



Struktur Data

Pendidikan Teknik Informatika

Universitas Negeri Padang

TIM DOSEN ©2022

JOBSHEET (JS-08)

Heap

Fakultas	Proram Studi	Kode MK	Waktu
Teknik	Pendidikan Teknik Informatika	TIK1.61.2317	4 x 50 Menit

TUJUAN PRAKTIKUM

1. Mahasiswa mampu mengaplikasikan konsep heap dalam menyelesaikan kasus.

HARDWARE & SOFTWARE

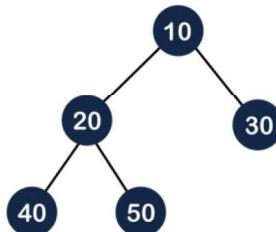
1. Personal Computer
2. IDE: Dev C++

TEORI SINGKAT

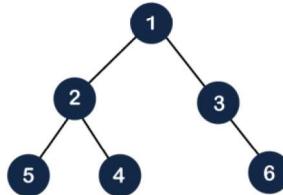
A. Binary Heap

Heap adalah pohon biner lengkap, dan pohon biner adalah pohon di mana node dapat memiliki paling banyak dua anak. Sebelum mengetahui lebih banyak tentang struktur data heap, kita harus tahu tentang pohon biner lengkap. Pohon biner lengkap adalah pohon biner di mana semua level kecuali level terakhir, yaitu, simpul daun harus terisi penuh, dan semua simpul harus rata kiri.

Mari kita pahami melalui sebuah contoh :



Pada gambar tersebut kita dapat mengamati bahwa semua simpul internal terisi penuh kecuali simpul daun; oleh karena itu, kita dapat mengatakan bahwa pohon di atas adalah pohon biner lengkap.



Gambar di atas menunjukkan bahwa semua simpul internal terisi penuh kecuali simpul daun, tetapi simpul daun ditambahkan di bagian kanan; oleh karena itu, pohon di atas bukanlah pohon biner lengkap.

Catatan: **Heap Tree adalah struktur data pohon biner seimbang khusus di mana simpul akar dibandingkan dengan anak-anaknya dan disusun sesuai dengan itu.**

Ada dua jenis heap:

1. Min Heap
2. Max Heap

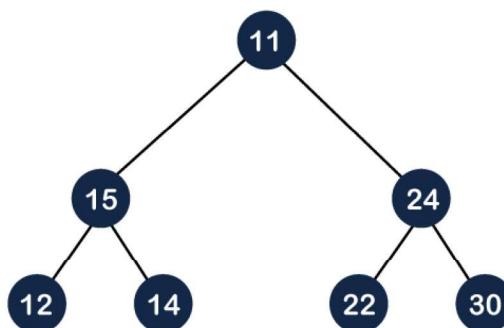
Min Heap: Nilai simpul induk harus kurang dari atau sama dengan salah satu anaknya.

Atau

Dengan kata lain, min-heap dapat didefinisikan sebagai, untuk setiap simpul i , nilai simpul i lebih besar atau sama dengan nilai induknya kecuali simpul akar. Secara matematis, itu dapat didefinisikan sebagai:

$$A[\text{Parent}(i)] \leq A[i]$$

Mari kita memahami min-heap melalui sebuah contoh:



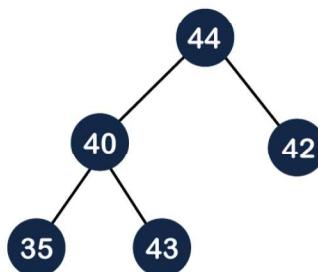
Pada gambar, 11 adalah simpul akar, dan nilai simpul akar lebih kecil dari nilai semua simpul lainnya (anak kiri atau anak kanan).

Max Heap: Nilai simpul induk lebih besar atau sama dengan anak-anaknya.

Atau

Dengan kata lain, max heap dapat didefinisikan sebagai untuk setiap node i ; nilai simpul i lebih kecil atau sama dengan nilai induknya kecuali simpul akar. Secara matematis, itu dapat didefinisikan sebagai:

$$A[\text{Parent}(i)] \geq A[i]$$



Pohon di atas adalah pohon tumpukan maksimal karena memenuhi properti tumpukan maksimal. Sekarang, mari kita lihat representasi array dari max heap.

B. Binomial Heap

Binomial Tree B_k adalah pohon terurut yang didefinisikan secara rekursif, di mana k didefinisikan sebagai urutan pohon binomial.

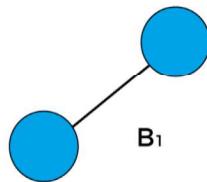
1. Jika pohon binomial direpresentasikan sebagai B_0 , maka pohon tersebut terdiri dari satu simpul.
2. Secara umum, B_k terdiri dari dua pohon binomial, yaitu B_{k-1} dan B_{k-1} yang dihubungkan satu sama lain dimana satu pohon menjadi subpohon kiri dari pohon binomial lainnya.

