

Subject Name (Gujarati)

1333203 -- Summer 2024

Semester 1 Study Material

Detailed Solutions and Explanations

પ્રશ્ન 1(અ) [3 ગુણ]

રેખીય ડેટા સ્ટ્રક્ચર વ્યાખ્યાપિત કરો અને તેના ઉદાહરણો આપો.

જવાબ

રેખીય ડેટા સ્ટ્રક્ચર એ એલિમેન્ટ્સનો એવો સંગ્રહ છે કે જેમાં દરેક એલિમેન્ટની પહેલાં અને પછી એક જ એલિમેન્ટ હોય છે (સિવાય કે પ્રથમ અને છેલ્લા એલિમેન્ટ).

Table 1: રેખીય ડેટા સ્ટ્રક્ચરના ઉદાહરણો

ડેટા સ્ટ્રક્ચર	વર્ણન
Array	નિશ્ચિત સાઇઝનો એલિમેન્ટ્સનો સંગ્રહ જે ઇન્ડેક્સ દ્વારા ઓક્સેસ થાય છે
Linked List	નોડ્સની શ્રેણી જેમાં ડેટા અને આગામના નોડનો રેફરન્સ હોય છે
Stack	LIFO (લાસ્ટ ઇન ફર્સ્ટ આઉટ) સ્ટ્રક્ચર
Queue	FIFO (ફર્સ્ટ ઇન ફર્સ્ટ આઉટ) સ્ટ્રક્ચર

મેમરી ટ્રીક

"ALSQ એ લાઇનમાં છે"

પ્રશ્ન 1(બ) [4 ગુણ]

ટાઇમ અને સ્પેસ કોમ્પ્લેક્સિટી વ્યાખ્યાપિત કરો.

જવાબ

ટાઇમ અને સ્પેસ કોમ્પ્લેક્સિટી એલગોરિધમની કાર્યક્ષમતાને એક્ઝિક્યુશન ટાઇમ અને મેમરી વપરાશના સંદર્ભમાં માપે છે, જેમ ઇનપુટ સાઇઝ વધે છે.

Table 2: કોમ્પ્લેક્સિટી કમ્પેરિઝન

કોમ્પ્લેક્સિટી પ્રકાર	વ્યાખ્યા	માપન	મહત્વ
ટાઇમ કોમ્પ્લેક્સિટી	એલગોરિધમના એક્ઝિક્યુશન ટાઇમને ઇનપુટ સાઇઝના ફુંક્શન તરીકે માપે છે	બિગ O નોટેશન ($O(n)$, $O(1)$, $O(n^2)$)	એલગોરિધમ કેટલી જડપથી ચાલે છે તે નક્કી કરે છે
સ્પેસ કોમ્પ્લેક્સિટી	એલગોરિધમને જરૂરી મેમરી સ્પેસને ઇનપુટ સાઇઝના ફુંક્શન તરીકે માપે છે	બિગ O નોટેશન ($O(n)$, $O(1)$, $O(n^2)$)	એલગોરિધમને કેટલી મેમરી જોઈએ છે તે નક્કી કરે છે

મેમરી ટ્રીક

"TS: ટાઇમ-સ્પીડ અને સ્પેસ-સ્ટોરેજ"

પ્રશ્ન 1(ક) [7 ગુણ]

કલાસ અને ઓફજેક્ટ ઉદાહરણ સાથે સમજાવો.

જવાબ

કલાસ અને ઓફ્જેક્ટ એ OOP ના મૂળભૂત કોન્સેપ્ટ છે જ્યાં કલાસ એ એટ્રિબ્યુટ્સ અને બિહેવિયર્સ ધરાવતા ઓફ્જેક્ટ બનાવવા માટેના બલુપ્રેન્ટ છે.

ડાયાગ્રામ: કલાસ અને ઓફ્જેક્ટ રિલેશનશિપ

```
classDiagram
    class Student {
        +name: string
        +rollNo: int
        +marks: float
        +displayInfo()
    }
    Student { |{-}{-} StudentObject: Creates}
```

કોડ ઉદાહરણ:

```
class Student:
    def __init__(self, name, rollNo, marks):
        self.name = name
        self.rollNo = rollNo
        self.marks = marks

    def displayInfo(self):
        print(f"Name: {self.name}, Roll No: {self.rollNo}, Marks: {self.marks}")

# student1 = Student(" ", 101, 85.5)
student1.displayInfo()
```

- કલાસ: એટ્રિબ્યુટ્સ (name, rollNo, marks) અને મેથડ્સ (displayInfo) વ્યાખ્યાયિત કરતા બલુપ્રેન્ટ
- ઓફ્જેક્ટ: કલાસથી બનાવેલ ઇન્સ્ટન્સ (student1) જેમાં ચોક્કસ વેલ્યુઝ હોય છે

મેમરી ટ્રીક

"CAR - કલાસ એટ્રિબ્યુટ્સ અને રૂટિન્સ વ્યાખ્યાયિત કરે છે"

પ્રશ્ન 1(ક) OR [7 ગુણ]

ઇન્સ્ટાન્સ મેથડ, કલાસ મેથડ અને સ્ટેટિક મેથડ ઉદાહરણ સાથે સમજાવો.

જવાબ

Python ત્રણ પ્રકારની મેથડ્સને સપોર્ટ કરે છે: ઇન્સ્ટન્સ, કલાસ અને સ્ટેટિક મેથડ, દરેક અલગ હેતુ માટે વપરાય છે.

