What is a Data Structure ?

Definition of DS

DS is one of the most fundamental subject in Computer Science & in-depth understanding of this topic is very important especially when you are into development/programming domain where you build efficient software systems & applications.

Definition

In computer science, a data structure is a data organization, management and storage format that

enables efficient access and modification.

In Simple Words

• Data Structure is a way in which data is stored on a computer

Why do we need Data Structures ?

Data structure is a particular way of storing and organizing information in a retrieved and used most productively.

Each Data Structure allows data to be stored in specific manner. Data Structure allows efficient data search and retrieval.

Specific Data structures are decided to work for specific problems.

It allows to manage large amount of data such as large databases and indexing services such as hash table.

Example of DS

Array Data Structure -

An array is a collection of elements of the same type placed in contiguous memory locations that can be individually referenced by using an index to a unique identifier.

int

size

numbers [4] = { 10, 20, 30, 40 };

What is an Algorithm ?

Dictionary Definition: A process or set of rules to be followed in calculations or other

problem-solving operations, especially by a computer. Formal Definition: An algorithm is a finite set of instructions that are carried in a specific order to perform specific task.

Algorithms typically have the following characteristics -

• Inputs: 0 or more input values.

• Outputs: 1 or more than 1 output.

Unambiguity: clear and simple instructions.

• Finiteness: Limited number of instructions.

• Effectiveness: Each instruction has an impact on the overall process.

Real World example of an Algorithm -

Algorithm(aka process) to make a lemonade - 1. Cut your lemon in half.

2. Squeeze all the juice out of it that you can.

3. Pour your juice into a container with 1/4 cup (2 oz) sugar.

4. Add a very small amount of water to your container.

5. Stir your solution until sugar dissolves.

6. Fill up container with water and add ice.

7. Put your lemonade in the fridge for five minutes.

1. Serve and enjoy!

Example of an Algorithm in Programming -

Write an algorithm to add two numbers entered by

user. -

1. Step 1: Start

2. Step 2: Declare variables num1, num2 and sum.

3. Step 3: Read values num1 and num2.

4. Step 4: Add num1 and num2 and assign the result to sum.(sumnum1+num2)

5. Step 5: Display sum

1. Step 6: Stop

What is Abstract Data(ADT) Type in DS ?

2 ways to study DS Abstract/Logical View Implementation view Example

What is Abstract Data Type?

Data Structures Definition: In computer science, a data structure is a data organization, management and storage format that enables efficient access and modification.

2 Ways of looking at Data Structures -

• Mathematical/Logical/Abstract Models/Views

• Implementation

Algorithm Analysis (Time Complexity):

Big O notation

What is Algorithm Analysis?

Why Algorithm Analysis?

Types of Complexities CPP

Big O Notation

Examples

What is Algorithm Analysis ?

Simple Definition -

Algorithm analysis is a study to provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. i.e. calculating efficiency. CPP

Generally, the efficiency an algorithm is related to the input length (number of steps), known as time complexity, or volume of memory, known as space complexity.

Why do we need Algorithm Analysis ?

Knowing efficiency of an algorithm is very vital on mission critical tasks.

Generally there are multiple

approaches/method/algorithms to solve one problem statement. Algorithm analysis is performed to figure out which

is the better/optimum

approaches/method/algorithms out of the options.

What does a BETTER Algorithm mean?

Faster? (Less execution time) - Time Complexity

• Less Memory? - Space Complexity

Easy to read?

Less Line of Code ?

Less Hw/Sw needs?

\*Note: Algorithm Analysis does not give you accurate/exact values(time, space etc), however it gives estimates which can be used to study the behavior of the algorithm.

What is Asymptotic Algorithm Analysis ?

Definition: In mathematical analysis, asymptotic analysis of algorithm is a method of defining the mathematical boundaries of its run-time performance. Using the asymptotic analysis, we can easily

estimate about the average case, best case and worst case scenario of an algorithm. Simple words: It is used to mathematically calculate the running time of any operation inside an algorithm.

Asymptotic Algorithm analysis is to estimate the time complexity function for arbitrarily large input.

Time Complexity: is a computational way to show how(behavior) runtime of a program increases as the size of its input increases.

