

Algoritma Struktur Data

ARDIANSYAH

*Departemen Informatika
Universitas Ahmad Dahlan*

Kandidat Penerbit



Ucapan Terima Kasih

Banyak pihak terlibat dan berkontribusi selama penulisan buku ini yang persis dimulai sejak Semester Gasal 2023/2024. Peran mereka sangat bervariasi, mulai dari koreksi kesalahan ketik dan algoritma, hingga masukan-masukan berupa pendalaman materi, variasi soal latihan dan banyak lagi lainnya. Tanpa mereka, buku ini tidak akan terwujud seperti yang pembaca pegang saat ini. Oleh karena itu, halaman ini penulis dedikasikan khusus untuk mengucapkan terima kasih kepada mereka.

Mahasiswa Struktur Data 211830731 Informatika UAD Kelas A & J 2023/2024, Kelas A 2024/2025. Mereka adalah pembaca generasi awal sewaktu buku ini sewaktu masih berupa lembaran catatan kuliah. Mereka pula yang menjadi “*proof-reader*” awal *cum* “objek percobaan” penerapan materi buku ini ke dalam perkuliahan dan praktikum. Bapak Suprihatin (SI UAD), Wahyu Pujiyono (IF UAD), Adhi Prahara (IF UAD), Profesor Paulus Insap Santosa (DTETI UGM), Profesor Mhd. Reza Pulungan (FMIPA UGM). Dr. Rinaldi Munir (IF ITB), Dr. Dewi Pramudi Ismi (IF UAD), Bambang Prastowo (FMIPA UGM), Narendra Wicaksono (Dicoding), Programer Zaman Now (), Xx, xx, xx dan xx untuk studi kasus Pemutar Lagu, xx, xx, dan xx untuk studi kasus Antrian Berprioritas.

Prasyarat

Agar bisa efektif memahami buku ini, paling tidak pembaca telah memahami Algoritma dan Pemrograman dalam salah satu bahasa pemrograman tingkat tinggi seperti Python, C, atau C++, dan sebagainya. Walaupun di buku ini pseudocode yang diberikan bernuansa Python dan C++.

Profil Penulis



Ardiansyah adalah Dosen dan Peneliti di Program Studi S1 Informatika dan S2 Teknik Elektro Universitas Ahmad Dahlan (UAD). Karir akademiknya dimulai sejak tahun 2004. Ardiansyah menyelesaikan S1 Teknik Informatika UAD tahun 2003 dengan judul Skripsi “*Model Pembayaran Pada Mobile Commerce*”. Studi S2 ditempuhnya hanya dalam waktu 18 bulan di Program Studi Magister Ilmu Komputer UGM dengan judul Tesis “*Aplikasi Bibliografi Digital Berbasis Social Cataloging*” yang mengantarkannya sebagai lulusan tercepat di tahun 2009-2010. Ia memungkas studi Pascasarjanya di Program Studi Doktor Teknik Elektro DTETI UGM dalam kurun waktu 3,8 tahun (2019 – 2022). Riset doktoralnya di bidang *software engineering* dengan judul Disertasi “*MUCPSO: Metode Estimasi Use Case Points xxx*”.

Ardiansyah telah mengarang xx buku, xx artikel ilmiah. Mata kuliah yang diampunya antara lain: Algoritma dan Pemrograman, Struktur Data, Rekayasa Perangkat Lunak, Teknik Optimasi, Metodologi Penelitian, dan Penjaminan Kualitas Perangkat Lunak.

Bidang riset yang ditekuninya adalah rekayasa perangkat lunak cerdas (*intelligent software engineering*). Di sela-sela aktivitasnya, ia kerap membantu para Dosen yang hendak studi S3 di bidang Informatika dalam kegiatan Informatics Doctoral Bootcamp.

Ardiansyah pernah meraih beberapa hibah riset kompetitif antara lain Riset Dasar DRTPM tahun 2023. Saat ini Ardiansyah menjadi Ketua Kelompok Keilmuan Rekayasa Perangkat Lunak dan Data (Relata) Prodi S1 Informatika UAD.

Ardiansyah terbuka untuk kolaborasi penelitian di beberapa bidang *software engineering* antara lain: *software requirement prioritization*, *code vulnerability*, *software defect prediction*, *software effort estimation*, *software mining repositories*, dan *empirical software engineering*.

Ardiansyah bisa dihubungi melalui:

Email: ardiansyah@tif.uad.ac.id

HP: 0815 689 2648

Daftar Isi

Ucapan Terima Kasih	2
Prasyarat.....	3
Profil Penulis.....	4
Daftar Algoritma.....	9
Konvensi Notasi Algoritmik	10
Pendahuluan.....	11
Apa itu Struktur Data?	11
Mengapa belajar Struktur Data?	11
Apa bedanya Algoritma dengan Struktur Data?	12
Komponen utama struktur data	12
Contoh umum struktur data	12
Array	14
Apa itu Array	14
Ciri utama Array	14
Bentuk Array.....	14
Konsep penting Array	14
Operasi Array.....	14
Inisiasi Array.....	14
Insert satu elemen	14
Insert elemen.....	15
Hapus elemen.....	16
Latihan	17
Linked List.....	18
Konsep Umum Linked List.....	18
Node.....	18
Traverse (jelajah)	18
Linked list	18
Kepala (<i>head</i>).....	18
Ekor (<i>tail</i>).....	18
Singly Linked List.....	18

Membuat linked list	18
Menambah node di kepala (<i>head</i>).....	18
Menambah node di ekor (<i>tail</i>).....	19
Menambah Node setelah sembarang node.....	21
Menambah Node sebelum sembarang node	22
Mencetak Linked List	23
Pencarian Node	23
Menghapus Node	24
Menghapus node pada sembarang posisi setelah node kepala.....	24
Menghapus node kepala.....	26
Circular Singly Linked List	26
Mengurutkan Node	28
Mengurutkan secara menaik (<i>ascending</i>)	28
Mengurutkan secara menurun (<i>descending</i>)	28
Latihan	28
Dolby Linked List.....	29
Insert di kepala.....	29
Insert di ekor	30
Cetak Maju.....	30
Cetak Mundur	30
Studi Kasus Aplikasi Pemutar Lagu	31
Latihan	31
Bibliografi	32
Antrian (Queue)	33
Antrian menggunakan <i>array</i>	34
Menambah antrian (<i>enqueue</i>).....	34
Menghapus antrian (<i>dequeue</i>).....	36
Antrian menggunakan <i>linked list</i>	37
Latihan	38
Tumpukan (Stack).....	39
Stack representasi <i>array</i>	39
Menambah elemen (<i>push</i>).....	39
Menghapus elemen (<i>pop</i>).....	40

Stack representasi <i>linked list</i>	40
Latihan	41
Binary Search Tree	42
Menghitung jumlah <i>node</i>	42
Menghitung Jumlah Leaf Node.....	43
Menghitung height suatu tree.....	44
Penambahan dan pembuatan <i>node</i>	46
Traverse	49
Preorder traversal	49
In-order traversal.....	51
Postorder traversal	52
Pencarian Node	53
Mencari nilai terbesar	55
Mencari nilai terkecil.....	56
Penghapusan Simpul.....	56
Latihan	59
Bibliografi.....	60
AVL Tree.....	61
Heap.....	66
Antrian Berprioritas	66
Pembuatan binary max heap	66
Pencarian node pada max binary heap.....	71
Pencarian node pada min binary heap.....	71
Menghapus node pada max binary heap.....	71
Menghapus node pada min binary heap.....	72
Latihan	72
Bibliografi.....	72
B++	73
Graf	74
Trie.....	75
Tabel Hash.....	76

Referensi	77
-----------------	----

Daftar Algoritma

Konvensi Notasi Algoritmik

<code>⌊</code>	: operasi <i>floor</i> untuk pembulatan ke bawah
<code>⌈</code>	: operasi <i>ceil</i> untuk pembulatan ke atas
<code>pop()</code>	: menghapus elemen di posisi terakhir <i>array</i>
<code>push()</code>	: menambah elemen baru di awal <i>array</i>
<code>len</code>	: panjang atau ukuran <i>array</i>
<code>[]</code>	: deklarasi variabel <i>array</i>
<code>=</code>	: operator penugasan (<i>assignment</i>) atau pengisian
<code>div()</code>	:
<code>mod()</code>	:
<code>!=</code>	: operator tidak sama dengan
<code>==</code>	: operator sama dengan atau setara
<code>></code>	: operator lebih dari
<code><</code>	: operator kurang dari
<code>>=</code>	: operator lebih dari sama dengan
<code><=</code>	: operator kurang dari sama dengan
<code>()</code>	: pengelompokan kondisi pada suatu ekspresi logika atau kelompok parameter pada suatu fungsi atau prosedur
<code>and</code>	: operator logika AND
<code>or</code>	: operator logika OR
<code>if</code>	: statement kondisional
<code>else if</code>	:
<code>else</code>	:
<code>return</code>	: pengembalian dari sebuah fungsi
<code>for</code>	: statement perulangan
<code>while</code>	: statement perulangan
<code>ASCII()</code>	: mendapatkan nilai integer dari sebuah karakter (huruf, angka, atau karakter khusus)
<code>[-1]</code>	: elemen di posisi terakhir pada <i>array</i>
<code>null</code>	: variabel atau objek yang tidak memiliki nilai atau referensi yang valid

Pendahuluan

Apa itu Struktur Data?

Struktur data adalah cara **mengelola** dan **mengakses** data pada program komputer agar bisa **efisien, efektif** dalam pengoperasian seperti **pencarian, penyortiran, dan manipulasi** data.

Mengapa belajar Struktur Data?

Struktur Data adalah bagian yang melekat dalam suatu algoritma. Setelah Anda belajar dasar pemrograman dan algoritma, maka penguasaan struktur data adalah mutlak. Tanpa paham struktur data, maka algoritma yang Anda gunakan bisa dipertanyakan efektivitas dan efisiensinya. Jika dibuat dalam bentuk daftar, maka berikut ini adalah beberapa alasan mengapa belajar struktur data itu sangat penting.

1. Efisiensi Algoritma

Efisiensi adalah kunci dalam suatu program. Kita harus mampu mengoptimalkan sumber daya komputasi yang dimiliki yang kadang terbatas. Pemahaman yang baik dalam memilih struktur data yang tepat dan cocok untuk kasus atau tugas tertentu memberi dampak besar pada efisiensi algoritma. Dengan kata lain, struktur data yang tepat bisa mengoptimalkan waktu eksekusi dan penggunaan memori komputer.

2. Penyelesaian masalah yang kompleks

Selain efisiensi, pemilihan struktur data yang tepat berarti menunjukkan kemampuan kita dalam merepresentasikan dan memanipulasi data untuk masalah yang sesuai yang pada gilirannya membantu kita dalam memecahkan permasalahan yang kompleks. Dengan kata lain, tantangan pemrograman yang sulit bisa diatasi dengan pemilihan struktur data yang tepat.

3. Kemampuan analisis

Salah satu kompetensi orang Informatika adalah kemampuan analisis. Belajar struktur data mengasah keterampilan kita dalam menganalisis data. Apalagi saat ini penguasaan ilmu tentang data menjadi sangat diperlukan di era saat ini. Dengan penguasaan struktur data kita bisa mengelola, mengolah, dan menganalisis data dengan lebih efektif yang sangat diperlukan dalam dunia bisnis dan penelitian

4. Pengembangan software yang scalable

Salah satu bentuk dari permasalahan yang kompleks adalah ketika kita menjumpai suatu perangkat lunak yang semakin hari makin kompleks dan besar. Apabila kita menjadi bagian di dalamnya, maka pemilihan struktur data yang tepat bisa mempermudah softwarenya terus diperluas sehingga bisa tetap mengelola pertumbuhan data yang berkelanjutan sambil tetap bisa menjaga kinerja aplikasinya optimal.

5. Standar industri

Struktur data adalah konsep umum yang digunakan dalam dunia pemrograman dan Informatika. Banyak bahasa pemrograman dan kerangka kerja yang didasarkan pada penggunaan struktur data tertentu. Oleh karena itu, pemahaman struktur data yang baik akan membantu Anda untuk beradaptasi dengan berbagai teknologi dan lingkungan pengembangan software.

6. Persiapan untuk wawancara kerja

Hampir di setiap wawancara kerja untuk lowongan programmer selalu ada pertanyaan berupa struktur data untuk memecahkan permasalahan yang diberikan. Pertanyaan ini diajukan untuk mengetahui sejauh mana pemahaman Anda dalam penggunaan struktur data. Jika Anda sudah memahami dan menguasai dengan baik tentu sangat membantu dalam melewati wawancara kerja tersebut

Apa bedanya Algoritma dengan Struktur Data?

Komponen utama struktur data

Beberapa komponen utama struktur data yaitu:

1. Elemen data

Elemen data merupakan unit dasar suatu data yang ingin kita simpan atau proses. Sebagai contoh pada daftar nama karyawan, **setiap nama** di dalam daftar tersebut merupakan **elemen data**. Begitu pula misalnya daftar nilai ujian mahasiswa, elemen datanya adalah **setiap nilai ujian** di dalam daftar tersebut.

2. Tipe data

Tipe data adalah jenis nilai yang dapat disimpan dalam struktur data seperti integer, string, karakter, objek atau bahkan record

3. Keterkaitan data

Keterkaitan data adalah cara elemen data memiliki kaitan satu sama lain. Data dapat dihubungkan secara linear, hirarkis, atau bahkan berupa jaringan kompleks

4. Operasi data

Operasi data merupakan tindakan yang dapat dilakukan pada struktur data seperti penambahan, penghapusan, pencarian, penyortiran, atau iterasi melalui elemen-elemen data.

Contoh umum struktur data

Berikut ini adalah beberapa contoh struktur data yang biasanya dipelajari dalam mata kuliah struktur data yaitu:

1. Array

Kumpulan elemen data bertipe sama yang posisi elemennya saling berdempetan (contiguous) dan diakses melalui indeks

2. Linked list

Struktur berantai dari simpul-simpul (node) yang menghubungkan elemen data satu sama lain.

3. Stack

Struktur data LIFO (Last-In, First-Out) yang mengelola elemen data dengan prinsip tumpukan

4. Queue

Struktur data FIFO (First-In, First-Out) yang mengelola elemen data menggunakan prinsip antrian

5. Tree

Struktur data hirarkis yang mengorganisasi elemen data dalam bentuk pohon

6. Graf

Struktur data yang merepresentasikan hubungan antara objek dalam jaringan yang kompleks

7. Hash table

Struktur data yang digunakan untuk mencari data dengan cepat menggunakan fungsi hash

Setiap struktur data tersebut selanjutnya akan dibahas secara lengkap pada bab-bab berikutnya di dalam buku ini.

Array

Tujuan Pembelajaran

Capaian Pembelajaran

Apa itu Array

Definisi array bisa dilihat dari dua konteks. Ketika belajar algoritma dan pemrograman, array bisa disebut sebagai variabel. Sedangkan ketika belajar struktur data, maka array merupakan salah satu struktur data yang mengelola kumpulan elemen yang posisi antarelelemen berdempetan/berurutan satu sama lain (contiguous) dan diakses menggunakan indeks.

Ciri utama Array

Array memiliki ciri utama antara lain:

1. Memiliki banyak sel
2. Tiap sel bisa menampung nilai atau elemen
3. Tiap sel berlokasi berurutan
4. Tiap sel memiliki indeks yang dimulai dari indeks nol (0)
5. Memiliki ukuran yang menunjukkan banyaknya sel
6. Bertipe data tunggal. Artinya satu array hanya bisa bertipe integer, string, atau float saja

Bentuk Array

Bentuk umum array

Contoh array bertipe string

Contoh array bertipe integer

Contoh array bertipe real

Konsep penting Array

Ukuran array \neq indeks array

Ukuran array = indeks terakhir + 1

Indeks terakhir = ukuran array - 1

Kata **elemen**, **nilai**, dan **data** memiliki makna yang sama dan sering digunakan secara bergantian. Jadi pastikan Anda tidak bingung.

