IF1210 Algoritma dan Pemrograman 1

ADT Set, Map, serta Hash

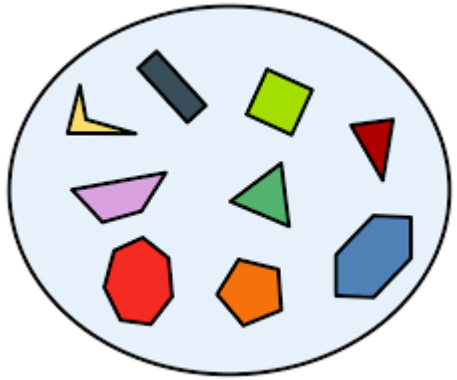
Tim Pengajar IF1210

Sekolah Teknik Elektro dan Informatika

ADT Set

Set – Definisi

- **Set** adalah kumpulan objek yang:
  - Memiliki tipe yang sama
  - Setiap elemennya unik
  - Elemen-elemennya tidak memiliki keterurutan: tidak ada istilah elemen ‘next’ dan ‘previous’.



Set – Definisi Fungsional

Jika diberikan *S*, *S1*, *S2* adalah **Set** dengan elemen *ElmtS*

```
   CreateSet: → S                  { Membuat sebuah set kosong }
     isEmpty: S → boolean          { Tes terhadap S: true jika S kosong,
                                     false jika S tidak kosong }
      length: S → integer          { Mengirimkan banyaknya elemen S }
         add: ElmtS × S → S        { Menambahkankan ElmtS ke S, jika ElmtS belum
                                     terdapat di dalam S }
      remove: ElmtS × S → S        { Menghapus ElmtS dari S }
        isIn: ElmtS × S → boolean  { Mengembalikan true jika ElmtS ada di dalam S }
     isEqual: S1 × S2 → boolean    { Mengembalikan true jika S1 dan S2 memiliki
                                     elemen yang sama }
       union: S1 × S2 → S          { Menghasilkan gabungan S1 dan S2 }
intersection: S1 × S2 → S          { Menghasilkan irisan S1 dan S2 }
setDifference: S1 × S2 → S         { Menghasilkan S1 dikurangi S2 }
        copy: S → S                { Mengcopy set S ke set baru }
    isSubset: S1 × S2 → boolean    { Mengembalikan true jika S1 adalah subset
                                     dari S2 }
```

Axiomatic Semantics (fungsional)

1) CreateSet() returns a set
2) isIn(v, CreateSet()) = false
3) isIn(v, add(v, S)) = true
4) isIn(v, add(u, S)) = isIn(v , S) if v ≠ u
5) remove(v, CreateSet()) = CreateSet()
6) remove(v, add(v, S)) = remove(v, S)
7) remove(v, add(u, S)) = add(u, remove(v, S)) if v ≠ u
8) isEmpty(CreateSet()) = true
9) isEmpty(add(v, S)) = false

Dengan S adalah set dan u, v adalah elemen.
(Dapatkah Anda membuat aksioma tambahan untuk union, intersection, setDifference?)

Implementasi Set dengan Tabel

- Memori tempat penyimpan elemen adalah sebuah array dengan indeks 0..CAPACITY-1.
- Pertimbangan implementasi: operasi terhadap set biasanya memeriksa keanggotaan sebuah elemen di dalam set.
  - Jika himpunan elemen set memiliki definisi urutan, elemen-elemen set dapat disusun secara terurut.
    - Efisiensi operasi pemeriksaan keanggotaan (isIn) → algoritma *binary search*.
  - Jika elemen-elemen set berukuran besar/berstruktur kompleks, operasi pemeriksaan keanggotaan (isIn) dapat menjadi sangat “mahal”.
    - Untuk efisiensi, bisa memanfaatkan *hash table*.