Problem Statement : Write an Algorithm/program to find the SUM of N numbers (0 -

Algorithm 1

function sumOfNumbers (N)

1.sum = 0

2. for (i = 0 to N)

3.sum sum + i

4.print(sum)

Algorithm 2

function sumOfNumbers (N)

1.sum = (N\* (N+1))/2

2.print (sum)

Asymptotic Algorithm Analysis (Time Complexity)

Big O Notation 0

Omega Notation

Theta Notation Ꮎ

Examples

Mohammad Asif, [05.06.21 21:49]

Big O notation (0)

1 < log(n) < sqrt(n) <n<nx log (n) < n² <n³<.... <n"

>> Big O notation specifically describes worst case scenario. >> It represents the upper bound running time complexity of an algorithm.

Mathematically

Let f and g be functions of n-> where n is natural no denoting size or steps of the algorithm then -

f(n) O(g(n))

IFF

f(n) <= c.g(n) where n>=ne, c>0, ne>=1

Mohammad Asif, [05.06.21 21:50]

Big Omega (2)

1 < log(n) < sqrt(n) <n<nx log (n) < n² <n³ < .... <n"

>> Big Omega notation specifically describes best case scenario, >> It represents the lower bound running time complexity of an algorithm.yaxis >> Basically it tells you what is the fastest time/behavior in

which the algorithm cap run. Mathematically

Let f and g be functions of n-> where n is natural no denoting size or steps of the algorithm then

f(n) = (g(n))

IFF

f(n) >= c.g(n) where n>ene, c>0, n0>=1

Mohammad Asif, [05.06.21 21:51]

Big Theta (0)

1 <log(n) <sqrt(n) <n<nx log (n) < n² <n³<.... <nn

>> Big Omega notation specifically describes average case scenario. >> It represents the most realistic time complexity of an algorithm.

Mathematically -

Let f and g be functions of n-> where n is natural no denoting size or steps of the algorithm then

f(n) = (g(n))

IFF

c1.g(n) <= f(n) <= c2.g(n) where n>=ne, c1, c2>0, n>=ne, ne>=1

Mohammad Asif, [05.06.21 21:51]

1<log(n)<sqrt(n)<n<nxlog(n)<n² <n³<...<nª

y

- axis

Why 3 different analysis?

Big a Best Case

Big O Worst Case

Big - Average Case

Example

Find best case, average case & worst case of Linear Search Algorithm. Integer array on N numbers

Mohammad Asif, [05.06.21 21:52]

Algorithm Analysis (Space Complexity) : Big O notation

- What is Algorithm Analysis?

Why Algorithm Analysis?

Types of Complexities

- Space Complexity?

Examples

Mohammad Asif, [05.06.21 21:52]

What is Algorithm Analysis ?

Simple Definition -

Algorithm analysis is a study to provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. i.e. calculating

efficiency.

CPP

Generally, the efficiency an algorithm is related to the input length (number of steps), known as time complexity, or volume of memory, known as space complexity.

Mohammad Asif, [05.06.21 21:52]

Why do we need Algorithm Analysis ?

• Knowing efficiency of an algorithm is very vital on mission critical tasks.

Generally there are multiple

approaches/method/algorithms to solve one problem statement. Algorithm analysis is performed to figure out which

is the better/optimum

approaches/method/algorithms out of the options.

Mohammad Asif, [05.06.21 21:53]

What does a BETTER Algorithm mean?

• Faster? (Less execution time) - Time Complexity

Less Memory? - Space Complexity

Easy to read?

• Less Line of Code?

• Less Hw/Sw needs?

\*Note: Algorithm Analysis does not give you accurate/exact values(time, space etc), however it gives estimates which can be used to study the behavior of the algorithm.

ADEN

Mohammad Asif, [05.06.21 21:53]

What is Space complexity?

Definition: The space complexity of an algorithm or a computer program is the amount of memory space required to solve an instance of the computational problem as a function of the size of the input.

Simple words: It is the memory required by an algorithm to execute a program and produce output. CPP

Similar to time complexity, Space complexity is often expressed asymptotically in big O notation, such as O(n), O(nlog(n)), O(n^2) etc., where n is the input size in units of bits needed to represent the input.

Mohammad Asif, [05.06.21 21:54]

For any algorithm, memory is required for the following purposes -

To store program instructions.

To store constant values.

To store variable values.