Table 3: મેથડ પ્રકારોની તુલના

મેથડ પ્રકાર	ડેકોરેટર	પ્રથમ પેરામીટર	હેતુ	એક્સેસ
ઇન્સ્ટન્સ મેથડ	કોઈ નહીં	self	ઇન્સ્ટન્સ ડેટા પર કામ કરે	ઇન્સ્ટન્સ સ્ટેટને એક્સેસ/મૌંડિફાઇ કરી શકે
કલાસ મેથડ	@classmethod	cls	કલાસ ડેટા પર કામ કરે	કલાસ સ્ટેટને એક્સેસ/મૌંડિફાઇ કરી શકે
સ્ટેટિક મેથડ	@staticmethod	કોઈ નહીં	યુટિલિટી ફંક્શન્સ	ઇન્સ્ટન્સ કે કલાસ સ્ટેટને એક્સેસ કરી શકતી નથી

કોડ ઉદાહરણ:

```
class Student:  
    school = "ABC School"  #  
  
    def __init__(self, name):  
        self.name = name  #  
  
    def instance_method(self):  #  
        return f"Hi {self.name} from {self.school}"  
  
    @classmethod  
    def class_method(cls):  #  
        return f"School is {cls.school}"  
  
    @staticmethod  
    def static_method():  #  
        return "This is a utility function"
```

મેમરી ટ્રીક

“ICS: ઇન્સ્ટન્સ-Self, કલાસ-Clસ, સ્ટેટિક-Solo”

પ્રશ્ન 2(અ) [3 ગુણ]

રીકર્જન ફંક્શન નો કોરોટ સમજાવો.

જવાબ

રિકર્જન ફંક્શન એ એટું ફંક્શન છે જે પોતાની એક્ઝિક્યુશન દરમિયાન સમાન સમસ્યાના નાના ઉદાહરણોને હલ કરવા માટે પોતાને જ કોલ કરે છે.

ડાયાગ્રામ: રિકર્જન ફંક્શન એક્ઝિક્યુશન

Mermaid Diagram (Code)

```
{Shaded}  
{Highlighting} []  
graph LR  
    A["factorial(3)"] --{-{-}{}} B["factorial(2)"]  
    B --{-{-}{}} C["factorial(1)"]  
    C --{-{-}{}} D[Return 1]  
    D --{-{-}{}} E[Return 1*2=2]  
    E --{-{-}{}} F[Return 2*3=6]  
{Highlighting}  
{Shaded}
```

મેમરી ટ્રીક

“BASE અને RECURSE - બેઝ કેસ સ્ટોચ્સ, રિકર્જન રિપીટ્સ”

પ્રશ્ન 2(બ) [4 ગુણ]

સ્ટેક અને ક્યુ વ્યાખ્યાપિત કરો.

જવાબ

સ્ટેક અને ક્યુ એ લીનિયર ડેટા સ્ટ્રક્ચર છે જેમાં ડેટા ઇન્સર્શન અને રિમૂવલ માટે અલગ એક્સેસ પેટર્ન છે.

Table 4: સ્ટેક વિ. ક્યુ

ફીચર	સ્ટેક	ક્યુ
એક્સેસ પેટર્ન	LIFO (લાસ્ટ ઇન ફર્સ્ટ આઉટ)	FIFO (ફર્સ્ટ ઇન ફર્સ્ટ આઉટ)
ઓપરેશન્સ	પુશ (ઇન્સર્ટ), પોપ (રિમૂવ)	અન્ક્યુ (ઇન્સર્ટ), ડિક્યુ (રિમૂવ)
એક્સેસ પોઇન્ટ્સ	સિંગલ એન્ડ (ટોપ)	ટુ એન્ડ્સ (ફાન્ટ, રિપર)
વિઝ્યુઅલાઇઝેશન	ઉિબા થાંભલામાં ગોઠવેલી થાળીઓ જેવું	લાઇનમાં ઉિબેલા લોકો જેવું
એપ્લિકેશન્સ	ફંક્શન કોલ્સ, અન્ડુ ઓપરેશન્સ	પ્રિન્ટ જોબ્સ, પ્રોસેસ શોક્યુલિંગ

મેમરી ટ્રીક

"SLIFF vs QFIFF - સ્ટેક-LIFO vs ક્યુ-FIFO"

પ્રશ્ન 2(ક) [7 ગુણ]

સ્ટેક ના બેઝિક ઓપરેશન સમજાવો.

જવાબ

સ્ટેક ઓપરેશન્સ LIFO (લાસ્ટ ઇન ફર્સ્ટ આઉટ) સિદ્ધાંતને અનુસરે છે અને નીચેના મૂળભૂત ઓપરેશન્સ ધરાવે છે:

Table 5: સ્ટેક ઓપરેશન્સ

ઓપરેશન	વર્ણન	ટાઈમ કોમ્પ્લેક્શન્ટી
પુશ	ટોપ પર એલિમેન્ટ ઇન્સર્ટ કરવું	O(1)
પોપ	ટોપથી એલિમેન્ટ રિમૂવ કરવું	O(1)
પીક/ટોપ	રિમૂવ કર્યા વિના ટોપ એલિમેન્ટ જોવું	O(1)
isEmpty	ચેક કરવું કે સ્ટેક ખાલી છે કે નહીં	O(1)
isFull	ચેક કરવું કે સ્ટેક ભરેલો છે કે નહીં (એરે ઇમ્પિલમેન્ટેશન માટે)	O(1)

ડાયાગ્રામ: સ્ટેક ઓપરેશન્સ

```
+{--}{-}+ Push
| 8 | {{-}{-}{-}{-}{-}{-}{-}{-}}
Top {- +{-}{-}{-}+
| 5 | Pop
+{--}{-}+ {-}{-}{-}{-}{-}{-}{-}
| 3 |
+{--}{-}+
| 1 |
+{--}{-}+
```

કોડ ઉદાહરણ:

```
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.isEmpty():
            return self.items.pop()

    def peek(self):
        if not self.isEmpty():
            return self.items[-1]

    def isEmpty(self):
        return len(self.items) == 0
```

મેમરી ટ્રીક

“PIPES - Push In, Pop Exit, See top”

પ્રશ્ન 2(અ) OR [3 ગુણ]

સિંગલી લિંકડ લિસ્ટ વ્યાખ્યાયિત કરો.

