IKI 20100: Struktur Data & Algoritma

Graph

Ruli Manurung & Ade Azurat
(acknowledgments: Denny, Suryana Setiawan)

Fasilkom UI

Materi

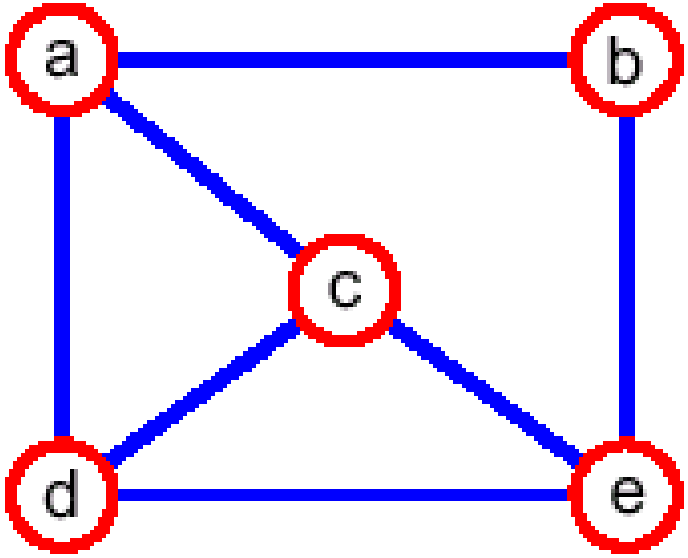
- Motivasi
- Definisi dan Istilah
- Representasi Graph
- Algoritma mencari shortest path
- Topological Sort
- Minimum spanning tree
  - Prim's Algoritma
  - Kruskal's Algoritma

Penggunaan Graph

- Jaringan
- Peta
  - Mencari jalur terpendek
- Penjadwalan (Perencanaan Proyek)

Definisi

- Sebuah graph G = (V, E) terdiri dari:
  - V: kumpulan simpul (*vertices/nodes)*
  - E: kumpulan *sisi/busur (edge)* yang menghubungkan simpul-simpul.
- Sebuah sisi e = (*a*, *b*) memiliki informasi dua simpul yang dihubungkannya.



V= {a,b,c,d,e}

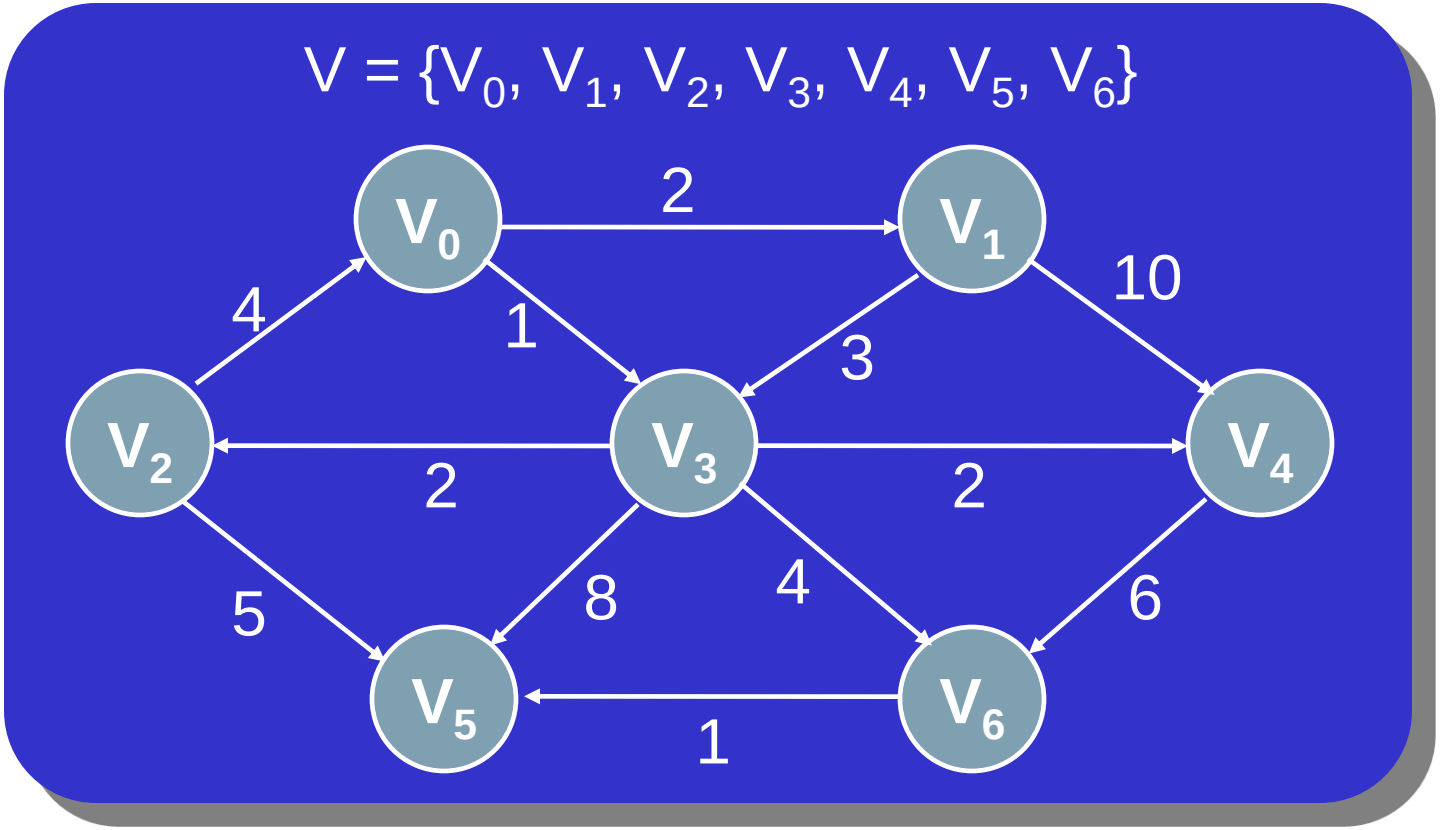
E=
{(a,b),(a,c),(a,d),
(b,e),(c,d),(c,e),
(d,e)}

Istilah

- **undirected graph**
- **directed graph**
- **adjacent vertices**: adalah simpul-simpul yang dihubungkan oleh sebuah sisi *(edge)*
- **degree** (of a vertex): adalah jumlah simpul lain yang terhubung langsung melalui sebuah sisi.
  - Untuk kategori directed graph
    - in-degree
    - out-degree

Weighted Graph

- **weighted graph**: setiap sisi memiliki **bobot/nilai**.

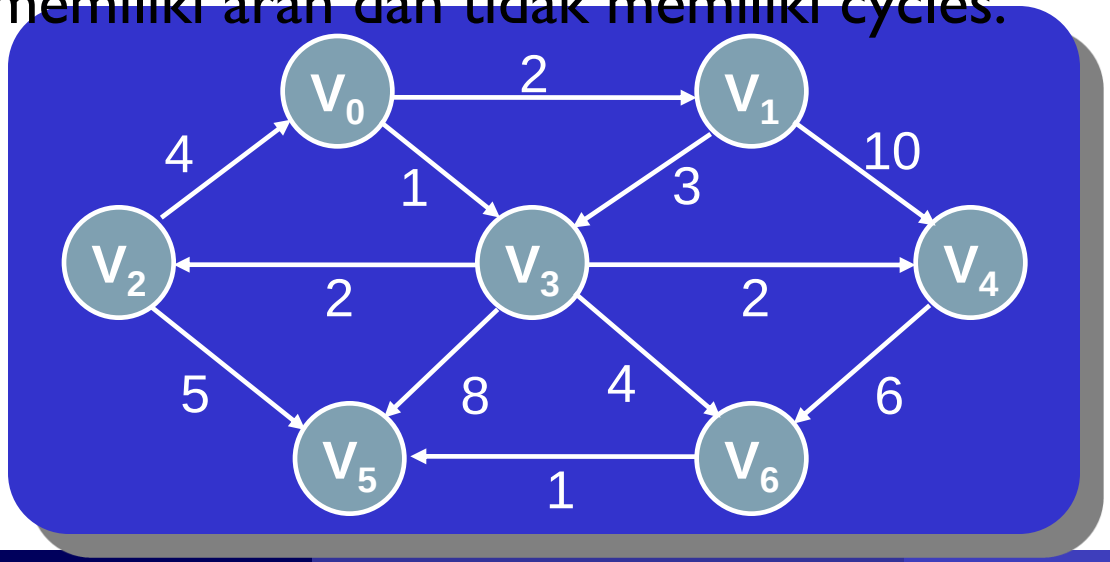


$(V_0, V_1, 2), (V_0, V_3, 1), (V_1, V_3, 3), (V_1, V_4, 10)$
$(V_3, V_4, 2), (V_3, V_6, 4), (V_3, V_5, 8), (V_3, V_2, 2)$
$(V_2, V_0, 4), (V_2, V_5, 5), (V_4, V_6, 6), (V_6, V_5, 1)$

- $|V| = 7; |E| = 12$

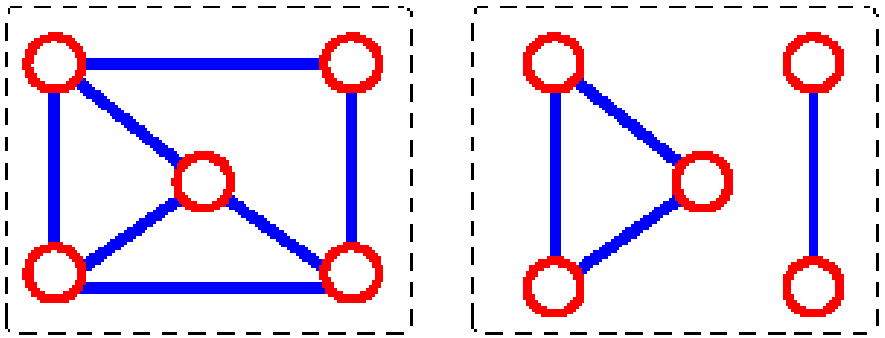
Istilah

- **Jalur/path**: urutan simpul (vertices) v_1, v_2 ,. . .v_k sedemikian sehingga simpul yang berurutan v_i dan v_{i+1} adalah simpul yang terhubung.
- **simple path**: tidak ada simpul yang diulang.
- **cycle**: simple path, dengan catatan simpul awal sama dengan simpul akhir
- **DAG** (**Directed Acyclic Graph**): Graph dengan busur/sisi yang memiliki arah dan tidak memiliki cycles.



