ડેટા સ્ટ્રક્ચર વિથ પાયથોન (4331601) - શિયાળો 2024 સોલ્યુશન

Milav Dabgar

December 03, 2024

પ્રશ્ન 1(a) [3 ગુણ]

પાયથોનમાં સેટ ડેટા સ્ટ્રક્ચર સમજાવો?

જવાબ

Set એ પાયથોનમાં અનન્ય એલિમેન્ટ્સનો અક્રમ સંગ્રહ છે. સેટ્સ mutable છે પરંતુ તેમાં ફક્ત immutable એલિમેન્ટ્સ જ હોય છે.
મુખ્ય લક્ષણો:

કોષ્ટક 1. Set Properties

લક્ષણ	વર્ણન
અનન્ય એલિમેન્ટ્સ	કોઈ ડુપ્લિકેટ વેલ્યુઝ મંજૂર નથી
અક્રમ	કોઈ ઇન્ડેક્સિંગ અથવા સ્લાઇસિંગ નથી
Mutable	એલિમેન્ટ્સ ઉમેરી/દૂર કરી શકાય છે
Iterable	એલિમેન્ટ્સમાં લૂપ ચલાવી શકાય છે

મૂળભૂત ઓપરેશન્સ:

Listing 1. Set Operations

```
# Create set
my_set = {1, 2, 3, 4}
# Add element
my_set.add(5)
# Remove element
my_set.remove(2)

```

મેમરી ટ્રીક

"Sets are Unique Unordered Collections"

પ્રશ્ન 1(b) [4 ગુણ]

પાયથોનમાં Tuple ની વ્યાખ્યા આપો? પાયથોનમાં Tuple data structure ના operations સમજાવો.

જવાબ

Tuple એ આઇટમ્સનો ક્રમિત સંગ્રહ છે જે immutable છે (બનાવ્યા પછી બદલી શકાતું નથી).
Tuple વ્યાખ્યા:
- **ક્રમિત**: એલિમેન્ટ્સનો નિશ્ચિત ક્રમ
- **Immutable**: બનાવ્યા પછી બદલી શકાતું નથી

- **ડુપ્લિકેટ્સ મંજૂર**: સમાન વેલ્યુઝ આવી શકે છે
- **ઇન્ડેક્સ્ડ**: ઇન્ડેક્સ વાપરીને એક્સેસ કરી શકાય છે

Tuple ઓપરેશન્સ:

કોષ્ટક 2. Tuple Operations

ઓપરેશન	ઉદાહરણ	વર્ણન
બનાવવું	t = (1, 2, 3)	Tuple બનાવો
ઇન્ડેક્સિંગ	t[0]	પ્રથમ એલિમેન્ટ એક્સેસ કરો
સ્લાઇસિંગ	t[1:3]	સબસેટ મેળવો
લેન્થ	len(t)	એલિમેન્ટ્સ ગણો
જોડાણ	t1 + t2	Tuples જોડો

Listing 2. Tuple Examples

```
# Example operations
tup = (10, 20, 30, 40)
print(tup[1])    # Output: 20
print(tup[1:3])   # Output: (20, 30)

```

મેમરી ટ્રીક

"Tuples are Immutable Ordered Collections"

પ્રશ્ન 1(c) [7 ગુણ]

પાયથોનમાં કન્સ્ટ્રક્ટરના પ્રકારો સમજાવો? Static methodનો ઉપયોગ કરીને બે સંખ્યાઓના ગુણાકાર માટે પાયથોન પ્રોગ્રામ લખો.

જવાબ

કન્સ્ટ્રક્ટરના પ્રકારો:

કોષ્ટક 3. Constructor Types

કન્સ્ટ્રક્ટર પ્રકાર	વર્ણન	ઉપયોગ
Default Constructor	કોઈ પેરામીટર નથી	__init__(self)
Parameterized Constructor	પેરામીટર લે છે	__init__(self, params)
Non-parameterized Constructor	ફક્ત self પેરામીટર	મૂળભૂત પ્રારંભિકરણ

Static Method પ્રોગ્રામ:

Listing 3. Static Method for Multiplication

```
class Calculator:
    def __init__(self):
        pass

    @staticmethod
    def multiply(num1, num2):
        return num1 * num2

# Usage
result = Calculator.multiply(5, 3)
print(f"Multiplication: {result}")  # Output: 15

