Fakultas Informatika
School of Computing
Telkom University
Think Big! Go Forward!

Telkom University

ANALISIS ALGORITMA

Week 01: Pengantar Analisis Algoritma

PROGRAM PASCA SARJANA INFORMATIKA
FAKULTAS TEKNIK INFORMATIKA
UNIVERSITAS TELKOM

2022/2023

Apakah Algoritma?
Mengapa perlu dianalisis?
Apa yang perlu dianalisis?

Algoritma

- **Urutan/komposisi instruksi** komputasi untuk menyelesaikan **suatu masalah**
- Eksekusinya **selesai** dalam waktu/jumlah langkah terbatas
- Diberikan **kondisi awal** yang jelas, komputasi akan **memberikan jawaban** yang jelas pula saat terminasi (no GIGO)

- Berbasis model Serial Random Access Machine (RAM):
  - Simplifikasi cara kerja komputer modern
  - Operasi primitif: aritmatika, logika, (ALU) dan lainnya
  - Instruksi Input-Output
  - Instruksi Assigment instructions (perpindahan data dalam memori)
  - Struktur kontrol: percabangan (if-else) dan pengulangan (while-loop)

Kasus 1:
Pencarian Nilai Maksimum

Mencari Nilai Maksimum dalam Array

Diberikan sebanyak **n** data dalam array
Cari nilai terbesar/terkecil (ekstrim) dalam array tersebut

- **Caranya?**

  Operasi primitif yang tersedia adalah membandingkan sepasang data

- **Ada cara yang lebih cerdik?**

Mencari Nilai Maksimum dalam Array

Strategi Brute-force

- Nyata sejak awal bahwa **seluruh data harus diperiksa**
- Jika tidak maka data yang terlewat diperiksa mungkin adalah yang terbesar

Dengan operasi primitif membandingkan 2 data

- Bandingkan setiap data satu persatu dan
- Simpan/catat data yang sejauh ini diketahui sebagai yang terbesar

Tapi:

- Jika urutan data diketahui dimana **nilainya dapat diduga dari posisi**nya
- Misalnya terurut membesar, maka data terakhir pastilah yang terbesar.

Mencari Nilai Maksimum dalam Array

```
func FindMax( A[1..n] ) : index
    imax := 1
    i := 2
    while i ≤ n do
        if A[i] > A[imax] then
            imax = i
        endif
        i = i + 1
    endwhile
    return imax
```

- Proses harus dilakukan secara sekuensial
- Dengan membandingkan 2 data
  - Yang satu diketahui saat ini yang terbesar
  - Yang lain data baru yang harus diuji

Analisis Fungsional Kasus 1: Pencarian Nilai Maksimum

Analisis Algoritma: Kebenaran Algoritma

- INDUKSI
  - **Loop invarian**: Properti yang terbentuk dari logika loop tersebut
  - **Basis**: Properti tersebut berlaku sejak awal memasuki loop pertama kali
  - **Induksi**: Properti tersebut berlaku setiap kembali ke-awal loop**,** s.d. iterasi terakhir
  - **Terminasi**: Kombinasi kondisi terminasi dan loop invarian membuktikan kebenaran algoritma
- KONTRADIKSI
  - Mulai dengan **klaim bahwa algoritma tersebut salah**
  - Turunkan **konsekuensi** dari klaim tersebut
  - Temukan **kondisi konflik** dengan bagaimana algoritma tersebut bekerja
- Bukti DENGAN CONTOH
  - Untuk menunjukkan bahwa algoritma tersebut **SALAH**

Mencari Nilai Maksimum dalam Array

```
func FindMax( A[1..n] ) : index
    imax := 1
    i := 2
    { L.I.: A[imax] ≥ A[1..i-1] }
    while i ≤ n do
        if A[i] > A[imax] then
            imax := i
        endif
        i := i + 1
    endwhile
    { TERM.: i > n }
    return imax
```

Loop Invarian (L.I.):

- Kondisi yang relevan dengan obyektif algoritma tersebut
- Situasi incremental menuju hasil akhir algoritma tsb.

- Bukan kondisi iterasi
- Karena kondisi tersebut tidak berlaku setelah terminasi!

Mencari Nilai Maksimum dalam Array

```
func FindMax( A[1..n] ) : index
    imax := 1
    i := 2
    { L.I.: A[imax] ≥ A[1..i-1] }
    while i ≤ n do
        if A[i] > A[imax] then
            imax := i
        endif
        i := i + 1
    endwhile
    { TERM.: i > n }
    return imax
```