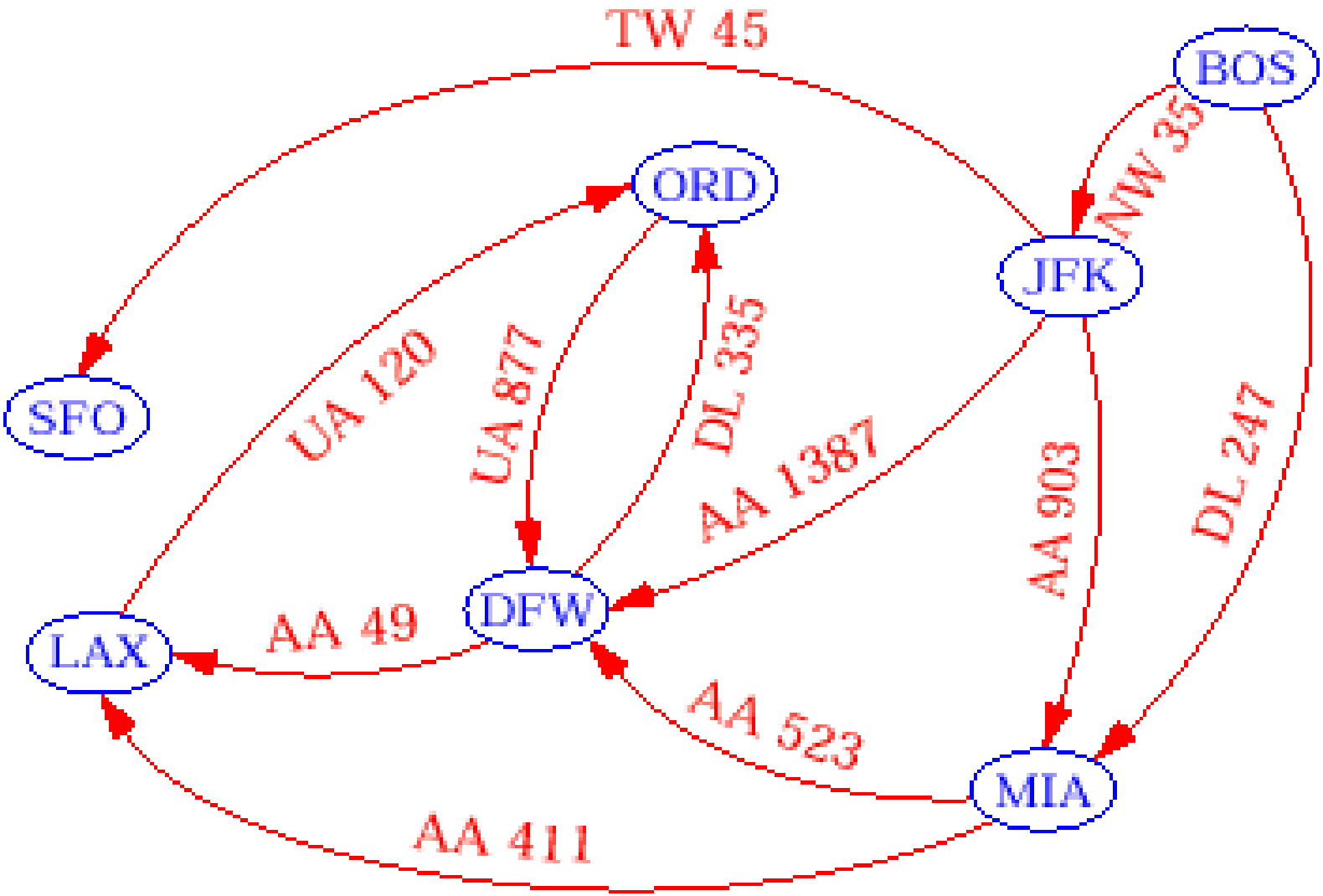
Istilah

- *connected graph*: tiap simpul terhubung dengan simpul lain



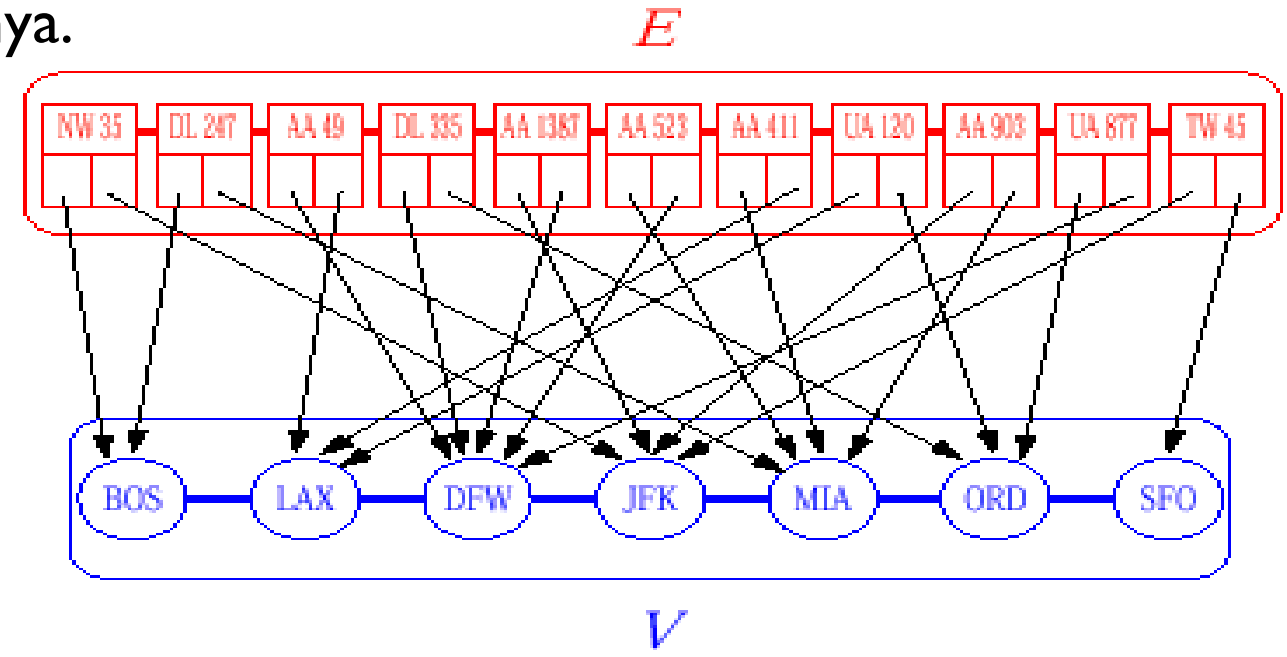
- *subgraph*: bagian simpul dan sisi yang dapat membentuk graph

Representasi



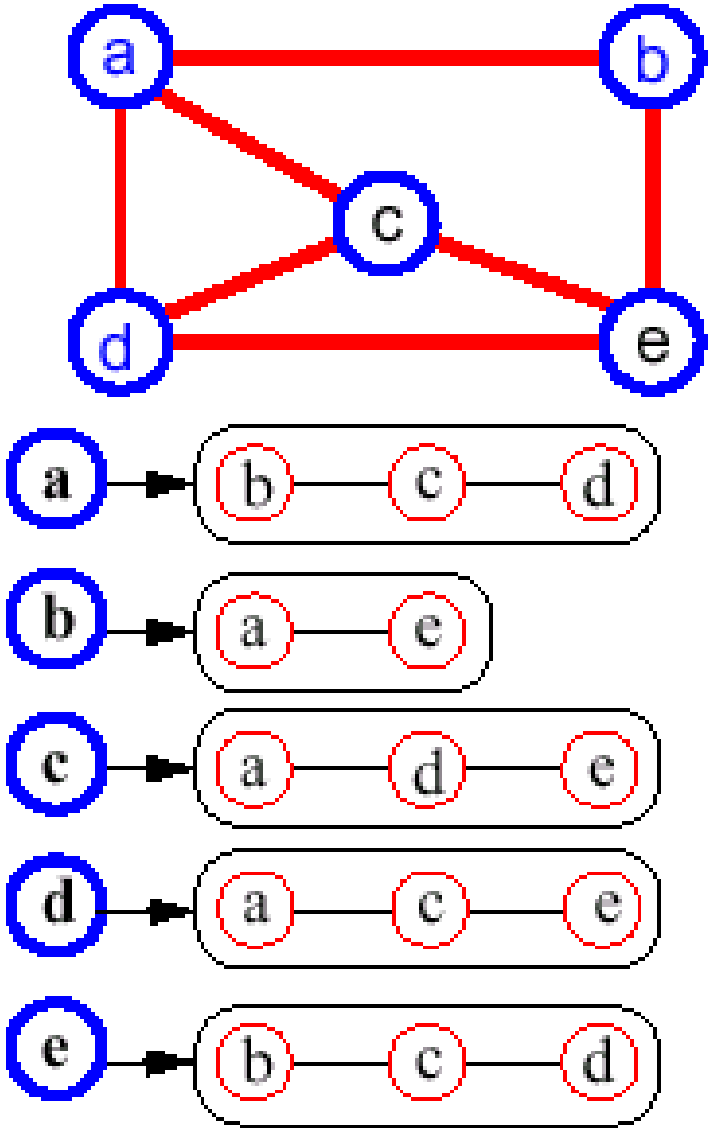
Representasi: *Edge List*

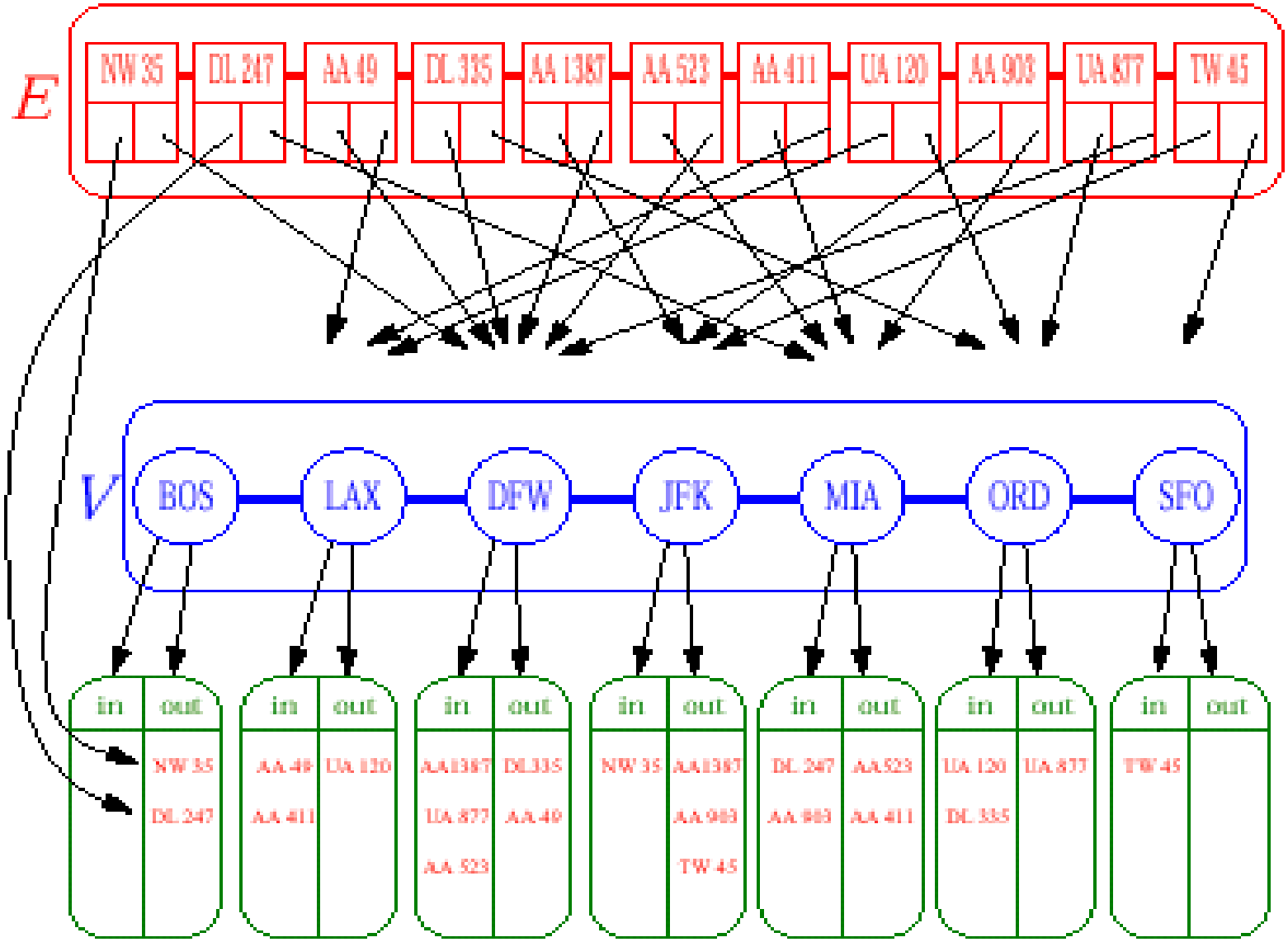
- Struktur ***edge list*** hanya menyimpan simpul dan sisi dalam sebuah list yang tidak terurut.
- Pada tiap sisi disimpan informasi simpul yang terhubung oleh sisi tersebut.
- mudah diimplementasikan.
- Tidak efisien dalam keperluan mencari sisi bila diketahui simpulnya.



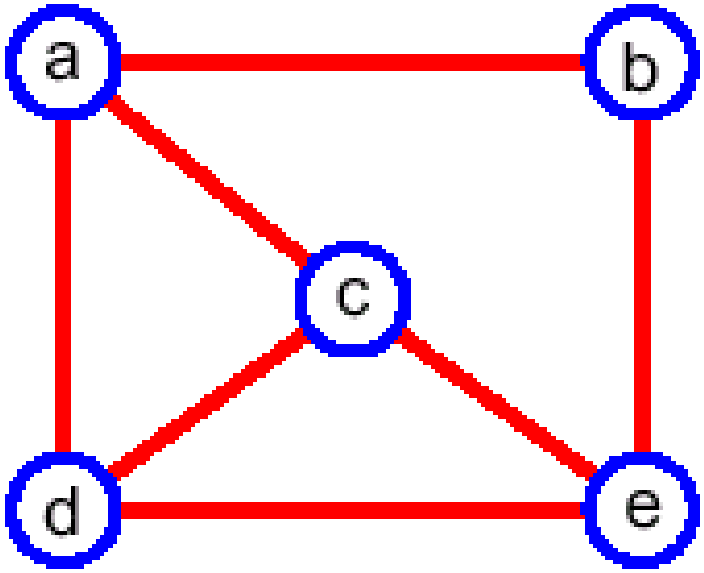
Representasi: *Adjancency List* (traditional)

- **Adjacency list** dari sebuah *vertex* ***v*** adalah sekumpulan *vertex* yang terhubung dengan ***v***
- Merepresentasikan graph, dengan menyimpan daftar *adjacency lists* dari seluruh *vertex*.
- struktur **adjacency list** dapat digabungkan dengan struktur edge list.




E
NW 35
DL 247
AA 49
DL 335
AA 1387
AA 523
AA 411
UA 120
AA 903
UA 877
TW 45
V
BOS
LAX
DFW
JFK
MIA
ORD
SFO
in
out
NW 35
DL 247
AA 49
AA 411
UA 120
AA1387
UA 877
AA 523
DL335
AA 49
NW 35
AA1387
AA 903
TW 45
DL 247
AA 903
AA523
AA 411
UA 120
DL 335
UA 877
TW 45

Representasi: Adjacency Matrix (traditional)



	a	b	c	d	e
a	F	T	T	T	F
b	T	F	F	F	T
c	T	F	F	T	T
d	T	F	T	F	T
e	F	T	T	T	F

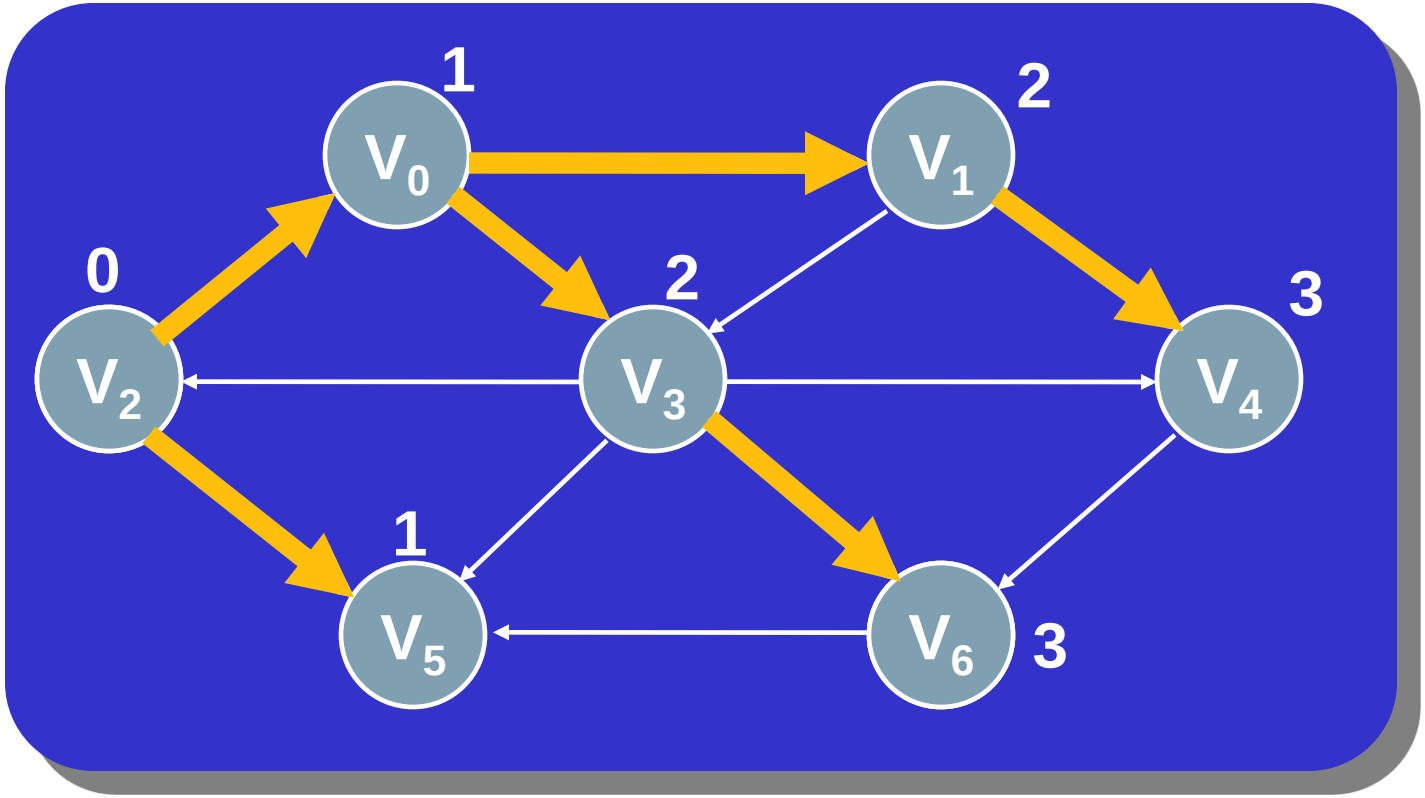
- matrix M dengan eleman setiap pasang simpul
  - M[i,j] = *true* artinya ada sisi dari simpul (i,j) di graph.
  - M[i,j] = *false* artinya tidak ada sisi dari simpul (i,j) di graph.

