ડેટા સ્ટ્રક્ચર અને એપ્લિકેશન (1333203) - વિન્ટર 2023 સોલ્યુશન

Milav Dabgar

જાન્યુઆરી 18, 2024

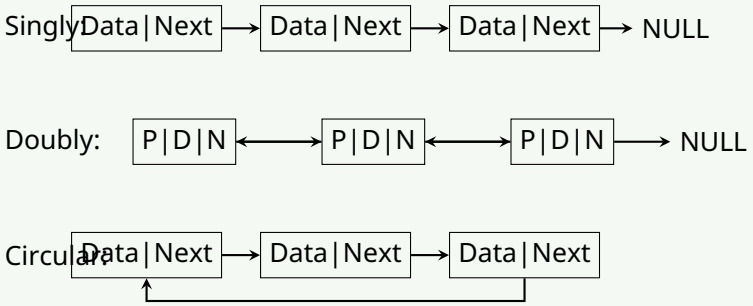
પ્રશ્ન 1(a) [3 ગુણ]

લીન્કડ લીસ્ટની વ્યાખ્યા આપો. વિવિધ પ્રકારના લિન્કડ લીસ્ટ ની યાદી આપો.

જવાબ

કોષ્ટક 1. લિન્કડ લિસ્ટ વ્યાખ્યા અને પ્રકાર

વ્યાખ્યા	લિન્કડ લિસ્ટના પ્રકાર
લિન્કડ લિસ્ટ એ લીનિયર ડેટા સ્ટ્રક્ચર છે જેમાં એલિમેન્ટ્સ નોડ્સમાં સ્ટોર થાય છે, અને દરેક નોડ ક્રમમાં આગળના નોડને પોઇન્ટ કરે છે	1. સિંગલી લિન્કડ લિસ્ટ 2. ડબલી લિન્કડ લિસ્ટ 3. સર્ક્યુલર લિન્કડ લિસ્ટ 4. સર્ક્યુલર ડબલી લિન્કડ લિસ્ટ



આકૃતિ 1. લિન્કડ લિસ્ટના પ્રકારો

મેમરી ટ્રીક

``એક, બે, ગોળ, બે-ગોળ''

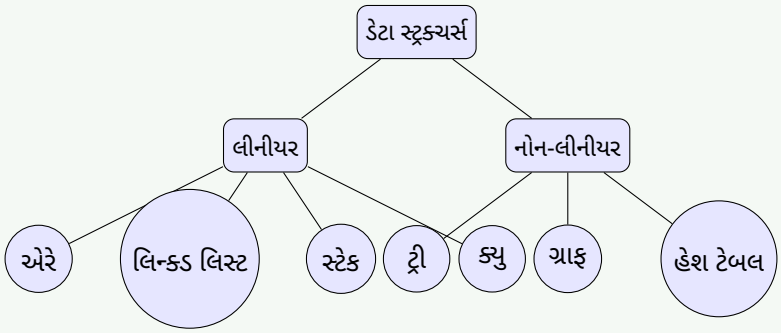
પ્રશ્ન 1(b) [4 ગુણ]

પાયથનમાં લીનીયર અને નોન-લીનીયર ડેટા સ્ટર્ચર ઉદાહરણ સાથે સમજાવો.

જવાબ

કોષ્ટક 2. લીનીયર વિરુદ્ધ નોન-લીનીયર ડેટા સ્ટ્રક્ચર

ડેટા સ્ટ્રક્ચર	વર્ણન	પાયથન ઉદાહરણો
લીનીયર	એલિમેન્ટ્સ ક્રમિક રીતે ગોઠવાયેલા હોય છે જેમાં દરેક એલિમેન્ટને એકદમ એક અગાઉનું અને એક પછીનું એલિમેન્ટ હોય છે (પ્રથમ અને છેલ્લા સિવાય)	Lists: [1, 2, 3] Tuples: (1, 2, 3) Strings: "abc" Queue: queue.Queue()
નોન-લીનીયર	એલિમેન્ટ્સ ક્રમિક રીતે ગોઠવાયેલા નથી; એક એલિમેન્ટ અનેક એલિમેન્ટ્સ સાથે જોડાઈ શકે છે	Dictionary: {"a": 1, "b": 2} Set: {1, 2, 3} Tree: કસ્ટમ ઇમ્પ્લીમેન્ટેશન Graph: કસ્ટમ ઇમ્પ્લીમેન્ટેશન



આકૃતિ 2. ડેટા સ્ટ્રક્ચર્સનું વર્ગીકરણ

મેમરી ટ્રીક

``લીનીયર લાઈનમાં, નોન-લીનીયર ચારે બાજુ''

પ્રશ્ન 1(c) [7 ગુણ]

પાયથનમાં ક્લાસ, એટ્રીબ્યુટ, ઓબ્જેક્ટ અને ક્લાસ મેથડ યોગ્ય ઉદાહરણ સાથે સમજાવો.

જવાબ

Student
- roll_no- name
+ __init__()+ display()

આકૃતિ 3. ક્લાસ ડાયાગ્રામ ઉદાહરણ

કોષ્ટક 3. OOP સંકલ્પનાઓ

શબ્દ	વર્ણન
ક્લાસ	ઓબ્જેક્ટ્સ બનાવવા માટેનો બ્લૂપ્રિન્ટ, જેમાં શેર્ડ એટ્રિબ્યુટ્સ અને મેથડ્સ હોય છે
એટ્રિબ્યુટ્સ	ક્લાસની અંદર ડેટા સ્ટોર કરતા વેરિએબલ્સ
ઓબ્જેક્ટ	ક્લાસનું ઇન્સ્ટન્સ, જેમાં ચોક્કસ એટ્રિબ્યુટ વેલ્યુ હોય છે
ક્લાસ મેથડ	ક્લાસની અંદર ડિફાઇન થયેલા ફંક્શન્સ જે ક્લાસની સ્થિતિને એક્સેસ અને મોડિફાય કરી શકે છે

```
class Student:
    # ક્લાસ એટ્રબ્યુટ
    school = "GTU"
```

```
    # કન્સ્ટ્રક્ટર
    def __init__(self, roll_no, name):
        # ઇન્સ્ટન્સ એટ્રિબ્યુટ્સ
        self.roll_no = roll_no
        self.name = name

    # ઇન્સ્ટન્સ મેથડ
    def display(self):
        print(f"Roll No: {self.roll_no}, Name: {self.name}")

    # ક્લાસ મેથડ
    @classmethod
    def change_school(cls, new_school):
        cls.school = new_school

# ઓબ્જેક્ટ બનાવવું
student1 = Student(101, "રાજ")
student1.display()  # આઉટપુટ: Roll No: 101, Name: રાજ
```

મેમરી ટ્રીક

``ક્લાસ બનાવે, એટ્રિબ્યુટ સંગ્રહે, ઓબ્જેક્ટ વાપરે, મેથડ ક્રિયા કરે''

