Kebenaran Algoritma

Teosofi Hidayah Agung
Hafidz Mulia

Departemen Matematika
Institut Teknologi Sepuluh Nopember

15 Maret 2025

Masalah

Saat pertama kali melihat suatu kode program pastilah kita tidak akan langsung paham dengan apa yang dilakukan oleh kode tersebut. Sehingga diperlukan adanya **analisis** terhadap program apa yang dijalankan nantinya.
Analisis dapat berupa apa saja, mulai dari analisa alur program, output yang dihasilkan, kompleksitas waktu dan ruang, dsb. Dalam materi kali ini kita cukup berfokus pada menganalisa kebenaran alur dan *output* dari sebuah program.

Daftar isi

1. Pengertian *Correctness of Algorithm*
2. Tipe-tipe Kebenaran
   - Partial Correctness
   - Total Correctness
3. Teknik Pembuktian
   - Induksi Matematika
   - *Loop Invariant*
   - Metode Formal
4. Latihan

Pengertian *Correctness of Algorithm*

Kebenaran Algoritma

Correctness of Algorithm mengacu pada kepastian bahwa suatu algoritma memberikan hasil yang benar sesuai dengan spesifikasi masalah yang diberikan. Sebuah algoritma dikatakan benar (*correct*), jika untuk setiap masukan yang valid, algoritma tersebut menghasilkan keluaran yang diharapkan dalam waktu yang wajar.

Analisa kebenaran algoritma cukup penting terlepas dari benar atau salah. Disisi lain analisa tersebut dapat membantu untuk menemukan sebuah solusi optimal dari suatu permasalahan.

Pengertian *Correctness of Algorithm*

Suatu Algoritma haruslah **efektif**, artinya:

- Algoritma harus benar dalam menyelesaikan masalah atau konteks yang diberikan.
- Tujuan dari pembuatannya dapat dicapai.
- Konsisten dalam memberikan hasil yang sama untuk masukan yang sama.

Menurut kalian apakah program berikut sudah benar sesuai intruksi yang diberikan?

(86-100) : Nilai A
(76-85) : Nilai AB
(66-75) : Nilai B /
(61-65) : Nilai BC
(56-60) : Nilai C /
(41-55) : Nilai D /
(0-40) : Nilai E / k

```
if (nilai >= 86 && nilai <= 100)
  grade = "A";
else if (nilai >= 76 && nilai <= 85)
  grade = "AB";
else if (nilai >= 66 && nilai <= 75)
  grade = "B";
else if (nilai >= 61 && nilai <= 65)
  grade = "BC";
else if (nilai >= 56 && nilai <= 60)
  grade = "C";
else if (nilai >= 41 && nilai <= 55)
  grade = "D";
else
  grade = "E";
```

Daftar isi

1. Pengertian *Correctness of Algorithm*
2. Tipe-tipe Kebenaran
   - Partial Correctness
   - Total Correctness
3. Teknik Pembuktian
   - Induksi Matematika
   - *Loop Invariant*
   - Metode Formal
4. Latihan

Tipe-tipe Kebenaran

Correctness dalam algoritma biasanya dikategorikan menjadi dua bagian:

1. **Partial Correctness**
   Berarti bahwa jika algoritma berhenti, maka hasil yang dihasilkan adalah benar. Namun, tidak menjamin bahwa algoritma akan selalu berhenti (terminate) untuk semua masukan yang valid.
2. **Total Correctness**
   Sudah dipastikan bahwa algoritma tidak hanya menghasilkan hasil yang benar, tetapi juga selalu menyelesaikan eksekusinya dalam waktu yang wajar.

Kata ***partial*** berarti sebagian, artinya bisa jadi terdapat suatu masukan yang dimana menghasilkan *output* yang tidak benar.

Kode: Nilai Maksimum Array (*partial*)

```java
int[] numbers = {13, 4, 24, 7};
int maxNum = -1;
for (int i = 0; i < numbers.length; i++) {
    if (numbers[i] > maxNum) maxNum = numbers[i];
}
System.out.println(maxNum);
```

Kira-kira apa yang salah dari program di atas?

Tipe-tipe Kebenaran

Total Correctness

Kata ***total*** berarti keseluruhan, artinya algoritma tersebut akan selalu menghasilkan *output* yang benar jika inputnya juga valid.