Operasi Array

Inisiasi Array

array = []

Insert satu elemen

Sama saja dengan insert elemen di belakang array atau *head*

nilai1 = 3

[3]

nilai2 = 7

[3, 7]

Nilai3 = 2

[3, 7, 2]

Nilai4 = 9

[3, 7, 2, 9]

Insert elemen

Bila diberikan array [3, 7, 2, 9], maka yang dimaksud dengan insert elemen di tengah adalah menambahkan elemen di antara indeks pertama dan indeks terakhir. Sehingga beberapa hal yang perlu dipersiapkan untuk menginsert elemen di tengah adalah:

- Indeks atau posisi elemen baru akan diletakkan
- Elemen yang akan ditambahkan
- Siapkan array temporary untuk menampung array utama
- Perbesar ukuran array utama

Misal:

newValue = 11

Posisi = 2

ArrSize = 4

Hasil yang diharapkan: [3, 7, 11, 2, 9]

Caranya

- Salin array utama ke array temporary
- Kosongkan array utama
- Perbesar ukuran array utama
- Insert seluruh elemen array temporary ke array utama

```
1. fungsi insertArray(values, newValue, position, arraySize)
2.   tempValues = values
3.   values = []
4.   values(arraySize + 1)
5.   for i=0, i<arraySize, i++
6.     values[i] = tempValues[i]
7.   for i=arraySize, i>position, i--
8.     values[i] = tempValues[i-1]
9.   values[position] = newValue
10.  return data
```

Eksekusi Utama

Baris-2	tempValues = [3, 7, 2, 9]	[3, 7, 2, 9]
Baris-3	Values = []	[]
Baris-4	Values[4+1]	[None, None, None, None, None]
Baris-5	For i=0, 0<4, 1++	True

Baris-6	Values[0] = 3	[3, None, None, None, None]
Baris-5	For i=1, 1<4, 2++	True
Baris-6	Values[1] = 7	[3, 7, None, None, None]
Baris-5	For i=2, 2<4, 3++	True
Baris-6	Values[2] = 2	[3, 7, 2, None, None]
Baris-5	For i=3, 3<4, 4++	True
Baris-6	Values[3] = 9	[3, 7, 2, 9, None]
Baris-5	For i=4, 4<4, 5++	False
Baris-7	For i=4, 4>2, 3--	True
Barus-8	values[4] = 9	[3, 7, 2, 9, 9]
Baris-7	For i=3, 3>2, 2--	True
Barus-8	values[3] = 2	[3, 7, 2, 2, 9]
Baris-7	For i=2, 2>2, 1--	True
Barus-8	values[2] = 7	[3, 7, 7, 2, 9]
Baris-7	For i=1, 1>2, 0--	False
Baris-9	Values[2] = 11	[3, 7, 11, 2, 9]
Baris-10	Return [3, 7, 11, 2, 9]	

Intinya, untuk menambahkan elemen di tengah berarti kita harus memperbesar ukuran array utama. Konsekuensinya kita membutuhkan array sementara, lalu memasukkannya kembali ke array utama yang sudah diperbesar, diakhir dengan menggeser tiap elemen hingga ketemu posisi elemen baru untuk ditempatkan.

Artinya, untuk melakukan insert pada array membutuhkan aktivitas yang lumayan banyak, menandakan bahwa struktur data ini memiliki efisiensi yang rendah.

Hapus elemen

```

1. fungsi delArray(values, arrSize, posisi)
2.   tempValues([arrSize]) //deklarasi array dengan ukuran=arrSize
3.   for i=posisi, i<arrSize-1, i++)
4.     values[i] = values[i+1]
5.   tempValues = values
6.   values([arrSize-1]) //deklarasi array dengan ukuran=arrSize-1
7.   for i=0, i<arrSize-1, i++
8.     values[i] = tempValues[i]
9.   return values

```

values = [9, 8, 3, 11, 2]

posisi = 1

arrSize = 5

Baris-1	Fungsi delArray([9, 8, 3, 11, 2], 5, 1)	
---------	---	--

Baris-2	tempValues([arrSize])	[None, None, None, None, None]
Baris-3	For i=1, i<5-1, 2++	True
Baris-4	Values[1] =	[9, 3, 3, 11, 2]
Baris-3	For i=2, 2<5-1, 3++	True
Baris-4	Values[2] = 11	[9, 3, 11, 11, 2]
Baris-3	For i=3, 3<5-1, 4++	True
Baris-4	Values[3] = 2	[9, 3, 11, 2, 2]
Baris-3	For i=4, 4<5-1, 5++	false
Baris-5	tempValues = [9, 3, 11, 2, 2]	[9, 3, 11, 2, 2]
Baris-6	Values([arrSize-1])	[None, None, None, None]
Baris-7	For i=0, 0<5-1, 1++	True
Baris-8	Values[0] = 9	[9, None, None, None]
Baris-7	For i=1, 1<5-1, 2++	True
Baris-8	Values[1] = 3	[9, 3, None, None]
Baris-7	For i=2, 2<5-1, 3++	True
Baris-8	Values[2] = 11	[9, 3, 11, None]
Baris-7	For i=3, 3<5-1, 4++	True
Baris-8	Values[3] = 2	[9, 3, 11, 2]
Baris-7	For i=4, 4<5-1, 5++	False
Baris-9	Return [9, 3, 11, 2]	

Latihan

1. Diberikan array berikut:

72	110	12	66	90
----	-----	----	----	----

Hilangkanlah elemen Null pada array tersebut sehingga array-nya menjadi:

72	12	66	90
----	----	----	----

Linked List

Jika pada array, data disimpan dalam satu grup blok memori yang letak tiap datanya berurutan. Maka, bagaimana jika datanya disimpan secara tersebar di memori? Inilah yang mendasari dibutuhkannya linked list. Pada linked list data disimpan dalam sebuah *node* atau simpul. Setiap simpul terdiri dari dua sel memori. Sel pertama berisi data atau nilai. Sedangkan sel kedua berisi alamat ke simpul berikutnya.

Konsep Umum Linked List

Node

Traverse (jelajah)

Linked list

Kepala (*head*)

Ekor (*tail*)

Konsep utama linked list:

1. Data terletak terpisah-pisah di memori
2. Tiap data direpresentasikan dalam bentuk *node*
3. Tiap *node* menggunakan dua sel memori
4. Sel pertama berisi nilai data, sel kedua berisi alamat ke node berikutnya
5. Node paling kiri disebut *head*, dan node paling kanan disebut *tail*
6. Untuk node *tail* alamat berikutnya selalu *null* atau *none*

Singly Linked List

Membuat linked list

Membuat list dimulai dari menginisiasi linked list kosong. Setelah diinisiasi, dilanjutkan dengan membuat node-node. Penempatan node ke dalam linked list ada dua macam yaitu, penempadan di kepala atau di ekor.

Menambah node di kepala (*head*)

```
1. def createNode(data):
2.     return {'data': data, 'next': None}
3.
4. def insertAtHead(linkedList, data):
5.     newNode = createNode(data)
6.     newNode['next'] = linkedList
7.     return newNode
8.
9. data = ['universitas', 'ahmad', 'dahlan']
10. linkedList = None
11. for item in data:
12.     linkedList = insertAtHead(linkedList, item)
```

Baris	Skrip	Output/Status
4	Def insertAtHead(None, 'universitas')	
5	newNode = createNode('universitas')	{'data': 'universitas', 'next':None}
6	newNode['next'] = None	
7	Return {'data': 'universitas', 'next':None}	
4	Def insertAtHead({'data': 'universitas', 'next':None}, 'ahmad')	
5	newNode = createNode('ahmad')	{'data': 'ahmad', 'next':None}
6	newNode['next'] = {'data': 'universitas', 'next':None},	{'data': 'ahmad', 'next': {'data': 'universitas', 'next':None}}
7	Return {'data': 'ahmad', 'next': {'data': 'universitas', 'next':None}}	
4	Def insertAtHead({'data': 'ahmad', 'next': {'data': 'universitas', 'next':None}}, 'dahlan')	
5	newNode = createNode('dahlan')	{'data': 'dahlan', 'next':None}
6	newNode['next'] = {'data': 'ahmad', 'next': {'data': 'universitas', 'next':None}}	{'data': 'dahlan', 'next': {'data': 'ahmad', 'next': {'data': 'universitas', 'next':None}}}
7	Return {'data': 'dahlan', 'next': {'data': 'ahmad', 'next': {'data': 'universitas', 'next':None}}}	

Menambah node di ekor (*tail*)

<pre> 1. def createNode(data): 2. return {'data': data, 'next': None} 3. 4. def insertAtTail(linkedList, data): 5. newNode = createNode(data) 6. 7. if linkedList is None: 8. return newNode 9. 10. current = linkedList 11. 12. while current['next'] is not None: 13. current = current['next'] 14. 15. current['next'] = newNode 16. 17. return linkedList 18. 19. data = ['universitas', 'ahmad', 'dahlan', 'PTM'] 20. linkedList = None 21. for item in data: 22. linkedList = insertAtTail(linkedList, item) </pre>
--

Eksekusi

Baris	Skrip	Output/Status
-------	-------	---------------

20	linkedList = None	None
21	for 'universitas' in ['universitas', 'ahmad', 'dahlan']	
22	linkedList = insertAtTail(None, 'universitas')	
4	def insertAtTail(None, 'universitas')	
5	newNode = createNode('universitas')	
1	def createNode('universitas')	
2	return {'data': 'universitas', 'next':None}	
7	if None is None:	True
8	return {'data': 'universitas', 'next':None}	
21	for 'ahmad' in ['universitas', 'ahmad', 'dahlan']	
22	linkedList = insertAtTail({'data': 'universitas', 'next':None}, 'ahmad')	
4	def insertAtTail({'data': 'universitas', 'next':None}, 'ahmad')	
5	newNode = createNode('ahmad')	
1	def createNode('ahmad')	
2	return {'data': 'ahmad', 'next':None}	
7	if {'data': 'universitas', 'next':None} is None	False
10	current = {'data': 'universitas', 'next':None}	
12	while None is not None:	False
15	current['next'] = {'data': 'ahmad', 'next':None}	{'data': 'universitas', 'next': {'data': 'ahmad', 'next':None}}
17	return {'data': 'universitas', 'next': {'data': 'ahmad', 'next':None}}	
21	for 'dahlan' in ['universitas', 'ahmad', 'dahlan']	
22	linkedList = insertAtTail({'data': 'universitas', 'next': {'data': 'ahmad', 'next':None}}, 'dahlan')	
4	def insertAtTail({'data': 'universitas', 'next': {'data': 'ahmad', 'next':None}}, 'dahlan')	
5	newNode = createNode('dahlan')	
1	def createNode(dahlan')	
2	return {'data': 'dahlan', 'next':None}	
7	if {'data': 'universitas', 'next': {'data': 'ahmad', 'next':None}} is None	False
10	current = {'data': 'universitas', 'next': {'data': 'ahmad', 'next':None}}	
12	while {'data': 'ahmad', 'next':None} is not None	True
13	current = {'data': 'ahmad', 'next':None}	{'data': 'ahmad', 'next':None}
12	while None is not None	False
15	current['next'] = {'data': 'dahlan', 'next':None}	{'data': 'ahmad', 'next': {'data': 'dahlan', 'next':None}}
17	return {'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'dahlan', 'next':None}}}	

Menambah Node setelah sembarang node

Selain menempatkan atau menambahkan node di kepala atau di ekor, kita juga bisa menambahkan node setelah atau sebelum node tertentu. Dengan demikian, kita sama saja melakukan pencarian node menggunakan keyword tertentu yang selanjutnya akan membuat node baru sebelum atau setelah node yang ditemukan tersebut.

```
1. def insertAfterNode(linkedList, targetData, newData):
2.     newNode = createNode(newData)
3.     current = linkedList
4.     while current:
5.         if current['data'] == targetData:
6.             newNode['next'] = current['next']
7.             current['next'] = newNode
8.             return linkedList
9.         current = current['next']
10.
11. targetData = 'ahmad'
12. newData = 'muhammadiyah'
13. insertAfterNode(linkedList, targetData, newData)
```

Misalnya linked list saat ini:

```
{'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': None}}}
```

Baris	Skrip	Output/Status
2	<code>newNode = {'data': 'muhammadiyah', 'next': None}</code>	<code>{'data': 'muhammadiyah', 'next': None}</code>
3	<code>Current = {'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': None}}}</code>	<code>{'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': None}}}</code>
4	<code>While {'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': None}}}</code>	true
5	<code>If 'universitas' == 'ahmad'</code>	False
10	<code>Current = {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': None}}</code>	<code>{'data': 'ahmad', 'next': {'data': 'dahlan', 'next': None}}</code>
4	<code>While {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': None}}</code>	true
5	<code>If 'ahmad' == 'ahmad'</code>	True
6	<code>newNode['next'] = {'data': 'dahlan', 'next': None}</code>	<code>{'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': None}}</code>

--	--	--

Menambah Node sebelum sembarang node

```

1. def insertBeforeNode(linkedList, targetData, newData):
2.     newNode = createNode(newData)
3.     current = linkedList
4.
5.     while current is not None and current['next'] is not None:
6.         if current['next']['data'] == targetData:
7.             newNode['next'] = current['next']
8.             current['next'] = newNode
9.             return linkedList
10.        current = current['next']
11.
12. targetData = 'dahlan'
13. newData = 'muhammadiyah'
14. insertAfterNode(linkedList, targetData, newData)

```

Misalnya linked list saat ini:

```

{'data': 'universitas',
 'next': {'data': 'ahmad',
          'next': {'data': 'dahlan',
                   'next': {'data': 'PTM',
                             'next': None}}}}

```

Baris	Skrip	Output/Status
2	<code>newNode = {'data': 'muhammadiyah', 'next': None}</code>	<code>{'data': 'muhammadiyah', 'next': None}</code>
3	<code>Current = {'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}}</code>	<code>{'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}}</code>
5	<code>while {'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}} and {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}</code>	True and true
6	<code>If 'ahmad' == 'dahlan'</code>	false
10	<code>Current = {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}</code>	<code>{'data': 'ahmad', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}</code>

5	While {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}} and {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}	True and true
6	If 'dahlan' == 'dahlan'	true
7	newNode['next'] = {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}	{'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}
8	current['next'] = {'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}	{'data': 'ahmad', 'next': {'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}}
9	Return {'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}}}	

Mencetak Linked List

Mencetak linked list berarti kita akan menampilkan data pada node.

Linked list saat ini

```
{'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': None}}}
```

```
1. fungsi printLinkedList(linkedList)
2. while linkedList:
3.     print(linkedList ['data'], end='->')
4.     linkedList = linkedList ['next']
5.     print('none')
```

```
printLinkedList(linkedList)
```

Output: universitas->ahmad->dahlan->none

Linked list saat ini

```
{'data': 'dahlan', 'next': {'data': 'ahmad', 'next': {'data': 'universitas', 'next': None}}}
```

Output: dahlan->ahmad->universitas->none

Pencarian Node

Mencari node berarti hendak menemukan apakah keyword yang diberikan terdapat atau sama pada data yang tersimpan pada node-node. Dengan demikian kita harus membaca tiap node satu per satu hingga ditemukan atau hingga node terakhir.