```

મુખ્ય મુદ્દાઓ:
- **Static methods**: ઑબ્જેક્ટ ઇન્સ્ટન્સની જરૂર નથી
- **staticmethod decorator**: Static method દર્શાવે છે
- **કોઈ self પેરામીટર નથી**: ક્લાસ ઇન્સ્ટન્સથી સ્વતંત્ર

મેમરી ટ્રીક

"Static methods Stand Separate from Self"

પ્રશ્ન 1(c) અથવા [7 ગુણ]

Data Encapsulation ની વ્યાખ્યા આપો? પાયથોનમાં વિવિધ પ્રકારની methods ની યાદી આપો. Multilevel inheritance માટે પાયથોન પ્રોગ્રામ લખો.

જવાબ

Data Encapsulation: ડેટા એન્કેપ્સુલેશન એ ડેટા અને methods ને ક્લાસની અંદર બાંધવાની અને કેટલાક ઘટકોની સીધી પહોંચ મર્યાદિત કરવાની વિભાવના છે.

Methods ના પ્રકારો:

કોષ્ટક 4. Method Types

Method પ્રકાર	પહોંચ સ્તર	ઉદાહરણ
Public	બધે પહોંચી શકાય છે	method()
Protected	ક્લાસ અને સબક્લાસ	_method()
Private	ફક્ત ક્લાસની અંદર	__method()
Static	ક્લાસ લેવલ	staticmethod
Class	ક્લાસ અને સબક્લાસીઝ	classmethod

Multilevel Inheritance પ્રોગ્રામ:

Listing 4. Multilevel Inheritance

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} makes sound")

class Mammal(Animal):
    def __init__(self, name, warm_blooded):
        super().__init__(name)
        self.warm_blooded = warm_blooded

class Dog(Mammal):
    def __init__(self, name, breed):
        super().__init__(name, True)
        self.breed = breed

    def bark(self):
        print(f"{self.name} barks")

# Usage
dog = Dog("Buddy", "Golden Retriever")
dog.speak()  # From Animal class
```

```
dog.bark()  # From Dog class

```

મેમરી ટ્રીક

"Encapsulation Hides Internal Details"

પ્રશ્ન 2(a) [3 ગુણ]

Simple Queue અને Circular Queue વચ્ચેનો તફાવત આપો.

જવાબ

કોષ્ટક 5. Simple vs Circular Queue

લક્ષણ	Simple Queue	Circular Queue
સ્ટ્રક્ચર	રેખીય ગોઠવણી	વર્તુળાકાર ગોઠવણી
મેમરી ઉપયોગ	બગાડજનક (ખાલી જગ્યાઓ)	કાર્યક્ષમ (જગ્યા પુનઃઉપયોગ)
Rear Pointer	રેખીય રીતે આગળ વધે છે	ફરીથી આગળ વળે છે
Front Pointer	રેખીય રીતે આગળ વધે છે	ફરીથી આગળ વળે છે
જગ્યા ઉપયોગ	નબળો	ઉત્તમ

મુખ્ય તફાવતો:
- **Simple Queue**: Front અને rear ફક્ત એક દિશામાં જાય છે
- **Circular Queue**: Rear ફરીથી front સ્થાને જોડાય છે
- **કાર્યક્ષમતા**: Circular queue મેમરી વેસ્ટેજ ટાળે છે

મેમરી ટ્રીક

"Circular Queues Complete the Circle"

પ્રશ્ન 2(b) [4 ગુણ]

પાયથોનમાં પોલીમોર્ફિઝમ ઉદાહરણ સાથે સમજાવો.

જવાબ

Polymorphism એટલે "અનેક સ્વરૂપો" - સમાન method નામ અલગ અલગ ક્લાસમાં અલગ રીતે વર્તે છે.
પોલીમોર્ફિઝમના પ્રકારો:

કોષ્ટક 6. Polymorphism Types

પ્રકાર	વર્ણન	અમલીકરણ
Method Overriding	ચાઇલ્ડ ક્લાસ પેરેન્ટ method ફરીથી વ્યાખ્યાયિત કરે છે	Inheritance
Duck Typing	અલગ ક્લાસમાં સમાન method	Interface સમાનતા
Operator Overloading	સમાન ઓપરેટર અલગ વર્તન	Magic methods

Listing 5. Polymorphism Example

```
class Animal:
    def make_sound(self):
```

```
        pass

class Dog(Animal):
    def make_sound(self):
        return "Woof!"

class Cat(Animal):
    def make_sound(self):
        return "Meow!"