Kode: Nilai Maksimum Array (*total*)

```
int[] numbers = {13, 4, 24, 7};
int maxNum = numbers[0];
for (int i = 0; i < numbers.length; i++) {
    if (numbers[i] > maxNum) maxNum = numbers[i];
}
System.out.println(maxNum);
```

Bagaimana membuktikan bahwa program tersebut benar secara keseluruhan?

Tipe-tipe Kebenaran

Contoh

Berikut ini manakah yang termasuk dalam kategori *Partial Correctness* dan *Total Correctness*?

```
int factorial(int n) {
  int hasil=n;
  while(n>1){
    hasil *= n-1;
    n--;
  }
  return hasil;
}
```

Kode: Faktorial

```
static int fibo(int n) {
  double A = (1 + Math.sqrt(5)) / 2;
  double B = (1 - Math.sqrt(5)) / 2;
  return (int)((Math.pow(A,n) -
      Math.pow(B,n)) / Math.sqrt(5));
}
```

Kode: Fibonacci

Daftar isi

1. Pengertian *Correctness of Algorithm*
2. Tipe-tipe Kebenaran
   - Partial Correctness
   - Total Correctness
3. Teknik Pembuktian
   - Induksi Matematika
   - *Loop Invariant*
   - Metode Formal
4. Latihan

Pembuktian atau Penyangkalan

Dalam membuktikan sesuatu yang umum, maka argumen-argumen kita harus diperumum juga. Janganlah sesekali membuktikan sebuah kebenaran di matematika menggunakan beberapa contoh kasus.
Sebaliknya untuk menyangkal sesuatu yang umum, pakailah contoh penyangkal (***Counter Example***) jika memang betul bahwa pernyataan tersebut salah.

Contoh

Buktikan atau sangkal pernyataan berikut:

- "$\forall x, y \in \mathbb{R}, |x+y| = |x| + |y|$"
- "$\forall x \forall y (xy \geq x)$"

Prinsip Induksi Matematika

Jika kita ingin membuktikan suatu pernyataan $P(n)$ benar untuk setiap bilangan bulat $n \geq n_0$, maka kita harus membuktikan dua hal:

1. **Basis Induksi** : $P(n_0)$ benar.
2. **Hipotesis Induksi** : Asumsikan $P(k)$ benar untuk suatu $k \geq n_0$.
3. **Langkah Induksi** : Tunjukan bahwa $P(k+1)$ juga benar.

Dengan mengansumsikan k adalah sebarang bilangan bulat dan jika untuk bilangan setelahnya juga benar, maka dapat disimpulkan bahwa pernyataan tersebut benar untuk setiap bilangan bulat.

Contoh

Buktikan bahwa program dibawah ini menghitung nilai $\dfrac{n(n+1)(2n+1)}{6}$

```
int sumOfSquares(int n) {
  int sum = 0;
  for (int i = 1; i <= n; i++) {
    sum += i * i;
  }
  return sum;
}
```

Bukti

- **Basis Induksi** :
- **Hipotesis Induksi** :
- **Langkah Induksi** :

Bukti

- **Basis Induksi** : $P(1)$ benar, karena $1^2 = \dfrac{1(1+1)(2\cdot 1+1)}{6}$
- **Hipotesis Induksi** :
- **Langkah Induksi** :

Bukti

- **Basis Induksi** : $P(1)$ benar, karena $1^2 = \frac{1(1+1)(2\cdot 1+1)}{6}$
- **Hipotesis Induksi** : Asumsikan $P(k) := 1^2 + 2^2 + 3^2 + \cdots + k^2 = \frac{k(k+1)(2k+1)}{6}$ benar untuk suatu $k \geq 1$.
- **Langkah Induksi** :

Bukti

- **Basis Induksi** : $P(1)$ benar, karena $1^2 = \dfrac{1(1+1)(2\cdot 1+1)}{6}$
- **Hipotesis Induksi** : Asumsikan $P(k) := 1^2 + 2^2 + 3^2 + \cdots + k^2 = \dfrac{k(k+1)(2k+1)}{6}$ benar untuk suatu $k \geq 1$.
- **Langkah Induksi** : Akan ditunjukan bahwa $P(k+1)$ juga benar.

