Pengantar Struktur Data

Subhan Hafiz Nanda Ginting, Hansi Effendi, Suresh Kumar,
Waris Marsisno, Yuliana Ria Uli Sitanggang, Khoerul Anwar,
Ni Putu Linda Santiari, Sigit Setyowibowo, Toar Romario Sigar,
Ibnu Atho'illah, Dwipo Setyantoro, Ni Nyoman Emang Smrti

Pengantar Struktur Data

Subhan Hafiz Nanda Ginting, Hansi Effendi, Suresh Kumar, Waris Marsisno, Yuliana Ria Uli Sitanggang, Khoerul Anwar, Ni Putu Linda Santiari, Sigit Setyowibowo, Toar Romario Sigar, Ibnu Atho'illah, Dwipo Setyantoro, Ni Nyoman Emang Smrti

PT. MIFANDI MANDIRI DIGITAL

Pengantar Struktur Data

Subhan Hafiz Nanda Ginting, Hansi Effendi, Suresh Kumar, Waris Marsisno, Yuliana Ria Uli Sitanggang, Khoerul Anwar, Ni Putu Linda Santiari, Sigit Setyowibowo, Toar Romario Sigar, Ibnu Atho'illah, Dwipo Setyantoro, Ni Nyoman Emang Smrti

ISBN: 978-623-88663-0-4

Editor : Sarwandi

Penyunting : Miftahul Jannah

Desain sampul : Rifki Ramadhan

Penerbit
PT. Mifandi Mandiri Digital

Redaksi
Komplek Senda Residence Jl. Payanibung Ujung D Dalu Sepuluh-B Tanjung Morawa Kab. Deli Serdang Sumatera Utara

Distributor Tunggal
PT. Mifandi Mandiri Digital
Komplek Senda Residence Jl. Payanibung Ujung D Dalu Sepuluh-B Tanjung Morawa Kab. Deli Serdang Sumatera Utara

Cetakan Pertama, Agustus 2023

Kata Pengantar

Selamat datang dalam buku "**Pengantar Struktur Data**". Buku ini adalah panduan komprehensif yang dirancang untuk memberikan pemahaman mendalam tentang konsep dasar dan pentingnya struktur data dalam pengembangan perangkat lunak. Dalam era di mana informasi menjadi semakin berharga, pengelolaan dan manipulasi data dengan efisien menjadi kunci keberhasilan dalam dunia teknologi.

Struktur data adalah landasan yang mendasari desain perangkat lunak yang efisien dan efektif. Melalui buku ini, kita akan menjelajahi beragam struktur data, mulai dari array sederhana hingga struktur yang lebih kompleks seperti linked list, stack, queue, tree, dan graph. Pemahaman yang kuat tentang struktur data akan membantu pembaca mengembangkan solusi perangkat lunak yang optimal, baik dalam hal kinerja maupun skalabilitas.

Buku ini dirancang untuk menjadi panduan yang ramah bagi pembaca dari berbagai latar belakang, termasuk para mahasiswa, pengembang perangkat lunak pemula, dan siapa pun yang tertarik untuk memperdalam pemahaman tentang struktur data. Setiap konsep akan dijelaskan dengan bahasa yang sederhana dan disertai dengan contoh-contoh praktis guna membantu pembaca memahami konsep tersebut dengan baik.

Seiring dengan penjelasan konsep struktur data, buku ini juga memberikan wawasan tentang bagaimana struktur data digunakan dalam dunia nyata, seperti dalam pengembangan aplikasi, pengolahan data besar, kecerdasan buatan, dan banyak bidang lainnya. Ini akan memberikan pembaca gambaran yang lebih luas tentang relevansi struktur data dalam industri teknologi modern.

Kami berharap bahwa buku "**Pengantar Struktur Data**" ini akan menjadi sumber pengetahuan yang berharga dan membantu Anda memahami konsep struktur data secara

mendalam. Terima kasih telah memilih buku ini sebagai panduan Anda. Selamat menikmati perjalanan Anda dalam memahami dunia yang menarik dari struktur data!

Medan, Juli 2023

Penulis

Daftar Isi

BAB 1 KONSEP STRUKTUR DATA

Pendahuluan

Dalam dunia komputasi, pengelolaan data yang efektif menjadi hal yang sangat penting, terutama dengan pertumbuhan data yang semakin besar dan kompleks. Inilah mengapa konsep struktur data menjadi sangat relevan. Struktur data merujuk pada cara data diatur dan disimpan di dalam komputer atau penyimpanan lainnya agar dapat diakses dan dimanipulasi dengan efisien.

Dalam pemrograman, pemilihan jenis struktur data memiliki dampak yang signifikan pada kinerja program, kompleksitas kode, dan penggunaan sumber daya komputer. Struktur data tidak hanya tentang cara sederhana seperti larik atau daftar, tetapi juga melibatkan konsep lebih kompleks seperti tumpukan, antrian, pohon, dan graf. Setiap jenis struktur data memiliki karakteristik khusus, kelebihan, dan kelemahan yang mempengaruhi cara data dapat diolah, dimodifikasi, dan dicari.

Pentingnya pemahaman tentang struktur data tidak hanya berlaku dalam pemrograman, tetapi juga dalam berbagai bidang lain seperti ilmu komputer, basis data, bioinformatika, dan pengolahan citra. Dengan memahami prinsip dasar dan performa berbagai struktur data, kita dapat merancang solusi yang efisien dan optimal untuk berbagai masalah di dunia teknologi informasi. Karenanya, pemahaman tentang struktur data serta cara mereka berinteraksi dengan algoritma merupakan langkah penting

dalam mengembangkan solusi teknologi yang inovatif dan efisien.

Struktur Data

Struktur data merujuk pada metode penyimpanan dan pengaturan data dengan cara tertata dalam sistem komputer atau basis data, bertujuan untuk mempermudah proses akses. Pengertianlain dari struktur data merupakan pendekatan dalam menyimpan, mengambil, dan mengatur data, sehingga memungkinkan data yang terdapat dalam komputer dapat diakses dan diperbaharui dengan lebih mudah.

Struktur data melibatkan pemilihan jenis struktur yang sesuai dengan jenis data yang akan diolah dan jenis operasi yang akan dilakukan pada data tersebut. Berikut beberapa konsep penting dalam struktur data:

1. Tipe Data Dasar
   Merupakan dasar dari struktur data, seperti bilangan bulat, bilangan desimal, karakter, boolean, dan lainnya. Tipe data dasar ini membentuk dasar untuk membangun struktur data yang lebih kompleks. Tipe data dasar dalam struktur data adalah jenis-jenis data yang tidak dapat dibagi menjadi bagian yang lebih kecil atau lebih sederhana. Tipe data dasar ini merupakan komponen dasar yang digunakan untuk membangun struktur data yang lebih kompleks. Berikut adalah contoh-contoh tipe data dasar:
   a. Integer
      Representasi bilangan bulat, seperti 1, -5, 100, dan sebagainya.
   b. Floating-Point (Bilangan Desimal)
      Representasi bilangan dengan pecahan, seperti 3.14, -0.5, 2.71828, dan sebagainya.

c. Karakter (Character)
   Representasi simbol atau huruf tunggal, seperti 'A', 'z', '@', dan lain-lain.
d. Boolean
   Representasi nilai kebenaran, hanya memiliki dua nilai: `true` atau `false`.
e. Byte
   Representasi bilangan bulat kecil (biasanya 8 bit), sering digunakan dalam pengolahan data yang lebih rendah.
f. Short
   Representasi bilangan bulat yang lebih besar dari byte namun lebih kecil dari integer.
g. Long
   Representasi bilangan bulat yang lebih besar dari integer, yang dapat menampung nilai yang lebih besar.
h. Double
   Representasi bilangan desimal dengan presisi ganda dibandingkan floating-point.
i. Float
   Representasi bilangan desimal dengan presisi tunggal.
j. Void
   Tipe khusus yang menunjukkan bahwa sebuah fungsi tidak mengembalikan nilai apa pun.

Tipe-tipe data dasar ini membentuk fondasi dari pemrograman dan pengembangan struktur data yang lebih kompleks. Dengan menggunakan tipe-tipe data dasar ini, kita dapat membangun jenis-jenis struktur data seperti array, list, *Stack*, *Queue*, dan lainnya untuk mengorganisir dan memanipulasi data dengan lebih efisien.

2. Larik dan Daftar

   Merupakan kumpulan data dengan tipe yang sama. Larik memiliki ukuran tetap, sedangkan daftar (atau array dinamis) memiliki ukuran yang dapat berubah sesuai kebutuhan. Struktur ini memungkinkan akses berdasarkan indeks. Larik (array) dan daftar (list) adalah dua jenis struktur data yang digunakan untuk mengelompokkan sejumlah elemen atau nilai. Kedua struktur data ini memungkinkan kita untuk menyimpan dan mengakses kumpulan data dengan cara yang terstruktur. Namun, ada perbedaan penting antara keduanya.

   a. Larik (Array)

      Larik adalah kumpulan elemen dengan tipe data yang sama, yang diindeks secara berurutan dan dikelompokkan di dalam satu variabel. Setiap elemen dalam larik diakses menggunakan indeks numerik yang dimulai dari 0. Ukuran larik biasanya tetap setelah dideklarasikan. Berikut ini contoh dari larik (array):

```
# Deklarasi dan inisialisasi larik integer
angka = [10, 20, 30, 40, 50]

# Mengakses elemen larik menggunakan indeks
print(angka[0])  # Output: 10
print(angka[2])  # Output: 30
```

Gambar 1. Contoh dari array

   b. Daftar (List)

      Daftar adalah kumpulan elemen dengan tipe data yang dapat berbeda-beda, yang juga

diindeks secara berurutan. Daftar lebih fleksibel karena dapat berisi elemen-elemen dengan tipe data yang beragam. Ukuran daftar dapat berubah setelah dideklarasikan. Berikut ini contoh dari daftar (list):

```
# Deklarasi dan inisialisasi daftar dengan beragam tipe data
data = [10, "hello", 3.14, True]

# Mengakses elemen daftar menggunakan indeks
print(data[1])    # Output: "hello"
print(data[3])    # Output: True
```

Gambar 2. Contoh dari array

3. Tumpukan (*Stack*)
   Merupakan struktur data linear di mana elemen-elemen ditambah atau diambil hanya dari ujung tertentu yang disebut "top". Prinsip kerjanya mirip dengan tumpukan buku, di mana buku yang terakhir diletakkan adalah yang pertama diambil. Tumpukan (*Stack*) adalah jenis struktur data yang mengikuti prinsip "last-in, first-out" (LIFO), yang berarti elemen yang terakhir dimasukkan ke dalam tumpukan akan menjadi yang pertama dikeluarkan. Ini mirip dengan tumpukan buku yang diletakkan satu per satu di atas meja. Elemen yang diletakkan terakhir harus diambil pertama kali sebelum kita dapat mengakses elemen-elemen sebelumnya.
   Operasi utama pada tumpukan adalah penambahan elemen ke bagian atas tumpukan (push) dan pengambilan elemen dari bagian atas tumpukan (pop). Tidak seperti daftar atau larik, tumpukan umumnya tidak memungkinkan akses langsung ke elemen-elemen di tengah tumpukan.

Contoh penggunaan tumpukan dalam bahasa pemrograman Python:

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

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

    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        else:
            return None

    def is_empty(self):
        return len(self.items) == 0

    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        else:
            return None

# Membuat tumpukan baru
stack = Stack()

# Menambahkan elemen ke dalam tumpukan
stack.push(10)
stack.push(20)
stack.push(30)

# Mengambil elemen dari tumpukan
print(stack.pop())  # Output: 30
print(stack.pop())  # Output: 20

# Melihat elemen teratas tanpa menghapusnya
print(stack.peek())  # Output: 10

# Mengecek apakah tumpukan kosong
print(stack.is_empty())  # Output: False
```

Gambar 3. Penggunaan tumpukan

4. Antrian (*Queue*)
   Mirip dengan tumpukan, antrian adalah struktur linear di mana elemen-elemen ditambahkan di salah satu ujung (antrian belakang) dan diambil dari ujung lainnya (antrian depan). Antrian (*Queue*) adalah jenis struktur data yang mengikuti prinsip "first-in, first-out" (FIFO), yang berarti elemen pertama kali dimasukkan ke dalam antrian akan menjadi yang pertama dikeluarkan. Ini mirip dengan antrian di

toko atau tempat lain, di mana orang yang pertama kali mengantri adalah yang pertama dilayani.

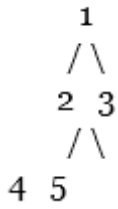
Operasi utama pada antrian adalah penambahan elemen ke ujung belakang antrian (en*Queue*) dan pengambilan elemen dari ujung depan antrian (de*Queue*). Seperti halnya tumpukan, antrian juga umumnya tidak memungkinkan akses langsung ke elemen-elemen di tengah antrian.

Contoh penggunaan antrian dalam bahasa pemrograman Python:

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

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

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        else:
            return None

    def is_empty(self):
        return len(self.items) == 0

    def peek(self):
        if not self.is_empty():
            return self.items[0]
        else:
            return None

# Membuat antrian baru
queue = Queue()

# Menambahkan elemen ke dalam antrian
queue.enqueue(10)
queue.enqueue(20)
queue.enqueue(30)

# Mengambil elemen dari antrian
print(queue.dequeue())  # Output: 10
print(queue.dequeue())  # Output: 20

# Melihat elemen pertama tanpa menghapusnya
print(queue.peek())  # Output: 30

# Mengecek apakah antrian kosong
print(queue.is_empty())  # Output: False
```

Gambar 4. Penggunaan antrian

5. Pohon (Tree)
   Struktur data hierarkis yang terdiri dari simpul-simpul yang saling terhubung. Pohon digunakan untuk merepresentasikan hierarki data seperti struktur direktori dalam sistem operasi. Pohon (Tree) adalah jenis struktur data hirarkis yang terdiri dari simpul-simpul (node) yang terhubung melalui cabang (edge). Node teratas dalam pohon disebut "akar" (root), dan setiap node dapat memiliki beberapa node anak di bawahnya, membentuk cabang-cabang. Node yang tidak memiliki anak disebut "daun" (leaf). Setiap node, kecuali akar, memiliki satu node "induk" (parent) yang berada di atasnya dalam struktur, dan setiap node dapat memiliki beberapa node "anak" (children) yang berada di bawahnya.
   Pohon digunakan untuk menggambarkan struktur hierarki, seperti struktur direktori dalam sistem file, struktur organisasi, atau hubungan keluarga. Contoh pohon dalam representasi grafis:



   Dalam contoh ini, node 1 adalah akar pohon. Node 2 dan 3 adalah anak-anak dari node 1. Node 4 dan 5 adalah anak-anak dari node 2. Node 1 adalah induk dari node 2 dan 3, dan node 2 adalah induk dari node 4 dan 5. Contoh pohon dalam representasi kode:

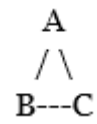
```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []

# Membuat pohon
root = TreeNode(1)
root.children.append(TreeNode(2))
root.children.append(TreeNode(3))
root.children[0].children.append(TreeNode(4))
root.children[0].children.append(TreeNode(5))
```

Gambar 5. Pohon dalam representasi kode

6. Graf

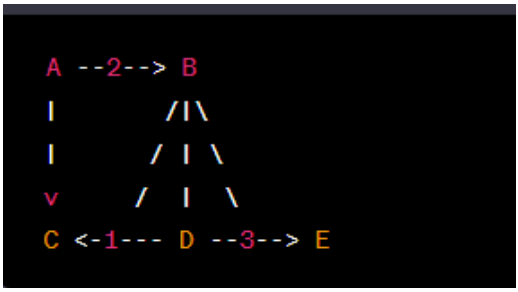
   Struktur data yang terdiri dari simpul-simpul yang terhubung oleh tepi. Graf digunakan untuk merepresentasikan hubungan kompleks antara berbagai entitas. Graf adalah struktur data yang terdiri dari simpul-simpul (node) yang terhubung oleh sisi atau tepi. Graf digunakan untuk merepresentasikan hubungan antara objek atau entitas dalam bentuk yang lebih abstrak. Setiap sisi menghubungkan dua simpul dan dapat memiliki arah (untuk graf terarah) atau tidak memiliki arah (untuk graf tak terarah).

   Ada beberapa jenis graf, termasuk graf terarah, graf tak terarah, graf berbobot, dan lain-lain, tergantung pada apakah sisi memiliki arah dan apakah sisi memiliki bobot (nilai) terkait. Contoh graf tak terarah:



   Dalam contoh ini, terdapat tiga simpul: A, B, dan C. Sisi yang menghubungkan B dan C serta sisi yang menghubungkan A dan B adalah contoh dari graf tak

terarah. Contoh graf terarah dengan bobot:



Gambar 6. Contoh graf terarah

Dalam contoh ini, terdapat lima simpul: A, B, C, D, dan E. Panah menunjukkan arah dari satu simpul ke simpul lainnya, dan angka di atas panah adalah bobot sisi.

Graf digunakan dalam berbagai aplikasi seperti jaringan sosial (hubungan antara pengguna), sistem rute perjalanan (jalan atau penerbangan antara lokasi), representasi data dalam basis data terdistribusi, analisis jaringan, dan banyak lagi.

7. Hash Map

   Struktur data yang menggunakan fungsi hash untuk menghubungkan kunci dengan nilai-nilai terkait. Ini memungkinkan akses cepat ke data berdasarkan kunci. Hash Map adalah struktur data yang digunakan untuk menyimpan pasangan kunci-nilai (key-value pairs) di dalamnya. Ini memungkinkan pengaksesan cepat terhadap nilai berdasarkan kuncinya. Hash Map bekerja dengan cara menghitung nilai hash dari kunci untuk menentukan posisi penyimpanan dalam struktur data, yang memungkinkan akses dan pencarian nilai dengan waktu yang sangat cepat. Hash Map biasanya digunakan untuk mengimplementasikan kamus, basis data terdistribusi, dan dalam berbagai situasi

di mana kecepatan pencarian nilai berdasarkan kunci adalah prioritas utama. Contoh Hash Map dalam Python:

```python
# Menggunakan built-in dict untuk membuat Hash Map
phone_book = {
    "Alice": "123-456-7890",
    "Bob": "987-654-3210",
    "Charlie": "555-123-4567"
}

# Mengakses nilai berdasarkan kunci
print(phone_book["Alice"])  # Output: "123-456-7890"
print(phone_book["Charlie"])  # Output: "555-123-4567"

# Menambahkan pasangan kunci-nilai baru
phone_book["Eve"] = "111-222-3333"

# Menghapus pasangan kunci-nilai
del phone_book["Bob"]
```

Gambar 7. Hash Map

Dalam contoh ini, phone_book adalah Hash Map yang menyimpan nomor telepon berdasarkan nama. Anda dapat dengan cepat mengakses nilai nomor telepon berdasarkan nama tanpa perlu mencari melalui daftar. Hash Map memanfaatkan fungsi hash untuk mengkonversi kunci menjadi indeks penyimpanan, sehingga pencarian atau akses nilai menjadi sangat cepat. Namun, penting untuk diingat bahwa jika terjadi tabrakan hash (dua kunci menghasilkan indeks yang sama), teknik resolusi tabrakan diperlukan untuk menangani situasi tersebut.

Dari pembelajaran tentang konsep struktur data, kita dapat mengambil beberapa kesimpulan penting:

1. Optimalisasi Kinerja

   Pemilihan struktur data yang tepat dapat secara signifikan meningkatkan kinerja program. Dengan memahami karakteristik masing-masing struktur

data, kita dapat memilih yang paling cocok untuk mengatasi jenis operasi yang dibutuhkan.

2. Penggunaan Sumber Daya
   Struktur data yang efisien dapat mengurangi penggunaan sumber daya seperti memori dan waktu komputasi. Ini penting terutama ketika menghadapi data dalam skala besar.
3. Pemecahan Masalah
   Konsep struktur data membantu kita memecahkan berbagai masalah secara lebih efektif. Dengan memilih struktur data yang sesuai, kita dapat merancang solusi yang lebih sederhana dan mudah dimengerti.
4. Pentingnya Pemahaman
   Memahami bagaimana struktur data bekerja, serta kompleksitas operasi yang terkait, merupakan kunci untuk merancang solusi yang efisien dan efektif. Pemahaman ini memungkinkan kita menghindari penggunaan struktur data yang tidak sesuai atau overkill.
5. Skalabilitas
   Dalam dunia yang terus berkembang, konsep struktur data memungkinkan solusi untuk tumbuh secara skalabel seiring dengan pertumbuhan data dan kompleksitas masalah.
6. Beragam Solusi
   Terdapat banyak jenis struktur data yang dapat digunakan untuk memecahkan masalah yang berbeda-beda. Setiap struktur memiliki kelebihan dan batasan tertentu, sehingga memilih yang tepat menjadi perhatian krusial.
7. Pengaruh Terhadap Algoritma
   Struktur data dan algoritma memiliki hubungan yang erat. Pemahaman yang baik tentang struktur

data akan membantu dalam merancang dan menerapkan algoritma yang optimal.

8. Penggunaan dalam Berbagai Bidang
   Konsep struktur data tidak hanya relevan dalam dunia pemrograman, tetapi juga memiliki aplikasi dalam berbagai bidang seperti ilmu komputer, bioinformatika, pengolahan citra, dan lebih banyak lagi.

Dengan memahami dan menguasai konsep struktur data, kita dapat menjadi pengembang yang lebih efektif dan kreatif dalam merancang solusi untuk masalah teknis yang kompleks. Memilih struktur data yang tepat dan menerapkannya dengan baik adalah langkah penting dalam menghasilkan aplikasi dan sistem yang lebih baik, efisien, dan handal.

BAB 2 TIPE DATA

Pendahuluan

Selamat datang di Bab 2! Pada bab ini, kita akan memasuki dunia yang menarik tentang tipe data dalam pemrograman. Mungkin kamu pernah mendengar istilah "tipe data" sebelumnya, tapi apa sebenarnya artinya? Nah, di bab ini, kita akan mengupas tuntas konsep tersebut.

Tipe data merupakan bagian penting dalam pemrograman. Mereka berperan dalam mengelompokkan dan mengatur nilai-nilai yang ada dalam program kita. Bayangkan tipe data sebagai wadah yang membantu kita menyimpan dan memanipulasi informasi dengan lebih terstruktur.

Sebagai contoh, bayangkan kamu memiliki beberapa data seperti nama, umur, dan nilai. Dengan menggunakan tipe data yang tepat, kita dapat mengelompokkan nama-nama dalam satu wadah, umur-umur dalam wadah lain, dan nilai-nilai dalam wadah yang berbeda pula. Hal ini memudahkan kita dalam menyimpan dan mengakses data dengan cara yang terorganisir.

Analogi wadah ini dapat membantu kita memahami peran tipe data dalam pemrograman. Seperti ketika kita mengatur barang-barang di rumah, tipe data membantu kita mengorganisir dan menyimpan nilai-nilai dalam program kita. Dengan tipe data yang tepat, kita dapat dengan mudah mencari, mengubah, dan menggunakan nilai-nilai tersebut sesuai kebutuhan.

Pentingnya pemahaman tentang tipe data tidak bisa

diabaikan dalam pemrograman. Dengan memahami tipe data dengan baik, kita dapat mengoptimalkan penggunaan memori dan meningkatkan efisiensi program. Selain itu, pemilihan tipe data yang tepat juga membantu mencegah kesalahan dalam pengolahan data dan meningkatkan kualitas program secara keseluruhan.

Jadi, bersiaplah untuk memasuki dunia tipe data yang menarik ini! Kita akan menjelajahi berbagai jenis tipe data dan belajar bagaimana menggunakannya dengan efektif. Dengan pemahaman yang baik tentang tipe data, kita akan menjadi programmer yang lebih terampil dalam mengelola dan memanipulasi informasi. Ayo, mari kita mulai petualangan kita dalam pemahaman tipe data dalam pemrograman!.

Pengertian Tipe Data

Pengertian tipe data sangat penting dalam pemrograman. Tipe data dapat didefinisikan sebagai jenis kategori atau wadah yang digunakan untuk mengelompokkan nilai-nilai yang ada dalam program. Dalam pemrograman, tipe data menentukan jenis nilai yang dapat disimpan dan diolah oleh program. Misalnya, kita dapat memiliki tipe data seperti angka bulat (integer), angka desimal (float), nilai kebenaran (boolean), karakter (char), dan sebagainya.

Pemilihan tipe data yang tepat sangat penting karena dapat mempengaruhi penggunaan memori dan kinerja program secara keseluruhan. Ketika kita menyimpan data dalam variabel atau struktur data, kita perlu mempertimbangkan ukuran memori yang akan digunakan. Misalnya, jika kita hanya perlu menyimpan angka bulat kecil, menggunakan tipe data integer dengan ukuran yang lebih kecil, seperti byte atau short, akan lebih

efisien dibandingkan menggunakan tipe data integer dengan ukuran yang lebih besar, seperti long atau bigint. Dengan memilih tipe data yang tepat, kita dapat mengoptimalkan penggunaan memori dan menghindari pemborosan sumber daya.

Selain itu, pemilihan tipe data yang tepat juga mempengaruhi kinerja program. Ketika kita melakukan operasi matematika, perbandingan, atau manipulasi data lainnya, tipe data yang digunakan akan mempengaruhi hasil perhitungan dan akurasi. Misalnya, jika kita perlu melakukan operasi matematika yang melibatkan angka desimal, menggunakan tipe data float atau double yang tepat akan memastikan hasil perhitungan yang akurat. Jika kita salah memilih tipe data, misalnya menggunakan tipe data integer saat melakukan perhitungan desimal, hal ini dapat mengakibatkan kesalahan dalam perhitungan dan menghasilkan output yang tidak diharapkan.

Pentingnya memilih tipe data yang tepat juga terkait dengan keamanan program. Misalnya, jika kita menerima input pengguna, kita perlu memvalidasi tipe data yang diterima agar sesuai dengan yang diharapkan. Jika kita tidak memvalidasi input dengan benar, hal ini dapat menyebabkan kesalahan atau kerentanan keamanan dalam program.

Dengan memahami pengertian tipe data dan peranannya dalam pemrograman, kita dapat membuat program yang lebih efisien dan dapat diandalkan. Memilih tipe data yang tepat memungkinkan kita untuk mengatur dan mengolah nilai dengan lebih efisien, menjaga penggunaan memori yang optimal, serta memastikan kinerja program yang baik. Oleh karena itu, pemahaman yang baik tentang tipe data sangat penting bagi seorang pemrogram untuk menciptakan program yang efektif, aman, dan berkualitas.

Tipe Data Primitif

Tipe data primitif adalah tipe data dasar yang telah didefinisikan dalam bahasa pemrograman. Mereka merupakan tipe data yang tidak dapat dibagi lagi menjadi komponen yang lebih kecil. Tipe data primitif termasuk dalam kategori tipe data yang paling mendasar dan sering digunakan dalam pemrograman.

Integer

Salah satu tipe data primitif yang umum adalah tipe data integer. Integer adalah tipe data yang digunakan untuk menyimpan bilangan bulat, baik positif maupun negatif, tanpa bagian pecahan. Tipe data ini berguna untuk menghitung jumlah, mengindeks array, atau merepresentasikan entitas diskrit dalam program. Contohnya, kita dapat menggunakan tipe data integer untuk menyimpan umur seseorang atau jumlah barang dalam stok. Berikut adalah contoh pengkodean untuk menyimpan dan melakukan operasi pada bilangan bulat:

```
# Deklarasi variabel integer
umur = 25
jumlah_barang = 10

# Operasi pada bilangan bulat
total_umur = umur + 5
sisa_barang = jumlah_barang - 3
```

Float

Selain integer, tipe data primitif lainnya adalah float. Float digunakan untuk menyimpan bilangan desimal, termasuk angka pecahan. Tipe data ini berguna untuk

perhitungan yang melibatkan bilangan dengan bagian pecahan, seperti perhitungan matematika yang akurat. Contohnya, kita dapat menggunakan tipe data float untuk menyimpan nilai tinggi badan seseorang atau hasil perhitungan yang melibatkan pecahan. Berikut adalah contoh pengkodean untuk menyimpan dan melakukan operasi pada bilangan desimal:

```
# Deklarasi variabel float
nilai_pi = 3.14
tinggi_badan = 1.75

# Operasi pada bilangan desimal
luas Lingkaran = nilai_pi * (jari_jari ** 2)
bmi = berat_badan / (tinggi_badan ** 2)
```

Boolean

Selanjutnya, terdapat tipe data boolean yang digunakan untuk menyimpan nilai kebenaran. Tipe data boolean hanya memiliki dua nilai, yaitu true dan false. Tipe data ini berguna untuk pengambilan keputusan dan kontrol aliran program berdasarkan kondisi tertentu. Contohnya, kita dapat menggunakan tipe data boolean untuk memeriksa apakah suatu kondisi benar atau salah, atau untuk mengaktifkan atau menonaktifkan fitur dalam program. Berikut adalah contoh pengkodean untuk menyimpan dan menggunakan nilai kebenaran:

```
# Deklarasi variabel boolean
benar = True
salah = False

# Penggunaan nilai kebenaran
```

```
if benar:
   print("Pernyataan ini benar")
else:
   print("Pernyataan ini salah")
```

Char

Terakhir, terdapat tipe data char yang digunakan untuk menyimpan karakter tunggal. Tipe data ini berguna untuk merepresentasikan huruf, angka, atau simbol dalam program. Misalnya, kita dapat menggunakan tipe data char untuk menyimpan huruf inisial seseorang atau karakter kunci dalam enkripsi data. Berikut adalah contoh pengkodean untuk menyimpan dan memanipulasi karakter:

```
# Deklarasi variabel char
huruf = 'A'
simbol = '*'

#Penggunaan nilai karakter
print("Huruf awal:", huruf)
print("Karakter kunci:", simbol)

#Manipulasi karakter
huruf = chr(ord(huruf) + 1)
print("Huruf setelah:", huruf)
```

Dalam contoh di atas, kita menyimpan karakter tunggal 'A' dalam variabel huruf dan simbol '*' dalam variabel simbol. Kemudian, kita dapat menggunakan nilai karakter ini dalam berbagai operasi dan manipulasi. Dalam contoh tersebut, kita menggunakan fungsi chr() dan ord() untuk mengubah karakter menjadi kode ASCII dan

sebaliknya. Selanjutnya, kita dapat melakukan manipulasi karakter dengan melakukan operasi matematika atau operasi string lainnya.

Tipe data primitif, seperti integer, float, boolean, dan char, merupakan komponen penting dalam pemrograman. Dengan menggunakan tipe data primitif, kita dapat menyimpan, mengolah, dan memanipulasi nilai-nilai dengan tepat sesuai dengan kebutuhan program kita. Dengan memahami pengertian dan penggunaan tipe data primitif, kita dapat mengoptimalkan penggunaan memori, meningkatkan kinerja program, dan menciptakan program yang akurat dan efisien.

Tipe Data Terstruktur

Tipe data terstruktur merupakan konsep dalam pemrograman yang memungkinkan penggabungan beberapa nilai ke dalam satu entitas. Tipe data ini memungkinkan kita untuk mengorganisasi data secara lebih kompleks dan terstruktur. Beberapa tipe data terstruktur yang umum digunakan adalah array, record, dan pointer.