જવાબ

સિંગલી લિંકડ લિસ્ટ એ એક લીનિયર ડેટા સ્ટ્રક્ચર છે જેમાં નોડ્સનો કલેક્શન હોય છે જ્યાં દરેક નોડમાં ડેટા અને આગળના નોડનો રેફરન્સ હોય છે.

ડાયાગ્રામ: સિંગલી લિંકડ લિસ્ટ

```
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+ +{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+ +{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+
| Data:10 |     | Data:20 |     | Data:30 |     | Data:40 |
| Next:{-}{-}|{-}{-}{-}{-}| Next:{-}{-}|{-}{-}{-}{-}| Next:{-}{-}|{-}{-}{-}{-}| Next:/0|
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+ +{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+ +{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+
Head Node                                     Tail Node
```

મેમરી ટ્રીક

“DNL - ડેટા અને નેકસ્ટ લિંક”

પ્રશ્ન 2(બ) OR [4 ગુણ]

ક્યુ ઉપર એનક્યુ ડિક્યુ ઓપરેશન સમજાવો.

જવાબ

એનક્યુ અને ડિક્યુ ક્યુ ડેટા સ્ટ્રક્ચરમાં એલિમેન્ટ્સ ઉમેરવા અને કાઢવા માટેના મુખ્ય ઓપરેશન્સ છે.

Table 6: ક્યુ ઓપરેશન્સ

ઓપરેશન	વર્ણન	ઇમ્પ્લેમેન્ટેશન	ટાઈમ કોમ્પ્લેક્ઝન્ટી
એનક્યુ	રિચર એન્ડ પર એલિમેન્ટ ઉમેરવું	queue.append(element)	O(1)
ડિક્યુ	ફાન્ટ એન્ડથી એલિમેન્ટ કાઢવું	element = queue.pop(0)	લિંકડ લિસ્ટ સાથે O(1), એરે સાથે O(n)

ડાયાગ્રામ: ક્યુ ઓપરેશન્સ

Enqueue	Dequeue
$\{ - \} \{ - \} \{ - \} \{ - \} \{ - \} \{ - \} \{ - \}$ $+ \{ - \} \{ - \} \{ - \} \{ - \} +$ Rear 30 20 10 Front $+ \{ - \} \{ - \} \{ - \} +$	$\{ - \} \{ - \} \{ - \} \{ - \} \{ - \} \{ - \} \{ - \}$ $+ \{ - \} \{ - \} \{ - \} \{ - \} +$ $+ \{ - \} \{ - \} \{ - \} \{ - \} +$ $+ \{ - \} \{ - \} \{ - \} \{ - \} +$

મેમરી ટ્રીક

“ERfDFr - Enqueue at Rear, Dequeue from Front”

પ્રશ્ન 2(ક) OR [7 ગુણ]

A+B/C+D પદ ને પોસ્ટફિક્સ મા ફેરવો અને STACK નો ઉપયોગ કરીને A,B,C અને D ની કોઇ યક્ષમત ધારીને એનુભૂત શોધો.

જવાબ

“A+B/C+D” એકસપ્રેશનને પોસ્ટફિક્સમાં કન્વર્ટ કરીને સ્ટેકનો ઉપયોગ કરીને તેનું મૂલ્યાંકન કરવું:

સ્ટેપ 1: પોસ્ટફિક્સમાં કન્વર્ટ કરતું

Table 7: ઇનફિક્સથી પોસ્ટફિક્સ કન્વર્જન

સિમ્બોલ	સ્ટેક	આઉટપુટ	એકશન
A	A		આઉટપુટમાં ઉમેરો
+	+ A		સ્ટેકમાં પુશ કરો
B	+ A B		આઉટપુટમાં ઉમેરો
/	+ / A B		સ્ટેકમાં પુશ કરો (ઉચ્ચ પ્રિસિડન્સ)
C	+ / A B C		આઉટપુટમાં ઉમેરો
+	+ A B C /		ઉચ્ચ/સમાન પ્રિસિડન્સના બધા ઓપરેટર્સ પોપ કરો, + પુશ કરો
D	+ A B C / + D		આઉટપુટમાં ઉમેરો
End	A B C / + D +		બાકીના ઓપરેટર્સ પોપ કરો

ફાઇનલ પોસ્ટફિક્સ: A B C / + D +

સ્ટેપ 2: વેલ્યુઝ અનુભૂતિ કરતું

Table 8: પોસ્ટફિક્સ ઇવેલ્યુઅશન

સિમ્બોલ	સ્ટેક	કેલક્યુલેશન
5 (A)	5	વેલ્યુ પુશ કરો
10 (B)	5, 10	વેલ્યુ પુશ કરો
2 (C)	5, 10, 2	વેલ્યુ પુશ કરો
/	5, 5	$10/2 = 5$
+	10	$5+5 = 10$
3 (D)	10, 3	વેલ્યુ પુશ કરો
+	13	$10+3 = 13$

રિઝલ્ટ: 13

મેમરી ટ્રીક

“PC-SE - પુશ ઓપરેન્ટ્સ, કેલક્યુલેટ ઓપરેટર્સ પર, સ્ટેક બધું સ્ટોર કરે”

પ્રશ્ન 3(અ) [3 ગુણ]

લિંકડ લિસ્ટ ના ઉપયોગો લખો.