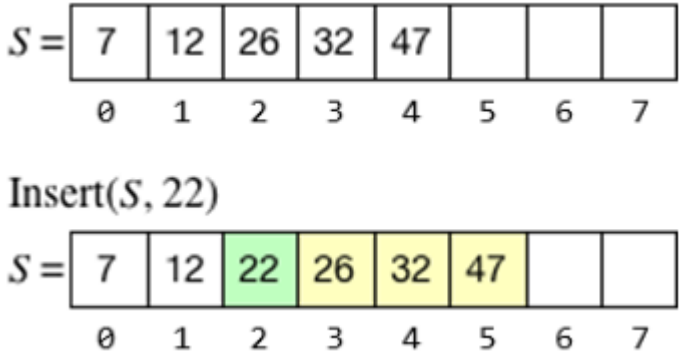
ADT Set (dengan array eksplisit-statik)

```
KAMUS UMUM
constant CAPACITY: integer = ... { Banyaknya elemen maksimum }

type ElType: integer  { elemen set }

type Set:
    < buffer: array [0..CAPACITY-1] of ElType,  { array penyimpan elemen set }
      length: integer  >                        { jumlah elemen set }
```

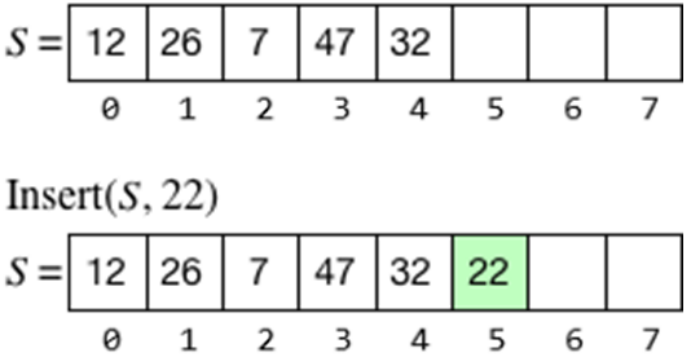
Latihan

Tuliskan algoritma-algoritma untuk **`add`**, **`remove`**, **`isIn`**, dan **`union`** terhadap Set of integer, jika elemen-elemen Set disimpan secara:

1. Terurut berdasarkan nilainya



2. Berdasarkan urutan dilakukannya insert



Lakukan analisis terhadap kedua alternatif cara penyimpanan, yang mana yang memberikan kinerja lebih baik untuk operasi-operasi tersebut!

Penjelasan Latihan: Ilustrasi Operasi

```
s = {12, 26, 7, 47, 32}
t = {5, 12}

add(s,22)    → s = {12, 26, 7, 47, 32, 22}
add(s,26)    → s = {12, 26, 7, 47, 32, 22}
remove(s,47) → s = {12, 26, 7, 32, 22}
isIn(7,s)  = true
isIn(7,t)  = false
union(s,t) = {12, 26, 7, 32, 22, 5}
```

ADT Map/Associative Array

Map – Definisi

- **Map** adalah kumpulan pasangan/*binding* ⟨*key*, *value*⟩, dengan nilai *key* bersifat unik di dalam kumpulan tsb.
- Dikenal juga sebagai *associative array*, *symbol table*, atau *dictionary*.
- Operasi terhadap Map:
  - Penambahan sebuah pasangan baru
  - Penghapusan suatu pasangan
  - Modifikasi nilai *value* dari suatu pasangan
  - Pencarian nilai *value* dari suatu *key* tertentu.

Map – Definisi Fungsional

Jika diberikan M adalah **Map** dengan elemen pasangan ⟨K,V⟩

```
   CreateMap: → M                 { Membuat sebuah Map kosong }
      isEmpty: M → boolean        { Tes terhadap M: true jika M kosong,
                                    false jika M tidak kosong }
          set: K × V × M → M      { Menambahkankan pasangan (K,V) ke M jika
                                    belum ada elemen dengan nilai key=K pada M,
                                    atau mengubah nilai value dari pasangan
                                    dengan nilai key=K menjadi V }
        unset: K × M → M          { Menghapus pasangan dengan nilai key=K
                                    dari M }
         find: K × M → V          { Mengembalikan value dari pasangan dengan
                                    nilai key=K }
```

Axiomatic Semantics (fungsional)

1) CreateMap() returns a map
2) find(k, set(k, v, M)) = v
3) find(k, set(j, v, M)) = find(k, M) if k ≠ j
4) unset(k, CreateMap()) = CreateMap()
5) unset(k, set(k, v, M)) = unset(k, M)
6) unset(k, set(j, v, M)) = set(j, v, unset(k, M)) if k ≠ j

dengan k dan j adalah key, v adalah value, dan M adalah map.