# Polymorphic behavior
animals = [Dog(), Cat()]
for animal in animals:
    print(animal.make_sound())

```

મેમરી ટ્રીક

"Polymorphism Provides Multiple Personalities"

પ્રશ્ન 2(c) [7 ગુણ]

વ્યાખ્યા આપો. a). Infix b).Postfix સ્ટેકનો ઉપયોગ કરીને આપેલ Infix expression ને Postfix expression માં ફેરવો. A+(B*C/D)

જવાબ

વ્યાખ્યાઓ:

કોષ્ટક 7. Expression Types

Expression પ્રકાર	વર્ણન	ઉદાહરણ
Infix	ઓપરેટર ઓપરેન્ડ્સ વચ્ચે	A + B
Postfix	ઓપરેટર ઓપરેન્ડ્સ પછી	A B +

રૂપાંતરણ એલ્ગોરિધમ:

1. Infix expression ને ડાબેથી જમણે સ્કેન કરો.
2. જો operand છે, આઉટપુટમાં ઉમેરો.
3. જો operator છે, સ્ટેક ટોપ સાથે precedence સરખાવો.
4. વધુ precedence → સ્ટેકમાં push કરો.
5. ઓછી/સમાન precedence → pop કરીને આઉટપુટમાં ઉમેરો.

પગલાં પ્રમાણે રૂપાંતરણ: A+(B*C/D)

કોષ્ટક 8. Infix to Postfix Trace

પગલું	સિમ્બલ	સ્ટેક	આઉટપુટ
1	A	[ ]	A
2	+	[+]	A
3	(	[+, (]	A
4	B	[+, (]	AB
5	*	[+, (, *]	AB
6	C	[+, (, *]	ABC
7	/	[+, (, /]	ABC*
8	D	[+, (, /]	ABC*D
9	)	[+]	ABC*D/
10	અંત	[ ]	ABC*D/+

અંતિમ જવાબ: ABC*D/+

મેમરી ટ્રીક

"Stack Stores Operators Strategically"

પ્રશ્ન 2(a) અથવા [3 ગુણ]

Queue ના ગેરફાયદા સમજાવો.

જવાબ

કોષ્ટક 9. Queue Disadvantages

ગેરફાયદો	વર્ણન	અસર
મેમરી વેસ્ટેજ	ખાલી જગ્યાઓ પુન:ઉપયોગ નથી	નબળો જગ્યા ઉપયોગ
નિયત કદ	મર્યાદિત ક્ષમતા	Overflow સમસ્યાઓ
રેન્ડમ એક્સેસ નથી	ફક્ત front/rear એક્સેસ	મર્યાદિત લવચીકતા

મુખ્ય સમસ્યાઓ:
- **રેખીય Queue**: આગળની જગ્યાઓ અનુપયોગી બને છે
- **Insertion/Deletion**: ફક્ત ચોક્કસ છેડાઓથી
- **શોધ ઓપરેશન્સ**: શોધવા માટે કાર્યક્ષમ નથી

મેમરી ટ્રીક

"Queues Quietly Queue with Quirks"

પ્રશ્ન 2(b) અથવા [4 ગુણ]

પાયથોનમાં Abstract class ની વ્યાખ્યા આપો? પાયથોનમાં abstract method નું declaration સમજાવો?

જવાબ

Abstract Class: એક ક્લાસ જેનું instantiation કરી શકાતું નથી અને જેમાં એક અથવા વધુ abstract methods હોય છે જે સબક્લાસીઝ દ્વારા અમલ કરવા જોઈએ.
Abstract Method Declaration:

કોષ્ટક 10. Abstract Methods

ઘટક	હેતુ	સિન્ટેક્સ
ABC Module	Abstract base class પ્રદાન કરે છે	from abc import ABC
@abstractmethod	Abstract methods માટે decorator	@abstractmethod
અમલીકરણ	સબક્લાસમાં ફરજિયાત override	આવશ્યક

Listing 6. Abstract Class Example

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

```

મેમરી ટ્રીક

"Abstract classes Are Blueprints Only"

પ્રશ્ન 2(c) અથવા [7 ગુણ]

Infix to postfix માટેનો અલ્ગોરિધમ લખો. નીચેની પોસ્ટફિક્સ એક્સપ્રેશન મૂલ્યાંકન કરો. 5 6 2 + * 12 4 / -