જવાબ

લિંકડ લિસ્ટ એ વર્સોટાઇલ ડેટા સ્ટ્રક્ચર છે જેના ઘણા વ્યવહારિક ઉપયોગો છે.

Table 9: લિંકડ લિસ્ટના ઉપયોગો

એપ્લિકેશન	શા માટે લિંકડ લિસ્ટ વપરાય છે
ડાયનેમિક મેમરી એલોકેશન	રિએલોકેશન વિના કાર્યક્ષમ ઇન્સર્શન/ડિલીશન
સ્ટેક અને ક્યુ ઇમ્પ્રિલમેન્ટેશન	જરૂરિયાત મુજબ વધી અને ઘટી શકે છે
અનદુ ફંક્શનાલિટી	હિસ્ટરીમાંથી ઓપરેશન્સ સરળતાથી ઉમેરી/કાઢી શકાય છે
હેશ ટેબલ્સ	ચેઇનિંગ દ્વારા કોલિઝન હેન્ડલિંગ માટે
મ્યુઝિક પ્લેલિસ્ટ	ગોતો વરચે સરળ નેવિગેશન (આગળ/પાછળ)

મેમરી ટ્રીક

“DSUHM - ડાયનેમિક એલોકેશન, સ્ટેક & ક્યુ, અનદુ, હેશ ટેબલ્સ, મ્યુઝિક પ્લેયર”

પ્રશ્ન 3(બ) [4 ગુણ]

પાયથનમા સિંગલી લિંકડ લિસ્ટ કેવી રીતે બનાવી શકાય એ સમજાવો.

જવાબ

પાયથનમા સિંગલી લિંકડ લિસ્ટ બનાવવા માટે નોડ કલાસ ડિફાઇન કરવી અને બેજિક ઓપરેશન્સ ઇમ્પલેમેન્ટ કરવા પડે છે.
કોડ ઉદાહરણ:

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

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        # ...
        if self.head is None:
            self.head = new_node
            return
        # ...
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node
```

ડાયાગ્રામ: લિંકડ લિસ્ટ બનાવવી

```
flowchart LR
    A[" "] --> B[""]
    A --> C[" "]
    C --> D[""]
```

મેમરી ટ્રીક

"CHEN - Create nodes, Head first, End attachment, Next pointers"

પ્રશ્ન 3(ક) [7 ગુણ]

સિંગલી લિંકડ લિસ્ટ ની શરૂઆતમાં અને અંતમાં નવા નોડ ઉમેરવાનો કોડ લખો.

જવાબ

સિંગલી લિંકડ લિસ્ટની શરૂઆત અને અંતમાં નોડ ઉમેરવા માટે અલગ અલગ અભિગમની જરૂર પડે છે.
કોડ ઉદાહરણ:

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

class LinkedList:
    def __init__(self):
        self.head = None

    # (prepend)
    def insert_at_beginning(self, data):
        new_node = Node(data)
```

```

        new\_node.next = self.head
        self.head = new\_node

    \#      (append)
def insert\_at\_end(self, data):
    new\_node = Node(data)
    \#
    if self.head is None:
        self.head = new\_node
        return

    \#
    current = self.head
    while current.next:
        current = current.next

    \#
    current.next = new\_node

```

ડાયાગ્રામ: ઇન્સર્શન ઓપરેશન્સ

Insert at Beginning:	Insert at End:
+{--}{-}{-}{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}{-}{-}+	+{-}{-}{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}{-}{-}+
New Node {--}{-}{-}{-}{-}{-} Head Head {--}{-}{-}{-}{-} ... {--}{-}{-}{-}{-} New Node	+{--}{-}{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}{-}{-}+

મેમરી ટ્રીક

``BEN - Beginning is Easy and Next-based, End Needs traversal''

પ્રશ્ન 3(અ) OR [3 ગુણ]

સિંગલી લિંકડ મા રહેલ નોડ ની સંખ્યા ગણવા માટેનો કોડ લખો.

જવાબ

નોડની સંખ્યા ગણવા માટે હેડથી ટેઇલ સુધી આખી લિંકડ લિસ્ટ ટ્રેવર્સ કરવી પડે છે.
કોડ ઉદાહરણ:

```

def count\_nodes(self):
    count = 0
    current = self.head

    \#
    while current:
        count += 1
        current = current.next

    return count

```

ડાયાગ્રામ: નોડ્સ ગણવા

```

flowchart LR
    A["count=0"] --> B[""]
    B --> C[""]
    C --> D["count"]
    D --> E[""]
    E --> C
    C --> F["count"]

```

પ્રશ્ન 3(બ) OR [4 ગુણ]

કોલમ એ અને કોલમ બી ના યોગ્ય વિકલ્પ જોડો.

જવાબ

વિવિધ લિંકડ લિસ્ટ પ્રકારો અને તેમના લક્ષણો વર્ચેનું મેર્ચિંગ:

Table 10: લિંકડ લિસ્ટ પ્રકારો અને લક્ષણોનું મેર્ચિંગ

કોલમ એ	કોલમ બી	મેચ
1. સિંગલી લિંકડ લિસ્ટ	c. નોડ્સમાં ડેટા અને આગામી નોડનો સંદર્ભ હોય છે	1-c
2. ડબલી લિંકડ લિસ્ટ	d. નોડ્સમાં આગામી અને પાછલા બંને નોડ્સનો ડેટા અને સંદર્ભો હોય છે	2-d
3. સર્કુલર લિંકડ લિસ્ટ	b. નોડ્સ એક લૂપ બનાવે જેમા છેલ્લો નોડ પ્રથમ નોડ તરફ નિર્દેશ કરે	3-b
4. લિંકડ લિસ્ટ નો એક નોડ	a. મૂળભૂત એકમ કે જેમા ડેટા અને સંદર્ભ હોય	4-a

ડાયાગ્રામ: વિવિધ લિંકડ લિસ્ટ પ્રકારો

```

:   A{-B{-}C{-}D{-}null}
:   A{-}B{-}C{-}D{-}null
:   A{-B{-}C{-}D{-}+}
      \^{
        |
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}+
  
```

મેમરી ટ્રીક

પ્રશ્ન 3(ક) OR [7 ગુણ]

સિંગલી લિંકડ લિસ્ટ મા પ્રથમ અને છેલ્લો નોડ ને કાઢી નાખવાનું સમજાવો.