Map – Implementasi

- Jika jumlah elemen sedikit, dapat digunakan *association list* dalam bentuk array dengan elemen pasangan ⟨*key, value*⟩.
- Jika nilai *key* terbatas pada selang nilai integer tertentu, dapat digunakan *direct addressing* terhadap tabel: pasangan ⟨*key, value*⟩ disimpan dengan memberi nilai *value* pada indeks ke-*key*.
- Jika nilai *key* memiliki rentang yang besar namun data hanya sedikit, *direct addressing* menjadi boros memori.
  - Perlu dilakukan pemetaan *key* ke *address*/indeks.
  - Implementasi yang paling umum digunakan adalah menggunakan *hash table*.

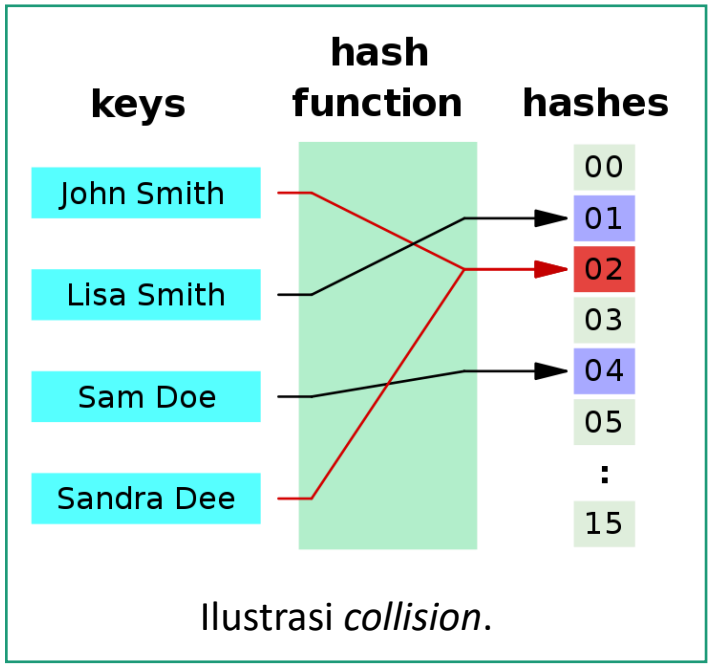
ADT Map (bandingkan dengan Set!)

```
KAMUS UMUM
constant  CAPACITY: integer = ... { Banyaknya elemen maksimum }
constant VAL_UNDEF: ElType = ...

type KeyType: ...
type ElType: ...
type MapEntry: < key: KeyType, value: ElType > { elemen map }

type Map:
    < buffer: array [0..CAPACITY-1] of MapEntry, { array penyimpan elemen map }
      length: integer >                          { jumlah elemen map }
```

Hash dan ADT Map

Hash dan Fungsi Hash

- **Fungsi hash**: fungsi yang dapat digunakan untuk memetakan data berukuran "berapa pun" menjadi nilai berukuran tetap/tertentu.
  - Misal, string dengan panjang berbeda-beda dipetakan ke satu byte.
- **key**: data yang menjadi masukan **fungsi hash**.
- **hash** atau **digest**: nilai hasil perhitungan **fungsi hash**.
- Karena nilai **hash** berukuran tetap (dan umumnya berukuran lebih kecil dari pada **key**), dapat terjadi **collision**. ☞
- Fungsi hash yang baik:
  1. Komputasinya cepat,
  2. Meminimalisir terjadinya collision.

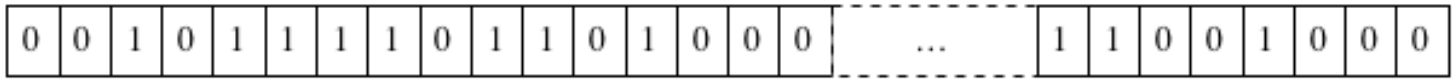


Ilustrasi *collision*.

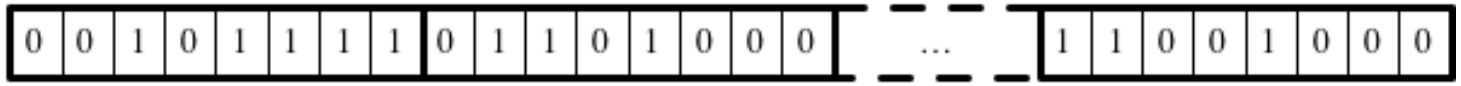
Contoh Fungsi Hash

Berikut contoh **fungsi hash** sangat sederhana menggunakan operasi XOR.
(Tidak *secure*, rentan *collision*.)