જવાબ

Infix to Postfix અલ્ગોરિધમ:
1. ખાલી સ્ટેક અને આઉટપુટ સ્ટ્રિંગ પ્રારંભ કરો.
2. ડાબેથી જમણે infix expression સ્કેન કરો.
3. જો operand છે → આઉટપુટમાં ઉમેરો.
4. જો '(' છે → સ્ટેકમાં push કરો.
5. જો ')' છે → '(' સુધી pop કરો.
6. જો operator છે → વધુ/સમાન precedence operators pop કરો.
7. વર્તમાન operator સ્ટેકમાં push કરો.
8. બાકીના operators pop કરો.

Postfix મૂલ્યાંકન: 5 6 2 + * 12 4 / -

કોષ્ટક 11. Postfix Evaluation Trace

પગલું	ટોકન	સ્ટેક	ઓપરેશન
1	5	[5]	Operand push કરો
2	6	[5, 6]	Operand push કરો
3	2	[5, 6, 2]	Operand push કરો
4	+	[5, 8]	Pop 2, 6 $\rightarrow 6 + 2 = 8$
5	*	[40]	Pop 8, 5 $\rightarrow 5 * 8 = 40$
6	12	[40, 12]	Operand push કરો
7	4	[40, 12, 4]	Operand push કરો
8	/	[40, 3]	Pop 4, 12 $\rightarrow 12/4 = 3$
9	-	[37]	Pop 3, 40 $\rightarrow 40 - 3 = 37$

અંતિમ પરિણામ: 37

મેમરી ટ્રીક

"Postfix Processing Pops Pairs Precisely"

પ્રશ્ન 3(a) [3 ગુણ]

સિંગલ લિંક લિસ્ટમાં નોડને traverse કરવા માટે અલ્ગોરિધમ લખો.

જવાબ

Traversal અલ્ગોરિધમ:

Listing 7. Linked List Traversal

```
def traverse_linked_list(head):
    current = head
    while current is not None:
        print(current.data)
        current = current.next

```

અલ્ગોરિધમ પગલાં:

1. Head નોડથી શરૂ કરો.
2. ચકાસો કે current $\neq$ NULL.
3. વર્તમાન નોડ પ્રોસેસ કરો.
4. આગલા નોડ પર જાઓ.
5. અંત સુધી પુનરાવર્તન કરો.

મેમરી ટ્રીક

"Traverse Through Till The Tail"

પ્રશ્ન 3(b) [4 ગુણ]

લિસ્ટનો ઉપયોગ કરીને Queueના Dequeue ઓપરેશન માટેનો અલ્ગોરિધમ લખો.

જવાબ

Dequeue અલ્ગોરિધમ:

Listing 8. Dequeue Operation

```
def dequeue(queue):
    if len(queue) == 0:
        print("Queue is empty")
        return None
    else:
        element = queue.pop(0)
        return element

```

અલ્ગોરિધમ પગલાં:
1. ખાલી ચકાસો: જો queue ખાલી છે.
2. Underflow હેન્ડલ કરો: એરર મેસેજ દર્શાવો.
3. એલિમેન્ટ દૂર કરો: આગળનું એલિમેન્ટ ડિલીટ કરો.
4. એલિમેન્ટ પરત કરો: દૂર કરેલી વેલ્યુ પરત કરો.
5. સ્ટ્રક્ચર અપડેટ કરો: Queue pointers એડજસ્ટ કરો.

Time Complexity: $O(n)$ લિસ્ટ shifting ને કારણે

મેમરી ટ્રીક

"Dequeue Deletes from Front Door"

પ્રશ્ન 3(c) [7 ગુણ]

Double linked list ની વ્યાખ્યા આપો. લિંક લિસ્ટના મુખ્ય ઓપરેશનની નોંધણી કરો. Single Linked list મા શરૂઆતમાં નોડ દાખલ કરવા માટેનો અલ્ગોરિધમ લખો.

જવાબ

Double Linked List: એક રેખીય ડેટા સ્ટ્રક્ચર જ્યાં દરેક નોડમાં ડેટા અને બે pointers હોય છે - એક આગલા નોડ તરફ અને બીજો પાછલા નોડ તરફ.