And for few other things like function calls, jumping statements etc.

Auxiliary Space: is the temporary space (excluding the input size) allocated by your algorithm to solve the problem, with respect to input

size.

Space complexity includes both Auxiliary space and space used by input.

Mohammad Asif, [05.06.21 21:55]

Space Complexity Input Size + Auxillary Space

Algorithm 1- Addition of 2 numbers

function add(n1, n2)

sum=

200

return sum

ni 4 bytes 0(1)

m₂ 4 bytes

Sum 4 bites

Aux Sp-34 bytes

Total 16 bytes:

Algorithm 2- Sum of all elements in array function sumofNumbers (arr[],N)

1.sum e 2.for(i= to N)

}

3.sum sum + arr[1]

4.print (sum)

arr Nx4byle

Sum → 4 bytes

i 4 bytes

Aux→ 4 bytes

Total (space

Mohammad Asif, [05.06.21 21:55]

Space Complexity = Input Size

Factorial of a number(iterative)

+

Auxillary Space

y axis : size in bytes

x axis: N value

40 bytes

36 bytes

32 bytes

28 bytes 24 bytes

20 bytes

16 bytes

12 bytes 8 bytes

4 bytes

40 bytest

36 bytes

32 bytes

28 bytes 24 bytes

20 bytes

16 bytes

12 bytes

8 bytes 4 bytes+

Algorithm 3 -

int fact =

1;

for (int i = 1; i <= n; ++i)

}

fact \*= i;

return fact;

4

5

y axis : size in bytes

x axis: N value

9 16

Algorithm 4 Factorial of a number (recursive)

if(n<=1)

}

return 1;

else

{

}

return (n\*factorial2(n-1));

5

6

7

8

9

1

Mohammad Asif, [05.06.21 21:56]

Linked List Data Structure

What is a Linked List?

Linked List vs Arrays

Pros & Cons

- Operations of Linked List

Types of Linked List

Applications

- Tanmay Sakpal

Mohammad Asif, [05.06.21 21:56]

What is Linked List Data Structure?

Definition: A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers(entity that point to the next element)

In simple words, a linked list consists of nodes where each node contains a data field and a reference (link) to the next node in the list.

Mohammad Asif, [05.06.21 21:57]

Working of Linked List -

Linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

1) Node/Link/Element/Object -

Each node In the linked list consists of

two parts

a) data

b) link to the next node

2) Next - This points to the next node/element in the linked list (since they are not stored in a contiguous memmory locations)

Mohammad Asif, [05.06.21 21:57]

Linked List vs Arrays

Advantages of Linked List over Arrays -

1) Dynamic size L

2) Ease of insertion/deletion L

Disadvantages of Linked List over Arrays -

1) Random access is not allowed. We have to access elements sequentially starting from the first node.

2) Extra memory space for a pointer is required with each element of the list.

3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case

of linked lists.

Mohammad Asif, [05.06.21 21:58]

Operations of Linked List- N next

1. Traversing a linked list.

(head)

DN

4. Inserting a new node to a specific position in the list

(head)

DN

DN

2. Append a new node (to the end) of a list

(head)

5. Deleting a node from the list

(head) (head)

DNDNDN

/simplesnippets/simplesnippets/simplesnippets/simplesnippet

3. Prepend a new node (to start) of list

(head)

(head)

DN-DN-DN-DN

6. Updating a node in the list

(head)

DN→DN→DN

D₂

Mohammad Asif, [05.06.21 21:58]

Types of Linked List -

D = Data P = Previous N= Next

1) Singly Linked List

(head)

DN

DN

2

DN

3

2) Doubly Linked List

PDN

3) Circular Linked List

(head)

Mohammad Asif, [05.06.21 21:59]

Some Applications of Linked List Data Structure

Some Applications of Linked List are as follows -

• Linked Lists can be used to implement Stacks, Queues.

• Linked Lists can also be used to implement Graphs. (Adjacency list representation of Graph).

Implementing Hash Tables - Each Bucket of the hash table

can itself be a linked list. (Open chain hashing). Undo functionality in Photoshop or Word. Linked list of states

Mohammad Asif, [05.06.21 22:59]

What is an Array ?

Introduction & Comparison

What is a Linked List?

