

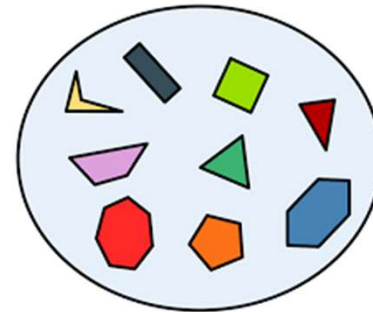
ADT Set

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

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

Axiomatic Semantics (fungsional)

- 1) $\text{new}()$ returns a set
- 2) $\text{isIn}(v, \text{new}()) = \text{false}$
- 3) $\text{isIn}(v, \text{add}(v, S)) = \text{true}$
- 4) $\text{isIn}(v, \text{add}(u, S)) = \text{isIn}(v, S)$ if $v \neq u$
- 5) $\text{remove}(v, \text{new}()) = \text{new}()$
- 6) $\text{remove}(v, \text{add}(v, S)) = \text{remove}(v, S)$
- 7) $\text{remove}(v, \text{add}(u, S)) = \text{add}(u, \text{remove}(v, S))$ if $v \neq u$
- 8) $\text{isEmpty}(\text{new}()) = \text{true}$
- 9) $\text{isEmpty}(\text{add}(v, S)) = \text{false}$

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 teratur.
 - Efisiensi operasi pemeriksaan keanggotaan ($isIn$) \rightarrow 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* }

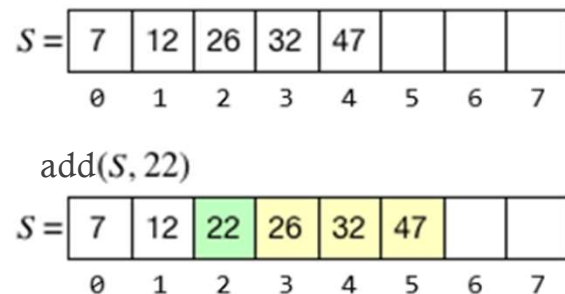
type Set:

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

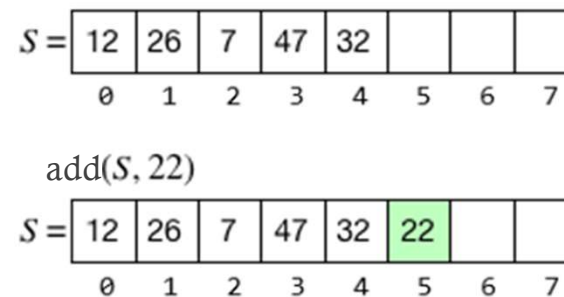
Latihan

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

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

$\text{add}(s, 22) \rightarrow s = \{12, 26, 7, 47, 32, \mathbf{22}\}$

$\text{add}(s, 26) \rightarrow s = \{12, \mathbf{26}, 7, 47, 32, 22\}$

$\text{remove}(s, 47) \rightarrow s = \{12, 26, 7, 32, 22\}$

$\text{isIn}(7, s) = \text{true}$

$\text{isIn}(7, t) = \text{false}$

$\text{union}(s, t) = \{12, 26, 7, 32, 22, 5\}$

ADT Map/ Associative Array

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Map – Definisi

Map adalah kumpulan pasangan/*binding* $\langle key, value \rangle$, 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 $\langle K, V \rangle$

CreateMap: $\rightarrow M$ { Membuat sebuah Map kosong }

isEmpty: $M \rightarrow \text{boolean}$ { Tes terhadap M : true jika M kosong,
false jika M tidak kosong }

set: $K \times V \times M \rightarrow M$ { 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: $K \times M \rightarrow M$ { Menghapus pasangan dengan nilai key= K
dari M }

find: $K \times M \rightarrow V$ { Mengembalikan value dari pasangan dengan
nilai key= K }

Axiomatic Semantics (fungsional)

- 1) $\text{new}()$ returns a map
- 2) $\text{find}(k, \text{set}(k, v, M)) = v$
- 3) $\text{find}(k, \text{set}(j, v, M)) = \text{find}(k, M)$ if $k \neq j$
- 4) $\text{unset}(k, \text{new}()) = \text{new}()$
- 5) $\text{unset}(k, \text{set}(k, v, M)) = \text{unset}(k, M)$
- 6) $\text{unset}(k, \text{set}(j, v, M)) = \text{set}(j, v, \text{unset}(k, M))$ if $k \neq j$

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 $\langle \text{key}, \text{value} \rangle$.

Jika nilai *key* terbatas pada selang nilai integer tertentu, dapat digunakan *direct addressing* terhadap tabel: pasangan $\langle \text{key}, \text{value} \rangle$ 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: EType = ...

type KeyType: ...

type EType: ...

type MapEntry: < key: KeyType, value: EType > { *elemen map* }

type Map:

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

Hash dan ADT Map

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

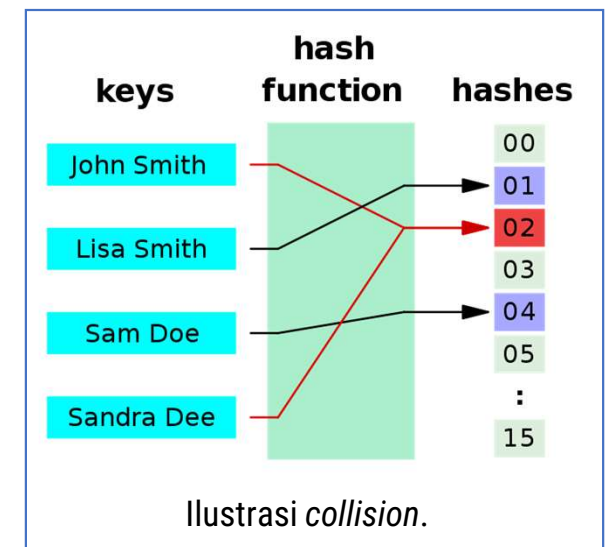
Hash dan Fungsi Hash

- **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*.