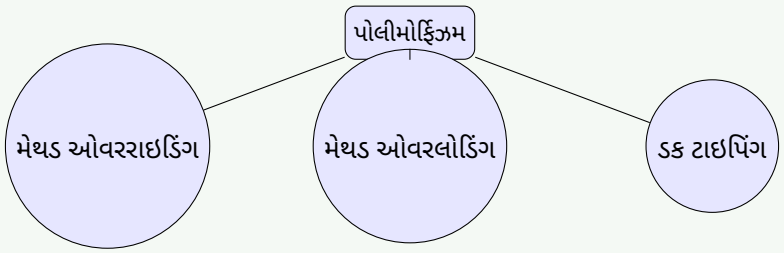
પ્રશ્ન 1(c) OR [7 ગુણ]

ડેટા એન્કેપ્સુલેસન અને પોલી મોર્ફીસમની વ્યાખ્યા આપો. પોલી મોર્ફીસમ સમજાવવા માટેનો પાયથન કોડ વિકસાવો.

જવાબ

કોષ્ટક 4. વ્યાખ્યાઓ

કોન્સેપ્ટ	વ્યાખ્યા
ડેટા એન્કેપ્સુલેસન	ડેટા અને મેથડ્સને એક એકમ (ક્લાસ)માં બંધ કરવા અને કેટલાક કોમ્પોનન્ટ્સને સીધી એક્સેસથી પ્રતિબંધિત કરવા
પોલીમોર્ફિઝમ	વિવિધ ક્લાસને એક જ નામના મેથડનો પોતાનો અમલ પૂરો પાડવાની ક્ષમતા



આકૃતિ 4. પોલીમોર્ફિઝમના પ્રકાર

```
# પોલીમોરફિઝમ ઉદાહરણ
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "ભૌ ભૌ!"

```

```python
class Cat(Animal):
    def speak(self):
        return "મ્યાઉ!"

class Duck(Animal):
    def speak(self):
        return "ક્વેક!"

# પોલીમોર્ફિઝમ દર્શાવતું ફંક્શન
def animal_sound(animal):
    return animal.speak()

# ઓબ્જેક્ટ્સ બનાવવા
dog = Dog()
cat = Cat()
duck = Duck()

# એક જ ફંક્શન વિવિધ પ્રાણી ઓબ્જેક્ટ્સ માટે કામ કરે છે
print(animal_sound(dog))  # આઉટપુટ: ભૌ ભૌ!
print(animal_sound(cat))  # આઉટપુટ: મ્યાઉ!
print(animal_sound(duck))  # આઉટપુટ: ક્વેક!
```

મેમરી ટ્રીક

``એન્કેપ્સુલેશન છુપાવે છે, પોલીમોર્ફિઝમ બદલાય છે''

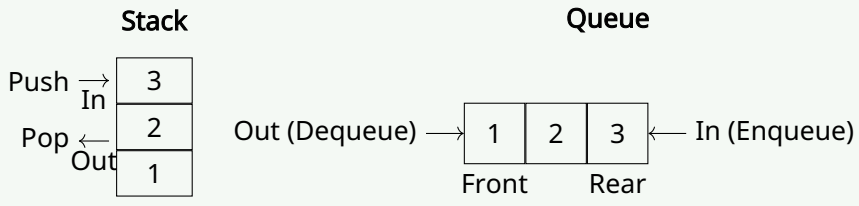
પ્રશ્ન 2(a) [3 ગુણ]

સ્ટેક અને ક્યુ નો તફાવત આપો.

જવાબ

કોષ્ટક 5. સ્ટેક વિરુદ્ધ ક્યુ

ફીચર	સ્ટેક	ક્યુ
સિદ્ધાંત	LIFO (છેલ્લું આવે પહેલું જાય)	FIFO (પહેલું આવે પહેલું જાય)
ઓપરેશન	પુશ, પોપ	એનક્યુ, ડિક્યુ
એક્સેસ	એલિમેન્ટ્સ ફક્ત એક છેડેથી ઉમેરાય/દૂર થાય છે (ટોપ)	એલિમેન્ટ્સ છેલ્લે ઉમેરાય છે અને આગળથી દૂર થાય છે



આકૃતિ 5. સ્ટેક અને ક્યુ

મેમરી ટ્રીક

``સ્ટેક ઉપરનું પહેલા, ક્યુ આગળનું પહેલા''

પ્રશ્ન 2(b) [4 ગુણ]

પુશ અને પોપ ઓપરેશન માટેનો અલ્ગોરીધમ લખો.

જવાબ

PUSH અલ્ગોરિધમ:

```
શરુઆત

  1. ચેક કરો કે સ્ટેક ભરેલો છે કે નહીં
  2. જો ભરેલો ન હોય, તો top ને 1 વધારો
  3. 'top' પોઝિશન પર એલમેન્ટ ઉમેરોસમાપ્ત
```

POP અલ્ગોરિધમ:

```
શરુઆત

  1. ચેક કરો કે સ્ટેક ખાલી છે કે નહીં
  2. જો ખાલી ન હોય, તો 'top' પરના એલમેન્ટને લો
  3. top ને 1 ઘટાડો
  4. મેળવેલ એલમેન્ટ પાછો આપોસમાપ્ત
```

```python
class Stack:
    def __init__(self, size):
        self.stack = []
        self.size = size
        self.top = -1

    def push(self, element):
        if self.top >= self.size - 1:
            return "Stack Overflow"
        else:
            self.top += 1
            self.stack.append(element)
            return "Pushed " + str(element)

    def pop(self):
        if self.top < 0:
            return "Stack Underflow"
        else:
            element = self.stack.pop()
            self.top -= 1
            return element
```

મેમરી ટ્રીક

``ટોપ પર પુશ, ટોપથી પોપ''

પ્રશ્ન 2(c) [7 ગુણ]

નીચે. આપેલ સમીકરણ ને ઇન્ફીક્સ માંથી પોસ્ટફિક્ષ માં બદલો.
A * (B + C) - D / (E + F)

જવાબ

Infix	$A * (B + C) - D/(E + F)$
Postfix	$ABC + *DEF + /-$

કોષ્ટક 6. ઇન્ફીક્સ માંથી પોસ્ટફિક્ષ ટ્રેસ

સ્ટેપ	સિમ્બોલ	સ્ટેક	આઉટપુટ
1	A		A
2	*	*	A
3	(	* (	A
4	B	* (	A B
5	+	* (+	A B
6	C	* (+	A B C
7	)	*	A B C +
8	-	-	A B C + *
9	D	-	A B C + * D
10	/	- /	A B C + * D
11	(	- / (	A B C + * D
12	E	- / (	A B C + * D E
13	+	- / (+	A B C + * D E
14	F	- / (+	A B C + * D E F
15	)	- /	A B C + * D E F +
16	end		A B C + * D E F + / -

જવાબ: A B C + * D E F + / -

મેમરી ટ્રીક

``ઓપરેટર સ્ટેક પર, ઓપરન્ડ સીધા પ્રિન્ટ''

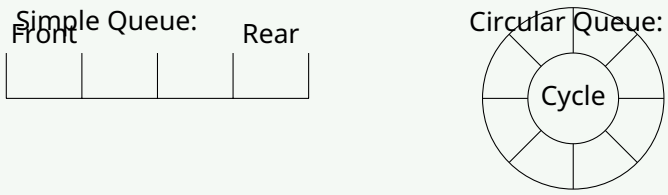
પ્રશ્ન 2(a) OR [3 ગુણ]

સિમ્પલ ક્યુ અને સર્ક્યુલર ક્યુ નો તફાવત આપો.