A linked list is a collection of elements(nodes) that are NOT stored

in contiguous memory locations. The nodes are linked to each other using pointers(entity that points ot the next node). A node usually consist of 2 fields namely-data & next. Sometimes a 3rd field called id/name

is also used.

Node (key, value, next)

n1 #100 145

#102

front

n2 #102 26

#300

Computer Memory

#100

n1 #102

145 #102

#200

2 6 #202

#300 n3 #302

377 #204

n3 300 377

#204

n2 #104

#300

#204

n4 # 204 4 54

NULL

#106

n4 #206

454 NULL

#304

#306

An array is a collection of elements of the same type, placed in contiguous memory locations than can be individually referenced by using an index to a unique identifier

int arr[4] = {45,

Mohammad Asif, [05.06.21 23:00]

1. Size of Data Structure

n1

#12

134

#34

front

ARRAYS

int arr[4] ={34,56,6,7}

index ->

34

#10

1

56

#14

2

LINKED LIST

n3

#21

36

#67

n4

3

4 bytes

4x4

=16 byte

n2

#34

256

#21

#67

41

NULL

ddress ->

#18

#22

Size is STATIC

Size is DYNAMI

Mohammad Asif, [05.06.21 23:01]

2. Data Acess/Read/Update (Cost)

1st

m1 #12

1134

#341

Int =4 byte

10(n) = 1

X

Size of lelement

n234

2 56

21

ARRAYS

int arr[4] (34,56,6,7)

index 0 1 2 3

34 56 6 75

address#10 #14 #18 #22

RANDOM ACCESS

arr [3] =15

Bas e

Address

index + Pos

= 10 + (3x4

22

LINKED LIST

#21

3 60

#67

#67

41

NULL

front

SEQUENTIAL ACESS

O(n) = n > Linear Time

Mohammad Asif, [05.06.21 23:01]

Help

3) Insertion & deletion of elements (Cost)

ARRAYS

2 3

6

n1 #12 n2 134

#34 2 56

#21

front

1) Insertion/Deletion at the Beginning -

LINKED LIST

n3 #21

36

#67

n4 #67

41

NULL

int arr[4]

index ->

34

1

56

address -> #10 #14 #18

1) Insertion/Deletion at the Beginning

#22

2) Insertion/Deletion at the End

2) Insertion/Deletion at the End

3) Insertion/Deletion at random nth postion

3) Insertion/Deletion at random nth postion

Mohammad Asif, [05.06.21 23:02]

Memory Requirement & Memory Usage/Efficiency

Mohammad Asif, [05.06.21 23:02]

Ease of use

Mohammad Asif, [05.06.21 23:02]

Searching method

Mohammad Asif, [05.06.21 23:04]

1. Understanding of What is Linked List & Singly Linked

3 2. C++ Objects & Classes

4 3. Pointers in C++

5 4. Dynamic Memory allocation

6

DISCLAIMER: There are different ways to implement Linked List (& pretty much all other DS) so this 9 Singly Linked List Operations

7

8

10 11

12

13

14

15

16 \*/ 17

18

19

20

21

22 23

24

1. appendNode()

2. prependNode ()

3. insertNodeAfter()

4. deleteNodeByKey()

5. updateNodeByKey()

6. print()

Mohammad Asif, [05.06.21 23:06]

Append node algorithm/pseudo code

1. Get the new node.

2. Check if node exists with same key

2a. if true, abort 2b. if false continue

3a. if head == NULL then append

at start

3b. else traverse to the end of the list

4. Append new node at the end

Mohammad Asif, [05.06.21 23:07]

Prepend node algorithm/pseudo code

1. Get the new Node.

2. Check if node exists with

same key

2a. if true, abort

2b. if false continue

3. head->previous = new\_node head

4. new\_node->next 5. head = n

Mohammad Asif, [05.06.21 23:07]

Insert node node after algorithm/pseudo code

1. Get the new Node & the key of the node in the list after which you want to link this new node.

2. Check if node exists with same key as new node 2a. if true, abort

2b. if false continue

3. Check if node exists with the key entered by user 3a. if false, abort

3b. if true continue

4. access node N after which you want to append new node new\_N 4a. if N is at the end then

N->next = new\_N new\_N->previous = N;

4b. if N is in between. new\_N->next = N->next N->next->previous = new\_N new\_N->previous = N

n->next = new\_N

Mohammad Asif, [05.06.21 23:08]

Delete node by key algorithm 1. Get the key of node you want

to delete. 2. Check if node exists with

same key

2a. if false, abort

2b. if true continue

3a. if head== NULL then list empty 3b. if head!=NULL && key matches

head node then head=head->next

3c. if head!=NULL && key doesn't matches head node then Traverse to the node with the key.

