Struktur Data Non-Linear: Disjoint Set

Tim Olimpiade Komputer Indonesia

Pendahuluan

Melalui dokumen ini, kalian akan:

- Mengenal dan mengimplementasikan struktur data *disjoint set*.
- Mengetahui mengapa diperlukan *disjoint set*.

Motivasi

Terdapat N orang di suatu ruangan, dinomori dari 1 sampai dengan N. Secara bertahap, mereka membentuk suatu kelompok.

Anda diberikan sejumlah operasi. Setiap operasi dapat berbentuk salah satu dari:

- join(a, b), artinya kelompok orang ke-a dan kelompok orang ke-b bergabung.
- check(a, b), artinya periksa apakah orang ke-a dan orang ke-b sedang berada di kelompok yang sama.

Contoh Perilaku

Dengan $N = 5$, Berikut contoh operasinya dan perilaku yang diharapkan:

- Pada kondisi awal, setiap orang dapat dianggap membentuk kelompok sendiri $[1], [2], [3], [4], [5]$.
- join(1, 4), kini kelompok yang ada adalah $[1, 4], [2], [3], [5]$.
- check(1, 2): laporkan bahwa 1 dan 2 berada di kelompok berbeda.
- join(1, 2), kini kelompok yang ada adalah $[1, 2, 4], [3], [5]$.
- ...

Contoh Perilaku (lanj.)

- ...
- check(1, 2): laporkan bahwa 1 dan 2 berada di kelompok yang sama.
- join(3, 5), kini kelompok yang ada adalah $[1, 2, 4], [3, 5]$.
- join(2, 3), kini kelompok yang ada adalah $[1, 2, 3, 4, 5]$.
- check(1, 5): laporkan bahwa 1 dan 5 berada di kelompok yang sama.

Solusi Sederhana

- Dengan konsep graf, representasikan setiap orang sebagai *node*.
- Setiap operasi join dapat dianggap dengan penambahan *edge*.
- Untuk operasi check, diperlukan penjelajahan graf untuk memeriksa apakah kedua *node* terhubung.
- Representasi graf yang paling efisien untuk keperluan ini adalah *adjacency list*.

Analisis Solusi Sederhana

Operasi	Solusi Sederhana
join(a, b)	$O(1)$
check(a, b)	$O(N)$ s.d. $O(N^2)$

Kompleksitas check mencapai $O(N^2)$ apabila penjelajahan graf dilakukan secara naif dan seluruh kemungkinan *edge* dilalui.

Masalah Solusi Sederhana

- Solusi sederhana ini jelas tidak efisien ketika banyak dilakukan operasi check(a, b).
- Kita akan mempelajari bagaimana *disjoint set* mengatasi masalah ini secara efisien.

Disjoint Set

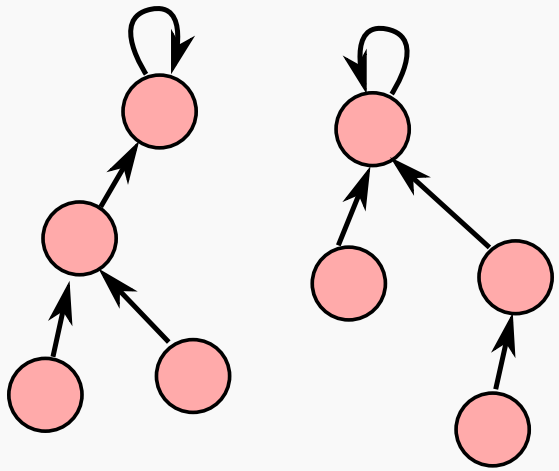
- *Disjoint set* merupakan struktur data yang efisien untuk mengelompokkan elemen-elemen secara bertahap.
- Operasi yang didukung adalah:
  - *join*, yaitu menggabungkan kelompok dari sepasang elemen.
  - *check*, yaitu memeriksa apakah sepasang elemen berada di kelompok yang sama.

Ide Dasar

- Untuk setiap kelompok yang ada, pilih suatu elemen sebagai "perwakilan kelompok".
- Setiap elemen perlu mengetahui siapa perwakilan kelompoknya.
- Untuk memeriksa apakah dua elemen berada pada kelompok yang sama, periksa apakah **perwakilan kelompok mereka sama**.
- Untuk menggabungkan kelompok dari dua elemen, **salah satu perwakilan kelompok elemen perlu diwakilkan oleh perwakilan kelompok elemen lainnya**.