લિંક લિસ્ટના મુખ્ય ઓપરેશન્સ:

કોષ્ટક 12. Linked List Operations

ઓપરેશન	વર્ણન	Time Complexity
Insertion	નવો નોડ ઉમેરો	$O(1)$ શરૂઆતમાં
Deletion	નોડ દૂર કરો	$O(1)$ જો નોડ ખબર છે
Traversal	બધા નોડ્સની મુલાકાત લો	$O(n)$
Search	ચોક્કસ નોડ શોધો	$O(n)$
Update	નોડ ડેટા બદલો	$O(1)$ જો નોડ ખબર છે

શરૂઆતમાં Insert અલ્ગોરિધમ:

Listing 9. Insert at Beginning

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def insert_at_beginning(head, data):
    new_node = Node(data)
```

```
    new_node.next = head
    head = new_node
    return head

```

અલ્ગોરિધમ પગલાં:
1. આપેલ ડેટા સાથે નવો નોડ બનાવો.
2. નવા નોડનો next વર્તમાન head પર સેટ કરો.
3. Head ને નવા નોડ તરફ અપડેટ કરો.
4. નવો head પરત કરો.

મેમરી ટ્રીક

"Insert at Beginning Builds Better Lists"

પ્રશ્ન 3(a) અથવા [3 ગુણ]

Single Linked List ની એપ્લિકેશન સમજાવો.

જવાબ

Single Linked List એપ્લિકેશન્સ:

કોષ્ટક 13. Applications

એપ્લિકેશન	ઉપયોગ કેસ	ફાયદો
Dynamic Memory	વેરિયેબલ સાઇઝ ડેટા	મેમરી કાર્યક્ષમ
Stack Implementation	LIFO ઓપરેશન્સ	સરળ push/pop
Queue Implementation	FIFO ઓપરેશન્સ	Dynamic sizing
Music Playlist	સિક્વેન્શિયલ પ્લેબેક	સરળ નેવિગેશન
Browser History	પેજ નેવિગેશન	ફોરવર્ડ traversal
Polynomial Representation	ગાણિતિક ઓપરેશન્સ	Coefficient સ્ટોરેજ

મુખ્ય ફાયદાઓ:
- **Dynamic Size**: રનટાઇમ દરમિયાન વધે/ઘટે છે
- **મેમરી કાર્યક્ષમતા**: જરૂર પ્રમાણે allocate કરે છે
- **Insertion/Deletion**: કોઈપણ સ્થાને કાર્યક્ષમ

મેમરી ટ્રીક

"Linked Lists Link Many Applications"

પ્રશ્ન 3(b) અથવા [4 ગુણ]

લિસ્ટનો ઉપયોગ કરીને સ્ટેકના PUSH ઓપરેશન માટે અલ્ગોરિધમ લખો.

જવાબ

PUSH અલ્ગોરિધમ:

Listing 10. Stack Push

```
def push(stack, element):
```

```
    stack.append(element)
    print(f"Pushed {element} to stack")

```

અલ્ગોરિધમ પગલાં:
1. **ક્ષમતા ચકાસો**: સ્ટેક ભરાયો નથી તે ચકાસો (fixed size માટે).
2. **એલિમેન્ટ ઉમેરો**: લિસ્ટના અંતે append કરો.
3. **ટોપ અપડેટ કરો**: ટોપ છેલ્લા એલિમેન્ટ તરફ પોઇન્ટ કરે છે.
4. **ઓપરેશન કન્ફર્મ કરો**: સફળતાનો મેસેજ દર્શાવો.

Time Complexity: O(1)

મેમરી ટ્રીક

"PUSH Puts on Stack Summit"

પ્રશ્ન 3(c) અથવા [7 ગુણ]

Linked list ના ફાયદા સમજાવો. Single linked list માંથી last નોડ કાઢી નાખવા માટે અલ્ગોરિધમ લખો.

જવાબ

Linked List ફાયદાઓ:

કોષ્ટક 14. Advantages

ફાયદો	વર્ણન	લાભ
Dynamic Size	રનટાઇમે સાઇઝ બદલાય છે	મેમરી લવચીક
મેમરી કાર્યક્ષમ	જરૂર પ્રમાણે allocate કરે છે	કોઈ વેસ્ટેજ નથી
સરળ Insertion	ગમે ત્યાં કાર્યક્ષમ રીતે ઉમેરો	O(1) ઓપરેશન
સરળ Deletion	કાર્યક્ષમ રીતે દૂર કરો	O(1) ઓપરેશન
મેમરી Shift નથી	એલિમેન્ટ્સ ખસતા નથી	ઝડપી ઓપરેશન્સ

છેલ્લો નોડ ડિલીટ કરવાનો અલ્ગોરિધમ:

Listing 11. Delete Last Node

