Pemrosesan Bahasa Alami Lanjut

Perkuliahan Pekan Ketujuh
Dependency parsing

Oleh: Ade Romadhony

Referensi

- Speech and Languange Processing, Dan Jurafsky, https://web.stanford.edu/~jurafsky/slp3/15.pdf
- Slide dari Natalie Parde, http://www.natalieparde.com/teaching/cs_421_fall2019/Syntactic%20and%20Dependency%20Parsing.pdf
- http://www.cs.umd.edu/class/fall2017/cmsc723/slides/slides_12.pdf

Kerangka Bahasan

- Apa itu *dependency parsing*
- *Transition-based dependency parsing*
- *Graph-based dependency parsing*

Apa itu *dependency parsing*?

Proses otomatis untuk menentukan keterkaitan tata bahasa (sintaktik) dan makna (semantik) antar kata dalam sebuah kalimat

- Keterkaitan sintaktik: berfokus pada struktur kalimat
- Keterkaitan semantik: berfokus pada arti/makna

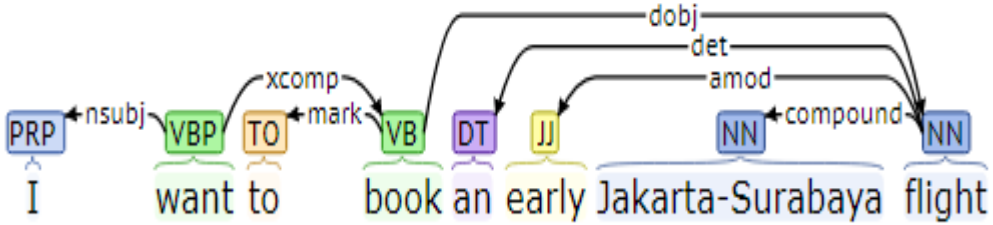
Perbedaan dengan CFG?

- CFG digunakan untuk membangkitkan struktur frase (serangkaian kata): NP (*noun phrase),* VP *(verb phrase)*, dll
- *Dependency grammar* tidak menggunakan struktur frase, melainkan memperhatikan ketarkaitan antar kata. Contoh tipe keterkaitan: *direct object, nominal object, nominal subject*, dll
- Persamaan: tipe keterkaitan (label) sudah terdefinisi

Tata bahasa *dependency* sesuai untuk bahasa dengan karakteristik sebagai berikut:

- Morfologisnya kaya: banyak infleksi (perubahan bentuk kata yang mempengaruhi makna atau relasi tata bahasa). Contoh: Bahasa Jerman, Bahasa Arab
- Urutan kata dalam sebuah kalimat relatif bebas

Contoh perbandingan pohon parse *constituency* vs *dependency*



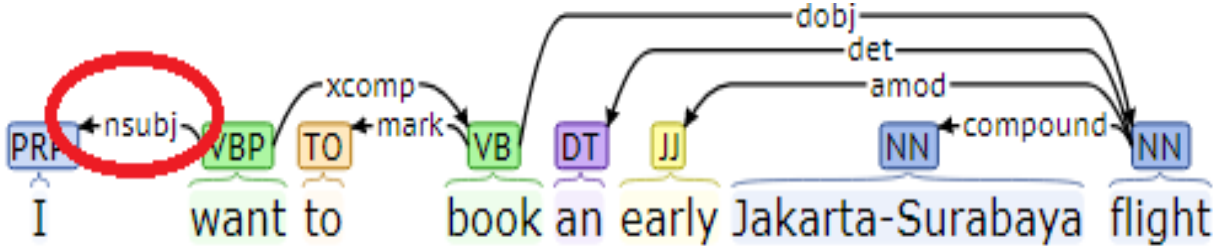
Parse tools yang digunakan:
https://parser.kitaev.io
http://nlp.stanford.edu:8080/corenlp/

Manfaat *dependency parsing*

- Memberikan (perkiraan) informasi keterkaitan semantik antar kata dalam kalimat
- Informasi keterkaitan semantik antar kata berguna untuk beberapa aplikasi, contoh:
  - Ekstraksi informasi
  - *Coreference resolution*
  - Sistem tanya jawab

Relasi dependensi

- Terdapat dua komponen:
  - *Head*/kepala
  - *Dependent*/yang bergantung
- Kepala terhubung secara langsung ke kata yang bergantung kepadanya
- Tipe relasi menunjukkan peran *dependent* terhadap *head*-nya. Contoh:



Label tipe dependensi antar kata

- Terdapat beberapa taksonomi yang dapat digunakan untuk pelabelan tipe dependensi (umumnya disebut sebagai Treebank). Contoh *dependency treebank* Bahasa Inggris:
  - Stanford *dependencies*
  - CoNLL *dependencies*
  - Universal *dependencies*
- Bagaimana caranya menyusun *treebank*?
  - Pelabelan manual oleh manusia
  - Awalnya dilakukan pelabelan secara otomatis, kemudian akan diperiksa oleh annotator manusia

Universal dependencies

- Sebuah kerangka kerja anotasi/pelabelan tata-bahasa yang berlaku global, tidak spesifik untuk sebuah bahasa tertentu
- Lihat https://universaldependencies.org/u/dep/index.html untuk deskripsi lebih detail

	Nominals	Clauses	Modifier words	Function Words
Core arguments	nsubj obj iobj	csubj ccomp xcomp		
Non-core dependents	obl vocative expl dislocated	advcl	advmod* discourse	aux cop mark
Nominal dependents	nmod appos nummod	acl	amod	det clf case
Coordination	**MWE**	**Loose**	**Special**	**Other**
conj cc	fixed flat compound	list parataxis	orphan goeswith reparandum	punct root dep

Definisi formal dependensi

- Struktur dependensi adalah graf berarah
  - G = (V, A)
    - V adalah himpunan simpul
    - A adalah himpunan sisi/busur
  - V: kata-kata dalam sebuah kalimat
    - Tanda baca dapat masuk ke dalam V
    - Pada bahasa yang morfologi-nya kompleks, dapat mencakup stem dan afiks
  - Busur menunjukkan keterkaitan tata-Bahasa antar kata
- Berdasar sebagian besar teori tata-bahasa, struktur dependensi harus memenuhi syarat berikut:
  - Terhubung
  - Mempunyai simpul akar
  - Asiklik