Array

Array adalah tipe data terstruktur yang digunakan untuk menyimpan kumpulan nilai dengan tipe data yang sama. Array memungkinkan kita untuk menyimpan beberapa nilai dalam satu variabel, sehingga memudahkan pengelolaan data yang terkait. Array memiliki indeks untuk mengakses setiap elemen dalam array tersebut. Contohnya, kita dapat menggunakan tipe data array untuk menyimpan daftar angka atau nama-nama dalam program. Berikut adalah contoh pengkodean untuk deklarasi, mengakses, dan memanipulasi elemen array:

```
# Deklarasi array
angka = [1, 2, 3, 4, 5]

# Mengakses elemen array
print(angka[0])  # Output: 1

# Memanipulasi elemen array
angka[2] = 10
print(angka) # Output: [1, 2, 10, 4, 5]
```

Records

Selanjutnya, tipe data record adalah tipe data terstruktur yang memungkinkan kita untuk menggabungkan beberapa variabel dengan tipe data yang berbeda ke dalam satu entitas. Record digunakan untuk merepresentasikan objek atau entitas yang memiliki atribut-atribut yang berbeda. Misalnya, kita dapat menggunakan tipe data record untuk menyimpan informasi tentang seseorang, seperti nama, usia, dan alamat. Berikut adalah contoh pengkodean untuk deklarasi, pengisian, dan pengaksesan atribut dalam record:

```
# Deklarasi record
class Mahasiswa:
    def __init__(self, nama, usia, alamat):
        self.nama = nama
        self.usia = usia
        self.alamat = alamat

# Pengisian atribut dalam record
mahasiswa1 = Mahasiswa("John Doe", 20, "Jl. Jendral Sudirman")
print(mahasiswa1.nama)  # Output: John Doe
```

```
print(mahasiswa1.usia)  # Output: 20
print(mahasiswa1.alamat)    # Output: Jl. Jendral Sudirman
```

Pointer

Terakhir, tipe data pointer adalah tipe data terstruktur yang digunakan untuk menyimpan alamat memori suatu variabel. Pointer memungkinkan kita untuk mengakses dan memanipulasi data yang disimpan dalam alamat tersebut. Tipe data pointer sering digunakan dalam manajemen memori dan penggunaan struktur data yang lebih kompleks. Berikut adalah contoh pengkodean untuk deklarasi, penggunaan, dan manajemen memori dengan pointer:

```
# Deklarasi pointer
nilai = 10
pointer = None
pointer = id(nilai)

# Mengakses data melalui pointer
print(pointer)  # Output: 140730127458000
print(cast(pointer, ctypes.py_object).value)
# Output: 10
```

Dengan menggunakan tipe data terstruktur seperti array, record, dan pointer, kita dapat mengatur dan mengelola data secara lebih efisien dan terorganisir dalam pemrograman. Tipe data array memungkinkan kita untuk mengelompokkan sejumlah nilai dengan tipe data yang sama, sehingga memudahkan akses dan manipulasi data dalam satu variabel. Contohnya, kita dapat menggunakan tipe data array untuk menyimpan data suhu dalam

beberapa hari terakhir atau nilai-nilai dalam matriks.

Tipe data record memungkinkan kita untuk menggabungkan beberapa variabel dengan tipe data yang berbeda ke dalam satu entitas. Dengan demikian, kita dapat merepresentasikan objek atau data yang lebih kompleks. Misalnya, dalam program pengelolaan data karyawan, kita dapat menggunakan tipe data record untuk menyimpan informasi seperti nama, alamat, gaji, dan posisi pekerjaan dari setiap karyawan.

Sementara itu, tipe data pointer digunakan untuk mengelola alamat memori suatu variabel. Dalam pemrograman, penggunaan pointer sering berkaitan dengan manajemen memori dan akses langsung ke data. Misalnya, kita dapat menggunakan pointer untuk mengalokasikan memori secara dinamis, mengakses elemen dalam struktur data yang kompleks, atau mempercepat proses manipulasi data.

Dalam keseluruhan, tipe data terstruktur memungkinkan pemrogram untuk mengorganisasi, mengelompokkan, dan mengelola data secara lebih efektif dalam program. Dengan memahami penggunaan dan fungsionalitas tipe data terstruktur, kita dapat merancang program yang lebih terstruktur, efisien, dan mudah dipahami. Penting untuk memilih tipe data terstruktur yang sesuai dengan kebutuhan program, sehingga dapat mengoptimalkan kinerja dan penggunaan memori dalam pengembangan perangkat lunak.

BAB 3 POINTER

Pendahuluan

Kita akan memulai perjalanan yang mengasyikkan menuju pointer, sebuah konsep penting dalam struktur data. Pointer adalah komponen mendasar dari bahasa pemrograman, memungkinkan kita memanipulasi alamat memori dan mengakses data dengan lebih efisien dan fleksibel. Memahami pointer sangat penting untuk mengembangkan algoritma yang efisien dan mengimplementasikan struktur data yang kompleks. Jadi, mari selami dan jelajahi dunia penunjuk yang menakjubkan bersama-sama.

Pengertian dan Tujuan Pointer

Pertama, mari kita definisikan pointer dan mengapa itu penting. Secara sederhana, pointer adalah variabel yang menyimpan alamat memori dari variabel lain. Anggap saja sebagai "penunjuk" yang mengarahkan kita ke lokasi di memori tempat data yang kita inginkan berada. Menggunakan pointer memungkinkan kita untuk mengakses dan memanipulasi data secara tidak langsung, memungkinkan kita membangun struktur data yang dinamis dan melakukan manajemen memori yang efisien.

Salah satu contoh klasik penggunaan pointer adalah dalam pemrograman C. Di C, pointer banyak digunakan untuk alokasi memori dinamis, memungkinkan kita mengalokasikan memori saat runtime dan melepaskannya

saat tidak lagi diperlukan. Fleksibilitas ini sangat berharga saat berhadapan dengan struktur data yang memerlukan pengubahan ukuran dinamis, seperti daftar tertaut atau pohon biner (Smith, 2010).

Selain itu, pointer memungkinkan kita untuk meneruskan variabel dengan referensi, bukan dengan nilai, ke fungsi. Ini bisa lebih efisien, karena menghindari penyalinan data yang tidak perlu. Dengan meneruskan penunjuk ke suatu posisi, kita dapat memodifikasi variabel asli secara langsung, menjadikannya alat yang ampuh untuk membangun algoritme dan memanipulasi struktur data kompleks secara efisien (Johnson, 2018).

Representasi Memori dan Alamat

Sekarang, mari selami dunia representasi dan alamat memori yang menarik. Setiap variabel yang kita deklarasikan dalam program kita diberi lokasi memori, alamat unik tempat nilainya disimpan. Pointer memungkinkan kita untuk mengakses dan memanipulasi alamat memori ini, memberi kita kendali penuh atas data kita.

Misalnya, pertimbangkan variabel 'x' dengan nilai 42. Kita bisa mendapatkan alamat memori tempat 'x' disimpan menggunakan pointer. Alamat memori ini dapat berupa nilai heksadesimal, seperti 0x7FFB9F4A. Penting untuk diperhatikan bahwa alamat memori bersifat unik dan berfungsi sebagai pengidentifikasi untuk lokasi tertentu di memori komputer.

Memahami alamat memori menjadi sangat penting ketika berhadapan dengan dataset besar atau struktur data yang kompleks. Pointer memungkinkan kita mengakses, memodifikasi, dan melintasi struktur ini secara efisien dengan mereferensikan alamat memori mereka secara

langsung.

Aritmatika Pointer dan Kegunaannya

Sekarang setelah kita memahami konsep alamat memori, mari jelajahi aritmatika pointer dan signifikansinya. Aritmetika pointer memungkinkan kita untuk melakukan operasi aritmatika pada pointer, seperti penjumlahan, pengurangan, dan perbandingan. Fungsionalitas ini memungkinkan kita untuk menavigasi array dan memanipulasi data secara efisien.

Sebagai contoh, pertimbangkan array integer 'arr' yang berisi elemen [10, 20, 30, 40, 50]. Menggunakan aritmatika penunjuk, kita dapat mengakses fitur individual dalam larik dengan memanipulasi alamat memori. Misalnya, dengan menambahkan satu pointer ke 'arr', kita dapat mengakses elemen kedua dari array, '20'. Kemampuan untuk berpindah melalui alamat memori ini memungkinkan kita untuk mengulang array, melakukan algoritma pengurutan, dan mengimplementasikan berbagai struktur data secara efisien.

Selain itu, aritmatika penunjuk digunakan dalam pemrograman tingkat rendah, seperti sistem operasi atau driver perangkat, untuk berinteraksi langsung dengan register perangkat keras atau memanipulasi blok memori. Aplikasi ini memerlukan kontrol yang tepat atas alamat memori dan manajemen memori yang efisien, menjadikan aritmatika penunjuk sebagai alat penting (Davis, 2016).

Bekerja dengan Pointer

Mendeklarasikan dan Menginisialisasi Pointer

Mari kita mulai dengan memahami cara mendeklarasikan dan menginisialisasi pointer. Saat

mendeklarasikan pointer, kita menentukan tipe data yang ditunjuknya. Ini memungkinkan kompiler untuk mengalokasikan ruang memori yang sesuai untuk pointer.

Untuk mendeklarasikan pointer, kita menggunakan simbol asterisk (*) diikuti dengan nama pointer. Misalnya, untuk mendeklarasikan pointer ke bilangan bulat, kita akan menulis:

```
int *ptr;
```

Dalam contoh ini, ptr adalah penunjuk ke bilangan bulat. Itu belum menunjuk ke alamat memori tertentu. Penting untuk dicatat bahwa tanda bintang (*) adalah bagian dari deklarasi tipe penunjuk, yang menunjukkan bahwa itu adalah variabel penunjuk.

Setelah mendeklarasikan sebuah pointer, kita perlu menginisialisasinya dengan alamat memori yang harus ditunjuknya. Kita dapat menginisialisasi pointer dengan menggunakan alamat-operator (&) diikuti dengan nama variabel. Misalnya:

```
int num = 10;
int *ptr = &num;
```

Di sini, pointer ptr diinisialisasi dengan alamat memori dari variabel num menggunakan alamat-operator (&). Sekarang, ptr menunjuk ke lokasi memori tempat num disimpan.

Mengakses Variabel Menggunakan Pointer

Setelah kita mendeklarasikan dan menginisialisasi pointer, kita dapat menggunakannya untuk mengakses variabel secara tidak langsung. Ini dicapai dengan

mendereferensi pointer, yang memungkinkan kita untuk mengakses nilai yang disimpan di alamat memori yang ditunjuknya.

Untuk melakukan dereferensi pointer, kita menggunakan simbol asterisk (*) lagi, kali ini sebagai operator. Misalnya:

```
int num = 10;
int *ptr = &num;

printf("Value of num: %d\n", *ptr);
```

Dalam contoh ini, ekspresi *ptr mendereferensi pointer, memberi kita akses ke nilai yang disimpan di alamat memori yang ditunjuknya. Dalam hal ini, itu akan mencetak nilai 10.

Alokasi dan Dealokasi Memori Dinamis

Salah satu fitur kuat dari pointer adalah kemampuannya untuk mengalokasikan dan membatalkan alokasi memori secara dinamis saat runtime. Fleksibilitas ini sangat penting saat bekerja dengan struktur data yang memerlukan pengubahan ukuran dinamis, seperti daftar tertaut atau pohon.

Di C, kita menggunakan fungsi malloc() untuk mengalokasikan memori secara dinamis. Ini mengembalikan pointer ke blok memori yang dialokasikan. Misalnya:

```
int *ptr = (int *)malloc(sizeof(int));
```

Dalam contoh ini, kita mengalokasikan memori secara dinamis untuk bilangan bulat menggunakan

malloc(). sizeof(int) menentukan ukuran blok memori yang akan dialokasikan. Kita mengetikkan nilai pengembalian malloc() ke pointer integer (int *) agar sesuai dengan tipe pointer.

Setelah mengalokasikan memori secara dinamis, penting untuk membatalkan alokasinya saat tidak diperlukan lagi untuk menghindari kebocoran memori. Kita menggunakan fungsi free() untuk melepaskan memori. Misalnya:

```
int *ptr = (int *)malloc(sizeof(int));
// Use the allocated memory

free(ptr);
```

Di sini, fungsi free() melepaskan memori yang ditunjuk oleh ptr, membuatnya tersedia untuk penggunaan di masa mendatang.

Meneruskan Pointer ke Fungsi

Melewati pointer ke fungsi memungkinkan kita untuk memodifikasi variabel secara langsung dan berbagi data secara efisien di antara berbagai bagian program kita. Ketika pointer diteruskan ke suatu fungsi, setiap perubahan yang dilakukan pada variabel melalui pointer akan mempengaruhi variabel asli di luar fungsi.

Untuk meneruskan penunjuk ke suatu fungsi, kita mendeklarasikan parameter fungsi sebagai penunjuk dan menggunakan operator dereferensi (*) untuk mengakses dan memodifikasi variabel. Misalnya:

```
        void modifyValue(int *ptr)
{
```

```
    *ptr = *ptr + 10;
}

int num = 5;
modifyValue(&num);
```

Dalam contoh ini, fungsi modifiedValue() mengambil pointer ke integer sebagai parameter. Dengan mendereferensi penunjuk, kita dapat mengakses dan memodifikasi nilai yang disimpan di alamat memori yang ditunjuknya. Dalam hal ini, nilai num akan diubah menjadi 15 setelah memanggil fungsi.

Aplikasi Pointer

Pada bagian ini, kita akan menjelajahi aplikasi pointer di berbagai struktur data. Pointer memainkan peran penting dalam mengimplementasikan struktur data dinamis secara efisien. Kami akan fokus pada penggunaannya dalam daftar tertaut, pohon, grafik, dan struktur data lainnya.

Linked Lists dan Pointer

Linked lists adalah struktur data fundamental yang terdiri dari node yang dihubungkan bersama menggunakan pointer. Pointer memungkinkan kita untuk menghubungkan node secara dinamis, memungkinkan penyisipan, penghapusan, dan traversal elemen yang efisien (Cormen et al., 2009).

Dalam Linked lists, setiap simpul berisi data dan penunjuk ke simpul berikutnya dalam daftar. Linked lists inilah yang memberikan fleksibilitas untuk menambah dan menghapus elemen secara dinamis. Pointer memfasilitasi penjelajahan melalui daftar, karena setiap pointer node

menunjukkan lokasi node berikutnya.

Sebagai contoh, mari pertimbangkan linked list tunggal dengan struktur berikut:

```
struct Node
{
   int data;
   struct Node* next;
};
```

Di sini, pointer berikutnya menghubungkan satu node ke node berikutnya. Dengan memanipulasi pointer ini, kita dapat melakukan operasi seperti menyisipkan elemen di awal, tengah, atau akhir daftar, serta menghapus elemen secara efisien.

Trees and Pointers

Trees adalah struktur data hierarkis dengan kumpulan node yang terhubung, di mana setiap node dapat memiliki node anak. Pointer digunakan untuk membangun koneksi antara node induk dan anak, memungkinkan operasi traversal, penyisipan, dan penghapusan yang efisien (Weiss, 2016).

Dalam pohon biner, misalnya, setiap simpul memiliki maksimal dua simpul anak, biasanya disebut sebagai anak kiri dan anak kanan. Pointer memungkinkan kita menavigasi melalui struktur pohon secara efisien.

Dengan menggunakan pointer, kita dapat mengimplementasikan algoritma tree traversal seperti in-order, pre-order, dan post-order traversal. Algoritme ini mengandalkan pointer untuk mengunjungi node dalam urutan tertentu, menjadikannya penting untuk mencari, menyortir, dan memanipulasi data yang disimpan di

pohon.

Grafik dan Pointer

Grafik adalah struktur data serbaguna yang terdiri dari node (simpul) yang dihubungkan oleh tepi. Pointer memainkan peran penting dalam representasi grafik dan traversal. Dengan pointer, kita dapat membuat koneksi antar node dan melintasi grafik secara efisien (Sedgewick & Wayne, 2011).

Misalnya, dalam representasi daftar adjacency dari grafik, setiap node dikaitkan dengan daftar tertaut dari node yang berdekatan. Pointer digunakan untuk membangun koneksi antar node ini, memungkinkan eksplorasi grafik yang efisien.

Pointer sangat penting dalam algoritma grafik seperti pencarian pertama luas (BFS) dan pencarian pertama mendalam (DFS), di mana mereka membantu melintasi grafik, melacak node yang dikunjungi, dan menjelajahi node yang berdekatan secara efisien.

Pointer di Struktur Data Lain

Selain linked lists, trees, dan grafik, pointer memiliki berbagai aplikasi dalam struktur data lainnya. Misalnya:

1. Tabel hash
   Pointer digunakan untuk menangani tabrakan dan menyelesaikannya dengan menghubungkan beberapa elemen secara bersamaan.
2. Tumpukan dan antrian
   Pointer memfasilitasi penyisipan dan penghapusan elemen yang efisien dalam struktur data linier ini (Lafore, 2017).
3. Array dinamis
   Pointer memungkinkan pengubahan ukuran array

secara dinamis, memungkinkan manajemen memori yang efisien dan manipulasi data.

4. Heaps
   Pointer memungkinkan implementasi yang efisien dari antrian prioritas berbasis heap dan algoritma sortir heap (Cormen et al., 2009).

Dengan menggunakan pointer, kita dapat mengimplementasikan dan memanipulasi berbagai struktur data secara efisien, memungkinkan kita untuk memecahkan masalah kompleks secara efektif.

Jebakan Penunjuk dan Praktik Terbaik

Di bagian ini, kita akan menjelajahi jebakan umum yang terkait dengan petunjuk dan mendiskusikan praktik terbaik untuk menghindarinya. Memahami dan mengikuti praktik ini sangat penting untuk menulis program yang tangguh dan bebas kesalahan. Kami akan membahas topik seperti null pointer, kebocoran memori, dan cara untuk mengurangi masalah ini.

Null Pointer dan Kebocoran Memori

Null pointer terjadi ketika pointer tidak menunjuk ke alamat memori yang valid. Mengakses atau mendereferensi penunjuk nol dapat menyebabkan kesalahan runtime, seperti kesalahan segmentasi. Kebocoran memori, di sisi lain, terjadi ketika memori yang dialokasikan secara dinamis tidak dialokasikan dengan benar, mengakibatkan pemborosan memori dan potensi ketidakstabilan program (Prata, 2013).

Untuk menghindari null pointer, praktik yang baik adalah menginisialisasi pointer dengan nilai null saat dideklarasikan. Dengan cara ini, kita dapat memeriksa apakah sebuah pointer null sebelum mengakses atau

melakukan dereferensi, mencegah potensi kesalahan.

Dealokasi memori yang tepat sangat penting untuk menghindari kebocoran memori. Setiap kali kita mengalokasikan memori secara dinamis menggunakan fungsi seperti `malloc()` atau `new`, kita harus memastikan bahwa memori yang dialokasikan dibebaskan menggunakan `free()` atau `delete` saat tidak diperlukan lagi. Penting untuk melepaskan memori tepat waktu untuk mencegah kebocoran memori dan memastikan manajemen memori yang efisien (Jones & Ohlund, 2012).

Dangling Pointers dan Garbage Values

Dangling Pointer terjadi ketika pointer terus menunjuk ke lokasi memori bahkan setelah memori telah dibatalkan alokasinya. Mengakses atau mendereferensi penunjuk yang menggantung dapat menyebabkan perilaku yang tidak terdefinisi, yang menyebabkan program macet atau hasil yang salah. Garbage values tidak dapat diprediksi dan data sewenang-wenang yang mungkin berada di lokasi memori yang tidak dialokasikan.

Untuk menghindari dangling pointer, sangat penting untuk mengatur pointer ke null atau memberikannya alamat memori yang valid setelah memori terkait dibebaskan. Ini mencegah penunjuk menjadi penunjuk menggantung dan memastikan bahwa penunjuk menunjuk ke lokasi yang valid.

Untuk mengurangi garbage values, praktik yang baik adalah menginisialisasi pointer saat dideklarasikan. Menginisialisasi pointer ke null atau menetapkannya sebagai alamat memori yang valid memastikan bahwa pointer tersebut tidak berisi data yang tidak dapat diprediksi (Seacord, 2005).

Kesalahan Pointer dan Teknik Debugging

Kesalahan pointer, seperti mengakses memori di luar batas atau membatalkan alokasi memori yang masih digunakan, dapat menyebabkan program macet atau perilaku yang tidak dapat diprediksi. Men-debug error semacam itu bisa jadi menantang karena sifatnya yang non-deterministik.

Untuk mendiagnosis dan memperbaiki kesalahan penunjuk, berbagai teknik debug dapat digunakan. Alat seperti debugger memori, seperti Valgrind, dapat membantu mengidentifikasi masalah terkait memori dengan mendeteksi kebocoran memori, akses memori ilegal, dan kesalahan lainnya. Peninjauan kode yang hati-hati, pengujian, dan penggunaan fitur debug yang disediakan oleh lingkungan pengembangan terintegrasi (IDE) juga dapat membantu dalam mengidentifikasi dan menyelesaikan kesalahan pointer (Schilthuizen, 2019).

Optimasi Berbasis Pointer dan Pertimbangan Kinerja

Pointer dapat menawarkan manfaat kinerja saat digunakan secara optimal. Memahami arsitektur perangkat keras yang mendasari dan teknik manajemen memori dapat membantu mengoptimalkan penggunaan penunjuk dan meningkatkan kinerja program.

Optimalisasi berbasis pointer mencakup teknik seperti aritmatika pointer, yang bisa lebih efisien daripada pengindeksan array dalam skenario tertentu. Selain itu, menggunakan penunjuk untuk meneruskan argumen ke fungsi dengan referensi dapat menghindari penyalinan data yang tidak perlu, sehingga menghasilkan peningkatan kinerja.

Namun, penting untuk menyeimbangkan

pengoptimalan dengan keterbacaan dan pemeliharaan. Manipulasi penunjuk yang terlalu rumit dapat menyebabkan kode yang sulit dipahami dan dipelihara. Oleh karena itu, penting untuk mempertimbangkan trade-off dan menggunakan teknik pengoptimalan dengan bijaksana.

Studi Kasus dan Contoh

Pada bagian ini, kita akan mendalami studi kasus dan contoh yang mendemonstrasikan penerapan praktis pointer dalam berbagai struktur data. Kami akan mengeksplorasi implementasi daftar tertaut menggunakan pointer, algoritma traversal pohon yang menggunakan pointer, algoritma grafik berbasis pointer, dan aplikasi pointer dunia nyata.

Mengimplementasikan Linked List Menggunakan Pointer

Daftar tertaut adalah struktur data dinamis di mana setiap elemen, yang dikenal sebagai simpul, berisi data dan penunjuk ke simpul berikutnya. Pointer memainkan peran mendasar dalam menghubungkan node dan memungkinkan operasi penyisipan, penghapusan, dan traversal yang efisien.

Mari pertimbangkan contoh penerapan daftar tertaut tunggal menggunakan pointer di C:

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
```

```
    int data;
    struct Node* next;
};

void insert(struct Node** head, int value)
{
    struct Node* newNode =
    (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

void display(struct Node* head)
{
    struct Node* current = head;
    while (current != NULL)
{
        printf("%d ", current->data);
        current = current->next;
            }
    printf("\n");
}

int main()
{
    struct Node* head = NULL;
    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 30);
    display(head);
    return 0;
}
```

Dalam contoh ini, kita mendefinisikan struktur Node yang menyimpan data dan penunjuk ke node berikutnya. Fungsi insert() menyisipkan node baru di awal daftar tertaut, dan fungsi display() melintasi daftar dan mencetak data dari setiap node(Lafore, 2017).

Dengan memanfaatkan pointer untuk membuat koneksi antar node, kita dapat secara efisien melakukan operasi pada daftar tertaut.

Algoritma Tree Traversal Menggunakan Pointer

Pohon adalah struktur hierarkis yang terdiri dari simpul-simpul yang saling berhubungan. Algoritme traversal memungkinkan kita untuk mengunjungi setiap node dalam urutan tertentu. Pointer memfasilitasi traversal pohon yang efisien dengan menyediakan akses ke node anak dan menentukan urutan traversal.

Mari pertimbangkan contoh algoritme in-order traversal untuk pohon biner yang diimplementasikan menggunakan pointer:

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
   int data;
   struct Node* left;
   struct Node* right;
};

struct Node* createNode(int value)
{
   struct Node* newNode =
```

```
    (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void inorderTraversal(struct Node* root) {
    if (root == NULL)
  {
        return;
    }

    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

int main()
{
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);

    printf("In-order traversal: ");
    inorderTraversal(root);

    return 0;
}
```

Dalam contoh ini, kita mendefinisikan struktur Node dengan data, pointer kiri, dan kanan. Fungsi createNode()

membuat node baru dengan nilai yang diberikan. Fungsi inorderTraversal() secara rekursif melintasi pohon dalam urutan berurutan (kiri, akar, kanan) dan mencetak data dari setiap node (Cormen et al., 2009).

Pointer memungkinkan kita melintasi pohon secara efisien dengan mengikuti koneksi antar node dan mengunjungi setiap node dalam urutan yang diinginkan.

Algoritma Graf Berbasis Pointer

Grafik terdiri dari node (simpul) yang dihubungkan oleh tepi, membentuk struktur yang kompleks. Algoritme berbasis pointer sangat penting untuk traversal grafik dan menyelesaikan masalah terkait grafik secara efisien.

Mari pertimbangkan contoh algoritma grafik berbasis pointer, khususnya pencarian pertama luas (BFS), untuk melintasi grafik:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_VERTICES 100

struct Node
{
   int vertex;
   struct Node* next;
};

struct Graph
{
   struct Node* adjacencyList[MAX_VERTICES];
};
```

```
void addEdge(struct Graph* graph, int src, int dest) {
  struct Node* newNode =
  (struct Node*)malloc(sizeof(struct Node));
  newNode->vertex = dest;
  newNode->next = graph->adjacencyList[src];
  graph->adjacencyList[src] = newNode;

  // Uncomment the code below for an undirected
graph
  /*
  newNode = (struct Node*)malloc(sizeof(struct
Node));
  newNode->vertex = src;
  newNode->next = graph->adjacencyList[dest];
  graph->adjacencyList[dest] = newNode;
  */
}

void bfs(struct Graph* graph, int startVertex)
{
  int visited[MAX_VERTICES] = {0};

  struct Node* Queue = NULL;
  visited[startVertex] = 1;
  struct Node* newNode =
  (struct Node*)malloc(sizeof(struct Node));
  newNode->vertex = startVertex;
  newNode->next = NULL;
  Queue = newNode;

  while (Queue != NULL) {
    int currentVertex = Queue->vertex;
    printf("%d ", currentVertex);
```

```
        struct Node* temp = Queue;
        Queue = Queue->next;
        free(temp);

        struct Node* adjListPtr =
        graph->adjacencyList[currentVertex];
        while (adjListPtr != NULL)
        {
            int adjVertex = adjListPtr->vertex;
            if (visited[adjVertex] == 0)
            {
                visited[adjVertex] = 1;
                newNode =
                (struct Node*)malloc(sizeof(struct Node));
                newNode->vertex = adjVertex;
                newNode->next = NULL;
                if (Queue == NULL)
                {
                    Queue = newNode;
                } else
                  {
                    struct Node* QueuePtr = Queue;
                    while (QueuePtr->next != NULL) {
                        QueuePtr = QueuePtr->next;
                    }
                    QueuePtr->next = newNode;
                }
            }
            adjListPtr = adjListPtr->next;
        }
    }
}

int main()
```

```
{
    struct Graph* graph =
   (struct Graph*)malloc(sizeof(struct Graph));
    for (int i = 0; i < MAX_VERTICES; i++)
  {
        graph->adjacencyList[i] = NULL;
    }

    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 0);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 3);

    printf("BFS traversal starting from vertex 2: ");
    bfs(graph, 2);