Dari aspek teknis, teknik cetak linked list bisa kita gunakan pada pencarian node.

```
1. def searchLinkedList(linkedList, target):
2.     while linkedList is not None:
3.         if linkedList['data'] == target:
```

```

4.         return linkedList
5.         linkedList = linkedList['next']
6.         return None

```

```

target = 'dahlan'
if searchLinkedList(linkedList, target):
    print('keyword', target, 'ditemukan')
else:
    print('keyword', target, 'tidak ditemukan')

```

Misalnya Linked list saat ini

```
{'data': dahlan, 'next': {'data': 'ahmad', 'next': {'data': universitas, 'next': None}}}
```

Ketika dieksekusi, maka tahapannya seperti berikut:

Baris-2	While linkedList	True
Baris-3	If 'dahlan' == 'dahlan'	True
Baris-4	Return linkedList	keyword dahlan ditemukan

Misalnya linked list saat ini:

```
{'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': None}}}
```

Ketika dieksekusi, maka tahapannya seperti berikut:

Baris-2	while linkedList	true
Baris-3	if 'universitas' == 'dahlan'	false
Baris-5	linkedList = {'data': 'ahmad', 'next': {'data': 'dahlan', 'next': none}}	
Baris-2	while linkedList	true
Baris-3	if 'ahmad' == 'dahlan'	false
Baris-5	linkedList = {'data': 'dahlan', 'next': none}	
Baris-2	while linkedList	true
Baris-3	if 'dahlan' == 'dahlan'	false
Baris-4	return linkedList	keyword dahlan ditemukan

Output: keyword dahlan ditemukan

Menghapus Node

Menghapus node berarti membuang sebuah node dari sebuah linked list. Ada beberapa kemungkinan penghapusan node yaitu:

Menghapus node pada sembarang posisi setelah node kepala

Prinsip kerjanya adalah:

- Iterasi tiap node sambil mengecek apakah data simpul sama dengan data target
- Jika tidak sama, maka update linked list dengan membuang node yang tidak sama

- c. Jika sama maka arahkan 'next' linked list saat ini ke node 'next' 'next', alias melompati node yang dihapus

```

1. def deleteNode(linkedList, targetData):
2.     current = linkedList
3.     while current['next']:
4.         if current['next']['data'] == targetData:
5.             current['next'] = current['next']['next']
6.             return linkedList
7.         current = current['next']
8.
9. targetData = 'dahlan'
10. deleteNode(linkedList, targetData)

```

Linked list saat ini:

```

{'data': 'universitas',
 'next': {'data': 'ahmad',
          'next': {'data': 'muhammadiyah',
                   'next': {'data': 'dahlan',
                            'next': {'data': 'PTM',
                                       'next': None}}}}}

```

Baris	Skrip	Output/Status
2	Current = {'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}}}	{'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}}}
3	while {'data': 'ahmad', 'next': {'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}}	true
4	If 'ahmad' == 'dahlan'	False
7	Current = {'data': 'ahmad', 'next': {'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}}	{'data': 'ahmad', 'next': {'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}}
3	while {'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}	true
4	If 'muhammadiyah' == 'dahlan'	False
7	Current = {'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}	{'data': 'muhammadiyah', 'next': {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}}

3	While {'data': 'dahlan', 'next': {'data': 'PTM', 'next': None}}	True
4	If 'dahlan' == 'dahlan'	True
5	Current['next'] = {'data': 'muhammadiyah', 'next': {'data': 'PTM', 'next': None}}	{'data': 'muhammadiyah', 'next': {'data': 'PTM', 'next': None}}
6	Return {'data': 'universitas', 'next': {'data': 'ahmad', 'next': {'data': 'muhammadiyah', 'next': {'data': 'PTM', 'next': None}}}}	

Menghapus node kepala

Mungkin pembaca ada yang bertanya-tanya, mengapa operasi penghapusan *node* di ekor tidak dibahas? Jawabannya memang disengaja. Karena nanti penghapus *node* di ekor dibahas sekalian pada materi *dequeue* antrian berbentuk *linked list*. Silakan lihat pada subbab xx.

Circular Singly Linked List

Circular linked list berarti antara node kepala dan node ekor terhubung. Dengan kata lain, node ekor akan merujuk ke node kepala.

```

1. def createSong(song):
2.     return {'song': song, 'next': None}
3.
4. def insertSong(playLists, song):
5.     newSong = createSong(song)
6.     if not playLists:
7.         newSong['next'] = newSong
8.         return newSong
9.
10. playLists = None
11. playLists = insertSong(playLists, 'Satu')
12.
13. print("Play lists lagu Dewa 19:")
14. print(playLists)

```

Eksekusi

Baris	Skrip	Output/Status
10	playLists = None	None
11	playLists = insertSong(None, 'Satu')	
4	def insertSong(None, 'Satu')	
5	newSong = createSong('Satu')	
1	def createSong('Satu')	
2	return {'song': 'Satu' 'next': None}	
5	newSong = {'song': 'Satu' 'next': None}	{'song': 'Satu' 'next': None}
6	if not None:	True
7	newSong['next'] = {'song': 'Satu' 'next': None}	{'song': 'Satu' 'next': {...}}
8	return {'song': 'Satu' 'next': {...}}	

Linked list saat ini: {'song': 'Satu' 'next': {...}}

Perlu diingat bahwa tanda {...} menunjukkan bahwa *next* menunjuk ke *node* yang sama kembali, alias ke node kepala.

```

1. def createSong(song):
2.     return {'song': song, 'next': None}
3.
4. def insertSong(playLists, song):
5.     newSong = createSong(song)
6.
7.     if not playLists:
8.         newSong['next'] = newSong
9.         return newSong
10.    else:
11.        tempPlayLists = playLists
12.
13.        while tempPlayLists['next'] != playLists:
14.            tempPlayLists = tempPlayLists['next']
15.
16.        tempPlayLists['next'] = newSong
17.        newSong['next'] = playLists
18.
19.        return playLists
20.
21. playLists = None
22. playLists = insertSong(playLists, 'Satu')
23. playLists = insertSong(playLists, 'Dewi')
24. playLists = insertSong(playLists, 'Kangen')

```

Eksekusi

Baris	Skrip	Output/Status
23	playLists = insertSong({'song': 'Satu' 'next': {...}}, 'Dewi')	
4	def insertSong({'song': 'Satu' 'next': {...}}, 'Dewi')	
5	newSong = createSong('Dewi')	
1	def createSong('Dewi')	
2	return {'song': 'Dewi' 'next': None}	
5	newSong = {'song': 'Dewi' 'next': None}	{'song': 'Dewi' 'next': None}
6	if not {'song': 'Satu' 'next': {...}}:	False
10	else:	True
11	tempPlayLists = {'song': 'Satu' 'next': {...}}	
13	while {'song': 'Satu' 'next': {...}} != {'song': 'Satu' 'next': {...}}: False	
16	tempPlayLists['next'] = {'song': 'Dewi' 'next': None}	
	Output: tempPlayLists = {'song': 'Satu' 'next': {'song': 'Dewi' 'next': None}} playLists = {'song': 'Satu' 'next': {'song': 'Dewi' 'next': None}}	
17	newSong['next'] = {'song': 'Satu' 'next': {'song': 'Dewi' 'next': None}} #output: newSong = {'song': 'Dewi' 'next': {'song': 'Satu' 'next': {...}}} #output: playLists = {'song': 'Satu' 'next': {'song': 'Dewi' 'next': {...}}}	
19	return {'song': 'Satu' 'next': {'song': 'Dewi' 'next': {...}}}	
24	playLists = insertSong({'song': 'Satu' 'next': {'song': 'Dewi' 'next': {...}}}, 'Kangen')	
4	def insertSong({'song': 'Satu' 'next': {'song': 'Dewi' 'next': {...}}}, 'Kangen')	
5	newSong = createSong('Kangen')	
1	def createSong('Kangen')	
2	return {'song': 'Kangen' 'next': None}	
5	newSong = {'song': 'Kangen' 'next': None}	{'song': 'Kangen' 'next': None}

6	if not {'song': 'Satu' 'next': {'song': 'Dewi' 'next': {...}}}: 'next': {...}}}::	False
10	else:	True
11	tempPlayLists = {'song': 'Satu' 'next': {'song': 'Dewi' 'next': {...}}} output: tempPlayLists['next'] = {'song': 'Dewi' 'next': {...}}	
13	while {'song': 'Dewi' 'next': {...}} != {'song': 'Satu' 'next': {'song': 'Dewi' 'next': {...}}}: True	
14	tempPlayLists = {'song': 'Dewi' 'next': {...}} output: tempPlayLists['next'] = {...}	
16	tempPlayLists['next'] = {'song': 'Kangen' 'next': None} output: tempPlayLists = {'song': 'Dewi' 'next': {'song': 'Kangen' 'next': None}} output: playLists = {'song': 'Satu' 'next': {'song': 'Dewi' 'next': {'song': 'Kangen' 'next': None}}}	
17	newSong['next'] = {'song': 'Satu' 'next': {'song': 'Dewi' 'next': {'song': 'Kangen' 'next': None}}} output: newSong = {'song': 'Kangen' 'next': {'song': 'Satu' 'next': {'song': 'Dewi' 'next': {...}}}} output: playLists = {'song': 'Satu' 'next': {'song': 'Dewi' 'next': {'song': 'Kangen' 'next': {...}}}}	
19	return {'song': 'Satu' 'next': {'song': 'Dewi' 'next': {'song': 'Kangen' 'next': {...}}}}	

Sebagai catatan. Agar seluruh baris kode bisa dieksekusi, maka dibutuhkan paling tidak ada tiga data yang membentuk tiga *node* dalam *linked list*.

Mengurutkan Node

Mengurutkan secara menaik (*ascending*)

Mengurutkan secara menurun (*descending*)

Latihan

1. Diberikan persamaan berikut

$$f(x) = \sum_{i=1}^5 x_i^2$$

Gambarlah bentuk *linked list* nya, lalu buatlah code/pseudocode nya. Cetaklah tiap nilainya beserta hasil akhir fungsi tersebut.

Output:

Gambar

Nilai 1^2 , dst

Total/jumlah

Dolby Linked List

Konsep utama *dolby linked list* (DLL) adalah :

1. Tiap *node* memiliki penunjuk ke *node* sebelum (*previous*) dan penunjuk ke *node* berikutnya (*next*)
2. *Node* pertama memiliki penunjuk *prev* ke *Null* atau *None*

Insert di kepala

Prinsip kerjanya sama dengan *singly linked list* yaitu tiap *node* lama akan bergeser ke ekor tiap kali ada *node* baru yang ditambahkan.

1. Jika linked list kosong, maka linked list hanya terdiri dari satu *node*, dengan *prev* dan *next* bernilai *None*
2. Jika linked list terisi, maka:
 - Penunjuk *next* node baru yang *None* diarahkan ke *linked list* saat ini
 - Penunjuk *prev linked list* saat ini yang *None* diarahkan ke node baru
 - Retun fungsi adalah node baru

```
1. def createNewSong(title):
2.     return {'title': title, 'prev':None, 'next':None}
3.
4. def addSongList(playList, title):
5.     newSong = createNewSong(title)
6.     if playList is None:
7.         return newSong
8.
9.     newSong['next'] = playList
10.    playList['prev'] = newSong
11.
12.    return newSong
13.
14. playList = None
15. playList = addSongList(playList, 'Angin')
16. playList = addSongList(playList, 'Satu')
17. playList = addSongList(playList, 'Kirana')
```

Eksekusi

14	playList = None
15	playList = addSongList(None, 'Angin')
4	def addSongList(None, 'Angin')
5	newSong = createNewSong('Angin')
1	def createNewSong('Angin')
2	return {'title': 'Angin', 'prev':None, 'next':None}
5	newSong = {'title': 'Angin', 'prev':None, 'next':None}
	# playList = None
6	if None is None:
	# True
7	return {'title': 'Angin', 'prev':None, 'next':None}
15	playList = {'title': 'Angin', 'prev':None, 'next':None}
16	playList = addSongList(None, 'Satu')
4	def addSongList(None, 'Satu')
5	newSong = createNewSong('Satu')
1	def createNewSong('Satu')

2	return {'title': 'Satu', 'prev':None, 'next':None}
5	newSong = {'title': 'Satu', 'prev':None, 'next':None} # playlist = {'title': 'Angin', 'prev':None, 'next':None}
6	if {'title': 'Angin', 'prev':None, 'next':None} is None # False # newSong['next'] = None
9	newSong['next'] = {'title': 'Angin', 'prev':None, 'next':None} # newSong = {'title': 'Satu', 'prev':None, 'next': {'title': 'Angin', 'prev':None, 'next':None}} # playlist['prev'] = None
10	playlist['prev'] = {'title': 'Satu', 'prev':None, 'next': {'title': 'Angin', 'prev':None, 'next':None}} # newSong = {'title': 'Satu', 'prev':None, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}
12	return {'title': 'Satu', 'prev':None, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}
17	playlist = addSongList(None, 'Kirana')
4	def addSongList(None, 'Kirana')
5	newSong = createNewSong('Kirana')
1	def createNewSong('Kirana')
2	return {'title': 'Kirana', 'prev':None, 'next':None}
5	newSong = {'title': 'Kirana', 'prev':None, 'next':None} # playlist = {'title': 'Satu', 'prev':None, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}
6	if {'title': 'Satu', 'prev':None, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}} is None # False # newSong['next'] = None
9	newSong['next'] = {'title': 'Satu', 'prev':None, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}} # newSong = {'title': 'Kirana', 'prev':None, 'next': {'title': 'Satu', 'prev':None, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}} # playlist['prev'] = None
10	playlist['prev'] = {'title': 'Kirana', 'prev':None, 'next': {'title': 'Satu', 'prev':None, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}} # newSong = {'title': 'Kirana', 'prev':None, 'next': {'title': 'Satu', 'prev':{...}, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}}
12	return {'title': 'Kirana', 'prev':None, 'next': {'title': 'Satu', 'prev':{...}, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}}

Insert di ekor

Cetak Maju

Cetak Mundur

Prinsip cetak mundur adalah:

1. Telusuri *linked list* hingga *node* ekor menggunakan penunjuk 'next'.
2. Telusuri *linked list* hingga *node* kepala menggunakan penunjuk 'prev' sambil mencetak data tiap *node*.

Tiap loop akan memperbarui *linked list* dengan penunjuk *linked list* 'prev'

1.	fungsi printBackward(songList):
2.	tempSongList = songList
3.	
4.	while tempSongList['next'] != None:
5.	tempSongList = tempSongList['next']
6.	
7.	while tempSongList != None:
8.	print(tempSongList['title'])
9.	tempSongList = tempSongList['prev']
10.	

11. printBackward(songList)

Eksekusi

Linked list saat ini:

```
{'title': 'Kirana', 'prev':None, 'next':
{'title': 'Satu', 'prev':{...}, 'next':
{'title': 'Angin', 'prev':{...}, 'next':None}}}
```

11	printBackward({'title': 'Kirana', 'prev':None, 'next': {'title': 'Satu', 'prev':{...}, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}})
1	fungsi printBackward({'title': 'Kirana', 'prev':None, 'next': {'title': 'Satu', 'prev':{...}, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}})
2	tempSongList = {'title': 'Kirana', 'prev':None, 'next': {'title': 'Satu', 'prev':{...}, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}} # tempSongList['next'] = {'title': 'Satu', 'prev':{...}, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}}
4	while {'title': 'Satu', 'prev':{...}, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}} != None # True
5	tempSongList = {'title': 'Satu', 'prev':{...}, 'next': {'title': 'Angin', 'prev':{...}, 'next':None}} # tempSongList['next'] = {'title': 'Angin', 'prev':{...}, 'next':None}
4	while {'title': 'Angin', 'prev':{...}, 'next':None} # True
5	tempSongList = {'title': 'Angin', 'prev':{...}, 'next':None} # tempSongList['next'] = None
4	While None is not None # False
7	while {'title': 'Angin', 'prev':{...}, 'next':None} != None: # True
8	print('Angin') # tempSongList['prev'] = {'title': 'Satu', 'prev': {'title': 'Kirana', 'prev': None, 'next': {...}}, 'next': {'title': 'Angin', 'prev': {...}, 'next': None}}
9	tempSongList = {'title': 'Satu', 'prev': {'title': 'Kirana', 'prev': None, 'next': {...}}, 'next': {'title': 'Angin', 'prev': {...}, 'next': None}}
7	while {'title': 'Satu', 'prev': {'title': 'Kirana', 'prev': None, 'next': {...}}, 'next': {'title': 'Angin', 'prev': {...}, 'next': None}} != None: # True
8	print('Satu') # tempSongList['prev'] = {'title': 'Kirana', 'prev': None, 'next': {'title': 'Satu', 'prev': {...}, 'next': {'title': 'Angin', 'prev': {...}, 'next': None}}}
9	tempSongList = {'title': 'Kirana', 'prev': None, 'next': {'title': 'Satu', 'prev': {...}, 'next': {'title': 'Angin', 'prev': {...}, 'next': None}}}
7	while {'title': 'Kirana', 'prev': None, 'next': {'title': 'Satu', 'prev': {...}, 'next': {'title': 'Angin', 'prev': {...}, 'next': None}}} != None: # True
8	print('Kirana') # tempSongList['prev'] = None
9	tempSongList = None
7	while None != None: # False

Contoh output: Angin Satu Kirana

Studi Kasus Aplikasi Pemutar Lagu

Latihan

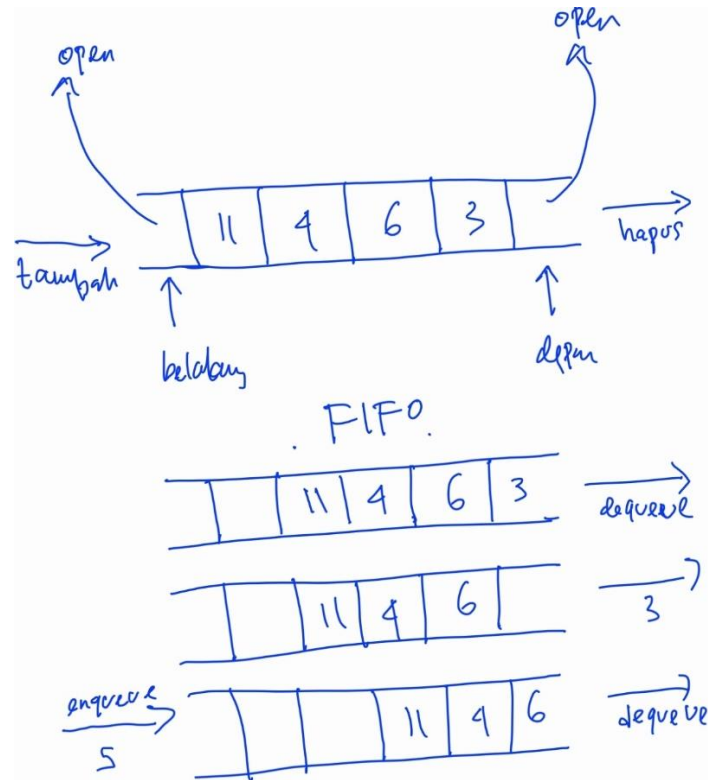
Bibliografi

Penerapan *linked list* sangatlah luas, di antaranya yaitu untuk mempercepat algoritma FP-Growth (Zhenguo et al., 2009), mekanisme *locked-free* (Fomitchev & Ruppert, 2004) untuk berbagi data pada sistem terdistribusi. (Yang et al., 2010) menerapkan *concurrent linked-list* pada prosesor grafis modern yang bermanfaat untuk membuat efek rendering yang kompleks secara *real-time*. Pengurangan *latency* pengaksesan memori (Yamamura et al., 2002).