Pohon dependensi

- Graf berarah yang memenuhi konstrain berikut:
  - Mempunyai simpul akar tunggal. Simpul akar tidak boleh mempunyai busur masuk (*incoming arc*)
  - Semua simpul kecuali simpul akar mempunyai 1 busur masuk
  - Terdapat lintas unik dari simpul akar ke semua simpul lain

Mengubah struktur dependensi *constituency* ke *dependency*

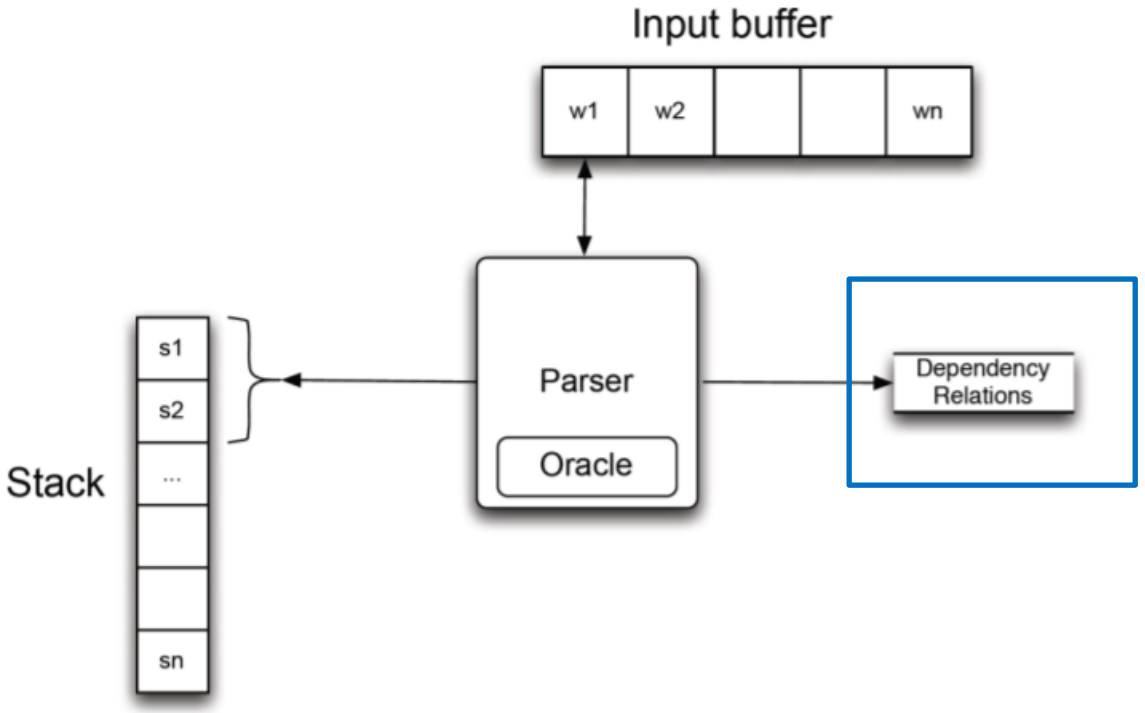
- Terdapat dua langkah untuk melakukan perubahan struktur tersebut:
  - Identifikasi semua relasi *head-dependent* pada pohon *constituent*
  - Identifikasi semua relasi dependensi yang benar untuk relasi yang ditemukan pada langkah sebelumnya
- Algoritma:
  - Tandai *head-child* pada tiap frase, berdasarkan aturan tertentu
  - Pada struktur dependensi, buat semua *head* dari *non-head child* bergantung pada *head* dari *head-child*

Tipe *dependency parser*

- *Transition-based*
  - Membangun sebuah pohon tunggal, dari arah kiri ke kanan (asumsi untuk bahasa yang teksnya dituliskan dari arah kiri ke kanan)
- *Graph-based*
  - Melakukan proses pencarian dari sejumlah pohon yang mungkin berdasar sebuah kalimat masukan, mencoba untuk mencari pohon dengan skor maksimum

Gambaran umum *transition-based dependency parsing*

- Berdasarkan *shift-reduce parsing* [Aho dan Ullman, 1927]
- Konfigurasi:
  - Stack
  - Input buffer kata-kata dalam kalimat
  - Himpunan relasi dependensi
- Tujuan *parsing*:
  - Mencari konfigurasi final di mana semua kata dalam kalimat mempunyai relasi dengan kata lainnya



Operator Transisi

- Transisi akan memproduksi sebuah konfigurasi baru berdasar konfigurasi saat ini
- *Parsing*: mencari sekuens transisi, dari *state* awal ke *state* akhir
- *State* awal/*start state*:
  - *Stack* diinisialisasi dengan simpul akar/*ROOT*
  - Input buffer diinisialisasi dengan kata-kata pada kalimat masukan
  - Himpunan relasi dependensi = {}
- *State* akhir/*end state*:
  - *Stack* dan daftar kata/*input buffer* kosong
  - Himpunan relasi dependensi -> hasil *parse* akhir

Sistem transisi *arc-standard* [1]

- Terdapat tiga operator transisi [Covington, 2001; Nivre 2003]:
  - LEFT-ARC:
    - Membuat relasi *head-dependent* antara kata yang berada di *top stack* dan yang berada tepat di bawah *top*
    - Kata yang berada di bawah *top* dihapus dari *stack*
  - RIGHT-ARC:
    - Membuat relasi *head-dependent* antara kata yang berada berada tepat di bawah *top* dan kata di *top stack*
    - *Top stack* dihapus dari *stack*
  - SHIFT:
    - Hapus kata yang ada di *head input buffer*
    - Masukkan kata tersebut ke *stack* (*push*)

Sistem transisi *arc-standard* [2]

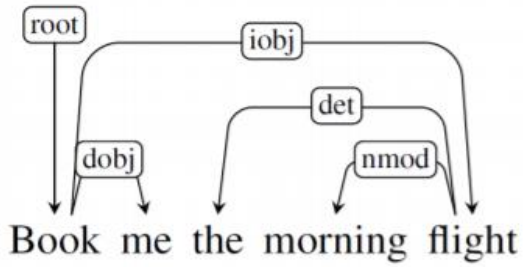
- Prekondisi:
  - Ingat: simpul akar/ROOT tidak mempunyai busur masuk
  - LEFT-ARC tidak dapat dilakukan saat ROOT berada dalam posisi nomor 2 di *stack*
  - LEFT-ARC dan RIGHT-ARC hanya dapat dilakukan saat pada *stack* terdapat minimal 2 elemen