જવાબ

કોષ્ટક 7. સિમ્પલ વિરુદ્ધ સર્ક્યુલર ક્યુ

ફીચર	સિમ્પલ ક્યુ	સર્ક્યુલર ક્યુ
સ્ટ્રક્ચર	લીનિયર ડેટા સ્ટ્રક્ચર	જોડાયેલા છેડાવાળો લીનિયર ડેટા સ્ટ્રક્ચર
મેમરી	ડિક્યુ પછી ખાલી જગ્યાઓને કારણે અકાર્યક્ષમ મેમરી વપરાશ	ખાલી જગ્યાઓનો ફરીથી ઉપયોગ કરીને કાર્યક્ષમ મેમરી વપરાશ
ઇમ્પ્લિમે-ન્ટેશન	ફ્રન્ટ હંમેશા ઇન્ડેક્સ 0 પર, રીયર વધે	ફ્રન્ટ અને રીયર મોડ્યુલો ઓપરેશન સાથે સર્ક્યુલર રીતે ફરે



આકૃતિ 6. સિમ્પલ વિરુદ્ધ સર્ક્યુલર ક્યુ સ્ટ્રક્ચર

મેમરી ટ્રીક

``સાદી વેડફે, ગોળ ફરીથી વાપરે''

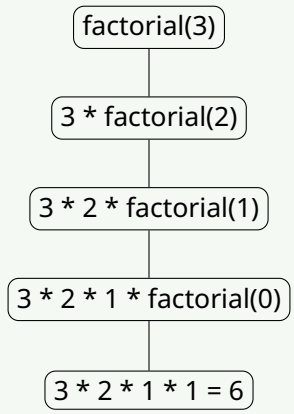
પ્રશ્ન 2(b) OR [4 ગુણ]

રીકસીવ ફંક્શનનો કોન્સેપ્ટ યોગ્ય ઉદાહરણ સાથે સમજાવો.

જવાબ

કોષ્ટક 8. રિકર્ઝન કોન્સેપ્ટ

મુખ્ય પાસાઓ	વર્ણન
વ્યાખ્યા	એવું ફંક્શન જે એક જ સમસ્યાના નાના ભાગને હલ કરવા માટે પોતાને જ કોલ કરે છે
બેઝ કેસ	એવી સ્થિતિ જ્યાં ફંક્શન પોતાને કોલ કરવાનું બંધ કરે છે
રિકર્સિવ કેસ	એવી સ્થિતિ જ્યાં ફંક્શન સમસ્યાના સરળ સ્વરૂપ સાથે પોતાને કોલ કરે છે



આકૃતિ 7. રિકર્સિવ કોલ્સ ટ્રેસ

```python
def factorial(n):
    # બેઝ કેસ
    if n == 0:
        return 1
    # રકિર્સવિ કેસ
    else:
        return n * factorial(n-1)

# ઉદાહરણ
result = factorial(5)  # 5! = 120
```

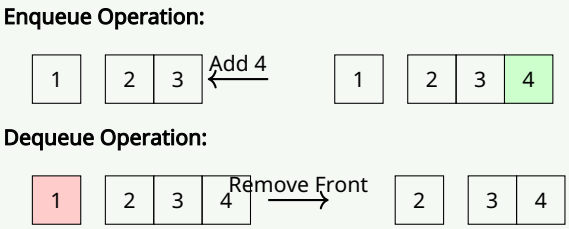
મેમરી ટ્રીક

``બેઝ તોડે, રિકર્શન પાછું આપે''

પ્રશ્ન 2(c) OR [7 ગુણ]

Enqueue અને Dequeue ઓપરેશન માટેનો પાયથન કોડ વિકસાવો.

જવાબ



આકૃતિ 8. Enqueue અને Dequeue વિઝ્યુઅલાઈઝેશન

```python
class Queue:
    def __init__(self, size):
        self.queue = []
        self.size = size
        self.front = 0
        self.rear = -1
        self.count = 0

    def enqueue(self, item):
        if self.count >= self.size:
            return "ક્યુ ભરેલી છે"
        else:
            self.rear += 1
            self.queue.append(item)
            self.count += 1
            return "Enqueued " + str(item)

    def dequeue(self):
        if self.count <= 0:
            return "ક્યુ ખાલી છે"
        else:
            item = self.queue.pop(0)
            self.count -= 1
            return item

    def display(self):
        return self.queue

# ટેસ્ટ
q = Queue(5)
q.enqueue(10)
q.enqueue(20)
q.enqueue(30)
print(q.display())  # [10, 20, 30]
print(q.dequeue())  # 10
print(q.display())  # [20, 30]
```

મેમરી ટ્રીક

``છેડે ઉમેરો, શરૂઆતથી કાઢો''

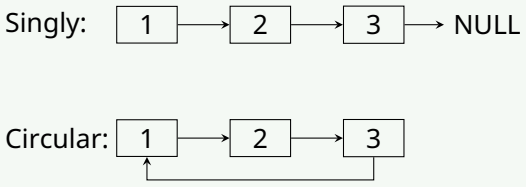
પ્રશ્ન 3(a) [3 ગુણ]

સીન્ગલી લિન્ક્ડ લીસ્ટ અને સર્ક્યુલર લિન્ક્ડ લીસ્ટ નો તફાવત આપો.

જવાબ

કોષ્ટક 9. સિંગલી વિરુદ્ધ સર્ક્યુલર લિન્ક્ડ લિસ્ટ

ફીચર	સિંગલી લિન્ક્ડ લિસ્ટ	સર્ક્યુલર લિન્ક્ડ લિસ્ટ
છેલ્લો નોડ	NULL તરફ પોઇન્ટ કરે છે	પહેલા નોડ તરફ પાછો પોઇન્ટ કરે છે
ટ્રાવર્સલ	ચોક્કસ અંત ધરાવે છે	સતત ટ્રાવર્સ કરી શકાય છે
મેમરી	દરેક નોડને એક પોઇન્ટર જોઈએ	દરેક નોડને એક પોઇન્ટર જોઈએ



આકૃતિ 9. સિંગલી વિરુદ્ધ સર્ક્યુલર સ્ટ્રક્ચર

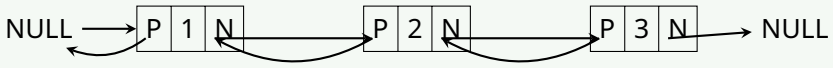
મેમરી ટ્રીક

``સિંગલી અટકે, સર્ક્યુલર ફરે''

પ્રશ્ન 3(b) [4 ગુણ]

ડબલી લિન્ક્ડ લીસ્ટ નો કોન્સેપ્ટ સમજાવો.