$$
\begin{aligned}
P(k+1) :&= 1^2 + 2^2 + \cdots + k^2 + (k+1)^2 = \frac{k(k+1)(2k+1)}{6} + (k+1)^2 \\
&= \frac{k(k+1)(2k+1) + 6(k+1)^2}{6} = \frac{(k+1)(k(2k+1) + 6(k+1))}{6} \\
&= \frac{(k+1)(2k^2+7k+6)}{6} = \frac{(k+1)(k+2)(2k+3)}{6} \qquad \square
\end{aligned}
$$

Loop Invariant

Loop Invariant adalah suatu kondisi yang selalu benar pada setiap iterasi dari suatu *loop*. *Loop Invariant* biasanya digunakan untuk membuktikan kebenaran dari suatu program yang menggunakan *loop*. Terdapat tiga tahap pembuktian:

1. **Inisialisasi** : *Loop Invariant* benar sebelum iterasi pertama.
2. **Maintenance** : Jika *Loop Invariant* benar sebelum iterasi ke-k, maka *Loop Invariant* juga benar pada iterasi ke-$(k+1)$.
3. **Terminasi** : Ketika *Loop Invariant* berhenti, maka haruslah memberikan hasil yang benar sesuai spesifikasi masalah.

Contoh

Buktikan bahwa algoritma berikut menentukan keberadaan (index) dari nilai `target` dalam sebuah array

```
int linearSearch(int[] arr, int target) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

Kode: Linear Search

Bukti

Definisikan *Loop Invariant* sebagai berikut:

> "*Pada iterasi ke-i, semua elemen dalam array dari indeks 0 sampai $i-1$ sudah diperiksa, dan jika $arr[i]==target$ maka nilai telah ditemukan di indeks-i. Selain hal itu, target tidak ada dalam subaaray $arr[0:i-1]$.*"

1. **Inisialisasi** :

2. **Maintenance** :

Bukti

Definisikan *Loop Invariant* sebagai berikut:

> "*Pada iterasi ke-i, semua elemen dalam array dari indeks 0 sampai i-1 sudah diperiksa, dan jika $arr[i]$==$target$ maka nilai telah ditemukan di indeks-i. Selain hal itu, target tidak ada dalam subaaray $arr[0:i$-$1]$.*"

1. **Inisialisasi** : Sebelum loop dijalankan (`i = 0`), belum ada elemen yang diperiksa. Pernyataan *loop invariant* tetap benar karena tidak ada elemen yang diperiksa sebelum iterasi pertama.
2. **Maintenance** :

Bukti

Definisikan *Loop Invariant* sebagai berikut:

> "*Pada iterasi ke-i, semua elemen dalam array dari indeks 0 sampai $i-1$ sudah diperiksa, dan jika $arr[i]==target$ maka nilai telah ditemukan di indeks-i. Selain hal itu, target tidak ada dalam subaaray $arr[0:i-1]$.*"

1. **Inisialisasi** : Sebelum loop dijalankan (`i = 0`), belum ada elemen yang diperiksa. Pernyataan *loop invariant* tetap benar karena tidak ada elemen yang diperiksa sebelum iterasi pertama.
2. **Maintenance** : Pada iterasi ke-`i`, kita mengecek elemen `arr[i]`:
   - Jika `arr[i] == target`, maka kita mengembalikan `i`, yang berarti target ditemukan.
   - Jika tidak, kita lanjut ke elemen berikutnya.

   Loop invariant tetap benar karena elemen sebelum indeks `i+1` telah diperiksa.

Bukti (lanj.)

3. **Terminasi** :

Bukti (lanj.)

3. **Terminasi** : Ketika loop berhenti, ada dua kemungkinan:
   - Jika `target` ditemukan, maka kita mengembalikan indeksnya.
   - Jika tidak, maka kita mengembalikan -1, yang berarti `target` tidak ada dalam array.

   Kedua kasus tersebut sesuai dengan spesifikasi masalah.

Bukti (lanj.)

3. **Terminasi** : Ketika loop berhenti, ada dua kemungkinan:
   - Jika `target` ditemukan, maka kita mengembalikan indeksnya.
   - Jika tidak, maka kita mengembalikan -1, yang berarti `target` tidak ada dalam array.

   Kedua kasus tersebut sesuai dengan spesifikasi masalah.

Jadi algoritma tersebut benar dalam menentukan keberadaan/indeks `target` dalam array.

Metode Formal

Metode formal adalah teknik pembuktian kebenaran algoritma yang menggunakan logika matematika. Metode ini biasanya digunakan untuk membuktikan kebenaran algoritma yang kompleks. Ada dua kunci dalam pembuktian ini, antara lain:

1. **Preconditions**: Keadaan yang harus benar sebelum algoritma dijalankan.
2. **Postconditions**: Keadaan yang harus benar setelah algoritma dijalankan.

Contoh

Buktikan bahwa program dibawah ini menghitung nilai x^n dengan n bulat non-negatif.