Kita dapat memahaminya dengan contoh yang diberikan di bawah ini :

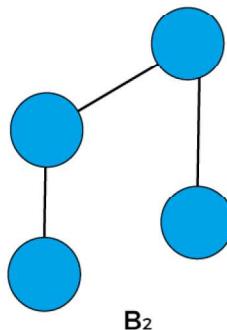
Jika B_0 , di mana k adalah 0, hanya akan ada satu simpul di pohon.



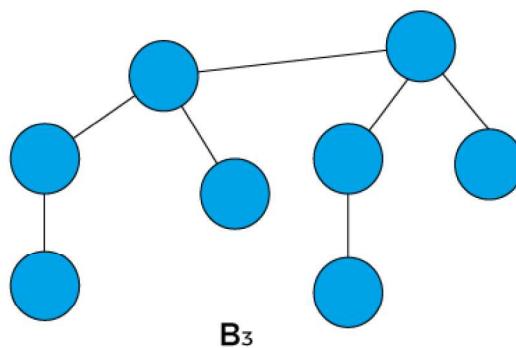
Jika B_1 , di mana k adalah 1. Oleh karena itu, akan ada dua pohon binomial B_0 di mana satu B_0 menjadi subpohon kiri dari B_0 lainnya .



Jika B_2 , di mana k adalah 2. Oleh karena itu, akan ada dua pohon binomial B_1 di mana satu B_1 menjadi subpohon kiri dari B_1 lainnya .



Jika B_3 , di mana k adalah 3. Oleh karena itu, akan ada dua pohon binomial B_2 di mana satu B_2 menjadi subpohon kiri dari B_2 lainnya .



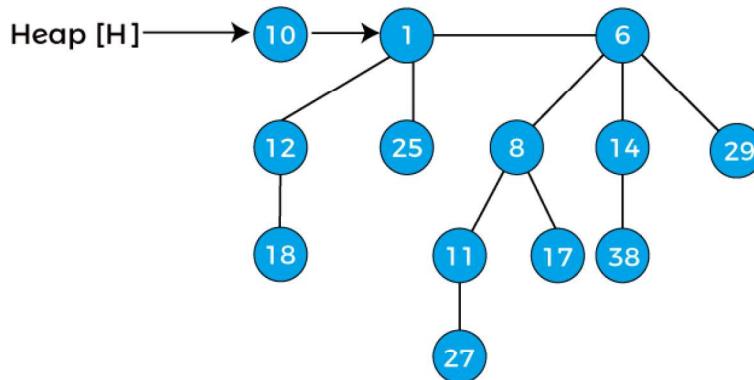
Binomial Heap dapat didefinisikan sebagai kumpulan pohon binomial yang memenuhi sifat heap, yaitu min-heap. Min-heap adalah heap di mana setiap node memiliki nilai lebih rendah dari nilai node anaknya. Terutama, tumpukan Binomial digunakan untuk mengimplementasikan antrian prioritas. Ini adalah perpanjangan dari tumpukan biner yang memberikan operasi penggabungan atau penyatuan yang lebih cepat bersama dengan operasi lain yang disediakan oleh tumpukan biner.

Ada properti berikut untuk tumpukan binomial dengan n node :

1. Setiap pohon binomial di heap harus mengikuti properti **min-heap**, yaitu, kunci dari sebuah simpul lebih besar atau sama dengan kunci induknya.
2. Untuk sembarang bilangan bulat non-negatif k , harus ada setidaknya satu pohon binomial di tumpukan di mana akarnya memiliki derajat k .

Properti pertama dari heap memastikan bahwa properti min-heap dipegang di seluruh heap. Sedangkan properti kedua yang tercantum di atas memastikan bahwa pohon biner dengan n node harus memiliki paling banyak $1 + \log_2 n$ pohon binomial, di sini \log_2 adalah logaritma biner.