જવાબ



આકૃતિ 10. ડબલી લિન્ક્ડ લિસ્ટ સ્ટ્રક્ચર

કોષ્ટક 10. ડબલી લિન્ક્ડ લિસ્ટ ફીચર્સ

ફીચર	વર્ણન
નોડ સ્ટ્રક્ચર	દરેક નોડમાં ડેટા અને બે પોઇન્ટર્સ (previous અને next) હોય છે
નેવિગેશન	આગળ અને પાછળ એમ બંને દિશામાં ટ્રાવર્સ કરી શકાય છે
ઓપરેશન્સ	બંને છેડેથી ઇન્સર્શન અને ડિલીશન કરી શકાય છે
મેમરી વપરાશ	વધારાના પોઇન્ટરને કારણે સિંગલી લિન્ક્ડ લિસ્ટ કરતા વધુ મેમરી જોઈએ

```
class Node:
    def __init__(self, data):
        self.data = data
```

```
    self.prev = None
    self.next = None
```

મેમરી ટ્રીક

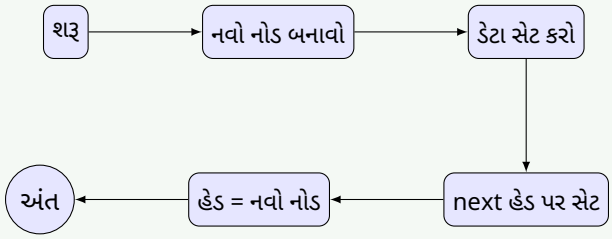
``બે પોઇન્ટર, બે દિશા''

પ્રશ્ન 3(c) [7 ગુણ]

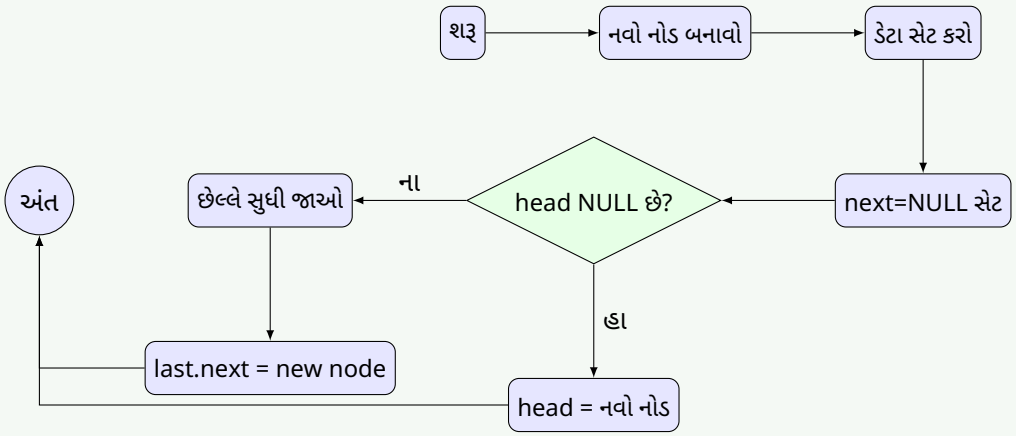
નીચે આપેલ ઓપરેશન માટે અલગોરિધમ લખો:
૧. લીસ્ટ ની શરૂઆતમાં નોડ દાખલ કરવા
૨. લીસ્ટ ના અંતમાં નોડ દાખલ કરવા

જવાબ

શરૂઆતમાં ઇન્સર્ટ:



અંતે ઇન્સર્ટ:




```
def insert_at_beginning(head, data):
    new_node = Node(data)
    new_node.next = head
    return new_node  # નવો head

def insert_at_end(head, data):
    new_node = Node(data)
    new_node.next = None

    # જો લિન્ક્ડ લિસ્ટ ખાલી હોય
    if head is None:
        return new_node

```

```
    # છેલ્લા નોડ સુધી ટ્રાવર્સ કરો
    temp = head
    while temp.next:
        temp = temp.next

    # છેલ્લા નોડને નવા નોડ સાથે જોડો
    temp.next = new_node
    return head
```

મેમરી ટ્રીક

``શરૂઆત: નવો જૂનાને આગળ કરે, અંત: જૂનો નવાને આગળ કરે''

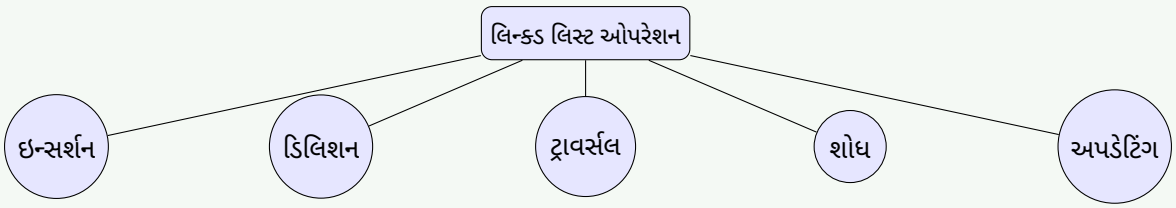
પ્રશ્ન 3(a) OR [3 ગુણ]

સીન્ગલી લિન્ક્ડ લીસ્ટ પરના વિવિધ ઓપરેશન ની યાદી આપો.

જવાબ

કોષ્ટક 11. સિંગલી લિન્ક્ડ લિસ્ટ પરના ઓપરેશન

સિંગલી લિન્ક્ડ લિસ્ટ પરના ઓપરેશન
1. ઇન્સર્શન (શરૂઆતમાં, મધ્યમાં, અંતે)
2. ડિલીશન (શરૂઆતથી, મધ્યમાંથી, અંતથી)
3. ટ્રાવર્સલ (દરેક નોડની મુલાકાત)
4. શોધ (ચોક્કસ નોડ શોધવો)
5. અપડેટિંગ (નોડ ડેટા બદલવો)



આકૃતિ 11. લિન્ક્ડ લિસ્ટ ઓપરેશન

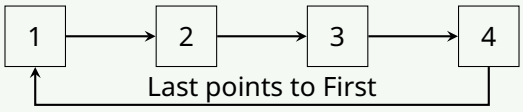
મેમરી ટ્રીક

``ઉમેરો કાઢો ફરો શોધો બદલો''

પ્રશ્ન 3(b) OR [4 ગુણ]

સર્ક્યુલર લિન્ક્ડ લીસ્ટ નો કોન્સેપ્ટ સમજાવો.