Algoritma generik transition-based parsing

```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← ORACLE(state)             ; choose a transition operator to apply
    state ← APPLY(t, state)       ; apply it, creating a new state
  return state
```

- Terdapat asumsi oracle
- Kompleksitas *parsing* linier sesuai dengan panjang kalimat
- Termasuk dalam algoritma *Greedy*

Ilustrasi *transition-based parsing*



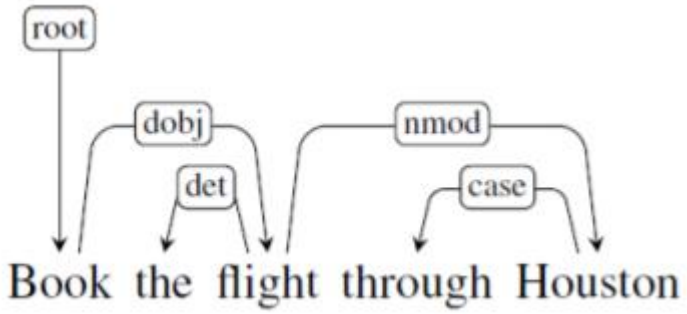
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Bagaimana cara mendapatkan *oracle*?

- Persoalan klasifikasi multi kelas
  - *Input*: *current parsing state*
  - *Output*: 1 transisi, dari semua kemungkinan transisi
- *Supervised classifier* seperti SVM, perceptron dapat digunakan
- Topik penelitian
  - Fitur apa yang sesuai untuk digunakan?
  - Data latih didapatkan dari mana?

Memperoleh data latih

- Informasi yang ada di Treebank

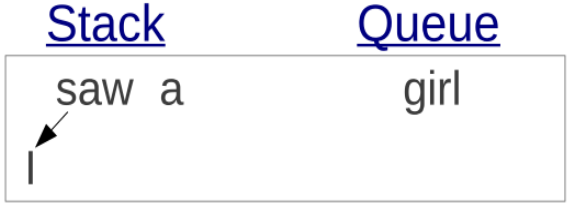


- Kebutuhan untuk melatih *oracle*: pasangan konfigurasi dan prediksi aksi

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Contoh klasifikasi untuk *shift-reduce*

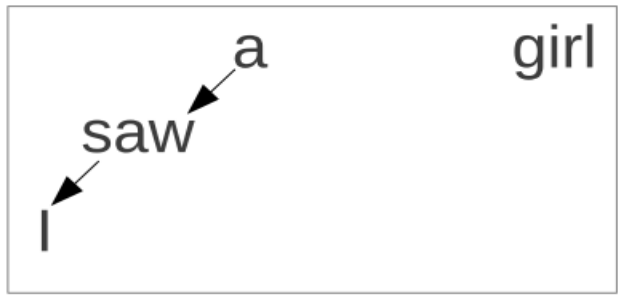
- Diberikan sebuah *state* (*stack* dan *queue* kata) sebagai berikut:



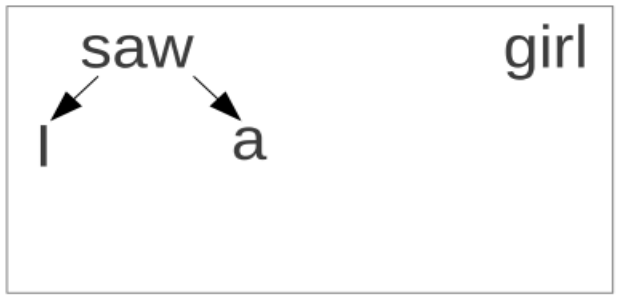
- Aksi mana yang akan dipilih?

shift

arc left



arc right



Graph-based dependency parsing

- Mencari pohon *parse* dengan skor paling maksimum
- Skor yang dimaksud secara umum adalah skor sebuah sub-pohon dari keseluruhan pohon
- Pendekatan *edge-factor* : penentuan skor berdasarkan skor sisi pohon
  - overall_score(t) = $\sum_{e \epsilon t}$ score(e)
  - di mana t adalah sebuah pohon/*tree*(t) dan e adalah sisi-sisi/*edges* dari pohon tersebut

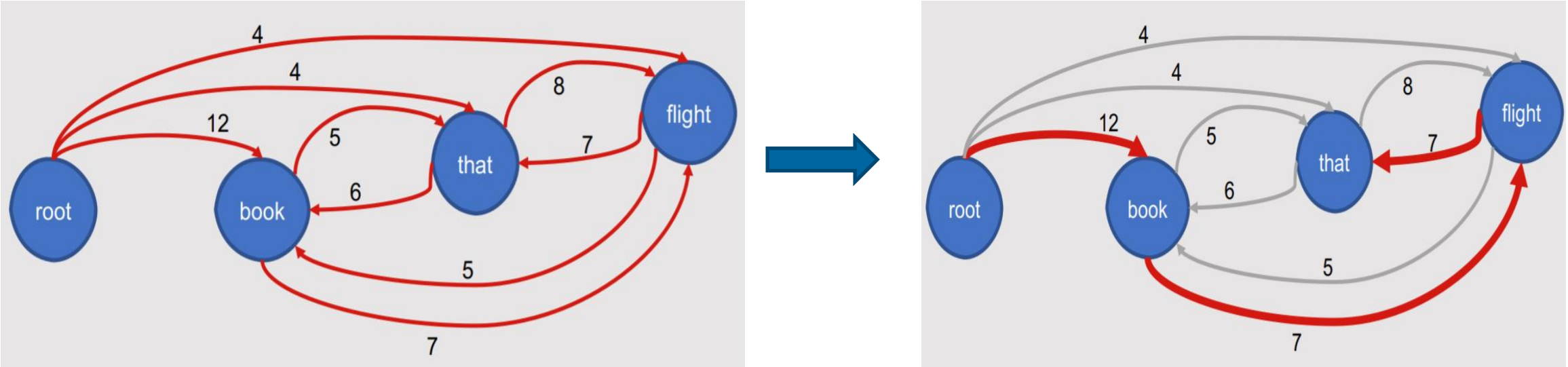
Graph-based vs transition-based dependency parsing

- Metode *transition-based* lebih baik kinerjanya pada kalimat yang lebih pendek.
  - Disebabkan oleh pendekatan Greedy pada *transition-based parsing*, sehingga lebih cenderung menemukan solusi maksimum lokal
  - Metode *graph-based* memberikan skor untuk keseluruhan pohon, sehingga dapat mengatasi kelemahan tersebut