Antrian (Queue)

Capaian Pembelajaran

Antrian adalah salah satu struktur data abstrak.



Ciri utama antrian adalah:

- Kedua ujung antrian bersifat terbuka
- Bagian belakang merupakan tempat menambah data atau data yang ditambahkan terakhir
- Bagian depan tempat data yang akan segera dihapus atau data yang masuk pertama kali
-

Operasi penting pada antrian yaitu

1. Enqueue atau tambah antrian
2. Dequeue atau hapus antrian
3. Peek atau mengambil data paling depan tanpa dequeue
4. isFull atau antrian penuh
5. isEmpty atau antrian kosong

Antrian menggunakan *array*

Menambah antrian (*enqueue*)

```
1. fungsi enqueue(queue, queueMaxCapacity, newData):
2.     frontIndex = queueMaxCapacity - 1
3.     rearIndex = 0
4.
5.     if queue[frontIndex] == None:
6.         queue[frontIndex] = newData
7.         return queue
8.
9.     if queue[rearIndex] != None:
10.        print 'antrian penuh'
11.        return queue
12.
13.    for (i = queueMaxCapacity-2; i > -1; i--):
14.        if queue[i] == None:
15.            queue[i] = newData
16.            return queue
17.
18. queueMaxCapacity = 4
19. queue = [None, None, None, None]
20.
21. queue = enqueue(queue, queueMaxCapacity, 5)
22. print(queue)
23. queue = enqueue(queue, queueMaxCapacity, 3)
24. print(queue)
25. queue = enqueue(queue, queueMaxCapacity, 9)
26. print(queue)
27. queue = enqueue(queue, queueMaxCapacity, 2)
28. print(queue)
29.
30. newData = 12
31. queue = enqueue(queue, queueMaxCapacity, newData)
32. print(queue)
```

Eksekusi

Baris	Eksekusi
18	queueMaxCapacity = 4
19	queue = [None, None, None, None]
21	queue = enqueue([None, None, None, None], 4, 5)
1	enqueue([None, None, None, None], 4, 5)
2	frontIndex = 3
3	rearIndex = 0 #queue[3] = None
5	if None == None:
6	queue[3] = 5 #queue = [None, None, None, 5]
7	return [None, None, None, 5]
21	queue = [None, None, None, 5]
22	None, None, None, 5
23	queue = enqueue([None, None, None, 5], 4, 3)
1	enqueue([None, None, None, 5], 4, 3)
2	frontIndex = 3 #queue[3] = 5
3	rearIndex = 0 #queue[0] = None
5	if 5 == None #False

```

9         if None != None:         #False
12        for (i=2; 2 > -1; 2--) #True #queue[2] = None
13            if None == None      #True
14                queue[2] = 3      # queue = [None, None, 3, 5]
15                return [None, None, 3, 5]
23    queue = [None, None, 3, 5]
24    print None, None, 3, 5

25    queue = enqueue([None, None, 3, 5], 4, 9)
1    enqueue([None, None, 3, 5], 4, 9)
2        frontIndex = 3            #queue[3] = 5
3        rearIndex = 0            #queue[0] = None
5        if 5 == None             #False
9        if None != None         #False
12        for (i=2; 2 > -1; 2--)    #True #queue[2] = 3
13            if 3 == None:         #False
12        for (i=1; 1 > -1; 1--)    #True #queue[1] = None
13            if None == None:     #True
14                queue[1] = 9      #queue = [None, 9, 3, 5]
15                return [None, 9, 3, 5]
25    queue = [None, 9, 3, 5]
26    print None, 9, 3, 5

27    queue = enqueue([None, 9, 3, 5], 4, 2)
1    enqueue([None, 9, 3, 5], 4, 2)
2        frontIndex = 3            #queue[3] = 5
3        rearIndex = 0            #queue[0] = None
5        if 5 == None:            #False
9        if None != None:         #False
12        for (i=2; 2 > -1; 2--)    #True #queue[2] = 3
13            if 3 == None:         #False
12        for (i=1; 1 > -1; 1--)    #True #queue[1] = 9
13            if 9 == None:         #False
12        for (i=0; 0 > -1; 0--)    #True #queue[0] = None
13            if None == None      #True
14                queue[0] = 2      # queue = [2, 9, 3, 5]
15                return [2, 9, 3, 5]
27    queue = [2, 9, 3, 5]
28    print 2, 9, 3, 5

30    newData = 12
31    queue = enqueue([2, 9, 3, 5], 4, 12)
1    enqueue([2, 9, 3, 5], 4, 12)
2        frontIndex = 3            #queue[3] = 5
3        rearIndex = 0            #queue[0] = 2
5        if 5 == None:            #False
9        if 2 != None:            #True
10            print 'antrian penuh'
11            return [2, 9, 3, 5]
31    queue = [2, 9, 3, 5]
32    print 2, 9 3, 5

```

Menghapus antrian (*dequeue*)

```

1. def queueIsEmpty(frontIndex, queue):
2.     return queue[frontIndex] == None
3.
4. def dequeue(queue, queueMaxCapacity):
5.     frontIndex = -1
6.     tempQueue = queue
7.     queue = [None] * queueMaxCapacity
8.
9.     if queueIsEmpty(frontIndex, tempQueue):
10.        return tempQueue
11.
12.    for i in range(queueMaxCapacity-2, -1, -1):
13.        queue[i+1] = tempQueue[i]
14.    return queue
15.
16. queueMaxCapacity = 4
17. newData = 12
18.
19. queue = dequeue(queue, queueMaxCapacity)
20. print(queue)

```

Eksekusi

Queue saat ini: queue = [2, 9, 3, 5]

16	queueMaxCapacity = 4
17	newData = 12
19	queue = dequeue([2, 9, 3, 5], 4)
4	def dequeue([2, 9, 3, 5], 4)
5	frontIndex = -1
6	tempQueue = [2, 9, 3, 5]
7	queue = [None, None, None, None]
9	if queueIsEmpty(-1, [2, 9, 3, 5])
1	queueIsEmpty(-1, [2, 9, 3, 5])
	#queue[-1] = 5
2	return 5 == None #False
9	if false:
	#tempQueue =
12	for 2 in range(2, -1, -1) #True
13	queue[3] = 3
	#queue = [None, None, None, 3]
12	For 1 in range(2, -1, -1) #True
13	queue[2] = 9
	#queue = [None, None, 9, 3]
12	For 0 in range(2, -1, -1) #True
13	Queue[1] = 2
	#queue = [None, 2, 9, 3]
12	for -1 in range(2, -1, -1) #False
14	return [None, 2, 9, 3]
19	Queue = [None, 2, 9, 3]
20	cetak {None, 2, 9, 3}

Untuk *dequeue* sama dengan materi menghapus elemen *array*. Silakan dibaca kembali subbab xx. Perbedaannya adalah jika pada Subbab xx tersebut, setiap kali menghapus elemen, ukuran *array* langsung berkurang. Sedangkan pada *dequeue*, elemen yang dihapus tidak mengurangi ukuran *array*. Sehingga sel tempat elemen yang dikosongkan tersebut diganti dengan **None** atau **Null**.

Antrian menggunakan *linked list*

Pada prinsipnya sama dengan *linked list* pada umumnya. Perbedaannya terletak pada penambahan kapasitas antrian yang perlu didefinisikan di awal.

Operasi yang dibahas pada materi kali ini adalah *dequeue*. Pada antrian berbentuk *linked list*, *dequeue* berarti menghapus sebuah *node* yang terletak di ekor. Itulah sebabnya mengapa pada bab Singly Linked List, penulis tidak membahas penghapusan *node* di ekor. Karena memang akan dibahas pada materi *dequeue* antrian berupa *linked list*.

```

1. def dequeue(queue):
2.
3.     if queue is None or queue['next'] is None:
4.         return 'Queue is empty.'
5.
6.     tempQueue = queue
7.
8.     while tempQueue['next']['next'] is not None:
9.         tempQueue = tempQueue['next']
10.
11.    tempQueue['next'] = None
12.    return queue
13.
14. queue = {
15.    'data': 10, 'next': {'data': 5, 'next': {'data': 19, 'next': None}}
16. }
17.
18. queue = dequeue(queue)
19. print(queue)
20. queue = dequeue(queue)
21. print(queue)
22. queue = dequeue(queue)
23. print(queue)

```

Eksekusi

14	queue = {'data': 10, 'next': {'data': 5, 'next': {'data': 19, 'next': None}}}
18	queue = dequeue({'data': 10, 'next': {'data': 5, 'next': {'data': 19, 'next': None}}})
1	def queue({'data': 10, 'next': {'data': 5, 'next': {'data': 19, 'next': None}}}): # queue['next'] = {'data': 5, 'next': {'data': 19, 'next': None}}
3	if {'data': 10, 'next': {'data': 5, 'next': {'data': 19, 'next': None}}} is None or {'data': 5, 'next': {'data': 19, 'next': None}} is None #False
6	tempQueue = {'data': 10, 'next': {'data': 5, 'next': {'data': 19, 'next': None}}} #tempQueue['next'] = {'data': 5, 'next': {'data': 19, 'next': None}}

	<code>#tempQueue['next']['next'] = {'data': 19, 'next': None}</code>
8	<code>while {'data': 19, 'next': None} is not None: #True</code>
9	<code>tempQueue = {'data': 5, 'next': {'data': 19, 'next': None}}</code> <code>#tempQueue['next']['next'] = None</code>
8	<code>while None is not None: #False</code> <code>#tempQueue['next'] = {'data': 19, 'next': None}</code>
11	<code>tempQueue['next'] = None</code> <code>#tempQueue = {'data': 10, 'next': {'data': 5, 'next': None}}</code>
12	<code>return {'data': 10, 'next': {'data': 5, 'next': None}}</code>
	LANJUTKAN...

Konsep antrian yang kita bahas di subbab ini merupakan antrian biasa. Artinya antrian tersebut murni menerapkan konsep *first in first out*. Namun, pada situasi tertentu antrian biasa seperti ini tidaklah cukup. Misalnya, ada suatu *node* yang memiliki urgensi yang tinggi sehingga FIFO tidak berlaku. Oleh karena itu digunakanlah antrian berprioritas atau dikenal sebagai *priority queue*. Khusus mengenai antrian berprioritas ini akan dibahas pada Bab xx mengenai struktur data *heap*. Jika pembaca tertarik, langsung saja loncat ke bab tersebut.

Latihan

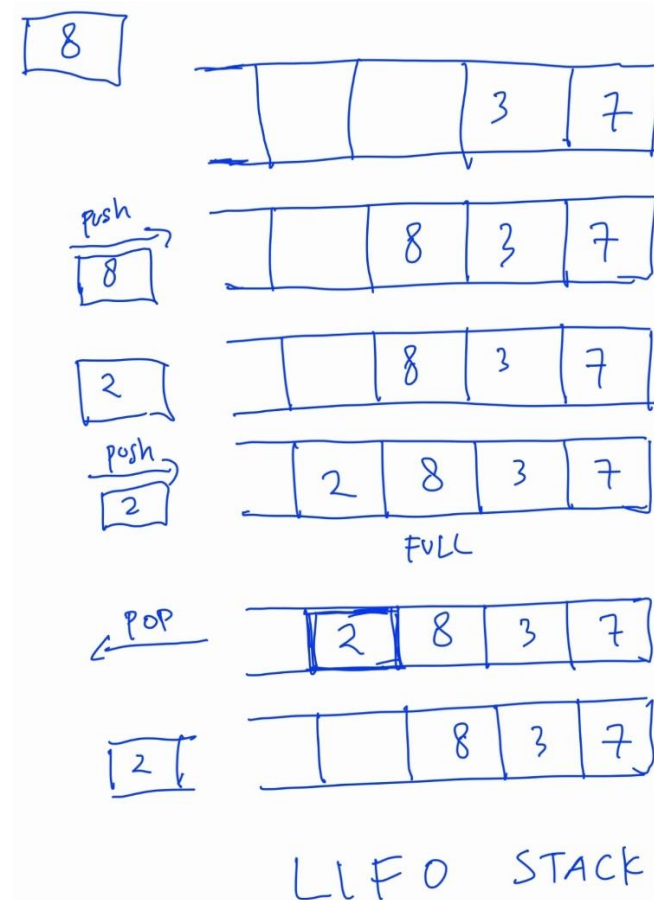
Tumpukan (Stack)

Stack adalah salah satu struktur data yang menggunakan prinsip *last in first out* atau LIFO. Artinya, elemen terakhir yang masuk ke tumpukan adalah elemen yang duluan keluar. Sebaliknya, elemen pertama yang masuk ke tumpukan merupakan elemen yang keluar terakhir.

Prinsip *stack* adalah hanya memiliki satu bagian yang terbuka, yaitu bagian atas (jika vertikal), atau kepala (jika horizontal).

Operasi *stack*

1. menambah elemen baru yaitu *push*
2. mengambil elemen paling atas yaitu *pop*
3. membaca elemen paling atas tanpa menghapus/mengambil yaitu *peek*



Stack representasi array

Menambah elemen (*push*)

- ```
1. fungsi pushStack(stack, stackMaxCapacity, newData):
2. bottomIndex = -1
3. topIndex = 0
4.
5. #jika stack kosong
6. if stack[bottomIndex] == None:
```

```

7. stack[bottomIndex] = newData
8. return stack
9.
10. for (i=stackMaxCapacity-1; i > -1; i--):
11. #jika stack penuh
12. if stack[topIndex] != None:
13. cetak ('Stack sedang penuh')
14.
15. if stack[i] == None:
16. stack[i] = newData
17. return stack
18.
19. stackMaxCapacity = 4
20. stack = [None, None, None, None]
21.
22. newData = 6
23. stack = pushStack(stack, stackMaxCapacity, newData)
24. print(stack)
25. newData = 3
26. stack = pushStack(stack, stackMaxCapacity, newData)
27. print(stack)
28. newData = 11
29. stack = pushStack(stack, stackMaxCapacity, newData)
30. print(stack)

```

### Menghapus elemen (*pop*)

```

1. fungsi popStack(stack):
2. bottomIndex = -1
3. topIndex = 0
4.
5. for (i=0, i < stackMaxCapacity, i++)
6. if stack[bottomIndex] == None:
7. cetak ('Stack sedang kosong')
8. break
9.
10. if stack[i] != None:
11. stack[i] = None
12. return stack
13.
14. stack = popStack(stack)
15. print(stack)
16. stack = popStack(stack)
17. print(stack)

```

### Stack representasi *linked list*



## Latihan

1. Diketahui terdapat  $n$  data yang akan di-*push* ke *stack* [None, None, 4, 2, 12].
  - a. Buatlah fungsi yang merepresentasikan *stack* menggunakan *array*
  - b. Buatlah fungsi yang merepresentasikan *stack* dalam *linked list*
  - c. Buatlah fungsi untuk memeriksa apakah  $n$  data bisa di-*push* atau tidak ke *stack*. Contoh:  $n = 2$ . Outputnya: Data bisa di-*push* ke *stack*. Contoh  $n = 3$ . Output: Data tidak bisa di-*push* ke *stack*

# Binary Search Tree

Binary Search Tree (BST) adalah salah satu struktur data nonlinear yang semua *node*-nya diletakkan secara terurut.

salah satu jenis struktur data yang pemakaiannya sangat populer selain *linked list*. Karakteristik utama *tree*:

**Root node:** adalah *node* yang terletak paling atas. Sehingga, bila `node == None` atau `Null`, berarti tidak ada *tree* alias kosong.