જવાબ

સિંગલી લિંકડ લિસ્ટમાંથી નોડ કાઢવાની જટિલતા પોઝિશન (પ્રથમ વિ. છેલ્લો) પર આધારિત હોય છે.

Table 11: ડિલીશન કંપેરિઝન

પોઝિશન	અભિગમ	ટાઇમ કોમ્પ્લેક્સિટી	સ્પેશિયલ કેસ
પ્રથમ નોડ	હેડ પોઇન્ટર બદલો	O(1)	એક કરો કે લિસ્ટ ખાલી છે કે નહીં
છેલ્લો નોડ	બીજા છેલ્લા નોડ સુધી ટ્રેવર્સ કરો	O(n)	સિંગલ નોડ લિસ્ટ હેન્ડલ કરો

કોડ ઉદાહરણ:

```
def delete\_first(self):
    \#
    if self.head is None:
        return

    \#
    self.head = self.head.next

def delete\_last(self):
    \#
    if self.head is None:
        return

    \#
    if self.head.next is None:
        self.head = None
        return

    \#
    current = self.head
    while current.next.next:
        current = current.next

    \#
    current.next = None
```

ડાયગ્રામ: ડિલીશન ઓપરેશન્સ

Delete First:	Delete Last:
+{--}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+	+{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+
Head {--}{-}{-}{-} Next = Head {--}{-}{-}{-} Next {--}{-}{-}{-} Last =	Head {--}{-}{-}{-} Next {--}{-}{-}{-} Last =
+{--}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+	+{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+
+{--}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+	+{--}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+
Head {--}{-}{-}{-} Next {--}{-}{-}{-} X	Head {--}{-}{-}{-} Next {--}{-}{-}{-} X
+{--}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+	+{--}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+

મેમરી ટ્રીક

“FELO - First is Easy, Last needs One-before-last”

પ્રશ્ન 4(અ) [3 ગુણ]

ડબલી લિંકડ લિસ્ટ નો કો-સૈપ્ટ સમજાવો.

જવાબ

ડબલી લિંકડ લિસ્ટ એ બાયડાયરેક્શનલ લીનિયર ડેટા સ્ટ્રક્ચર છે જેમાં નોડ્સમાં ડેટા, અગાઉના, અને આગામના રેફરન્સ હોય છે.

ડાયગ્રામ: ડબલી લિંકડ લિસ્ટ

+{--}{-}{-}{-}{-}+ prev data next prev data next prev data next	+{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+ +{-}{-}{-}{-}{-}+
NULL{--}{-}{-}{-}{-}+ 10 {-}{-}{-}{-}{-} {-}{-}{-}{-}{-} 20 {-}{-}{-}{-}{-} {-}{-}{-}{-}{-} 30 {-}{-}{-}{-}{-}+	+{--}{-}{-}{-}{-}+ +{--}{-}{-}{-}{-}+ +{--}{-}{-}{-}{-}+ +{--}{-}{-}{-}{-}+ +{--}{-}{-}{-}{-}+

મેમરી ટ્રીક

“PDN - Previous, Data, Next”

પ્રશ્ન 4(બ) [4 ગુણ]

લિનિયર સર્ચ નો કોન્સૈપ્ટ સમજાવો.

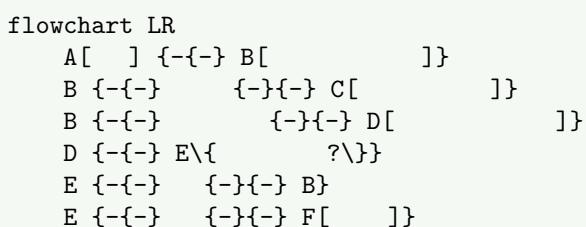
જવાબ

લિનિયર સર્ચ એ સરળ સિક્વેન્શિયલ સર્ચ અલ્ગોરિધમ છે જે ટાર્ગેટ શોધવા માટે એક પણી એક દરેક એલિમેન્ટ ચેક કરે છે.

Table 12: લિનિયર સર્ચ લક્ષણો

પાસું	વર્ણન
કાર્યપ્રણાલી	શરૂઆતથી અંત સુધી કુમશ: દરેક એલિમેન્ટ ચેક કરો
ટાઇમ કોમ્પ્લેક્સિટી	$O(n)$ - વર્સ્ટ અને એવરેજ કેસ
બેસ્ટ કેસ	$O(1)$ - પ્રથમ પોઝિશન પર એલિમેન્ટ મળો
અનુકૂળતા	નાના લિસ્ટ અથવા અનસોર્ટ ડેટા માટે
ફિયદો	સરળ ઇમ્પ્લિમેન્ટેશન, કોઈપણ કલેક્શન પર કામ કરે છે

ડાયાગ્રામ: લિનિયર સર્ચ પ્રોસેસ



મેમરી ટ્રીક

"SCENT - Search Consecutively Each element until Target"

પ્રશ્ન 4(ક) [7 ગુણ]

બાયનરી સર્ચ અલ્ગોરિધમ ઇમ્પ્લિમેટ કરવા માટેનો કોડ લખો.