	0	1	2	3	4	5	6
0	Ø	Ø	NW 35	Ø	DL 247	Ø	Ø
1	Ø	Ø	Ø	AA 49	Ø	DL 335	Ø
2	Ø	AA 1387	Ø	Ø	AA 903	Ø	TW 45
3	Ø	Ø	Ø	Ø	Ø	UA 120	Ø
4	Ø	AA 523	Ø	AA 411	Ø	Ø	Ø
5	Ø	UA 877	Ø	Ø	Ø	Ø	Ø
6	Ø	Ø	Ø	Ø	Ø	Ø	Ø

BOS 0 DFW 1 JFK 2 LAX 3 MIA 4 ORD 5 SFO 6

Shortest Path:

- Vertex awal: V_2
- Bila sisi tidak memiliki bobot, gunakan algoritma BFS (**Breadth First Search**).



Dijkstra's Algorithm

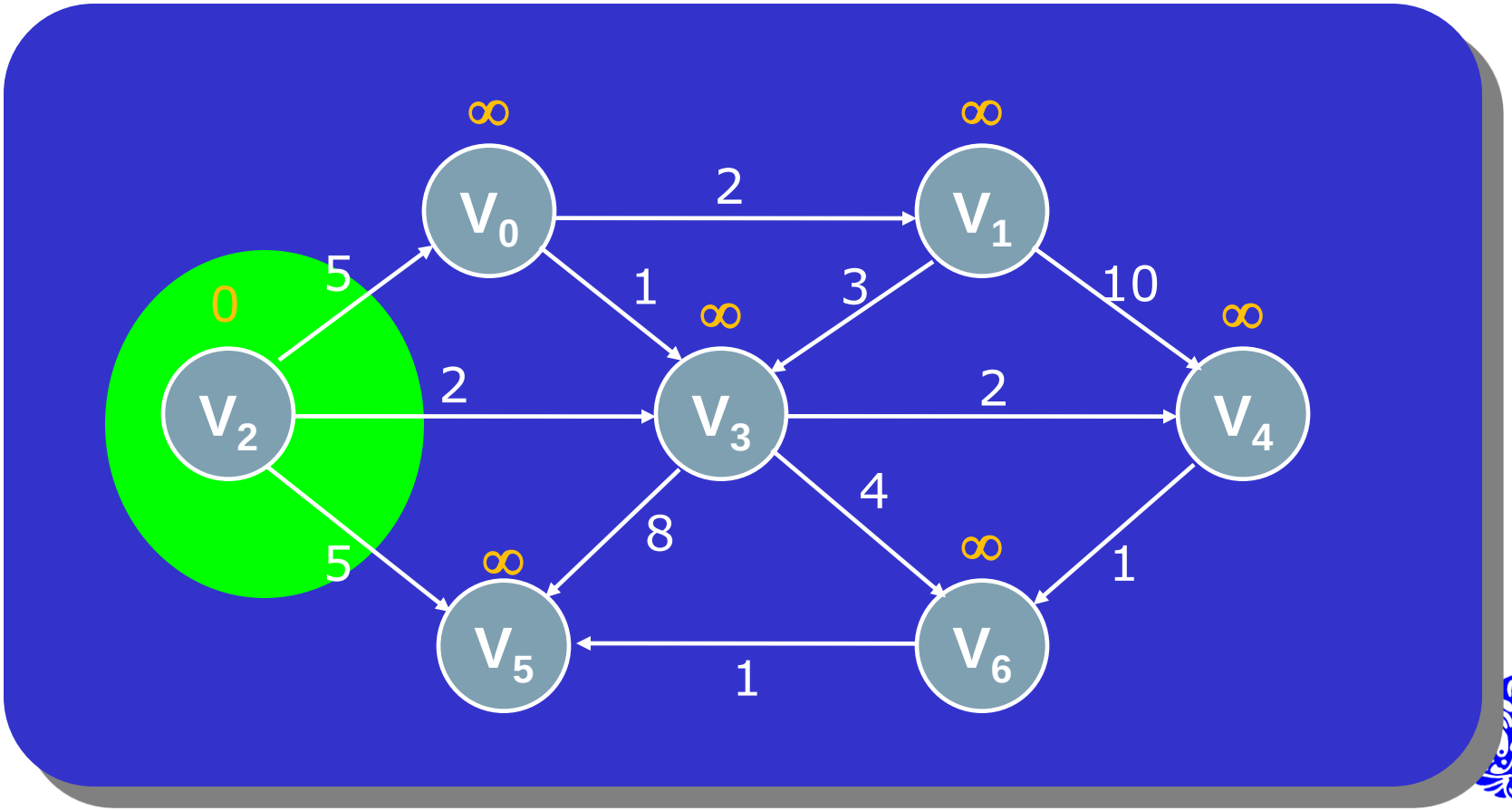
- Banyak masalah → weighted graph (mis: jaringan transport)
- Algoritma **Dijkstra** menghitung jarak tiap simpul dari simpul awal hingga akhirnya diketahui jarak terpendek simpul akhir yang diinginkan.
- Algoritma mengingat simpul mana saja yang telah dihitung jarak terpendeknya dan dinyatakan dalam kelompok hijau (pada literatur dinyatakan sebagai awan putih/white cloud).
- Untuk simpul yang baru sebagian dihitung jaraknya dan belum bisa dipastikan apakah itu jarak terpendek, dinyatakan dengan kelompok abu-abu.
- Untuk simpul yang sama sekali belum dihitung, dinyatakan dalam kelompok hitam.

Dijkstra's Algorithm

- Algoritma menggunakan label ***D[v]*** untuk menyimpan perkiraan jarak terpendek antara s dan v.
- Ketika sebuah simpul v ditambahkan kedalam kelompok aba-abu nilai D[v] sama dengan bobot antara s dan v.
- pada awalnya, nilai label D untuk setiap simpul adalah:
  - D[s] = 0
  - D[v] = ∞ untuk v $\neq$ s

Dijkstra's Algorithm: *stages*

- Awal: Tentukan simpul awal.



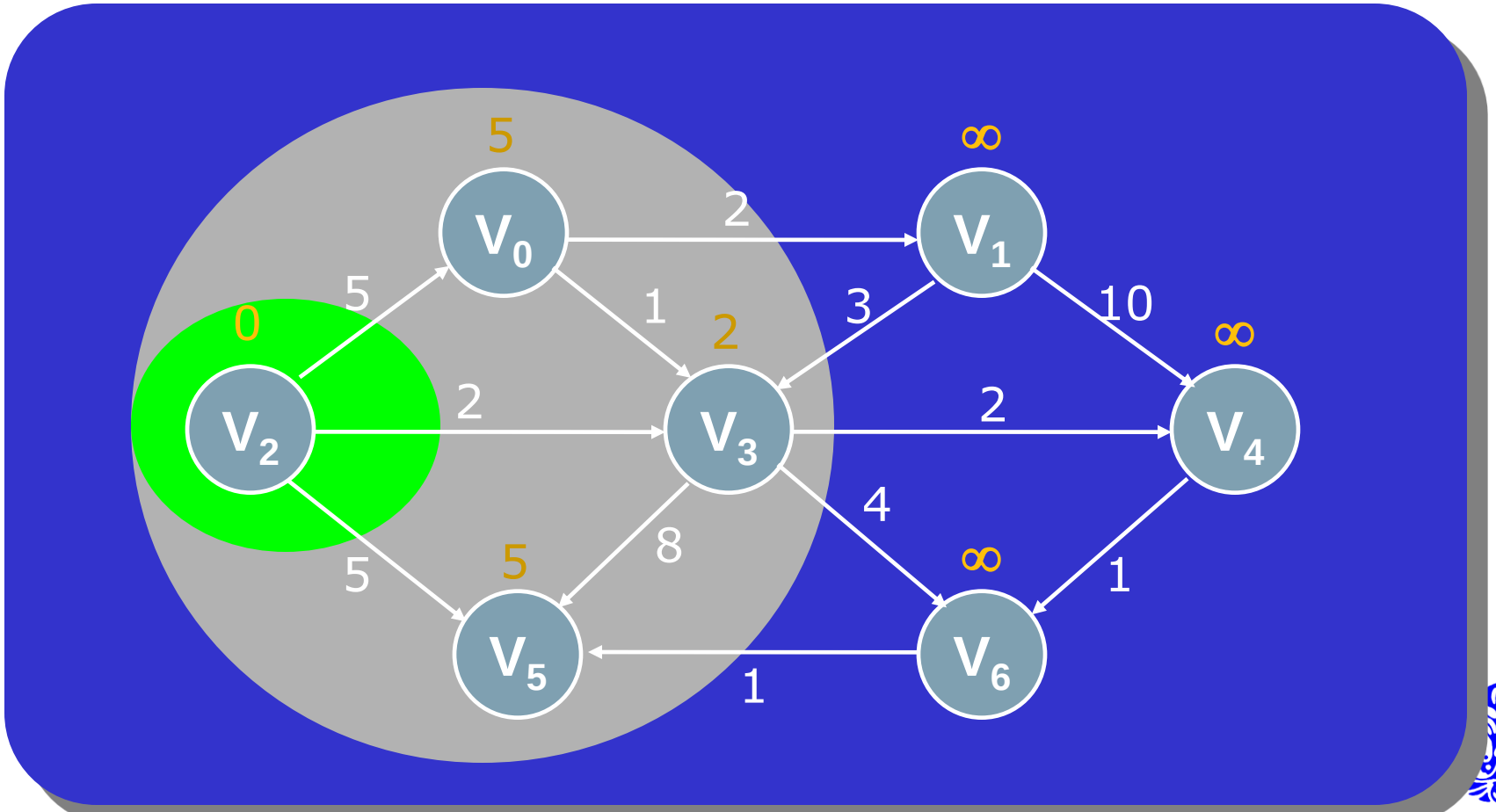
Expanding the White Cloud

- Setiap penambahan simpul, kita harus uji apakah jalur melalui *u* lebih baik.
- Misalkan **u** adalah sebuah simpul yang tidak berada di **kelompok hijau**, tapi sudah diketahui jarak terpendeknya dari s
  - tambahkan u ke dalam kelompok hijau
  - hitung jarak simpul lain dengan algoritma berikut:

```
Untuk tiap simpul z yang terhubung ke u
lakukan:
   jika z tidak di kelompok hijau maka
   if D[u] + bobot(u,z) < D[z] then
      D[z] = D[u] + bobot(u,z)
```

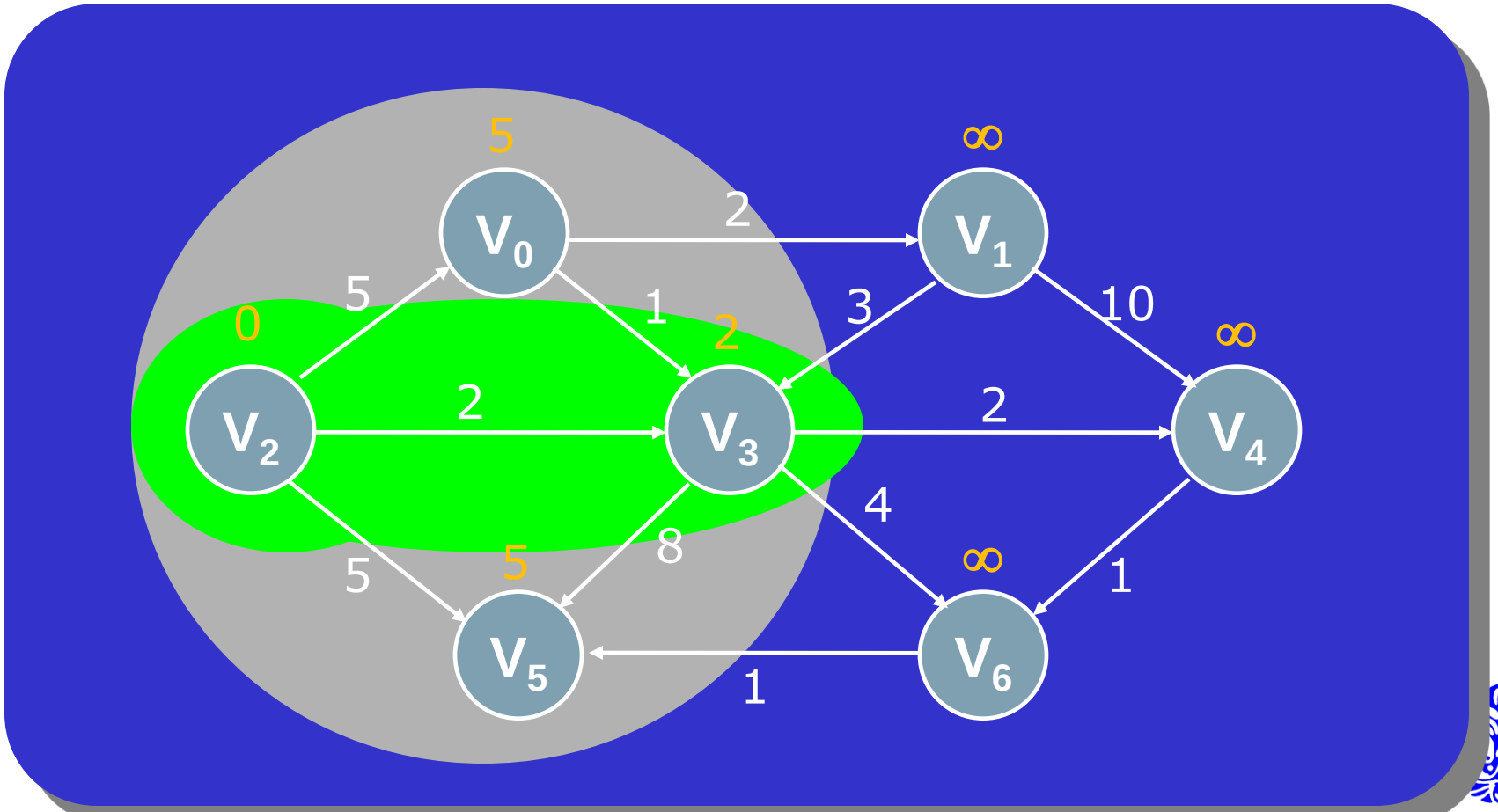
Dijkstra's Algorithm: *stages*

- setelah V_2 ditambahkan ke kelompok hijau, hitung $D[V_x]$ untuk setiap V_x yang terhubung ke V_2