**Sub-tree:** Bila *tree* tidak dalam keadaan kosong, maka ada tiga kemungkinan struktur *tree* yang terbentuk.

*Pertama*, *tree* memiliki sub-*tree* di sebelah kiri, memiliki sub-*tree* di sebelah kanan, atau memiliki sub-*tree* di sebelah kiri dan kanan. Bila disimbolkan, kedua sub-*tree* tersebut adalah  $T_1$  dan  $T_2$ .

**Path:** Urutan garis yang menghubungkan tiap *node*.

**Ancestor node:** Ancestor sebuah *node* adalah serangkaian *node* mulai dari *root node* sampai ke *node* tersebut

**Descendant node:** descendant sebuah *node* adalah serangkaian *node* mulai *leaf node* sampai ke *node* tersebut

**Level number:**

**Degree:**

**In-degree:**

**Out-degree:**

**Height:**

## Menghitung jumlah *node*

### Algoritma xx Menghitung jumlah *node* suatu *tree*

```
1. fungsi hitungNode(node)
2. if node == null
3. return 0
4. else
5. return 1 + hitungNode(node→L) + hitungNode(node→R)
```

### Eksekusi Algoritma xx Menghitung jumlah *node* suatu *tree*

```
root = {'L': N, 'D': 5, 'R': {'L': N, 'D': 9, 'R': N}}
jumlahNode = hitungNode({'L': N, 'D': 5, 'R': {'L': N, 'D': 9, 'R': N}})

1. fungsi hitungNode({'L': N, 'D': 5, 'R': {'L': N, 'D': 9, 'R': N}})
2. if {'L': N, 'D': 5, 'R': {'L': N, 'D': 9, 'R': N}} == None #F
4. else #root[L] = None #root[R] = {'L': N, 'D': 9, 'R': N}
5. return 1 + hitungNode(None) + hitungNode(root[R])
 1. fungsi hitungNode(None)
 2. if None == None #True
```

```

3. return 0
5. return 1 + 0 + hitungNode(root[R])
 1. fungsi hitungNode({'L': N, 'D': 9, 'R': N})
 2. if {'L': N, 'D': 9, 'R': N} == None #F
 4. else
 5. return 1 + hitungNode(None) + hitungNode(None)
 1. fungsi hitungNode(None)
 2. if None == None #True
 3. return 0
 5. return 1 + 0 + hitungNode(None)
 1. fungsi hitungNode(None)
 2. if None == None #True
 3. return 0
 5. return 1 + 0 + 0
 5. return 1
5. return 1 + 0 + 1
5. return 2

```

## Menghitung Jumlah Leaf Node

Algoritma xx Menghitung jumlah *leaf node* suatu *tree*

```

1. function hitungLeafNode(node)
2. if node is null
3. return 0
4. else if (node->left == null) && (node->right == null)
5. return 1
6. else
7. return hitungLeafNode(node->left) + hitungLeafNode(node->right)
8.
9. # Contoh penggunaan
10. totalLeafNode = hitungLeafNode(root)
11. print("Jumlah leaf node pada pohon:", totalLeafNode)

```

Eksekusi Algoritma xx Menghitung jumlah *leaf node* suatu *tree*

```

root = {'L': N, 'D': 5, 'R': {'L': N, 'D': 9, 'R': N}}
jumlahLeafNode = hitungLeafNode({'L': N, 'D': 5, 'R': {'L': N, 'D': 9, 'R': N}})

1. fungsi hitungLeafNode({'L': N, 'D': 5, 'R': {'L': N, 'D': 9, 'R': N}})
2. if ({'L': N, 'D': 5, 'R': {'L': N, 'D': 9, 'R': N}}) == null #F
4. else if null == null and {'L': N, 'D': 9, 'R': N} == null #F
6. else
7. return hitungLeafNode(null) + hitungLeafNode({'L': N, 'D': 9, 'R': N})
 1. hitungLeafNode(null)
 2. if null == null #T
 3. return 0
7. return 0 + hitungLeafNode({'L': N, 'D': 9, 'R': N})
 1. hitungLeafNode({'L': N, 'D': 9, 'R': N})
 2. if {'L': N, 'D': 9, 'R': N} == null #F
 4. else if null == null and null == null #T

```

```

5. return 1
7. return 0 + 1
7. return 1

```

## Menghitung height suatu tree

### Algoritma xx Menghitung *height* (tinggi) suatu *tree*

```

1. function treeHeight(node)
2. if node == null
3. return 0
4. else
5. leftHeight = treeHeight(node→leftChild)
6. rightHeight = treeHeight(node→rightChild)
7.
8. # Tinggi pohon adalah maksimum dari tinggi anak kiri dan anak kanan, ditambah 1
9. height = max(leftHeight, rightHeight) + 1
10. return height

```

### Eksekusi Algoritma xx Menghitung *height* (tinggi) suatu *tree*

```

tree = {{{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}}

1. treeHeight({{{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}})
2. if {{{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}} == null #False
4. else
5. leftHeight = treeHeight({N, 2, N}, 3, N)
 1. treeHeight({N, 2, N}, 3, N)
 2. if {{N, 2, N}, 3, N} == null #False
 4. else
 5. leftHeight = treeHeight({N, 2, N})
 1. treeHeight({N, 2, N})
 2. if {N, 2, N} == null #F
 4. else
 5. leftHeight = treeHeight(null)
 1. treeHeight(null)
 2. if null == null #T
 3. return 0
 5. leftHeight = 0
 6. rightHeight = treeHeight(null)
 1. treeHeight(null)
 2. if null == null #T
 3. return 0
 6. rightHeight = 0
 9. height = max(0, 0) + 1
 9. height = 0 + 1
 9. height = 1
 10. return 1
 5. leftHeight = 1
 6. rightHeight = treeHeight(null)

```

```

1. treeHeight(null)
2. if null == null #T
3. return 0

6. rightHeight = 0
9. height = max(1, 0) + 1
9. height = 1 + 1
9. height = 2
10. return 2

5. leftHeight = 2
6. rightHeight = treeHeight({{N, 7, N}, 9, {N, 10, N}})
 1. treeHeight({{N, 7, N}, 9, {N, 10, N}})
 2. if {{N, 7, N}, 9, {N, 10, N}} == null #F
 4. else
 5. leftHeight = treeHeight({N, 7, N})
 1. treeHeight({N, 7, N})
 2. if {N, 7, N} == null #F
 4. else
 5. leftHeight = treeHeight(null)
 1. treeHeight(null)
 2. if null == null #T
 3. return 0
 5. leftHeight = 0
 6. rightttHeight = treeHeight(null)
 1. treeHeight(null)
 2. if null == null #T
 3. return 0
 6. rightttHeight = 0
 9. height = max(0, 0) + 1
 9. height = 1
 10. return 1
 5. leftHeight = 1
 6. rightHeight = treeHeight({N, 10, N})
 1. treeHeight({N, 10, N})
 2. if {N, 10, N} == null #F
 4. else
 5. leftHeight = treeHeight(null)
 1. treeHeight(null)
 2. if null == null #T
 3. return 0
 5. leftHeight = 0
 6. rightttHeight = treeHeight(null)
 1. treeHeight(null)
 2. if null == null #T
 3. return 0
 6. rightttHeight = 0
 9. height = max(0, 0) + 1
 9. height = 1
 10. return 1
 6. rightHeight = 1
 9. height = max(1, 1) + 1
 9. height = 2
 10. return 2

6. rightHeight = 2

```

```

9. height = max(2, 2) + 1
10. return 3

```

## Penambahan dan pembuatan *node*

1. Tiap *node* terdiri dari tiga segmen yaitu *left*, *data* dan *right*. *Left* berarti penunjuk ke *node* kiri, *data* berisi nilai yang disimpan, dan *right* penunjuk ke *node* kanan
2. Simpul yang penunjuk *left* dan *right*-nya tidak menunjuk ke simpul lain, disebut sebagai simpul daun (*leaf*) atau juga disebut simpul akhir.
3. Sebuah *node*, setiap kali dibuat secara *default* tersusun atas: {'left':None, 'data':data, 'right':None}
4. Aturan penambahan *node* pada *tree*:
  - a. Jika *tree* masih kosong, maka *node baru* otomatis mengikuti *node default*
  - b. Jika *tree* tidak kosong, maka bandingkan antara data pada *node baru* dengan data pada *node root*
    - i. Jika data *node baru* lebih kecil dari *data* pada *root node*, maka isi penunjuk *left* pada *root node* dengan *node baru*
    - ii. Jika data *node baru* lebih besar dari *data* pada *root node*, maka isi penunjuk *right* pada *root node* dengan *node baru*

```

1. fungsi insert(tree, data):
2. if tree == None:
3. tree = {'left':None, 'data':data, 'right':None}
4. else:
5. if data < tree["data"]:
6. if tree["left"] == None:
7. tree["left"] = create_node(data)
8. else:
9. insert(tree["left"], data)
10. else:
11. if tree["right"] == None:
12. tree["right"] = create_node(data)
13. else:
14. insert(tree["right"], data)
15. return tree
16.
17. tree = None
18. tree = insert(tree, 5)
19. tree = insert(tree, 6)
20. tree = insert(tree, 3)
21. tree = insert(tree, 8)
22. tree = insert(tree, 9)

```

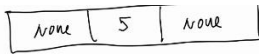
| Baris | Tahapan Eksekusi                                                                                                                  |
|-------|-----------------------------------------------------------------------------------------------------------------------------------|
| 17    | tree = None<br><div style="border: 1px solid black; display: inline-block; padding: 2px; margin: 5px;"> None   None   None </div> |
| 18    | tree = insert(None, 5)                                                                                                            |
| 1     | fungsi insert(None, 5)                                                                                                            |
| 2     | if None is None:      #True                                                                                                       |



```

3 tree = {None, 5, None}
4 else: #False
15 return {None, 5, None}
18 tree = {None, 5, None}

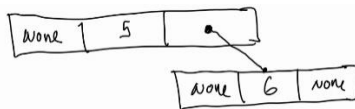
```



```

19 tree = insert({None, 5, None}, 6)
1 fungsi insert({None, 5, None}, 6)
2 if {None, 5, None} is None: #False
4 else: #True
 #tree['data'] = 5
5 if 6 < 5: #False
10 else: #True
 #tree['right'] = None
11 if None is None: #True
12 tree['right'] = {None, 6, None}
 #tree = {None, 5, {None, 6, None}}
13 else: #False
15 return {None, 5, {None, 6, None}}
19 tree = {None, 5, {None, 6, None}}

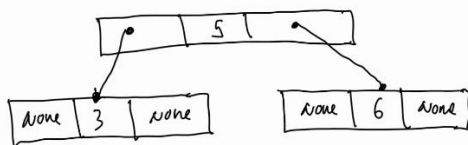
```



```

20. tree = insertNode({N,5,{N,6,N}}, 3)
1. fungsi insertNode({N,5,{N,6,N}}, 3)
2. if {N,5,{N,6,N}} == None #False
4. else #True #tree['data'] = 5
5. if 3 < 5 #True #tree['left'] = None
6. if None == None #True
7. tree['L'] = {N,3,N} #tree = {{N,3,N},5,{N,6,N}}
15 return {{N,3,N},5,{N,6,N}}
20. tree = {{N,3,N},5,{N,6,N}}

```



```

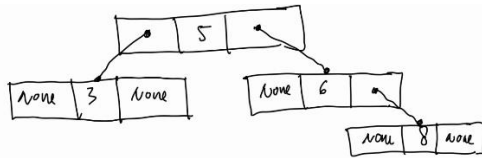
21 tree = insert({{None, 3, None}, 5, {None, 6, None}}, 8)
1 fungsi insert({{None, 3, None}, 5, {None, 6, None}}, 8)
2 If {{None, 3, None}, 5, {None, 6, None}} is None: #False
4 Else: #True
 #tree['data'] = 5
5 if 8 < 5: #False
10 Else: #True
 #tree['right'] = {None, 6, None}
11 if {None, 6, None} is None: #False
13 Else: #True
 insert({None, 6, None}, 8)
14 fungsi insert({None, 6, None}, 8)
15 if {None, 6, None} is None: #False
4 Else: #True

```

```

5 #tree['data'] = 6
 if 8 < 6: #False
10 Else: #True
 #tree['right'] = None
11 if None is None: #True
12 #tree['right'] = {None, 8, None}
 #tree['right'] = {None, 6, {None, 8, None}}
15 return {{None, 3, None}, 5, {None, 6, {None, 8, None}}}
21 tree = {{None, 3, None}, 5, {None, 6, {None, 8, None}}}

```

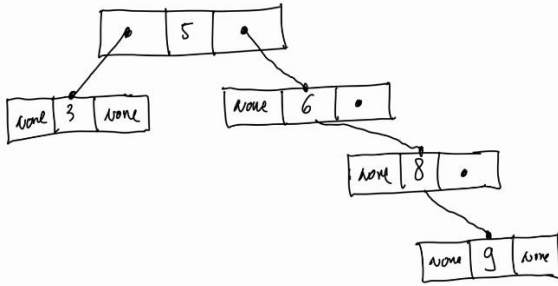


```

22 tree = insert({None, 3, None}, 5, {None, 6, {None, 8, None}}, 9)
1 fungsi insert({None, 3, None}, 5, {None, 6, {None, 8, None}}, 9)
2 if {{None, 3, None}, 5, {None, 6, {None, 8, None}}} is None: #False
4 Else: #True
 #tree['data'] = 5
5 if 9 < 5: #False
10 Else: #True
 #tree['right'] = {None, 6, {None, 8, None}}
11 if {None, 6, {None, 8, None}} is None: #False
13 Else: #True
 insert({None, 6, {None, 8, None}}, 9)
14 Fungsi insert({None, 6, {None, 8, None}}, 9)
1 if {None, 6, {None, 8, None}} is None: #False
2 Else: #True
 #tree['data'] = 6
5 if 9 < 6: #False
10 Else: #True
 #tree['right'] = {None, 8, None}
11 if {None, 8, None} is None: #False
13 Else: #True
 insert({None, 8, None}, 9)
14 Fungsi insert({None, 8, None}, 9)
1 if insert({None, 8, None} is None: #False
2 Else: #True
 #tree['data'] = 8
5 if 9 < 8: #False
10 Else: #True
 #tree['right'] = None
11 if None is None: #True
12 tree['right'] = {None, 9, None}
 #tree['right'] = {None, 8, {None, 9, None}}
 #tree['right'] = {None, 6, {None, 8, {None, 9, None}}}
15 return {{None, 3, None}, 5, {None, 6, {None, 8, {None, 9, None}}}}
22 tree = {{None, 3, None}, 5, {None, 6, {None, 8, {None, 9, None}}}}

```





## Traverse

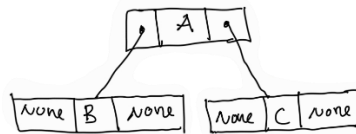
Traverse digunakan untuk menelusuri atau menjelajahi *node* pada *tree*. Kegunaannya adalah untuk mencetak atau mengambil data pada *node*.

Terdapat beberapa teknik untuk menjelajahi *node* pada *tree*.

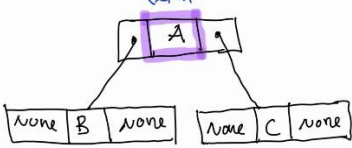
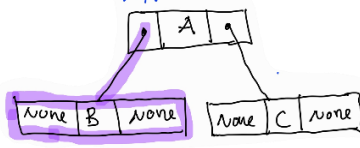
### Preorder traversal

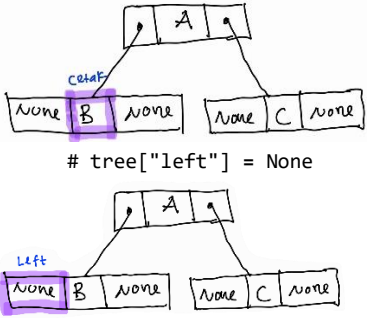
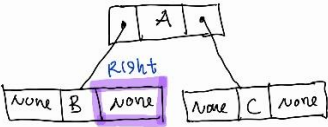
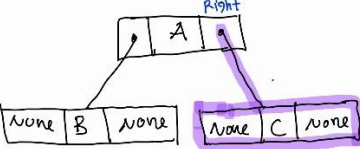
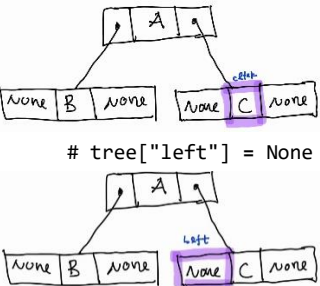
Teknik ini bekerja dengan cara menelusuri *root node*, lalu ke *node* kiri, dan berakhir ke *node* kanan. Istilah lain dari POT adalah *depth-first traversal*.