Maximum Spanning Tree (MST)

- Diberikan sebuah kalimat masukan, akan dibentuk sebuah graf berarah, berbobot, dan terhubung lengkap (*fully-connected*)
  - Simpul adalah kata pada kalimat masukan
  - Sisi berarah mewakili semua kemungkinan relasi *head-dependent* antar kata
  - Bobot menunjukkan skor untuk tiap kemungkinan relasi *head-dependent*, diperoleh berdasarkan proses *supervised machine learning*
- *Maximum spanning tree* dari graf tersebut merepresentasikan pohon *parse* yang paling sesuai untuk kalimat tersebut, diputuskan berdasarkan bobot

Contoh *Maximum Spanning Tree*



Intuisi *Maximum Spanning Tree* (MST)

- Tiap simpul hanya dapat mempunyai 1 busur masuk
- Nilai absolut untuk skor sisi tidak penting, yang penting adalah bobot relatif sisi yang masuk ke sebuah simpul

Kriteria *spanning tree*:

- Diberikan sebuah graf terhubung lengkap G=(V,E), sebuah sub-graf T=(V,F) adalah *spanning tree* bila:
  - Tidak terdapat *cycle*
  - Setiap simpul kecuali ROOT mempunyai TEPAT 1 busur masuk

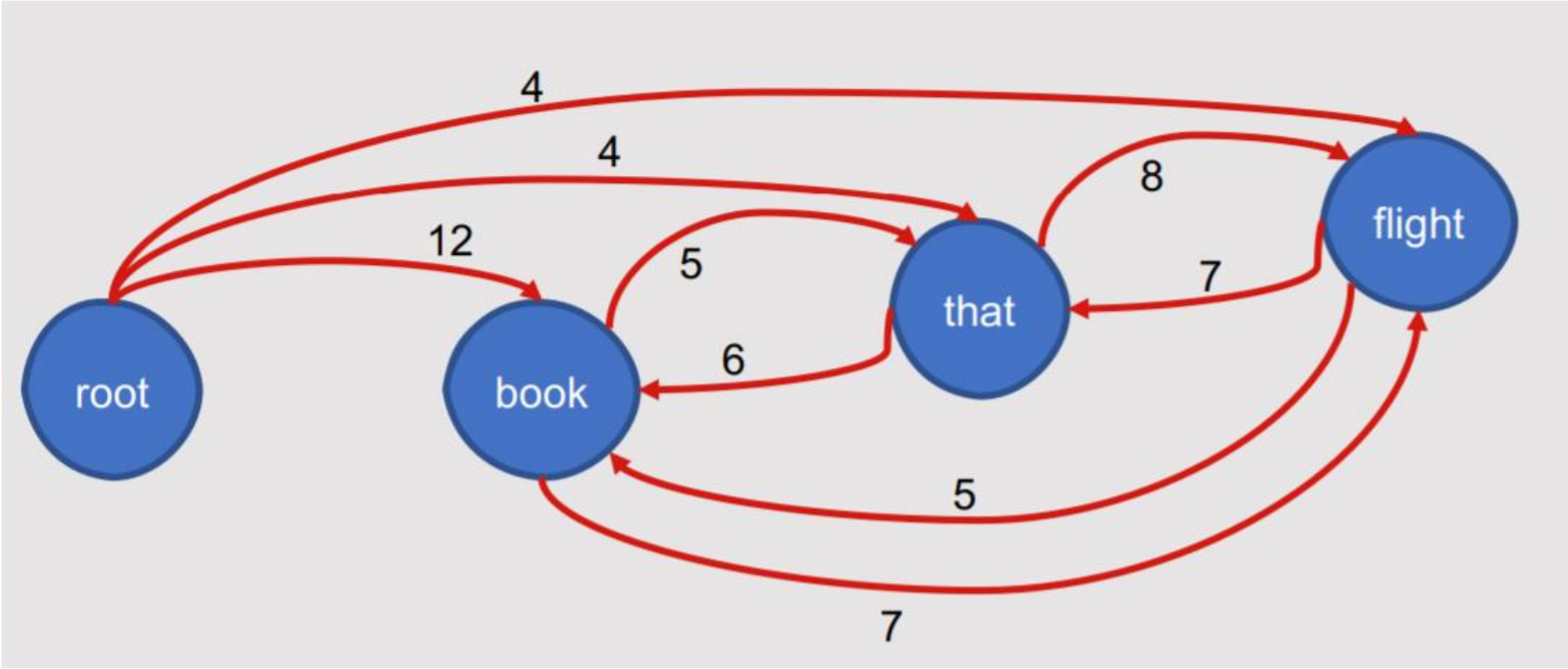
Bagaimana menghasilkan MST?

- Jika proses pemilihan secara Greedy menghasilkan sebuah *spanning tree*, maka pohon tersebut adalah MST
- Akan tetapi, proses pemilihan secara Greedy tersebut dapat memilih sisi yang akan menghasilkan *cycle*
- Jika *cycle* terbentuk, maka sebuah graf lain dapat dibentuk dengan menghancurkan *cycle* menjadi simpul baru, dengan sisi yang sebelumnya masuk atau keluar ke *cycle* akan masuk ke atau keluar dari simpul baru
- Proses pemilihan secara Greedy akan diulang secara rekursif hingga terbentuk sebuah MST