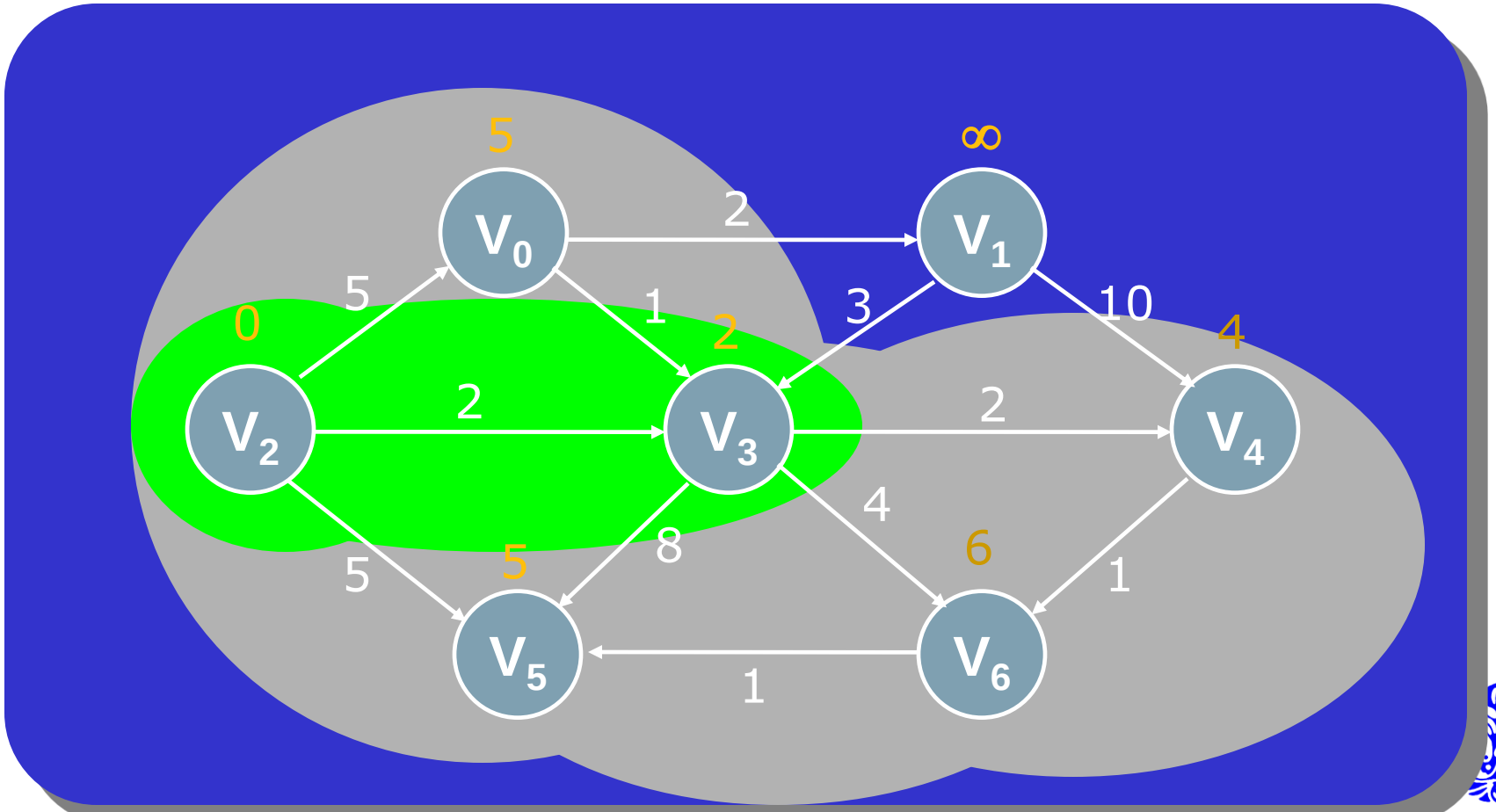
Dijkstra's Algorithm: stages

- tambahkan ke dalam kelompok hijau simpul pada kelompok abu-abu yang memiliki nilai D[V] minimum.
- Pada contoh adalah simpul V_3



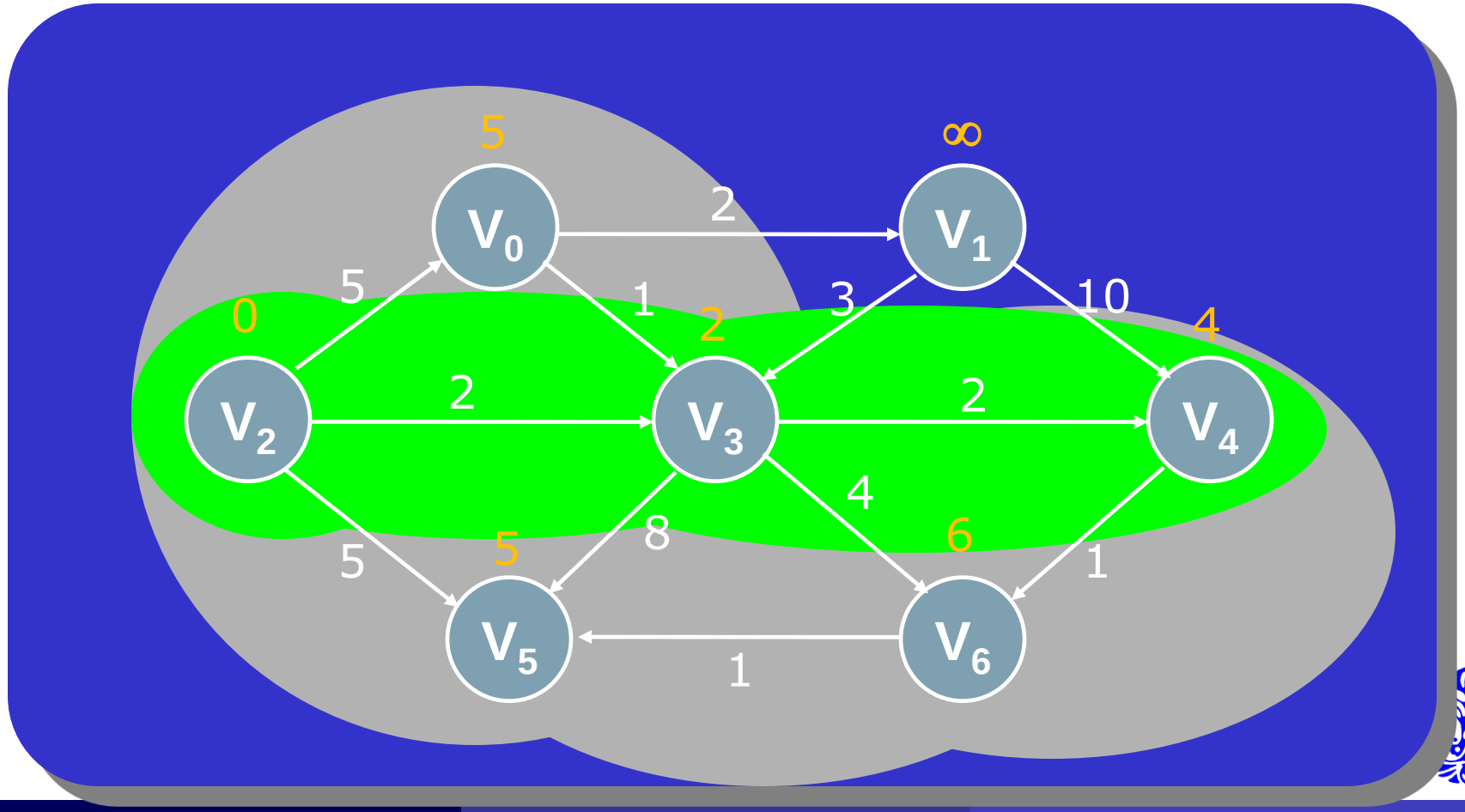
Dijkstra's Algorithm: stages

- setelah V_3 ditambahkan ke kelompok hijau, hitung $D[V_x]$ untuk setiap V_x yang terhubung dengan V_3. Simpul-simpul tersebut menjadi kelompok abu-abu.



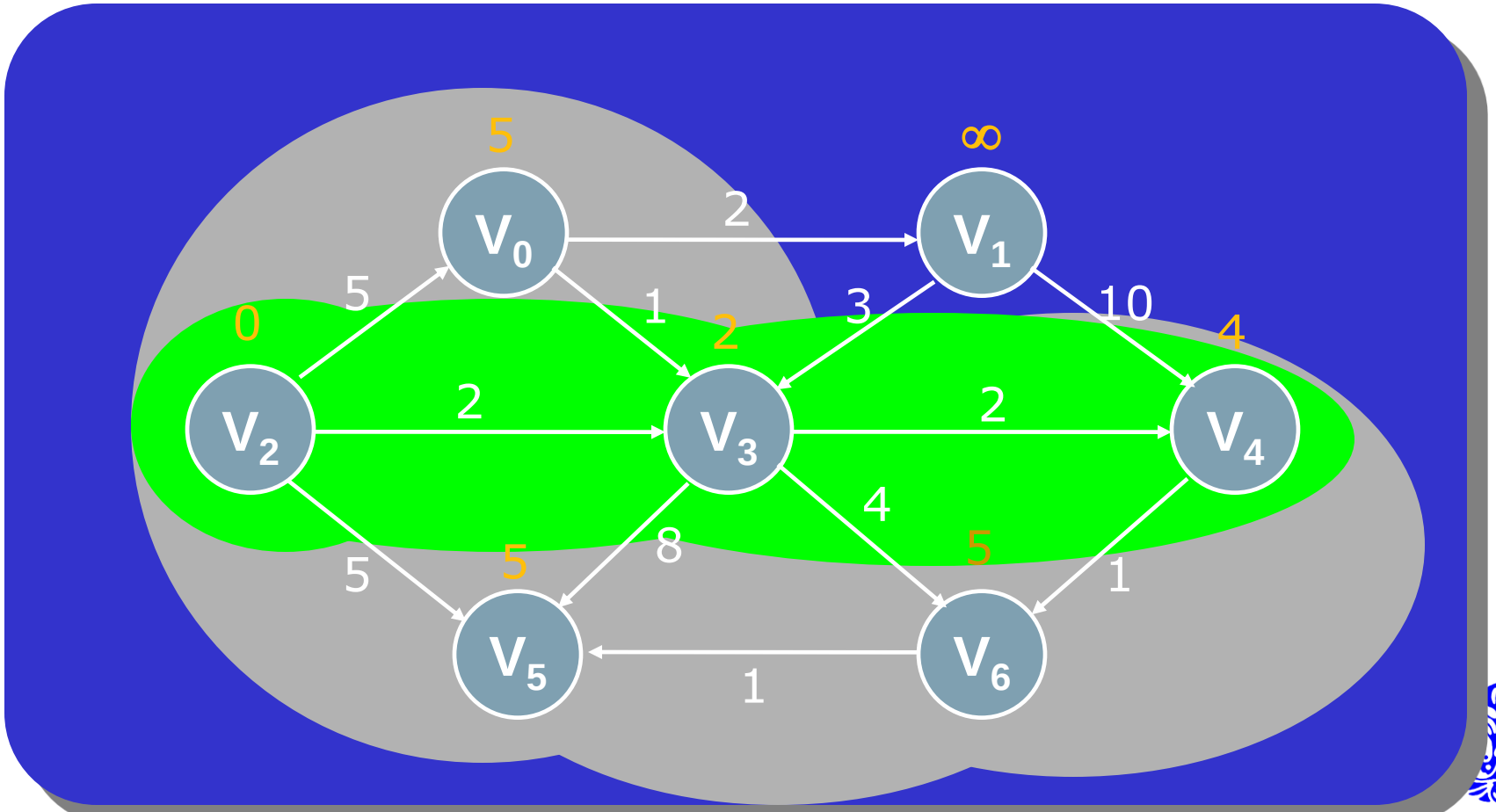
Dijkstra's Algorithm: stages

- pilih dari kelompok abu-abu, simpul yang memiliki nilai D[V] paling minimum dan tambahkan pada kelompok hijau.