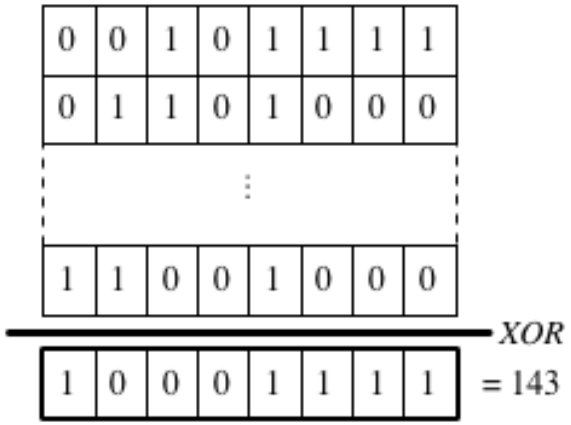
① Untuk memetakan *key* sepanjang apapun menjadi *hash* sepanjang 8 bit…



② Bagi *key* menjadi potongan-potongan dengan panjang 8 bit…



③ Kemudian lakukan operasi bitwise XOR secara beruntun terhadap setiap potongan tersebut.



④ Didapatlah nilai *hash* = 143.

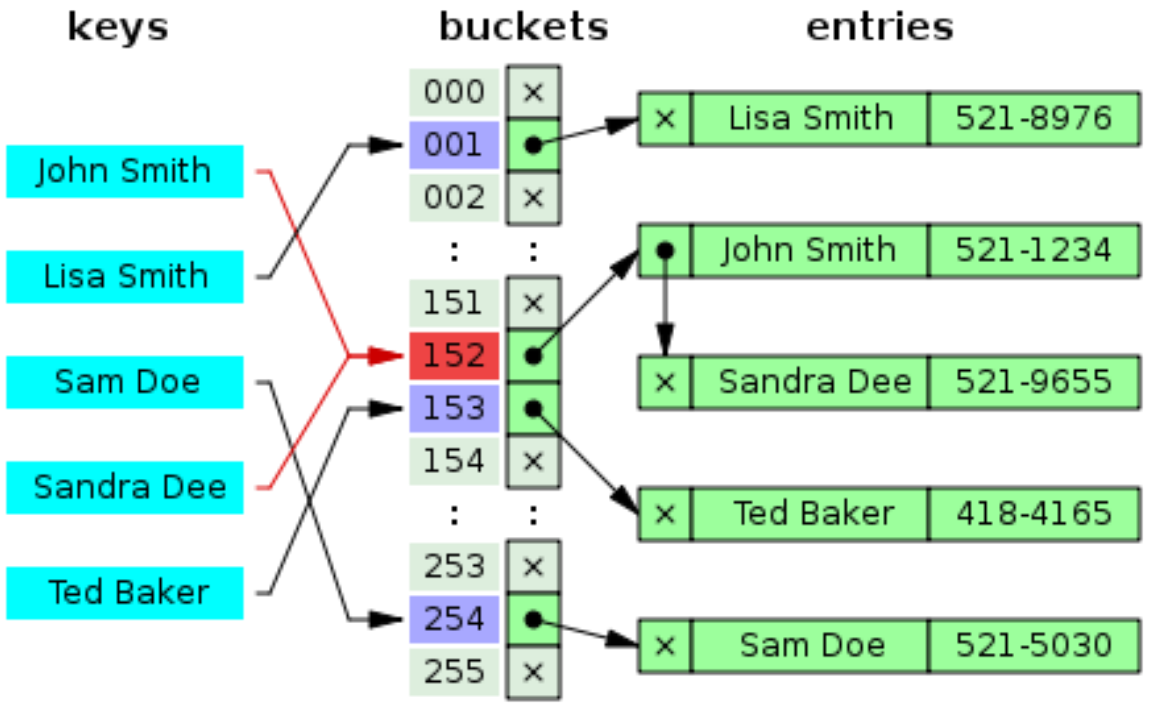
Hash Table

Dengan memanfaatkan **fungsi hash** untuk menentukan di mana data disimpan dan bagaimana menemukannya kembali:

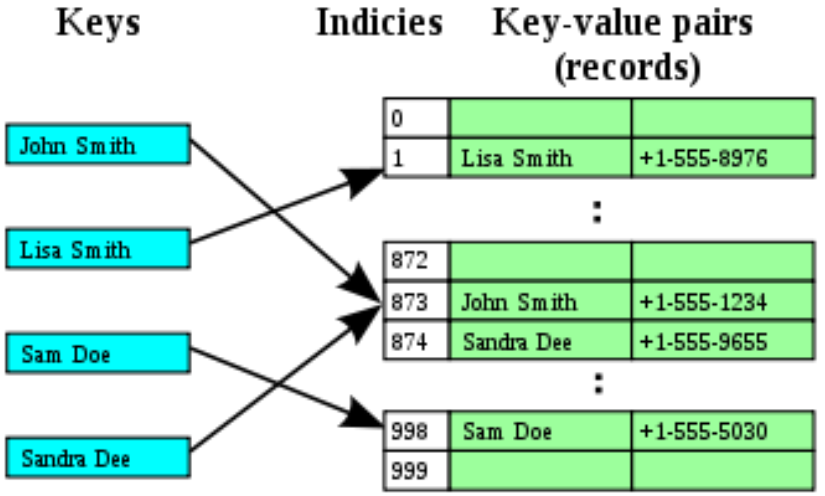
1) **Fungsi hash** mengubah **key** menjadi sebuah **hash** yang digunakan sebagai **indeks**.
2) **Key** disimpan pada **slot** sesuai **indeks** tersebut.
3) Pada saat hendak menyimpan, **slot** bisa *kosong* atau *terisi*.
   Slot *terisi* artinya sudah ada data lain yang **hash**-nya sama (terjadi **collision**).

   Sejumlah ***collision resolution strategies*** dikembangkan, biasanya berbasis:
   a) *Hash chaining*: menyimpan sebuah *association list* berukuran kecil pada setiap sel *hash table*.
   b) *Open addressing*.
4) **Key** dan **value** disimpan pada **slot** *kosong* yang ditemukan.

a) Hash Chaining



b. Open Addressing/Closed hashing



Collision diselesaikan dengan cara mencari pada lokasi alternatif hingga pasangan yang diinginkan ditemukan atau ditemukan slot array yang belum terpakai, yang berarti belum ada pasangan dengan nilai *key* yang dicari.

Metode pencarian yang dikenal:

1) *Linear probing*: interval pencarian tetap – biasanya 1.
2) *Quadratic probing*: interval pencarian bertambah secara linier – indeks dideskripsikan dengan fungsi kuadratik.
3) *Double hashing*: interval pencarian berikutnya ditentukan menggunakan fungsi *hash* yang lain.

Kembali ke ADT Map...

Contoh Algoritma *set* dengan Hash

```
procedure set(input/output m: Map, input k: KeyType, input v: ElType)
{ I.S. m terdefinisi, tidak penuh. }
{ F.S. Terdapat sebuah entri dengan key k pada m, dengan value=v. }
KAMUS LOKAL
  idx: integer
  found: boolean
ALGORITMA
  found ← false
  idx ← hash(k)
  while m.buffer[idx] ≠ NIL and not found do
    if m.buffer[idx].key = k then
      found ← true
    else
      idx ← idx+1
  if found then
    m.buffer[idx].value ← v
  else
    m.buffer[idx] ← <k,v>
    m.length ← m.length+1
```

Carilah kejanggalan pada algoritma ini!

Asumsi: fungsi hash(*k*:keytype) → address terdefinisi.
Collision ditangani dengan *linear probing*.
Slot kosong ditandai dengan nilai NIL.

Contoh Algoritma *find* dengan Hash

```
function find(m: Map, k: KeyType) → infotype
{ Mengembalikan value yang terasosiasi dengan key k pada m, atau
  VAL_UNDEF jika tidak ada key k di dalam m. }
KAMUS LOKAL
  idx: integer
  found: boolean
ALGORITMA
  found ← false
  idx ← hash(k)
  while m.buffer[idx] ≠ NIL and not found do
    if m.buffer[idx].key = k then
      found ← true
    else
      idx ← idx+1
  if found then
    → m.buffer[idx].value
  else
    → VAL_UNDEF
```

Asumsi: fungsi hash(*k*:keytype) → address terdefinisi.
Collision ditangani dengan *linear probing*.
Slot kosong ditandai dengan nilai NIL.