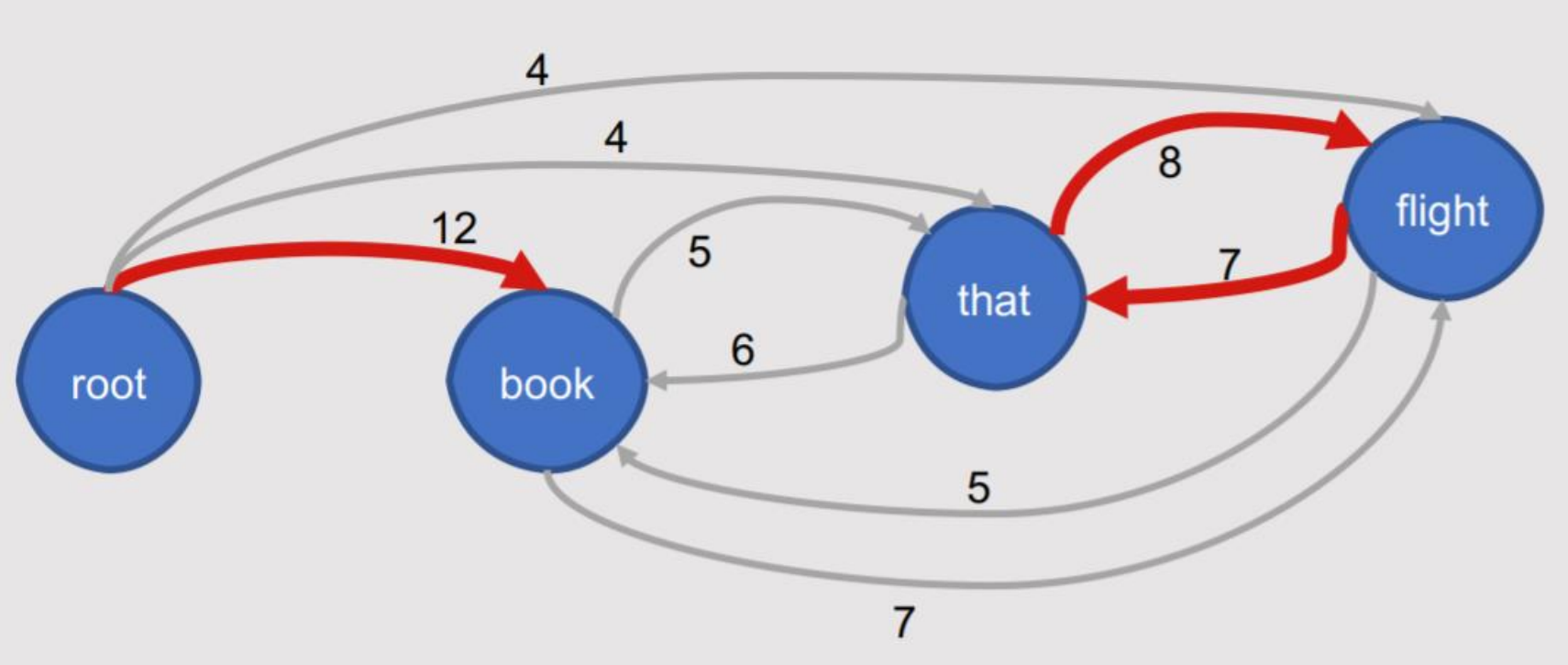
Algoritma MST

```
function MAXSPANNINGTREE(G=(V,E), root, score) returns spanning tree

    F ← []
    T' ← []
    score' ← []
    for each v ∈ V do
        bestInEdge ← argmax_{e=(u,v)∈ E} score[e]
        F ← F ∪ bestInEdge
        for each e=(u,v) ∈ E do
            score'[e] ← score[e] − score[bestInEdge]

        if T=(V,F) is a spanning tree then return it
        else
            C ← a cycle in F
            G' ← CONTRACT(G, C)
            T' ← MAXSPANNINGTREE(G', root, score')
            T ← EXPAND(T', C)
            return T

function CONTRACT(G, C) returns contracted graph

function EXPAND(T, C) returns expanded graph
```

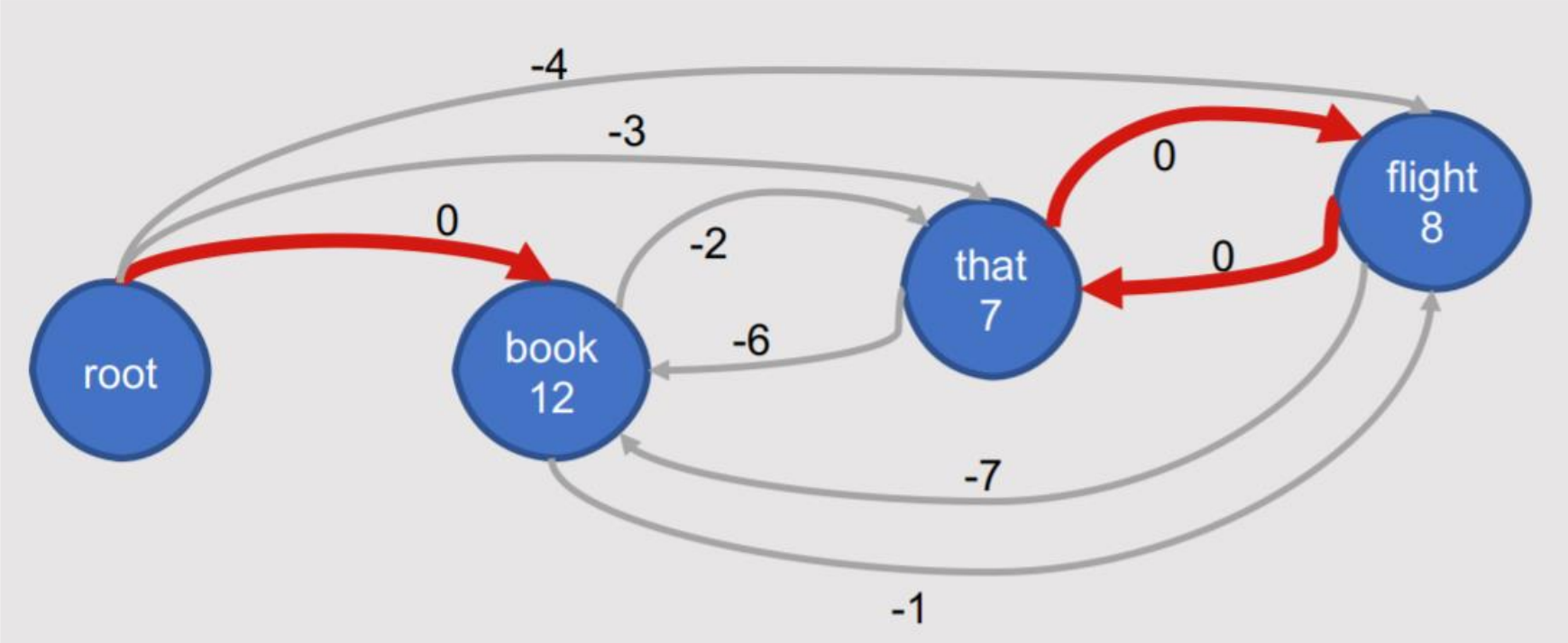
Ilustrasi proses untuk mendapatkan MST (1)