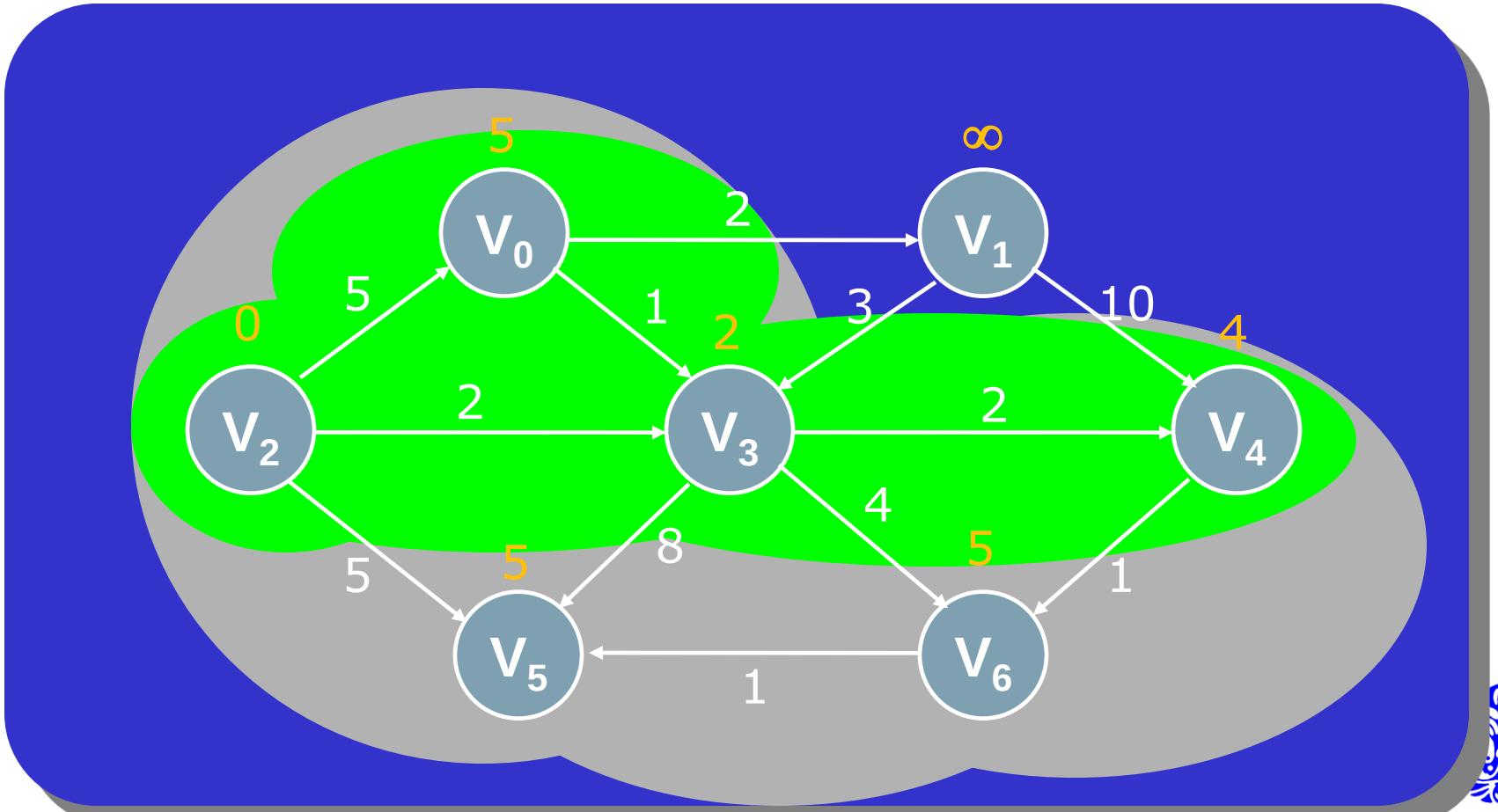
Dijkstra's Algorithm: stages

- setelah V_4 ditambahkan ke kelompok hijau, hitung $D[V_x]$ untuk setiap V_x yang terhubung dengan V_4. Simpul-simpul tersebut menjadi kelompok abu-abu.



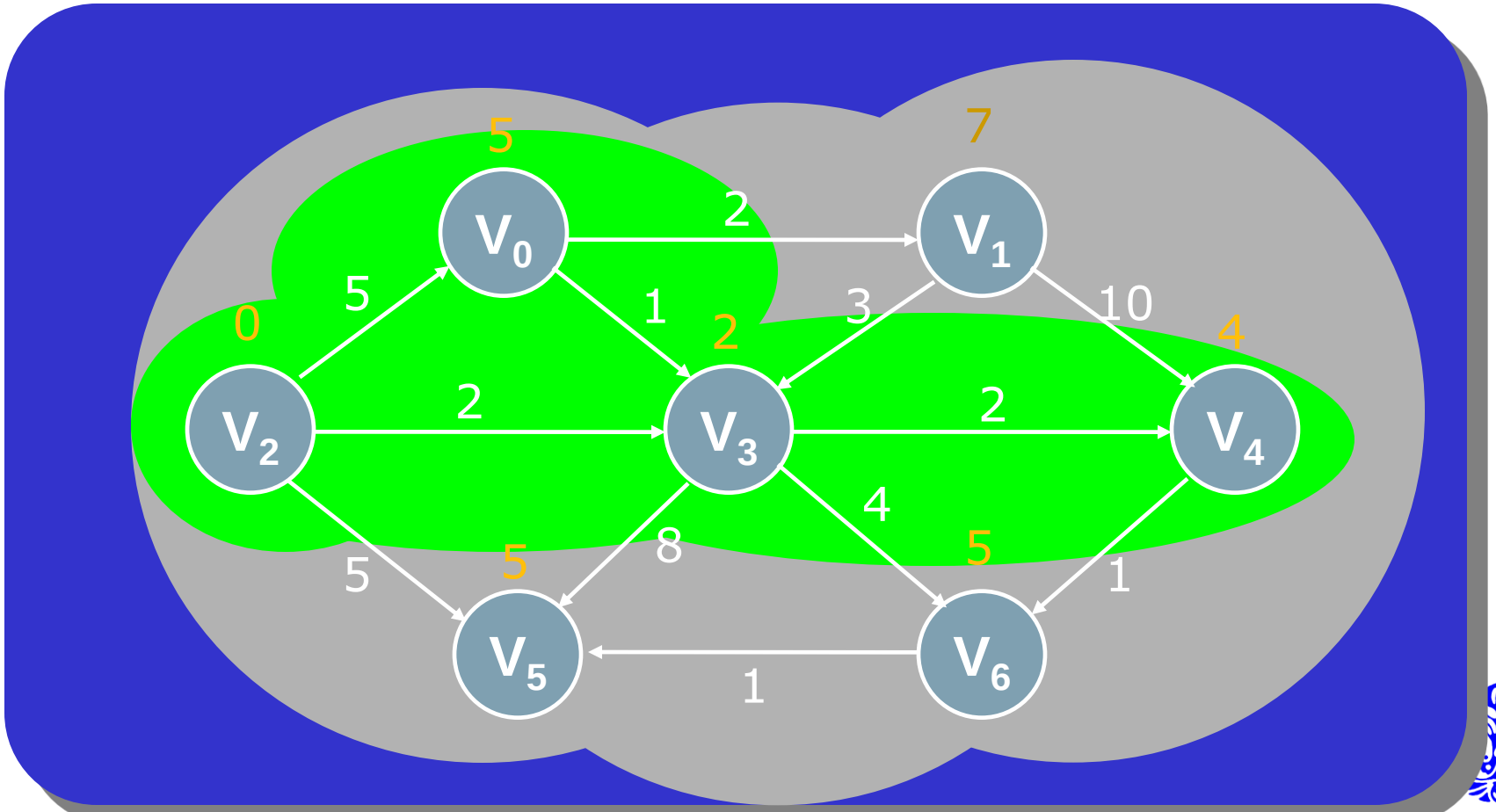
Dijkstra's Algorithm: stages

- pilih dari kelompok abu-abu, simpul yang memiliki nilai D[V] paling minimum dan tambahkan pada kelompok hijau.

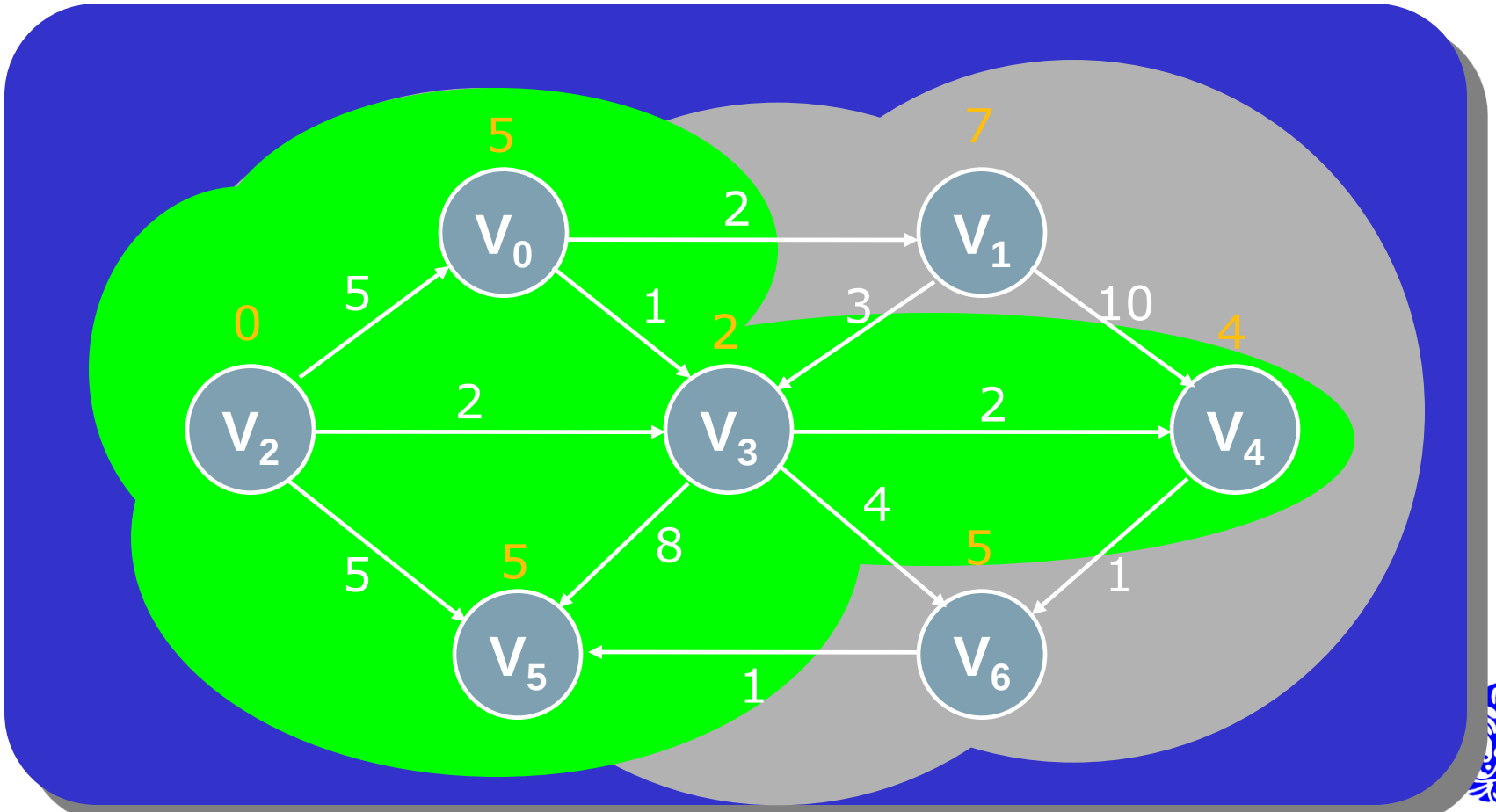


Dijkstra's Algorithm: stages

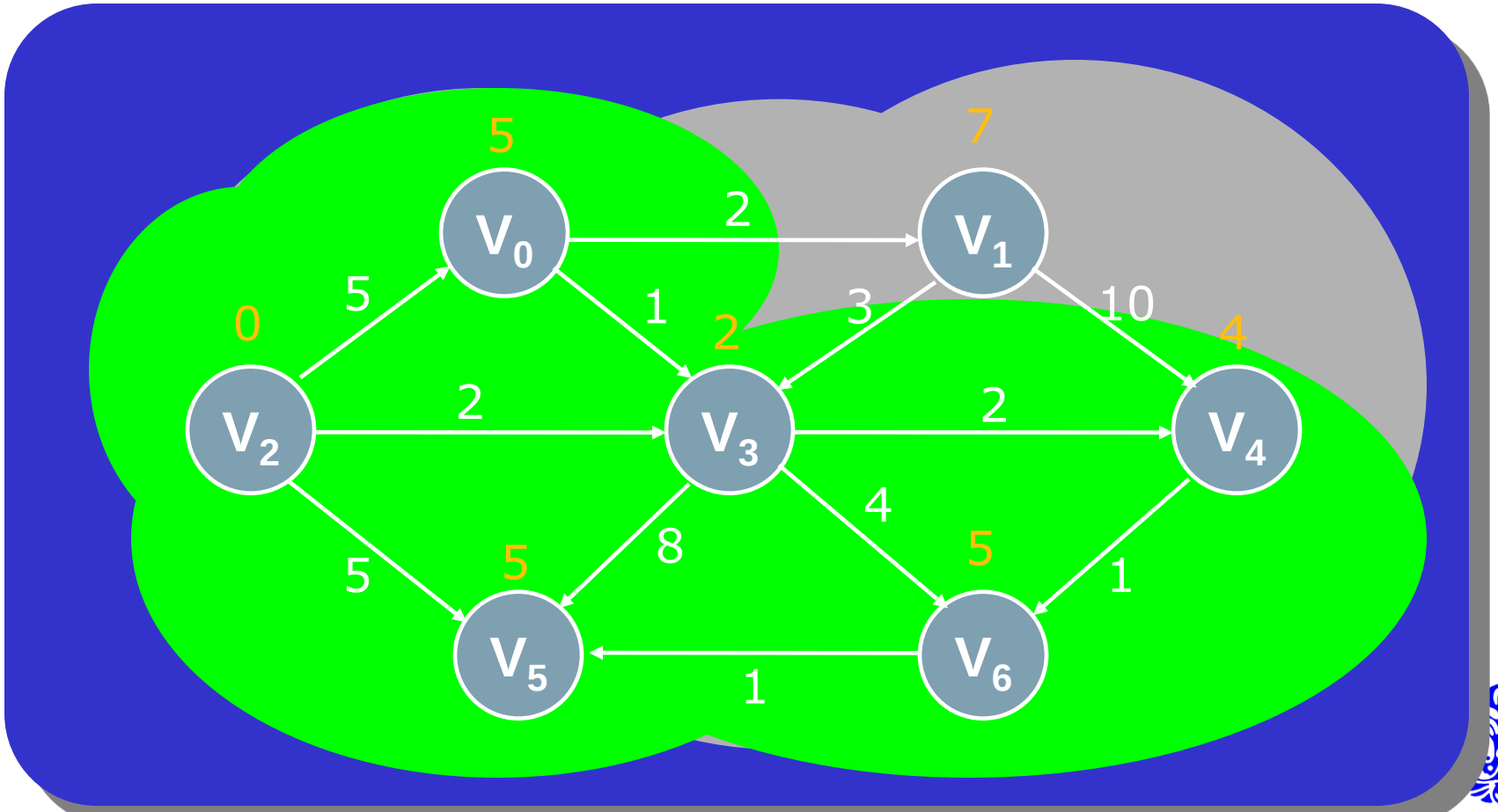
- setelah V_4 ditambahkan ke kelompok hijau, hitung $D[V_x]$ untuk setiap V_x yang terhubung dengan V_4. Simpul-simpul tersebut menjadi kelompok abu-abu.