Kita dapat memahami properti yang tercantum di atas dengan bantuan contoh :



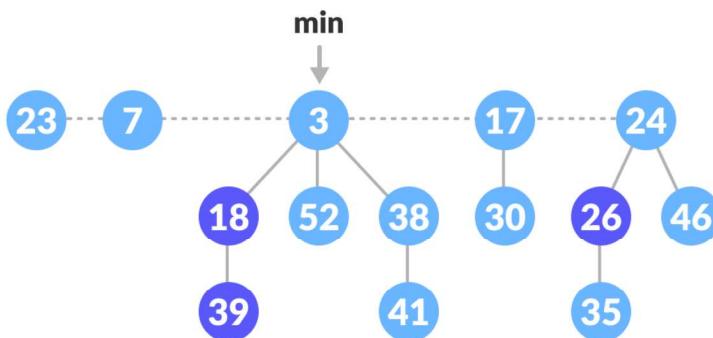
Gambar di atas memiliki tiga pohon binomial, yaitu B_0 , B_2 , dan B_3 . Ketiga pohon binomial di atas memenuhi properti min heap karena semua node memiliki nilai yang lebih kecil daripada node anak. Gambar di atas juga memenuhi sifat kedua dari tumpukan binomial. Misalnya, jika kita menganggap nilai k sebagai 3, kita dapat mengamati pada gambar di atas bahwa pohon binomial derajat 3 ada di tumpukan.

C. Fibonacci Heap

Fibonacci heap adalah struktur data yang terdiri dari kumpulan pohon yang mengikuti properti min heap atau max heap. Kedua sifat ini adalah karakteristik pohon yang ada di fibonacci heap.

Dalam fibonacci heap, sebuah simpul dapat memiliki lebih dari dua anak atau tidak memiliki anak sama sekali. Juga, ia memiliki operasi tumpukan yang lebih efisien daripada yang didukung oleh tumpukan binomial dan biner.

Fibonacci heap disebut **fibonacci heap** karena pohon dibangun sedemikian rupa sehingga pohon urutan n memiliki setidaknya node di dalamnya, di mana adalah angka Fibonacci $F_{n+2}F_{n+2}(n + 2)^{\text{th}}$

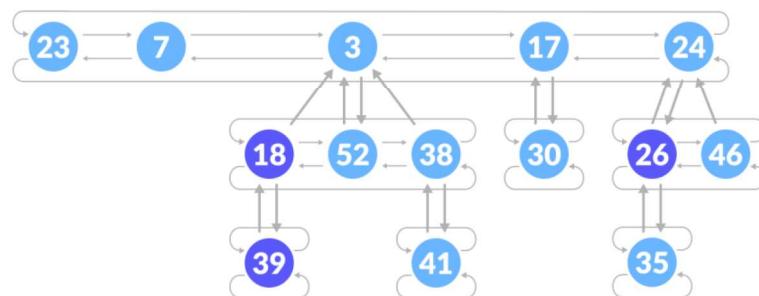


Gambar. Fibonacci Heap

Sifat-sifat penting dari Fibonacci heap adalah:

1. Ini adalah satu set **minimum tumpukan** pohon yang **dipesan**. (yaitu Orang tua selalu lebih kecil dari anak-anak.)
2. Pointer dipertahankan pada node elemen minimum.
3. Ini terdiri dari satu set node yang ditandai. (Turunkan operasi kunci)
4. Pohon-pohon dalam tumpukan Fibonacci tidak teratur tetapi berakar.

Akar dari semua pohon dihubungkan bersama untuk akses yang lebih cepat. Node anak dari node induk terhubung satu sama lain melalui circular doubly linked list seperti yang ditunjukkan di bawah ini :



Gambar. Struktur Fibonacci Heap

Ada dua keuntungan utama menggunakan circular doubly linked list :

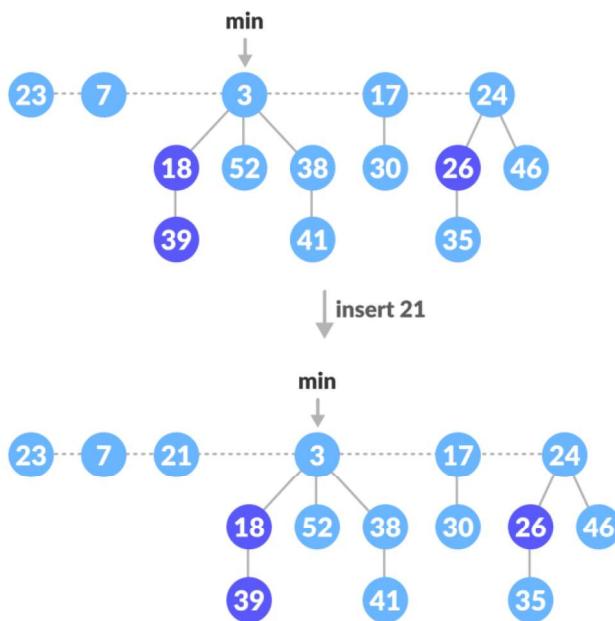
1. Menghapus sebuah node dari pohon membutuhkan $O(1)$ waktu.
2. Penggabungan dua daftar tersebut membutuhkan $O(1)$ waktu.