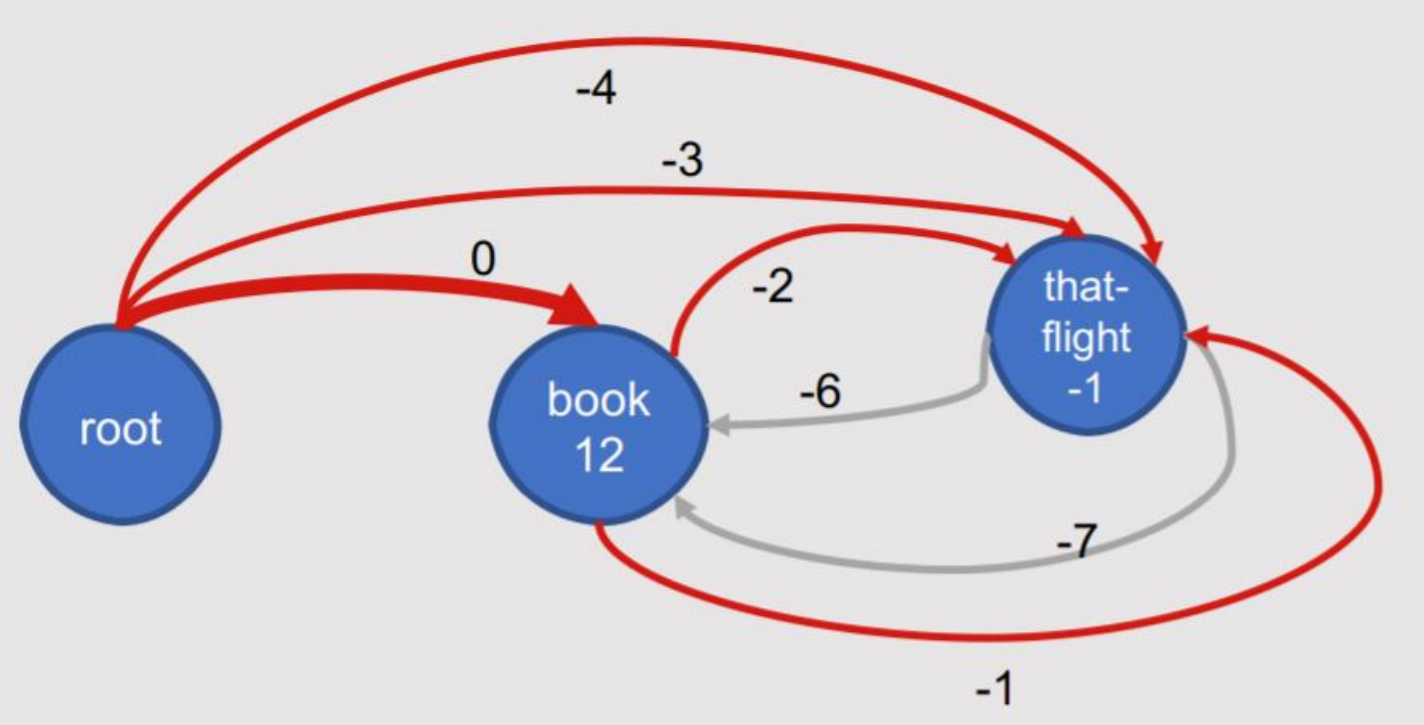
Ilustrasi proses untuk mendapatkan MST (2)



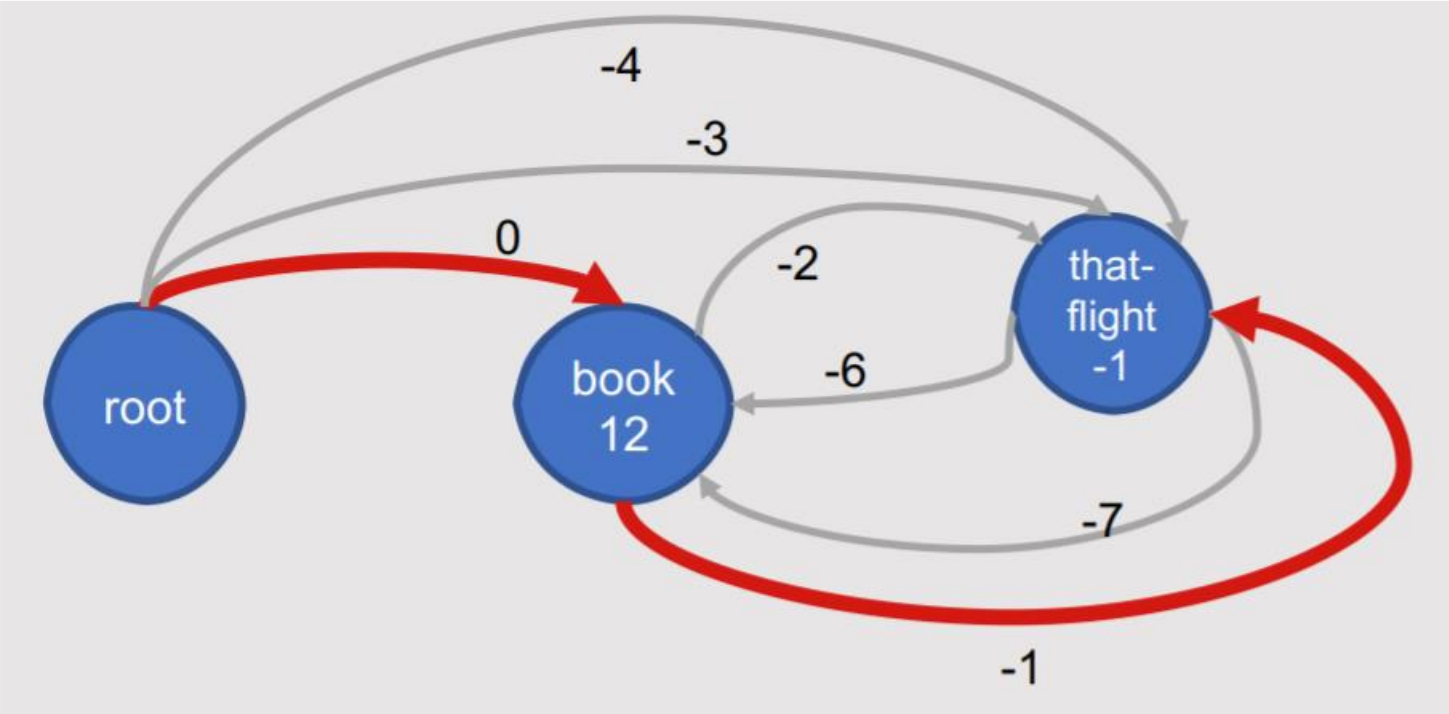
Ilustrasi proses untuk mendapatkan MST (3)