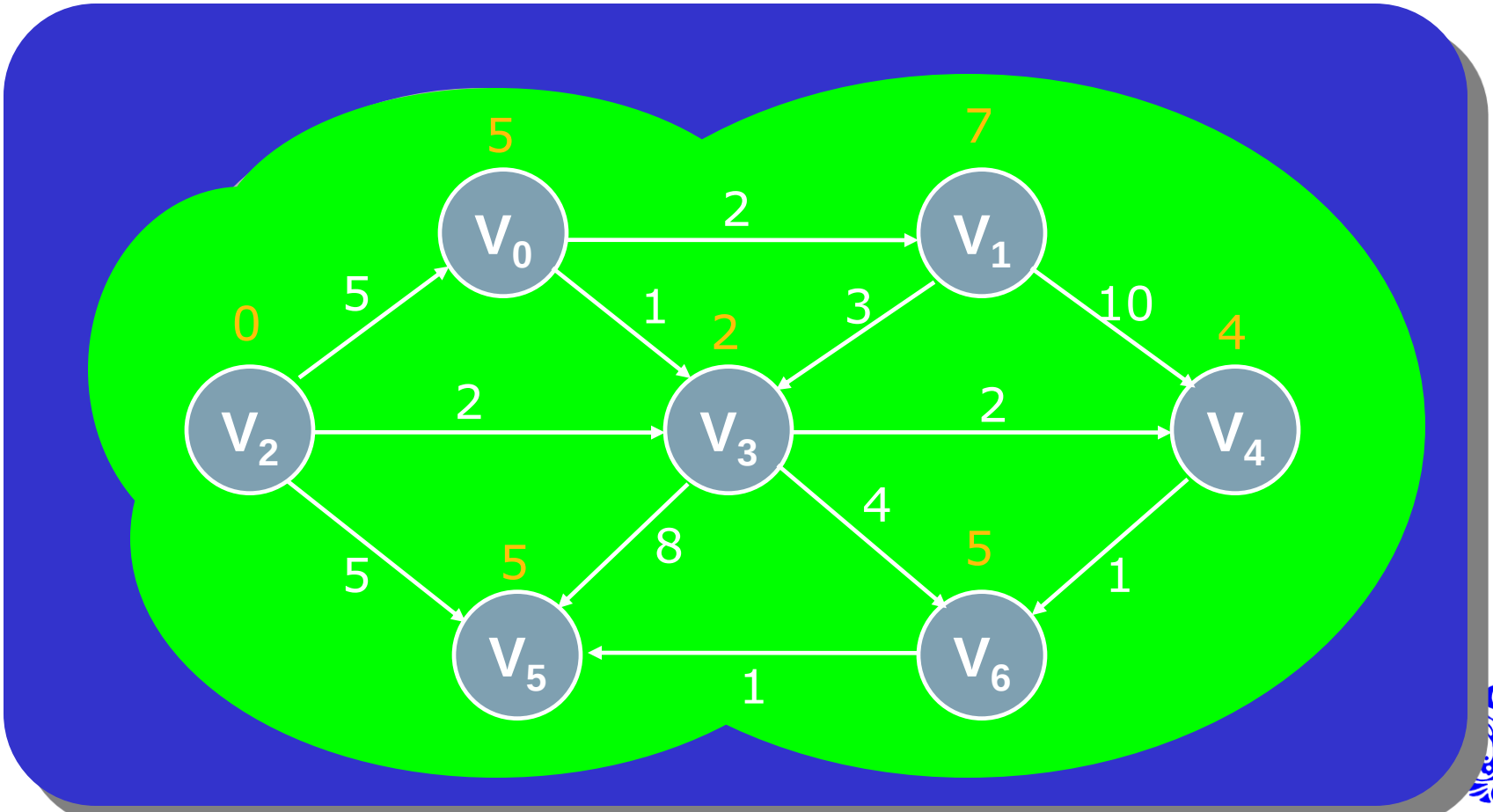
- pilih dari kelompok abu-abu, simpul yang memiliki nilai D[V] paling minimum dan tambahkan pada kelompok hijau.
- setelah V_5 ditambahkan ke kelompok hijau, hitung $D[V_x]$ untuk setiap V_x yang terhubung dengan V_5. Simpul-simpul tersebut menjadi kelompok abu-abu.



- pilih dari kelompok abu-abu, simpul yang memiliki nilai D[V] paling minimum dan tambahkan pada kelompok hijau.
- setelah V_6 ditambahkan ke kelompok hijau, hitung D[V_x] untuk setiap V_x yang terhubung dengan V_6.

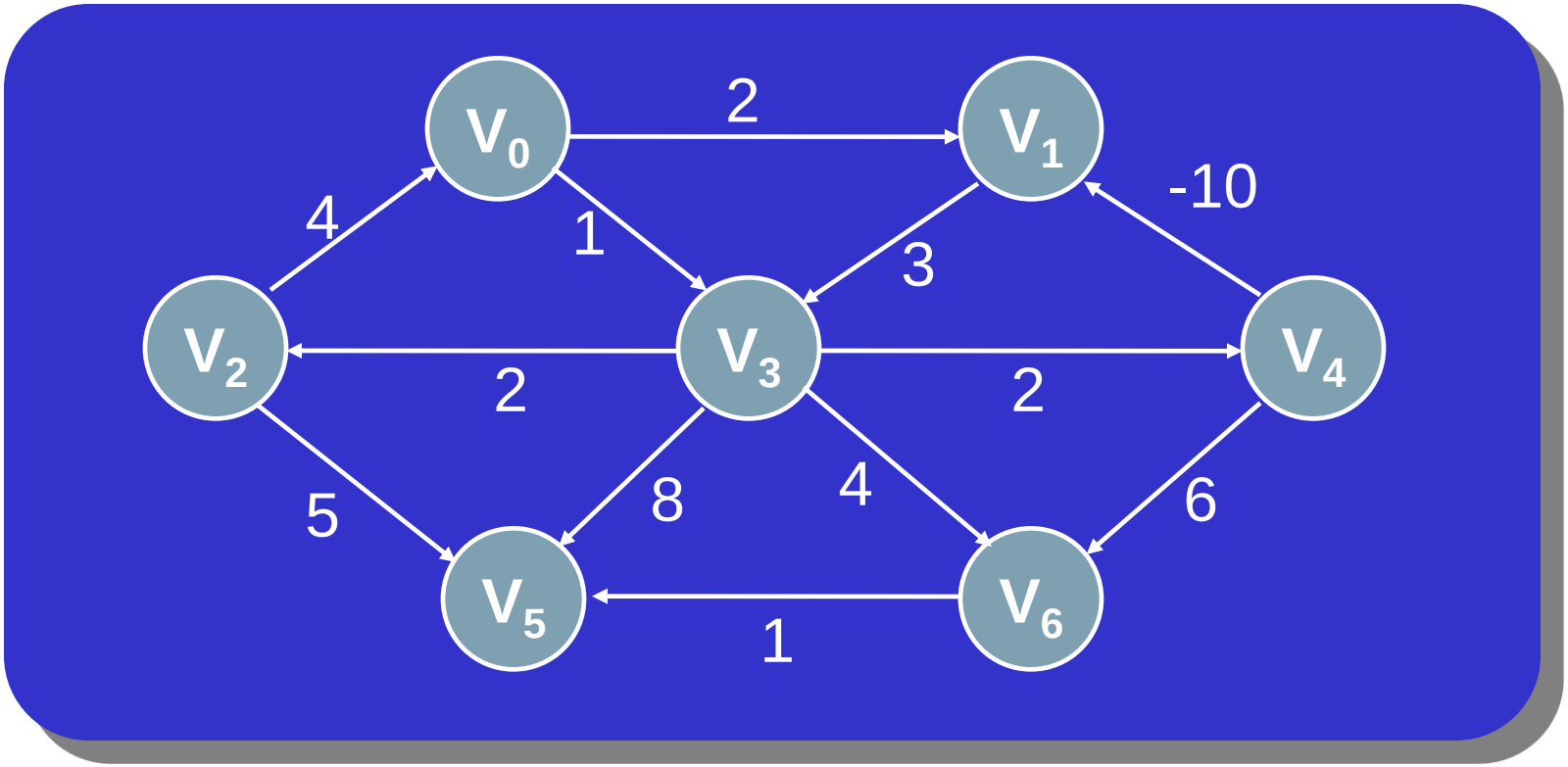


- pilih dari kelompok abu-abu, simpul yang memiliki nilai D[V] paling minimum dan tambahkan pada kelompok hijau.
- setelah V_l ditambahkan ke kelompok hijau, hitung $D[V_x]$ untuk setiap V_x yang terhubung dengan V_l.