- **Basis**: i==2, A[imax==1]≥A[1..i-1=1] atau A[1] ≥ A[1] berarti kondisi tersebut berlaku diawal eksekusi.
- **Asumsi** A[imax] ≥ A[1..t-1] betul untuk semua 1 < t ≤ n
  - (t berlaku dari nilai awal s.d. sebelum terminasi)
- **Induksi** terhadap proses dalam loop:
  - Ketika **i = t → L.I benar untuk i**
  - Jika A[i] ≤ A[imax] maka imax tidak diubah, dan berlaku kondisi A[imax] ≥ A[1..i]
  - Jika A[i] > A[imax] maka imax diubah ke-i dan berlaku A[imax==i] ≥ A[1..i]
  - Pada baris terakhir dalam loop, nilai i bertambah, atau berlaku i == t+1, dan kita peroleh kembali kondisi LI semula **A[imax] > A[1..i-1] untuk nilai i yang baru**
- **Terminasi**:
  - i > n (karena incremental, i==n+1)
  - dan A[imax] ≥ A[i..i-1], sehingga A[imax] ≥ A[1..n]
  - Kesimpulannya **A[imax] nilai terbesar dalam array tsb**

Mencari Nilai Maksimum dalam Array

```
func FindMax( A[1..n] ) : index
    imax := 1
    i := 2
    { L.I.: A[imax] ≥ A[1..i-1] }
    while i ≤ n do
        if A[i] > A[imax] then
            imax := i
        endif
        i := i + 1
    endwhile
    { TERM.: i > n }
    return imax
```

Jika A[ix] ≠ A[imax] maka apakah A[ix] > A[imax] dan/atau A[ix] < A[imax]?
Jika A[ix] > A[imax] akan masuk ke if-..
Jika A[ix] < A[imax], asumsi bilang ..

- **Ambil** A[imax], untuk imax=1..n, bukan nilai yang terbesar dalam array atau A[imax] ⊁ A[1..n]
- **Maka** tentunya ada ix=1..n dimana A[ix] > A[imax]
- **Tetapi** dalam loop yang diberikan, seluruh data dalam array pasti diuji paling sedikit satu kali, termasuk A[ix]
- **Artinya** pada saat iterasi i=ix A[imax] saat itu akan diuji terhadap A[ix]
- **Karena** pengujian (instruksi if) tidak mungkin salah, tentunya indeks imax akan diisi nilai indeks ix
- **Dan** iterasi lanjutan setelah i=ix s.d. n dengan alasan yang sama (instruksi if) tentunya juga tidak akan mengubah nilai imax lagi.
- **Sehingga** diperoleh A[ix] == A[imax]
- Terjadi **kontradiksi** dan hanya bisa disimpulkan bahwa A[imax] ≥ A[1..n] atau A[imax] nilai terbesar dalam array

Mencari Nilai Maksimum dalam Array

```
func FindMax( A[1..n] ) : index
    imax := 1
    i := 2
    while i < n do
        if A[i] > A[imax] then
            imax := i
        endif
        i := i + 1
    endwhile
    return imax
```

- Contoh: n=5 dengan A[1..n] = {3, 2, 13, 7, 17}
- dst . . .

Analisis Non-Fungsional Kasus 1: Pencarian Nilai Maksimum

Analisis Algoritma

1. Kebenaran Fungsi
   a. Pembuktian BENAR dengan INDUKSI atau KONTRAKDIKSI
   b. Pembuktian SALAH dengan CONTOH
2. Karakteristik Non-Fungsi
   a. Sumber daya: WAKTU, MEMORI, BANDWIDTH, ...
   b. Skalabilitas, Bounded, Real-time,
   c. Stabilitas
3. Analisis Teori vs. Evaluasi Eksperimen?
   a. Algoritma vs. Program (Bukti vs. Afirmasi)
   b. Besar obyek masalah yang dihadapi (Lingkung lab vs. Data riil)

Analisis Non-Fungsional Algoritma: Kecepatan

- Kecepatan **eksekusi program** bergantung pada:
  - Platform hardware yang digunakan
  - Situasi eksternal eksekusi (temperatur, tegangan listrik, overclocking, …)
  - Interaksi dengan proses lain (persaingan sumber daya, sinkronisasi I/O, …)
  - Bahasa pemrograman dan perkakas yang digunakan (interpreted/translated)
  - Style pemrograman / pemrogram
  - **Algoritma** yang diterapkan
- Mem**prediksi dari algoritma** atas eksekusi program
  - Evaluasi setiap detil instruksi yang digunakan, atau
  - Identifikasi instruksi yang esensial yang menjadi parameter penting (independen),
  - Dan kalkulasi pengaruhnya dalam eksekusi algoritma
  - Biaya per jenis instruksi: seragam / berbeda

Mencari Nilai Maksimum dalam Array

```
func FindMax( A[1..n] ) : index
    imax := 1
    i := 2
    while i ≤ n do
        if A[i] > A[imax] then
            imax := i
        endif
        i := i + 1
    endwhile
    return imax
```

- Operasi perbandingan diduga paling dominan karena yang akan paling sering dikerjakan dalam algoritma ini
- Instruksi assignment imax := i belum tentu sesering operasi perbandingan
- Iterasi terjadi untuk variabel i dari 2 s.d. n, atau (n-1) kali
- Sehingga jumlah perbandingan adalah (n-1) kali
- Diperkirakan waktu eksekusi akan berbanding lurus dengan jumlah perbandingan.
    - Jika waktu eksekusi untuk $n=n_0$ diketahui $t=t_0$
    - Maka diprediksi untuk $n=n_1$ waktu eksekusinya adalah $t_1=(t_0/n_0)*n_1$
    - Dimana $c=t_0/n_0$ adalah konstanta eksekusi

