf nodes are at the same level

→ All non-leaf node are either binary / ternary.

Keys < K1 → p1

Keys > K2 → p3

K1 < Keys < K2 → p2.

CT

Time complexity

Sorting

Stack & Queue

Recursion

Tree Node

binary p1 K1 p2 K2 p3

Operations

$2^h - 1 <= n <= 3^h - 1$

$\Rightarrow \log_3(n+1) \leq h \leq \log_2(n+2)$

Insert a node (Insertion Examples)

Deletion Examples

B-Tree

Definition

→ Each node contains atmost d records & at least $\lceil d/2 \rceil$ records.

→ Root node can have atmost d records & as few as one record

→ $1 \leq k \leq d$ with key values k_1 & k_2 have pointers to $k+1$ subintervals of keys stored in sub-trees.

→ Leaf node has empty sub-tree below it. All leaf nodes are at the same level.

B-Tree Node

$$\left[\log_{\left[\frac{d}{2}\right]+1}^{n+1}\right] \geq h \geq \left[\log_{d+1}^{(n+1)}\right]$$

2-3 Tree is a B-tree of order ②.

Insertion Examples on a B-Tree of order 4

Deletion Example

23/11/2019

B-Tree

Prof. Chandan Mazumder

→ Root node has atmost d records and at few as one record.

→ All leaves are at the same level.

→ Height becomes extremely small, ⇒ Super fast search.

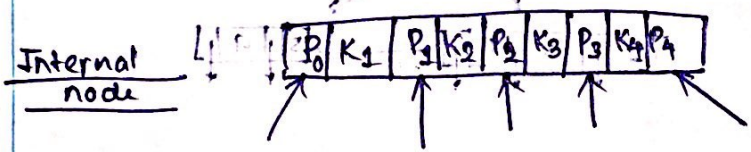
B+ Tree

Internal nodes → Keys are stored just like a B-Tree.

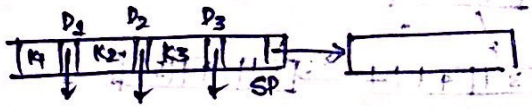
Leaf node → within them, keys are unique and sorted.

One key value can have a max of 2 occurences.

Other than the root, we can have d keys & (d+1) pointers.



$< K_1$ $K_1 \leq$? $< K_2$ $\geq K_2$ & $< K_3$ ...



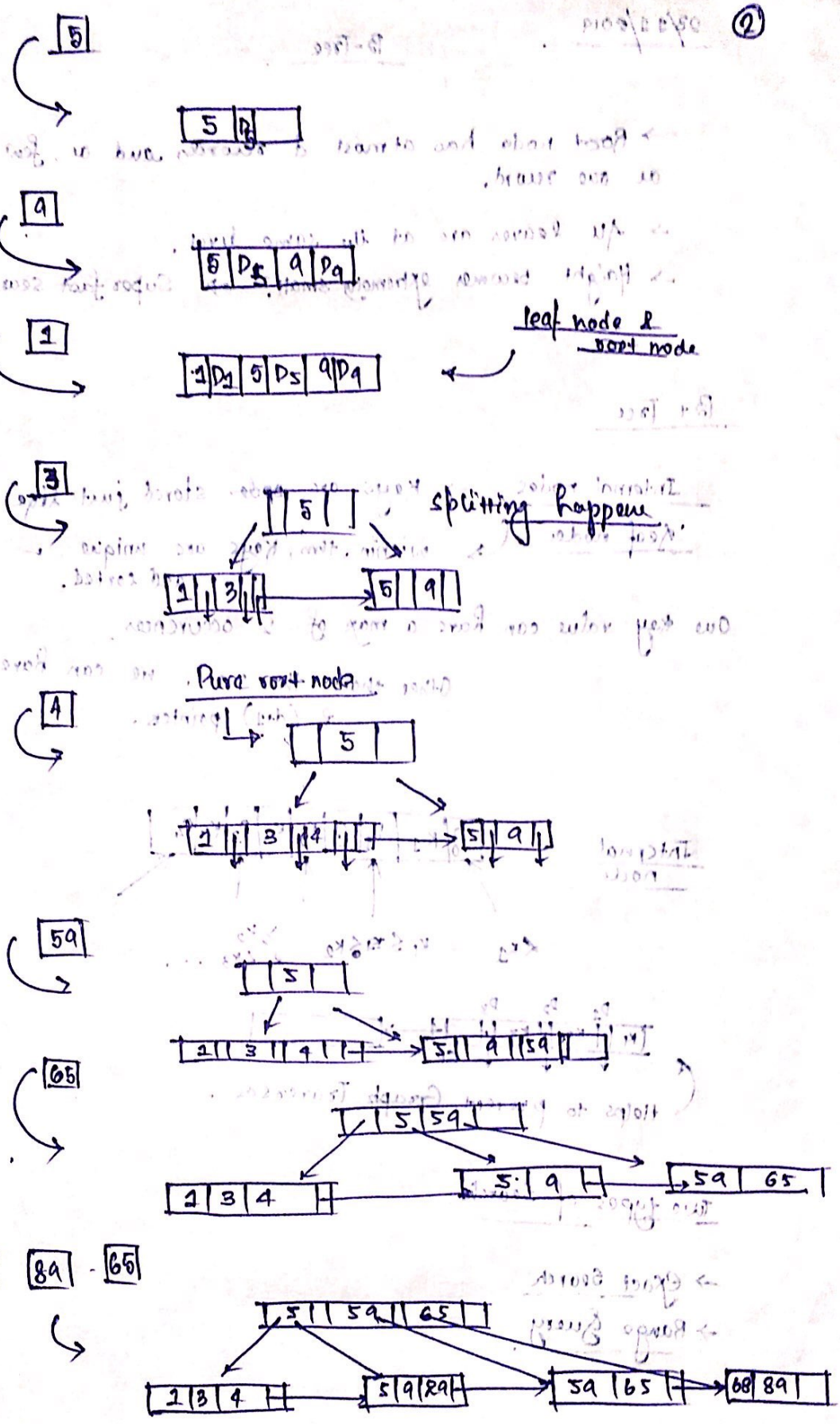
Helps to prevent Graph Traversal.