Operasi pada Tumpukan Fibonacci

Insertion

Memasukkan node ke dalam heap yang sudah ada ada mengikuti langkah-langkah di bawah ini :

1. Buat simpul baru untuk elemen tersebut.
2. Periksa apakah tumpukan kosong.
3. Jika heap kosong, atur simpul baru sebagai simpul akar dan tandai **min**.
4. Jika tidak, masukkan simpul ke dalam daftar root dan perbarui **min**.



Gambar. Contoh Penyisipan

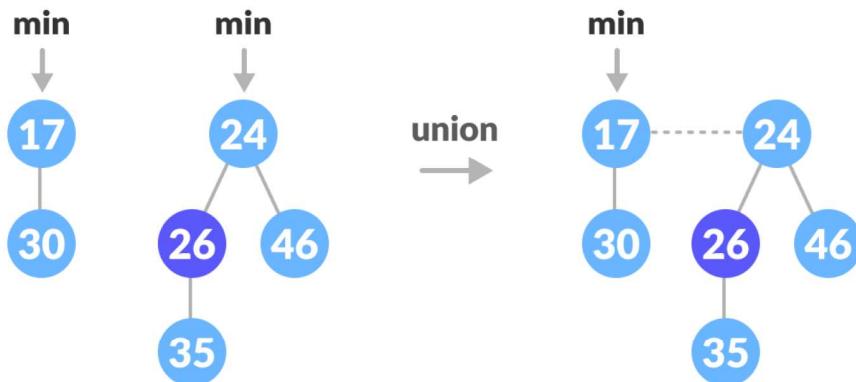
Find Min

Elemen minimum selalu diberikan oleh **min** penunjuk.

Union

Union dari dua fibonacci heap terdiri dari langkah-langkah berikut:

1. Gabungkan akar dari kedua tumpukan.
2. Memperbarui **min** dengan memilih kunci minimum dari daftar akar baru.



Gambar. Union dari 2 heap

Extract Min

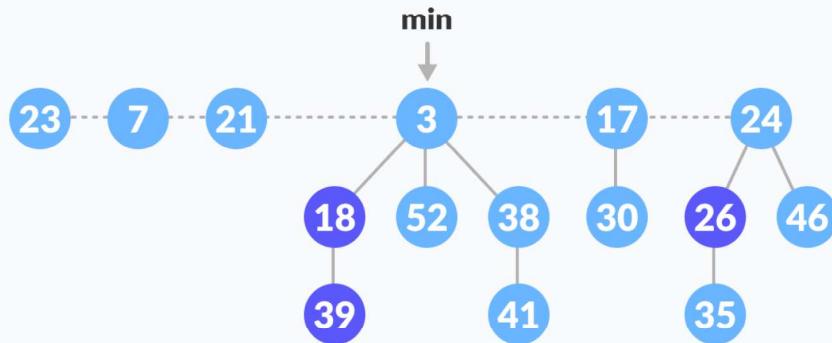
Ini adalah operasi yang paling penting pada fibonacci heap. Dalam operasi ini, node dengan nilai minimum dihapus dari heap dan pohon disesuaikan kembali.

Langkah-langkahnya sebagai berikut:

1. Hapus simpul min.
2. Atur penunjuk min ke root berikutnya dalam daftar root.
3. Buat array dengan ukuran yang sama dengan tingkat maksimum pohon di heap sebelum dihapus.
4. Lakukan langkah berikut (langkah 5-7) sampai tidak ada akar kelipatan dengan derajat yang sama.
5. Petakan derajat akar saat ini (penunjuk-min) ke derajat dalam larik.
6. Petakan derajat akar berikutnya ke derajat dalam array.
7. Jika ada lebih dari dua pemetaan untuk derajat yang sama, maka terapkan operasi gabungan ke akar-akar tersebut sehingga properti min-heap dipertahankan (yaitu minimum ada di akar).

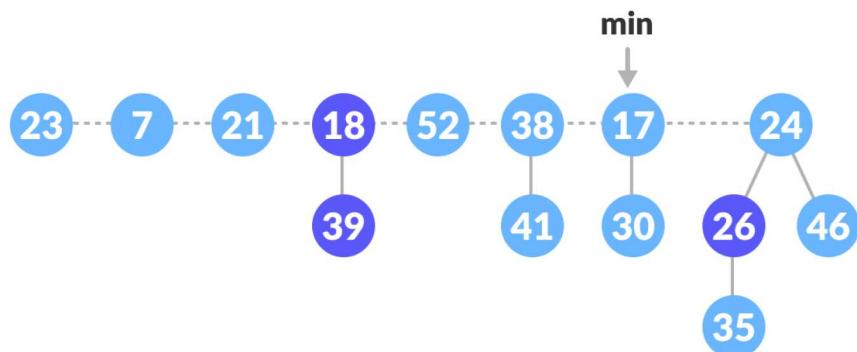
Implementasi dari langkah-langkah di atas dapat dipahami pada contoh di bawah ini :