    return 0;
}
```

Dalam contoh ini, kita mendefinisikan struktur Node untuk representasi daftar adjacency dari grafik dan struktur Graph untuk menampung daftar adjacency. Fungsi addEdge() menambahkan tepi antara dua simpul dalam grafik. Fungsi bfs() melakukan pencarian luas-pertama mulai dari simpul yang ditentukan dan mencetak urutan penjelajahan (Cormen et al., 2009).

Pointer digunakan untuk menghubungkan simpul dan melintasi grafik secara efisien, memastikan bahwa setiap simpul dikunjungi hanya sekali.

Aplikasi Pointer di Dunia Nyata

Pointer memiliki aplikasi yang luas dalam skenario dunia nyata, memfasilitasi manajemen memori yang efisien dan manipulasi data. Berikut beberapa contohnya (Cormen et al., 2009; Lafore, 2017):

Sistem Operasi: Pointer digunakan dalam sistem operasi untuk mengelola proses, mengalokasikan dan membatalkan alokasi memori, dan mengaktifkan komunikasi antarproses.

Driver Perangkat: Pointer digunakan dalam driver perangkat untuk berinteraksi dengan register perangkat keras dan mengelola buffer memori secara efisien.

Sistem Basis Data: Pointer digunakan dalam sistem basis data untuk menangani kumpulan data besar, mengelola hubungan catatan, dan mengoptimalkan pengambilan dan penyimpanan data.

Pemrosesan Gambar: Pointer digunakan dalam aplikasi pemrosesan gambar untuk memanipulasi piksel, mengakses data gambar secara efisien, dan melakukan berbagai transformasi.

Jaringan: Pointer berperan dalam aplikasi jaringan, di mana mereka digunakan untuk menangani protokol jaringan, pemrosesan paket, dan transmisi data yang efisien.

Dengan memanfaatkan petunjuk dalam aplikasi dunia nyata ini, pengembang dapat mencapai solusi yang efisien dan optimal untuk masalah yang rumit.

Kesimpulan dan Rekap

Dalam bab ini, kita telah menjelajahi konsep pointer dan signifikansinya dalam struktur data. Mari rekap konsep-konsep kunci yang tercakup, pahami pentingnya pointer dalam struktur data, dan sediakan sumber daya

untuk pembelajaran lebih lanjut.

Rangkuman Konsep Kunci

1. Pointer
   Pointer adalah variabel yang menyimpan alamat memori, memungkinkan kita mengakses dan memanipulasi data secara tidak langsung.
2. Representasi dan Alamat Memori
   Pointer menyediakan cara untuk mewakili dan mengakses data yang disimpan dalam memori dengan menyimpan alamat memori dari variabel atau struktur data.
3. Pointer Arithmetic
   Pointer arithmetic memungkinkan kita untuk melakukan operasi aritmatika pada pointer, seperti penambahan, pengurangan, dan mengakses lokasi memori yang berdekatan.
4. Null Pointer dan Dangling Pointer
   Null pointer mengacu pada pointer yang tidak mengarah ke alamat memori yang valid. Penunjuk menggantung terjadi ketika penunjuk menunjuk ke lokasi memori yang telah dibatalkan alokasinya atau tidak lagi menyimpan data yang diinginkan.
5. Kesalahan Pointer dan Debugging
   Kesalahan penunjuk, seperti mengakses lokasi memori yang tidak valid atau membatalkan alokasi memori yang sedang digunakan, dapat menyebabkan crash program dan perilaku yang tidak dapat diprediksi. Berbagai teknik debug, termasuk debugger memori dan peninjauan kode, dapat membantu mengidentifikasi dan memperbaiki kesalahan pointer.
6. Optimasi Berbasis Pointer

Pointer menawarkan peluang untuk mengoptimalkan kode dengan mengaktifkan manajemen memori yang efisien, menghindari penyalinan data yang tidak perlu, dan mengimplementasikan optimalisasi kinerja seperti aritmatika pointer.

Pentingnya Pointer dalam Struktur Data

Pointer sangat penting dalam struktur data karena beberapa alasan:

1. Alokasi Memori Dinamis
   Pointer mengaktifkan alokasi memori dinamis, memungkinkan struktur data tumbuh dan menyusut sesuai kebutuhan selama eksekusi program. Fleksibilitas ini sangat penting untuk menerapkan struktur data dinamis seperti daftar tertaut, pohon, dan grafik.
2. Manipulasi Efisien
   Pointer memfasilitasi manipulasi struktur data yang efisien dengan menyediakan akses langsung ke lokasi memori dan memungkinkan penyisipan, penghapusan, dan operasi traversal yang efisien.
3. Struktur Data Kompleks
   Pointer berperan penting dalam mengimplementasikan struktur data kompleks seperti daftar tertaut, pohon, grafik, dan tabel hash, yang penting untuk memecahkan berbagai masalah.
4. Pengoptimalan Kinerja
   Penggunaan pointer yang tepat dapat menghasilkan pengoptimalan kinerja, seperti menghindari penyalinan data yang tidak perlu, mengurangi beban memori, dan menerapkan algoritme yang efisien untuk operasi struktur data.

Sumber untuk Pembelajaran Lebih Lanjut

Untuk lebih meningkatkan pemahaman Anda tentang pointer dan perannya dalam struktur data, berikut adalah beberapa sumber yang direkomendasikan:

1. Kernighan, B.W., & Ritchie, D.M. (1988). Bahasa Pemrograman C (edisi ke-2). Balai Prentice.
2. Prata, S. (2013). C Primer Plus (edisi ke-6). Addison-Wesley Profesional.
3. Weiss, M.A. (2016). Struktur Data dan Analisis Algoritma di C++. Pearson.
4. Sedgewick, R., & Wayne, K. (2011). Algoritma (edisi ke-4). Addison-Wesley Profesional.
5. Tutorial dan kursus online tentang struktur dan penunjuk data, seperti yang tersedia di situs web seperti Coursera, Udemy, dan YouTube.

Dengan menjelajahi sumber daya ini dan menerapkan konsep dalam latihan pengkodean praktis, Anda dapat memperkuat pengetahuan dan kemahiran Anda dalam bekerja dengan pointer dan struktur data.

BAB 4 VARIABEL DAN KONSTANTA

Pendahuluan

Untuk mengimplementasikan suatu struktur data diperlukan bahasa pemrograman komputer tertentu. Sebagaimana bahasa natural, bahasa yang digunakan dalam pemrograman komputer juga memiliki kumpulan karakter dasar dan berbagai aturan yang berlaku untuk membentuk kata yang pada gilirannya digunakan untuk membentuk pernyataan ketika menulis suatu program komputer.

Suatu program komputer pada umumnya menggunakan berbagai jenis data dan menerapkan cara tertentu untuk menyimpan suatu nilai data. Dalam hal ini nilai data yang dimaksud dapat berupa angka (bilangan) atau karakter. Untuk menyimpan nilai data tersebut, setiap bahasa pemrograman pada dasarnya memiliki pendekatan masing-masing. Namun demikian secara umum terdapat dua opsi yang dapat digunakan untuk menyimpan nilai data, yaitu dengan menggunakan konstanta atau variabel.

Konstanta dan variabel merupakan elemen fundamental dari suatu program komputer. Bahkan sebagian besar orang berpendapat bahwa suatu program komputer pada dasarnya tidak lebih dari sekedar mendefinisikan variabel dan atau konstanta yang kemudian dilakukan manipulasi menggunakan sekumpulan operasi, sesuai keperluan.

Penggunaan variabel dan konstanta dapat mempermudah pengimplementasian struktur data pada

suatu program komputer. Dalam konteks pemrograman, variabel dan konstanta digunakan untuk menyimpan nilai data yang kemudian dapat dilakukan manipulasi sesuai dengan kebutuhan. Dengan demikian penggunaan variabel dan konstanta dapat mempermudah operasi pada data yang pada gilirannya akan memungkinkan suatu program untuk menjadi lebih dinamis dan fleksibel.

Pada bab ini akan diuraikan tentang variabel dan konstanta, dengan referensi utama menggunakan Bahasa Pemrograman C++ (sering juga disingkat sebagai CPP, C-plus plus). Uraian akan diawali dengan penjelasan tentang identifier (pengidentifikasi) dan reserved words (kata-kata khusus yang digunakan oleh Bahasa Pemrogram).

Identifier dan *Reserved Words*

Identifier

Semua variabel dan konstanta yang digunakan dalam pemrograman (apa pun Bahasa Pemrograman yang digunakan) harus dapat dikenali dengan suatu nama yang unik. Nama-nama unik yang digunakan sebagai nama variabel atau konstanta inilah yang disebut sebagai identifier (pengidentifikasi).

Sebenarnya identifier bukan secara khusus digunakan untuk penamaan variabel atau konstanta, akan tetapi digunakan juga memberikan nama terhadap fungsi, kelas, modul, label atau jenis item yang dibuat oleh pengguna (user-defined item). Secara sederhana, dalam konteks pemrograman, identifier dapat diartikan sebagai "nama" (Stroustrup, 1997. hal. 69).

Penamaan identifier memiliki aturan yang berbeda antar Bahasa Pemrograman. Misalnya untuk Bahasa C++ penamaan identifier mengikuti beberapa aturan sebagai berikut (Lee, 2009, hal. 29):

1. Namanya harus dimulai dengan sebuah huruf atau garis bawah (_).
2. Karakter berikuntnya boleh berupa huruf, garis bawah, atau bilangan.
3. Panjang karakter untuk suatu identifier dapat mencapai 200 karakter.
4. Tidak boleh berupa reserved words atau nama lain yang telah digunakan (harus unik).
5. C++ merupakan Bahasa Pemrograman yang case-sensitive, artinya huruf kapital diperlakukan tidak sama dengan huruf kecil. Sehingga "aPertama" berbeda dengan "aPERTAMA".

Berikut adalah beberapa contoh nama identifier:

Sahih (valid): _berkas01, NamaMahasiswa, persPertama.
Tidak sahih (invalid): -berkas02, nama%mahasiswa, pers/1.

Walaupun setiap orang dapat membuat nama apapun untuk suatu identifier, namun sangat disarankan untuk membuat nama yang mudah diingat dan dipahami, misalnya yang memiliki makna tertentu. Menurut Lee (2009) suatu nama identifier sebaiknya deskriptif, sehingga ketika seorang pemrogram membaca kembali program yang telah dikembangkannya dapat dengan mudah mengingat dan memahami kembali berbagai nama identifier yang digunakannya. Misal "fungsiPertama" untuk menunjukkan fungsi pertama yang dieksekusi suatu program, "agregatMikro" untuk merujuk agregasi pada level mikro dan sejenisnya.

Reserved Words

Syarat keempat dari penamaan identifier menyebutkan tidak boleh berupa reserved words, yaitu suatu kata yang sudah "dipesan" atau digunakan oleh

Bahasa Pemrograman untuk keperluan tertentu. Beberapa buku teks menggunakan istilah keywords untuk menyebut reserved words.

Dalam Bahasa Pemrograman C++, misalnya, yang termasuk dalam reserved words antara lain adalah:

1. return: yang digunakan untuk mengakhiri suatu fungsi dan kembali ke perintah pemanggilannya dengan memberikan nilai (jika ada) dari eksekusi fungsi tersebut.
2. short: yang digunakan sebagai modifier untuk jenis data integer yang menyiapkan tempat penyimpanan sebanyak 16 bit.
3. true: merupakan nilai dari Boolean yang neniliki makna benar.
4. float: yang digunakan untuk mendefinisikan data jenis float presisi tunggal.

Variabel

Pengertian Variabel

Dalam dunia pemrograman, variabel dapat dipandang sebagai representasi dari suatu konsep (Lee, 2009, hal. 25-26). Misal seorang pemrogram yang ingin menghitung rata-rata nilai untuk suatu mata kuliah dari suatu kelas di suatu perguruan tinggi, maka sekurang-kurangnya terdapat tiga konsep yang perlu digunakan, yaitu banyaknya mahasiswa, jumlah nilai seluruh mahasiswa, dan rata-rata nilai. Ketiga konsep ini dapat direpresentasikan dengan nMhs, jmlNilai, rataanNilai.

Setiap variabel memiliki tiga komponen informasi terkait, yaitu:

1. Identifier: nama yang dijadikan referensi oleh pemrogram;
2. Type: tipe atau jenis data yang menentukan jenis dan

cara operasi apa yang dapat dilakukan dengan variabelnya; dan

3. Data: nilai data dari variabelnya.

Jadi, sebenarnya variabel merupakan bagian tertentu dari memori di dalam komputer. Sebagaimana diketahui, program komputer pada dasarnya adalah suatu operasi untuk menggunakan dan memanipulasi data. Sementara untuk merujuk suatu data dapat digunakan dua cara: secara literal (nilai datanya, misal 2) dan dengan menggunakan variabel. Untuk merujuk nilai data yang tersimpan dalam variabel, cukup menggunakan nama variabelnya, dan komputer akan mengkases nilai data dari variabel tersebut.

Pada beberapa kasus, penggunaan variabel dapat meningkatkan efisiensi dalam pemrograman. Misal pada baris perintah manipulasi yang dilakukan secara berulang tinggal merujuk nama variabel yang dimanipulasi dan tidak perlu mengganti perintah hanya karena nilai datanya berubah.

Berdasarkan uraian tersebut dapat disimpulkan bahwa variabel adalah tempat penyimpanan data di dalam memori komputer yang dapat diakses melalui namanya. Nilai dari suatu variabel dapat dimanipulasi sesuai dengan kebutuhan. Arinya, variabel merupakan lokasi penyimpanan memori di dalam komputer yang nilainya dapat diubah-ubah sesuai kebutuhan.

Jenis Variabel

Jenis variabel akan mengikuti jenis datanya. Terdapat empat jenis dasar dari suatu variabel, yaitu

1. Variabel Boolean, yaitu variabel yang dapat menyimpan nilai Boolean (true atau false).
2. Variabel character, yaitu variabel yang dapat

menyimpan satu karakter, yang dapat berupa 256 karakter dan simbol ASCII, dan karakter lain dari himpunan karakter dalam extended ASCII.

3. Variabel integer, yaitu variabel yang dapat menyimpan nilai integer dengan berbagai variasinya (signed, unsigned, long, dan short).
4. Variabel floating-points, yaitu variabel yang dapat menyimpan nilai pecahan.

Seperti yang telah dikemukakan sebelumnya, variabel dalam pemrograman komputer memiliki kaitan dengan penggunaan memori di dalam komputer. Setiap variabel yang digunakan dalam pemrograman akan memerlukan tempat penyimpanan sementara di memori komputer. Tabel berikut memberikan gambaran tentang jenis variabel dan ukuran memori yang diperlukan serta kemungkinan nilainya.

Tabel 1. Jenis Variabel, Ukuran Memori, dan Kemungkinan Nilai

Jenis Variabel	**Ukuran Memori**	**Kemungkinan Nilai**
bool	I byte	*true* atau *false*
unsigned short int	2 bytes	0 sampai 65.535
short int	2 bytes	-32.768 sampai 32.767
unsigned long int	4 bytes	0 sampai 4.294967.295
long int	4 bytes	-2.147.483.648 sampai 2.147.483.647
int (16 bit)	2 bytes	-32.768 sampai 32.767
int (32 bit)	4 bytes	–2.147.483.648 sampai 2.147.483.647
unsigned int (16 bit)	2 bytes	0 sampai 65.535
unsigned int (32 bit)	4 bytes	0 sampai 4.294.967.295
char	1 byte	256 nilai karakter
float	4 bytes	1.2e–38 sampai 3.4e38
double	8 bytes	2.2e–308 sampai 1.8e308

Ada kemungkinan ukuran riil variabel pada suatu komputer agak berbeda dengan yang ada di dalam tabel tersebut. Hal ini akan sangat tergantung dai komiler dan komputer yang digunakan.

Mendefinisikan Variabel

Sebelum dapat digunakan dalam pemrograman, suatu variabel perlu didefinisikan terlebih dahulu. Pendefinisian variabel ini dikenal juga dengan istilah deklarasi variabel.

Terdapat dua komponen yang perlu didefinisikan dari suatu variabel, yaitu jenis data dan nama variabelnya. Jenis data yang dimaksudkan dapat berupa jenis data dasar (Boolean, character, integer, atau floating points) atau jenis data lain yang telah didefinisikan. Sementara nama variabel harus sesuai dengan aturan penamaan identifier seperti yang telah diuraikan.

Sebagai contoh suatu program untuk menghitung Body Mass Index (BMI) dengan formula berat badan (dalam kilogram) dibagi dengan kuadrat tinggi badan (dalam m2) akan memerlukan tiga variabel, yaitu BMI, berat, dan tinggi. Contoh program dalam C++ untuk masalah ini antara lain bisa berupa:

```
#include<iostream>
using namespace std;
int main()
{
   float Tinggi, Berat, BMI;
   cout<<"Isikan tinggi badan (dalam cm): ";
   cin>>Tinggi;
   cout<<"Isikan berat badan (dalam kg): ";
   cin>>Berat;
```

```
    BMI = Berat/(Tinggi*Tinggi/10000);
    cout<<"BMI anda adalah: "<<BMI<<endl;
    return 0;
}
```

Pernyataan “float Tinggi, Berat, BMI” merupakan deklarasi bahwa Tinggi, Berat, dan BMI merupakan variabel yang memiliki jenis data floating-points. “cout” merupakan perintah untuk menuliskan di layar dan “cin” merupakan perintah untuk membaca input dari papan ketik (keyboard). Perhatikan bahwa variabel-variabel tersebut harus didefinisikan terlebih dahulu sebelum digunakan dalam berbagai pernyataan atau perintah berikutnya!

Penamaan variabel-variabel dalam contoh program tersebut deskriptif dan mudah dimengerti. Bandingkan dengan penamaan variabel dalam contoh program berikut:

```
#include<iostream>
using namespace std;
int main()
{
    float x, y, z;
    cout<<"Isikan tinggi badan (dalam cm): ";
    cin>>x;
    cout<<"Isikan berat badan (dalam kg): ";
    cin>>y;
    z = y/(x*x/10000);
    cout<<"BMI anda adalah: "<<z<<endl;
    return 0;
}
```

Walaupun urutan pernyataan dan penggunaan variabelnya sama dengan program sebelumnya, untuk

memahami variabel x, y dan z relatif lebih sulit dibandingkan dengan variabel yang menggunakan penamaan deskriptif seperti Tinggi, Berat dan BMI.

Memberikan Nilai ke Variabel

Pemberian nilai ke suatu variabel akan sangat tergantung dari Bahasa Pemrograman. Untuk C++ misalnya dapat dilakukan dengan menggunakan tanda sama dengan "=", sementara untuk Pascal harus menggunakan tanda titik dua sama dengan ":=".

Terdapat tiga cara dasar yang dapat digunakan untuk memberikan nilai terhadap variabel, yaitu

1. Dalam pendeklarasian atau pendefinisian variabel;
2. Menggunakan ekspresi; dan
3. Menggunakan pemanggilan fungsi.

Berikut ini adalah contoh pemberian nilai dalam pendeklarasian vaariabel dalam C++:

```
float Tinggi=172, Berat=68, BMI;
```

Dalam deklarasi ini variabel Tinggi dinyatakan memiliki jenis data float dengan nilai = 178. Begitu juga dengan berat yang memiliki nilai =68. Sementara BMI belum diberikan nilai.

Contoh pemberian nilai melalui ekspresi dalah sebagai berikut:

```
BMI = Berat/(Tinggi*Tinggi/10000);
```

Dalam pernyataan ini variabel BMI diberikan nilai sesuai dengan hasil operasi di sebelah kanannya.

Sedangkan contoh pemberian nilai variabel melalui pemanggilan fungsi adalah sebagai berikut:

```
BMI = fungsiBMI(Tinggi,Berat);
```

di mana fungsiBMI merupakan suatu fungsi yang didefinsikan sebagai berikut:

```
int fungsiBMI(float t,float b)
{
  float nBMI;
  nBMI = b/(t*t/10000);
  return nBMI;
}
```

Konstanta

Sama halnya variabel, konstanta juga merupakan lokasi penyimpanan data. Namun sesuai dengan namanya, nilai dari konstanta tidak berubah atau tetap (konstan). Nilai konstanta harus diberikan pada saat pendefinisian atau pendeklarasian dan tidak dapat diubah setelahnya.

Pendefinisian Konstanta

Dalam C++ untuk mendeklarasikan konstanta digunakan reserved word “const”. Sebagai contoh:

```
const float cTinggi=172, cBerat=68;
```

dalam hal ini cTinggi dan cBerat disefinisikan sebagai konstanta dengan jenis data floating points yang masing-masing memiliki nilai 172 dan 68.

Jadi cara mendefinisikan konstanta mirip dengan mendefinisikan variabel, hanya menambahkan “const” di depannya. Hanya saja perlu diperhatikan, berbeda dengan variabel yang nilainya dapat dimanipualsi setelah dideklarasikan, nilai suatu konstanta tidak dapat diubah

setelah dideklasikan!

Jenis Konstanta

Secara umum terdapat dua jenis konstanta, yaitu konstanta literal dan konstanta simbolis.

Konstanta literal adalah suatu konstanta dalam bentuk nilai secara langsung. Misal 100, 170, “b”, dan sebagainya. Untuk menggunakan konstanta literal tidak perlu di deklarasikan terlebih dahulu.

Konstanta simbolis adalah konstanta yang didefinsikan dengan menggunakan nama identifier tertentu. cTinggi dan cBerat pada contoh pendeklarasian konstanta sebelumnya merupakan contoh dari konstanta simbolis.

BAB 5 *STACK*

Pendahuluan

Stack atau tumpukan merupakan struktur data linier yang mengikuti prinsip LIFO (last in first out), yaitu elemen yang terakhir dimasukkan ke dalam tumpukan merupakan elemen yang pertama kali akan dibuang dari tumpukan. Drozdek (2013, hal. 131) mengilustrasikan struktur data *Stack* sebagai proses penumpukan nampan di suatu kafetaria. Setiap nampan baru akan ditumpukkan di atas tumpukan nampan yang sudah ada dan nampan terakhir di dalam tumpukan merupakan nampan pertama yang dapat diambil dari tumpukan tersebut.

Stack merupakan salah satu struktur data penting, terutama dalam pemrosesan sistem seperti kompilasi dan kontrol program (Balagurusamy, 2019). Bahkan dapat dikatakan kalau *Stack* merupakan salah satu struktur data yag paling penting dan bermanfaat dalam sains komputer (Das, 2006).

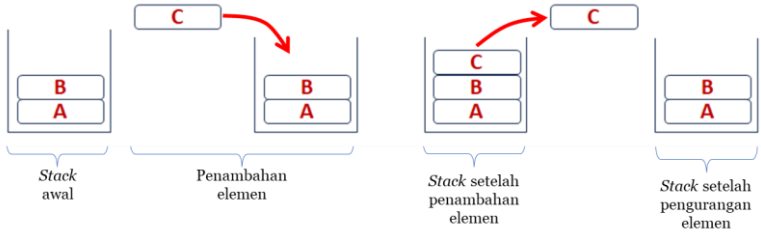
Bab ini akan membahas tentang pengertian, operasi yang dapat dilakukan, dan implementasi struktur data *Stack*.

Pengertian *Stack*

Stack merupakan struktur data linier di mana penambahan elemen baru ke dalam tumpukan dan pengambilan elemen dari tumpukan yang ada hanya dapat dilakukan pada tempat yang sama, yaitu pada bagian

teratas dari tumpukan.

Struktur data *Stack* dapat diilustrasikan melalui gambar berikut:



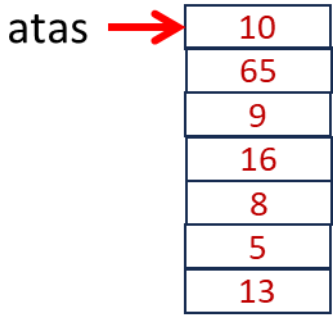
Gambar 8. Ilustrasi Struktur Data *Stack*

Dalam ilustrasi tersebut diandaikan suatu kotak telah berisi dua elemen, yaitu A dan B. Ketika elemen baru C akan ditambahkan, hanya dapat diletakkan di atas B, karena tidak ada cara lain untuk mengakses kotak. Begitu juga ketika tumpukan elemen akan diambil dari dalam kotak hanya elemen yang berada di paling atas yang dapat diambil dari dalam kotak, yaitu C.

Konsep struktur data *Stack* mengikuti analogi yang sama dengan ilustrasi tersebut. Struktur data *Stack* dikenal juga sebagai struktur data LIFO (Das, 2006, hal. 26), karena penambahan elemen ke dalam *Stack* dan pengurangan elemen dari *Stack* mengikuti prinsip LIFO, yaitu *last in first out* (yang secara harafiah dapat diartikan sebagai "terakhir masuk pertama keluar"). Berdasarkan hal ini maka elemen yang paling sering diakses dari suatu *Stack* adalah elemen teratas, sementara yang paling jarang diakses adalah elemen terbawah.

Sebagaimana di dunia nyata, pengelolaan elemen yang disimpan dalam *Stack* pada memori komputer memiliki cara yang sama. *Stack* dalam memori komputer diperlakukan sebagai suatu kelompok elemen yang disimpan pada lokasi yang berdekatan (contiguous) (Balagurusamy, 2019). Representasi logis dari suatu *Stack*

di dalam memori dapat diilustrasikan seperti pada gambar berikut:



Gambar 9. Representasi Logis *Stack* dalam Memori Komputer

Dalam ilustrasi tersebut, *Stack* memiliki 7 elemen. Elemen teratas adalah 10. Setiap elemen disimpan secara berdampingan membentuk suatu blok memori.

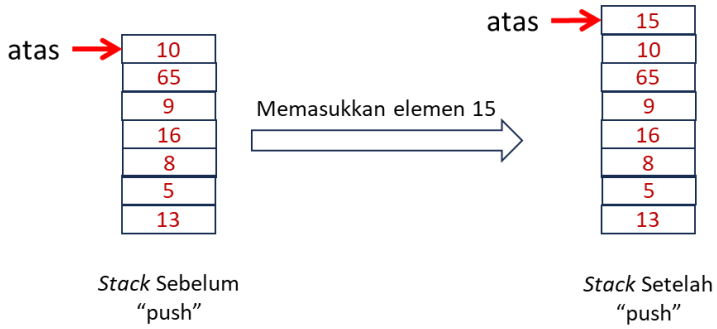
Operasi dalam *Stack*

Terdapat dua operasi kunci dalam struktur data *Stack* (Balagurusamy, 2019; Das, 2006; Drozdek, 2013), yaitu:

1. Operasi “push” atau pemasukan, dan
2. Operasi “pop” atau pengeluaran.

Operasi “push”

Operasi “push” merupakan operasi yang digunakan untuk menambahkan suatu elemen ke dalam *Stack*. Ilustrasi operasi “push” adalah seperti pada gambar berikut:



Gambar 10. Ilustrasi Operasi "push"

Sebelum dilakukan operasi "push", elemen teratas dari *Stack* adalah 10. Ketika elemen dengan nilai 15 di-push atau dimasukkan ke dalam *Stack*, maka posisinya diletakkan di atas 10. Dengan demikian elemen teratas *Stack* setelah operasi "push" adalah 15.

Berdasarkan hal ini maka operasi "push" dalam *Stack* melibatkan beberapa aktivitas berikut:

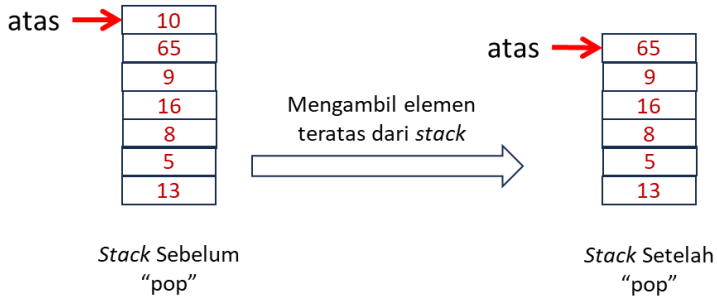
1. Menerima elemen yang akan dimasukkan,
2. Menaikkan pointer *Stack*, yaitu lokasi "atas" atau top,
3. Menyimpan elemen yang diterima pada lokasi "atas".

Untuk pengimplementasian operasi "push" ada kemungkinan akan dihadapi persoalan *Stack* overflow, yaitu suatu kondisi di mana lokasi yang tersedia untuk menyimpan suatu *Stack* sudah penuh. Dalam konteks pemrograman masalah ini perlu ditangani secara memadai, terutama dengan menerapkan mekanisme tertentu agar suatu program dapat memberikan hasil seperti yang diharapkan.

Operasi "pop"

Berkebalikan dengan operasi "push", operasi "pop" merupakan operasi yang digunakan untuk mengambil satu

elemen dari *Stack*. Ilustrasi dari operasi "pop" adalah seperti gambar berikut:



Gambar 11. Ilustrasi Operasi "pop"

Aktivitas dalam operasi "push" adalah sebagai berikut:

1. Mengakses dan menghilangkan elemen teratas pada *Stack*,
2. Menurunkan pointer *Stack*, dan menjadikannya sebagai posisi teratas dalam *Stack*.

Ada kemungkinan operasi "pop" dilakukan pada saat suatu *Stack* tidak memiliki elemen sama sekali atau kosong, Kondisi ini akan menimbulkan kesalahan *Stack* underflow dalam pemrograman. Jadi sangat disarankan pemrogram untuk melakukan pengecekan terlebih dahulu apakah suatu *Stack* masih memiliki elemen atau tidak sebelum melakukan operasi "pop".

Implementasi *Stack*

Sebagian besar Bahasa Pemrograman arus utama seperti C, C++, dan Java telah menyiapkan modul khusus yang dapat digunakan untuk bekerja dengan struktur data *Stack*. Selain menggunakan fasilitas built-in yang telah disiapkan, *Stack* umumnya diimplementasikan pula menggunakan struktur data yang didefinisikan oleh

pengguna seperti array atau list.

Mengingat pembahasan *Stack* ini dilakukan sebelum pembahasan array maupun list, maka pembahasan implementasi *Stack* dalam contoh berikut akan memanfaatkan modul built-in yang tersedia dalam Bahasa Pemrograman C++.

Beberapa hal yang perlu diperhatikan untuk mengimplementasikan struktur data *Stack* dalam C++ antara lain adalah sebagai berikut:

1. Aktifkan modul *Stack* dengan menambahkan pernyataan "#include <*Stack*>" pada header program.
2. Deklarasikan *Stack* dengan pernyataan "*Stack*<jenis-data> nama-*Stack*". "Jenis data" yang dimaksudkan adalah jenis data dari setiap elemen yang disimpan dalam *Stack*. Sementara "nama-*Stack*" yang dimaksudkan adalah identifier atau nama dari *Stack* yang akan digunakan.
3. Pernyataan yang digunakan untuk melakukan operasi "push" adalah "nama-*Stack*.push(nilai-yang-ditambahkan)".
4. Pernyataan yang digunakan untuk melakukan operasi "pop" adalah "nama-*Stack*.pop ()"
5. Pernyataan yang digunakan untuk mengecek apakah suatu *Stack* kosong atau tidak adalah "nama-*Stack*.empty ()". Pernyataan ini akan bernilai true jika *Stack*-nya kosong, dan bernilai false jika ada isian di dalam *Stack*.
6. Elemen *Stack* yang dapat diakses hanya elemen teratas.

Walaupun hal-hal tersebut hanya berlaku untuk C++, pengimplementasian struktur data *Stack* dalam Bahasa Pemrograman lain akan memiliki aturan-aturan yang kurang lebih mirip dengan apa yang telah diuraikan.

Berikut ini adalah contoh implementasi *Stack* dalam C++:

```
#include <iostream>
#include <Stack>
using namespace std;

int main ()
{
  // Stack kosong
  Stack<int> contohStack;
  contohStack.push(1);
  contohStack.push(2);
  contohStack.push(3);

  // mencetak isi Stack
  cout << ' ' << contohStack.top ();
}
```

Dalam program ini:

1. Nama *Stack* yang digunakan adalah "contoh*Stack*" dengan jenis data integer.
2. Pernyataan "contoh*Stack*.push(1)" digunakan untuk menambahkan elemen dengan nilai 1 ke dalam *Stack*. Begitu juga dengan pernyataan push yang lain.
3. Jadi isian *Stack* dari urutan yang terbawah sampai dengan yang teratas adalah 1,2,3.
4. Pernyataan "contoh*Stack*.top()" digunakan untuk mengakses nilai elemen teratas dari *Stack*.
5. Output dari program tersebut adalah 3, karena elemen terakhir yang dimasukkan ke dalam *Stack* (elemen teratas) adalah 3.

   Berikut adalah contoh implementasi operasi "pop"

dalam *Stack*:

```
#include <iostream>
#include <Stack>
using namespace std;

int main ()
{
   // Stack kosong
   Stack<int> contohStack;
   contohStack.push(1);
   contohStack.push(2);
   contohStack.push(3);

   // mencetak isi Stack
   contohStack.pop ();
   cout << ' ' << contohStack.top ();
}
```

Pernyataan "contoh*Stack*.pop ()" digunakan untuk menghilangkan atau membuang elemen teratas dari *Stack*. Sehingga output dari program tersebut adalah 2, karena elemen teratas *Stack* telah dihilangkan.

Baik operasi "push" maupun "pop" dapat digunakan dalam pengulangan. Sebagai contoh program untuk mengisi suatu *Stack* dengan bilangan 1 sampai dengan 9, kemudian mencetak seluruh isian *Stack* tersebut antara lain dapat digunakan program sebagai berikut:

```
#include <iostream>
#include <Stack>
using namespace std;

int main ()
```

```
{
   // Stack kosong
   Stack<int> aStack;
   int n=0;

   //mengisi Stack dengan push
   while (n<9) {
      n=n+1;
      aStack.push(n);
   }

   // mencetak seluruh isi Stack
   while (!aStack.empty ()) {
      cout << ' ' << aStack.top ();
      aStack.pop ();
   }
}
```

Output dari program ini adalah “9 8 7 6 5 4 3 2 1”. Ingat bahwa elemen *Stack* yang dapat diakses hanyalah elemen teratas, sehingga ketika mencetak akan diperoleh urutan terbalik dari pada saat penyimpanan elemennya.

Jika diinginkan untuk mencetak elemen dari suatu *Stack* dengan urutan mulai dari bawah sampai dengan yang teratas maka dapat digunakan pendekatan recursive seperti pada contoh program berikut:

```
#include <iostream>
#include <Stack>
using namespace std;

void cetakStack(Stack<int> aStack) {
   // jika Stack kosong kembali
   if (aStack.empty())
```

```
        return;
    int x = aStack.top ();
    // Pop the top element of the Stack
    aStack.pop ();

    // memanggil cetakStack secara recursive
    cetakStack(aStack);
    // mencetak Stack dari bawah
    cout << x << " ";

    // elemen dimasukkan kembali ke dalam Stackr
    aStack.push(x);
  }

  int main() {
    // Stack kosong
    Stack<int> aStack;
    int n=0;