જવાબ

બાયનરી સર્ચ એક કાર્યક્ષમ અલ્ગોરિધમ છે જે સર્ચ ઇન્ટરવલને વારંવાર અડધા ભાગમાં વિભાજિત કરીને સોર્ટેડ એરેમાં એલિમેન્ટ્સ શોધે છે.

કોડ ઉદાહરણ:

```

def binary\_search(arr, target):
    left = 0
    right = 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

```

ડાયાગ્રામ: બાયનરી સર્ચ પ્રોસેસ

Array: [10, 20, 30, 40, 50, 60, 70]

Search: 40

```

Step 1: mid = 3, arr[mid] = 40 (    !)
      left           right
      |               |
[10, 20, 30, 40, 50, 60, 70]
      \^{}_
          mid
  
```

મેમરી ટ્રીક

“MCLR - Middle Compare, Left or Right adjust”

પ્રશ્ન 4(અ) OR [3 ગુણ]

સિલેક્શન સોર્ટ અલ્ગોરિધમ નો કોન્સૈપ્ટ સમજાવો.

જવાબ

સિલેક્શન સોર્ટ એ સરળ કમ્પેરિઝન-બેડ સોર્ટિંગ અલ્ગોરિધમ છે જે એરેને સોર્ટ અને અનસોર્ટ રીજનમાં વિભાજિત કરે છે.

Table 13: સિલેક્શન સોર્ટ લક્ષણો

પાસું	વર્ણન
અભિગમ	અનસોર્ટ ભાગમાંથી મિનિમમ એલિમેન્ટ શોધો અને શરૂઆતમાં મૂકો
ટાઇમ કોમ્પલેક્સટી	$O(n^2)$ - , ,
સ્પેસ કોમ્પલેક્સટી	$O(1)$ - ઇન-પ્લેસ સોર્ટિંગ
સ્ટેબલ નથી (સમાન એલિમેન્ટ્સનો રિલેટિવ ઓર્ડર બદલાઈ શકે)	સ્ટેબલ નથી (સમાન એલિમેન્ટ્સનો રિલેટિવ ઓર્ડર બદલાઈ શકે)
ફાયદો	સરળ ઇમ્પલિમેન્ટેશન સાથે મિનિમલ મેમરી વપરાશ

મેમરી ટ્રીક

“FSMR - Find Smallest, Move to Right position, Repeat”

પ્રશ્ન 4(બ) OR [4 ગુણ]

બબલ સોર્ટ મેથડ સમજાવો.

જવાબ

બબલ સોર્ટ એ સરળ સોર્ટિંગ અલ્ગોરિધમ છે જે વારંવાર લિસ્ટમાં આગળ વધે છે, આસપાસના એલિમેન્ટ્સની તુલના કરે છે, અને જો તેઓ ખોટા ક્રમમાં હોય તો તેમને સ્વેપ કરે છે.

Table 14: બબલ સોર્ટ લક્ષણો

પાસું	વર્ણન
અભિગમ	આસપાસના એલિમેન્ટ્સની વારંવાર તુલના કરો અને જરૂર પડ્યે તો સ્વેપ કરો
પાસ	n એલિમેન્ટ્સ માટે ($n-1$) પાસ
ટાઇમ કોમ્પલેક્સટી	$O(n^2)$ - , $O(n)$ -
સ્પેસ કોમ્પલેક્સટી	$O(1)$ - ઇન-પ્લેસ સોર્ટિંગ
ઓપ્ટિમાઇઝેશન	જો કોઈ પાસમાં સ્વેપ ન થાય તો અલ્વા ટર્મિનેશન

ડાયાગ્રામ: બબલ સોર્ટ પ્રોસેસ

Array: [5, 3, 8, 4, 2]

Pass 1: [3, 5, 4, 2, 8]
 \^{\{-}\ \^{\{}} \^{\{-}\ \^{\{}} \^{\{-}\ \^{\{}}

Pass 2: [3, 4, 2, 5, 8]
 \^{\{-}\ \^{\{}} \^{\{-}\ \^{\{}}

Pass 3: [3, 2, 4, 5, 8]
 \^{\{-}\ \^{\{}}

Pass 4: [2, 3, 4, 5, 8] ()
 \^{\{-}\ \^{\{}}

મેમરી ટ્રીક

``CABS - Compare Adjacent, Bubble-up Swapping''

પ્રશ્ન 4(ક) OR [7 ગુણ]

ઉદાહરણ સાથે કવિક સોર્ટ મેથડનું વર્કિંગ સમજાવો.

જવાબ

કવિક સોર્ટ એ કાર્યક્ષમ ડિવાઇડ-અન્ડ-કોન્કર સોર્ટિંગ અલ્ગોરિધમ છે જે પિવોટ એલિમેન્ટ પસંદ કરીને અને એરેને પાર્ટિશન કરીને કામ કરે છે.