Variasi shortest path problem

- Negative-weighted Shortest-path



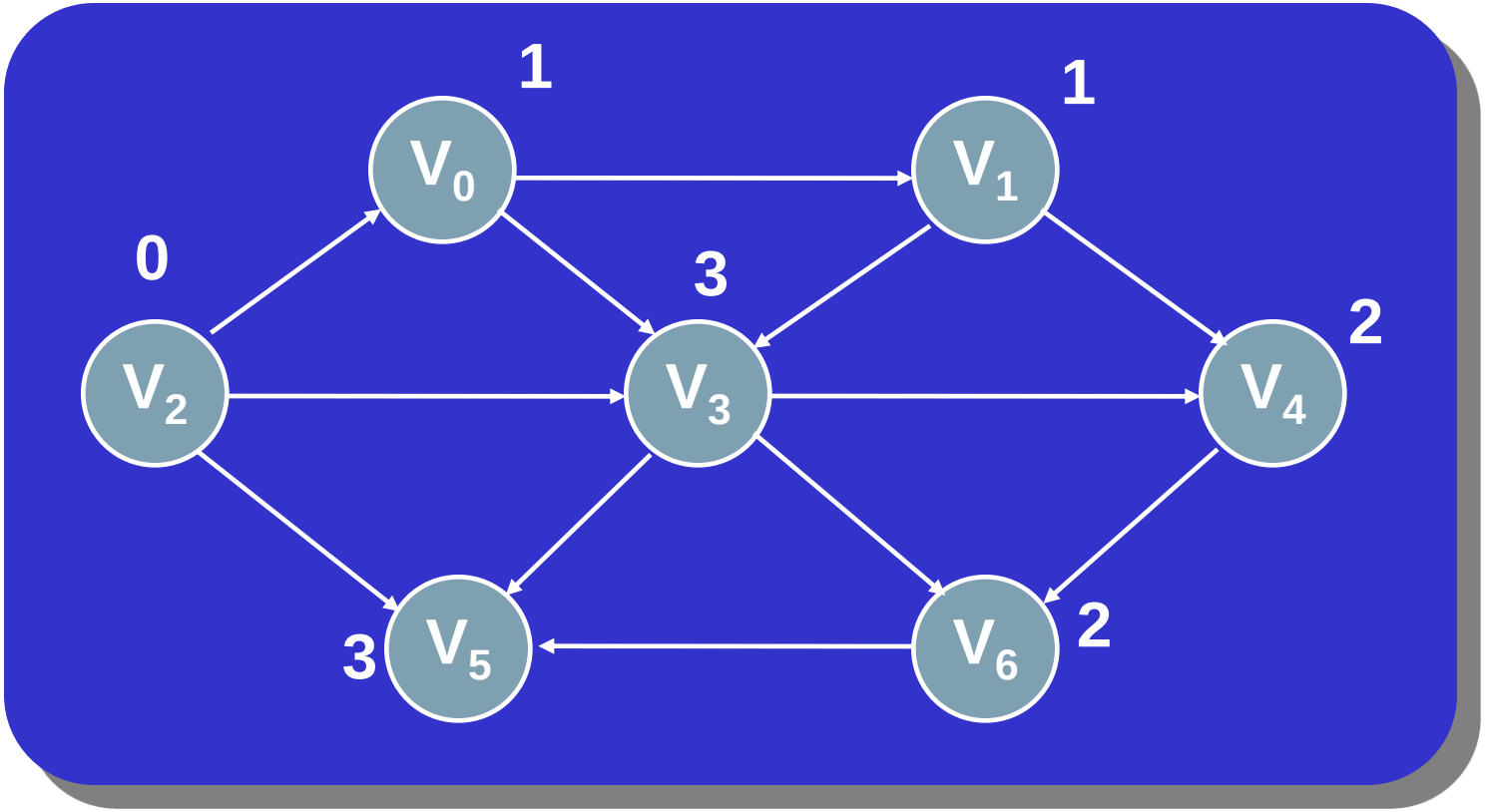
- All-Pair Shortest Path: Floyd

Topological Sorting

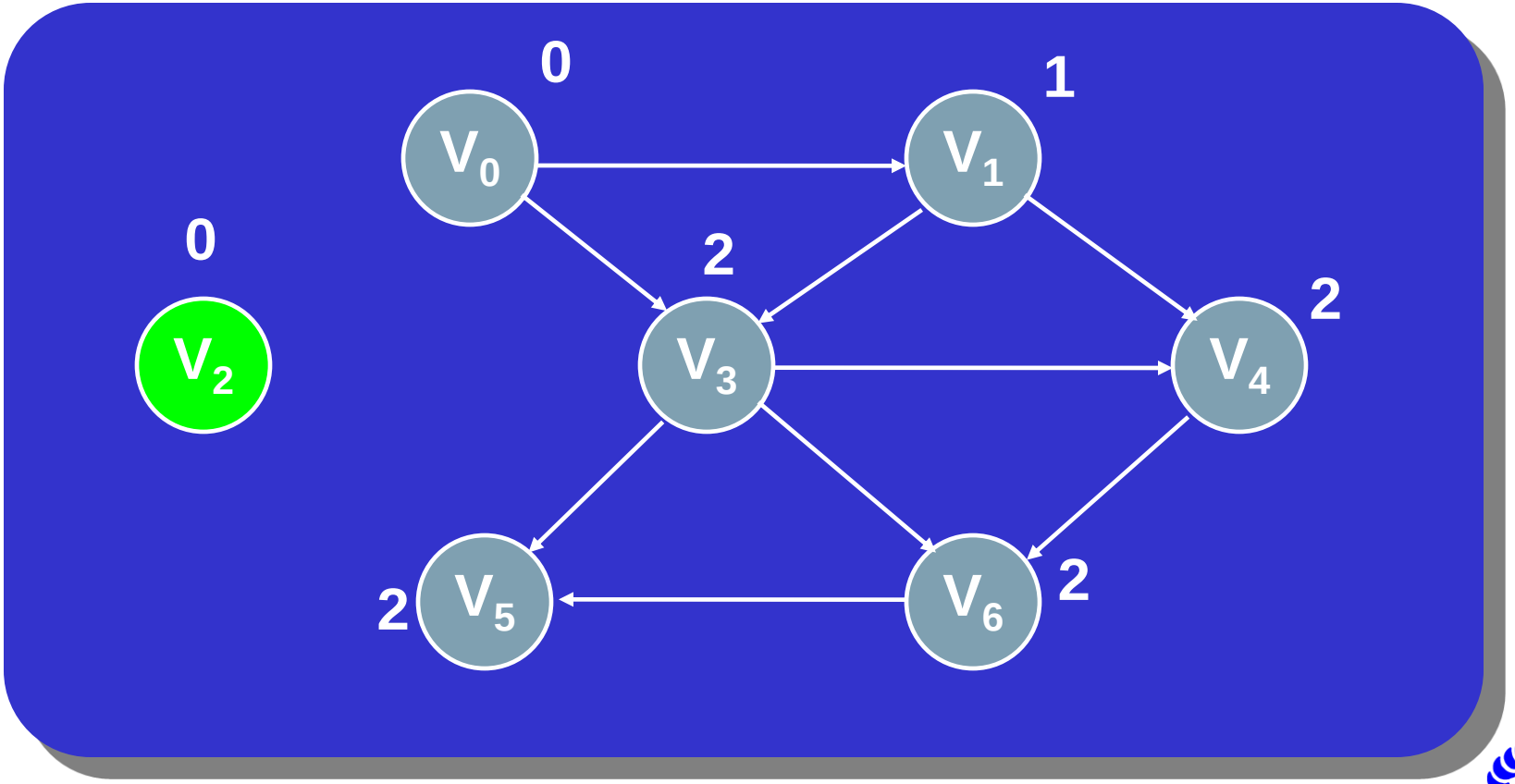
- Sebuah *topological sort* dari sebuah *directed acyclic graph* adalah sebuah urutan simpul sehingga tiap sisi/busur terurut dari kiri ke kanan (atau sebaliknya).
- Setiap DAG memiliki minimal satu topological sort.
- Gunakan algoritma *depth-first search* untuk menentukan topological sort dari sebuah DAG.
- sebuah graph yang memiliki *cycle*, tidak memiliki *topological sort*.
- Contoh permasalahan:
  - Urutan pengerjaan proyek bangunan

Topological Sorting: Algoritma

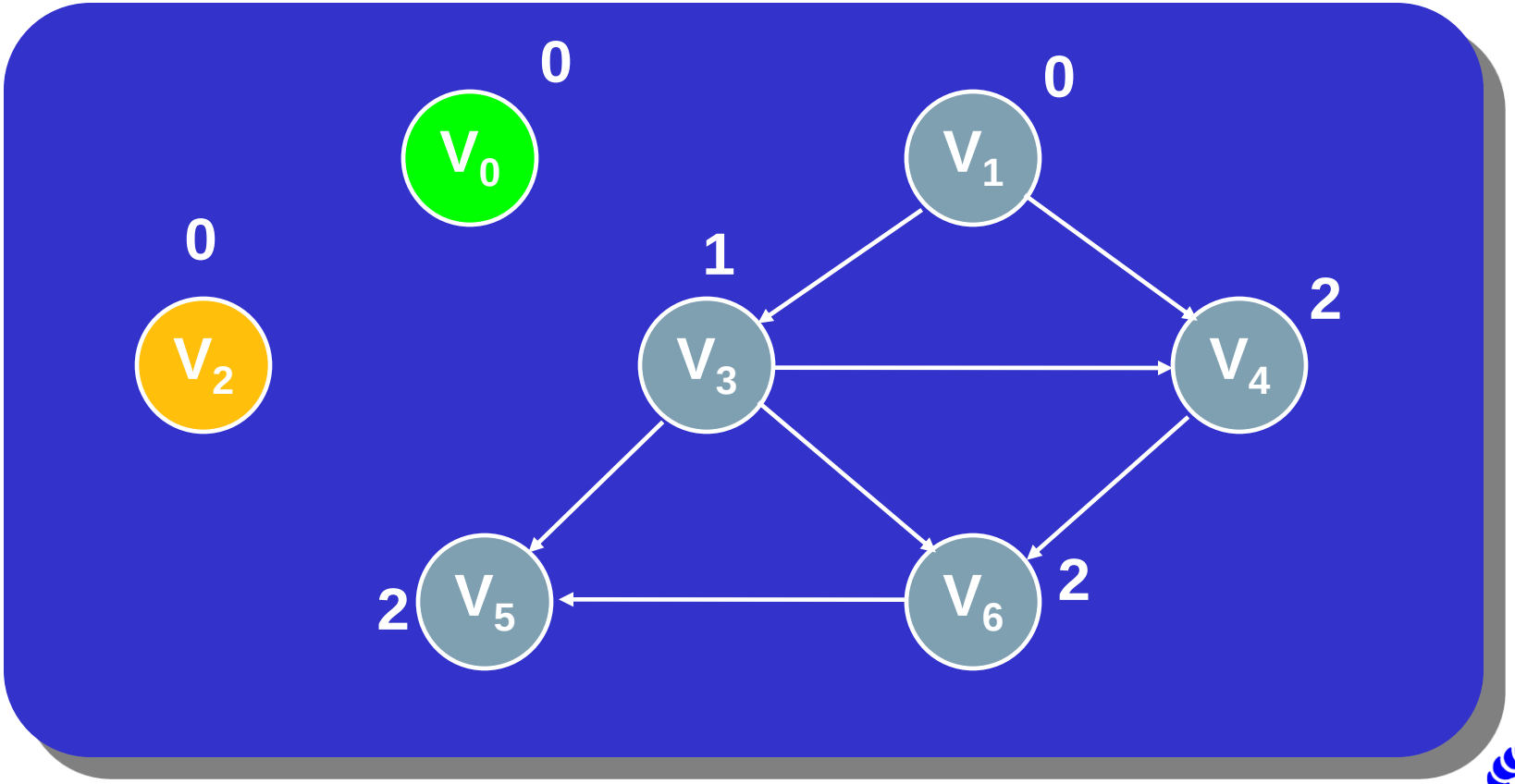
- Mulai dari sebuah simpul dengan in-degree = 0 (Tidak ada panah/sisi yang menuju simpul tersebut.)
- buang semua sisi yang berasal dari simpul tersebut.
- Sesuaikan nilai in-degree simpul lain-nya.