    //mengisi Stack dengan push
    while (n<9) {
       n=n+1;
       aStack.push(n);
    }
   cetakStack(aStack);
  }
```

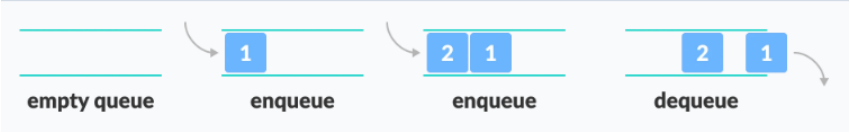
Output dari program tersebut adalah “1 2 3 5 6 7 8 9”.

BAB 6 *QUEUE*

Pendahuluan

Setelah tumpukan, abstraksi data paling sederhana berikutnya adalah antrian (*Queue*). Seperti halnya tumpukan, antrian dapat divisualisasikan dengan banyak contoh yang bisa dikenali dari kehidupan sehari-hari. Ilustrasi sederhananya adalah antrian orang yang menunggu masuk ke dalam teater. Hal yang membuat antrian berbeda dari tumpukan adalah antrian terbuka di kedua ujungnya Gambar 12. Properti dasar dari antrian adalah bahwa item disisipkan di salah satu ujung (bagian belakang array) dan dihapus dari yang lain. Ini berarti bahwa urutan item yang dihapus cocok dengan urutan yang dimasukkan. Sama seperti tumpukan yang dideskripsikan sebagai wadah LIFO (last-in, first-out), ini berarti antrian dapat dideskripsikan sebagai FIFO (first in, first out)

Antrian mengikuti aturan FIFO - yaitu identita data yang dimasukkan lebih awal akan diproes mendahului data lainya. Data dimasukkan ke dalam antrian melalui satu ujung dan dihapus darinya array menggunakan ujung lainnya.



Gambar 12. Ilustrasi proses antrian

Contoh antrian dalam kehidupan sebenarnya adalah jalur lalulintas satu arah, di mana kendaraan yang masuk

ke jalur lebih dahalu, akan keluar jalur lebih dahulu pula. Contoh dunia nyata berikutnya adalah antrian pembayaran di loket tiket pesawat dan tiket kapal laut.

Penerapan antrian dalam berbagai system berbasis konvensional atau komputer membantu kinerja sistem menjadi lebih teratur dan mudah dalam melakukan pengawasan, perbaikan, dan evalauasi. Oleh karena itu antrian banyak diterapkan dalam sistem yang memiliki kompleksitas tinggi maupun rendah.

Operasi Dasar Antrian

Antrian yang dimaksud dalam struktur data adalah objek atau lebih spesifik dikenal sebagai struktur data abstrak (ADT) yang memungkinkan lima tahapan proses seperti di tampilkan pada Gambar 12, yaitu:

1. En*Queue* yaitu menambahkan elemen ke akhir antrian (array). Artinya setiap penambahan elemen baru harus diletakkan pada posisi akhir antrian (array).
2. De*Queue* yaitu menghapus elemen dilakukan dari posisi terdepan di antrian.
3. IsEmpty yaitu memeriksa array antrian apakah tidak terisi
4. IsFull yaitu memeriksa array antrian apakah penuh
5. Peek yaitu mendapatkan nilai antrian pada posisi depan tanpa menghapusnya

Kerja dari Antrian

Beberapa istilah dalam antrian yang harus dipahami adalah sebagai berikut:

1. Petunjuk operasi yaitu FRONT dan REAR
2. FRONT petunjuk posisi elemen pertama dalam antrian

3. REAR petunjuk posisi elemen di akhir dari antrian awalnya,
4. Memberi inisialisasi nilai awal FRONT dan REAR dengan -1

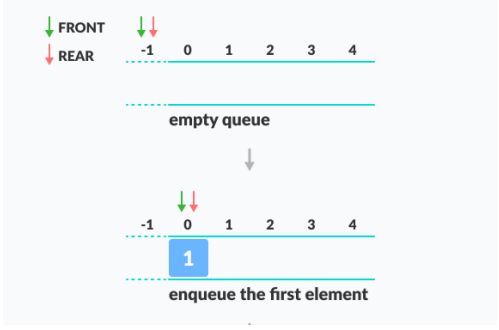
Prinsip Kerja En*Queue*

1. Memeriksa apakah antrian penuh
2. Elemen pertama, FRONT ditempatkan di indek ke 0
3. Tingkatkan indeks REAR sebanyak 1
4. Menambahkan data baru pada posisi yang ditunjukkan oleh REAR

Prisip Kerja De*Queue*

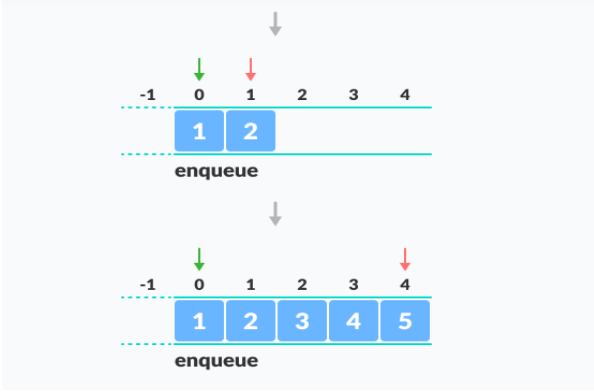
1. Memeriksa jika antrian terdapat kekosongan
2. Mengembalikan nilai yang ditunjukkan oleh FRONT
3. Naikkan indeks FRONT sebanyak 1
4. Pada posisi terakhir, melakukan setel ulang nilai FRONT dan REAR ke -1

Petunjuk operasi FRONT dan REAR berada pada -1 yang menandakan kondisi awal operasi. Kemudian index operasi ditigkatkan 1 maka FRONT dan REAR berada pada posisi 0. Pada posisi ini bisa memasukkan data baru. Pada posisi ini Front dan Rear masih dalam lokasi yang sama dalam antrian seperti di tunjukkan oleh Gambar 13.



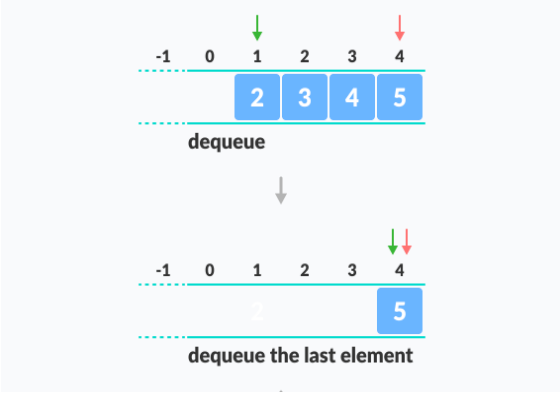
Gambar 13. Ilustrasi proses antrian

Penambahan data baru 2 dalam antrian akan ditempatkan dibelakang elemen 1. Pada kondisi ini maka Front dan Rear berbeda lokasi. Front berada di index 0 dan Rear berada di index 1. Semakin banyak jumlah data yang di antrikan maka Rear akan semakin jauh dari Front seperti ditunjukkan pada Gambar 14.



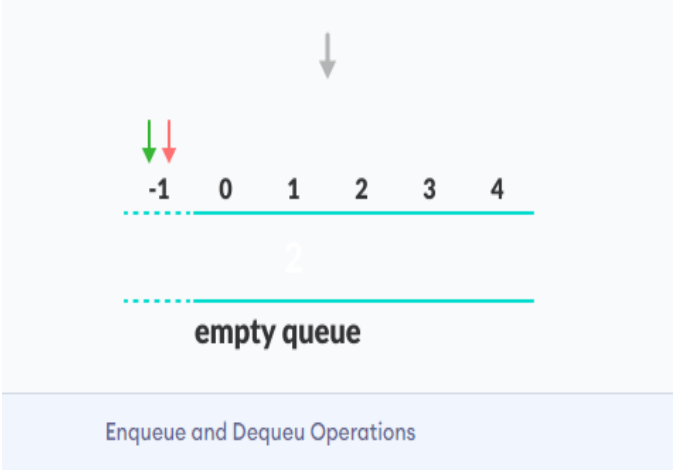
Gambar 14. Ilustrasi proses antrian

Sementara itu untuk mengeluarkan elemen data dari antrian mematuhi kaidah FIFO. Oleh karena itu elemen data 1 akan menempati prioritas utama untuk keluar. Proses pengeluaran data dilakukan sesuai dengan urutan antrian sampai seluruh elemen hilang dari posisi antrian seperti di tunjukkan pada Gambar 15.



Gambar 15. Ilustrasi proses antrian

Ketika seluruh elemen data keluar dari antrian maka kolom antrian akan menjadi kosong dan posisi Front dan Rear kembali pada index -1 seperti di tunjukkan pada Gambar 16.



Gambar 16. Prinsip kerja *Queue*

Algoritma menyisipkan (Insert)

1. memberi nilai awal FRONT = – 1; REAR = 1
2. REAR = (REAR + 1) % SIZE
3. Jika (FRONT sama dengan REAR)
   Tampilkan pesan “Antrian Penuh”
   Kelur
4. Selain itu
   Memasukkan nilai dan kemudian tetapkan ke variabel sebagai “DATA”
5. jika (FRONT sama dengan – 1)
   FRONT = 0
   REAR = 0
6. Q [REAR] = DATA
7. Ulang langka 2 sampai 5 untuk menyisipkan yang baru
8. Exit

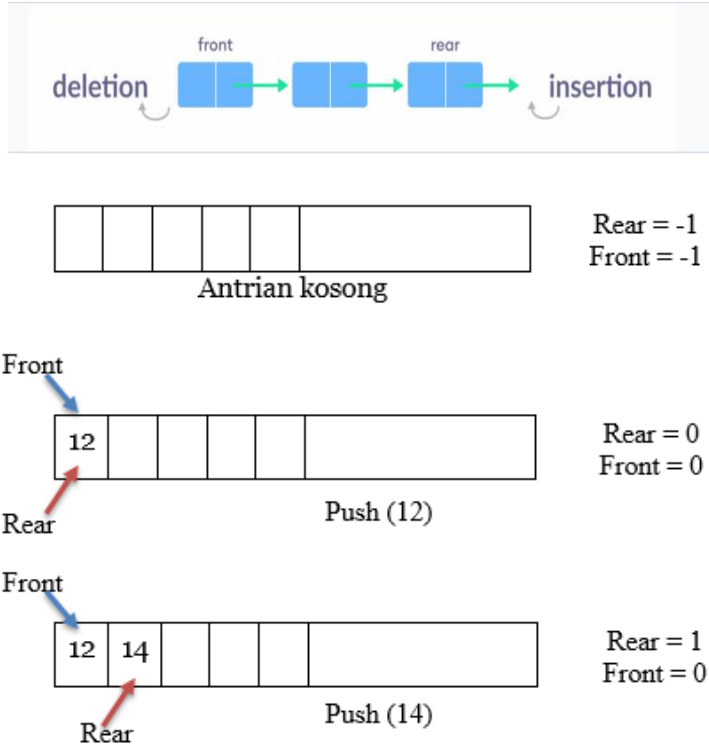
Algoritma menghapus (Delete)

1. jika (FRONT sama dengan – 1)
   Tampilkan pean “Antrian kosong”
   Keluar
2. Selain itu
   DATA = Q [FRONT]
3. jika (REAR sama dengan FRONT)
   FRONT = –1
   REAR = –1
4. Selain itu
   FRONT = (FRONT +1) % SIZE
5. Ulang langkah 1,2, dan 3 untuk menghapus selanjutnya
6. Exit

Tipe *Queue*

Queue dalam struktur data memiliki beberapa model yang dapat digunakan untuk pemrosesan data. Model yang diterapkan terdapat empat jenis yaitu:

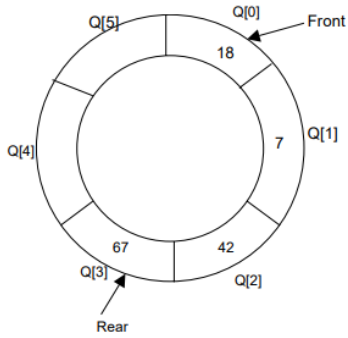
1. Simple *Queue*
   Pada simple *Queue*, penyisipan dilakukan di bagian belakang dan pemindahan terjadi di bagian depan. Ini dilakukan dengan cara ketat mengikuti aturan FIFO. Pada Gambar 6 diilustrasikan penyisipan elemen pada antrian kosong dengan angka 12. Angka 12 diletakkan pada posisi Front (0) dan Rear (0). Sementara itu penyisipan kedua angka 14 diletakan di belakang angka 12. Penyisipan berikutnya jika ada dilakukan dengan menempatkan elemen baru di belakang antrian terakhir. Proses penyisipan pada elemen pada terakhir antrian ini dikenal dengan Push().



Gambar 17. Simple *Queue*

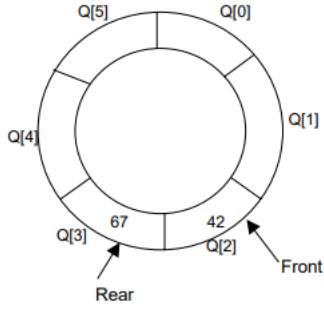
2. Circular *Queue*
   Antrian melingkar dinotasikan dengan simbol Q(x). Dalam antrian ini ada sejumlah elemen dan dituliskan dengan Q [0], Q [1], Q [2]....Q [n - 1] yang direpresentasikan secara melingkar dengan Q [1] dan mengikuti Q [n], sementara itu elemen dituliskan dengan Q(x). Antrian melingkar adalah antrian dimana penyisipan elemen baru dilakukan di lokasi pertama dari antrian jika lokasi terakhir pada antrian sudah penuh.
   Sebuah antrian melingkar dengan jumlah 5 array dengan perincian Q[0] = 18, Q[1] = 7, Q[2] = 42, Q[3] = 67, Q[4] dan Q [5] kosong maka Front berada

di index ke 0 dan Rear berada di index ke 3. Komposisi antrian ini dapat ditunjukkan seperti pada Gambar 18.



Gambar 18. Antrian melingkar (circular *Queue*)

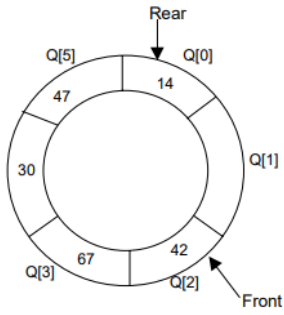
Jika dilakukan POP untuk dua element Q (18, 7) maka circular *Queue* menjadi seperti pada gambar 8. *Queue* POP adalah perintah yang digunakan untuk memindah element dalam antrian. *Queue* POP yang sering dituliskan sebagai POP saja. Setela dilakukan POP pada Q(18, 7) maka elemen pada Q[0] dan Q[1] menjadi kosong.



Gambar 19. Proses POP pada antrian melingkar

Jika dilakukan PUSH dengan tiga element Q (30, 47 dan 14) terhadap antrian pada Gambar 8 maka circular *Queue* menjadi seperti pada gambar 9. Perlu

di ingat proses menyisiokan pada antrian adalah pada posisi terakhir elemen. Oleh karena itu PUSH Q(30, 47 dan 14) akan menenpati index Q[4], Q[5] dan Q[0] secara berurutan. Hasil proses PUSH yaitu: Q[4] = 30, Q[5]=47 dan Q[0] = 14 seperti ditunjukkan oleh Gambar 20.



Gambar 20. Proses PUSH pada antrian melingkar

Aplikasi tipe antrian melingkar dapat diterapkan untuk proses kegiatan sebagai berikut:

a. Penjadwalan CPU
b. Pengaturan Memory
c. Pengaturan lalulintas data

3. Priorty *Queue*

Antrian prioritas adalah jenis antrian yang mengatur elemen berdasarkan nilai prioritasnya. Elemen dengan nilai prioritas lebih tinggi biasanya diambil sebelum elemen dengan nilai prioritas lebih rendah.

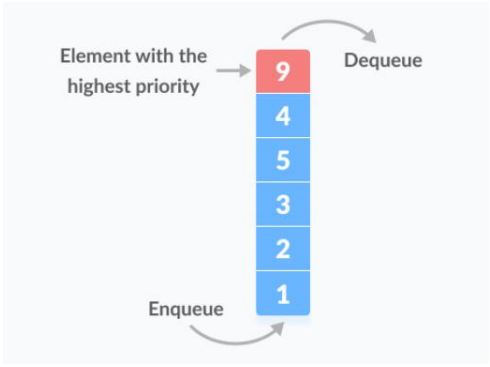
Dalam antrian prioritas, setiap elemen memiliki nilai prioritas yang terkait dengannya. Saat Anda menambahkan elemen ke antrean, elemen tersebut dimasukkan ke dalam posisi berdasarkan nilai prioritasnya. Misalnya, jika Anda menambahkan elemen dengan nilai prioritas tinggi ke antrean prioritas, elemen tersebut dapat disisipkan di dekat bagian depan antrean, sedangkan elemen dengan

nilai prioritas rendah dapat disisipkan di dekat bagian belakang.

Ada beberapa cara untuk mengimplementasikan antrian prioritas, termasuk menggunakan array, linked list, heap, atau binary search tree. Setiap metode memiliki kelebihan dan kekurangannya sendiri, dan pilihan terbaik akan bergantung pada kebutuhan spesifik aplikasi Anda.

Antrean prioritas sering digunakan dalam sistem real-time, di mana urutan pemrosesan elemen dapat memiliki konsekuensi yang signifikan. Mereka juga digunakan dalam algoritme untuk meningkatkan efisiensinya, seperti algoritme Dijkstra untuk menemukan jalur terpendek dalam graf dan algoritme pencarian A* untuk menemukan jalur.

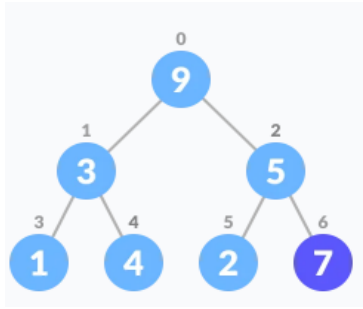
Antrian prioritas adalah jenis antrian khusus di mana setiap elemen dikaitkan dengan prioritas dan disajikan sesuai dengan prioritasnya. Jika elemen dengan prioritas yang sama terjadi, mereka disajikan sesuai dengan urutannya dalam antrian. Umumnya, nilai elemen itu sendiri dipertimbangkan untuk menetapkan prioritas. Misalnya, elemen dengan nilai tertinggi dianggap sebagai elemen dengan prioritas tertinggi. Namun, dalam kasus lain, kita dapat mengasumsikan elemen dengan nilai terendah sebagai elemen prioritas tertinggi. Dalam kasus lain, kita dapat menetapkan prioritas sesuai dengan kebutuhan kita. Contoh penerapan prioritas antrian adalah sebuah array yang terdiri dari beberapa nilai dan dinyatakan dalam Q[9,4,5,3,2,1]. Prioritas nilai tertinggi ditetapkan sebagai antrian terdepan. Untuk menggambarkan kasus ini dapat dilihat pada Gambar 21.



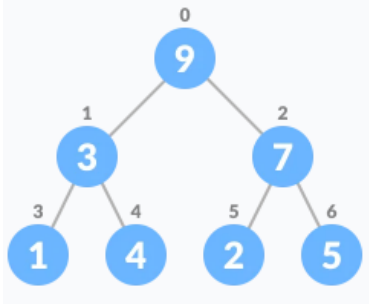
Gambar 21. Prioritas antrian

Operasi Antrian Prioritas:
Antrean prioritas tipikal mendukung operasi berikut:

a. Penyisipan dalam Antrian Prioritas
   Saat elemen baru dimasukkan ke dalam antrian prioritas, elemen tersebut berpindah ke slot kosong dari atas ke bawah dan dari kiri ke kanan. Namun, jika elemen tersebut tidak berada di tempat yang benar maka akan dibandingkan dengan node induknya. Jika elemen tidak dalam urutan yang benar, elemen ditukar. Proses penukaran berlanjut hingga semua elemen ditempatkan pada posisi yang benar.
   Memasukkan data baru ke dalam antrian prioritas (max-heap) dengan urutan proses seperti diperlihatkan oleh Gambar 22 dan Gambar 23.



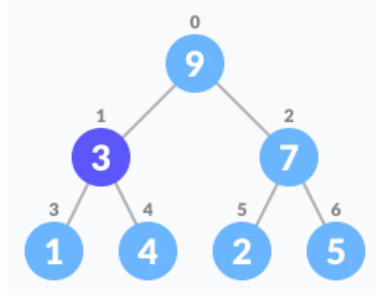
Gambar 22. Penyisipan elemen di antrian prioritas



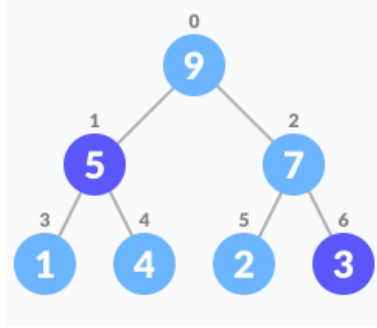
Gambar 23. Tumpukan elemen setelah penyisipan elemen

b. Penghapusan dalam Antrean Prioritas
Seperti yang Anda ketahui bahwa dalam tumpukan maksimum, elemen maksimum adalah simpul akar. Dan itu akan menghapus elemen yang memiliki prioritas maksimum terlebih dahulu. Dengan demikian, Anda menghapus simpul akar dari antrian. Penghapusan ini membuat slot kosong, yang selanjutnya akan diisi dengan penyisipan baru. Kemudian, membandingkan elemen yang baru disisipkan dengan semua elemen di dalam antrian untuk mempertahankan invarian heap.

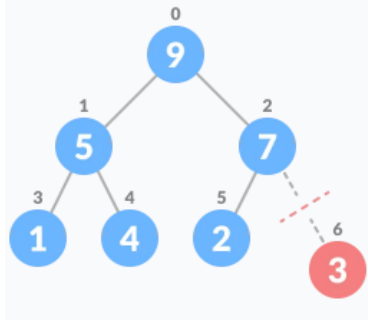
Menghapus data dari antrian prioritas (max-heap) dengan langkah-langkah seperti diperlihatkan oleh Gambar 24.



Gambar 24. Memilih elemen untuk dihapus



Gambar 25. Memindahkan elemen ke akhir



Gambar 26. Menghapus elemen

c. Mengintip Antrean Prioritas
Operasi ini membantu mengembalikan data

maksimum dari Max-Heap atau data minimum dari Min-Heap tanpa menghapus simpul dari antrian prioritas.

Jenis Antrian Prioritas:

a. Antrian Prioritas Urutan Ascending
   Seperti namanya, dalam antrian prioritas urutan menaik, elemen dengan nilai prioritas lebih rendah diberikan prioritas lebih tinggi dalam daftar prioritas. Misalnya, jika kita memiliki elemen berikut dalam antrian prioritas yang disusun dalam urutan menaik seperti 4,6,8,9,10. Di sini, 4 adalah angka terkecil, oleh karena itu, ia akan mendapatkan prioritas tertinggi dalam antrean prioritas dan saat kita keluar dari antrean prioritas jenis ini, 4 akan dihapus dari antrean dan de*Queue* mengembalikan 4.
b. Antrian Prioritas Urutan Menurun
   Simpul akar adalah elemen maksimum dalam tumpukan maksimum, seperti yang mungkin Anda ketahui. Itu juga akan menghapus elemen dengan prioritas tertinggi terlebih dahulu. Akibatnya, simpul akar dihapus dari antrian. Penghapusan ini menyisakan ruang kosong, yang akan diisi dengan sisipan baru di masa mendatang. Invarian heap kemudian dipertahankan dengan membandingkan elemen yang baru dimasukkan ke semua entri lain dalam antrean.

Antrean prioritas dapat diimplementasikan menggunakan struktur data berikut:

a. Array
b. Daftar tertaut
c. Struktur data tumpukan

d. Pohon pencarian biner
e. Aplikasi Antrian Prioritas:
f. Penjadwalan CPU
g. Algoritme grafik seperti algoritma jalur terpendek Dijkstra, Pohon Rentang Minimum Prim, dll.
h. Implementasi Tumpukan
i. Semua aplikasi antrean yang melibatkan prioritas.
j. Kompresi data dalam kode Huffman
k. Simulasi berbasis peristiwa seperti pelanggan menunggu dalam antrean.
l. Mencari Kth elemen terbesar/terkecil.

Keuntungan Antrian Prioritas:

a. Ini membantu untuk mengakses elemen dengan cara yang lebih cepat. Ini karena elemen dalam antrian prioritas diurutkan berdasarkan prioritas, seseorang dapat dengan mudah mengambil elemen dengan prioritas tertinggi tanpa harus mencari di seluruh antrian.
b. Pengurutan elemen dalam Priority *Queue* dilakukan secara dinamis. Elemen-elemen dalam antrean prioritas dapat diperbarui nilai prioritasnya, yang memungkinkan antrean untuk mengurutkan ulang secara dinamis saat prioritas berubah.
c. Algoritma yang efisien dapat diimplementasikan. Antrean prioritas digunakan dalam banyak algoritme untuk meningkatkan efisiensinya, seperti algoritme Dijkstra untuk menemukan jalur terpendek dalam graf dan algoritme pencarian A* untuk menemukan jalur.

d. Termasuk dalam sistem real-time. Ini karena antrean prioritas memungkinkan Anda mengambil elemen dengan prioritas tertinggi dengan cepat, antrean tersebut sering digunakan dalam sistem waktu nyata di mana waktu adalah hal yang sangat penting.

Kerugian Antrian Prioritas:

a. Kompleksitas tinggi
   Antrean prioritas lebih kompleks daripada struktur data sederhana seperti larik dan daftar tertaut, dan mungkin lebih sulit untuk diterapkan dan dipelihara.
b. Konsumsi memori yang tinggi
   Menyimpan nilai prioritas untuk setiap elemen dalam antrian prioritas dapat menghabiskan memori tambahan, yang mungkin menjadi perhatian dalam sistem dengan sumber daya terbatas.
c. Itu tidak selalu merupakan struktur data yang paling efisien
   Dalam beberapa kasus, struktur data lain seperti tumpukan atau pohon pencarian biner mungkin lebih efisien untuk operasi tertentu, seperti menemukan elemen minimum atau maksimum dalam antrean.
d. Kadang-kadang kurang dapat diprediksi
   Ini karena urutan elemen dalam antrean prioritas ditentukan oleh nilai prioritasnya, urutan pengambilan elemen mungkin kurang dapat diprediksi dibandingkan dengan struktur data lain seperti tumpukan atau antrean, yang mengikuti first-in, first-out (FIFO) atau last-in, first-out (LIFO).

Perbedaan antara antrian melingkar dan antrian prioritas adalah sebagai berikut:

Antrian Meligkar	Antrian Prioritas
Antrian melingkar tidak linier tetapi melingkar	Prioritas adalah jenis struktur data khusus di mana item dapat disisipkan atau dihapus berdasarkan prioritas.
Ini juga disebut sebagai buffer cincin.	Ini juga disebut antrian sederhana.
Item dapat dimasukkan atau dihapus dari antrian dalam O(1) waktu.	Itu dapat melakukan tiga operasi seperti insert delete dan display.
Baik pointer depan dan belakang membungkus ke awal array.	Itu tidak mengizinkan elemen dalam array yang diurutkan
Ini mengatasi masalah antrian linier	Ini memungkinkan elemen duplikat.
Itu membutuhkan lebih sedikit memori.	Itu membutuhkan lebih banyak memori
Lebih hemat	Kurang efisien.

4. De*Queue*

   Double-ended *Queue* (de*Queue* atau deque) adalah tipe data abstrak yang menggeneralisasi antrian, yang elemennya dapat ditambahkan atau dihapus dari depan atau belakang. Deque berbeda dengan tipe data abstrak antrian atau First-In-First-Out List (FIFO), di mana elemen hanya dapat ditambahkan ke satu ujung dan dihapus dari ujung lainnya. Kelas data umum ini memiliki beberapa kemungkinan subtipe:

   a. Deque yang dibatasi input adalah deque yang

dapat dihapus dari kedua ujungnya, tetapi penyisipan hanya dapat dilakukan di salah satu ujungnya.

b. Deque yang dibatasi keluarannya adalah salah satu tempat penyisipan dapat dilakukan di kedua ujungnya, tetapi penghapusan hanya dapat dilakukan dari satu ujung saja.

Ilustrai model yang menggambarkan mode deque seperti ditunjukkan pada Gambar 27.

Gambar 27. Tipe data deque

Langkah-langkah untuk en*Queue*:

a. Mengecek apakah antrian sudah penuh atau belum
b. Jika sudah penuh, mencetak luapan dan keluar
c. Jika antrian tidak penuh, tambahkan ujung (Rear) dan tambahkan elemen

Langkah-langkah untuk de*Queue*:

a. Memeriksa antrian apakah kosong atau tidak
b. Jika kosong, mencetak underflow dan keluar
c. Jika tidak kosong, cetak elemen di kepala (Front) dan increment Front

Algoritma Menyisipkan Elemen di Front

1. Masukkan DATA yang akan disisipkan
2. Jika ((left == 0 && right == MAX–1) || (left == right + 1))

(a) Tampilkan pesan “Antrain melimpah”
(b) keluar

3. Jika (left == –1)
   (a) maka left = 0
   (b) maka right = 0
4. Selain itu
   (a) Jika (right == MAX –1)
      (i) left = 0
   (b) Selain itu
      (i) right = right+1
5. Q[right] = DATA
6. Keluar

Algoritma Menyisipkan Elemen di Rear

1. Masukkan DATA yang akan di sisipkan
2. Jika ((left == 0 && right == MAX–1) || (left == right+1))
   (a) Tapilkan “Antrian Melimpah”
   (b) Keluar
3. Jika (left == – 1)
   (a) Left = 0
   (b) Right = 0
4. Selain itu
   (a) if (left == 0)
      (i) left = MAX – 1
   (b) Selain itu
      (i) left = left – 1
5. Q[left] = DATA
6. Keluar

Algoritma Menghapus Elemen di Front

1. Jika (left == – 1)
   (a) Tapilkan “Antrian Melimpah”

(b) Keluar
2. DATA = Q [right]
3. Jika (left == right)
   (a) left = – 1
   (b) right = – 1
4. Selain itu
   (a) if(right == 0)
      (i) right = MAX-1
   (b) Selain itu
      (i) right = right-1
5. Keluar

Algoritma Menghapus Elemen di Rear

1. Jika (left == – 1)
   (a) Tapilkan "Antrian Melimpah"
   (b) Keluar
2. DATA = Q [left]
3. Jika (left == right)
   (a) left = – 1
   (b) right = – 1
4. Selain itu
   (a) if (left == MAX-1)
      (i) left = 0
   (b) Selain itu
      (i) left = left +1
5. Keluar

Aplikasi *Queue*

1. Aplikasi pada pembagian sumber daya pada konsumen. Contohnya penjadwalan CPU, Penjadwalan Disk
2. Implementasi message *Queue* pada sistem presensi kehadiran berbasis digital.

3. Sistem antrian pencetakan dokumen pada perangkat printer.
4. Data ditransfer secara asinkron antara dua proses. Contohnya termasuk Buffer IO, file IO.
5. Penanganan proses prioritas tinggi dalam sistem operasi ditangani menggunakan antrian.
6. Saat Anda mengirim pesan di media sosial, pesan tersebut dikirim ke antrian ke server.
7. Aplikasi traversal dalam grafik: Breadth First Search menggunakan antrian untuk menyimpan node yang perlu diproses.
8. Antrian digunakan dalam menangani komunikasi asinkron antara dua aplikasi yang berbeda atau dua proses yang berbeda.

BAB 7 ARRAY

Pendahuluan

Dalam dunia pemrograman, struktur data array merupakan salah satu konsep yang paling dasar dan penting. Array adalah struktur data yang digunakan untuk menyimpan kumpulan elemen dengan tipe data yang sama. Setiap elemen dalam array memiliki posisi yang disebut indeks, yang digunakan untuk mengakses dan memanipulasi nilai elemen tersebut.

Array sering kali digunakan untuk menyimpan dan mengelola kumpulan data dalam pemrograman. Dalam array, elemen-elemen disusun secara berurutan dan dapat diakses dengan menggunakan indeks yang dimulai dari 0. Misalnya, dalam array integer dengan 5 elemen, indeksnya berkisar dari 0 hingga 4 (Visser, 2006).

Kelebihan utama dari struktur data array adalah kemampuannya untuk menyimpan dan mengakses data secara efisien. Dengan menggunakan indeks, kita dapat langsung mengakses nilai elemen tertentu tanpa perlu mencari melalui seluruh koleksi data. Hal ini membuat array sangat berguna dalam situasi di mana kita perlu mengakses dan memanipulasi elemen-elemen secara berulang (Suhendar, 2019).

Selain itu, array juga memungkinkan kita untuk melakukan operasi matematis, pengurutan, pencarian, dan banyak operasi lainnya dengan mudah. Kita dapat memodifikasi nilai elemen dalam array, menambahkan elemen baru, menghapus elemen, atau mengganti nilai elemen yang ada.

Deklarasi dan Penggunaan Array

Deklarasi dan penggunaan array merupakan konsep penting dalam pemrograman. Array adalah struktur data yang digunakan untuk menyimpan kumpulan elemen dengan tipe data yang sama. Dalam array, setiap elemen memiliki posisi yang disebut indeks, yang digunakan untuk mengakses elemen tersebut. Proses deklarasi dan penggunaan array melibatkan beberapa langkah penting (Efendi, 2022).

1. Deklarasi Array

   Dalam bahasa pemrograman, array perlu dideklarasikan sebelum digunakan. Proses deklarasi array melibatkan menentukan tipe data elemen dan ukuran array. Proses deklarasi array juga melibatkan menyediakan tipe data elemen array dan menentukan ukuran array (jumlah elemen). Setelah deklarasi, array dapat diisi dengan nilai dan elemen-elemennya dapat diakses menggunakan indeks numerik.

   Pseudocode:

   ```
   DECLARE array AS Integer Array
   SET array[0] = 5
   SET array[1] = 10
   SET array[2] = 15
   PRINT array[1]  // Output: 10
   ```

   Sebagai contoh, untuk mendeklarasikan array integer dengan nama "numbers" yang memiliki 5 elemen, kita dapat menuliskan:

   Java

   ```
   int[] numbers = new int[5];
   ```

Dalam contoh di atas, `int[]` menunjukkan bahwa kita mendeklarasikan array integer, dan `new int[5]` mengalokasikan memori untuk array dengan 5 elemen.

2. Inisialisasi Array
   Setelah array dideklarasikan, kita dapat menginisialisasi nilai awal untuk setiap elemennya. Ada beberapa cara untuk menginisialisasi array. Misalnya, kita dapat memberikan nilai langsung saat mendeklarasikan array, seperti:
   Java
   ```
   int[] numbers = {2, 4, 6, 8, 10};
   ```

   Dalam contoh di atas, kita menginisialisasi array "numbers" dengan 5 elemen yang bernilai 2, 4, 6, 8, dan 10.
   Alternatifnya, kita dapat mengisi setiap elemen array secara terpisah menggunakan indeks, seperti:
   Java
   ```
   int[] numbers = new int[5];
   numbers[0] = 2;
   numbers[1] = 4;
   numbers[2] = 6;
   numbers[3] = 8;
   numbers[4] = 10;
   ```

   Dalam contoh ini, kita menginisialisasi setiap elemen array secara terpisah dengan nilai yang diinginkan.
3. Mengakses Elemen Array
   Setelah array diinisialisasi, kita dapat mengakses setiap elemennya menggunakan indeks. Indeks dimulai dari 0 dan berakhir pada (ukuran array - 1) (Zein & Eriana, 2022). Misalnya, untuk mengakses

elemen pertama dari array "numbers", kita dapat menuliskan:
Java
int firstElement = numbers[0];

Dalam contoh ini, nilai dari elemen pertama (indeks 0) akan disimpan dalam variabel "firstElement".
Kita juga dapat menggunakan indeks untuk mengubah nilai elemen yang sudah ada. Misalnya, untuk mengubah nilai elemen kedua dari array "numbers" menjadi 12, kita dapat menuliskan:
Java
numbers[1] = 12;

4. Menggunakan Loop untuk Memanipulasi Array
   Looping merupakan teknik yang umum digunakan untuk memanipulasi elemen-elemen array (Tarigan, 2022). Misalnya, perulangan `for` dapat digunakan untuk mengakses dan memproses setiap elemen array secara berurutan. Berikut adalah contoh penggunaan loop untuk mencetak semua elemen array "numbers":
   Java
   for (int i = 0;

Akses dan Manipulasi Elemen Array

Akses dan manipulasi elemen array adalah operasi dasar dalam penggunaan array. Melalui akses elemen, kita dapat membaca nilai dari elemen array, sedangkan melalui manipulasi elemen, kita dapat mengubah nilai elemen atau melakukan operasi lainnya terhadap elemen-elemen array.

1. Membaca Nilai Elemen Array
   Untuk membaca nilai dari elemen array, kita perlu

menggunakan indeks yang menunjukkan posisi elemen dalam array. Indeks dimulai dari 0(Sabila, n.d.). Contoh berikut menunjukkan cara membaca nilai dari elemen array dalam berbagai bahasa pemrograman.

Java:

```
int[] numbers = {2, 4, 6, 8, 10};
int firstElement = numbers[0]; // Membaca nilai elemen pertama
```

2. Menulis Nilai Elemen Array

   Untuk menulis atau mengubah nilai elemen array, kita dapat menggunakan indeks yang sesuai untuk mengakses elemen tersebut dan menetapkan nilai baru (Dwi Rahmatya, 2020). Contoh berikut menunjukkan cara menulis atau mengubah nilai elemen array dalam berbagai bahasa pemrograman.

   Java:

```
int[] numbers = {2, 4, 6, 8, 10};
numbers[1] = 12; // Mengubah nilai elemen kedua menjadi 12
```

3. Menambahkan Elemen Baru ke Array

   Untuk menambahkan elemen baru ke dalam array, beberapa bahasa pemrograman menyediakan metode atau fungsi khusus (Putri et al., 2022). Contoh berikut menunjukkan cara menambahkan elemen baru ke array dalam berbagai bahasa pemrograman.

   Java:

```
int[] numbers = {2, 4, 6, 8, 10};
int newElement = 12;
int[] newNumbers = Arrays.copyOf(numbers, numbers.length + 1);
```

```
newNumbers[numbers.length] = newElement;
```

4. Menghapus Elemen dari Array
   Untuk menghapus elemen dari array, kita dapat menggunakan metode atau fungsi yang disediakan oleh bahasa pemrograman. Contoh berikut menunjukkan cara menghapus elemen dari array dalam berbagai bahasa pemrograman.
   Java:
   ```
   int[] numbers = {2, 4, 6, 8, 10};
   int[] newNumbers = new int[numbers.length - 1];
   int indexToRemove = 2;
   System.arraycopy(numbers, 0, newNumbers, 0, indexToRemove);
   System.arraycopy(numbers, indexToRemove + 1, newNumbers, indexToRemove, numbers.length - indexToRemove - 1);
   ```

5. Mengganti Nilai Elemen Array
   Untuk mengganti nilai elemen dalam array, kita dapat menggunakan indeks untuk mengakses elemen tersebut dan menetapkan nilai baru. Contoh berikut menunjukkan cara mengganti nilai elemen dalam array dalam berbagai bahasa pemrograman.
   Java:
   ```
   int[] numbers = {2, 4, 6, 8, 10};
   int indexToReplace = 2;
   int newValue = 7;
   numbers[indexToReplace] = newValue;
   ```

Jenis-Jenis Array

1. Array Satu Dimensi
   Array satu dimensi adalah struktur data yang

digunakan untuk menyimpan kumpulan elemen dengan tipe data yang sama dalam urutan linier. Setiap elemen dalam array memiliki posisi yang disebut indeks, dimulai dari 0 untuk elemen pertama, 1 untuk elemen kedua, dan seterusnya. Array satu dimensi sering digunakan untuk mengelola kumpulan data yang berurutan dan memungkinkan akses dan manipulasi elemen dengan mudah (Suryawan et al., 2020).

Contoh Pseudocode:

Berikut adalah contoh pseudocode yang mengilustrasikan penggunaan array satu dimensi dalam menampilkan elemen-elemen dan mencari nilai maksimum dari array:

a. Mulai
b. Deklarasikan array numbers dengan ukuran 5
c. Inisialisasikan array numbers dengan nilai [10, 5, 8, 12, 6]
d. Set max_number = numbers[0] // Asumsi elemen pertama adalah nilai maksimum awal
e. Set i = 1
f. Selama i < panjang(numbers) lakukan: Jika numbers[i] > max_number maka: Set max_number = numbers[i] Tambahkan 1 ke i
g. Tampilkan "Elemen-elemen array:"
h. Untuk setiap elemen dalam numbers, lakukan: Tampilkan elemen
i. Tampilkan "Nilai maksimum dalam array:", max_number
j. Selesai

Pseudocode di atas mendeklarasikan array satu dimensi dengan nama "numbers" dan menginisialisasinya dengan nilai [10, 5, 8, 12, 6]. Kemudian, pseudocode melalui perulangan untuk

mencari nilai maksimum dalam array dengan membandingkan setiap elemen dengan "max_number". Terakhir, pseudocode menampilkan elemen-elemen array serta nilai maksimum yang ditemukan.

Pseudocode ini memberikan gambaran tentang bagaimana kita dapat mengakses dan memanipulasi elemen dalam array satu dimensi menggunakan indeks serta melakukan operasi seperti mencari nilai maksimum. Implementasi sesungguhnya dari pseudocode ini dapat dilakukan dalam bahasa pemrograman yang mendukung array seperti Python, Java, C++, dan lainnya.

2. Array Multidimensi

   Array multidimensi adalah array dengan dua atau lebih dimensi (Pratama, 2020). Array multidimensi adalah struktur data yang terdiri dari dua atau lebih dimensi. Dalam array multidimensi, elemen-elemennya diatur dalam susunan berlapis yang membentuk matriks atau tabel. Setiap elemen dalam matriks memiliki koordinat yang didefinisikan oleh indeks untuk setiap dimensi. Misalnya, array dua dimensi mirip dengan tabel dengan baris dan kolom, sedangkan array tiga dimensi dapat digambarkan sebagai susunan dari beberapa matriks berlapis. Penggunaan array multidimensi memungkinkan penyimpanan dan pengolahan data yang lebih terstruktur. Beberapa contoh penggunaan array multidimensi yaitu matriks, citra dan gambar, serta data berlapis.

   Contoh paling umum adalah array dua dimensi, yang juga dikenal sebagai matriks. Salah satu contoh yang sering menggunakan array multidimensi yaitu penjumlahan matriks.

Contoh Pseudocode (Penjumlahan Matriks):

```
DECLARE matrix1 AS Integer 2D Array
DECLARE matrix2 AS Integer 2D Array
DECLARE result AS Integer 2D Array

// Inisialisasi matrix1 dan matrix2

FOR i = 0 TO matrix1.rows - 1
  FOR j = 0 TO matrix2.columns - 1
    SET result[i][j] = matrix1[i][j] + matrix2[i][j]
  END FOR
END FOR

// Tampilkan hasil penjumlahan matriks
FOR i = 0 TO result.rows - 1
  FOR j = 0 TO result.columns - 1
    PRINT result[i][j]
  END FOR
END FOR
```

Dalam contoh di atas, pseudocode menunjukkan penjumlahan matriks. Algoritma tersebut menjumlahkan elemen-elemen yang sesuai dari matrix1 dan matrix2, dan hasilnya disimpan dalam array result.

3. Array Terurut dan Tidak Terurut
   Array terurut adalah array di mana elemen-elemennya diatur dalam urutan tertentu, seperti secara menaik atau menurun berdasarkan nilai kunci. Elemen-elemen array terurut memiliki hubungan yang jelas antara satu sama lain, sehingga memungkinkan pencarian dan pengurutan yang lebih efisien. Contoh array terurut adalah array

terurut secara alfabetis (misalnya, array kata dalam kamus) atau array terurut secara numerik. Keuntungan penggunaan array terurut adalah kemampuan untuk melakukan pencarian biner yang lebih efisien, yang memiliki kompleksitas waktu O (log n), di mana n adalah jumlah elemen dalam array. Array terurut umumnya lebih berguna ketika kita perlu melakukan pencarian, penghapusan, atau penyisipan elemen dengan efisiensi tinggi.

Array tidak terurut adalah array di mana elemen-elemennya tidak diatur dalam urutan tertentu. Elemen-elemen ini dapat berada dalam posisi apa pun dalam array. Tidak ada hubungan langsung antara elemen-elemen array tidak terurut, sehingga memerlukan pencarian linier atau pengurutan khusus jika kita ingin mencari elemen tertentu. Contoh array tidak terurut adalah array acak yang berisi data yang tidak memiliki urutan atau pola tertentu. Keuntungan penggunaan array tidak terurut adalah kemudahan dalam pengisian elemen dan operasi penyisipan yang sederhana, karena tidak memerlukan pemindahan elemen-elemen lain. Array tidak terurut umumnya lebih berguna ketika kita hanya perlu mengakses elemen secara acak tanpa perlu melakukan pencarian berulang atau pengurutan yang rumit.

Contoh Pseudocode (Pencarian Elemen dalam Array Terurut):

```
DECLARE array AS Integer Array
SET array = [3, 5, 7, 10, 15, 20, 25]
SET target = 10
SET found = false
SET index = -1
SET low = 0
```

```
SET high = array.length - 1

WHILE low <= high AND found = false
  SET mid = (low + high) / 2
  IF array[mid] = target
    SET found = true
    SET index = mid
  ELSE IF array[mid] < target
    SET low = mid + 1
  ELSE
    SET high = mid - 1
  END IF
END WHILE

IF found = true
  PRINT "Elemen ditemukan pada indeks " + index
ELSE
  PRINT "Elemen tidak ditemukan"
```

Dalam contoh di atas, pseudocode menunjukkan algoritma pencarian biner dalam array terurut. Algoritma tersebut membagi array menjadi dua bagian berulang kali hingga menemukan elemen yang dicari atau menentukan bahwa elemen tersebut tidak ada dalam array.

Penggunaan Array dalam Pemecahan Masalah Nyata

1. Penghitungan Rata-rata Nilai Siswa
   Dalam contoh ini, array digunakan untuk menyimpan nilai-nilai siswa dan menghitung rata-rata nilai mereka. Array nilai adalah struktur data yang memungkinkan kita menyimpan sejumlah

nilai-nilai dalam urutan tertentu. Pada awalnya, kita meminta pengguna untuk memasukkan jumlah siswa (n) dan membuat array nilai dengan ukuran n. Kemudian, menggunakan perulangan, kita mengisi array nilai dengan input dari pengguna. Selanjutnya, kita menghitung total nilai dengan menjumlahkan semua elemen dalam array nilai. Rata-rata nilai diperoleh dengan membagi total dengan jumlah siswa. Hasil rata-rata nilai ditampilkan kepada pengguna.

Pseudocode:

a. Mulai
b. Deklarasikan array nilai dengan ukuran n
c. Inisialisasikan array nilai dengan input nilai-nilai siswa
d. Set total = 0
e. Untuk setiap elemen dalam array nilai, lakukan:
b. Tambahkan elemen ke total
a. Hitung rata-rata = total / n
b. Tampilkan rata-rata
c. Selesai

2. Penjumlahan Matriks

Penjelasan: Pada contoh ini, array digunakan untuk menyimpan matriks A, matriks B, dan matriks hasil penjumlahan C. Matriks adalah representasi struktural data dua dimensi yang menggunakan array (Siahaan, 2020). Dalam contoh ini, kita menggunakan array dua dimensi untuk menyimpan matriks A dan B. Setelah itu, kita membuat matriks C dengan ukuran yang sama untuk menyimpan hasil penjumlahan matriks A dan B. Melalui perulangan bersarang, kita menjumlahkan elemen-elemen matriks A dan B pada posisi yang sesuai dan

menyimpan hasilnya dalam matriks C. Setelah itu, matriks C ditampilkan kepada pengguna.

Pseudocode:

a. Mulai
b. Deklarasikan matriks A dengan ukuran mxn
c. Deklarasikan matriks B dengan ukuran mxn
d. Deklarasikan matriks C dengan ukuran mxn
e. Inisialisasikan matriks A dengan nilai-nilai
f. Inisialisasikan matriks B dengan nilai-nilai
g. Untuk setiap baris i dalam matriks A, lakukan:
   Untuk setiap kolom j dalam matriks A, lakukan:
   Set C[i][j] = A[i][j] + B[i][j]
h. Tampilkan matriks C
i. Selesai

Implementasi dalam Java:

```
public class PenjumlahanMatriks {
   public static void main(String[] args) {
      int[][] A = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
      int[][] B = {{10, 11, 12}, {13, 14, 15}, {16, 17, 18}};
      int m = A.length;
      int n = A[0].length;

      int[][] C = new int[m][n];

      for (int i = 0; i < m; i++) {
         for (int j = 0; j < n; j++) {
            C[i][j] = A[i][j] + B[i][j];
         }
      }

      System.out.println("Hasil penjumlahan matriks:");
      for (int i = 0; i < m; i++) {
```

```
            for (int j = 0; j < n; j++) {
               System.out.print(C[i][j] + " ");
            }
            System.out.println();
         }
      }
   }
```

3. Pencarian Elemen Maksimum
   Penjelasan: Dalam contoh ini, array digunakan untuk menyimpan angka-angka. Melalui perulangan, elemen-elemen array diiterasi untuk mencari elemen maksimum. Pada awalnya, kita meminta pengguna untuk memasukkan jumlah angka (n) dan membuat array angka dengan ukuran n. Selanjutnya, menggunakan perulangan, kita mengisi array angka dengan input dari pengguna. Variabel maks dideklarasikan dan diinisialisasi dengan elemen pertama dalam array. Kemudian, menggunakan perulangan, kita membandingkan setiap elemen dalam array dengan nilai maks saat ini. Jika ditemukan elemen yang lebih besar, maka nilai maks diperbarui. Pada akhirnya, elemen maksimum ditampilkan kepada pengguna.
   Pseudocode:
   a. Mulai
   b. Deklarasikan array angka dengan ukuran n
   c. Inisialisasikan array angka dengan input angka-angka
   d. Set maks = angka[0]
   e. Untuk setiap elemen dalam array angka, lakukan:
      Jika elemen > maks, maka:
      Set maks = elemen

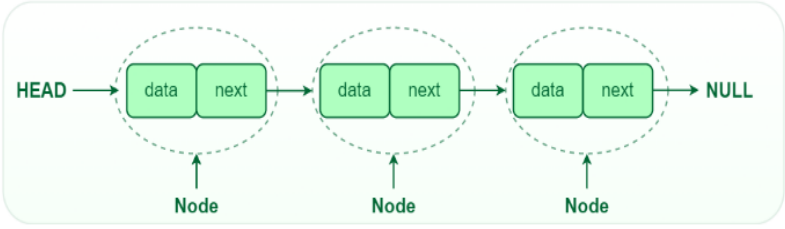
f. Tampilkan maks
g. Selesai

BAB 8 LINK LIST

Pendahuluan

Silahkan tambahkan pendahuluan sesuai dengan tema bab-nya, paling sedikit 3 paragrap.

Linked List adalah struktur data linier, di mana elemen tidak disimpan di lokasi yang berdekatan, tetapi ditautkan menggunakan pointer. Linked List membentuk rangkaian node yang terhubung, dimana setiap node menyimpan data dan alamat dari node berikutnya. (GeeksforGeeks, Understanding the basics of Linked List, 2023)



Gambar 28. Linked List

Struktur Node: Node dalam linked list biasanya terdiri dari dua komponen:

Data: Ini memegang nilai aktual atau data yang terkait dengan node.

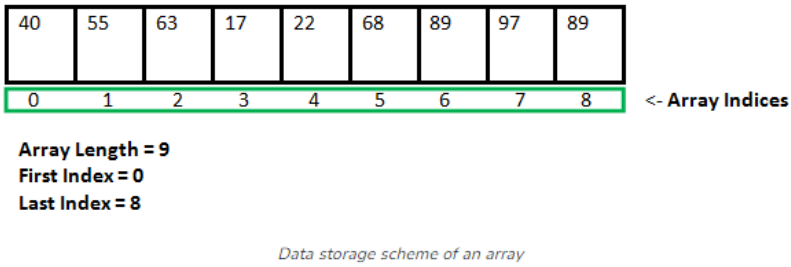
Next Pointer: Ini menyimpan alamat memori (referensi) dari node berikutnya dalam urutan.

Head and Tail: linked list diakses melalui simpul kepala, yang menunjuk ke simpul pertama dalam daftar. Node terakhir dalam daftar menunjuk ke NULL, menunjukkan akhir dari daftar. Simpul ini dikenal sebagai

simpul ekor.

Linked List vs Array

Array: Array menyimpan elemen di lokasi memori yang berdekatan, menghasilkan alamat yang mudah dihitung untuk elemen yang disimpan dan ini memungkinkan akses lebih cepat ke elemen pada indeks tertentu. (GeeksforGeeks, Linked List vs Array, 2023)



Gambar 29. Array vs Linked List

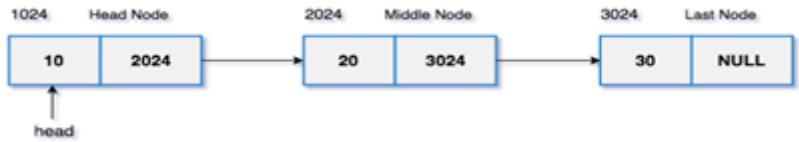
Dalam kasus array, ukuran dibatasi oleh definisi, tetapi dalam linked list, tidak ada ukuran yang ditentukan. Struktur ini memungkinkan penyisipan atau penghapusan elemen secara efisien dari posisi mana pun dalam urutan selama iterasi.

Kelemahan dari linked list adalah bahwa waktu akses data adalah fungsi linier dari jumlah node untuk setiap daftar tertaut (yaitu, waktu akses meningkat secara linier saat node ditambahkan ke daftar tertaut.) karena node ditautkan secara berurutan sehingga node perlu diakses terlebih dahulu untuk mengakses node selanjutnya (sehingga sulit untuk disalurkan).

Representasi dari linked list

Linked list dapat direpresentasikan sebagai koneksi node di mana setiap node menunjuk ke node berikutnya

dari daftar. Representasi dari Linked List ditunjukkan di bawah ini



Gambar 30. Ilustrasi Linked List (Log2Base2, 2023)

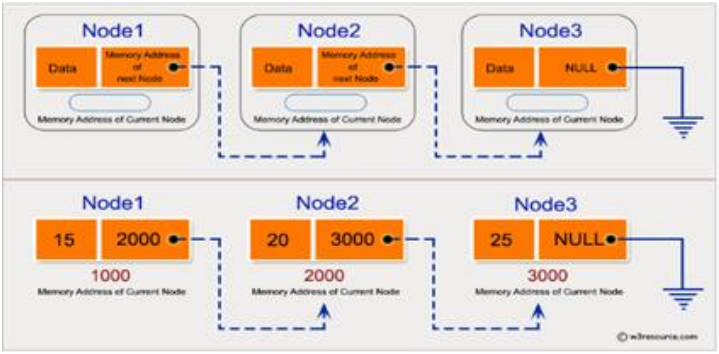
Menghubungkan setiap node

1. kepala => selanjutnya = tengah. Oleh karena itu head => next menyimpan alamat memori dari node tengah (2024).
2. tengah => selanjutnya = terakhir. Oleh karena itu middle => next menyimpan alamat memori dari node terakhir (3024).
3. last => next = NULL yang menunjukkan node terakhir dalam linked list.
4. Versi sederhana dari bagian memori heap.

Jenis-Jenis Linked List

Secara umum, linked list dapat dibagi ke dalam 4 jenis, yakni: Single linked list, Double linked list dan Circular linked list

1. Single-linked list
   Single-linked list seperti sistem kereta api, yang menghubungkan setiap gerbong ke kerbong berikutnya. Single linked list adalah linked list unidirectional; yaitu, Anda hanya dapat melintasinya dari simpul kepala ke simpul ekor.

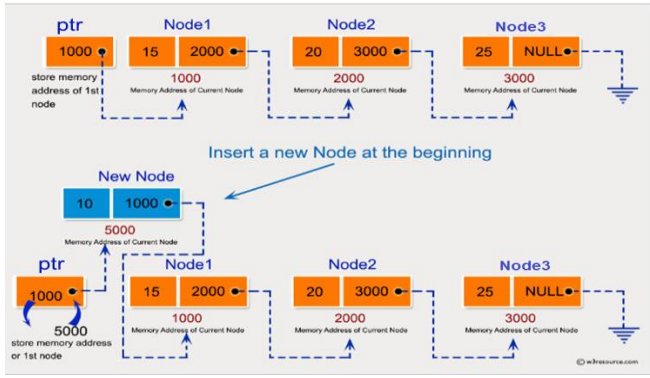


Gambar 31. Single Linked List (w3resource, 2023)

Operasi pada Single linked list
Berikut adalah daftar operasi dasar pada Single linked list:

a. Insertion - Penyisipan elemen baru ke linked list
b. Deletion - menghapus elemen yang ada
c. Searching - menemukan simpul pada linked list

a. Operasi Penyisipan
   Penyisipan node – diawal
   Algoritma untuk Penyisipan simpul baru di awal Single Linked List:
   1) Buat node untuk menyimpan data.
   2) Cek apakah simpul kepala Kosong (head = NULL)
   3) Jika Kosong, maka newNode→next = NULL dan head = newNode.
   4) Jika Tidak Kosong, maka newNode→next = head dan head = newNode.

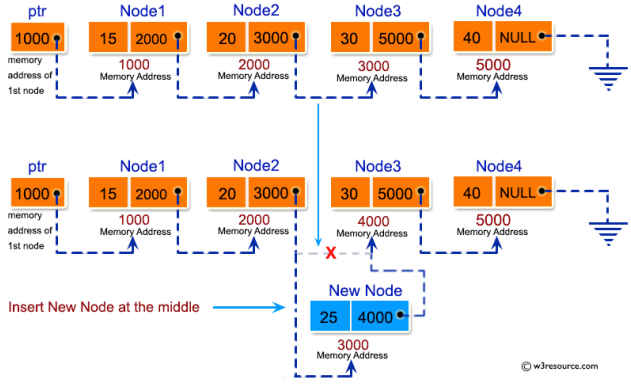


Gambar 32. Penyisipan di awal pada Single Linked List

Penyisipan node – Tenggah

Algoritma untuk Penyisipan simpul baru setelah simpul dalam Single Linked List:

1) Buat node untuk menyimpan data.
2) Cari posisi yang akan disisipkan dan sesuaikan penunjuk titik simpul baru ke lokasi posisi berikutnya.
3) Buat node baru menunjuk ke elemen berikutnya dari node saat ini.

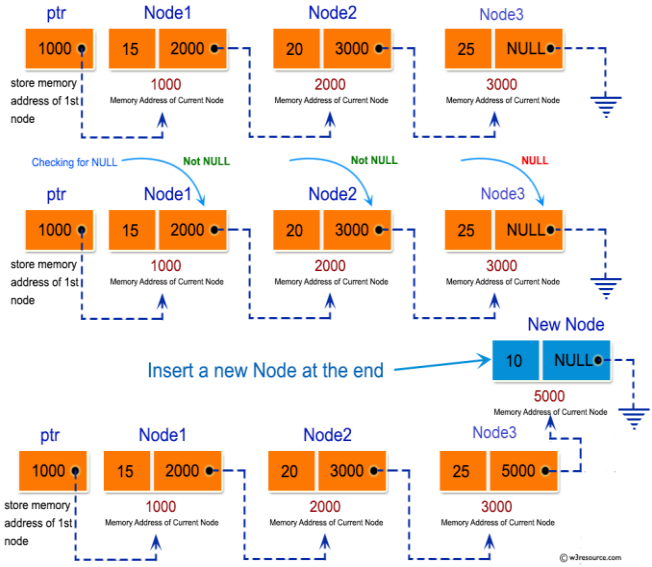


Gambar 33. Penyisipan di tenggah pada Single Linked List

Penyisipan node – Akhir

Algoritma untuk menyisipkan simpul baru di akhir Single Linked List:

1) Jelajahi daftar dari awal dan mencapai simpul terakhir. Jika panjang linked list adalah K, maka jelajahi dari node ke-1 ke node ke (K-N+1).
2) Buat titik ekor ke simpul baru dan kemudian buat penunjuk baru titik simpul berikutnya ke NULL untuk mewakili akhir daftar.



Gambar 34. Penyisipan di akhir pada Single Linked List

b. Operasi penghapusan

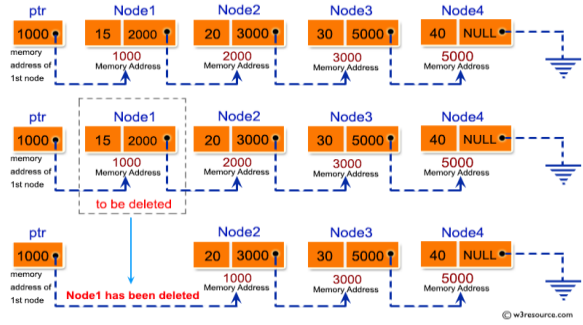
Penghapusan node di awal

Algoritma untuk menghapus simpul dari awal Single Linked List:

1) Cari node yang akan dihapus.
2) Jika node tidak ditemukan maka

kembalikan pesan yang menunjukkan bahwa node tidak ditemukan.

3) Jika simpul ditemukan – Tetapkan pointer kepala ke node berikutnya dalam daftar

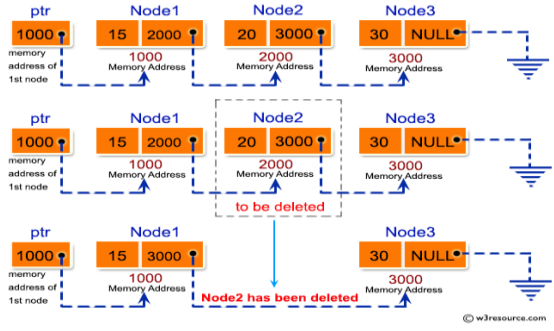


Gambar 35. Penghapusan di awal pada Single Linked List

Penghapusan node di tenggah

Algoritma untuk menghapus simpul tertentu dari Single Linked List:

1) Jelajahi hingga mencapai node yang diinginkan setelah itu node akan dihapus.
2) Buat simpul saat ini menunjuk ke elemen berikutnya berikutnya.

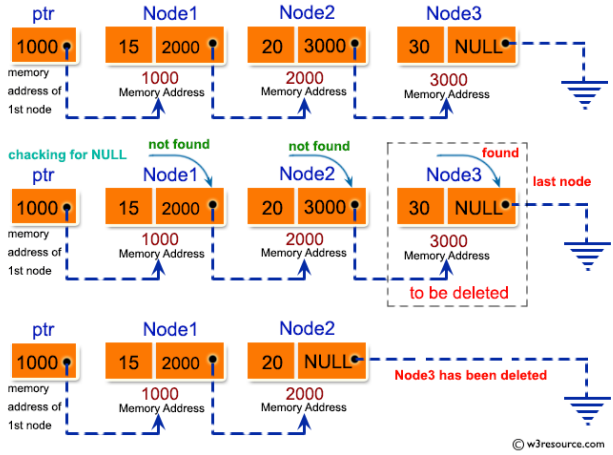


Gambar 36. Penghapusan di tenggah pada Single Linked List

Penghapusan node di akhir

Algoritma untuk menghapus simpul dari akhir Single Linked List:

1) Jelajahi hingga Anda menemukan elemen terakhir kedua dalam daftar.
2) Tetapkan NULL ke elemen terakhir kedua dalam daftar.



Gambar 37. Penghapusan di akhir pada Single Linked List

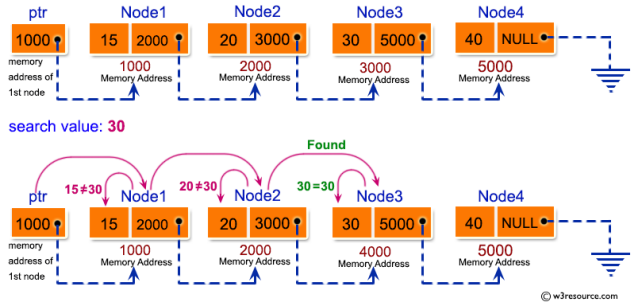
c. Pencarian node

Algoritma untuk menampilkan elemen dari Single Linked List:

1) Inisialisasi node curr dan arahkan ke kepala.
2) Sekarang, saat ini bukan NULL, telusuri daftar.
3) Bandingkan data curr dengan elemen X yang diberikan.
4) Jika suatu saat, data curr sama dengan X, itu berarti elemen tersebut ada dalam daftar, dan kami akan mengembalikan

nilai true.

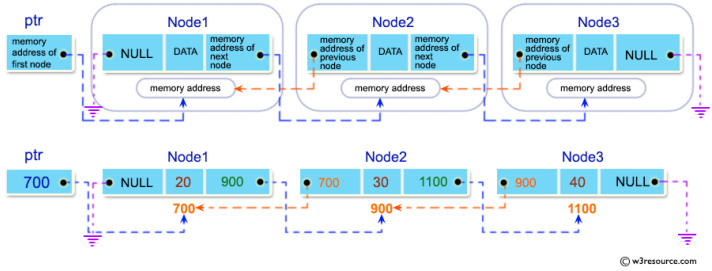
5) Terus tingkatkan curr.
6) Jika traversal selesai dan elemen tidak ditemukan, kembalikan salah.



Gambar 38. Pencarian pada Single Linked List

2. Double linked list

Double linked list adalah linked list dua arah. Jadi, kita bisa melintasinya secara dua arah, maju dan mundur. Dalam linked list terdapat 2 pointer yang merupakan pointer utama, yaitu pointer HEAD yang menunjuk ke node pertama pada linked list itu sendiri dan pointer TAIL yang menunjuk ke node terakhir pada linked list. Linked list dikatakan kosong jika kepala penunjuk adalah NULL. Juga, nilai prev pointer HEAD selalu NULL, karena ini adalah data pertama. Begitu juga dengan pointer selanjutnya dari TAIL yang selalu NULL sebagai penanda data terakhir. Dalam hal struktur, beginilah tampilan daftar tertaut ganda:



Gambar 39. Double Linked List

Operasi pada Double linked list

Berikut adalah daftar operasi dasar pada Double linked list:

a. Insertion - Penyisipan elemen baru ke linked list
b. Deletion - menghapus elemen yang ada
c. Searching - menemukan simpul pada linked list

a. Operasi Penyisipan

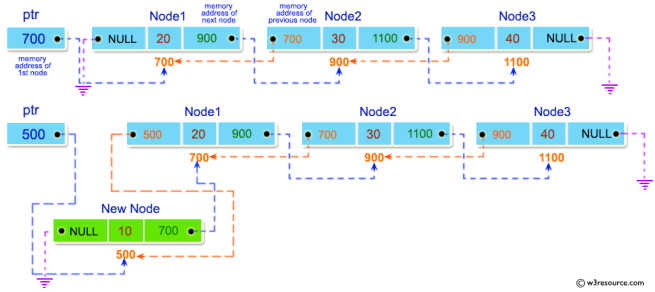
   Penyisipan di Awal

   Dalam operasi ini, kami membuat simpul baru dengan tiga kompartemen, satu berisi data, yang lain berisi alamat dari simpul sebelumnya dan berikutnya dalam daftar. Node baru ini dimasukkan di awal.

   Algoritma untuk menyisipkan node baru di awal double linked list:

   1) Buat newNode dengan nilai yang diberikan dan newNode → sebelumnya sebagai NULL.
   2) Periksa apakah daftar Kosong (head == NULL)
   3) Jika Kosong, tetapkan NULL ke newNode → next dan newNode ke head.

4) Jika tidak Kosong, tetapkan head ke newNode → next dan newNode ke head.

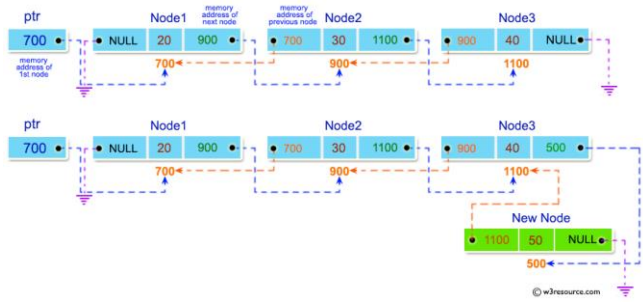


Gambar 40. Penyisipan di awal pada Double Linked List

Penyisipan Di Akhir

Algoritma untuk menyisipkan node baru di akhir double linked list:

1) Buat newNode dengan nilai yang diberikan dan newNode → selanjutnya sebagai NULL.
2) Periksa apakah daftar Kosong (head == NULL)
3) Jika Kosong, tetapkan NULL ke newNode → sebelumnya dan newNode ke head.
4) Jika tidak, lanjutkan temp hingga Anda mencapai node terakhir (hingga temp → next = NULL).
5) Tetapkan newNode ke temp → next dan temp ke newNode → sebelumnya.

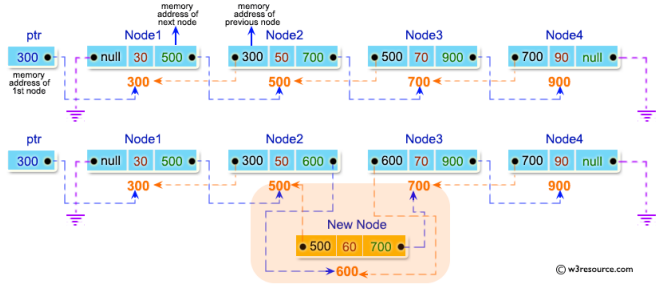


Gambar 41. Penyisipan di akhir pada Double Linked List

Penyisipan Di Tenggah

Algoritma untuk Penyisipan simpul baru setelah simpul dalam Double Linked List:

1) Node baru harus dibuat dan diberi nilai data.
2) Ulangi ke lokasi di daftar tertaut yang ingin Anda masukkan setelahnya.
3) Tetapkan node berikutnya dan sebelumnya dari node baru ini.
4) Di sebelah node baru ini, tempatkan node sebelumnya.
5) Berikan simpul baru ini simpul sebelumnya dari simpul berikutnya.



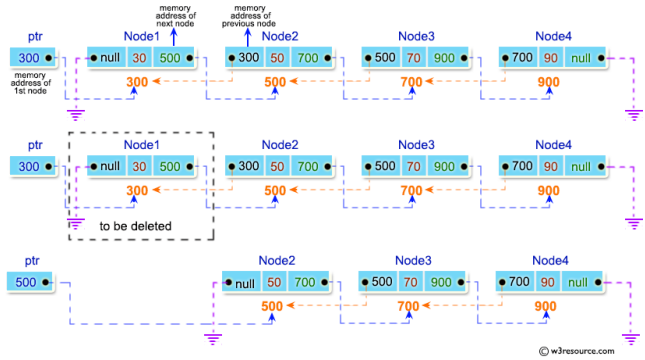
Gambar 42. Penyisipan di tenggah pada Double Linked List

b. Operasi penghapusan

Menghapus node di awal

Algoritma untuk menghapus node dari awal double linked list:

1) Periksa status daftar tertaut ganda (head == NULL)
2) Jika daftar kosong, penghapusan tidak dapat dilakukan
3) Jika list tidak kosong, maka head pointer digeser ke node berikutnya.

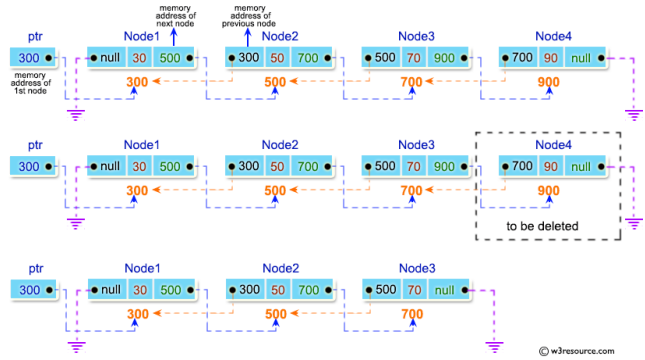


Gambar 43. Penghapusan di awal pada Double Linked List

Menghapus node di akhir

Algoritma untuk menghapus node dari akhir double linked list:

1) Jika daftar kosong, tambahkan simpul ke daftar dan arahkan kepala ke sana.
2) Jika daftar tidak kosong, temukan node terakhir dari daftar tersebut.
3) Buat tautan antara simpul terakhir dalam daftar dan simpul baru.
4) Node baru akan mengarah ke NULL karena merupakan node terakhir yang baru.

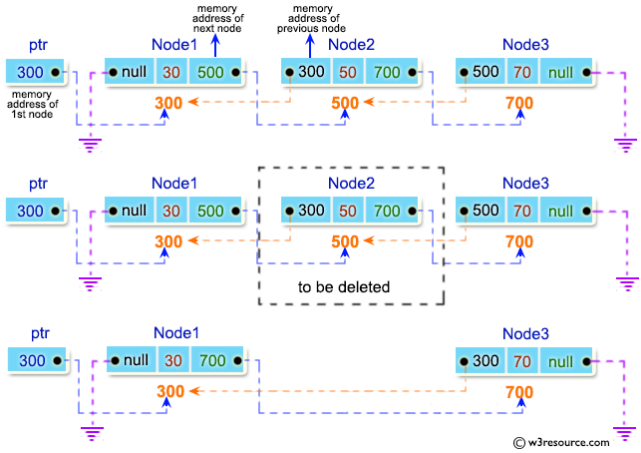


Gambar 44. Penghapusan di akhir pada Double Linked List

Menghapus Node Tenggah

Algoritma untuk menghapus simpul tertentu dari daftar tertaut ganda:

1) Jika daftar kosong, maka tidak ada yang dihapus, dikembalikan.
2) Jika simpul yang dihapus bukan simpul kepala atau ekor, maka perbarui penunjuk berikutnya dari simpul sebelumnya untuk menunjuk ke simpul berikutnya, dan perbarui penunjuk sebelumnya dari simpul berikutnya untuk menunjuk ke simpul sebelumnya.
3) Bebaskan memori yang dialokasikan ke node untuk dihapus.



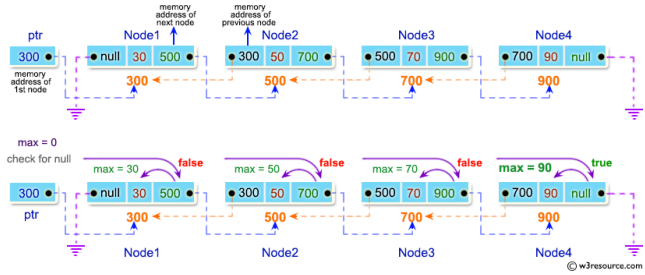
Gambar 45. Penghapusan di tenggah pada Double Linked List

c. Pencarian node

Algoritma untuk menampilkan elemen dari double linked list:

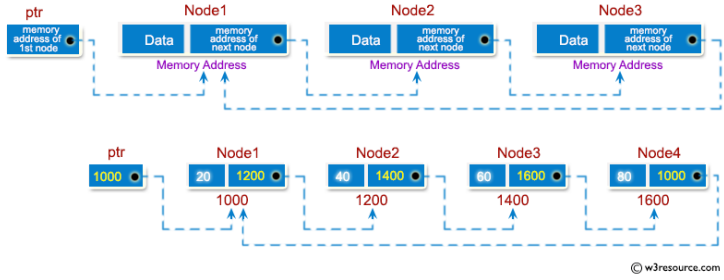
1) Pertama, kita perlu membuat posisi variabel yang akan melacak jumlah node yang dilalui.
2) Kemudian menggunakan pointer, katakanlah, temp, kami akan menelusuri daftar yang diberikan hingga node berikutnya dari temp adalah null, atau kami menemukan elemen yang kami cari.
3) Kemudian kami akan memeriksa apakah data node saat ini sama dengan elemen yang kami cari atau tidak.
4) Jika temp.data == X, variabel posisi kami akan memberikan lokasi elemen yang akan dicari dan kami akan menampilkannya.
5) Selain itu, itu berarti tidak ada simpul

dengan nilai X dalam daftar yang diberikan, jadi kami akan menampilkan -1



Gambar 46. Pencarian pada Double Linked List

3. Circular-linked list
   Circular linked lists adalah daftar tertaut di mana semua node terhubung untuk membentuk lingkaran. Tidak ada NULL di akhir. Seperti inilah tampilan daftar tertaut melingkar:



Gambar 47. Circular linked lists

BAB 9 TREE

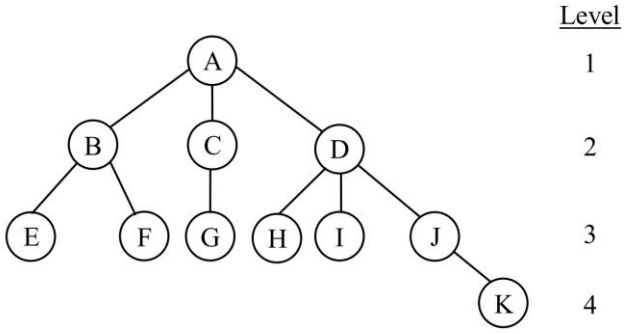
Pendahuluan

Pada bab ini, kita akan membahas salah satu struktur data non-linier terpenting dan sangat fleksibel dalam komputasi TREE. Terminologi utama dalam struktur data pohon berasal dari pohon keluarga, dengan istilah "parent," "child," "ancestor," dan "descendant" menjadi kata yang paling umum digunakan untuk menggambarkan hubungan.

TREE terdiri dari node (simpul) yang saling terhubung melalui garis-garis yang disebut edge (cabang). Node pertama pada pohon akan disebut root (akar), sedangkan node yang tidak memiliki anak disebut leaf (daun). Setiap node dapat memiliki satu atau lebih anak atau disebut subtree yang terhubung langsung melalui cabang.

Struktur data TREE sangat berguna dalam berbagai aplikasi seperti pemodelan hirarki organisasi, pemrosesan dan penyimpanan kata, implementasi struktur data lanjutan seperti heap dan set asosiatif, serta banyak algoritma yang melibatkan pengorganisasian data secara hierarkis.

Terminologi TREE



Gambar 48. Terminologi tree

Adapun istilah-istilah objek TREE dijelaskan sebagai berikut.

1. Node: Simpul (node) merupakan elemen paling dasar dalam sebuah TREE. Setiap simpul dapat memiliki nol atau lebih simpul anak, kecuali simpul pada level terbawah yang disebut simpul daun (leaf node). pohon pada gambar di atas. Pohon ini memiliki 11 node (A, B, C, D, E, F, G, H, I, J dan K)
2. Root: Akar (Root) adalah simpul paling atas dalam TREE. Ini adalah titik awal dari mana TREE bercabang menjadi simpul-simpul lain Pada gambar di atas, (A) merupakan simpul akar
2. Parent Node (Predecessor): Simpul induk (Parent Node) merupakan simpul yang memiliki satu atau lebih simpul anak. Setiap simpul, kecuali akar, memiliki tepat satu simpul induk Pada gambar di atas, simpul D adalah induk dari H, I dan J
3. Children Node: Simpul Anak merupakan simpul yang menjadi turunan dari simpul induk. Simpul anak terhubung ke simpul induk melalui percabangan ke kiri dan ke kanan. Pada gambar di

atas, H, J dan I adalah anak-anak dari simpul D.

4. Siblings: Anak-anak (atau node) dari orang tua yang sama dikatakan saudara kandung. Pada gambar di atas, H, I dan J adalah saudara kandung.
5. Subtree: Subtree merupakan bagian dari TREE yang terdiri dari simpul - simpul dan cabang yang merupakan turunan dari simpul tertentu dalam TREE.
6. Size: Size merupakan jumlah banyaknya simpul dalam suatu pohon atau TREE
7. Height: Heigth (Tinggi) atau Depth (Kedalaman) merupakan jarak dari suatu simpul terhadap akar dalam TREE. Depth akar adalah 0, dan depth suatu simpul adalah depth simpul induk ditambah 1.

Binary Tree

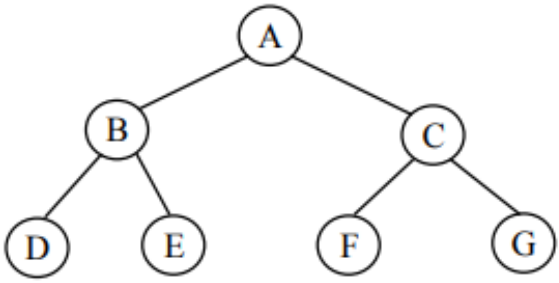
Struktir daa hirarkis yang terdiri dari simpul–simpul yang terhubung melalui tepi–tepi yang dikenal sebagai pohon biner. Dalam pohon biner, setiap simpul memiliki setidaknya dua anak, yang disebut anak kiri dan anak kanan. Pohon biner biasanya digunakan untuk menunjukkan strukur data seperti pohon pencarian biner, ekspersi matematika, dan struktur data lainnya.

Dalam pohon biner, setiap simpul biasanya terdiri dari 2 komponen utama yaitu data dan informasi yang disimpan di dalam simpul dan referensi atau tautan ke anak–anak simpul. Anak kiri dan anak kanan setiap simpul juga merupakan pohon biner yang lebih kecil. Tautan menunjukan ke nilai null atau tidak ada jika tidak ada anak.

Berikut merupakan Jenis-jenis pohon biner yang umum digunakan:

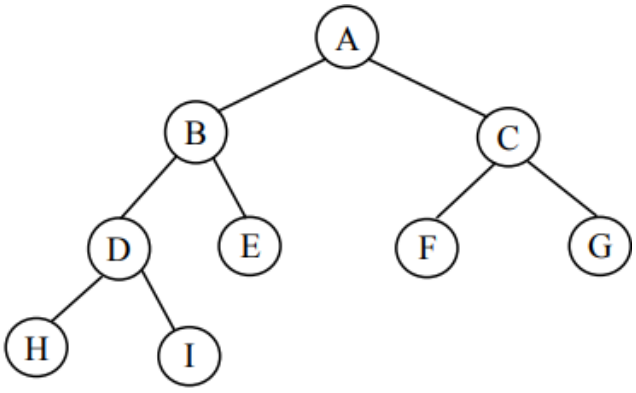
1. Full Binary Tree (Pohon Biner Penuh)

Pada jenis pohon seperti ini, bisa disebut pohon biner penuh. Dengan kata lain, dalam pohon biner penuh, Semua node internal memiliki dua anak dan semua daun berada pada tingkat yang sama. Pertimbangkan gambar berikut, pohon biner penuh kedalaman 3 level.



Gambar 49. Pohon Biner Penuh

2. Complate Binary Tree (Pohon Biner Lengkap)
   Pohon biner dinyatakan lengkap dimana semua level terisi dan level terakhir mungkin Sebagian terisi dari kiri ke kanan dan beberapa daun paling kanan mungkin hilang.



Gambar 50. Pohon Biner Lengkap

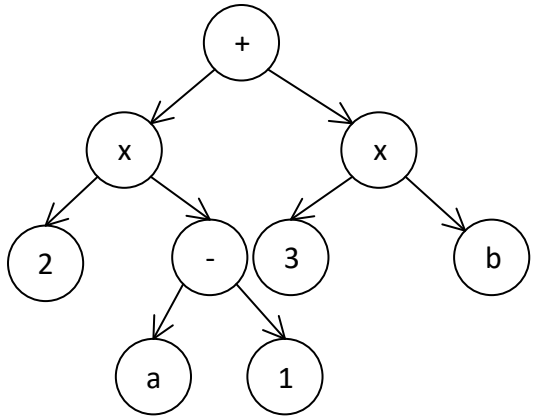
3. Balanced Binary Tree (Pohon Biner Seimbang)
   Pohon biner seimbang merupakan jenis pohon biner

di mana setiap simpul memiliki 2 subpohon dengan tinggi yang seimbang. Tidak boleh ada lebih dari satu perbedaan tinggi antara subpohon kiri dan kanan

Aplikasi Pada Binary Tree

Arithmetic expressions

Kelompok binary tree yang digunakan untuk mengekspresikan notasi aritmatika Contoh: arithmetic expression Tree untuk persamaan (2 × (a – 1) + (3 × b)).



Dalam pohon ekspresi aritmatika ini, kita dapat mengikuti langkah-langkah evaluasi untuk memahami urutan operasi yang benar.

1. Pertama, pada subtree pertama, operasi pengurangan (–) antara variabel "a" dan nilai bilangan 1 dievaluasi. Ini menghasilkan hasil pengurangan dari "a" dan 1.
2. Selanjutnya, pada subtree pertama, operasi perkalian (*) dijalankan dengan nilai 2 dan hasil pengurangan sebelumnya (a - 1).
3. Pada subtree kedua, operasi perkalian (*) dievaluasi antara nilai bilangan 3 dan variabel "b".
4. Setelah kedua subtree dievaluasi, hasil operasi

perkalian di masing-masing subtree dikembalikan ke parent node mereka.

5. Pada root node, operasi penjumlahan (+) dilakukan antara hasil operasi perkalian pertama (2 * (a - 1)) dan hasil operasi perkalian kedua (3 * b).
6. Hasil akhir dari penjumlahan tersebut akan menjadi hasil akhir dari ekspresi aritmatika.

Dengan menggunakan pohon ekspresi aritmatika, kita dapat dengan jelas melihat urutan evaluasi yang benar, yaitu terlebih dahulu melakukan operasi pengurangan dalam kurung, kemudian melakukan perkalian, dan terakhir melakukan penjumlahan.

Berikut adalah contoh kode dalam bahasa C++ untuk mengevaluasi ekspresi aritmatika "(2 * (a – 1) + (3 * b))" menggunakan pohon ekspresi:

```
#include <iostream>
#include <unordered_map>
#include <string>
using namespace std;
struct Node {
   string value;
   Node* left;
   Node* right;
};

// Fungsi untuk membuat node baru dalam pohon ekspresi
Node* createNode(string value) {
   Node* newNode = new Node;
   newNode->value = value;
   newNode->left = newNode->right = NULL;
   return newNode;
}
// Fungsi untuk mengevaluasi ekspresi aritmatika
```

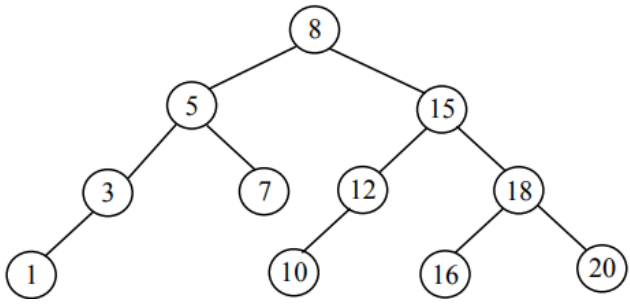
```
berdasarkan pohon ekspresi
int evaluateExpression(Node* root,
unordered_map<string, int>& variables) {
  if (root == NULL) {
    return 0;
  }
    if (root->left == NULL && root->right == NULL) {
    if (isdigit(root->value[0])) {
      return stoi(root->value);
    } else {
return variables[root->value];
       }
       }

       int leftValue = evaluateExpression(root->left,
variables);
       int rightValue = evaluateExpression(root->right,
variables);
return variables[root->value];
    }
  }

  int leftValue = evaluateExpression(root->left, variables);
  int rightValue = evaluateExpression(root->right,
variables);

  if (root->value == "+") {
    return leftValue + rightValue;
  } else if (root->value == "-") {
    return leftValue - rightValue;
  } else if (root->value == "*") {
    return leftValue * rightValue;
  } else if (root->value == "/") {
    return leftValue / rightValue;
```

```
    }
        return 0;
}
int main() {
    // Membangun pohon ekspresi
    Node* root = createNode("+");
    root->left = createNode("*");
    root->left->left = createNode("2");
    root->left->right = createNode("-");
    root->left->right->left = createNode("a");
    root->left->right->right = createNode("1");
    root->right = createNode("*");
    root->right->left = createNode("3");
    root->right->right = createNode("b");
    // Menentukan nilai variabel
    unordered_map<string, int> variables;
    variables["a"] = 5; // Gunakan Data Pengganti
    variables["b"] = 2; // Gunakan Data Pengganti
    // Evaluasi ekspresi aritmatika
    int result = evaluateExpression(root, variables);