1. Akan melakukan operasi ekstrak-min pada heap di bawah ini:



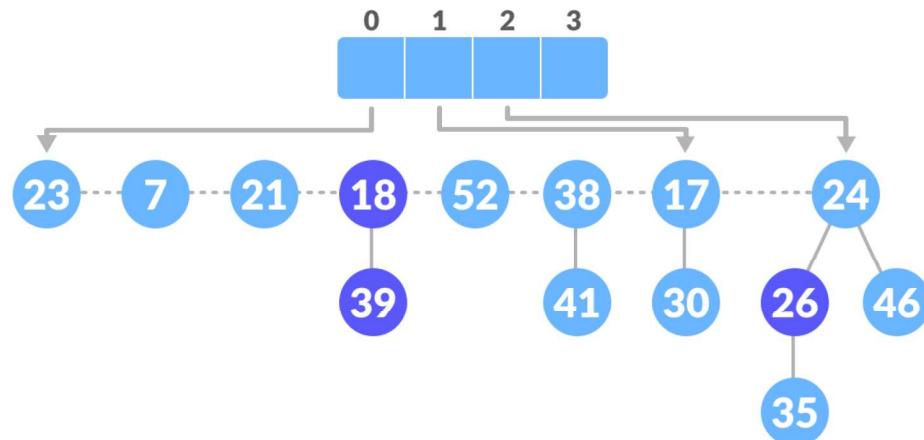
Gambar. Tumpukan Fibonacci

2. Hapus simpul min, tambahkan semua simpul turunannya ke daftar akar dan atur penunjuk-min ke akar berikutnya dalam daftar akar.

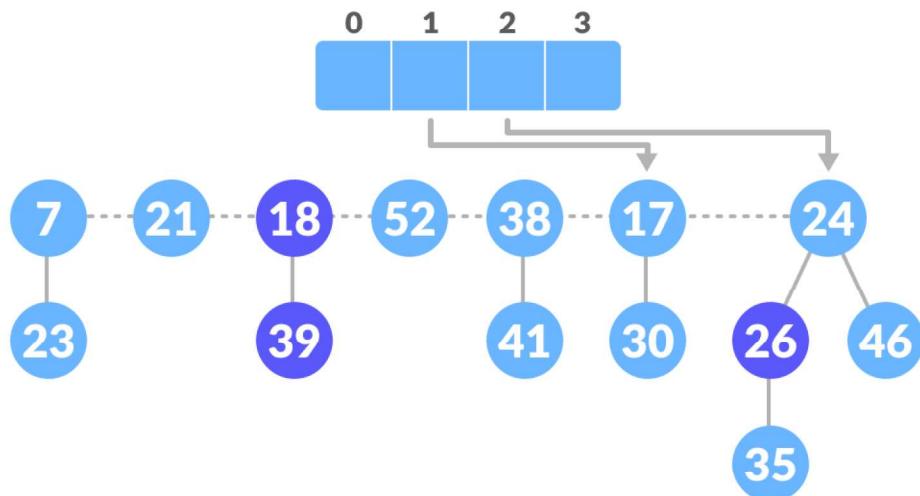


Gambar. Hapus simpul min

3. Derajat maksimum dalam pohon adalah 3. Buat larik berukuran 4 dan petakan derajat akar berikutnya dengan larik tersebut.