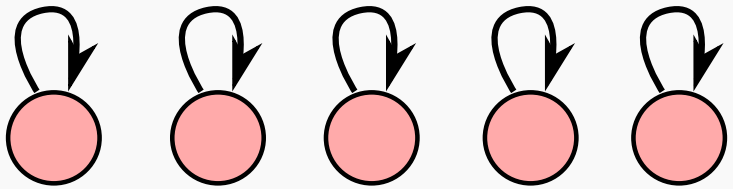
Cara Kerja Disjoint Set

- Setiap elemen perlu menyimpan *pointer* ke elemen yang merupakan perwakilannya.
- *Pointer* yang ditunjuk oleh suatu perwakilan kelompok adalah dirinya sendiri.



Cara Kerja Disjoint Set (lanj.)

- Karena pada awalnya setiap elemen membentuk kelompoknya sendiri, maka awalnya setiap *pointer* ini menunjuk pada dirinya sendiri.
- Untuk mempermudah, mari kita sebut *pointer* ini sebagai *parent*.



Inisialisasi Disjoint Set

Berikut prosedur inisialisasi *disjoint set* untuk N elemen.

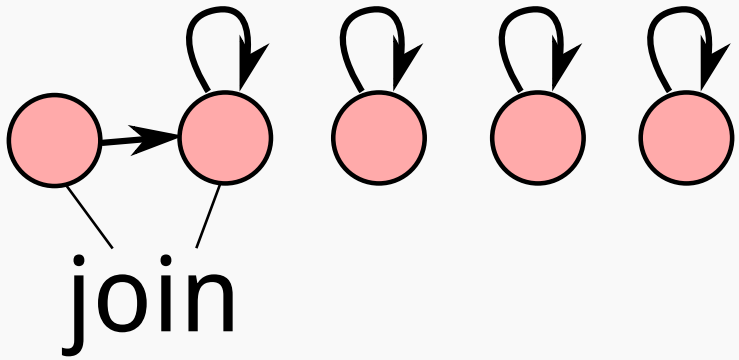
Asumsikan *array par* menyimpan indeks elemen yang ditunjuk sebagai *parent* dari suatu elemen.

```
INITIALIZE()
1   for i = 0 to N − 1 // Indeks elemen dimulai dari 0 (zero-based)
2       par[i] = i
```

Cukup sederhana, yaitu setiap elemen menunjuk ke dirinya sendiri.

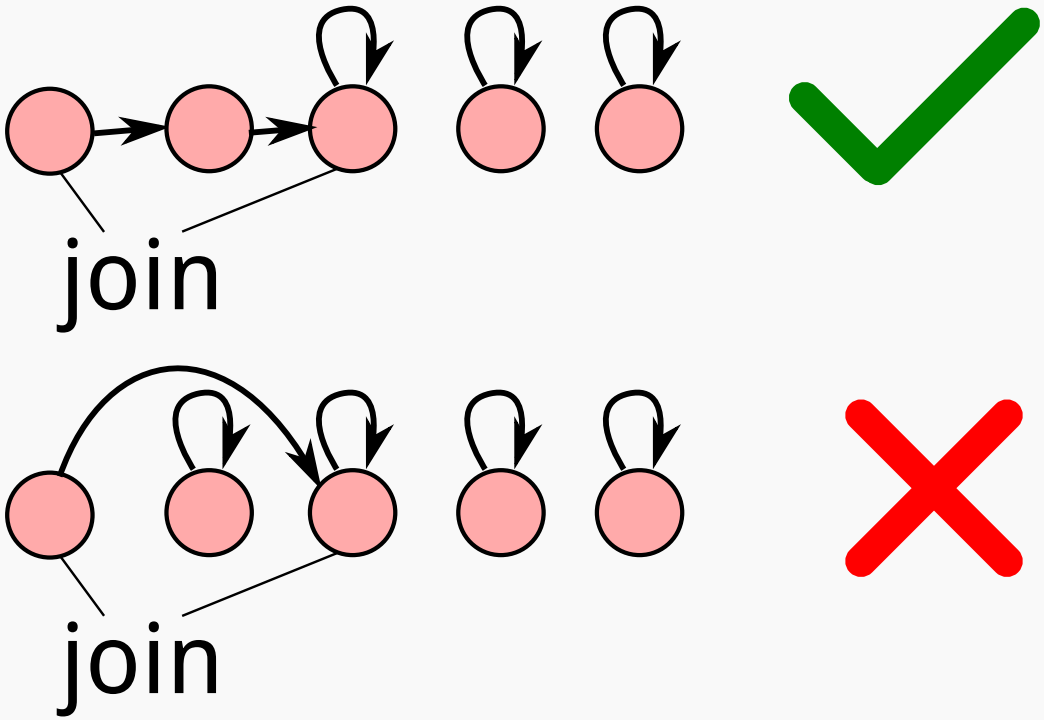
Operasi Join

- Ketika kelompok dua elemen perlu digabungkan, ubah *parent* dari salah satu perwakilan kelompok ke kelompok lainnya.