જવાબ



આકૃતિ 12. સર્ક્યુલર લિન્ક્ડ લિસ્ટ

કોષ્ટક 12. સર્ક્યુલર લિન્ક્ડ લિસ્ટ ફીચર્સ

ફીચર	વર્ણન
સ્ટ્રક્ચર	છેલ્લો નોડ NULL ને બદલે પહેલા નોડને પોઇન્ટ કરે છે
ફાયદો	બધા નોડમાં સતત ટ્રાવર્સલની અનુમતિ આપે છે
એપ્લિકેશન	રાઉન્ડ રોબિન શેડ્યુલિંગ, સર્ક્યુલર બફર ઇમ્પ્લિમેન્ટેશન
ઓપરેશન	છેલ્લા નોડ માટે ખાસ હેન્ડલિંગ સાથે સિંગલી લિન્ક્ડ લિસ્ટ જેવા ઇન્સર્શન અને ડિલીશન

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

# 3 નોડવાળી સર્ક્યુલર લિન્ક્ડ લિસ્ટ બનાવવી
head = Node(1)
node2 = Node(2)
node3 = Node(3)

head.next = node2
node2.next = node3
node3.next = head  # તેને સર્ક્યુલર બનાવે છે
```

મેમરી ટ્રીક

``છેલ્લો પહેલાને જોડે''

પ્રશ્ન 3(c) OR [7 ગુણ]

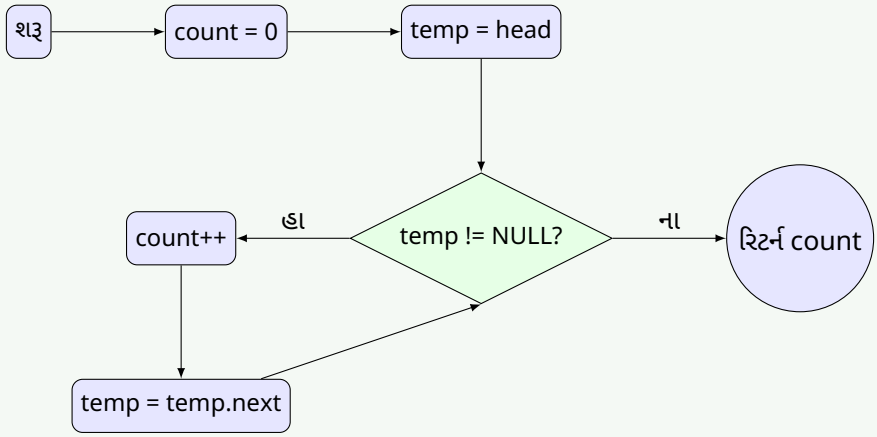
લિન્ક્ડ લીસ્ટની એપ્લીકેશનોની યાદી આપો. સીન્ગલી લિન્ક્ડ લીસ્ટમાં કુલ નોડ ગણવા માટેનો અલગોરિધમ લખો.

જવાબ

કોષ્ટક 13. એપ્લિકેશન

લિન્ક્ડ લિસ્ટની એપ્લિકેશન
1. સ્ટેક અને ક્યુનો અમલીકરણ
2. ડાયનેમિક મેમરી એલોકેશન
3. એપ્લિકેશનમાં અન્ડુ ફંક્શનાલિટી
4. હેશ ટેબલ્સ (ચેઇનિંગ)
5. ગ્રાફ્સ માટે એડજસન્સી લિસ્ટ

નોડ ગણવા માટેનો અલ્ગોરિધમ:




```python
def count_nodes(head):
    count = 0
    temp = head

    while temp:
        count += 1
        temp = temp.next

    return count

# ઉદાહરણ
# ધારી લો કે head લિન્ક્ડ લિસ્ટના પ્રથમ નોડને પોઇન્ટ કરે છે
total_nodes = count_nodes(head)
print(f"કુલ નોડ: {total_nodes}")
```

મેમરી ટ્રીક

``ગણો ત્યારે ખસો''

પ્રશ્ન 4(a) [3 ગુણ]

લીનીયર સર્ચ અને બાયનરી સર્ચની સરખામણી કરો.

જવાબ

કોષ્ટક 14. લીનીયર વિરુદ્ધ બાયનરી સર્ચ

ફીચર	લીનીયર સર્ચ	બાયનરી સર્ચ
ડેટા ગોઠવણ	સોર્ટેડ અને અનસોર્ટેડ બંને ડેટા પર કામ કરે છે	ફક્ત સોર્ટેડ ડેટા પર કામ કરે છે
ટાઇમ કોમ્પ્લેક્સિટી	O(n)	O(log n)
ઇમ્પ્લિમેન્ટેશન	સરળ	વધુ જટિલ
શેના માટે શ્રેષ્ઠ	નાના ડેટાસેટ અથવા અનસોર્ટેડ ડેટા	મોટા સોર્ટેડ ડેટાસેટ

આકૃતિ 13. સર્ચ સરખામણી

મેમરી ટ્રીક

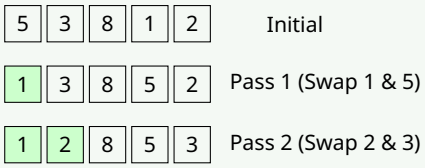
``લીનીયર બધું જુએ, બાઈનરી આધું કાપે''

પ્રશ્ન 4(b) [4 ગુણ]

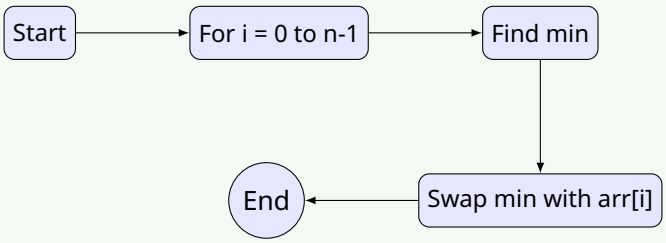
સિલેકશન સોર્ટ માટેનો અલગોરિધમ લખો.

જવાબ

વિઝ્યુઅલાઈઝેશન:



અલ્ગોરિધમ:




```python
def selection_sort(arr):
    n = len(arr)

    for i in range(n):
        min_idx = i

        # અનસોર્ટેડ એરેમાં લઘુતમ એલમેન્ટ શોધો
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j

        # શોધેલા લઘુતમ એલમેન્ટને પ્રથમ એલમેન્ટ સાથે સ્વેપ કરો
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

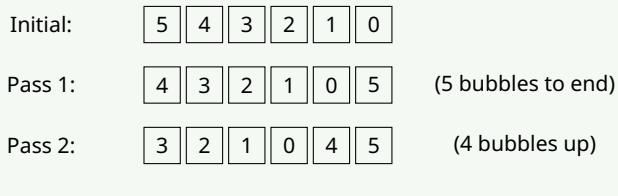
મેમરી ટ્રીક

``લઘુતમ શોધો, પોઝિશન બદલો''