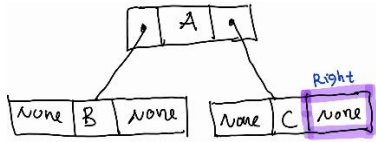
Misal: `tree = {{None, B, None}, A, {None, C, None}}`



1. fungsi `poTraversing(tree):`
2.     `if tree == None:`
3.         `return None`
4.     `print tree["data"]`
5.     `poTraversing(tree["left"])`
6.     `poTraversing(tree["right"])`

|   |                                                                                                                                                                                                                                                                                                 |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | <code>poTraversing({{None, B, None}, A, {None, C, None}})</code>                                                                                                                                                                                                                                |
|   | <code>#stack: poTraversing(A)</code>                                                                                                                                                                                                                                                            |
| 2 | <code>if {{None, B, None}, A, {None, C, None}} is None: #False</code>                                                                                                                                                                                                                           |
| 4 | <p><code>print A</code></p> <p><i>Cetak</i></p>  <p><code># tree["left"] = {None, B, None}</code></p> <p><i>Left</i></p>  |
| 5 | <code>poTraversing({None, B, None})</code>                                                                                                                                                                                                                                                      |

|   |                                                                                                                                    |
|---|------------------------------------------------------------------------------------------------------------------------------------|
|   | <b>#stack: poTraversing(A), poTraversing(B)</b>                                                                                    |
| 2 | if ({None, B, None}) is None: #False                                                                                               |
| 4 | print B<br> <pre># tree["left"] = None</pre>      |
| 5 | poTraversing(None)                                                                                                                 |
|   | <b>#stack: poTraversing(A), poTraversing(B), poTraversing(None)</b>                                                                |
| 2 | if None is None: #True                                                                                                             |
| 3 | return None<br><pre># tree["right"] = None</pre>  |
|   | <b>#stack: poTraversing(A), poTraversing(B)</b>                                                                                    |
| 6 | poTraversing(None)                                                                                                                 |
|   | <b>#stack: poTraversing(A), poTraversing(B), poTraversing(None)</b>                                                                |
| 2 | if None is None: #True                                                                                                             |
| 3 | return None                                                                                                                        |
|   | <b>#stack: poTraversing(A), poTraversing(B)</b>                                                                                    |
|   | <b>#stack: poTraversing(A)</b>                                                                                                     |
|   | <pre># tree["right"] = {None, C, None}</pre>    |
| 6 | poTraversing({None, C, None})                                                                                                      |
|   | <b>#stack: poTraversing(A), poTraversing(C)</b>                                                                                    |
| 1 | if {None, C, None} is None: #False                                                                                                 |
| 4 | print C<br> <pre># tree["left"] = None</pre>    |
| 5 | poTraversing(None)                                                                                                                 |
|   | <b>#stack: poTraversing(A), poTraversing(C), poTraversing(None)</b>                                                                |
| 2 | if None is None: #True                                                                                                             |

|   |                                                                                   |
|---|-----------------------------------------------------------------------------------|
| 3 | return None                                                                       |
|   | #stack: poTraversing(A), poTraversing(C)                                          |
|   | # tree["right"] = None                                                            |
|   |  |
| 6 | poTraversing(None)                                                                |
|   | #stack: poTraversing(A), poTraversing(C), poTraversing(None)                      |
| 2 | if None is None: #True                                                            |
| 3 | return None                                                                       |
|   | #stack: poTraversing(A), poTraversing(C)                                          |
|   | #stack: poTraversing(A)                                                           |
|   | #stack:                                                                           |

## In-order traversal

Teknik ini bekerja dengan cara menelusuri *node* kiri terlebih dahulu, lalu *node* tengah, terakhir *node* kanan.

Misalnya kita hendak mencetak seluruh data pada *node* di dalam *tree*

tree = {{None, 3, None}, 5, {None, 6, {None, 8, {None, 9, None}}}}

|    |                                  |
|----|----------------------------------|
| 1. | fungsi inorder_traversal(tree):  |
| 2. | if tree is None:                 |
| 3. | return None                      |
| 4. | inorder_traversal(tree["left"])  |
| 5. | print(tree["data"], end=' ')     |
| 6. | inorder_traversal(tree["right"]) |

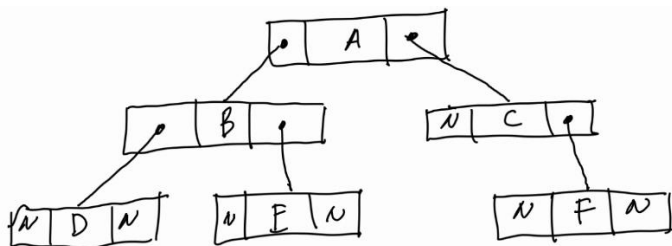
|       |                                                                                |
|-------|--------------------------------------------------------------------------------|
| Stack | [iot(5)]                                                                       |
| 1     | iot({{None, 3, None}, 5, {None, 6, {None, 8, {None, 9, None}}}}):              |
| 2.    | if {{None, 3, None}, 5, {None, 6, {None, 8, {None, 9, None}}}} is None: #False |
| 4     | iot({None, 3, None}):                                                          |
| Stack | [iot(5), iot(3)]                                                               |
| 2     | If if {None, 3, None} is None: #False                                          |
| 4     | iot(None)                                                                      |
| Stack | [iot(5), iot(3), iot(None)]                                                    |
| 2     | if None is None: #True                                                         |
| 3     | return None                                                                    |
| Stack | [iot(5), iot(3)]                                                               |
| 5     | Print(3)                                                                       |
| 6     | Iot(None)                                                                      |
| stack | [iot(5), iot(3), iot(None)]                                                    |
| 2     | if None is None: #True                                                         |
| 3     | return None                                                                    |
| Stack | [iot(5)]                                                                       |
| 5     | Print(5)                                                                       |
| 6     | Iot(6)                                                                         |
| Stack | [iot(5), iot(6)]                                                               |
| 2     | If {None, 6, {None, 8, {None, 9, None}}} is None: #False                       |

[illegible]

## Postorder traversal

Postorder traversal merupakan teknik penelusuran simpul dimulai dari kiri, ke kanan, lalu akar.

Misalnya diberikan tree berikut:  $\{\{\{n, D, n\}, B, \{n, E, n\}\}, A, \{n, C, \{n, F, n\}\}\}$



Maka hasil urutan penelusurannya adalah D, E, B, F, C, dan A

```

1. def postorderTraversal(tree):
2. if tree == None:
3. return None
4. inorder_traversal(tree["left"])
5. inorder_traversal(tree["right"])
6. print(tree["data"])

```

[illegible]

## Pencarian Node

## Pencarian pada pohon biner

Pencarian pada pohon biner bermaksud menemukan apakah kata kunci yang diberikan terdapat pada simpul di dalam pohon. Apabila kata kunci dengan data di dalam simpul sama, maka bisa diberikan pesan bahwa data yang dicari ditemukan. Sebaliknya bila kata kunci yang diberikan tidak sama, maka diberikan pesan bahwa data yang dicari tidak ditemukan.

[ Gambarkan dan jelaskan dulu dalam bentuk visual (tree) tentang tahap demi tahap terjadinya pencarian]

Secara algoritmik, pencarian dimulai dengan memeriksa apakah pohon yang akan ditelusuri kosong atau tidak. Jika kosong, maka diberikan pesan bahwa pohon sedang dalam keadaan kosong dan pencarian dihentikan. Sebaliknya, jika pohon tidak kosong, maka proses pencarian dimulai dengan memeriksa nilai pada simpul akar. Apabila, nilai pada simpul akar sama dengan kata kunci, maka diberikan pesan bahwa pencarian telah ditemukan dan pencarian dihentikan.

[Jelaskan kecepatan pencarian pohon biner dibanding array]

---

**Algoritma xx: Mencari elemen pada binary tree**

---

```
1. fungsi binarySearchTree(tree, searchKey)
2.
3. if tree == None:
4. cetak(searchKey, 'tidak ditemukan')
5. return
6.
7. if tree['V'] == searchKey:
8. cetak(searchKey, 'ditemukan.')
9. return
10.
11. if searchKey < tree['V']:
12. binarySearchTree(tree['L'], searchKey)
13.
14. if searchKey > tree['V']:
15. binarySearchTree(tree['R'], searchKey)
```

---

tree = {{{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}}

searchKey = 2  
binarySearchTree(tree, searchKey)

---

**Eksekusi Algoritma xx: Mencari elemen pada binary tree – Elemen ditemukan**

---

```
searchKey = 2
binarySearchTree({{{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}}, 2)

1. fungsi binarySearchTree({{{N,2,N},3,N}, 5, {{N, 7, N}, 9, {N, 10, N}}}, 2)
3. if {{{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}} == None #False #tree['V'] = 5
7. if 5 == 2 #False
11. if 2 < 5 #True #tree['L'] = {{N, 2, N}, 3, N}
12. binarySearchTree({{N, 2, N}, 3, N}, 2)

 1. fungsi binarySearchTree({{N, 2, N}, 3, N}, 2)
 3. if {{N, 2, N}, 3, N} == None #False #tree['V'] = 3
 7. if 3 == 2 #False
 11. if 2 < 3 #True #tree['L'] = {N, 2, N}
 12. binarySearchTree({N, 2, N}, 2)

 1. fungsi binarySearchTree({N, 2, N}, 2)
 3. if {N, 2, N} == None #False #tree['V'] = 2
 7. if 2 == 2 #True
 8. print 2 ditemukan
 9. return
```

---

---

**Eksekusi Algoritma xx: Mencari elemen pada binary tree – Elemen tidak ditemukan**

---

```
searchKey = 1
binarySearchTree({{{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}}, 1)

1. fungsi binarySearchTree({{{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}}, 1)
```

```

3. if {{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}} == None #False #tree['V'] = 5
7. if 5 == 1 #False
11. if 1 < 5 #True #tree['L'] = {{N, 2, N}, 3, N}
9. binarySearchTree({N, 2, N}, 3, N}, 1)

 1. fungsi binarySearchTree({N, 2, N}, 3, N}, 1)
 3. If {{N, 2, N}, 3, N} == None #False #tree['V'] = 3
 7. If 3 == 1 #False
 11. If 1 < 3 #True #tree['L'] = {N, 2, N}
 12. binarySearchTree({N, 2, N}, 1)

 1. fungsi binarySearchTree({N, 2, N}, 1)
 3. if {N, 2, N} == None #False # tree['V'] = 2
 7. If 2 == 1 #False
 11. If 1 < 2 #True # tree['L'] = None
 12. binarySearchTree(None, 1)

 1. Fungsi binarySearchTree(None, 1)
 3. if None = None #True
 4. print "1 tidak ditemukan"
 5. return

```

---

Keysearch = 9

| Baris | Eksekusi                                                                              |
|-------|---------------------------------------------------------------------------------------|
| 13    | searchKey = 9                                                                         |
| 14    | <b>binarySearchTree({{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}}, 9)</b>         |
| 1     | <b>fungsi binarySearchTree({{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}}, 9):</b> |
| 2     | If {{N, 2, N}, 3, N}, 5, {{N, 7, N}, 9, {N, 10, N}}} == None #False                   |
| 5     | If 5 == 9 #False                                                                      |
| 8     | If 9 < 5 #False                                                                       |
| 10    | If 9 > 5 #True #tree['right'] = {{N, 7, N}, 9, {N, 10, N}}                            |
| 11    | <b>binarySearchTree({N, 7, N}, 9, {N, 10, N}}, 9)</b>                                 |
| 1     | Fungsi binarySearchTree({N, 7, N}, 9, {N, 10, N}}, 9)                                 |
| 2     | If {{N, 7, N}, 9, {N, 10, N}} == None #False                                          |
| 5     | If 9 == 9 #True                                                                       |
| 6     | Print 9 ditemukan                                                                     |
| 7     | return                                                                                |

## Mencari nilai terbesar

Mencari nilai terbesar berarti menelusuri pohon hingga ke simpul paling kanan. Mengapa paling kanan? Karena sesuai prinsip pohon biner, bahwa simpul yang kanan bernilai lebih besar dari simpul akar atau di atasnya.

Pencarian dilakukan dengan menelusuri tiap simpul secara rekursif hingga penunjuk sebelah kanan simpul bernilai None.

---

### Algoritma xx: Mencari elemen terbesar pada tree

---

```

1. fungsi findLargestElement(tree)
2.
3. if tree == None
4. return None
5.
6. while tree['R'] != None
7. tree = tree['R']

```

---

---

```
8.
9. return tree['V']
```

---

## Mencari nilai terkecil

---

### Algoritma xx: Mencari elemen terkecil pada tree

---

```
1. fungsi findSmallestElement(tree)
2.
3. if tree == None
4. return None
5.
6. while tree['L'] != None
7. tree = tree['L']
8.
9. return tree['V']
```

---

## Penghapusan Simpul

Menghapus berarti menghilangkan suatu simpul dalam pohon. Ada tiga kondisi simpul pada *tree* yang hendak dihapus.

1. Penghapusan simpul tanpa subsimpul.
2. Penghapusan simpul dengan satu subsimpul. Artinya, simpul yang hendak dihapus memiliki sub simpul di salah satu penunjuknya, baik itu sebelah kiri atau sebelah kanan.
3. Penghapusan simpul dengan dua subsimpul.