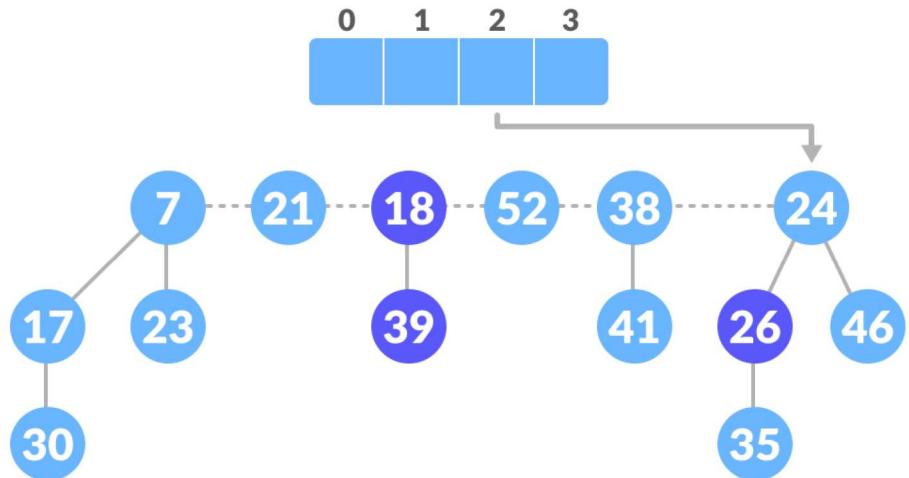


4. Buat larik, di sini, 23 dan 7 memiliki derajat yang sama, jadi satukan.



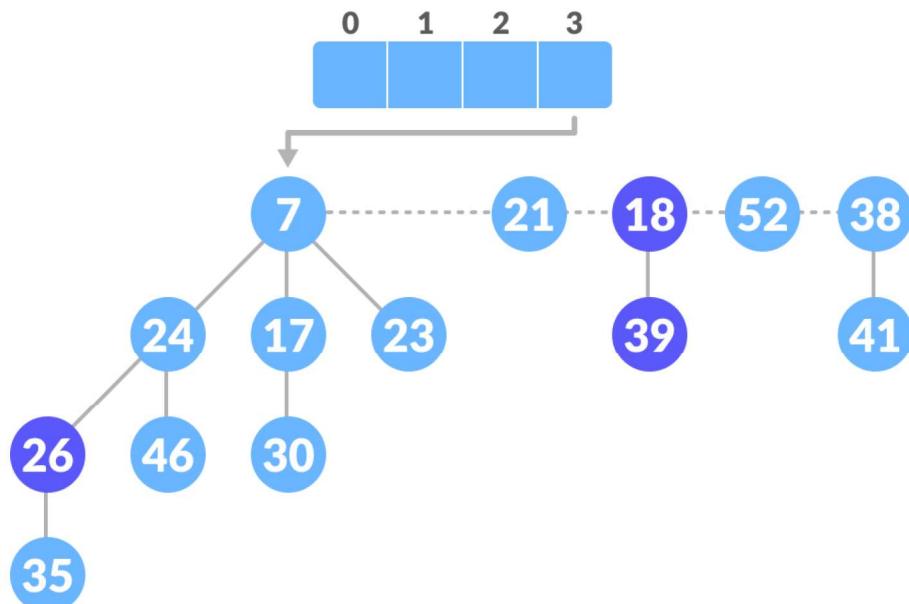
Satukan mereka yang memiliki derajat yang sama.

5. Sekali lagi, 7 dan 17 memiliki derajat yang sama, jadi satukan juga.



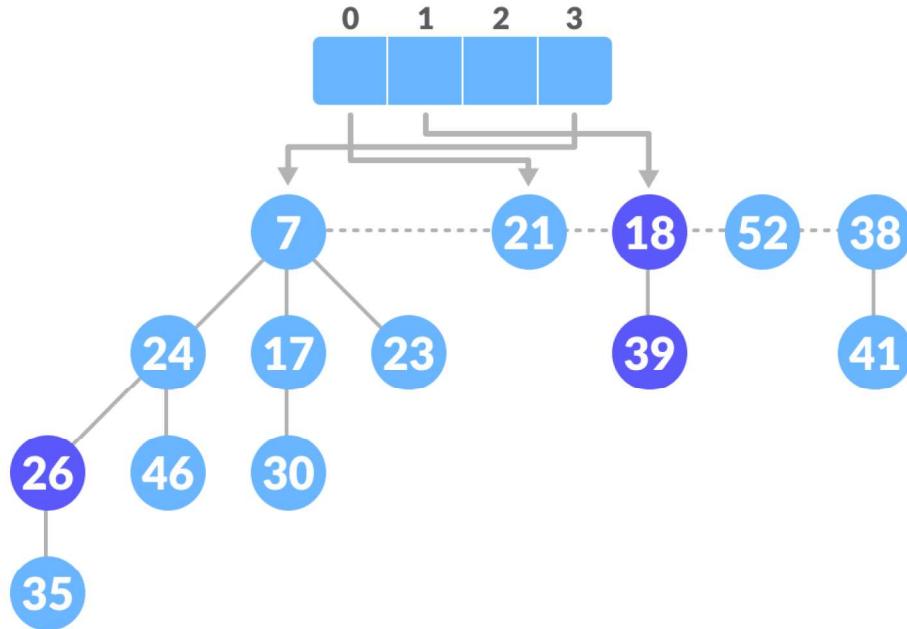
Satukan mereka yang memiliki derajat yang sama.