Two types of Search:

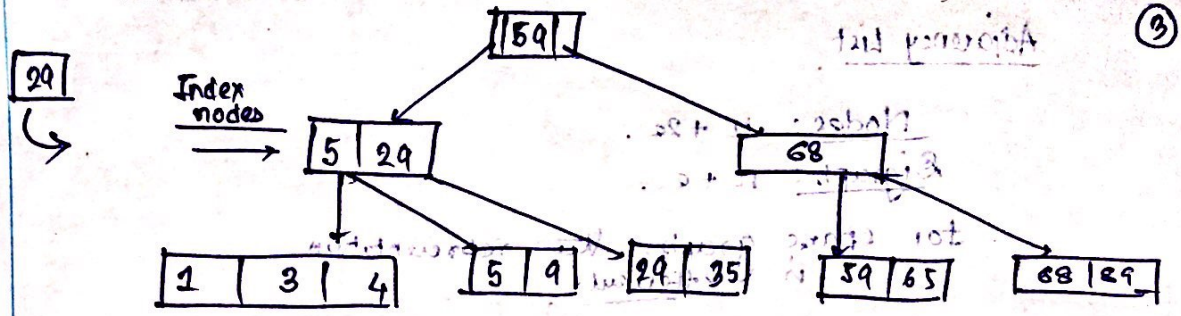
→ Exact Search

→ Range Query:

D=3
4 pointers

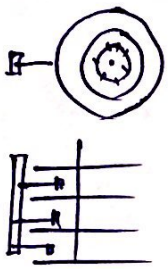


* Data pointers are not shown



Data stored in harddisk [Side Track Sector].

Front-view.



Graph.

$G = (V, E)$ → edges.
↑ vertices

$e_i = (v_m, v_n)$,

ordered pair ⇒ directed graph. (DiGraph).
unordered ⇒ un-directed graph

$(v_m, v_n) \subseteq V \times V$
⇒ v_m & v_n are said to be adjacent

DiGraph
in-degree & out-degree.

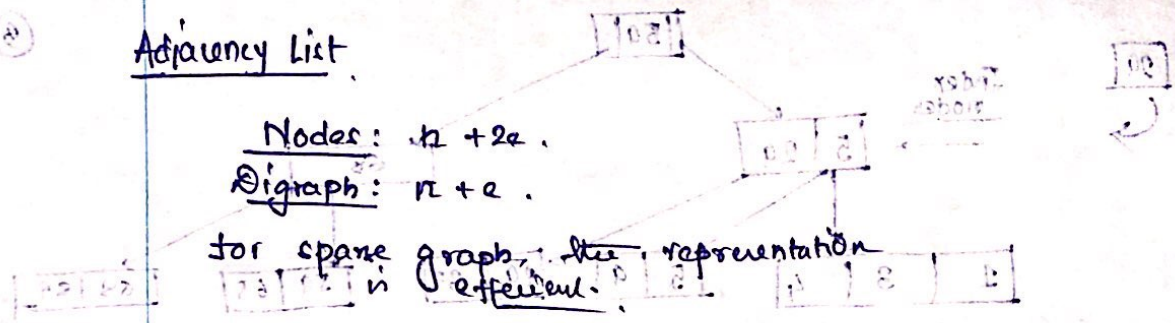
SubGraph.

Path from v_1 to v_n in a sequence $v_1, v_2, \ldots, v_k$.

Paths may have cycles & edges may have associated weights.

Adjacency matrix.

Adjacency List



Nodes: $n + 2e$.

Digraph: $n + e$.

for sparse graph, the representation is efficient.

Graph Traversal

→ Depth First Search (Recursion & Stack based).

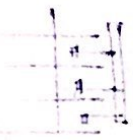
→ Breadth First Search.

Spanning Tree

Kruskal's Algorithm

Prim's Algorithm

Dijkstra's shortest path algorithm



Transitive Closure Matrix

All pair shortest path

Digraph with cycles & negative

25/11/2019

Hashing

Prof. Chandan Mazumder

Hash Table → list of record structures with distinct keys.

- Insert
- Search
- Delete
- Modify.

Goal

Example

Linear Probing

Collision Resolution Strategies

$\alpha = \frac{n}{M}$ → Load factor

0.5 – 0.8

Problem of Linear Probing

→ May lead to Primary Clustering. → degrades the performance of the hash.

Quadratic Probing

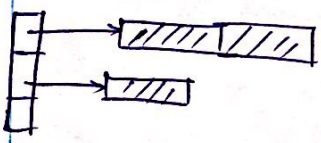
Small clusters will be formed instead of very large cluster.
→ Secondary clustering

Double Hashing

→ No. of collisions greatly reduced & cluster formation can be also avoided.

Chaining

a)



b) **Coalesced Chaining** ✓

Linked list are maintained in the same array.

Bucket Hashing

Expected no. of buckets for random input

$$N = M * \lceil [n/M]/b \rceil$$ ← ceil fn.

Expected no. of buckets in each chain $= \lceil [n/M]/b \rceil$

Filtering based on Set membership

Filtering with one Hash fn

Bloom Filter

Outline process: Streaming data comes

Analysis

False positive Analysis