પ્રશ્ન 4(c) [7 ગુણ]

નીચે આપેલા લીસ્ટ ને બબલ સોર્ટ મેથડ વડે ચઢતા ક્રમમાં ગોઠવવા માટેનો પાયથન કોડ વિકસાવો.
list1=[5,4,3,2,1,0]

જવાબ



આકૃતિ 14. બબલ સોર્ટ ટ્રેસ

```python
def bubble_sort(arr):
    n = len(arr)

    # બધા એરે એલમેન્ટ્સ પર ટ્રાવર્સ કરો
    for i in range(n):
        # છેલ્લા i એલમેન્ટ્સ પહેલેથી જ યોગ્ય જગ્યા પર છે
        for j in range(0, n-i-1):
            # જો વર્તમાન એલમેન્ટ આગળના એલમેન્ટ કરતાં મોટો હોય, તો સ્વેપ કરો
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

    return arr

# ઇનપુટ લિસ્ટ
list1 = [5, 4, 3, 2, 1, 0]

# લિસ્ટ સોર્ટ કરવી
sorted_list = bubble_sort(list1)

# રિઝલ્ટ ડિસ્પ્લે કરવું
print("સોર્ટેડ લિસ્ટ:", sorted_list)
# આઉટપુટ: સોર્ટેડ લિસ્ટ: [0, 1, 2, 3, 4, 5]
```

મેમરી ટ્રીક

``મોટા બબલ ઉપર જાય''

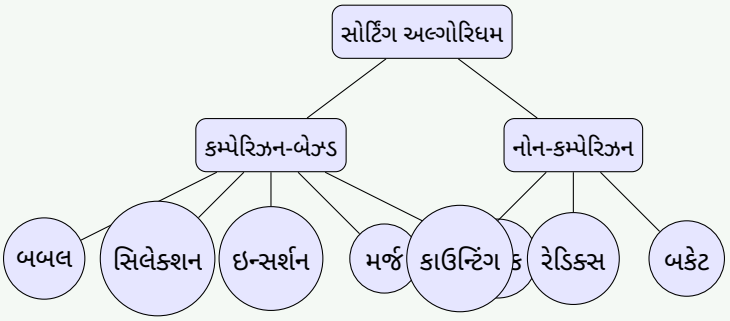
પ્રશ્ન 4(a) OR [3 ગુણ]

સોર્ટિંગ ની વ્યાખ્યા આપો. વિવિધ પ્રકારના સોર્ટિંગ ની યાદી આપો.

જવાબ

કોષ્ટક 15. સોર્ટિંગ વ્યાખ્યા

વ્યાખ્યા	સોર્ટિંગ મેથડ્સ
સોર્ટિંગ એટલે ડેટાને ચોક્કસ ક્રમમાં (ચઢતા અથવા ઉતરતા) ગોઠવવાની પ્રક્રિયા	1. બબલ સોર્ટ 2. સિલેક્શન સોર્ટ 3. ઇન્સર્શન સોર્ટ 4. મર્જ સોર્ટ 5. ક્વિક સોર્ટ 6. હીપ સોર્ટ 7. રેડિક્સ સોર્ટ



આકૃતિ 15. સોર્ટિંગ અલ્ગોરિધમ્સ

મેમરી ટ્રીક

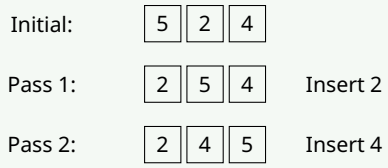
``સારા સોર્ટથી શોધવાનું સરળ બને''

પ્રશ્ન 4(b) OR [4 ગુણ]

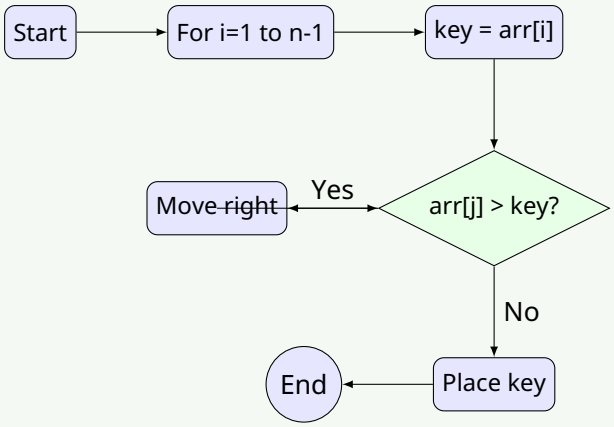
Insertion sort method નો અલગોરિધમ લખો.

જવાબ

વિઝ્યુઅલાઈઝેશન:



અલ્ગોરિધમ:




```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1

        # key કરતાં મોટા એલમેન્ટ્સને એક પોઝિશન આગળ ખસેડો
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
```

```
    arr[j + 1] = key
```

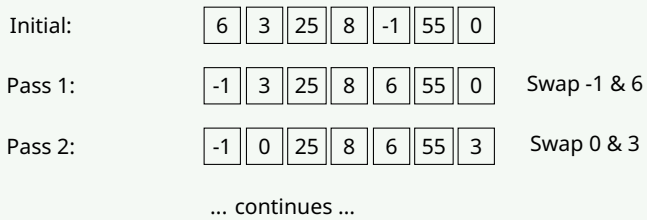
મેમરી ટ્રીક

``કાર્ડ લો, યોગ્ય ક્રમમાં મૂકો''

પ્રશ્ન 4(c) OR [7 ગુણ]

નીચે આપેલા લીસ્ટ ને સિલેકશન સોર્ટ મેથડ વડે ચઢતા ક્રમમાં ગોઠવવા માટેનો પાયથન કોડ વિકસાવો.
list1=[6,3,25,8,-1,55,0]

જવાબ



આકૃતિ 16. Selection Sort Trace

```
def selection_sort(arr):
    n = len(arr)

    for i in range(n):
        # બાકીના અનસોર્ટેડ એરેમાં લઘુતમ એલમેન્ટ શોધો
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j

        # શોધેલા લઘુતમ એલમેન્ટને પ્રથમ એલમેન્ટ સાથે સ્વેપ કરો
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

    return arr

# ઇનપુટ લિસ્ટ
list1 = [6, 3, 25, 8, -1, 55, 0]

# લિસ્ટ સોર્ટ કરવી
sorted_list = selection_sort(list1)

# રિઝલ્ટ ડિસ્પ્લે કરવું
print("સોર્ટેડ લિસ્ટ:", sorted_list)
# આઉટપુટ: સોર્ટેડ લિસ્ટ: [-1, 0, 3, 6, 8, 25, 55]
```