6. Sekali lagi 7 dan 24 memiliki derajat yang sama, jadi satukan mereka.



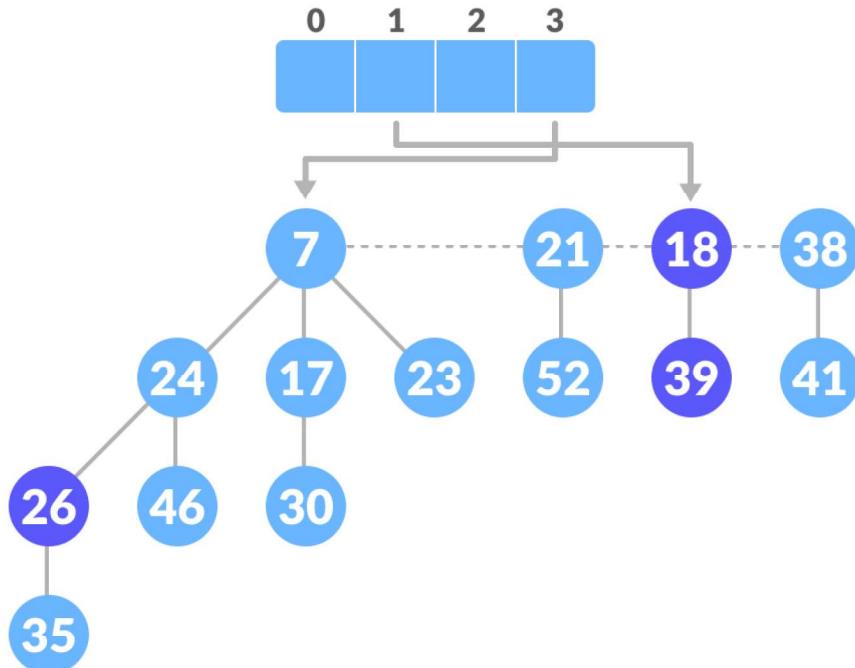
Satukan mereka yang memiliki derajat yang sama.

7. Petakan node berikutnya.

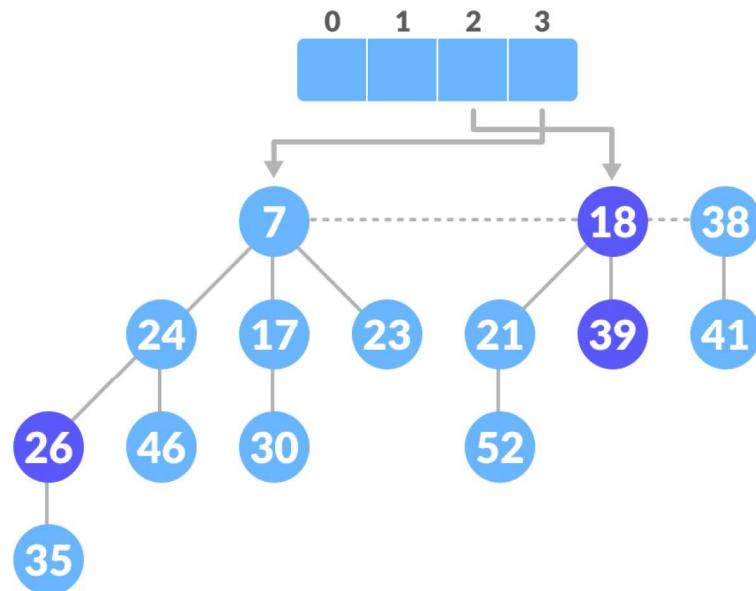


Petakan node yang tersisa.