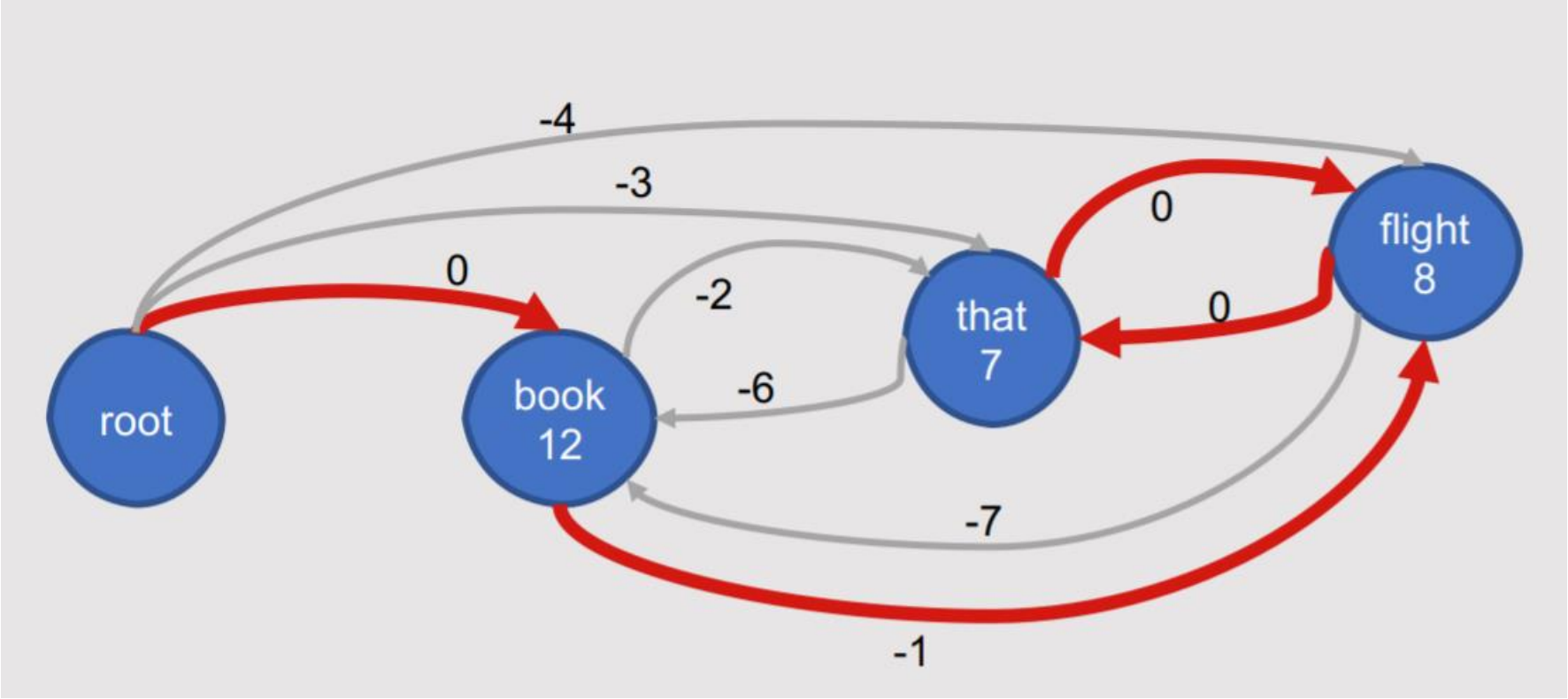
Ilustrasi proses untuk mendapatkan MST (4)



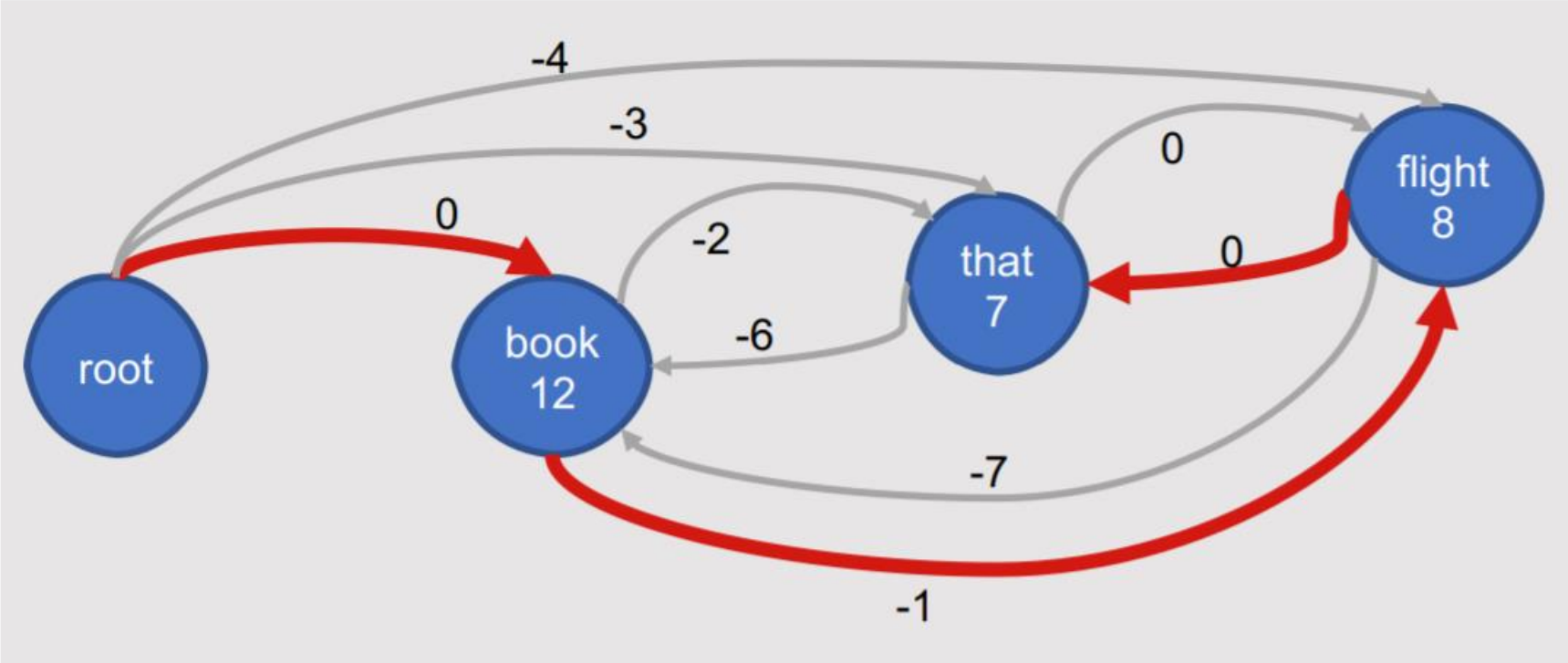
Ilustrasi proses untuk mendapatkan MST (5)