મેમરી ટ્રીક

``નાનામાં નાનું શોધો, આગળ મૂકો''

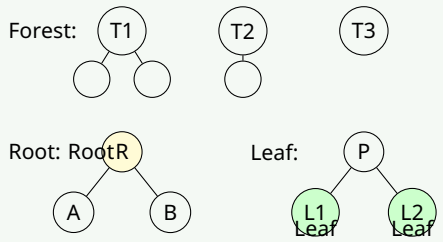
પ્રશ્ન 5(a) [3 ગુણ]

ટ્રી ડેટા સ્ટ્રક્ચરના સંદર્ભમાં નીચેના પદો વ્યાખ્યાયિત કરો:
૧. ફોરેસ્ટ
૨. રૂટ નોડ
૩. લીફ નોડ

જવાબ

કોષ્ટક 16. ટ્રી પરિભાષા

શબ્દ	વ્યાખ્યા
ફોરેસ્ટ	ડિસજોઇન્ટ ટ્રીનો સંગ્રહ (એકબીજા સાથે જોડાણ વિનાના અનેક ટ્રી)
રૂટ નોડ	પેરેન્ટ વગરનો ટ્રીનો સૌથી ઉપરનો નોડ, જ્યાંથી અન્ય તમામ નોડ્સ ઉતરી આવે છે
લીફ નોડ	જે નોડને કોઈ ચિલ્ડ્રન નથી (ટ્રીના તળિયે ટર્મિનલ નોડ)



આકૃતિ 17. ટ્રી ટર્મ્સ

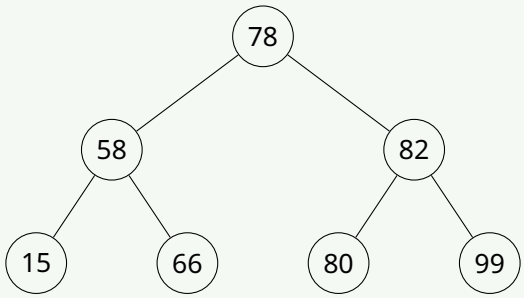
મેમરી ટ્રીક

``ફોરેસ્ટમાં ઘણા મૂળ, મૂળ બધાને દોરે''

પ્રશ્ન 5(b) [4 ગુણ]

78, 58, 82, 15, 66, 80, 99 માટે બાઈનરી સર્ચ ટ્રી દોરો અને ટ્રી માટે ઈન-ઓર્ડર ટ્રાવર્સલ લખો.

જવાબ



આકૃતિ 18. આપેલ ડેટા માટે બાઈનરી સર્ચ ટ્રી

ઈન-ઓર્ડર ટ્રાવર્સલ:

સ્ટેપ	મુલાકાત ક્રમ
1	78 નો ડાબો સબટ્રી મુલાકાત લો
2	58 નો ડાબો સબટ્રી મુલાકાત લો
3	15 ની મુલાકાત લો
4	58 ની મુલાકાત લો
5	66 ની મુલાકાત લો
6	78 ની મુલાકાત લો
7	82 નો ડાબો સબટ્રી મુલાકાત લો
8	80 ની મુલાકાત લો
9	82 ની મુલાકાત લો
10	99 ની મુલાકાત લો

ઈન-ઓર્ડર ટ્રાવર્સલ પરિણામ: 15, 58, 66, 78, 80, 82, 99

મેમરી ટ્રીક

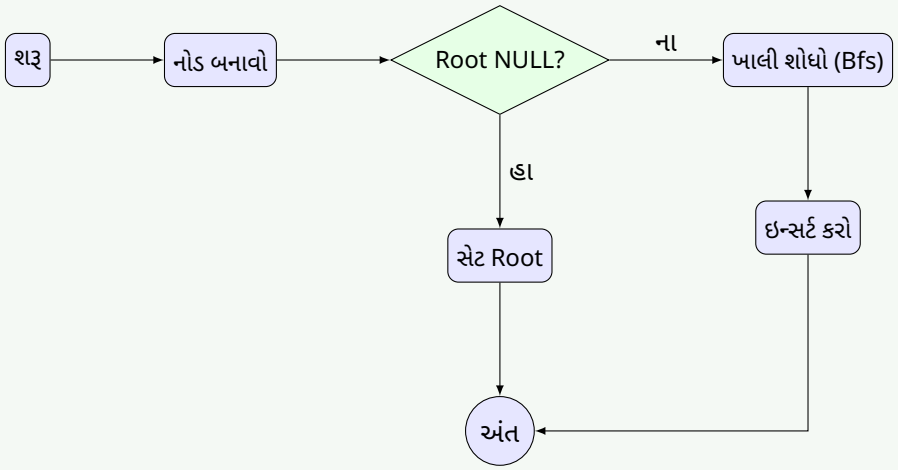
``ડાબે, મૂળ, જમણે''

પ્રશ્ન 5(c) [7 ગુણ]

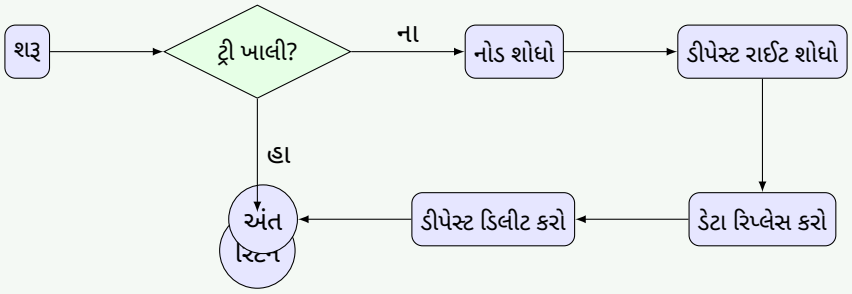
નીચેના ઓપરેશન માટે અલગોરિધમ લખો:
૧. બાઈનરી ટ્રીમાં નોડનું ઇન્સર્શન
૨. બાઈનરી ટ્રીમાં નોડનું ડિલીશન

જવાબ

ઇન્સર્શન અલ્ગોરિધમ:



ડિલીશન અલ્ગોરિધમ:




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

# બાઈનરી ટ્રીમાં ઇન્સર્શન
def insert(root, data):
    if root is None:
        return Node(data)

    # ખાલી જગ્યા શોધવા માટે લેવલ ઓર્ડર ટ્રાવર્સલ
    queue = []
    queue.append(root)

    while queue:
        temp = queue.pop(0)

        if temp.left is None:
            temp.left = Node(data)
            break
        else:
            queue.append(temp.left)

        if temp.right is None:
            temp.right = Node(data)
            break
        else:
            queue.append(temp.right)

    return root

# બાઈનરી ટ્રીમાં ડલિીશન
def delete_node(root, key):
    if root is None:
        return None

    if root.left is None and root.right is None:
        if root.data == key:
            return None
        else:
            return root