8. Sekali lagi, 52 dan 21 memiliki derajat yang sama, jadi satukan mereka.

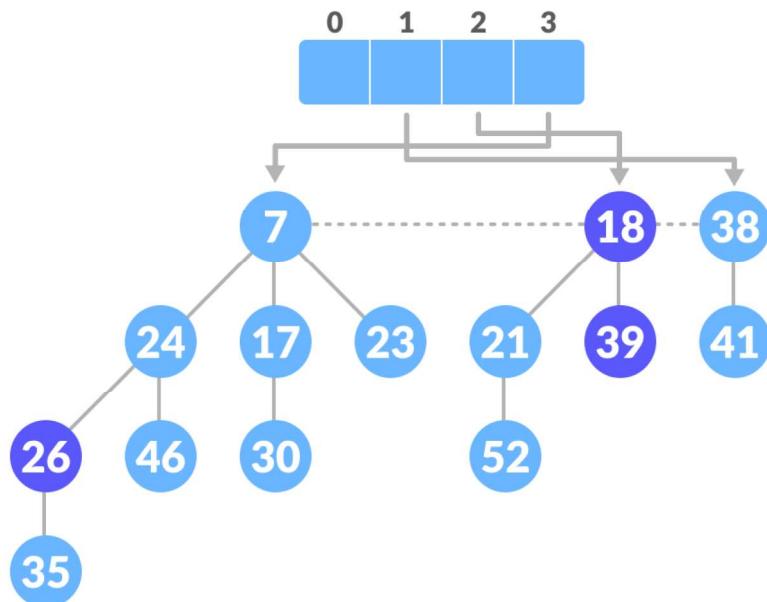


9. Demikian pula, satukan 21 dan 18.



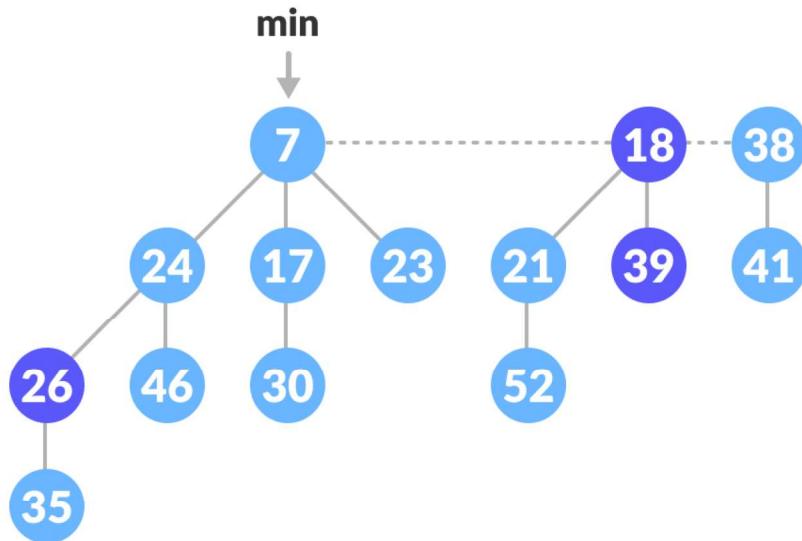
Satukan mereka yang memiliki derajat yang sama.

10. Petakan akar yang tersisa.



Petakan node yang tersisa.

11. Tumpukan terakhir adalah :



Tumpukan fibonacci terakhir.

LATIHAN

1. Max Heap

```
/* Nama File : max-heap
Pembuat : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/

// Max-Heap data structure in C

#include <stdio.h>
int size = 0;
void swap(int *a, int *b)
{
    int temp = *b;
    *b = *a;
    *a = temp;
}
void heapify(int array[], int size, int i)
```

```

if (size == 1)
{
    printf("Single element in the heap");
}
else
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < size && array[l] > array[largest])
        largest = l;
    if (r < size && array[r] > array[largest])
        largest = r;
    if (largest != i)
    {
        swap(&array[i], &array[largest]);
        heapify(array, size, largest);
    }
}
void insert(int array[], int newNum)
{
    if (size == 0)
    {
        array[0] = newNum;
        size += 1;
    }
    else
    {
        array[size] = newNum;
        size += 1;
        for (int i = size / 2 - 1; i >= 0; i--)
        {
            heapify(array, size, i);
        }
    }
}

```

```
}

void deleteRoot(int array[], int num)
{
    int i;
    for (i = 0; i < size; i++)
    {
        if (num == array[i])
            break;
    }

    swap(&array[i], &array[size - 1]);
    size -= 1;
    for (int i = size / 2 - 1; i >= 0; i--)
    {
        heapify(array, size, i);
    }
}

void printArray(int array[], int size)
{
    for (int i = 0; i < size; ++i)
        printf("%d ", array[i]);
    printf("\n");
}

int main()
{
    int array[10];

    insert(array, 3);
    insert(array, 4);
    insert(array, 9);
    insert(array, 5);
    insert(array, 2);

    printf("Max-Heap array: ");
    printArray(array, size);
}
```

```

    deleteRoot(array, 4);

    printf("After deleting an element: ");

    printArray(array, size);
}

```

TUGAS

Pahami dan modifikasilah program pada Latihan di atas sehingga menghasilkan sebuah program baru yang mempunyai output seperti tampilan di bawah ini :

```

D:\Bahan Ajar\Struktur Data\Paktikum Struktur Data\Latihan\Job 8\max_heap_tugas.exe
Max-Heap array: 17 15 16 10 14 12 13
After deleting an element: 17 14 16 13 12

-----
Process exited after 0.02457 seconds with return value 0
Press any key to continue . .

```

DAFTAR PUSTAKA

- Kelley, Al and Pohl, Ira. 2003. *C by Dissection: The Essentials of C Programming*. Addison-Weasley.
- Khannedy, Eko Kurniawan. 2007. *Diktat Pemrograman C*. Bandung : Unikom.
- Liem, Ingriani. 2003. *Catatan Singkat Bahasa C*. Jurusan Teknik Informatika. Institut Teknologi Bandung.
- Reema Thareja. 2014. *Data Structures Using C*. India : Oxford University Press.
- Solichin, Achmad. 2003. *Pemrograman Bahasa C dengan Turbo C*. Kuliah Berseri ilmukomputer.com