```
def delete_last_node(head):
    # Empty list
    if head is None:
        return None

    # Single node
    if head.next is None:
        return None

    # Multiple nodes
    current = head
    while current.next.next is not None:
        current = current.next

    current.next = None
    return head

```

મેમરી ટ્રીક

"Linked Lists Lead to Logical Advantages"

પ્રશ્ન 4(a) [3 ગુણ]

બબલ સૉર્ટમાટેના અલ્ગોરિધમ લખો.

જવાબ

Bubble Sort અલ્ગોરિધમ:

Listing 12. Bubble Sort

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

```

Time Complexity: $O(n^2)$

મેમરી ટ્રીક

"Bubbles Rise to Surface Slowly"

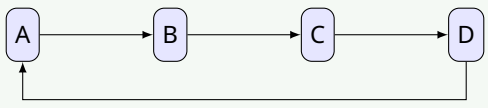
પ્રશ્ન 4(b) [4 ગુણ]

Circular linked list ને તેના ફાયદાઓ સાથે સમજાવો.

જવાબ

Circular Linked List: એક લિંક લિસ્ટ જ્યાં છેલ્લો નોડ પ્રથમ નોડ તરફ પોઇન્ટ કરે છે, વર્તુળાકાર સ્ટ્રક્ચર બનાવે છે.
ફાયદાઓ:

- **મેમરી કાર્યક્ષમ**: કોઈ NULL pointers નથી
- **વર્તુળાકાર Traversal**: સતત લૂપ કરી શકાય છે
- **Queue Implementation**: કાર્યક્ષમ enqueue/dequeue
- **Round Robin Scheduling**: CPU time sharing



આકૃતિ 1. Circular Linked List

મેમરી ટ્રીક

"Circular Lists Create Continuous Connections"

પ્રશ્ન 4(c) [7 ગુણ]

યોગ્ય ઉદાહરણ સાથે મર્જ સૉર્ટ સમજાવો.

જવાબ

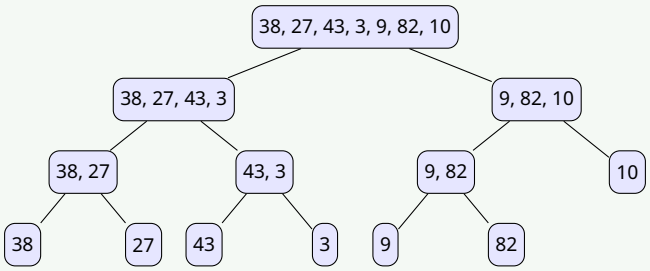
Merge Sort એક divide-and-conquer અલ્ગોરિધમ જે array ને ભાગોમાં વહેંચે છે, તેમને અલગ અલગ સૉર્ટ કરે છે, અને પછી ભેગા કરે છે.

અલ્ગોરિધમ તબક્કાઓ:

કોષ્ટક 15. Merge Sort Phases

તબક્કો	ક્રિયા	વર્ણન
Divide	Array વહેંચો	બે ભાગોમાં વહેંચો
Conquer	Subarrays સૉર્ટ કરો	ભાગોને recursively સૉર્ટ કરો
Combine	પરિણામો મર્જ કરો	સૉર્ટેડ ભાગો મર્જ કરો

ઉદાહરણ: [38, 27, 43, 3, 9, 82, 10]



આકૃતિ 2. Merge Sort Division

મર્જ પરિણામ: [3, 9, 10, 27, 38, 43, 82]
Time Complexity: O(n log n)

મેમરી ટ્રીક

"Merge Sort Methodically Merges Segments"

પ્રશ્ન 4(a) અથવા [3 ગુણ]

Selection sort માટેના અલ્ગોરિધમ લખો.