Table 15: કવિક સોર્ટ સ્ટેપ્સ

સ્ટેપ	વર્ણન
1	એરેમાંથી પિવોટ એલિમેન્ટ પસંદ કરો
2	પાર્ટિશન: એલિમેન્ટ્સને ફરીથી ગોઠવો (પિવોટથી નાના ડાબી બાજુ, મોટા જમણી બાજુ)
3	પિવોટની ડાબી અને જમણી બાજુના સબઅરે પર રિકર્સિવલી કવિક સોર્ટ લાગુ કરો

એરે [7, 2, 1, 6, 8, 5, 3, 4] સાથે ઉદાહરણ:

Pivot: 4

After partition: [2, 1, 3] 4 [7, 6, 8, 5]
 Left P Right

Recursively sort left: [1] 2 [3] \rightarrow [1, 2, 3]

Recursively sort right: [5] 7 [6, 8] \rightarrow [5, 6, 7, 8]

Final sorted array: [1, 2, 3, 4, 5, 6, 7, 8]

ડાયાગ્રામ: કવિક સોર્ટ પાર્ટિશનિંગ

```
flowchart LR
    A[ : 7,2,1,6,8,5,3,4] {-->} B[ : 4]
    B {-->} C[ ]
    C {-->} D[ : 2,1,3]
    C {-->} E[ : 4]
    C {-->} F[ : 7,6,8,5]
    D {-->} G[ ]
    F {-->} H[ ]
    G {-->} I[ : 1,2,3]
    H {-->} J[ : 5,6,7,8]
    I {-->} K[ : 1,2,3,4,5,6,7,8]
    E {-->} K
    J {-->} K
```

મેમરી ટ્રીક

"PPR - Pivot, Partition, Recursive divide"

પ્રશ્ન 5(અ) [3 ગુણ]

બાયનરી ટ્રી સમજાવો.

જવાબ

બાયનરી ટ્રી એ એક હાયરાર્કિલ ડેટા સ્ટ્રક્ચર છે જેમાં દરેક નોડને વધુમાં વધુ બે ચિલ્ડન હોય છે જેને લેફ્ટ અને રાઇટ ચાઇલ્ડ તરીકે ઓળખવામાં આવે છે.

ડાયાગ્રામ: બાયનરી ટ્રી

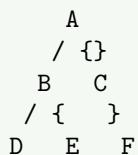


Table 16: બાયનરી ટ્રી પ્રોપર્ટીઝ

પ્રોપર્ટી	વર્ણન
નોડ	ડેટા અને લેફ્ટ અને રાઇટ ચિલ્ડના રેફરન્સ ધરાવે છે
ડિષ્ટ	રૂટથી નોડ સુધીના પાથની લંબાઈ
હાઇટ	નોડથી લીફ સુધીના સૌથી લાંબા પાથની લંબાઈ
બાયનરી ટ્રી	દરેક નોડને વધુમાં વધુ 2 ચિલ્ડન હોય છે

મેમરી ટ્રીક

"RLTM - Root, Left, Two, Maximum"

પ્રશ્ન 5(બ) [4 ગુણ]

ટ્રી ના સંદર્ભમ મા રૂટ, પાથ, પેરેન્ટ અને ચિલ્ડન પદો વ્યાખ્યાયિત કરો.

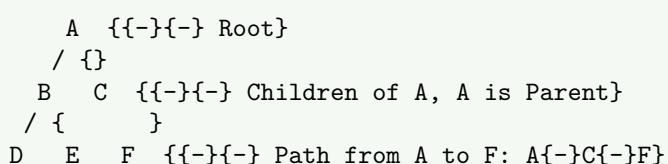
જવાબ

ટ્રીમાં હાયરાર્કીમાં નોડ્સ વચ્ચેના સંબંધોને વર્ણવવા માટે ચોક્કસ શબ્દાવલી છે.

Table 17: ટ્રી શબ્દાવલી

પદ	વ્યાખ્યા
રૂટ	ટ્રીનો સૌથી ઉપરનો નોડ જેને કોઈ પેરેન્ટ નથી હોતા
પાથ	એક નોડથી બીજા નોડ સુધી એજ દ્વારા જોડાયેલા નોડ્સનો સિક્વન્સ
પેરેન્ટ	એક અથવા વધુ ચાઇલ્ડ નોડ્સ ધરાવતો નોડ
ચિલ્ડન	પેરેન્ટ નોડથી સીધા જોડાયેલા નોડ્સ

ડાયાગ્રામ: ટ્રી શબ્દાવલી



મેમરી ટ્રીક

"RPPC - Root at Top, Path connects, Parent above, Children below"

પ્રશ્ન 5(ક) [7 ગુણ]

નીચે આપેલા ટ્રી માટે પ્રીઓર્ડર અને પોસ્ટઓર્ડર ટ્રાવર્સલ લાગુ કરો.

જવાબ

પ્રીઓર્ડર અને પોસ્ટઓર્ડર એ ડેપથ-ફર્સ્ટ ટ્રી ટ્રાવર્સલ મેથ્ડ્સ છે જેમાં અલગ અલગ નોડ વિઝિટિંગ સિક્વન્સ હોય છે.
આપેલ ટ્રી:

```

        40
       /   {}
      30   50
     / {   / }
    25 35  45 60
   / {       / }
  15 28     55 70
  
```

Table 18: ટ્રી ટ્રાવર્સલ કંપેરિઝન

ટ્રાવર્સલ	ઓર્ડર	આપેલ ટ્રી માટે રિઝલ્ટ
પ્રીઓર્ડર	રૂટ, લેફ્ટ, રાઇટ	40, 30, 25, 15, 28, 35, 50, 45, 60, 55, 70
પોસ્ટઓર્ડર	લેફ્ટ, રાઇટ, રૂટ	15, 28, 25, 35, 30, 45, 55, 70, 60, 50, 40

કોડ ઉદાહરણાઃ

```

def preorder(root):
    if root:
        print(root.data, end=" ", "#")
        preorder(root.left)      "#"
        preorder(root.right)     "#"

def postorder(root):
    if root:
        postorder(root.left)    "#"
        postorder(root.right)   "#"
        print(root.data, end=" ", "#")
  
```

મેમરી ટ્રીક

“PRE-NLR, POST-LRN - પ્રીઓર્ડર (Node-Left-Right), પોસ્ટઓર્ડર (Left-Right-Node)”

પ્રશ્ન 5(અ) OR [3 ગુણ]

બાયનરી ટ્રી ની એપ્લિકેશન્સ લખો.