    // Menampilkan hasil evaluasi
    cout << "Hasil evaluasi: " << result << endl;
    return 0;
}
```

Binary Search Tree (BST)

Binary Search Tree (BST) merupakan struktur Data pohon biner yang mempunyai sifat yang unik karena memungkinkan mencari, menambahkan, dan menghapus elemen dengan efisiensi tinggi. BST dikenal dengan sifat bahwa setiap simpul dalam pohon memiliki nilai yang lebih besar dari semua simpul di subpohon kirinya dan nilai yang

lebih kecil dari semua simpul di subpohon kanannya.



Gambar 51. Binary Search Tree

Operasi pada Pohon Pencari Biner

1. Insert pada pohon percari biner
   Misalkan sebuah item data baru memiliki kunci dan pohon di mana kunci tersebut akan dimasukkan sebagai input. Operasi penyisipan dimulai dari simpul akar. Jika pohon kosong, maka item baru dimasukkan sebagai simpul akar. Namun, jika pohon tidak kosong, bandingkan nilai kunci dengan simpul akar. Jika kunci lebih kecil daripada simpul akar, maka item tersebut dimasukkan ke dalam anak pohon kiri; jika tidak, item tersebut dimasukkan ke dalam anak pohon kanan.
   Berikut algoritma untuk menyisipkan item ke pohon pencarian biner menggunakan *Recursion Method*.

   Algorithm: INSERT (ROOT, DATA)
   Prosedur: INSERT (ROOT, DATA)
   1. IF ROOT = NULL THEN
      i. Alokasikan Memori untuk simpul ROOT.
      ii. ROOT->INFO = DATA
      iii. ROOT->LCHILD = NULL
      iv. ROOT->RCHILD = NULL
   2. ELSE, IF ROOT->INFO > DATA THEM

CALL INSERT(ROOT->LCHILD, DATA)
3. ELSE, IF ROOT->INFO < DATA MAKA
CALL INSERT(ROOT->RCHILD, DATA)
4. RETURN

2. Searching pada pohon pencari biner
Seperti halnya operasi traversal dan penyisipan dalam pohon pencarian biner, algoritma pencarian juga menggunakan teknik rekursi.
Anggaplah ada pohon dan kunci yang dicari. Sekarang mulai dari simpul akar dan lihat apakah nilai simpul saat ini sama dengan kunci. Jika simpul saat ini kosong, maka nilai kunci yang dicari tidak ada dalam pohon pencarian biner, dan jika simpul memiliki kunci yang dicari, maka pencarian berhasil. Nilai kunci pencarian harus lebih besar atau lebih kecil dari kunci node saat ini jika tidak. Dalam kasus pertama, nilai kunci pencarian lebih besar daripada semua kunci di subtree kiri. Ini menunjukkan bahwa Anda tidak perlu mencari di subtree kiri. Akibatnya, Anda hanya perlu mencari subtree yang tepat. Demikian pula, dalam kasus kedua, Anda hanya perlu mencari subtree yang tepat.
Berikut Algoritma untuk mencari item dari pohon pencarian biner menggunakan *recursion*

Algorithm: BSTSearch (ROOT, DATA, P)
1. IF ROOT = NULL THEN
i) PRINT: NOT FOUND
ii) P = NULL
iii) RETURN
2. IF ROOT->INFO=DATA THEN
SET P=ROOT
3. ELSE IF ROOT->INFO>DATA THEN

CALL BSTSearch(ROOT->LCHILD,DATA,P)

4. ELSE CALL BSTSearch(ROOT->RCHILD,DATA,P)
5. RETURN

Algoritma untuk mencari item dari pohon pencarian biner menggunakan iterative methods

Algorithm: BSTSearch (ROOT, DATA, P)

1. P = ROOT
2. Repeat while P ≠ Null
3. If DATA = P->INFO then Return
4. Else If DATA < P->INFO then
5. P = P->LCHILD
6. Else P = P->RCHILD
   [End of loop]
7. Return

3. Deletion pada pohon pencari biner
   Delele dalam binary search tree mempengaruhi struktur dari tree tersebut. Sehingga apabila node yang dihapus mempunyai child maka posisi node yang dihapus digantikan dengan leaf yang berada pada posisi yang terakhir

Model Kunjungan pada BST

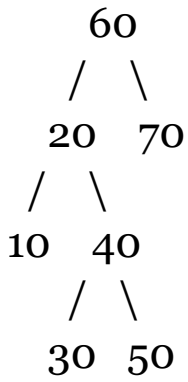
Sistem yang digunakan untuk kunjungan kali ini ialah LRO (left to right oriented). artinya anak kiri dikunjungi dulu baru kemudian anak kanan.

1. PreOrder (Depth First Order)
   Salah satu cara untuk menjelajahi atau mengunjungi node - node dalam struktur data pohon adalah

dengan menggunakan metode preorder, juga dikenal sebagai Preorder Depth First Order.

a. Cetak isi node yang dikunjungi
b. Kunjungi left child
c. Kunjungi right child

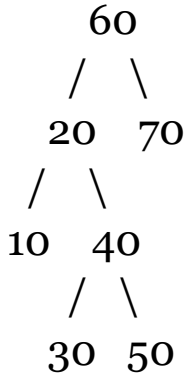
Bilamana diketahui pohon biner seperti terlihat pada gambar di bawah ini



maka hasil kunjungan menggunakan metode Preorder adalah: **60 20 10 40 30 50 70**

2. InOrder (Symetric Order)
   Salah satu cara untuk menjelajahi atau mengunjungi node - node dalam struktur data pohon adalah dengan menggunakan metode InOrder
   a. Kunjungi left child
   b. Kunjungi right child
   c. Cetak isi node yang dikunjungi

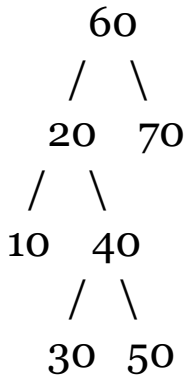
   Bila diketahui pohon biner seperti terlihat pada gambar di bawah



maka hasil kunjungan menggunakan metode InOrder adalah: **10 20 30 40 50 60 70**

3. PostOrder
   Salah satu cara untuk menjelajahi atau mengunjungi node - node dalam struktur data pohon adalah dengan menggunakan metode PostOrder
   a. Kunjungi left child
   b. Kunjungi right child
   c. Cetak isi node yang dikunjungi

   Bila diketahui pohon biner seperti terlihat pada gambar di bawah



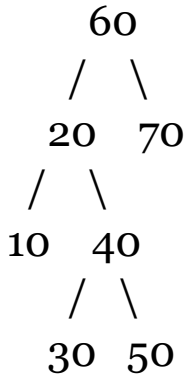
   maka hasil kunjungan menggunakan metode InOrder adalah: **10 30 50 40 20 70 60**

4. LevelOrder
   Salah satu cara untuk menjelajahi atau mengunjungi

node - node dalam struktur data pohon adalah dengan menggunakan metode LevelOrder,
kunjungi node pada tingkat yang sama dimulai dari root sampai node-node yang merupakan
leaf-nya.

a. Kunjungi left child
b. Kunjungi right child
c. Cetak isi node yang dikunjungi

Bila diketahui pohon biner seperti terlihat pada gambar di bawah



maka hasil kunjungan menggunakan metode LevelOrder adalah: **60 20 70 10 40 30 50**

Notasi Prefix, Infix, dan Postfix

Notasi prefix, infix, dan postfix adalah tiga cara yang berbeda untuk menuliskan atau merepresentasikan ekspresi matematika atau logika. Saat diterapkan pada struktur pohon, notasi tersebut merujuk pada urutan penempatan operator dan operand dalam pohon.

Berikut ini adalah representasi notasi prefix, infix, dan postfix untuk ekspresi "A + (B * C)":

1. Prefix

   Notasi prefix, juga dikenal sebagai notasi Polish, menempatkan operator sebelum operand-

operandnya. Dalam pohon, notasi prefix menempatkan operator pada simpul yang menjadi akar, sedangkan operand-operandnya ditempatkan pada simpul-simpul anak. Untuk mengilustrasikan ini.

Notasi Prefix:

a. Ekspresi: A + (B * C)
b. Notasi Prefix: + A * B C

2. InFix

Notasi infix adalah notasi yang paling umum digunakan dalam ekspresi matematika sehari-hari. Dalam notasi infix, operator ditempatkan di antara operand-operandnya. Namun, ketika menerapkan notasi infix pada pohon, kita perlu memperhatikan aturan prioritas operator dan menggunakan tanda kurung untuk mengelompokkan operasi yang harus dievaluasi terlebih dahulu.

Notasi Infix:

a. Ekspresi: A + (B * C)
b. Notasi Infix: A + B * C

3. PostFix

Notasi postfix, juga dikenal sebagai notasi Reverse Polish, menempatkan operator setelah operand-operandnya. Dalam pohon, notasi postfix menempatkan operator pada simpul yang menjadi anak paling kanan, sedangkan operand-operandnya ditempatkan pada simpul-simpul anak lainnya.

Notasi Postfix:

a. Ekspresi: A + (B * C)
b. Notasi Postfix: A B C * +

BAB 10 STRUKTUR DATA GRAPH

Pendahuluan

Dalam bab ini, akan dibahas struktur data graph dalam pemrograman dan bagaimana graph digunakan dan dimanfaatkan untuk menyelesaikan berbagai tantangan pemrograman. Dalam bab ini dibahas juga pengertian graph, tipe-tipe graph, representasi, serta algoritma-algoritma yang melibatkan penggunaan graph.

Graph adalah kumpulan node/vertex (simpul) yang terhubung oleh edge (sisi) yang menggambarkan hubungan antara simpul-simpul tersebut. Simpul dapat merepresentasikan entitas seperti orang, tempat, atau konsep, sedangkan sisi merepresentasikan hubungan atau koneksi antara entitas tersebut.

G = (V, E)

Dimana: G = Graph, Grafik
V = Vertex, Simpul, Node, Titik
E = Edge, Sisi, Tepi

Algoritma graph adalah serangkaian langkah atau prosedur yang dirancang untuk memanipulasi atau menganalisis data yang terstruktur dalam bentuk graph. Algoritma graph memungkinkan untuk melakukan berbagai operasi pada graph, termasuk analisis jaringan, optimisasi rute, pengenalan pola, dan masih banyak lagi. Algoritma-algoritma ini sangat penting dalam memecahkan masalah yang melibatkan entitas dan hubungan antara entitas tersebut.

Tipe-tipe Graph

Graph dapat diklasifikasikan ke dalam beberapa tipe berdasarkan karakteristik:

1. Directed Graph

   Directed graph, juga dikenal sebagai digraph, adalah jenis struktur data graph di mana setiap tepi memiliki arah yang ditentukan. Graph terdiri dari simpul-simpul (nodes) yang terhubung melalui tepi (edges), tetapi perbedaannya dengan undirected graph adalah bahwa dalam directed graph, tepi memiliki arah yang jelas.

   Dalam directed graph, setiap tepi memiliki dua ujung, yaitu ujung awal (source) dan ujung akhir (target). Tepi ini menggambarkan hubungan satu arah antara dua simpul. Misalnya, jika ada tepi dari simpul A ke simpul B, itu berarti ada koneksi atau jalur dari A ke B, tetapi tidak ada jalur langsung dari B ke A melalui tepi yang sama.

Gambar 52. Directed Graph

   Memahami arah tepi dalam directed graph sangat penting, karena arah ini mempengaruhi bagaimana cara melintasi graph. Ketika melakukan traversal (penjelajahan) pada directed graph, hanya dapat melintasi tepi dalam arah yang ditentukan oleh panah.

   Directed graph dapat digunakan untuk menggambarkan berbagai situasi dalam kehidupan nyata. Misalnya, dalam jaringan sosial, directed graph dapat digunakan untuk menggambarkan hubungan "mengikuti" antara pengguna, di mana

kita dapat mengikuti seseorang tetapi tidak diikuti kembali oleh orang yang sama. Directed graph juga berguna dalam memodelkan aliran informasi atau koneksi dalam jaringan komputer, aliran data dalam sistem, dan banyak lagi.

Dengan menggunakan directed graph, dapat memodelkan dan menganalisis berbagai jenis hubungan yang memiliki arah dan memahami bagaimana entitas-entitas terhubung satu sama lain dalam berbagai domain.

2. Undirected Graph

Undirected graph, juga dikenal dengan simple graph, adalah jenis struktur data graph di mana tepi atau sisi (edge) tidak memiliki arah tertentu. Dalam undirected graph, hubungan antara dua simpul (node) adalah simetris, yang berarti jika ada tepi yang menghubungkan simpul A dan simpul B, maka hubungan juga berlaku sebaliknya, yaitu ada tepi yang menghubungkan simpul B dan simpul A.

Gambar 53. Undirected Graph

Dalam undirected graph, setiap tepi hanya menggambarkan adanya koneksi atau relasi antara dua simpul. Tidak ada arah yang ditentukan dalam tepi tersebut, yang berarti koneksi atau jalur antara simpul-simpul adalah dua arah atau bisa dilintasi dalam kedua arah.

Undirected graph sering digunakan untuk menggambarkan hubungan yang bersifat saling mengikat atau simetris, seperti hubungan persahabatan dalam jaringan sosial. Jika simpul A

dan simpul B saling berteman, maka undirected graph dapat digunakan untuk menggambarkan hubungan tersebut dengan tepi yang menghubungkan A dan B tanpa memperhatikan arah.

Pada undirected graph, setiap simpul umumnya dihubungkan dengan beberapa simpul lain melalui tepi. Jumlah tepi yang terhubung dengan simpul tersebut disebut derajat (degree) simpul. Derajat simpul dalam undirected graph menunjukkan seberapa banyak simpul-simpul lain yang terhubung langsung dengan simpul tersebut.

Dengan undirected graph, dapat memodelkan hubungan yang tidak memiliki arah tertentu, mempelajari konektivitas antara simpul-simpul, dan menganalisis berbagai masalah yang melibatkan jaringan, koneksi, atau hubungan yang bersifat simetris.

3. Weighted Graph

   Weighted graph adalah jenis graph di mana setiap sisi atau edge memiliki bobot atau nilai yang terkait. Bobot ini dapat menggambarkan berbagai konsep, seperti jarak fisik antara simpul, biaya, tingkat kepentingan atau kekuatan koneksi antara simpul, waktu yang dibutuhkan untuk melakukan perjalanan antara simpul, atau atribut lainnya yang relevan dalam konteks aplikasi tertentu.

   Dalam weighted graph, setiap sisi memiliki nilai numerik yang dapat digunakan untuk membandingkan kualitas atau pentingnya sisi tersebut. Bobot biasanya dinyatakan dalam bentuk bilangan real atau integer, tetapi bisa juga dalam bentuk lain sesuai kebutuhan aplikasi.

Gambar 54. Weighted Graph

Weighted graph dapat direpresentasikan dalam bentuk matriks bobot (weighted adjacency matrix), di mana setiap elemen matriks mewakili bobot sisi antara simpul-simpul terkait. Alternatifnya, weighted graph juga dapat direpresentasikan dalam bentuk daftar tetangga dengan menyimpan bobot sisi bersama dengan informasi tetangga setiap simpul.

Dengan menggunakan weighted graph, dapat memodelkan dan memecahkan berbagai masalah yang melibatkan pengambilan keputusan berdasarkan bobot atau nilai numerik.

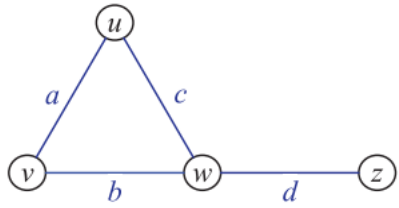
Representasi Graph

Representasi graph adalah cara untuk membuat struktur data graph dalam bentuk kode atau representasi visual. Ada beberapa cara berbeda untuk merepresentasikan graph, dan pilihan metode representasi tergantung pada jenis graph yang digunakan, besaran graph, dan jenis operasi atau algoritma yang akan diaplikasikan pada graph tersebut. Berikut adalah beberapa metode representasi graph yang umum digunakan:

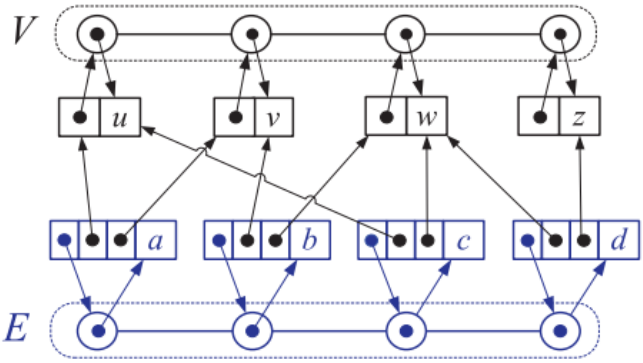
1. Edge List

   Edge List adalah salah satu metode representasi graph yang sederhana dan langsung. Dalam representasi ini, hanya mencatat daftar semua sisi (edge) yang ada dalam graph. Setiap sisi direpresentasikan sebagai pasangan simpul (node) yang terhubung oleh sisi tersebut. Edge List sangat

cocok untuk graph yang memiliki sedikit sisi atau ketika hanya perlu memahami struktur dasar graph tanpa memperhatikan hubungan terperinci antara simpul-simpulnya.



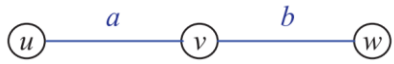
Gambar 55a. Grafik G



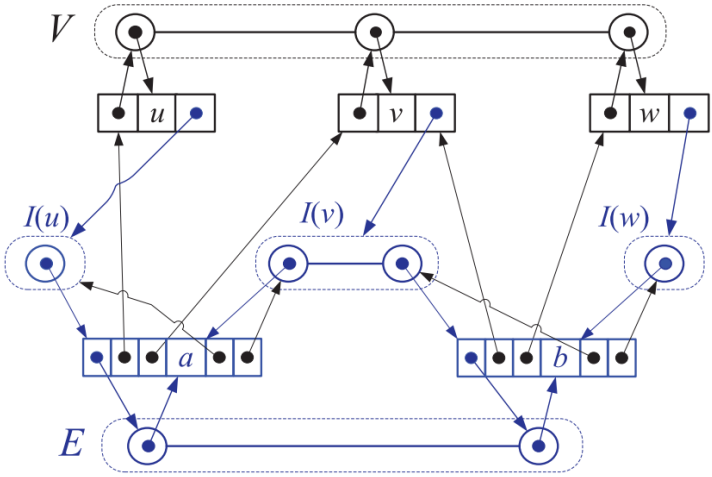
Gambar 55.4b. Representasi Edge List grafik G

2. Adjacency List

   Adjacency List adalah salah satu metode populer untuk merepresentasikan struktur data graph. Dalam representasi ini, setiap simpul (node) dalam graph memiliki daftar simpul-simpul lain yang terhubung langsung dengannya melalui sisi (edge). Masing-masing daftar tersebut mencatat tetangga-tetangga dari simpul tertentu dalam graph. Adjacency List sangat efisien untuk graph yang jarang terhubung, artinya setiap simpul hanya terhubung dengan sejumlah kecil simpul lainnya.



Gambar 56a. Grafik G

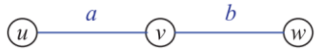


Gambar 56b. Representasi Adjacency List grafik G

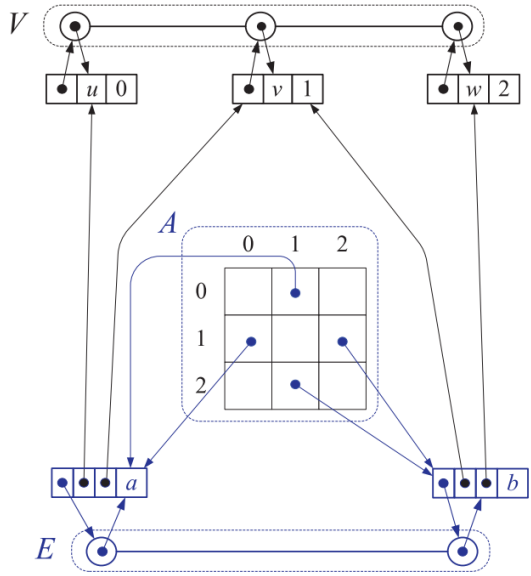
3. Adjacency Matrix

   Adjacency Matrix adalah salah satu metode representasi graph yang menggunakan matriks dua dimensi untuk merepresentasikan hubungan antara simpul-simpul dalam graph. Setiap elemen matriks merepresentasikan apakah ada sisi (edge) yang menghubungkan dua simpul tertentu dalam graph.

   Dalam matriks, baris dan kolom mewakili simpul-simpul dalam graph. Jika ada sisi yang menghubungkan simpul pada baris ke simpul pada kolom tertentu, maka nilai elemen matriks di baris dan kolom tersebut akan bernilai 1. Jika tidak ada sisi yang menghubungkan simpul tersebut, nilai elemen matriksnya akan bernilai 0. Jika graph berbobot, nilai matriks dapat diisi dengan bobot edge.



Gambar 57a. Grafik G



Gambar 57b. Representasi Adjacency Matrix grafik G

Traversal Graph

Traversal graph adalah proses melintasi atau menjelajahi semua simpul (node) dan sisi (edge) dalam sebuah struktur data graph dengan tujuan untuk mengunjungi setiap elemen dalam graph. Proses traversal ini digunakan dalam analisis dan pengolahan graph karena memungkinkan untuk mengakses informasi, menjelajahi keterhubungan, mencari jalur, dan melakukan berbagai operasi pada graph.

Terdapat dua metode utama untuk traversal graph yaitu: Depth-First Search (DFS) dan Breadth-First Search (BFS).

1. Depth-First Search (DFS)

Pada DFS, dimulai dari satu simpul awal dan mengunjungi simpul tetangga selama mungkin sebelum kembali dan menjelajahi cabang lain. Ini dilakukan dengan rekursi atau menggunakan tumpukan. DFS akan mencari sejauh mungkin ke dalam sebelum berpindah ke cabang lain. Ini sering digunakan untuk mencari jalur, mendeteksi siklus, dan menjelajahi komponen terhubung.

2. Breadth-First Search (BFS)
   BFS, sebaliknya, menjelajahi setiap simpul pada level yang sama sebelum melanjutkan ke level berikutnya. Ini dimulai dari simpul awal dan mengunjungi semua tetangganya sebelum melanjutkan ke tetangga dari tetangga tersebut. BFS umumnya digunakan untuk mencari jalur terpendek, menemukan komponen terhubung, dan melakukan pencarian dengan bergerak secara merata.

Kedua metode traversal ini memiliki peran dan kegunaan masing-masing tergantung pada masalah yang ingin dipecahkan dan karakteristik dari graph yang diberikan. Traversal graph menjadi dasar dari banyak algoritma graph lainnya dan membantu dalam memahami serta menganalisis struktur serta hubungan di dalam graph.

Tranversal Digraph

Traversal pada directed graph (digraph) memiliki prinsip yang sama dengan traversal pada graph biasa, tetapi dengan perhatian pada arah sisi (edge). Dalam directed graph, setiap sisi memiliki arah yang menunjukkan hubungan satu arah antara dua simpul. Traversal digraph melibatkan proses menjelajahi semua simpul dan sisi dalam digraph dengan mengikuti arah sisi.

Ada dua metode utama untuk traversal pada

directed graph: Depth-First Search (DFS) dan Breadth-First Search (BFS), yang memiliki karakteristik yang serupa dengan traversal pada graph biasa, tetapi mempertimbangkan arah sisi.

1. Depth-First Search (DFS) pada Digraph
   DFS pada directed graph juga dimulai dari satu simpul awal dan menjelajahi sejauh mungkin ke dalam cabang sebelum kembali. Namun, perbedaannya adalah DFS mengikuti arah sisi, artinya hanya mengunjungi simpul yang dapat dijangkau dari simpul awal secara langsung sesuai dengan arah sisi. DFS pada directed graph digunakan untuk mencari jalur, menghitung derajat keluar dan masuk dari simpul, serta mendeteksi siklus.
2. Breadth-First Search (BFS) pada Digraph
   BFS pada directed graph juga menjelajahi graph secara bertingkat, mengunjungi semua simpul pada level yang sama sebelum bergerak ke level berikutnya. Dalam directed graph, BFS memeriksa tetangga-tetangga yang dapat diakses melalui arah sisi yang keluar dari simpul awal. BFS pada directed graph dapat membantu dalam mencari jalur terpendek antara dua simpul, serta menemukan jalur yang memenuhi arah sisi tertentu.

Traversal pada directed graph membantu dalam menganalisis keterhubungan, mendeteksi siklus, menemukan jalur, dan menjelajahi graph dalam konteks yang sesuai dengan arah sisi. Pemilihan metode traversal tergantung pada masalah yang ingin dipecahkan dan informasi yang ingin diambil dari directed graph.

Shortest Path

Shortest path (jalur terpendek) adalah serangkaian simpul dan sisi yang menghubungkan dua simpul dalam sebuah graph dengan jarak atau biaya minimum di antara semua jalur yang mungkin. Dalam konteks weighted graph, jalur terpendek merujuk pada jalur yang memiliki total bobot atau jarak minimum di antara semua jalur yang menghubungkan dua simpul tertentu.

Jalur terpendek memiliki beberapa aplikasi penting dalam berbagai bidang, seperti transportasi, jaringan komputer, perencanaan rute, dan optimisasi. Dalam graph, terdapat beberapa algoritma yang digunakan untuk menemukan jalur terpendek, termasuk Dijkstra's algorithm, Bellman-Ford algorithm, dan A* algorithm.

Beberapa konsep terkait dengan shortest path adalah:

1. Weighted Graph
   Dalam graph yang memiliki bobot pada sisi-sisinya (weighted graph), jalur terpendek adalah jalur yang memiliki total bobot (jarak, biaya, atau nilai lainnya) paling rendah di antara semua jalur yang menghubungkan dua simpul.
2. Dijkstra's Algorithm
   Dijkstra's algorithm adalah algoritma yang digunakan untuk menemukan shortest path (jalur terpendek) dari satu simpul awal ke semua simpul lainnya dalam weighted graph. Algoritma ini dibuat dengan cara mengembangkan himpunan simpul yang sudah dikunjungi secara bertahap, dengan memilih simpul yang memiliki jarak terpendek dari simpul awal pada setiap langkah. Dijkstra's algorithm hanya cocok untuk graph dengan bobot non-negatif.

Berikut adalah langkah-langkah dari Dijkstra's algorithm:

a. Inisialisasi
    1) Pilih simpul awal sebagai simpul saat ini.
    2) Buat sebuah himpunan (atau priority queue) untuk menyimpan simpul yang belum dikunjungi bersama dengan jarak sementara dari simpul awal ke masing-masing simpul.
    3) Setel jarak awal dari simpul awal ke dirinya sendiri menjadi 0.
    4) Setel jarak awal dari simpul awal ke semua simpul lainnya menjadi tak terhingga (infiniti).
b. Iterasi
    1) Selama ada simpul yang belum dikunjungi dalam himpunan, lakukan langkah berikut:
    2) Pilih simpul dengan jarak terpendek dari simpul awal yang belum dikunjungi. Dalam implementasi dengan priority queue, ini dilakukan dengan mengambil simpul dengan bobot terkecil dari priority queue.
    3) Tandai simpul terpilih sebagai sudah dikunjungi.
    4) Untuk setiap sisi yang keluar dari simpul terpilih, hitung total jarak sementara dari simpul awal ke simpul tujuan melalui simpul terpilih. Jika jarak baru lebih pendek dari jarak sementara sebelumnya, perbarui jarak sementara.
c. Hasil Akhir
    1) Setelah semua simpul telah dikunjungi,

akan didapat jarak terpendek dari simpul awal ke semua simpul lainnya.

2) Dapat membangun jalur terpendek untuk setiap simpul dengan melacak simpul-simpul yang dilewati selama iterasi.

Dijkstra's algorithm memanfaatkan pendekatan greedy (serakah) dengan selalu memilih simpul dengan jarak terpendek dari simpul awal yang belum dikunjungi. Algoritma ini memberikan solusi yang akurat jika semua bobot sisi adalah non-negatif. Namun, jika terdapat sisi dengan bobot negatif, Dijkstra's algorithm mungkin memberikan hasil yang tidak benar.

Minimum Spanning Trees

Minimum Spanning Tree (MST) adalah subgraph yang menghubungkan semua simpul dalam sebuah graph tanpa membentuk siklus dan memiliki total bobot (weight) minimum di antara semua subgraph yang memenuhi kriteria ini. Dalam MST, setiap simpul terhubung dengan simpul lainnya melalui sisi-sisi yang membentuk pohon, sehingga tidak ada sisi yang berlebihan atau siklus.

MST memiliki banyak aplikasi dalam berbagai bidang, termasuk jaringan komunikasi, desain jaringan listrik, dan optimisasi biaya.

Ada beberapa algoritma terkenal yang digunakan untuk menemukan MST dalam graph berbobot:

1. Kruskal's Algorithm

   Kruskal's algorithm bekerja dengan mengurutkan semua sisi berdasarkan bobotnya dan kemudian memilih sisi dengan bobot terkecil satu per satu, dengan catatan tidak membentuk siklus dalam

subgraph yang sudah terbentuk. Algoritma ini menggunakan pendekatan greedy (serakah) dan sering digunakan dalam graph dengan jumlah sisi yang besar. Algoritma ini umumnya digunakan pada undirected weighted graph.

Berikut adalah langkah-langkah Kruskal's Algorithm:

a. Inisialisasi
   Urutkan semua sisi dalam weighted graph berdasarkan bobotnya, dari yang terkecil hingga yang terbesar.
b. Buat Himpunan Disjoint Set (Union-Find)
   Buat himpunan disjoint (setiap simpul ada dalam set sendiri) untuk merepresentasikan setiap simpul sebagai komponen terpisah awalnya.
c. Iterasi
   Untuk setiap sisi yang sudah diurutkan berdasarkan bobotnya, lakukan langkah berikut:
   1) Periksa apakah sisi tersebut akan membentuk siklus jika ditambahkan ke MST yang sedang terbentuk. Jika tidak, tambahkan sisi tersebut ke MST.
   2) Jika sisi tersebut membentuk siklus, lewati sisi tersebut.
d. Hasil Akhir
   Setelah semua sisi telah diiterasi, akan didapat Minimum Spanning Tree yang terbentuk.
   Langkah-langkah Kruskal's Algorithm memastikan bahwa sisi dengan bobot terkecil yang tidak membentuk siklus ditambahkan ke MST, sehingga memastikan bahwa total bobot

dari MST minimal.

Kruskal's Algorithm berguna dalam situasi Ketika ingin menghubungkan semua simpul dalam graph dengan biaya minimal, seperti dalam perencanaan jaringan, desain jaringan listrik, dan pengoptimalan biaya.

2. Prim's Algorithm

   Prim's algorithm dimulai dari satu simpul awal dan secara berulang memilih sisi dengan bobot terkecil yang menghubungkan simpul yang sudah ada dalam subgraph yang sedang terbentuk. Ini juga menggunakan pendekatan greedy dan sering lebih efisien daripada Kruskal's algorithm dalam graph yang padat.

3. Boruvka's Algorithm

   Algoritma ini menggunakan pendekatan berbasis komponen terhubung dan berjalan secara iteratif untuk memilih sisi terkecil dari setiap komponen terhubung yang berbeda. Boruvka's algorithm sangat efisien dalam graph yang jarang terhubung.

Minimum Spanning Tree memiliki beberapa sifat penting:

1. MST adalah subgraph yang mengandung semua simpul dari graph awal.
2. Jika graph asli memiliki N simpul, MST akan memiliki N-1 sisi.
3. Dalam graph yang tidak memiliki bobot, MST dapat ditemukan dengan algoritma breadth-first search (BFS) atau depth-first search (DFS), karena tidak ada perbedaan dalam pemilihan sisi.
4. Dalam graph berbobot, MST menunjukkan jalur terpendek yang menghubungkan semua simpul dengan bobot total minimum.

Pemilihan algoritma MST tergantung pada

karakteristik graph (jumlah simpul, sisi, bobot), tujuan optimisasi, dan kecepatan yang dibutuhkan. MST merupakan konsep yang penting dalam teori graph dan sering digunakan dalam perencanaan jaringan dan optimisasi sumber daya.

BAB 11 SORTING

Pendahuluan

Pada kehidupan sehari-hari, aktivitas mengurutkan sesuatu adalah hal yang sangat biasa dilakukan. Menyusun barisan pada sebuah kegiatan upacara, dimana peserta upacara diurutkan mulai yang terpendek di barisan terdepan hingga yang paling tinggi di akhir barisan, menyusun dokumen pada sebuah filling cabinet sesuai dengan kronologis waktu pembuatan dokumen-dokumen tersebut, menyusun kartu kehadiran siswa sekolah sesuai dengan urutan abjad nama siswa, dan banyak lain lagi.

Pada beberapa kasus pemrograman, seperti aplikasi kamus elektronik, aplikasi buku telepon, sistem informasi perpustakaan, dan lainnya, untuk memudahkan proses pencarian data perlu dilakukan proses pengurutan data pada program tersebut, atau dilakukan mekanisme penempatan data terurut pada saat data tersebut diinput pada program tersebut. Hal ini menunjukkan bahwa metode pengurutan data (sorting) kerap digunakan dan penting untuk dipahami dalam mempelajari struktur data.

Terdapat banyak metode pengurutan data, seperti Insertion Sort, Selection Sort, Bubble Sort, Heap Sort, Merge Sort, Radix Sort, dan banyak lainnya. Namun, dari antara sekian metode tersebut ada 3 (tiga) algoritma dasar yang kerap digunakan pada proses pengurutan data, yaitu metode:

1. Insertion Sort
2. Selection Sort

3. Exchange Sort

Insertion Sort

Konsep pengurutan dengan metode penyisipan atau insertion sort adalah menempatkan suatu besaran (bilangan) pada tempatnya. Kalau dibuat analogi dalam kehidupan sehari-hari nya adalah dalam menyusun kartu bridge, dimana jika diurutkan maka posisi teratas adalah kartu As, kemudian kartu 1 hingga 10, dan diikuti dengan kartu Jack, Queen dan King. Posisi awal adalah seluruh kartu berserak di atas meja. Kemudian kita mengambil satu persatu kartu tersebut dan menyusunnya sesuai dengan posisinya, bisa di atas kartu pertama, diantara 2 kartu atau di posisi terakhir pada tumpukan yang kita buat.

Gambar 58. Tumpukan kartu bridge (Sumber: Tahupedia.com)

Contoh:

Diketahui sebuah array berisi data: [5 1 7 2 4] yang akan kita urutkan secara ascending sebagai berikut:

Iterasi	Hasil pengurutan	Keterangan
1	1 5 7 2 4	Elemen array (El.) ke-2 (1) disisipkan di posisi ke-1 (dengan dipertukarkan)
2	1 5 7 2 4	El. ke-3 (7) sudah sesuai
3	1 2 5 7 4	El. ke-4 (2) disisipkan di posisi ke-2
4	1 2 4 5 7	El. ke-5 (4) disisipkan di posisi ke-3

Pengurutan dilakukan dengan memeriksa el. ke-2, ke-3, hingga ke posisi terakhir (N). Jika pada sebuah posisi el. belum terurut mulai dari el. pertama hingga el. tersebut, maka dilakukan pertukaran hingga menjadi terurut. Begitu dilakukan secara berulang dengan pemeriksaan dari posisi el. ke-2 hingga terakhir dengan jumlah perulangan/iterasi sebanyak N-1.

Algoritma Insertion Sort:

Data yang diurutkan disimpan pada array Data [1..N]

Procedure InsertionSort

```
    For I = 2 to N do
        While (Data[I-1] > Data[I]) and (I>1) do
            Temp = Data[I]
            Data[I]=Data[I-1]
            Data[I-1]=Temp
            I = I – 1
        Endwhile
    Endfor
End procedure
```

Selection Sort

Konsep pengurutan dengan metode pemilihan atau seleksi adalah mencari data terkecil untuk ditempatkan di awal barisan, lalu dari posisi ke-2 hingga data terakhir dicari kembali yang terkecil untuk diletakkan di posisi ke-2. Demikian selanjutnya untuk posisi ke-3 hingga ke-(N-1) (karena data di posisi ke N setelah proses pengurutan adalah yang terbesar). Analogi dalam kehidupan sehari-hari adalah pada saat kita sedang menyusun posisi sekelompok orang dalam sebuah barisan dimana tinggi

badan peserta barisan harus terurut. Dari sekelompok orang peserta barisan, dicarilah yang paling pendek untuk diletakkan di posisi awal barisan. Kemudian dari yang masih belum masuk barisan dicari lagi yang terkecil untuk diletakkan diposisi ke-2, dan seterusnya hingga orang terakhir yang tersisa adalah yang paling tinggi dan berdiri di akhir barisan.

Contoh:

Diketahui sebuah array berisi data: [5 1 7 2 4] yang akan kita urutkan secara ascending dengan metode selection sort sebagai berikut:

Iterasi	Hasil pengurutan	Keterangan
1	1 5 7 2 4	Elemen array (El.) ke-1 adalah bilangan terkecil dari seluruh el. yang ada (1). Sebelumnya 1 berada di posisi ke-2. El. tersebut ditukar dengan 5 di posisi ke-1
2	1 2 7 5 4	El. terkecil ke-2 adalah (2) di posisi ke-4, ditukar dengan (5) di posisi ke-2
3	1 2 4 5 7	El. terkecil ke-3 adalah (4) di posisi ke-5, ditukar dengan (7) di posisi ke-3
4	1 2 4 5 7	El. terkecil ke-4 adalah (5) di posisi ke-5. Karena posisi sudah sesuai, tidak ada pertukaran el.

Pengurutan dilakukan dengan mencari bilangan terkecil dari posisi pertama hingga terakhir untuk ditepatkan di posisi ke-1. Jika posisi bilangan tersebut tidak di posisi ke-1, dilakukan pertukaran dengan bilangan di posisi ke-1. Begitu selanjutnya, untuk posisi ke-2, dicari

yang terkecil dari posisi ke-2 hingga terakhir, dan ditukar dengan yang ada di posisi ke-2, sampai pada posisi N-1. Jika ternyata yang terkecil sudah berada di tempatnya, maka tidak perlu dilakukan pertukaran data.

Algoritma Selection Sort:

Data yang diurutkan disimpan pada array Data [1..N]

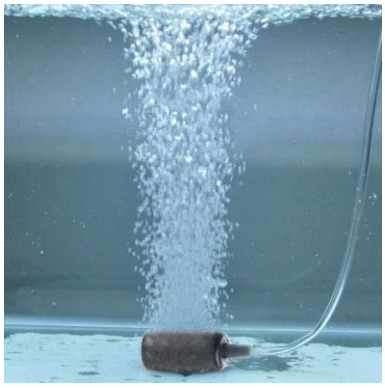
Procedure SelectionSort

```
    For I = 1 to N-1 do  {mencari bilangan terkecil}
       Min = Data[I]
       PosMin = I
       For J = I+1 to N do
              If Data[J] < Min then
                     Min = Data[J]
                     PosMin = J
              Endif
       Endfor
       If I != PosMin then  {menukar bilangan terkecil}
              Tmp = Data[I]
              Data[I] = Data[PosMin]
              Data[PosMin] = Tmp
       Endif
    Endfor
End Procedure
```

Exchange Sort

Pengurutan jenis ini sering dikenal dengan istilah Bubble Sort, dimana selama sebuah data/bilangan dalam sebuah barisan/array belum terurut, harus terus dilakukan pertukaran hingga status setiap bilangan terurut dengan bilangan di sebelahnya.

Mengapa dikenal dengan istilah Bubble Sort? Analogi metode ini adalah dengan melihat sifat gelembung-gelembung udara di dalam air. Gelembung-gelembung udara di dalam air akan bergerak ke permukaan air. Begitu pula dalam metoda pengurutan ini, bilangan-bilangan yang "ringan" akan cenderung "naik ke permukaan". Berikut contoh ilustrasi gelembung air di dalam air.



Gambar 59. Ilustrasi gelembung air (Sumber: aliexpress.com)

Jadi metode pengurutan ini adalah memeriksa setiap bilangan mulai dari posisi terakhir hingga ke awal barisan. Untuk bilangan di posisi N, selama bilangan di posisi N-1 masih lebih besar, maka harus dilakukan pertukaran. Kegiatan ini dilakukan selama status setiap iterasi pengurutan masih "belum urut". Untuk itu dperlukan sebuah flag yang menyimpan status sudah urut atau belum.

Contoh:

Diketahui sebuah array berisi data: [5 1 7 2 4] yang akan kita urutkan secara ascending dengan metode exchange sort sebagai berikut dimana iterasi dilakukan selama status masih belum urut dan pemeriksaan bilangan dilakukan dari posisi terakhir hingga posisi ke-2:

Iterasi	Hasil pengurutan	Keterangan
0	5 1 7 2 4	Posisi awal. Status URUT = true. Dilakukan pemeriksaan dari posisi el. ke-5 hingga ke-2.
1	1 5 7 2 4 1 5 7 2 4 1 5 2 7 4 1 2 5 7 4	- Status awal (URUT = true) - Posisi ke-5 dan ke-4 sudah urut - Posisi ke-4 dan ke-3 harus ditukar, status URUT berubah menjadi false - Posisi ke-3 dan ke-2 harus ditukar Karena masih ada pertukaran, status URUT = false
2	1 2 5 7 4 1 2 5 4 7 1 2 4 5 7 1 2 4 5 7	- Status awal (URUT = true) - Posisi ke-5 dan ke-4 harus ditukar, status URUT menjadi false - Posisi ke-4 dan ke-3 harus ditukar, - Posisi ke-3 dan ke-2 sudah urut Karena masih ada pertukaran, status URUT = false
3	1 2 4 5 7	- Status awal (URUT=true) - Posisi ke-5 dan ke-4 sudah urut - Posisi ke-4 dan ke-3 sudah urut - Posisi ke-3 dan ke-2 sudah urut Karena tidak ada pertukaran, status URUT masih true dan kegiatan pengurutan dihentikan

Jumlah iterasi pada metode pengurutan exchange sort ini pada kasus tertentu lebih sedikit dibandingkan dua metode lainnya (yang memerlukan N-1 iterasi), apalagi jika data sudah terurut, maka hanya diperlukan satu iterasi saja untuk memeriksa data - data yang ada.

Algoritma Exchange Sort:

Data yang diurutkan disimpan pada array Data [1..N]

Terdapat variabel logika URUT = false

Procedure ExchangeSort