જવાબ

Selection Sort અલ્ગોરિધમ:

Listing 13. Selection Sort

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

```

Time Complexity: $O(n^2)$

મેમરી ટ્રીક

"Selection Sort Selects Smallest Successfully"

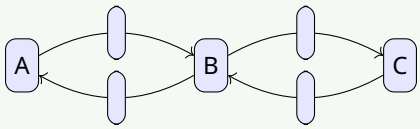
પ્રશ્ન 4(b) અથવા [4 ગુણ]

Double linked list ને તેના ફાયદાઓ સાથે સમજાવો.

જવાબ

Double Linked List: એક લિંક લિસ્ટ જ્યાં દરેક નોડમાં ડેટા અને બે pointers હોય છે - next અને previous.
ફાયદાઓ:

- **બાયડાયરેક્શનલ Traversal**: આગળ અને પાછળ બંને દિશામાં ચાલી શકાય છે
- **સરળ Deletion**: પાછલો નોડ જાણ્યા વગર ડિલીટ કરી શકાય છે
- **કાર્યક્ષમ Insertion**: કોઈપણ સ્થાને સરળતાથી insert કરી શકાય છે



આકૃતિ 3. Double Linked List

મેમરી ટ્રીક

"Double Links provide Dual Direction"

પ્રશ્ન 4(c) અથવા [7 ગુણ]

Insertion સૉર્ટ સમજાવો. Insertion સૉર્ટનો ઉપયોગ કરીને નીચેના નંબરોનો ટ્રેસ આપો: 25, 15,30,9,99,20,26

જવાબ

Insertion Sort: એક સમયે એક એલિમેન્ટ દ્વારા સૉર્ટેડ array બનાવે છે દરેક એલિમેન્ટને તેની યોગ્ય પોઝિશનમાં insert કરીને.
ટ્રેસ મૂલ્યાંકન:

કોષ્ટક 16. Insertion Sort Trace

પાસ	વર્તમાન	Array સ્થિતિ	ક્રિયા
પ્રારંભિક	-	[25, 15, 30, 9, 99, 20, 26]	શરૂ
1	15	[15, 25, 30, 9, 99, 20, 26]	Insert 15
2	30	[15, 25, 30, 9, 99, 20, 26]	No change
3	9	[9, 15, 25, 30, 99, 20, 26]	Insert 9
4	99	[9, 15, 25, 30, 99, 20, 26]	No change
5	20	[9, 15, 20, 25, 30, 99, 26]	Insert 20
6	26	[9, 15, 20, 25, 26, 30, 99]	Insert 26

અંતિમ સૉર્ટેડ Array: [9, 15, 20, 25, 26, 30, 99]

મેમરી ટ્રીક

"Insertion Inserts Into Increasing Order"

પ્રશ્ન 5(a) [3 ગુણ]

બાઈનરી ટ્રીની એપ્લિકેશન સમજાવો.

જવાબ

Binary Tree એપ્લિકેશન્સ:

કોષ્ટક 17. Applications

એપ્લિકેશન	ઉપયોગ કેસ	ફાયદો
Expression Trees	ગાણિતિક expressions	સરળ evaluation
Binary Search Trees	Searching/Sorting	O(log n) operations
Heap Trees	Priority queues	કાર્યક્ષમ min/max
File Systems	Directory structure	હાયરાર્કિકલ ઓર્ગેનાઇઝેશન

મેમરી ટ્રીક

"Binary Trees Branch into Many Applications"

પ્રશ્ન 5(b) [4 ગુણ]

લિસ્ટનો ઉપયોગ કરીને Binary search માટેનો અલ્ગોરિધમ લખો.

જવાબ

Binary Search અલ્ગોરિધમ:

Listing 14. Binary Search

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1

```

પૂર્વશરત: Array સૉર્ટેડ હોવું જોઈએ. **Time Complexity: O(log n)**

મેમરી ટ્રીક

"Binary Search Bisects to Find Faster"

પ્રશ્ન 5(c) [7 ગુણ]

Tree ની વ્યાખ્યા આપો. Tree ની યાદી બનાવો. પાયથોનનો ઉપયોગ કરીને બાઈનરી સર્ચ ટ્રીમાં નોડ દાખલ કરવા માટે અલ્ગોરિધમ લખો.

જવાબ

Tree: એક હાયરાર્કિકલ ડેટા સ્ટ્રક્ચર જેમાં edges દ્વારા જોડાયેલા nodes હોય છે, એક root node સાથે અને કોઈ cycles ન હોય.
Tree ના પ્રકારો:

- **Binary Tree**: નોડ દીઠ વધુમાં વધુ 2 બાળકો.
- **Binary Search Tree**: ક્રમિત binary tree (ડાબું < Root < જમણું).
- **Complete Binary Tree**: છેલ્લા સિવાય બધા લેવલ ભરેલા.
- **Full Binary Tree**: બધા nodes ને 0 અથવા 2 બાળકો.
- **AVL Tree**: સ્વ-સંતુલિત BST.