    # ડલિીટ કરવા માટેનો નોડ અને ડીપેસ્ટ નોડ શોધો
    key_node = None
    last = None
    parent = None
    queue = []
    queue.append(root)

    while queue:
```

```
        temp = queue.pop(0)
        if temp.data == key:
            key_node = temp
        if temp.left:
            parent = temp
            queue.append(temp.left)
            last = temp.left
        if temp.right:
            parent = temp
            queue.append(temp.right)
            last = temp.right

    if key_node:
        key_node.data = last.data
        if parent.right == last:
            parent.right = None
        else:
            parent.left = None

    return root
```

મેમરી ટ્રીક

``ખાલી જગ્યાએ ઉમેરો, સ્વેપ કરીને કાઢો''

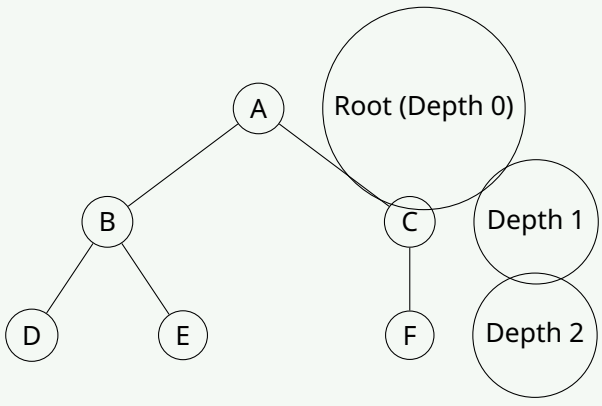
પ્રશ્ન 5(a) OR [3 ગુણ]

ટ્રી ડેટા સ્ટ્રક્ચરના સંદર્ભમાં નીચેના પદો વ્યાખ્યાયિત કરો:
૧. ઇન-ડિગ્રી
૨. આઉટ-ડિગ્રી
૩. ડેપ્થ

જવાબ

કોષ્ટક 17. વ્યાખ્યાઓ

શબ્દ	વ્યાખ્યા
ઇન-ડિગ્રી	નોડમાં આવતી એજેસની સંખ્યા (ટ્રીમાં રૂટ નોડ સિવાય દરેક નોડ માટે હંમેશા 1 હોય છે)
આઉટ-ડિગ્રી	નોડમાંથી બહાર જતી એજેસની સંખ્યા (ચિલ્ડ્રનની સંખ્યા)
ડેપ્થ	રૂટથી નોડ સુધીના પાથની લંબાઈ (પાથમાં એજેસની સંખ્યા)



આકૃતિ 19. ટ્રી ડેપ્થ અને ડિગ્રી

કોષ્ટક 18. ડિગ્રી વિશ્લેષણ

નોડ	ઇન-ડિગ્રી	આઉટ-ડિગ્રી
A	0	2
B	1	2
C	1	1
D	1	0
E	1	0
F	1	0

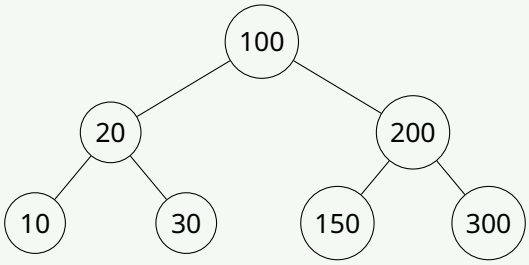
મેમરી ટ્રીક

``ઇન પેરેન્ટ ગણે, આઉટ ચાઈલ્ડ ગણે, ડેપ્થ રૂટથી એજેસ ગણે''

પ્રશ્ન 5(b) OR [4 ગુણ]

નીચેના બાઈનરી ટ્રી માટે પ્રી-ઓર્ડર અને પોસ્ટ-ઓર્ડર ટ્રાવર્સલ લખો.
100 -> (20 -> (10, 30), 200 -> (150, 300))

જવાબ



આકૃતિ 20. આપેલ બાઈનરી ટ્રી

કોષ્ટક 19. ટ્રાવર્સલ્સ

ટ્રાવર્સલ	ક્રમ	પરિણામ
પ્રી-ઓર્ડર	રૂટ, લેફ્ટ, રાઈટ	100, 20, 10, 30, 200, 150, 300
પોસ્ટ-ઓર્ડર	લેફ્ટ, રાઈટ, રૂટ	10, 30, 20, 150, 300, 200, 100

મેમરી ટ્રીક

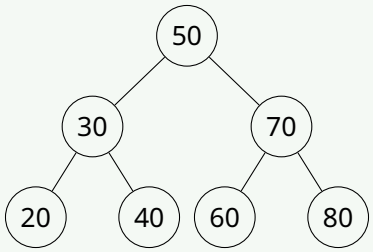
``પ્રી: રૂટ પહેલા, પોસ્ટ: ચિલ્ડ્રન પહેલા''

પ્રશ્ન 5(c) OR [7 ગુણ]

બાઈનરી સર્ચ ટ્રીના કન્સ્ટ્રકશન માટે પ્રોગ્રામ ડેવલપ કરો.

જવાબ

કન્સ્ટ્રક્શન વિઝ્યુઅલાઈઝેશન:



આકૃતિ 21. [50, 30, 20, 40, 70, 60, 80] માંથી બનાવેલ BST

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

def insert(root, key):
    # જો ટ્રી ખાલી હોય, તો નવો નોડ પરત કરો
    if root is None:
        return Node(key)

    # નહીંતર, ટ્રીમાં નીચે જાઓ
    if key < root.key:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)

    # યથાવત નોડ પોઇન્ટર પરત કરો
    return root

def inorder(root):
    if root:
        inorder(root.left)
        print(root.key, end=" ")
        inorder(root.right)

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

def postorder(root):
    if root:
        postorder(root.left)
        postorder(root.right)
        print(root.key, end=" ")

# ડ્રાઈવર પ્રોગ્રામ
def main():
    # આ એલમેન્ટ્સ સાથે BST બનાવો: 50, 30, 20, 40, 70, 60, 80
    root = None
    elements = [50, 30, 20, 40, 70, 60, 80]

    for element in elements:
        root = insert(root, element)

    # ટ્રાવર્સલ્સ પ્રિન્ટ કરો
```

```
    print("Inorder traversal: ", end="")
    inorder(root)
    print("\nPreorder traversal: ", end="")
    preorder(root)
    print("\nPostorder traversal: ", end="")
    postorder(root)

# પ્રોગ્રામ રન કરો
main()
```

ઉદાહરણ આઉટપુટ:

```
Inorder traversal: 20 30 40 50 60 70 80
Preorder traversal: 50 30 20 40 70 60 80
Postorder traversal: 20 40 30 60 80 70 50
```

મેમરી ટ્રીક

``નાના ડાબે, મોટા જમણે''