```
public static double power(double x, int n) {
  if (n == 0) {
      return 1;
  }
  double half = power(x, n / 2);
  if (n % 2 == 0) {
      return half * half;
  } else {
      return x * half * half;
  }
}
```

Bukti

1. **Preconditions** :

2. **Postconditions** :

Bukti

1. **Preconditions** :
   - x bisa berupa bilangan real (termasuk negatif dan pecahan).
   - n adalah bilangan bulat non-negatif.

   Jika n negatif, maka algoritma tidak valid karena tidak menangani eksponen negatif.
2. **Postconditions** :

Bukti

1. **Preconditions** :
   - x bisa berupa bilangan real (termasuk negatif dan pecahan).
   - n adalah bilangan bulat non-negatif.

   Jika n negatif, maka algoritma tidak valid karena tidak menangani eksponen negatif.
2. **Postconditions** : power$(x, n) = x^n$
   - Basis $n = 0$ berakibat $x^0 = 1$.
   - Jika n genap, maka $x^n = (x^{n/2})^2$.
   - Jika n ganjil, maka $x^n = x \cdot (x^{n/2})^2$.

Bukti

1. **Preconditions** :
   - x bisa berupa bilangan real (termasuk negatif dan pecahan).
   - n adalah bilangan bulat non-negatif.

   Jika n negatif, maka algoritma tidak valid karena tidak menangani eksponen negatif.
2. **Postconditions** : power$(x, n) = x^n$
   - Basis $n = 0$ berakibat $x^0 = 1$.
   - Jika n genap, maka $x^n = (x^{n/2})^2$.
   - Jika n ganjil, maka $x^n = x \cdot (x^{n/2})^2$.

Dengan menggunakan metode formal, kita dapat membuktikan bahwa program tersebut benar dalam menghitung nilai x^n dengan $n \in \mathbb{N} \cup 0$.

Tabel: Perbandingan Metode Pembuktian Kebenaran Algoritma

Metode	Fokus	Cakupan	Contoh Algoritma
Induksi Matematika	Untuk membuktikan kebenaran rumus matematika atau algoritma rekursif	Algoritma berbasis rekursi atau perulangan yang memiliki pola matematis	Faktorial, Fibonacci, jumlah deret, Hanoi Tower
Loop Invariant	Untuk membuktikan algoritma berbasis perulangan agar tetap mempertahankan sifat tertentu selama eksekusi	Algoritma yang menggunakan loop (iteratif)	*sorting* dan *searching*
Pembuktian Formal	Untuk membuktikan keabsahan algoritma secara umum tanpa harus bergantung pada perulangan atau rekursi.	Memberikan bukti formal untuk keseluruhan algoritma, termasuk rekursi, loop, dan kondisi lainnya.	Euclidean GCD, Exponentiation by Squaring, Divide and Conquer, Dynamic Programming

Daftar isi

1. Pengertian *Correctness of Algorithm*
2. Tipe-tipe Kebenaran
   - Partial Correctness
   - Total Correctness
3. Teknik Pembuktian
   - Induksi Matematika
   - *Loop Invariant*
   - Metode Formal
4. Latihan

Latihan 1

Buktikan bahwa fungsi berikut menghitung jumlahan digit suatu bilangan bulat

```
int jumlahDigit(int X) {
  if (X == 0) return 0;
  else return (X % 10) + jumlahDigit(X / 10);
}
```

Latihan 2

Telusuri fungsi berikut dan buktikan kebenaran dari fungsi tersebut.

```
boolean cek(string S) {
  for (int i = 0; i < n-1; i++) {
    if (S[i] > S[i+1]) return false;
    }
  return true;
}
```