જવાબ

બાયનરી ટ્રીના કમ્પ્યુટર સાયન્સના વિવિધ ક્ષેત્રોમાં અનેક વ્યવહારિક ઉપયોગો છે.

Table 19: બાયનરી ટ્રી એપ્લિકેશન્સ

એપ્લિકેશન	વર્ણન
બાયનરી સર્ટ્ટ ટ્રી	કાર્યક્રમ સર્વિંગ, ઇન્સર્શન, અને ડિલીશન ઓપરેશન્સ
એક્સપ્રેશન ટ્રી	ઇવેલ્યુઅ૱શન માટે મેથેમેટિકલ એક્સપ્રેશન્સ રજૂ કરવા
હફ્મેન કોડિંગ	ડેટા કમ્પ્રેશન અલ્ગોરિધમ્સ
પ્રાયોરિટી ક્યુ	હીપ ડેટા સ્ટ્રક્ચરનું ઇમ્પ્લિમેન્ટેશન
ડિસિઝન ટ્રી	મશીન લર્નિંગમાં કલાસિફિકેશન અલ્ગોરિધમ્સ

મેમરી ટ્રીક

“BEHPD - BST, Expression, Huffman, Priority queue, Decision tree”

પ્રશ્ન 5(બ) OR [4 ગુણ]

બાયનરી સર્ચ ટ્રી માં નોડ કેવી રીતે ઉમેરી શકાય તે સમજાવો.

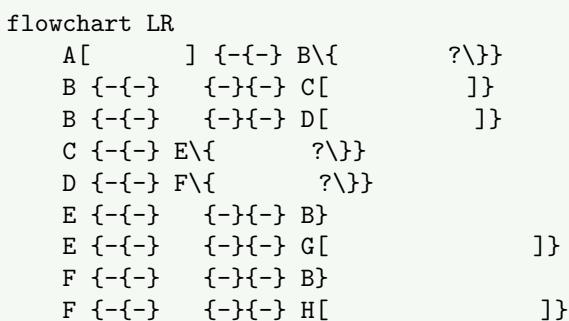
જવાબ

બાયનરી સર્ચ ટ્રી (BST)માં ઇન્સર્શન BST પ્રોપર્ટી અનુસરે છે: લેફ્ટ ચાઇલ્ડ < પેરેન્ટ < રાઇટ ચાઇલ્ડ.

Table 20: BST માં ઇન્સર્શન સ્ટેપ્સ

સ્ટેપ	વર્ણન
1	રૂટ નોડથી શરૂ કરો
2	જો નવી વેલ્યુ < કરેટ નોડ વેલ્યુ, તો લેફ્ટ સબટ્રીમાં જાઓ
3	જો નવી વેલ્યુ > કરેટ નોડ વેલ્યુ, તો રાઇટ સબટ્રીમાં જાઓ
4	ખાલી પોઝિશન (નલ પોઇન્ટર) મળે ત્યાં સુધી રિપીટ કરો
5	મળેલી ખાલી પોઝિશન પર નવો નોડ ઇન્સર્ટ કરો

ડાયાગ્રામ: BST ઇન્સર્શન



મેમરી ટ્રીક

“LSRG - Less-go-left, Same-or-greater-go-right”

પ્રશ્ન 5(ક) OR [7 ગુણ]

8, 4, 12, 2, 6, 10, 14, 1, 3, 5 નમ્બર માટે બાયનરી સર્ચ ટ્રી દોરો અને ટ્રી માટે ઇન ઓર્ડર ટ્રાવર્સલ લખો.

જવાબ

બાયનરી સર્ચ ટ્રી (BST) BST પ્રોપર્ટી જાળવીને નોડ્સ ઇન્સર્ટ કરીને બનાવવામાં આવે છે.
આપેલ એલિમેન્ટ્સ માટે બાયનરી સર્ચ ટ્રી:

```

8
/
4   12
/ { / }
2   6 10 14
/ { / }
1   3 5
  
```

Table 21: BST કન્સ્ટ્રક્શન પ્રોસેસ

સ્ટેપ	ઇન્સર્ટ	ટ્રી સ્ટ્રક્ચર
1	8	૩૨ = 8
2	4	8 ની ડાબી બાજુ
3	12	8 ની જમણી બાજુ
4	2	4 ની ડાબી બાજુ
5	6	4 ની જમણી બાજુ
6	10	12 ની ડાબી બાજુ
7	14	12 ની જમણી બાજુ
8	1	2 ની ડાબી બાજુ

9	3	2 ની જમણી બાજુ
10	5	6 ની ડાબી બાજુ

ઇન-ઓર્ડર ટ્રાવર્સલ:

ઇન-ઓર્ડર ટ્રાવર્સલ નોડ્સને આ કમમાં વિઝિટ કરે છે: લેફ્ટ સબટ્રી, કર્ટ નોડ, રાઇટ સબટ્રી.

આપેલ BST માટે, ઇન-ઓર્ડર ટ્રાવર્સલ આ છે: 1, 2, 3, 4, 5, 6, 8, 10, 12, 14

કોડ ઉદાહરણ:

```
def inorder\_traversal(root):  
    if root:  
        inorder\_traversal(root.left)      #  
        print(root.data, end=", ")      #  
        inorder\_traversal(root.right)    #
```

મેમરી ટ્રીક

“LNR - Left, Node, Right makes sorted order in BST”