---

### Algoritma xx: Menghapus simpul pada *binary tree*

---

```
1. fungsi delNode(tree, searchKey)
2. if tree == None
3. return None
4.
5. if tree['V'] == searchKey
6. if tree['L'] == None and tree['R'] == None
7. return None # Hapus node daun
8. else if tree['L'] != None and tree['R'] == None
9. return tree['L'] # Hapus node dengan satu sub node di kiri
10. else if tree['R'] != None and tree['L'] == None
11. return tree['R'] # Hapus node dengan satu sub node di kanan
12. else
13. # Hapus node dengan dua sub node
14. successor = findMinValNode(tree['R'])
15. tree['V'] = successor['V']
16. tree['R'] = delNode(tree['R'], successor['V'])
17. else
18. tree['L'] = delNode(tree['L'], searchKey)
19. tree['R'] = delNode(tree['R'], searchKey)
20.
21. return tree
22.
23. fungsi findMinDNode(tree)
24. current = tree
25. while current['L'] != None
26. current = current['L']
27. return current
```

---



---

**Eksekusi Algoritma xx: Menghapus *leaf node* pada *tree* – tidak memiliki sub-*node***

---

Sdfasdfsdfasdfsdf  
asdf

---

---

**Eksekusi Algoritma xx: Menghapus *node* pada *tree* – memiliki satu sub-*node* di kiri atau kanan**

---

Sdfasdfsdfasdfsdf  
asdf

---

---

**Eksekusi Algoritma xx: Menghapus *node* pada *tree* – memiliki dua sub-*node***

---

```
tree = {{N, 3, N}, 5, {N, 7, N}}, 10, {N, 15, {N, 20, N}}
searcyKey = 5
tree = delNode({{N, 3, N}, 5, {N, 7, N}}, 10, {N, 15, {N, 20, N}}, 5)

1. fungsi delNode({{N, 3, N}, 5, {N, 7, N}}, 10, {N, 15, {N, 20, N}}, 5)
2. if {{{N, 3, N}, 5, {N, 7, N}}, 10, {N, 15, {N, 20, N}}} == None #F #tree['V'] = 10
5. if 10 == 5 #F
17. else #tree['L'] = {{N, 3, N}, 5, {N, 7, N}}
18. tree['L'] = delNode({{N, 3, N}, 5, {N, 7, N}}, 5)

 1. fungsi delNode({{N, 3, N}, 5, {N, 7, N}}, 5)
 2. if {{{N, 3, N}, 5, {N, 7, N}}} == None #F #tree['V'] = 5
 5. if 5 == 5 #T #tree['L'] = {N, 3, N} #tree['R'] = {N, 7, N}
 6. if {N, 3, N} == None and {N, 7, N} == None #F
 8. else if {N, 3, N} != None and {N, 7, N} == None #F and #F => #F
 10. else if {N, 7, N} != None and {N, 3, N} #F and #F => #F
 12. else
 14. successor = findMinNodeVal({N,7,N})

 23. findMinNodeVal({N,7,N})
 24. current = {N,7,N} #current['L'] = None
 25. while None != None #F
 27. return {N,7,N}

 14. successor = {N, 7, N} #successor['V'] = 7
 15. tree['V'] = 7 #tree = {{N, 3, N}, 7, {N, 7, N}} #tree['R'] = {N, 7, N}
 16. tree['R'] = delNode({N, 7, N}, 7)

 1. fungsi delNode({N,7,N}, 7)
 2. if {N,7,N} == None #F
 5. if 7 == 7 #T #tree['L'] = N #tree['R'] = N
 6. if N == N and N == N #T
 7. return None

 16. tree['R'] = None #tree = {{N, 3, N}, 7, N}
 21. return {{N, 3, N}, 7, N}

18. tree['L'] = {{N, 3, N}, 7, N} #tree['R'] = {N, 15, {N, 20, N}}
19. tree['R'] = delTree({N, 15, {N, 20, N}}, 5)

 1. fungsi delNode({N, 15, {N, 20, N}}, 5)
 2. if {N, 15, {N, 20, N}} == None #F #tree['V'] = 15
 5. if 15 == 5 #F
 17. else #tree['L'] = None
 18. tree['L'] = delNode(N, 5)
 1. fungsi delNode(N, 5)
 2. if None == None #T
 3. return None
 18. tree['L'] = None #tree['R'] = {N, 20, N}
```

```

19. tree['R'] = delTree({N, 20, N}, 5)
 1. fungsi delNode({N, 20, N}, 5)
 2. if {N, 20, N} == None #F
 5. if 20 == 5 #F
 17. else #tree['L'] = None #tree['R'] = None
 18. tree['L'] = delNode(None, 5)
 1. fungsi delNode(None, 5)
 2. if None == None #T
 3. return None
 18. tree['L'] = None
 19. tree['R'] = delNode(None, 5)
 1. fungsi delNode(None, 5)
 2. if None == None #T
 3. return None
 19. tree['R'] = None
 21. return {N, 20, N}

 19. tree['R'] = {N, 20, N}
19. tree['R'] = {N, 15, {N, 20, N}}
21. return {{{N, 3, N}, 7, N}, 10, {N, 15, {N, 20, N}}}

tree = {{{N, 3, N}, 7, N}, 10, {N, 15, {N, 20, N}}}

```

---

## **Latihan**

## **Bibliografi**

# AVL Tree

Pada binary tree, prinsip yang digunakan adalah selama sebelah kiri lebih kecil dari akar, dan sebaliknya berlaku di sebelah kanan. Dampaknya, node akar selalu tetap tak tergantikan, sehingga sebuah tree akan sangat mungkin lebih berat atau panjang di sebelah kiri saja atau sebelah kanan saja. Dengan kata lain pohonnya menjadi tidak seimbang. Dengan tree seperti ini bila kita hendak mencari node xx seperti contoh tree berikut, maka kompleksitasnya menjadi  $O(xx)$ .

Hal inilah yang menjadi perhatian matematikawan Rusia yaitu Adelson dkk. Mereka memperbaiki ketidakefisienan tree selama ini dengan mengusulkan AVL tree di tahun 1962 (Adelson et al., 1962).

AVL tree atau tree seimbang.

Apa perbedaan binary tree dengan AVL tree?

Mengapa dan kapan harus menggunakan AVL tree?

---

## Algoritma xx: Membuat AVL tree

---

```
1. fungsi height(tree)
2. if tree == None
3. return 0
4. return tree['H']
5.
6. fungsi updateHeight(tree)
7. tree['H'] = 1 + max(height(tree['L']), height(tree['R']))
8.
9. fungsi balanceFactor(tree)
10. return height(tree['L']) - height([tree['R']])
11.
12. fungsi rotateRight(treeY)
13. treeX = treeY['L']
14. T2 = treeX['R']
15.
16. treeX['R'] = treeY
17. treeY['L'] = T2
18.
19. updateHeight(treeY)
20. updateHeight(treeX)
21.
22. Return treeX
23.
24. fungsi rotateLeft(treeX)
25. treeY = treeX['R']
26. T2 = treeY['L']
27.
28. treeY['L'] = treeX
29. treeX['R'] = T2
30.
31. updateHeight(treeX)
32. updateHeight(treeY)
33.
34. return treeY
35.
36. fungsi insert(tree, value)
37.
38. if tree == None
39. return {N,value,N,1} #left, value, right, height
40.
41. if value < tree['V']
42. tree['L'] = insert(tree['L'], value)
43. else
```

---

---

```

44. tree['R'] = insert(tree['R'], value)
45.
46. updateHeight(tree)
47.
48. balance = balanceFactor(tree)
49.
50. # Left Left Case
51. if (balance > 1) and (value < tree['L']['V'])
52. return rotateRight(tree)
53.
54. # Right Right Case
55. if (balance < -1) and (value > tree['R']['V'])
56. return rotateLeft(tree)
57.
58. # Left Right Case
59. if (balance > 1) and (value > tree['L']['V'])
60. tree['L'] = rotateLeft(tree['L'])
61. return rotateRight(tree)
62.
63. # Right Left Case
64. if (balance < -1) and (value < tree['R']['V'])
65. tree['R'] = rotateRight(tree['R'])
66. return rotateLeft(tree)
67.
68. return tree

```

---



---

### Eksekusi Algoritma xx: Membuat AVL tree

---

```

36. fungsi insert(None, 20)
38. if None == None #True
39. return {N,20,N,1}

```



```

36. fungsi insert({N,20,N,1}, 11)
41. if 11 < 20
42. tree['L'] =
25. fungsi insert(None, 11)
26. if None == None #True
27. return {'L': None, 'V': 11, 'R': None, 'H': 1}
29. tree['L'] = insert({'L': None, 'V': 11, 'R': None, 'H': 1}, 11)
32. before updateHeight({'L': {'L': None, 'V': 11, 'R': None, 'H': 1}, 'V': 20, 'R': None, 'H':
1})
5. fungsi updateHeight({'L': {'L': None, 'V': 11, 'R': None, 'H': 1}, 'V': 20, 'R': None, 'H':
1})
 1. fungsi height({'L': None, 'V': 11, 'R': None, 'H': 1}) ke-1
 2. if {'L': None, 'V': 11, 'R': None, 'H': 1} != None: #False
 4. return 1
 1. fungsi height(None) ke-2
 2. if None == None: #True
 3. return 0
 1. fungsi height({'L': None, 'V': 11, 'R': None, 'H': 1}) ke-3
 2. if {'L': None, 'V': 11, 'R': None, 'H': 1} != None: #False
 4. return 1
 1. fungsi height(None) ke-4
 2. if None == None: #True
 3. return 0
 6. 2 = 1 + max(1, 0)
 32. after updateHeight({'L': {'L': None, 'V': 11, 'R': None, 'H': 1}, 'V': 20, 'R': None, 'H':
2})
33. balance = balanceBactor({'L': {'L': None, 'V': 11, 'R': None, 'H': 1}, 'V': 20, 'R': None,
'H':
2})
7. fungsi balanceBactor({'L': {'L': None, 'V': 11, 'R': None, 'H': 1}, 'V': 20, 'R': None, 'H':
2})

```

---

```

1. fungsi height({'L': None, 'V': 11, 'R': None, 'H': 1}) ke-5
2. if {'L': None, 'V': 11, 'R': None, 'H': 1} != None: #False
4. return 1
 1. fungsi height(None) ke-6
2. if None == None: #True
3. return 0
8. return 1 - 0
 1. fungsi height({'L': None, 'V': 11, 'R': None, 'H': 1}) ke-7
2. if {'L': None, 'V': 11, 'R': None, 'H': 1} != None: #False
4. return 1
 1. fungsi height(None) ke-8
2. if None == None: #True
3. return 0
33. balance = 1

36. fungsi insert({'L': None, 'V': 20, 'R': None, 'H': 1}, 11)
38. if {N,20,N,1} == None #False
41. if 11 < 20 #True
42. tree['L'] = insert(None, 11)
 36. fungsi insert(None, 11)
 38. if None == None #True
 39. return {'L': None, 'V': 11, 'R': None, 'H': 1}
42. tree['L'] = {N,11,N,1} #tree = {{N,11,N,1},20,N,1}

 20
 / \
 11 N
 / \
 N N

46. updateHeight({{N,11,N,1},20,N,1})
 6. fungsi updateHeight({{N,11,N,1},20,N,1})
 7. tree['H'] = 1 + max(height({N,11,N,1}), height(N))
 1. fungsi height({N,11,N,1})
 2. if {N,11,N,1} != None: #False
 4. return 1

 1. fungsi height(None)
 2. if None == None: #True
 3. return 0

 7. tree['H'] = 1 + max(1, 0) = 2 #tree = {{N,11,N,1},20,N,2}

48. balance = balanceFactor({{N,11,N,1},20,N,2})
 9. fungsi balanceBactor({{N,11,N,1},20,N,2})
 10. return height({N,11,N,1}) - height(None)
 1. fungsi height({N,11,N,1})
 2. if {N,11,N,1} != None: #False
 4. return 1

 1. fungsi height(None)
 2. if None == None: #True
 3. return 0

 10. return 1 - 0 #1
48. balance = 1

51. if (1 > 1) && #False
55. if (1 < -1) && #False

```

---

```

59. if (1 > 1) && #False
64. if (1 < -1) && #False

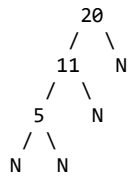
68. return {{N,11,N,1},20,N,2}

```

```

36. fungsi insert({{N,11,N,1},20,N,2}, 5)
38. if {{N,11,N,1},20,N,2} == None #False
41. if 5 < 20 #True
42. tree['L'] = insert({N,11,N,1}, 5)
 36. fungsi insert({N,11,N,1}, 5)
 38. if {N,11,N,1} == None #False
 41. if 5 < 11 #true
 42. tree['L'] = insert(None, 5)
 36. fungsi insert(None, 5)
 38. if None == None #True
 39. return {N,5,N,1}
 42. tree['L'] = {N,5,N,1} #tree['L'] = {{N,5,N,1},11,N,1}

```



```

46. updateHeight({{N,5,N,1},11,N,1})
 6. fungsi updateHeight({{N,5,N,1},11,N,1})
 7. tree['H'] = 1 + max(height({N,5,N,1}, height(N))
 1. fungsi height({N,5,N,1})
 2. if {N,5,N,1} != None: #False
 4. return 1

 1. fungsi height(None)
 2. if None == None: #True
 3. return 0

 7. tree['H'] = 1 + max(1, 0) = 2 #tree = {{N,5,N,1},11,N,2}

```

```

48. balance = balanceFactor({{N,5,N,1},11,N,2})
 9. fungsi balanceFactor({{N,5,N,1},11,N,2})
 10. return height({N,5,N,1}) - height(N)
 1. fungsi height({N,5,N,1})
 2. if {N,5,N,1} != None: #False
 4. return 1

 1. fungsi height(None)
 2. if None == None: #True
 3. return 0

 10. return 1 - 0

```

```

48. balance = 1
51. if (1 > 1) #False
55. if (1 < -1) #False
59. if (1 > 1) #False
64. if (1 < -1) #False
68. return {{N,5,N,1},11,N,2} #tree = {{N,5,N,1},11,N,2},20,N,2}

```

```

46. updateHeight({{N,5,N,1},11,N,2},20,N,2})
 6. fungsi updateHeight({{N,5,N,1},11,N,2},20,N,2})
 7. tree['H'] = 1 + max(height({N,5,N,1},11,N,2), height(N))

 1. fungsi height({'L': None, 'V': 5, 'R': None, 'H': 1})
 2. if {'L': None, 'V': 5, 'R': None,
'H': 1} != None: #False

 4. return 1

 1. fungsi height(None)
 2. if None == None: #True
 3. return 0

```



---

```

33. balance = 1
29. tree['L'] = insert({'L': {'L': None, 'V': 5, 'R': None, 'H': 1},
'V': 11, 'R': None, 'H': 2}, 5)
32. before updateHeight({'L': {'L': {'L': None, 'V': 5, 'R': None, 'H':
1}, 'V': 11, 'R': None, 'H': 2}, 'V': 20, 'R': None, 'H': 2})
5. fungsi updateHeight({'L': {'L': {'L': None, 'V': 5,
'R': None, 'H': 1}, 'V': 11, 'R': None, 'H': 2}, 'V': 20, 'R': None, 'H': 2})
1. fungsi height({'L': {'L': None, 'V':
5, 'R': None, 'H': 1}, 'V': 11, 'R': None, 'H': 2})
None, 'H': 1}, 'V': 11, 'R': None, 'H': 2} != None: #False
5, 'R': None, 'H': 1}, 'V': 11, 'R': None, 'H': 2})
None, 'H': 1}, 'V': 11, 'R': None, 'H': 2} != None: #False
6. 3 = 1 + max(2, 0)
32. after updateHeight({'L': {'L': {'L': None, 'V': 5, 'R': None, 'H':
1}, 'V': 11, 'R': None, 'H': 2}, 'V': 20, 'R': None, 'H': 3})
33. balance = balanceBactor({'L': {'L': {'L': None, 'V': 5, 'R': None,
'H': 1}, 'V': 11, 'R': None, 'H': 2}, 'V': 20, 'R': None, 'H': 3})
7. fungsi balanceBactor({'L': {'L': {'L': None, 'V':
5, 'R': None, 'H': 1}, 'V': 11, 'R': None, 'H': 2}, 'V': 20, 'R': None, 'H': 3})
5, 'R': None, 'H': 1}, 'V': 11, 'R': None, 'H': 2})
None, 'H': 1}, 'V': 11, 'R': None, 'H': 2} != None: #False
8. return 2 - 0
5, 'R': None, 'H': 1}, 'V': 11, 'R': None, 'H': 2})
None, 'H': 1}, 'V': 11, 'R': None, 'H': 2} != None: #False
33. balance = 2
34. if (2 > 1) and (5 < 11)
35. return rotateRight({'L': {'L': {'L': None, 'V': 5, 'R': None,
'H': 1}, 'V': 11, 'R': None, 'H': 2}, 'V': 20, 'R': None, 'H': 3})
9. fungsi rotateRight({'L': {'L': {'L': None, 'V': 5,
'R': None, 'H': 1}, 'V': 11, 'R': None, 'H': 2}, 'V': 20, 'R': None, 'H': 3})
10. treeX = {'L': {'L': None, 'V': 5, 'R': None,
'H': 1}, 'V': 11, 'R':

```

# Heap

## Antrian Berprioritas

Seperti yang sudah disinggung pada Bab xx mengenai antrian, antrian berprioritas adalah salah satu varian antrian yang khusus dibuat untuk menangani *node* yang memiliki prioritas tertinggi terlebih dahulu. Dengan kata lain, jika pada antrian biasa dikenal dengan FIFO, maka pada antrian berprioritas ini dikenal dengan Highest In First Out atau HIFO.

Mengeimplementasikan antrian berprioritas bisa menggunakan struktur data *array*, *linked list*, *heap*, dan *binary search tree*. Namun, berdasarkan (Deo & Prasad, 1992), *heap* adalah struktur data yang paling efisien untuk menerapkan antrian berprioritas.

(sertakan perbandingan Big O masing-masing)

### Prinsip Binary Heap

Ketika membentuk max heap, maka simpul akar berisi data terbesar. Sebaliknya ketika membentuk min heap, maka simpul akar berisi data terkecil.

Binary heap merupakan bentuk *binary tree* lengkap. Artinya, di tiap level akan selalu imbang berisi dua simpul di kiri dan kanan, kecuali level terakhir.