4a. if node at the end then previousNode->next = null 4b. if node in between then

prevNode->next=nextNode; nextNode->previous-prevNode;

Mohammad Asif, [05.06.21 23:08]

Update node by key algorithm -

1. Get the key of node you want

to update. 2. Check if node exists with

same key

2a. if false, abort 2b. if true continue

3. traverse to that node.

4. Update the data value.

Mohammad Asif, [05.06.21 23:09]

Append node algorithm/pseudo code

1. Get the new node. 2. Check if node exists with same key

2a. if true, abort 2b. if false continue

3a. if head == NULL then append

at start

3b. else traverse to the end of the list

4. Append new node at the end

Mohammad Asif, [05.06.21 23:10]

Prepend node algorithm/pseudo code

1. Get the new Node. 2. Check if node exists with

same key

2a. if true, abort 2b. if false continue

3. Assign head value to a

temporary pointer ptr 4. Use ptr to traverse to end of list

5. Use ptr to access next pointer of last node to point to this newly passed node

6. point next pointer

of n to head ptr. 7. change head ptr to n

Mohammad Asif, [05.06.21 23:10]

Insert node node after algorithm

1. Get the new Node & the key of the node in the list after which you

want to link this new node. 2. Check if node exists with same key as new node 2a. if true, abort

2b. if false continue 3. Check if node exists with the

key entered by user 3b. if true continue

3a. if false, abort

4. access node N after which you want to append new node new\_N

4a. If node to be inserted at the end then

new\_node->next = head previous\_node->next = new\_node

4b. if node to be inserted in between then new\_node->next = previous\_node->next previous\_node->next = new\_node

Mohammad Asif, [05.06.21 23:11]

Delete node by key algorithm

1. Get the key of node you want to delete.

2. Check if node exists with same key

2a. if false, abort 2b. if true continue

3a. if head== NULL then list empty 3b. if head!=NULL && key matches head node then head=head->next

last\_node->next = head 3c. if head!=NULL && key doesn't matches head node then Traverse to the node with the key.(ptr)

4a. if node at the end then previousNode->next = head 4b. if node in between then

prevNode->next=ptr->next

Mohammad Asif, [05.06.21 23:11]

Update node by key algorithm -

1. Get the key of node you want to update.

2. Check if node exists with same key

2a. if false, abort

2b. if true continue

1. traverse to that node. 4. Update the data value.

What is Stack Data Structure ?

Definition: Stack a linear data structure which operates in a LIFO(Last In First Out) or FILO (First In Last Out) pattern. • It is named stack as it behaves like a real-world stack, for example-a deck of cards or a pile of plates, etc. • Stack is an abstract (predefined) capacity. data type with a bounded • It is a simple data structure that allows adding and removing elements in a particular order. • The order may be LIFO(Last In First Out) or FILO(First In Last Out).

Working of Stack -

Stack Data Structure operates in a LIFO(Last In First Out) pattern or

FILO (First In Last Out) pattern.

>> Items are added on top of the stack. This is know as PUSH operation

>> Items are removed from top of the stack. This is know as POP operation

Standard Stack Operations -

1) push() -

Place an item onto the stack. If there is no place for new item, stack is in overflow state.

2) pop()

Return the item at the top of the stack and then remove it. If pop is called when stack is empty, it is in an underflow state.

3) isEmpty()

Tells if the stack is empty or not

4) isfull() -

Tells if the stack is full or not.

5) peek() -

Access the item at the i position

6) count() -

Get the number of items in the stack.

7) change() -

Change the item at the i position

8) display()

Display all items in the stack

Some Applications of Stack Data Structure

Balancing of symbols

Infix to Postfix/Prefix conversion

Redo-undo features at many places like editors, photoshop.

Forward and backward feature in web browsers

Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.

Other applications can be Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver

In Graph Algorithms like Topological Sorting and Strongly Connected Components