Ilustrasi proses untuk mendapatkan MST (6)



Ilustrasi proses untuk mendapatkan MST (7)



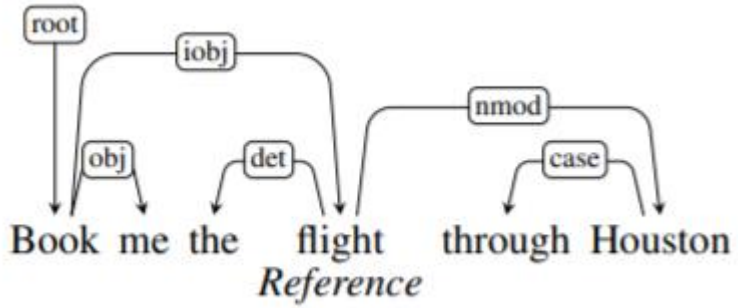
Bagaimana melatih model untuk memprediksi bobot sisi?

- Serupa dengan pelatihan untuk mendapatkan oracle di *transition-based parsing*
- Fitur yang umum digunakan:
  - Kata, lemma, PoS
  - Fitur yang sesuai untuk konteks (kata-kata sebelum dan sesudah)
  - *Word embeddings*
  - Tipe relasi dependensi
  - Arah relasi dependensi
  - Jarak dari *head* ke *dependent*

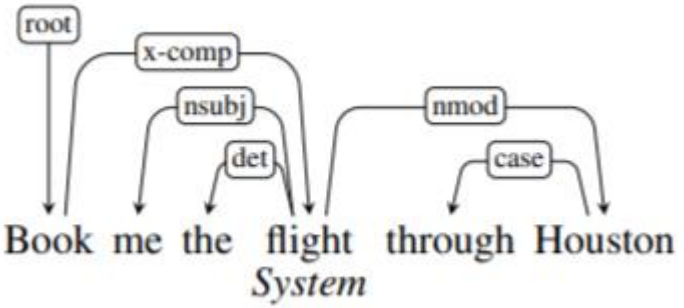
Evaluasi *dependency parsing*

- Metriks:
  - *Labeled attachment score* (LAS): *attachment* yang benar antara sebuah kata dengan *head*-nya berikut informasi relasi dependensi
  - *Unlabeled attachment score* (UAS): *attachment* yang benar antara sebuah kata dengan *head*-nya tanpa informasi relasi dependensi

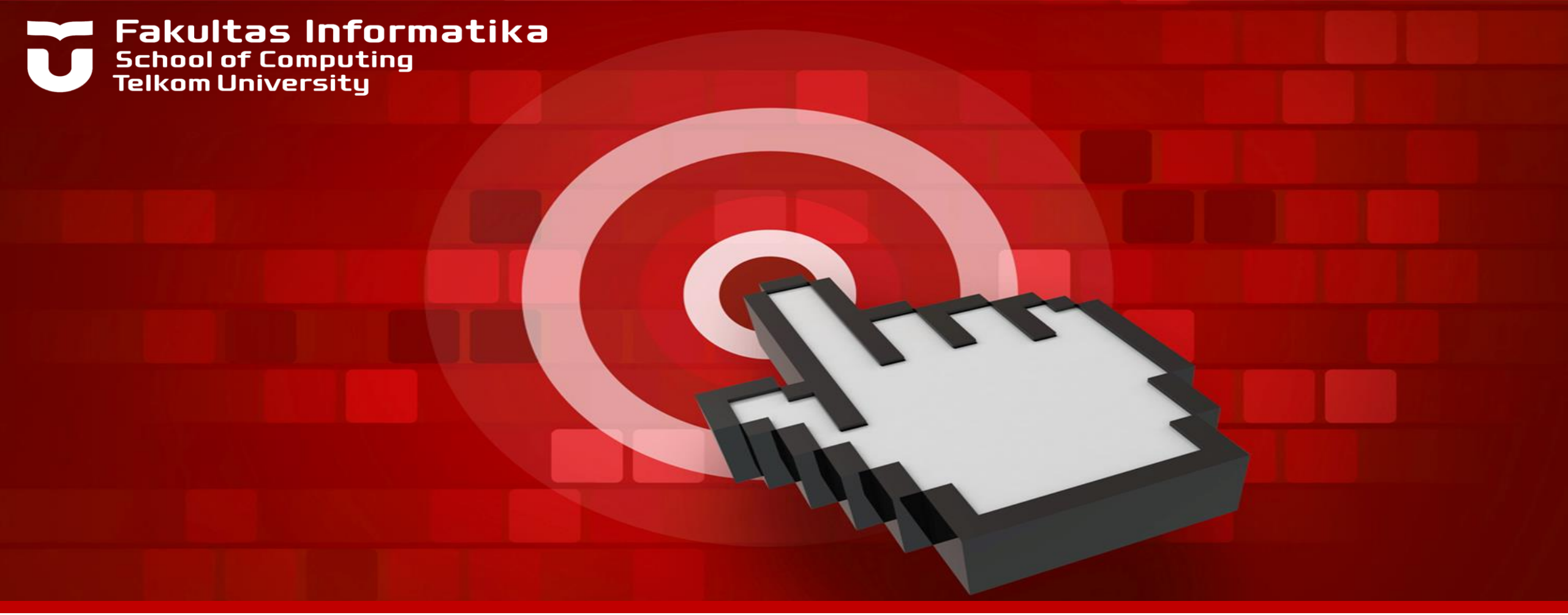
Coba hitung berapa LAS dan UAS contoh hasil *parsing* sistem berikut:



Reference



System



THANK YOU