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...

0	0	1	0	1	1	1	1	0	1	1	0	1	0	0	0	...	1	1	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----	---	---	---	---	---	---	---	---

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

0	0	1	0	1	1	1	1	0	1	1	0	1	0	0	0	...	1	1	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----	---	---	---	---	---	---	---	---

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

0	0	1	0	1	1	1	1
0	1	1	0	1	0	0	0
⋮							
1	1	0	0	1	0	0	0
<hr/>							
1	0	0	0	1	1	1	1

XOR
= 143

④ Didapatlah nilai *hash* = 143.

Hash Table

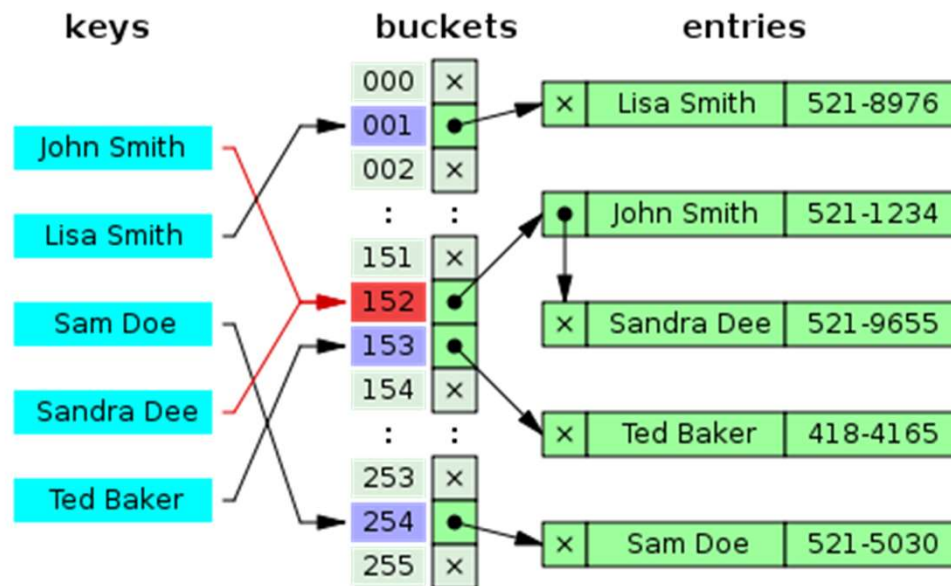
Dengan memanfaatkan **fungsi hash** untuk menentukan di mana data disimpan dan bagaimana menemukannya kembali:

- 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

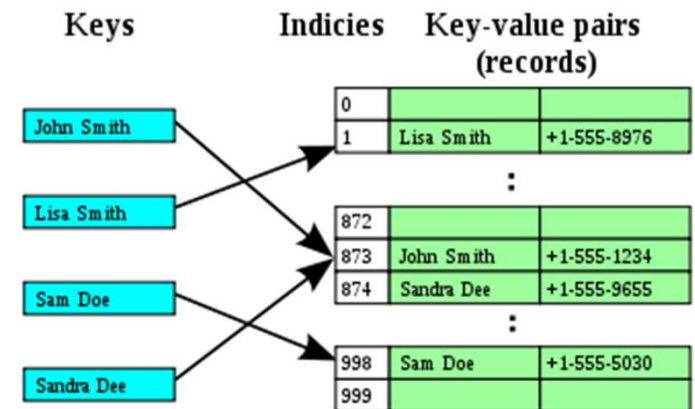


b. Open Addressing/Closed hashing

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 or 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 $\text{hash}(k:\text{keytype}) \rightarrow \text{address}$ 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 or 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 $\text{hash}(k:\text{keytype}) \rightarrow \text{address}$ 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.

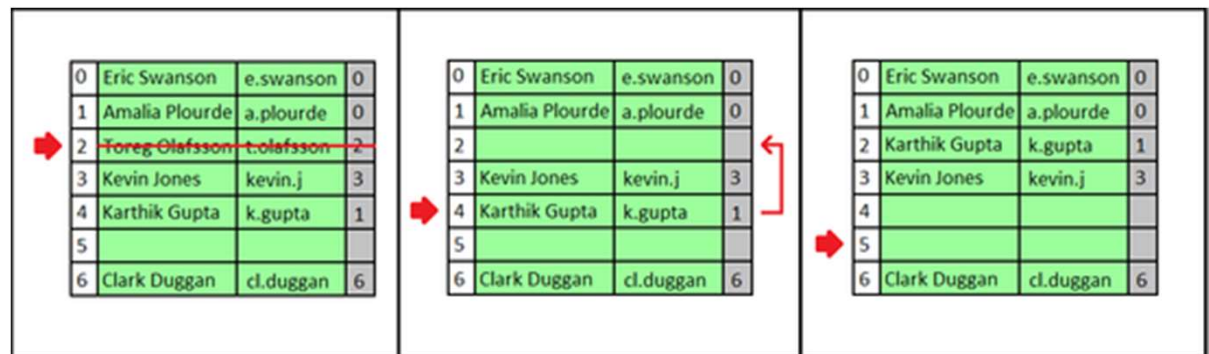
Konsekuensi: tabel harus dinamis.

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 $\leq i$.



Latihan

Jika MapEntry pada Map dengan Hash diganti dengan E1Type,

dan operasi/ekspresi yang melibatkan MapEntry/KeyType/E1Type diganti dengan E1Type,

→ didapatkan implementasi ADT Set menggunakan Hash.

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