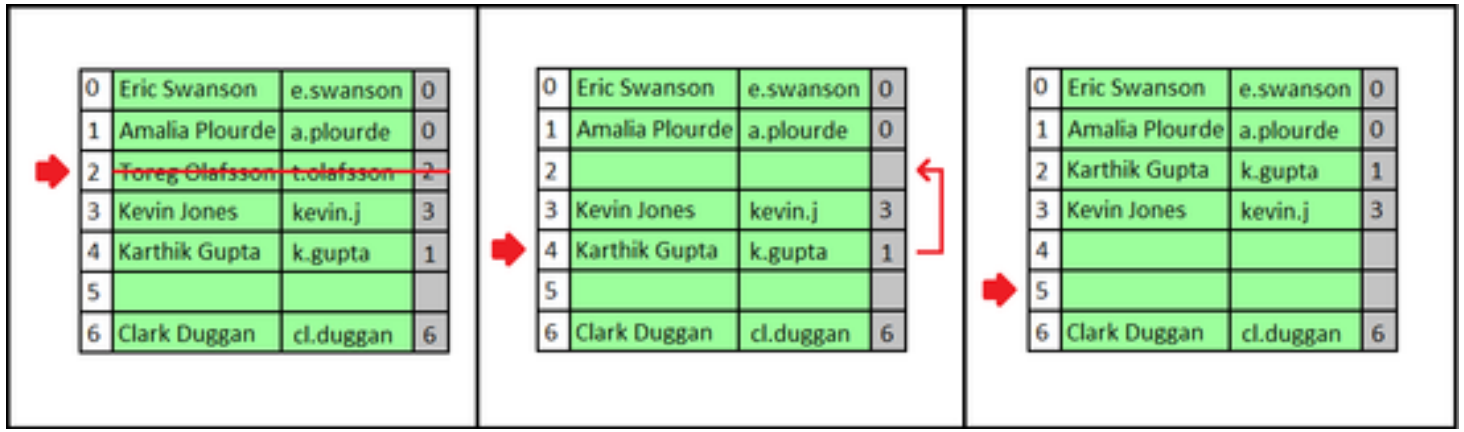
Load factor

- Pada algoritma set tadi, tidak ada kondisi berhenti jika pencarian slot kosong sudah mencapai ujung tabel.
- Alternatif:
  1. Pencarian berhenti → padahal mungkin masih ada slot kosong di indeks kecil?
  2. Pencarian lanjut ke indeks 0 lagi → menyulitkan operasi unset (bisa jadi harus memeriksa semua isi array)
- **Load factor**: jumlah slot terisi dibagi total slot yang tersedia.
- Hash table bekerja dengan baik saat load factor < 1. Maka alternatif berikutnya:
  3. Ketika mencapai suatu batas load factor tertentu, dialokasikan tabel baru yang lebih besar, kemudian setiap elemen Map di-hash ulang.

     Konsekuensi: tabel harus dinamis.

Penghapusan elemen?

- Karena algoritma pencarian berhenti jika menemukan slot kosong, penghapusan elemen (*unset*) bisa mengakibatkan pencarian berikutnya salah.
  - Mengira *not found*, padahal disimpan di slot berikutnya akibat *collision* ketika penambahan elemen.
- Untuk itu pada penghapusan elemen di indeks i diperlukan pencarian & pemindahan elemen yang mungkin bisa menempati slot i, yaitu elemen yang nilai *hash*-nya ≤ i.



Latihan

Jika `MapEntry` pada Map dengan Hash diganti dengan `ElType`,

dan operasi/ekspresi yang melibatkan `MapEntry`/`KeyType`/`ElType` diganti dengan `ElType`,

→ didapatkan implementasi ADT Set menggunakan Hash.

Buatlah kembali algoritma **`add`**, **`remove`**, **`isIn`**, dan **`union`** untuk Set dengan Hash, dan analisis lagi kinerjanya!