BST Insertion અલ્ગોરિધમ:

Listing 15. BST Insertion

```
class TreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def insert_bst(root, data):
    if root is None:
        return TreeNode(data)

    if data < root.data:
        root.left = insert_bst(root.left, data)
    elif data > root.data:
        root.right = insert_bst(root.right, data)

    return root

```

મેમરી ટ્રીક

"Trees Grow with Structured Organization"

પ્રશ્ન 5(a) અથવા [3 ગુણ]

ટ્રીના ઇન-ઓર્ડર ટ્રાવર્સલનો અલ્ગોરિધમ લખો.

જવાબ

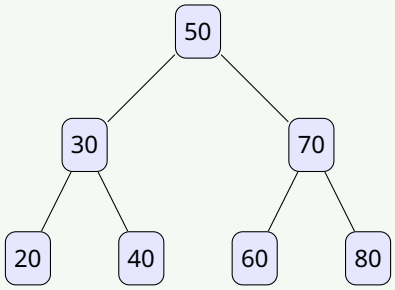
In-order Traversal અલ્ગોરિધમ:

Listing 16. In-order Traversal

```python
def inorder_traversal(root):
    if root is not None:
        inorder_traversal(root.left)   # Left
        print(root.data)               # Root
        inorder_traversal(root.right)  # Right

```

Traversal ક્રમ: ડાબું → Root → જમણું



આકૃતિ 4. Example Tree

In-order: 20, 30, 40, 50, 60, 70, 80

મેમરી ટ્રીક

"In-order: Left, Root, Right"

પ્રશ્ન 5(b) અથવા [4 ગુણ]

Search ની વ્યાખ્યા આપો? લિસ્ટનો ઉપયોગ કરીને Linear search માટેનો અલ્ગોરિધમ લખો.

જવાબ

Search: ડેટા સ્ટ્રક્ચરમાં ચોક્કસ એલિમેન્ટ શોધવાની અથવા એલિમેન્ટ અસ્તિત્વમાં છે કે નહીં તે ચકાસવાની પ્રક્રિયા.
Linear Search અલ્ગોરિધમ:

Listing 17. Linear Search

```python
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i  # Return index if found
    return -1  # Return -1 if not found

```

Time Complexity: O(n)

મેમરી ટ્રીક

"Linear Search Looks through Lists Linearly"

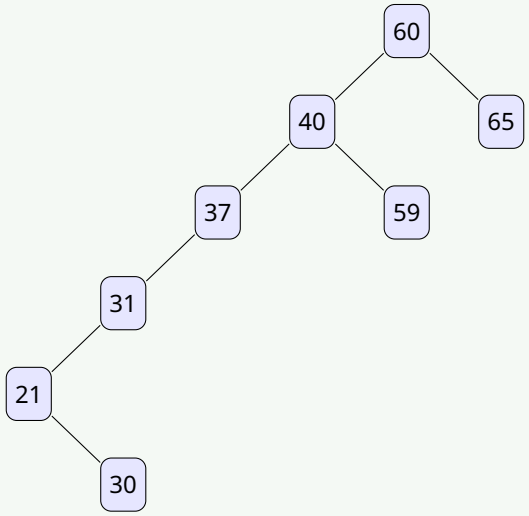
પ્રશ્ન 5(c) અથવા [7 ગુણ]

વ્યાખ્યા આપો: a) પાથ b). લીફ નોડ. નીચે આપેલ માહિતી ઉપરથી binary search tree બનાવો. 60, 40, 37,31,59,21,65,30

જવાબ

વ્યાખ્યાઓ:
- **Path**: એક નોડથી બીજા નોડ સુધીના nodes ની શ્રેણી.
- **Leaf Node**: કોઈ બાળકો ન હોય તેવો નોડ (કોઈ ડાબું કે જમણું બાળક નથી).

BST બનાવટ માટે: 60, 40, 37, 31, 59, 21, 65, 30



આકૃતિ 5. Final Binary Search Tree

Leaf Nodes: 30, 59, 65

મેમરી ટ્રીક

"BST Building follows Binary Search Tree rules"