Operasi Join (lanj.)

- Perhatikan bahwa yang perlu diubah adalah *parent* dari perwakilan kelompok suatu elemen, bukan elemen itu sendiri.



Operasi Join (Implementasi)

Secara sederhana, operasi *join* dapat dituliskan dalam prosedur:

JOIN(a, b)

1 $repA = \text{FINDREPRESENTATIVE}(a)$
2 $repB = \text{FINDREPRESENTATIVE}(b)$
3 $par[repA] = repB$

Fungsi FINDREPRESENTATIVE(x) mengembalikan elemen perwakilan dari kelompok tempat elemen x berada.

Operasi Join (implementasi findRepresentative)

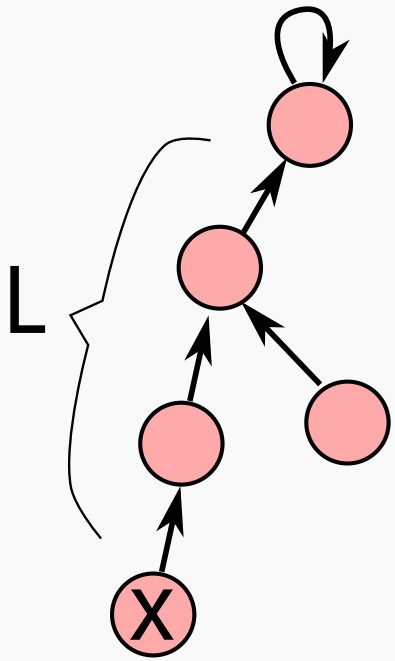
Fungsi FINDREPRESENTATIVE(x) dapat diimplementasikan secara rekursif, yaitu sampai ditemukan elemen yang memiliki *parent* berupa dirinya sendiri.

FINDREPRESENTATIVE(x)

```
1   if par[x] == x
2         return x
3   else
4         return FINDREPRESENTATIVE(par[x])
```

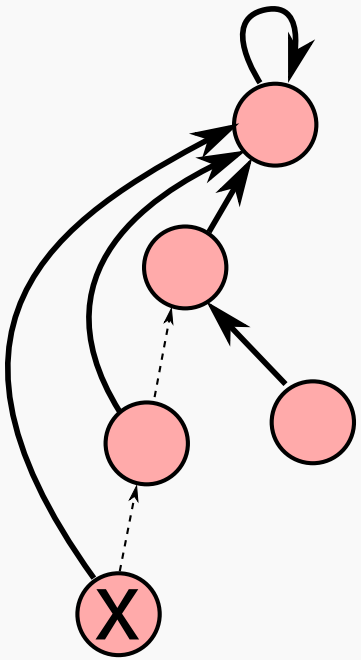
Kekurangan findRepresentative

- Fungsi FINDREPRESENTATIVE memiliki kompleksitas sebesar $O(L)$, dengan L adalah panjangnya jalur dari elemen x sampai elemen perwakilan kelompoknya.
- Ketika L mendekati N, fungsi ini tidak efisien bila dipanggil berkali-kali.



Memperbaiki findRepresentative

- Kita dapat menerapkan teknik *path compression*, yaitu mengubah nilai *parent* dari setiap elemen yang dilalui **langsung** ke elemen perwakilan kelompok.
- Hal ini menjamin untuk pemanggilan FINDREPRESENTATIVE berikutnya pada elemen yang bersangkutan bekerja secara lebih efisien.