Untuk penambahan simpul baru, ada dua prinsip utama yang dilakukan yaitu:

1. Letakkan simpul baru di level akhir pohon untuk menyempurnakan *binary tree* lengkap
2. Perbarui simpul agar menjadi max atau min binary heap

Representasi binary heap adalah berupa array. Posisi induk didefinisikan dengan fungsi  $\text{floor} \left[ \frac{\text{indek}-1}{2} \right]$ ,  
posisi kiri  $2 \times \text{indek} + 1$

## Pembuatan binary max heap

---

### Algoritma xx: Membuat *binary heap*

---

```
1. fungsi parent(idx)
2. return $\left\lfloor \frac{\text{idx}-1}{2} \right\rfloor$
3.
4. fungsi leftChild(idx)
5. return 2 * idx + 1
6.
7. fungsi rightChild(idx)
8. return 2 * idx + 2
9.
10. fungsi swap(heap, idx, j)
11. temp = 0
12. temp = heap[idx]
13. heap[idx] = heap[j]
```

---

---

```

14. heap[j] = temp
15.
16. fungsi heapifyDown(heap, idx)
17. size = |heap|
18. left = leftChild(idx)
19. right = rightChild(idx)
20. largest = idx
21.
22. if left < size and heap[left] > heap[largest]
23. largest = left
24. if right < size and heap[right] > heap[largest]
25. largest = right
26. if largest != idx
27. swap(heap, idx, largest)
28. heapifyDown(heap, largest)
29.
30. fungsi heapifyUp(heap, idx)
31. while (idx > 0) and (heap[idx] > heap[parent(idx)])
32. parentIdx = parent(idx)
33. swap(heap, idx, parentIdx)
34. idx = parentIdx
35.
36. fungsi insert(heap, value)
37. heap[] = value
38. heapifyUp(heap, |heap| - 1)
39.
40. fungsi extractMax(heap)
41. if len(heap) == 0
42. cetak("Heap is empty. Cannot extract maximum element.")
43. return None
44.
45. maxValue = heap[0]
46. heap[0] = heap[-1] # -1 adalah indeks terakhir pada array
47. pop(heap) # menghapus elemen terakhir di array heap
48. heapifyDown(heap, 0)
49. return maxValue
50.
51. fungsi printHeap(heap)
52. cetak("Heap elements:", end=" ")
53.
54. for (i=0; i < size(heap); i++)
55. cetak(heap[i])

```

---



---

#### Eksekusi Algoritma xx: Insert simpul ke *binary heap*

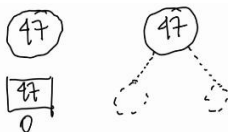
---

```

heap = []
insert(heap, 47)

36. fungsi insert([], 47)
37. heap[] = 47 #heap = [47] #|heap| = 1
38. heapifyUp([47], 0)
39. 30. fungsi heapifyUp([47], 0) #heap[0] = 47
40. 31. while 0 > 0 and 47 > heap[parent(0)]
41. 1. fungsi parent(0)
42. 2. return -1
43. 31. while (0 > 0) #False. Sisanya tidak perlu dieksekusi
heap = [47]

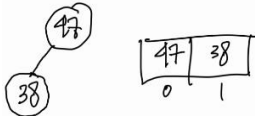
```



---

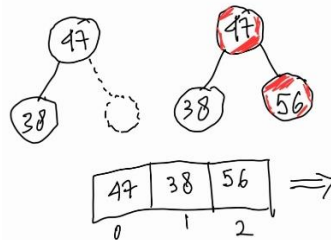
**insert(heap, 38)**

```
36. fungsi insert([47], 38)
37. heap[] = 38 #|heap| = 2 #heap[47, 38]
38. heapifyUp([47, 38], 1)
 30. fungsi heapifyUp([47, 38], 1) #heap[1] = 38
 31. while 1 > 0 and 38 > heap[parent(1)]
 1. fungsi parent(1)
 2. return 0
 29. while 1 > 0 and 38 > 47 #False
heap = [47, 38]
```

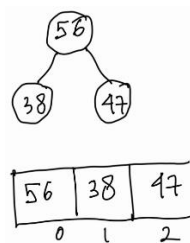


**insert(heap, 56)**

```
36. fungsi insert([47, 38], 56)
37. heap[] = 56 #heap[47, 38, 56] #|heap| = 3
```



```
38. heapifyUp([47, 38, 56], 2)
 30. fungsi heapifyUp([47, 38, 56], 2) #heap[2] = 56
 31. while 2 > 0 and 56 > heap[parent(2)]
 1. fungsi parent(2)
 2. return 0 #heap[0] = 47
 31. while 2 > 0 and 56 > 47 #True
 32. parentIdx = 0
 33. swap([47, 38, 56], 2, 0)
 10. fungsi swap([47, 38, 56], 2, 0) #heap[2] = 56 #heap[0] = 47
 11. temp = 0
 12. temp = 56
 13. heap[2] = 47 #heap = [47, 38, 47]
 14. heap[0] = 56 #heap = [56, 38, 47] #heap[0] = 56 #heap[2] = 47
 34. idx = 0
 31. while 0 > 0 #False. Sisanya tidak perlu dieksekusi
heap = [56, 38, 47]
```



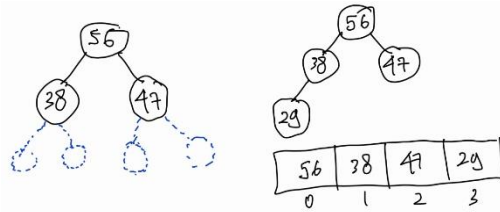
**insert(heap, 29)**

```
36. fungsi insert([56, 38, 47], 29)
```

---

---

37. heap[] = 29 #heap = [56, 38, 47, 29] #|heap| = 4

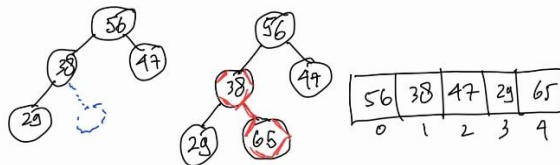


38. heapifyUp([56, 38, 47, 29], 3)  
 30. fungsi heapifyUp([56, 38, 47, 29], 3) #heap[3] = 29  
 31. while (3 > 0) and (29 > heap[parent(3)])  
     1. fungsi parent(3)  
     2. return 1 #heap[1] = 38  
 31. while (3 > 0) and (29 > 38) #False  
 heap = [56, 38, 47, 29]

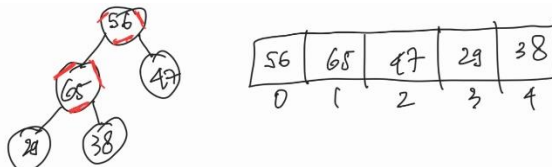
insert(heap, 65)

36. fungsi insert([56, 38, 47, 29], 65)

37. heap[] = 65 #heap = [56, 38, 47, 29, 65] #|heap| = 5



38. heapifyUp([56, 38, 47, 29, 65], 4)  
 30. heapifyUp([56, 38, 47, 29, 65], 4) #heap[4] = 65  
 31. while 4 > 0 and 65 > heap[parent(4)]  
     1. fungsi parent(4)  
     2. return 1  
 31. while (4 > 0) and (65 > 38) #True  
 32. parentIdx = 1  
 33. swap([56, 38, 47, 29, 65], 4, 1)  
     10. fungsi swap([56, 38, 47, 29, 65], 4, 1)  
     11. temp = 0 #heap[4] = 65  
     12. temp = 65 #heap[1] = 38  
     13. heap[4] = 38 #heap = [56, 38, 47, 29, 38]  
     14. heap[1] = 65 #heap = [56, 65, 47, 29, 38]

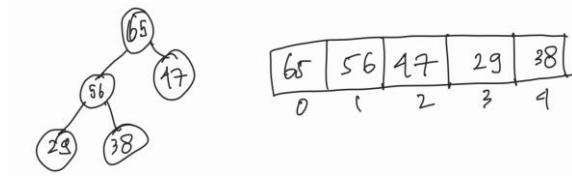


34. idx = 1 #heap[1] = 65  
 31. while (1 > 0) and (65 > heap[parent(1)])  
     1. fungsi parent(1)  
     2. return 0 #heap[0] = 56  
 31. while (1 > 0) and (65 > 56) #True  
 32. parentIdx = 0  
 33. swap([56, 65, 47, 29, 38], 1, 0)  
     10. fungsi swap([56, 65, 47, 29, 38], 1, 0)  
     11. temp = 0 #heap[1] = 65  
     12. temp = 65 #heap[0] = 56  
     13. heap[1] = 56 #heap = [56, 56, 47, 29, 38]  
     14. heap[0] = 65 #heap = [65, 56, 47, 29, 38]  
 34. idx = 0 #heap[0] = 65  
 31. while (0 > 0) #False. Sisanya tidak perlu dieksekusi

---

---

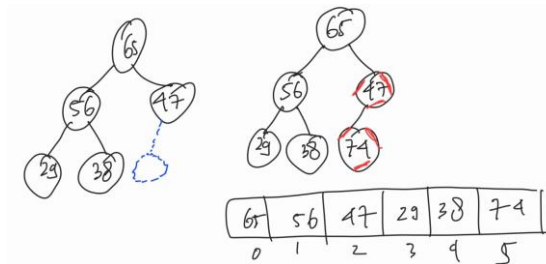
heap = [65, 56, 47, 29, 38]



insert(heap, 74)

36. fungsi insert([65, 56, 47, 29, 38], 74)

37. heap[] = 74 #heap = [65, 56, 47, 29, 38, 74] #|heap| = 6



38. heapifyUp([65, 56, 47, 29, 38, 74], 5)

30. fungsi heapifyUp([65, 56, 47, 29, 38, 74], 5) #heap[5] = 74

31. while (5 > 0) and (74 > heap[parent(5)])

1. fungsi parent(5)

2. return 2 #heap[2] = 47

31. while (5 > 0) and (74 > 47) #True

32. parentIdx = 2

33. swap([65, 56, 47, 29, 38, 74], 5, 2)

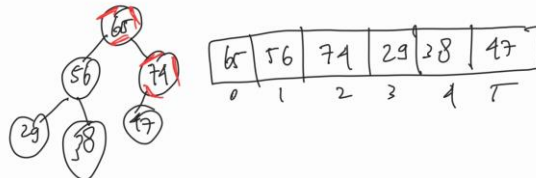
10. fungsi swap([65, 56, 47, 29, 38, 74], 5, 2)

11. temp = 0 #heap[5] = 74

12. temp = 74 #heap[2] = 47

13. heap[5] = 47 #heap = [65, 56, 47, 29, 38, 47]

14. heap[2] = 74 #heap = [65, 56, 74, 29, 38, 47]



34. idx = 2 #heap[2] = 74

31. while (2 > 0) and (74 > heap[parent(2)])

1. fungsi parent(2)

2. return 0 #heap[0] = 65

31. while (2 > 0) and (74 > 65) #True

32. parentIdx = 0

33. swap([65, 56, 74, 29, 38, 47], 2, 0)

10. fungsi swap([65, 56, 74, 29, 38, 47], 2, 0)

11. temp = 0 #heap[2] = 74

12. temp = 74 #heap[0] = 65

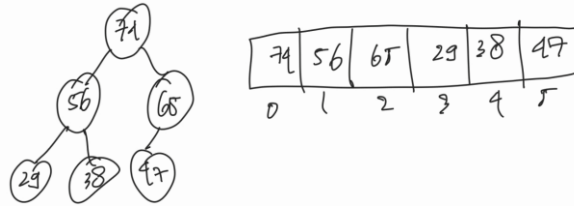
13. heap[2] = 65 #heap = [65, 56, 65, 29, 38, 47]

14. heap[0] = 74 #heap = [74, 56, 65, 29, 38, 47]

34. idx = 0

31. while (0 > 0) #False. Sisanya tidak perlu dieksekusi

---



## Pencarian node pada max binary heap

## Pencarian node pada min binary heap

## Menghapus node pada max binary heap

Menghapus node pada max maupun min heap berarti menghapus simpul akar (*root*). Konsekuensinya, node terbesar ..... akan naik menjadi simpul akar yang baru

---

### Algoritma xx: Hapus binary heap

---

```

1. fungsi heapifyDown(heap, idx)
2.
3. leftChildIdx = 2 * idx + 1
4. rightChildIdx = 2 * idx + 2
5. largestIdx = idx
6. heapSize = |heap|
7.
8. if (leftChildIdx < heapSize) and (heap[leftChildIdx] > heap[largestIdx])
9. largestIdx = leftChildIdx
10.
11. if (rightChildIdx < heapSize) and (heap[rightChildIdx] > heap[largestIdx])
12. largestIdx = rightChildIdx
13.
14. if largestIdx != idx
15. temp = None
16. temp = heap[idx]
17. heap[idx] = heap[largestIdx]
18. heap[largestIdx] = temp
19. heapifyDown(heap, largestIdx)
20.
21. fungsi deleteHeap(heap)
22.
23. if heap == None
24. return None
25.
26. lastElement = pop(heap)
27.
28. if |heap| > 0
29. heap[0] = lastElement
30. heapifyDown(heap, 0)
31.
32. return heap

33. # Example usage:
34. heap = [100, 33, 42, 10, 14, 35]
35. print("Heap before pop:", heap)

36. popped_value = heap_pop(heap)

```

---

---

```
37. print("Popped value:", popped_value)
38. print("Heap after pop:", heap)
```

---

---

#### Eksekusi algoritma xx: Hapus binary heap

---

```
heap = [100, 33, 42, 10, 14, 35]
deleteHeap(heap)

21. fungsi deleteHeap([100, 33, 42, 10, 14, 35])
23. if [100, 33, 42, 10, 14, 35] == None #False #heap[0] = 100
26. root = 100
27. lastElement = pop([100, 33, 42, 10, 14, 35]) #heap = [100, 33, 42, 10, 14]
27. lastElement = 35 #|heap| = 5
29. if 5 > 0 #True
30. heap[0] = 35 #heap = [35, 33, 42, 10, 14]
31. heapifyDown([35, 33, 42, 10, 14], 0)
 1. fungsi heapifyDown([35, 33, 42, 10, 14], 0)
 3. leftChildIdx = 1 #heap[1] = 33
 4. rightChildIdx = 2 #heap[2] = 42
 5. largestIdx = 0 #heap[0] = 35
 6. heapSize = 5
 8. if (1 < 5) and (33 > 35) #False
 11. if (2 < 5) and (42 > 35) #True
 12. largestIdx = 2
 14. if 2 != 0 #True #heap[0] = 35
 15. temp = None
 16. temp = 35 #heap[2] = 42
 17. heap[0] = 42 #heap = [42, 33, 42, 10, 14]
 18. heap[2] = 35 #heap = [42, 33, 35, 10, 14]
 19. heapifyDown([42, 33, 35, 10, 14], 2)
 1. fungsi heapifyDown([42, 33, 35, 10, 14], 2)
 3. leftChildIdx = 5 #heap[5] = None
 4. rightChildIdx = 6 #heap[6] = None
 5. largestIdx = 2 #heap[2] = 35
 6. heapSize = 5
 8. if(5 < 5) #False. Sisanya tidak perlu dieksekusi
 11. if (6 < 5) #False. Sisanya tidak perlu dieksekusi
 14. if 2 != 2 #False
33. return [42, 33, 35, 10, 14]
```

---

## Menghapus node pada min binary heap

### Latihan

### Bibliografi



**B++**

**Graf**

## **Trie**

## **Tabel Hash**

## Referensi

- Adelson, V., Georgii, M., & Landis, E. M. (1962). An Algorithm for The Organization of Information. *Russian Academy of Sciences*, 146(2), 263–266.
- Deo, N., & Prasad, S. (1992). Parallel heap: An optimal parallel priority queue. *The Journal of Supercomputing*, 6(1), 87–98. <https://doi.org/10.1007/BF00128644>
- Fomitchev, M., & Ruppert, E. (2004). Lock-free linked lists and skip lists. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, 23, 50–59. <https://doi.org/10.1145/1011767.1011776>
- Yamamura, S., Kadota, T., Hirata, H., Niimi, H., & Shibayama, K. (2002). Evaluation of a data preload mechanism for a linked list structure. *Systems and Computers in Japan*, 33(3), 21–30. <https://doi.org/10.1002/scj.1110>
- Yang, J. C., Hensley, J., Grün, H., & Thibieroz, N. (2010). Real-time concurrent linked list construction on the GPU. *Computer Graphics Forum*, 29(4), 1297–1304. <https://doi.org/10.1111/j.1467-8659.2010.01725.x>
- Zhenguo, D., Qinqin, W., & Xianhua, D. (2009). An improved FP-growth algorithm based on compound single linked list. *2009 2nd International Conference on Information and Computing Science, ICIC 2009*, 1(1), 351–353. <https://doi.org/10.1109/ICIC.2009.96>