```
URUT = false
While (Not URUT) do
    URUT = true; I = N;
    While I > 1 do
        If Data[I] < Data[I-1] then
            Tmp = Data[I]
            Data[I] = Data[I-1]
            Data[I-1] = Tmp
            URUT = false
        Endif
        I = I - 1
    Endwhile
Endwhile
```

End procedure

Contoh Kasus:

Terdapat sebuah array A = [6 1 5 7 4 2 9]
Lakukan proses pengurutan dengan metode :

a. Insertion Sort
   Posisi awal : [6 1 5 7 4 2 9]
   Iterasi-1 : [1 6 5 7 4 2 9], bil 1 ke posisi ke-1
   Iterasi-2 : [1 5 6 7 4 2 9], bil 5 ke posisi ke-2
   Iterasi-3 : [1 5 6 7 4 2 9], bil 7 tidak berpindah
   Iterasi-4 : [1 4 5 6 7 2 9], bil 4 ke posisi ke-2
   Iterasi-5 : [1 2 4 5 6 7 9], bil 2 ke posisi ke-2
   Iterasi-6 : [1 2 4 5 6 7 9], bil 9 tidak berpindah
   Ada 6 iterasi/perulangan → N - 1

b. Selection Sort
   Posisi awal : [6 1 5 7 4 2 9]
   Iterasi-1 : [1 6 5 7 4 2 9], min = 1, tukar posisi-1 dg 2
   Iterasi-2 : [1 2 5 7 4 6 9], min = 2, tukar posisi-2 dg 6
   Iterasi-3 : [1 2 4 7 5 6 9], min = 4, tidak tukar posisi
   Iterasi-4 : [1 2 4 5 7 6 9], min = 5, tukar posisi-4 dg 5
   Iterasi-5 : [1 2 4 5 6 7 9], min = 6, tukar posisi-5 dg 6
   Iterasi-6 : [1 2 4 5 6 7 9], min = 7, tidak tukar posisi
   Ada 6 iterasi (N-1)

c. Exchange Sort
   Posisi awal : [6 1 5 7 4 2 9]
   Iterasi-1 : [1 6 2 5 7 4 9], Urut = false
   Iterasi-2 : [1 2 6 4 5 7 9], Urut = false
   Iterasi-3 : [1 2 4 6 5 7 9], Urut = false
   Iterasi-4 : [1 2 4 5 6 7 9], Urut = false
   Iterasi-5 : [1 2 3 5 6 7 9], Urut = true
   Ada 5 iterasi

Dari contoh kasus di atas terlihat bahwa jumlah iterasi pada pengurutan dengan metode Insertion dan Selection harus sejumlah N – 1, sedangkan untuk metode Exchange maksimum sejumlah N kali. Pada kasus terurut secara descending, metode Exchange memerlukan N iterasi, sedangkan jika data sudah terurut ascending hanya diperlukan 1 iterasi oleh metode pengurutan Exchange. Perbandingan Kompleksitas Algoritma:

Metode Pengurutan	Jumlah iterasi pada Best Case	Jumlah iterasi pada Worst Case
Insertion	N-1	N-1
Selction	N-1	N-1
Exchange	1	N

BAB 12 SEARCHING

Pendahuluan

Searching atau pencarian adalah proses yang sangat penting dalam dunia pemrograman, karena pencarian ini merupakan menemukan data kembali yang telah disimpan di media penyimpanan. Data yang telah disimpan sudah dapat dipastikan akan dicari dan ditampilkan kembali baik untuk berbagai kebutuhan seperti pelaporan, proses update atau koreksi serta proses delete atau hapus.

Dalam proses pencarian diperlukan masukkan berupa data yang dicari yang menjadi kunci pencarian di dalam data yang telah disimpan. Kunci yang digunakan untuk pencarian harus memiliki tipe yang sama dengan field tempat data yang akan dicari.

Hasil dari pencarian adalah indeks atau urutan tempat data disimpan. Pada saat pencarian ada dua kemungkian yang terjadi pertama data ditemukan dan diketahui indeks data disimpan dan yang kedua adalah data tidak ditemukan.

Algoritma pencarian (searching algorithm) adalah algoritma yang menerima argument K (sebagai kunci), dan dengan langkah-langkah tertentu akan mencari rekaman yang kuncinya bernilai K. (Insap Santosa, 2004). Algoritma pengurutan dan pencarian memuat banyak sekali konsep dasar pemrograman, yaitu: (Rinaldi Munir, 2011)

1. Runtunan (sequence). Kaidah pemrograman yang menyatakan bahwa perintah-perintah dalam program Komputer akan dieksekusi menurut urutan

dari atas ke bawah.

2. Seleksi (selection), Perintah-perintah dalam pemrograman komputer akan dieksekusi berdasarkan nilai kebenaran boolean tertentu.
3. Perulangan (loop). Sejumlah perintah dalam program komputer yang akan dieksekusi beberapa kali (berulang) berdasarkan nilai kebenaran boolean-nya.

Kecepatan dalam pencarian bergantung pada beberapa faktor salah satunya algoritma serta jumlah data. Pada bab berikut ini akan dibahas tentang dua algoritma pencarian yang paling dasar.

Algoritma Searching

Algoitma searching yang paling sederhana adalah algoritma Sequential Search, algoritma ini akan dibahas pada sub bab pertama. Sub bab kedua akan dibahas mertode Binary Search.

Sequential Search

Sequential Search adalah algoritma pencarian yang teknik pencarian datanya menelusuri semua elemen-elemen data dari awal dibandingkan apakah sama dengan elemen kunci (data yang dicari) bila ketemu, maka pencarian akan berakhir jika tidak akan dilanjutkan pencariannya sampai sampai akhir data.

Pada algoritma pencarian Sequential Search memiliki kelebihan yaitu data yang dicari tidak diharuskan untuk diurutkan terlebih dahulu. Kelemahan dari algoritma ini adalah bila pencarian yang dilakukan datanya ketemu pada index yang terakhir atau sering disebut worst case. Dibalik kelemahan ada juga keuntungannya yaitu bila data yang dicari terdapat di awal index, sehingga pencarian

cukup satu kali pencarian data sudah ditemukan atau sering disebut dengan best case. Kelembahan berikutnya adalah semakin banyak datanya semakin banyak pula waktu yang diperlukan.

Pembahasan Algoritma sequential search ini agar mudah dipahami akan disajikan dalam bentuk Natural Language dan Flowchart. Selanjutnya implementasi ke program C++.

1. Algoritma Sequential Search dengan Natural Language

   Berikut langka-langkah pencarian dengan menggunakan algoritma Sequential Search.

   a. Data = (50, 75, 38, 67, 41, 81)
   b. Berikan harga awal untuk variabel penanda bahwa data ditemukan
      KETEMU = false
   c. Masukkan data kunci yang dicari
      CARI = 67 (misalnya data yang dicari = 67)
   d. Masuk dalam perulangan untuk membaca data yang telah tersimpan kemudian dibandingkan dengan data kunci pencarian dengan elmen data pertama
      Apakah CARI = DATA[1]
      bila data pertama tidak sama lanjut pada data kedua dan seterusnya sampai data ke N.
   e. Bila ketemu maka variabel KETEMU = true dan catat index data tempat data ditemukan.
   f. Bila semua data telah dibandingkan dan variabel KETEMU = false, maka buat pesan "DATA TIDAK KETEMU"
   g. Selesai.

   Dari Langkah-langkah di atas akan dicoba untuk menerapkan dengan contoh kasus data = (50, 75, 38, 67, 41, 81) dan berikut Langkah-langkah pencarian

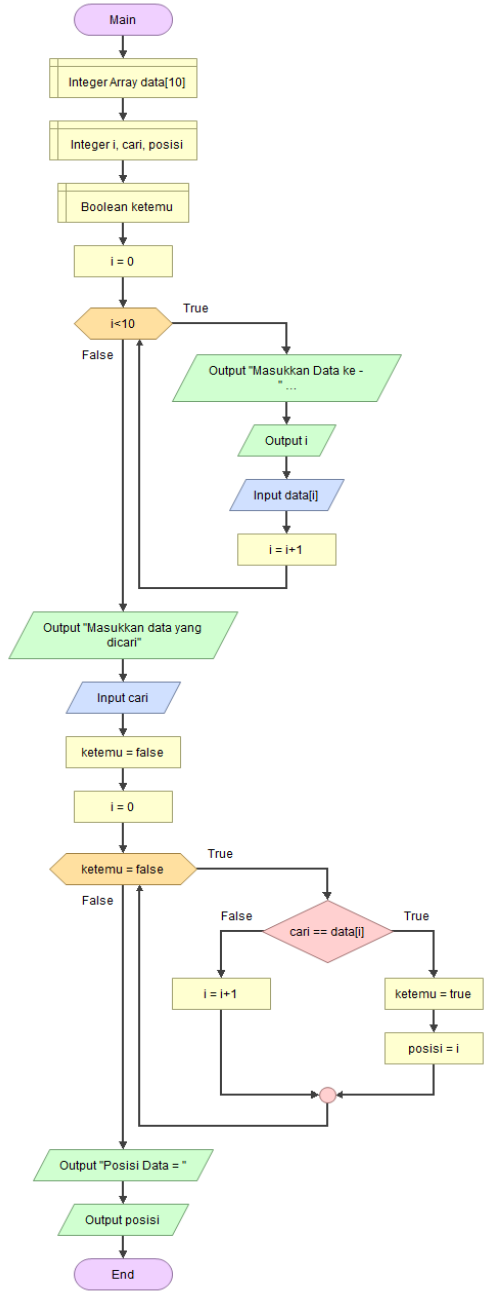
sesuai dengan algoritma di atas:

a. Data[1] = 50
   Data[2] = 75
   Data[3] = 38
   Data[4] = 67
   Data[5] = 41
   Data[6] = 81
b. KETEMU = false
c. CARI = 67
d. Masuk dalam perulangan
   Apakah Data[1] = 67?
   50 = 67 tidak
   Apakah Data[2] = 67?
   75 = 67? Tidak
   Apakah Data[3] = 67?
   38 = 67? Tidak
   Apakah Data[4] = 67?
   67 = 67? Ya
e. KETEMU = true

Dari pencarian di atas data ditemukan pada indeks ke-4, variabel KETEMU sebagai indikator

2. Flowchart Program Algoritma Sequential Search

   Flowchart program algoritma Sequential Search akan digambarkan dengan menggunakan flowgorithm. Flowgorithm adalah perangkat lunak yang khusus untuk menggambar flowchart untuk membantu pemula dalam belajar pemrograman dasar. Kelebihan dari flowgorithm adalah gratis dan terdapat fitur konversi lebih dari 12 bahasa pemrograman.



Gambar 58. Flowchart Input data dan Squential Search

3. Implementasi Algoritma Sequential Search
   Implementasi algoritma Sequential Search ke dalam Bahasa pemrograman C++

```
#include <iostream>
(http://www.flowgorithm.org/, n.d.)
using namespace std;

int main()
{
   int data[10];
   int i, cari, posisi;
   bool ketemu;

   i = 0;
   while (i < 10) {
      cout << "Masukkan Data ke - ";
      cout << i << " " ;
      cin >> data[i];
      i = i + 1;
   }
   cout << "Masukkan data yang dicari" ;
   cin >> cari;
   ketemu = false;
   i = 0;
   while (ketemu == false) {
      if (cari == data[i]) {
         ketemu = true;
         posisi = i;
      } else {
         i = i + 1;
      }
   }
   if (ketemu == true) {
```

```
        cout << "Posisi Data = " << posisi << endl;
    } else {
        cout << "data tidak ketemu" << endl;
    }
    return 0;
}
```

Binary Search

Binary Search adalah algoritma pencarian yang syarat utamanya adalah data harus terurut. Proses pencariannya diawali dengan mencari index tengah dari data. Nilai dari data yang indexnya adalah tengah ini akan dibandingakan dengan data yang dicari. Bila data data yang dicari sama dengan data yang indexnya tengah maka data akan ketemu.

Pada awal pencarian seluruh data akan dijadikan tempat pencarian yaitu dari posisi 1 sampai dengan posisi akhir (N) tergantung dari jumlah data. Pada buku ini implementasi programnya menggunakan C++, maka index arraynya dimulai dari 0 (Nol), maka data pertama memiliki index 0. Berikut Langkah-langkah algoritma pencariannya:

1. Masukkan data dari index 0 (nol) sampai terakhir (N)
2. Selanjutnya cari index tengah dengan rumus indextengah = int (indexawal + indexakhir) / 2
   Int disini adalah harga mutlak dari hasil pembagian
3. Kemudian data yang dicari dibandingkan dengan data yang indexnya adalah hasil dari perhitungan indextengah, yang memiliki tiga kemungkinan yaitu sama dengan, lebih kecil, atau lebih besar.
    a. Bila data yang dicari lebih besar dan data terurut dari kecil ke besar (ascending) maka proses pencarian akan berpindah diposisi

kanan dari indextengah. Area pencarian akan menyempit yaitu indexawal akan berubah, berikut rumus untuk indexawal yang baru yaitu indexawal = indextengah + 1
Dilanjutkan Langkah 2

b. Jika lebih kecil, maka proses pencarian dilakukan pada posisi kiri dari index tengah. Area pencarian akan menyempit yaitu indexakhir akan berubah, berikut rumus untuk indexakhir yang baru yaitu Indexakhir = indextengah – 1
Dilanjutkan Langkah 2

c. Jika data sama, berarti ketemu.
Contoh
Misalnya data yang dicari 17
Catatan yang diasir merupakan are pencarian

Langkah 1
Awal = 0 akhir=8
0 1 2 3 4 5 6 7 8 › index
3 9 11 12 15 17 23 31 35 › data
indextengah = int (indexawal+indexakhir)/2
indextengah = int(0+8)/2=4
Apakah data [indextengah] = 17?(tdk)
Karena 17 > 15, maka lanjut Langkah 2

Langkah 2
indexawal = indextengah + 1
indexawal = 4 + 1 = 5 (baru) akhir=8 (tetap)
0 1 2 3 4 5 6 7 8
3 9 11 12 15 17 23 31 35 (pencarian fokus di data sebelah kanan)
indextengah = int (indexawal+indexakhir)/2
indextengah = int (5+8)/2=6

Apakah data[indextengah] = 17 (tdk)
Karena 17 < 23 , maka lanjut Langkah 3

Langkah 3
indexakhir = indextengah – 1
indexakhir = 6 – 1 = 5
indexawal = 5 (tetap)

0	1	2	3	4	5	6	7	8
3	9	11	12	15	17	23	31	35

Karena 17 = 17 (data tengah), maka KETEMU!

Pembahasan Algoritma Binary Search ini disajikan dalam bentuk Natural Language dan Flowchart. Selanjutnya implementasi ke program C++.

1. Algoritma Binary Search dengan Natural Language
   Berikut Algoritma Binary Search
   a. Input data yang sudah terurut
      Data[0] = 21
      Data[1] = 27
      Data[2] = 31
      Data[3] = 37
      Data[4] = 41
   b. Input data yang akan dicari
   c. Memberikan harga awal variabel
      Awal = 0
      Akhir =4
      Ketemu = false
      Tengah = (awal + akhir)/2
   d. Masuk dalam perulangan dengan kondisi awal lebih kecil sama dengan akhir
      1) Jika data[tengah] < cari,
         Jika benar maka pencarian dilakukan disebelah kanan dari data[tengah] dengan cara merubah awal = tengah + 1
      2) Jika salah, apakah data[tengah] = cari

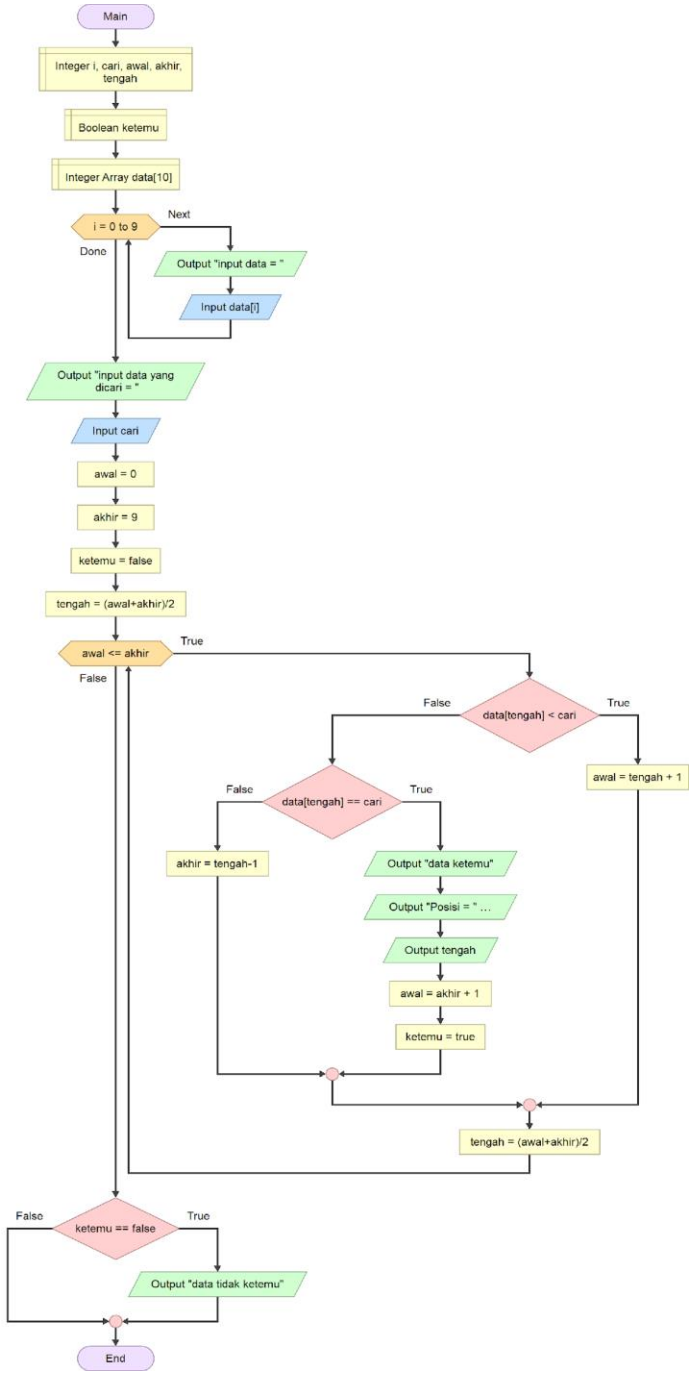
jika benar data yang dicari ketemu

3) Jika salah, maka pencarian dilakukan disebelah kiri, karena data[tengah] > cari dengan cara merubah akhir = tengah – 1

e. Hitung ulang nilai tengah Kembali ke step 4

2. Flowchart Program Algoritma Binary Search

Flowchart Program Algoritma Binary Search di gambar dengan menggunakan flowgorithm.



Gambar 59. Flowchart Program Binary Search

3. Implementasi Algoritma Binary Search
   Program pencarian dengan metode Binary Search akan ditulis dengan Bahasa pemrograman c++.

```
#include <iostream>

using namespace std;

int main()
{
   int i, cari, awal, akhir, tengah;
   bool ketemu;
   int data[10];

   for (i = 0; i <= 9; i++) {
      cout << "input data = ";
      cin >> data[i];
   }
   cout << "input data yang dicari = " << endl;
   cin >> cari;
   awal = 0;
   akhir = 9;
   tengah = (awal + akhir) / 2;
   ketemu = false;
   while (awal <= akhir) {
      if (data[tengah] < cari) {
         awal = tengah + 1;
      } else {
         if (data[tengah] == cari) {
            cout << "data ketemu" << endl;
            cout << "Posisi = ";
            cout << tengah << endl;
            awal = akhir + 1;
            ketemu = true;
```

```
            } else {
                akhir = tengah - 1;
            }
        }
        tengah = (awal + akhir) / 2;
    }
    if (ketemu == false) {
        cout << "data tidak ketemu" << endl;
    }
    return 0;
}
```

Daftar Pustaka

Abdul Kadir, 2015. “Teori dan Aplikasi Struktur Data mengunakan Java”, Andi Publisher.

Asyiknya Belajar Struktur Data di Planet C++. (n.d.). (n.p.): Elex Media Komputindo.

B Herawan, H. (2022). Buku Algoritma Dan Struktur Data.

Balagurusamy, E. (2019). Data Structures. McGraw Hill Education (India) Private Limited.

Chaniago, M. B., & Sastradipraja, C. K. (2022). Buku Ajar Algoritma dan Struktur Data. Kaizen Media Publishing.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). The MIT Press.

Das, V. V. (2006). Principles of Data Structures Using C and C++. New Age International Publisher.

Das, V. V. (2006). Principles of Data Structurs Using C and C++. New Delhi: New Age International (P) Limited, Publishers.

Drozdek, A. (2013). Data Structures and Algorithms in C++ (Fourth Edi). Cengage Learning.

Dwi Rahmatya, M. (2020). Praktikum Array Dan Struct.

Efendi, Y. (2022). Buku Ajar Pendidikan Algoritma dan Struktur Data.

Efendi, Y. (2022). Buku Ajar Pendidikan Algoritma dan Struktur Data.

GeeksforGeeks. (2023, 3 29). Linked List vs Array. Diakses dari https://www.geeksforgeeks.org/linked-list-vs-array/

GeeksforGeeks. (2023, 6 7). Understanding the basics of Linked List. Diakses dari https://www.geeksforgeeks.org/what-is-linked-list//

Gootooru, N. (2023, 06 25). double-ended-queue. Retrieved from java2novice.com/: https://www.java2novice.com/data-structures-in-java/queue/double-ended-queue/

Hariyanto, B. (2000). Buku Teks Ilmu Komputer Struktur Data. Informatika.

Info. (n.d.). Retrieved 7 7, 2023, from http://www.flowgorithm.org/about/info.htm

Jay Wengrow, 2018. “Data Structures and Algorithms: Level Up Your Core Programming Skills 1st Edition, Kindle Edition”.

Johnson, M. (2018). Programming Language Pragmatics. Morgan Kaufmann.

Lafore, R. (2017). Data Structures and Algorithms in Java. Sams Publishing.

Lafore, R. (2017). Data Structures and Algorithms in Java. Sams Publishing.

Lee, M. (2009). Programing: C++ Programming for the Absolute Beginner (Second Ed.). CENGAGE Learning.

Log2Base2. (2023, 5 9). Linked List. Diakses dari

https://www.log2base2.com/data-structures/linked-list/linked-list-in-c.html

Michael T. Goodrich, Roberto Tamassia, David M. Mount, 2011, “Data Structures and Algorithms in C++”, Second Edition, John Wiley & Sons.

Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, 2014, “Data Structures and Algorithms in Java™”, Sixth Edition, John Wiley & Sons.

Muhamad, A. (2013). Buku Algoritma dan Pemrograman.

Munir, R. (2011). Algoritma dan pemrograman / Rinaldi Munir. Retrieved 7 7, 2023, from http://library.um.ac.id/free-contents/index.php/buku/detail/algoritma-dan-pemrograman-rinaldi-munir-42910.html

Noname. (2023, 5 1). queue. Retrieved from www.programiz.com: https://www.programiz.com/dsa/queue

Novia Hasdyna, H., & Rozzi Kesuma Dinata, D. Buku Pembelajaran Struktur Data Dalam Pemrograman C++.

Pal, D., & Halder, S. (2018). Data Structure & Algorithm With C. Alpha Science International Ltd.

Prata, S. (2013). C Primer Plus (6th ed.). Addison-Wesley Professional.

Pratama, M. A. (2020). Struktur Data Array Dua Dimensi Pada Pemrograman C++.

Putri, M. P., Barovih, G., Azdy, R. A., Yuniansyah, Y.,

Saputra, A., Sriyeni, Y., Rini, A., & Admojo, F. T. (2022). Algoritma Dan Struktur Data.

Rosa, A. S. (2018). Struktur Data Terapan Dalam Berbagai Bahasa Pemrograman Buku Pemrograman.

Sabila, N. K. (n.d.). Pembahasan Struktur Data Dan Bahasa Pemograman.

Santosa, P. I. (1992). Struktur data menggunakan turbo pascal 6.0 / P. Insap Santosa. Retrieved 7 7, 2023, from http://library.um.ac.id/free-contents/index.php/buku/detail/struktur-data-menggunakan-turbo-pascal-p-insap-santosa-36991.html

Santosa, P. I. (n.d.).

Schilthuizen, C. (2019). Optimized C++. O'Reilly Media. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). The MIT Press.

Seacord, R. C. (2005). Secure Coding in C and C++. Addison-Wesley Professional.

Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley Professional.

Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley Professional.

Setyaningsih, E. (2012). Struktur Data. Akprind Press.

Siahaan, V. (2020). Buku Resep Algoritma dan Struktur Data dengan Java. Balige Publishing.

Siahaan, V. (2020). Buku Resep Algoritma dan Struktur Data dengan Java. Balige Publishing.

Smith, J. (2010). C Programming Language. Addison-Wesley.

Stroustrup, B. (1997). The C++ Programming Language (Third Ed.). Addison Wesley.

Struktur Data & Pemrograman Dengan Pascal. (n.d.). (n.p.): Penerbit Andi.

Struktur Data Dan Implementasi Algoritma (SDIA): Pemrograman Python C C++ Java. (2022). (n.p.): Zayid Musiafa.

Struktur Data Dengan Python. (2021). (n.p.): Ahlimedia Book.

Suhendar, A. (2019). Struktur data sederhana (statis array).

Suryawan, F., Thamrin, H., Anggoro, D. A., Supriyanti, W., & Ardiyanto, Y. (2020). Modul Praktikum Algoritma & Struktur Data Versi 4.3. Muhammadiyah University Press.

Tarigan, W. (2022). Algoritma Pemrograman dan Struktur Data.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, 2009, “Introduction to Algorithms”, Third Edition, The MIT Press, Massachusetts London.

Visser, H. J. (2006). Array and phased array antenna basics. John Wiley & Sons.

w3resource. (2023, 5 20). C Programming Exercises,

Practice, Solution: Linked List. Diakses dari https://www.w3resource.com/c-programming-exercises/linked_list/index.php

Weiss, M. A. (2016). Data Structures and Algorithm Analysis in C++. Pearson. Jones, P., & Ohlund, J. (2012). Advanced C Programming. Pearson Education.

Weiss, M. A. (2016). Data Structures and Algorithm Analysis in C++. Pearson. Davis, J. (2016). Low-Level Programming: C, Assembly, and Program Execution on Intel® 64 Architecture. Apress.

Zein, A., & Eriana, E. S. (2022). Algoritma Dan Struktur Data. Unpampress.

Tentang Penulis

Subhan Hafiz Nanda Ginting, Lahir dan besar di Medan dengan latar belakang pendidikan S2 Teknik Informatika dari Universitas Sumatera Utara. Saat ini aktif sebagai dosen tetap prodi Sistem Informasi fakultas Teknologi Universitas Battuta. Selain mengajar penulis juga aktif dalam organisasi bidang keilmuan seperti APTIKOM, IPKIN, KORIKA. Bidang keahlian penulis berfokus kepada Software Engineering, Expert System dan Computer Network.

Hansi Effendi, buku ini adalah salah satu karya dan inshaa allah secara konsisten akan disusul dengan buku-buku berikutnya. Pokok bahasan buku yang ditulis semata-mata untuk berbagi ilmu pengetahuan.

Suresh Kumar, S.T., M.Si, Lahir di Kota Tanjung Balai, Provinsi Sumatera Utara, pada tanggal 7 September 1976. Menyelesaikan pendidikan Sarjana Teknik Elektro jurusan Ilmu Komputer dari Universitas Trisakti pada tahun 2001 dan Magister Ilmu Manajemen dari Universitas Sumatera Utara pada tahun 2004. Sedang menempuh program Doktor Ilmu Manajemen di Universitas Padjadjaran, Jawa Barat. Beliau menjabat sebagai Head of Business Administration di President University sejak tahun 2016 dan menjadi anggota Asosiasi Administrasi Bisnis Indonesia (AIABI) hingga saat ini. Ia menjadi Wakil Ketua Peradi Bersatu Advisory pada 2021 dan menjadi konsultan bisnis di Vijay Learning Center, lembaga pendidikan di Kota Tanjungbalai dan tutor bahasa Inggris di President Development Center. Bidang studi yang diminati adalah Literasi Digital, Statistika, Matematika Bisnis, Metodologi Riset, Bisnis Ritel, Manajemen Operasi, dan Kewirausahaan. Aktif sebagai reviewer artikel di beberapa Sinta dan jurnal terindeks internasional.

Dr. Drs. Waris Marsisno, M.Stat. adalah dosen Prodi Komputasi Statistik di Politeknik Statistika Jakarta. Bab yang ditulisnya dalam buku ini merupakan salah satu karya dan mudah-mudahan secara konsisten akan disusul dengan buku-buku berikutnya. Pokok bahasan buku yang ditulis semata-mata untuk berbagi ilmu pengetahuan.

Dr. Yuliana Ria Uli Sitanggang, SSi, MSi, buku ini adalah salah satu karya dan semoga akan segera disusul dengan buku-buku berikutnya. Pokok bahasan buku yang ditulis semata-mata untuk berbagi ilmu pengetahuan. Penulis merupakan dosen tidak tetap pada Prodi Komputasi di Politeknik Statistika STIS dan Widyaiswara di Pusdiklat Badan Pusat Statistik.

Khoerul Anwar, buku ini adalah salah satu karya dan inshaa allah secara konsisten akan disusul dengan buku-buku berikutnya. Pokok bahasan buku yang ditulis semata-mata untuk berbagi ilmu pengetahuan.

Ni Putu Linda Santiari, Saat ini penulis aktif mengajar di ITB STIKOM Bali prodi Sistem Informasi. Buku ini adalah salah satu karya dan semoga bisa secara konsisten akan disusul dengan buku-buku berikutnya. Pokok bahasan buku yang ditulis semata-mata untuk berbagi ilmu pengetahuan.

Sigit Setyowibowo, buku ini adalah salah satu karya dan inshaa allah secara konsisten akan disusul dengan buku-buku berikutnya. Pokok bahasan buku yang ditulis semata-mata untuk berbagi ilmu pengetahuan.

Toar Romario Sigar, ST., M.Kom Pendidikan penulis dimulai pada pendidikan strata 1 selesai di Universitas Katolik De la Salle Manado pada Fakultas Teknik Program Studi Teknik Informatika tahun 2012 Pendidikan strata 2 penulis selesai di Universitas Diponegoro pada Magister Sistem Informasi pada tahun 2016. Pengalaman praktisi, penulis pernah bekerja di PT Kawanua Dasa Pratama dan peran mengajar di SMK PPN Kalasey. Namun saat ini penulis memilih untuk fokus mengabdikan diri sebagai Dosen tetap dan aktif mengajar di Sekolah Tinggi Ilmu Ekonomi PETRA. Penulis memiliki kepakaran dibidang Sistem Informasi.

Ibnu Atho'illah, S.T., M.T. Lahir di Pasuruan, 20 Agustus 1975. Penulis memulai Pendidikan di SD Nahdlatul Ulama Pasuruan Jawa Timur tahun 1982. Alumni SMPN 2 Pasuruan tahun 1991, dan SMAN 1 Pasuruan tahun 1994. Setelah tamat SMA, penulis melanjutkan Pendidikan S1 di Institut Teknologi Nasional (ITN) Malang Jurusan Teknik Kimia lulus tahun 1999. Pada tahun yang sama melanjutkan studi S2 di Universitas Gadjah Mada (UGM) di jurusan yang sama. Setelah lulus pada tahun 2002 sempat bekerja di PT. Kutrindo Indonesia (2002 – 2006). Selepas tahun 2006 penulis banyak berkecimpung di dunia Pendidikan, Pengajaran dan Pelatihan. Pada tahun 2009 – sekarang, penulis merupakan dosen pengajar di STMIK Bandung Bali, juga di tahun 2011 – sekarang sebagai Instruktur IT di LPK Emerald Informatika Bali.
buku ini adalah salah satu karya dan Insya Allah secara konsisten akan disusul dengan buku-buku berikutnya. Pokok bahasan buku yang ditulis semata-mata untuk berbagi ilmu pengetahuan.

Dwipo Setyantoro, adalah seorang ayah dari 3 anak yang dilahirkan di Jakarta pada pertengahan Mei 1969. Penulis menyelesaikan studi S-1 di Ilmu Komputer Universitas Indonesia pada tahun 1995 dan melanjutkan studi S-2 di program studi Magister Manajemen Sistem Informasi di Universitas Gunadarma. Penulis mengawali karirnya sebagai dosen pada tahun 2008 dan mengajar di beberapa kampus ternama di Jakarta. Mata kuliah yang kerap diampu adalah Algoritma dan Pemrograman, Basis Data, Struktur Data, serta Analisa & Perancangan Sistem Informasi. Buku ini adalah salah satu karya penulis dan semoga secara konsisten akan disusul dengan buku-buku berikutnya. Pokok bahasan buku yang ditulis semata-mata untuk berbagi ilmu pengetahuan.

Ni Nyoman Emang Smrti, lahir di Banyuwangi, 21 Februari 1973. Pendidikan dasar di SDK Santa Maria Banyuwangi lulus Tahun 1985. Alumni 1988 SMPN 2 Banyuwangi dan Alumni 1991 SMAN 2 Banyuwangi. Tahun 1992 melanjutkan Pendidikan D3 STIKI Malang, kemudian melanjutkan jenjang S1 sampai dengan tahun 1999. Tahun 2010 melanjutkan ke jenjang S2 di Universitas Udayana. Pengalaman kerja dari tahun 1994 – 1997 Asisten Laboratorium di STIKI Malang. Tahun 2000 – 2004 Dosen di STIK PGRI Banyuwangi. Tahun 2005 – sekarang dosen dpk LLDIKTI wilayah VIII dan di Tempatkan di STMIK Bandung Bali.

Dalam dunia yang semakin terhubung dan tergantung pada teknologi, pemahaman tentang bagaimana data diorganisir dan dimanipulasi menjadi kunci untuk mengembangkan solusi perangkat lunak yang efisien dan efektif. "Pengantar Struktur Data" adalah buku yang mengajak pembaca untuk menjelajahi dunia konsep dasar struktur data dan pentingnya perannya dalam pengembangan perangkat lunak modern.

Buku ini memulai perjalanan dengan menguraikan konsep dasar struktur data seperti array dan linked list. Penjelasan yang jelas dan disertai contoh nyata membantu pembaca memahami cara kerja dan kegunaan masing-masing struktur. Dari situ, pembaca diajak untuk memahami bagaimana stack dan queue dapat digunakan untuk mengatasi berbagai tantangan dalam pemrograman.

Baik bagi mahasiswa yang ingin memahami dasar-dasar struktur data atau pengembang perangkat lunak yang ingin mengasah pemahaman mereka, "Pengantar Struktur Data" memberikan panduan yang komprehensif dan mudah diikuti. Buku ini bukan hanya sekadar menjelaskan teori, tetapi juga memberikan gambaran praktis tentang bagaimana struktur data memengaruhi cara kita membangun dan menggunakan teknologi di sekitar kita.

DITERBITKAN OLEH
PT. MIFANDI MANDIRI DIGITAL

Jln Payanibung Ujung D
Dalu Sepuluh-B, Tanjung Morawa
Kab. Deli Serdang Sumatera Utara

ISBN 978-623-88663-0-4