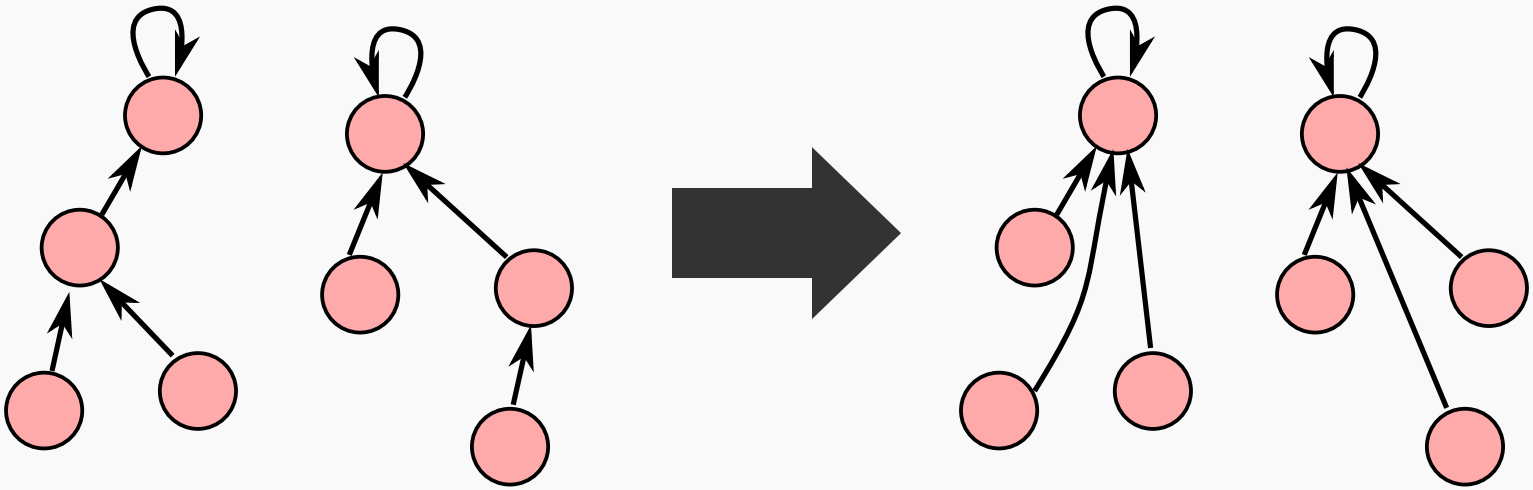
Implementasi Path Compression

Tambahkan pencatatan elemen perwakilan kelompok untuk setiap elemen yang dilalui.

```
FINDREPRESENTATIVE(x)
1   if par[x] == x
2         return x
3   else
4         // Catat elemen representatifnya
5         par[x] = FINDREPRESENTATIVE(par[x])
6         return par[x]
```

Analisis Kompleksitas findRepresentative

- Untuk *disjoint set* dengan N elemen, paling banyak terdapat N *parent* yang dikenakan *path compression*.
- Apabila seluruh *parent* elemen sudah dikenakan *path compression*, maka setiap elemen langsung menunjuk ke elemen perwakilan kelompoknya.
- Artinya, kini fungsi FINDREPRESENTATIVE bekerja dalam $O(1)$.



Analisis Kompleksitas findRepresentative (lanj.)

- Kompleksitas satu kali pemanggilan FINDREPRESENTATIVE tidak dapat didefinisikan secara pasti.
- Yang pasti adalah, **kompleksitas total** untuk pemanggilan FINDREPRESENTATIVE secara **berkali-kali** tidak akan lebih dari $O(N)$; sebab lebih dari itu dipastikan seluruh *parent* sudah terkompresi secara merata.
- Setelah seluruh *parent* terkompresi secara merata, kompleksitasnya adalah $O(1)$.

Analisis Kompleksitas Join

- Kembali ke operasi join, kompleksitasnya bergantung pada FINDREPRESENTATIVE.
- Dapat dikatakan kompleksitas operasi join sama dengan kompleksitas FINDREPRESENTATIVE.

Operasi Check

- Untuk operasi *check*, cukup periksa apakah elemen perwakilan kelompok kedua elemen sama.
- Lagi-lagi, kompleksitas operasi *check* sama dengan sama dengan kompleksitas FINDREPRESENTATIVE.

```
CHECK(a, b)
1  return FINDREPRESENTATIVE(a) == FINDREPRESENTATIVE(b)
```

Penutup

- *Disjoint set* merupakan struktur data yang sederhana dan mudah diimplementasikan.
- Biasanya struktur data ini dipakai untuk membantu implementasi algoritma lainnya, seperti *Minimum Spanning Tree Kruskal*.
- Setiap Anda mengingat operasi "gabung" dan "periksa", ingatlah *disjoint set*.