Misalkan untuk n_0=1K data, eksekusi adalah t_0=1 detik, berapa waktu eksekusi jika n=1M data**?**
Juga berapa data yang dapat diolah jika waktu yang tersedia adalah t=1jam**?**

Kasus 2:
Pencarian Suatu Nilai

Mencari Suatu Nilai dalam Array

Diberikan suatu nilai yang ingin dicari, dan sejumlah data/nilai dalam array
Cari nilai tersebut dalam array dan berikan posisi/indeks nilai tersebut dalam array (jika ada)

Tidak perlu menguji semua data, dapat berhenti apabila data yang dicari sudah ditemukan.

- Saat mencari suatu nilai, nilai yang dicari diketahui, tetapi keberadaan dalam array tidak bisa dipastikan sebelumnya
- Saat mencari nilai ekstrim, nilai yang dicari belum tahu tapi pasti ada dalam array

Mencari Suatu Nilai dalam Array

- Strategi Brute-force diperlukan:
  - Nyata sejak awal bahwa **mungkin** seluruh data harus diperiksa
  - Jika tidak maka data yang terlewat diperiksa mungkin adalah yang dicari
- Dengan membandingkan nilai dicari terhadap setiap elemen dalam array
- Jika posisi atau nilai data dalam array mempunyai **keteraturan**:
  - Posisi dapat diduga dari nilainya
  - Berhenti mencari jika sisa data diduga tidak berisi nilai yang dicari
  - Dapat memangkas data dengan cerdik (binary search)

Mencari Suatu Nilai dalam Array

```
func Search( A[1..n], v ) : index
    i := n
    { L.I.: . . . }
    while i ≥ 1 and A[i] <> v do
        i := i - 1
    endwhile
    { TERM: . . . }
    return i
```

- Proses harus dilakukan secara sekuensial
- Dengan membandingkan v terhadap setiap elemen array A
- Jika data v tidak ada dalam array, maka fungsi mengembalikan nilai 0

Mencari Suatu Nilai dalam Array

```
func Search( A[1..n], v ) : index
    i := n
    { L.I.: A[i+1..n] <> v }
    while i ≥ 1 and A[i] <> v do
        i := i - 1
    endwhile
    { TERM: . . . }
    return i
```

- Setelah proses setengah berjalan
- Dan masih belum menemukan data yang dicari
- Apakah properti dari data yang sudah diproses?

Mencari Suatu Nilai dalam Array

```
func Search( A[1..n], v ) : index
    i := n
    { L.I.: A[i+1..n] <> v }
    while i ≥ 1 and A[i] <> v do
        i := i - 1
    endwhile
    return i
    { TERM.: i < 1 or A[i] == v }
```

Jika rentang A[n..n] artinya:...?

Mungkinkah i == 0 and A[i] == v?

- **Basis**: i==n, A[n+1..n] rentang array nol, L.I. terpenuhi.
- **Asumsi** L.I. berlaku saat memasuki loop
- **Induksi** terhadap proses dalam loop:
  - Saat masuk loop L.I. berlaku A[i+1..n] <> v
  - Kalau masuk loop maka otomatis A[i] <> v
  - Sehingga menjadi berlaku A[i..n] <> v
  - Setelah eksekusi satu[2]nya instruksi dalam loop, kembali kita peroleh kondisi L.I. A[i+1..n] <> v
- **Terminasi**:
  - i < 1 (karena decremental maka i==0), dengan kondisi A[i+1..n] <> v, maka berlaku A[1..n] <> v yang artinya TIDAK ADA v dalam array A
  - **ATAU** A[i]==v dengan A[i+1..n] <> v
  - Disimpulkan bahwa **v ditemukan pada posisi ke-i jika i > 0 atau tidak ada v dalam array A jika i == 0**

Mencari Suatu Nilai dalam Array

```
func Search( A[1..n], v ) : index
    i := n
    while i ≥ 1 and A[i] <> v do
        i := i - 1
    endwhile
    return i
```

- Badan loop praktis kosong
- Sehingga operasi yang dominan adalah perbandingan elemen array terhadap v
- Bagian dominan masih loop tersebut, dengan iterasi untuk i dari n s.d. 1, atau maksimum sebanyak n kali

- Jika nilai v tidak ditemukan, iterasi adalah sebanyak n kali
- Jika nilai v ditemukan sebagai elemen pertama, iterasi sebanyak n-1 kali
- Jika nilai v ditemukan sebagai elemen terakhir, iterasi tidak ada, 0 kali
- **Rerata** jumlah iterasi saat mencari nilai v baik ditemukan maupun tidak adalah T=(0+1+...+n-1+n)/(n+1) = ((n+0)×(n+1)/2) / (n+1) = ½n iterasi
  (Dan asumsi kemungkinan sebaran nilai v adalah uniform/merata)

Akhir Topik Minggu 01