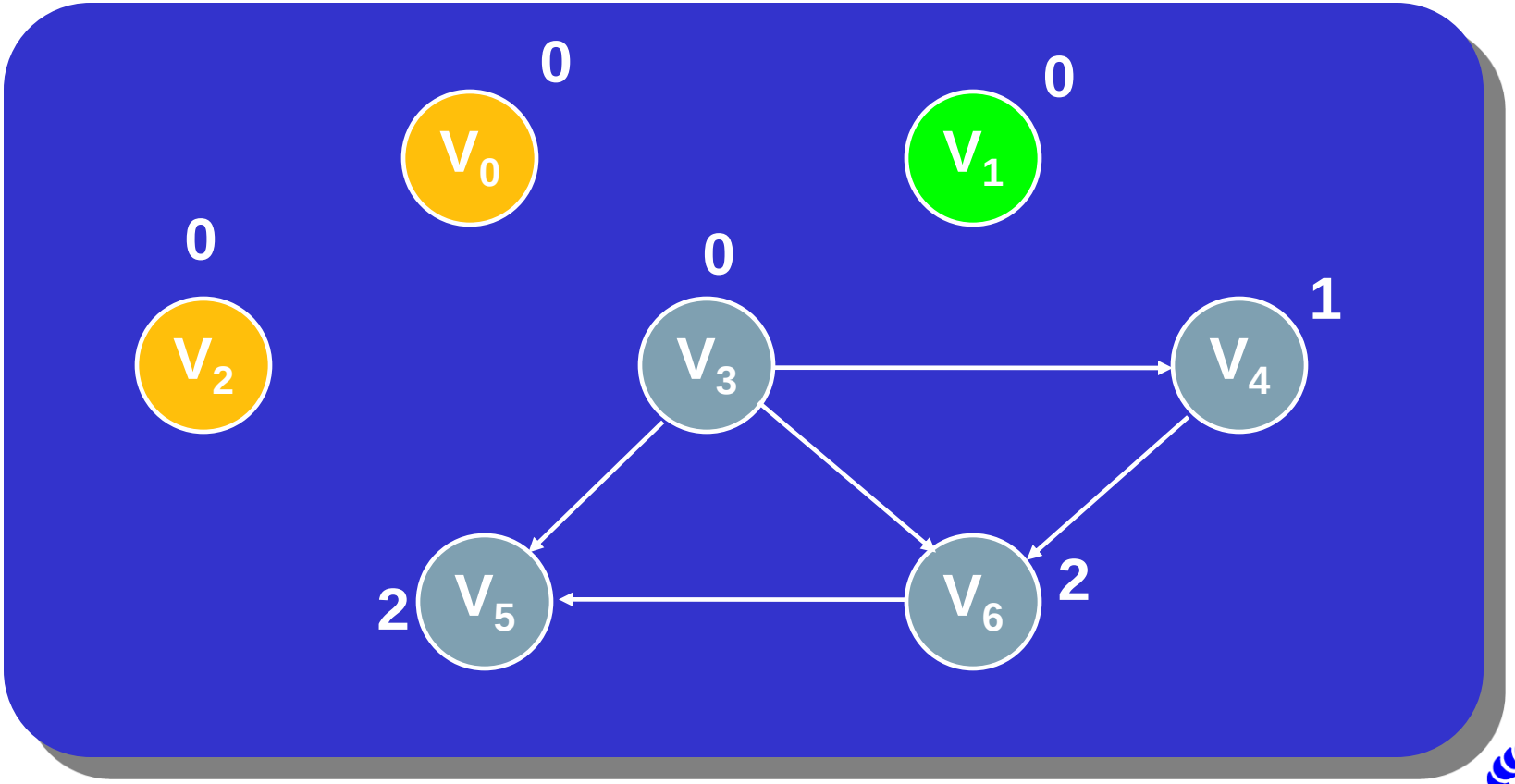
Topological Sorting



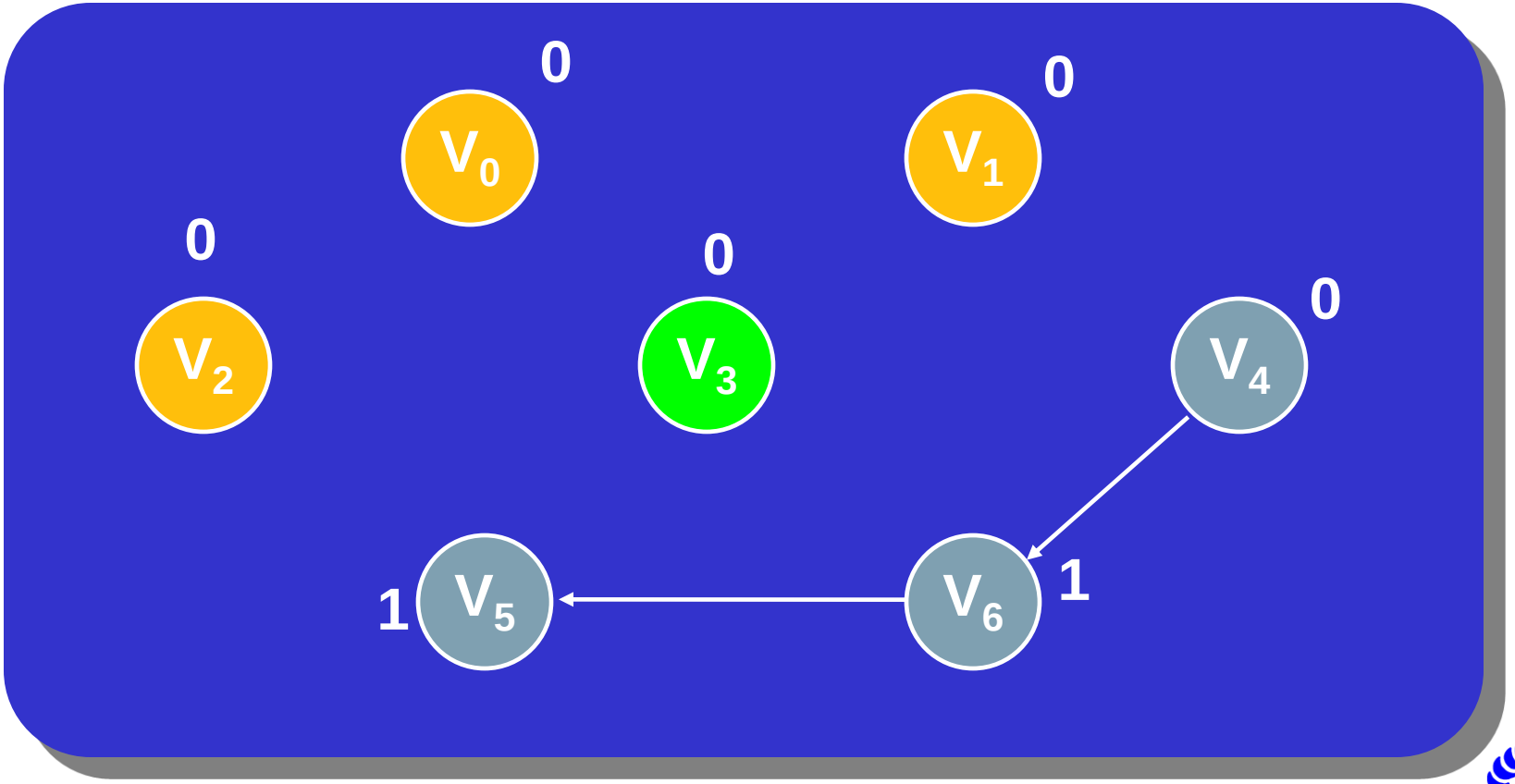
Topological Sorting



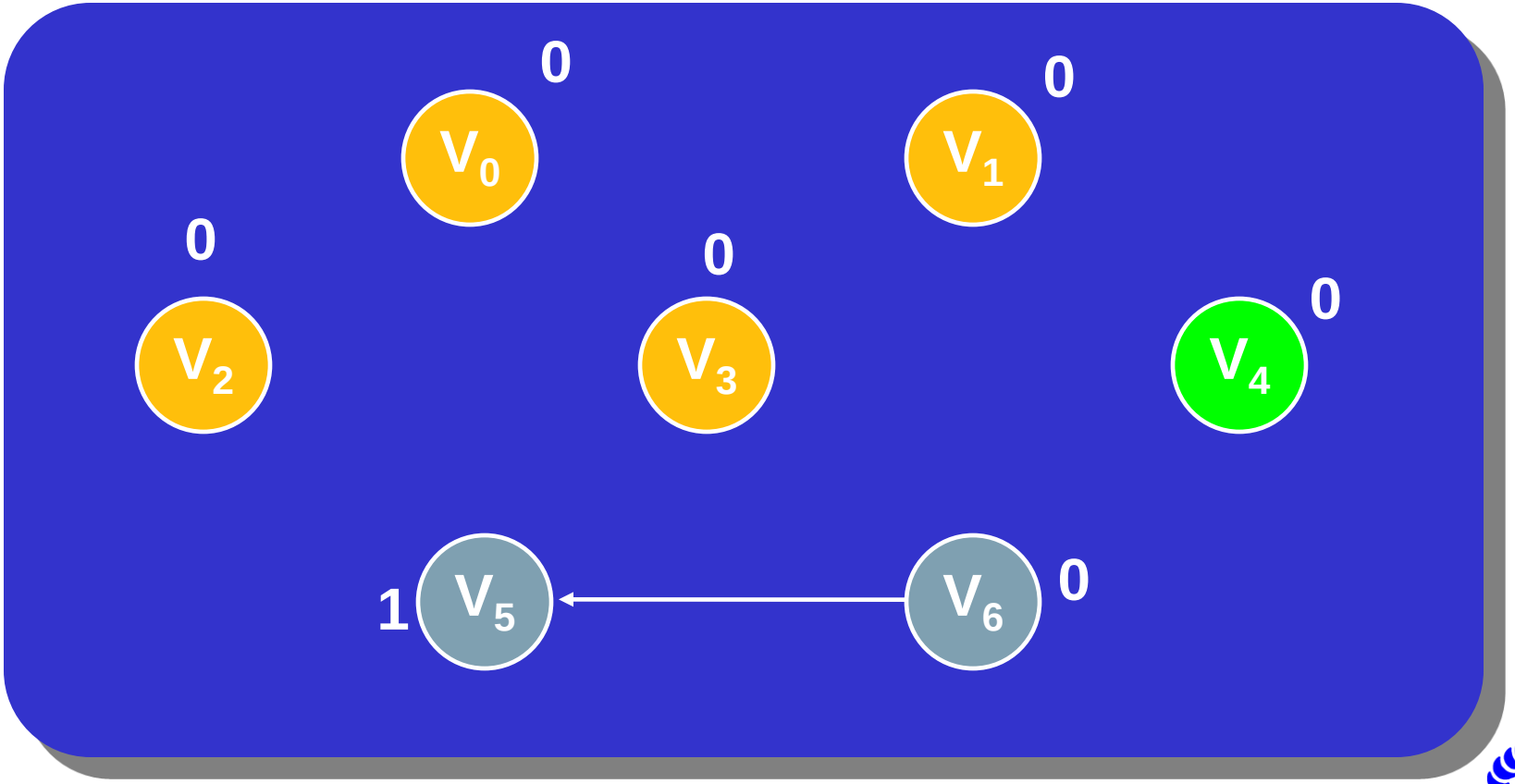
Topological Sorting



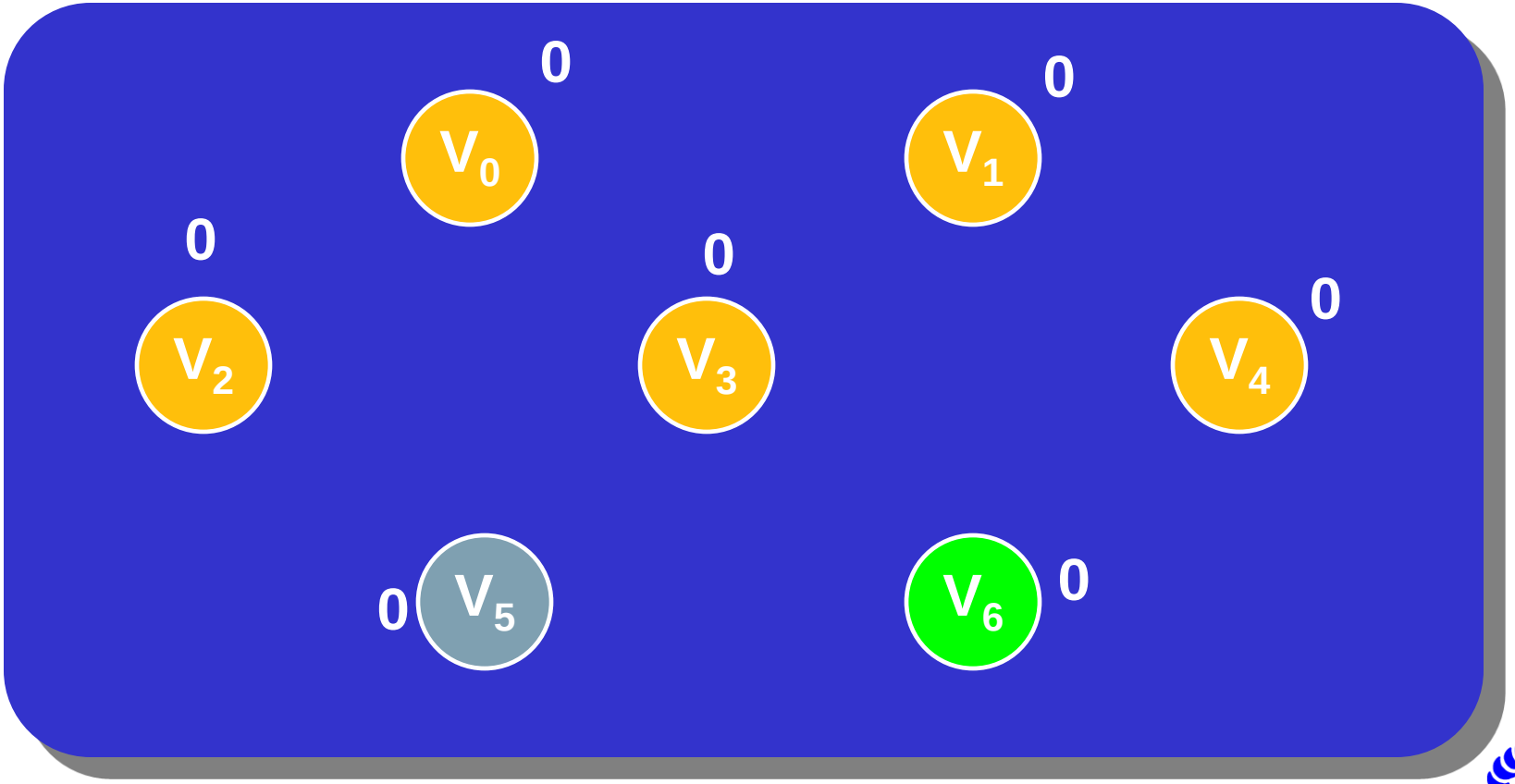
Topological Sorting



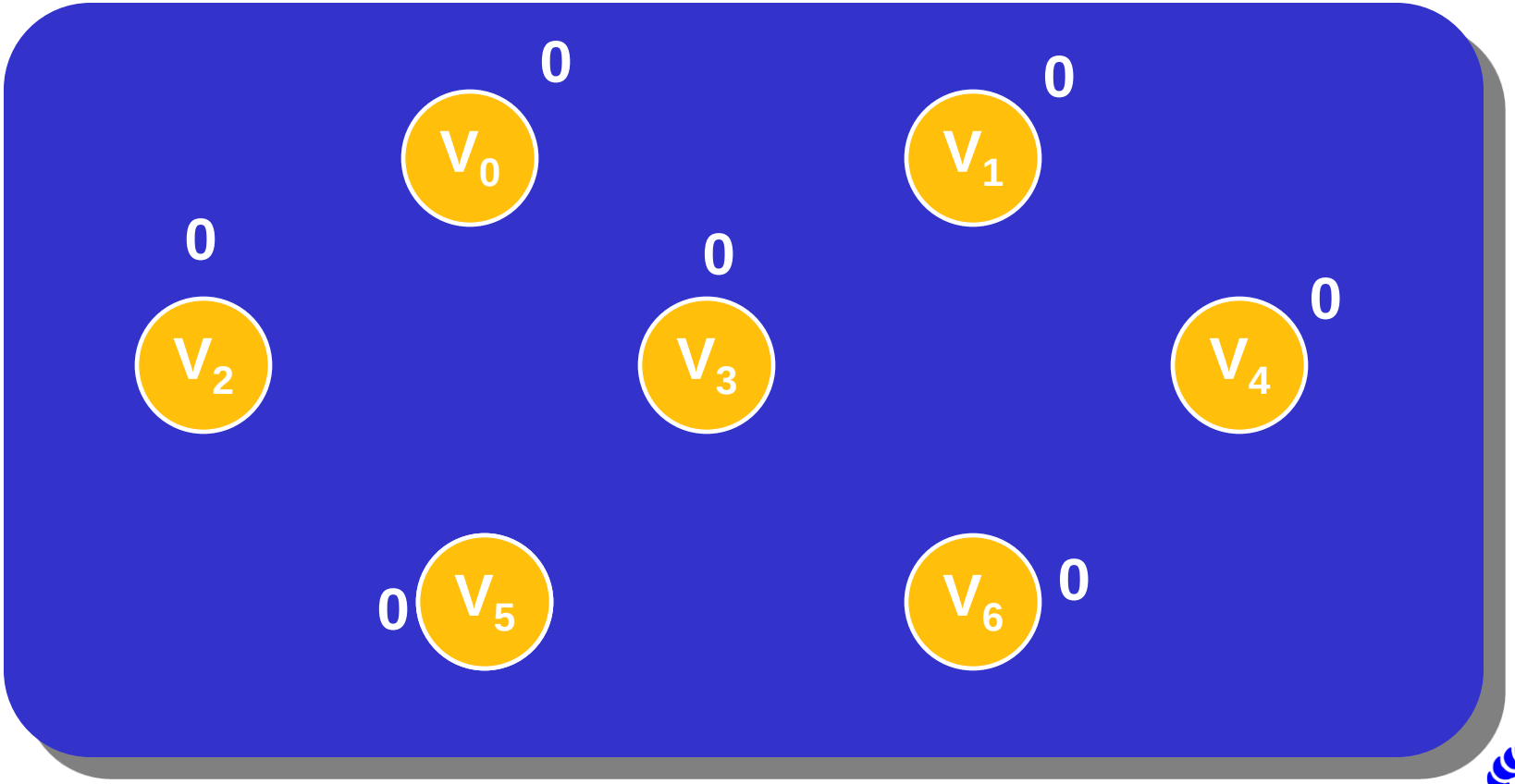
Topological Sorting



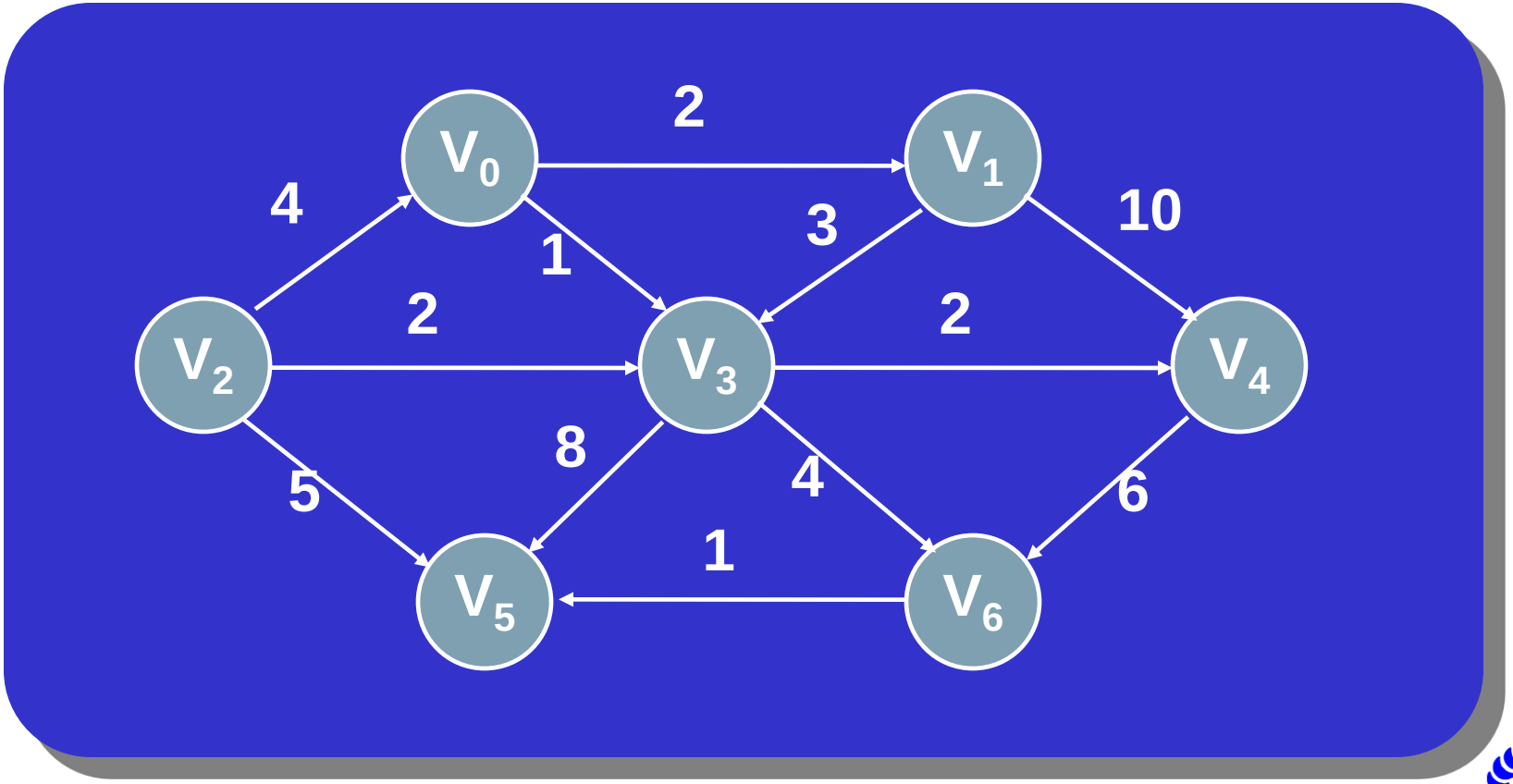
Topological Sorting



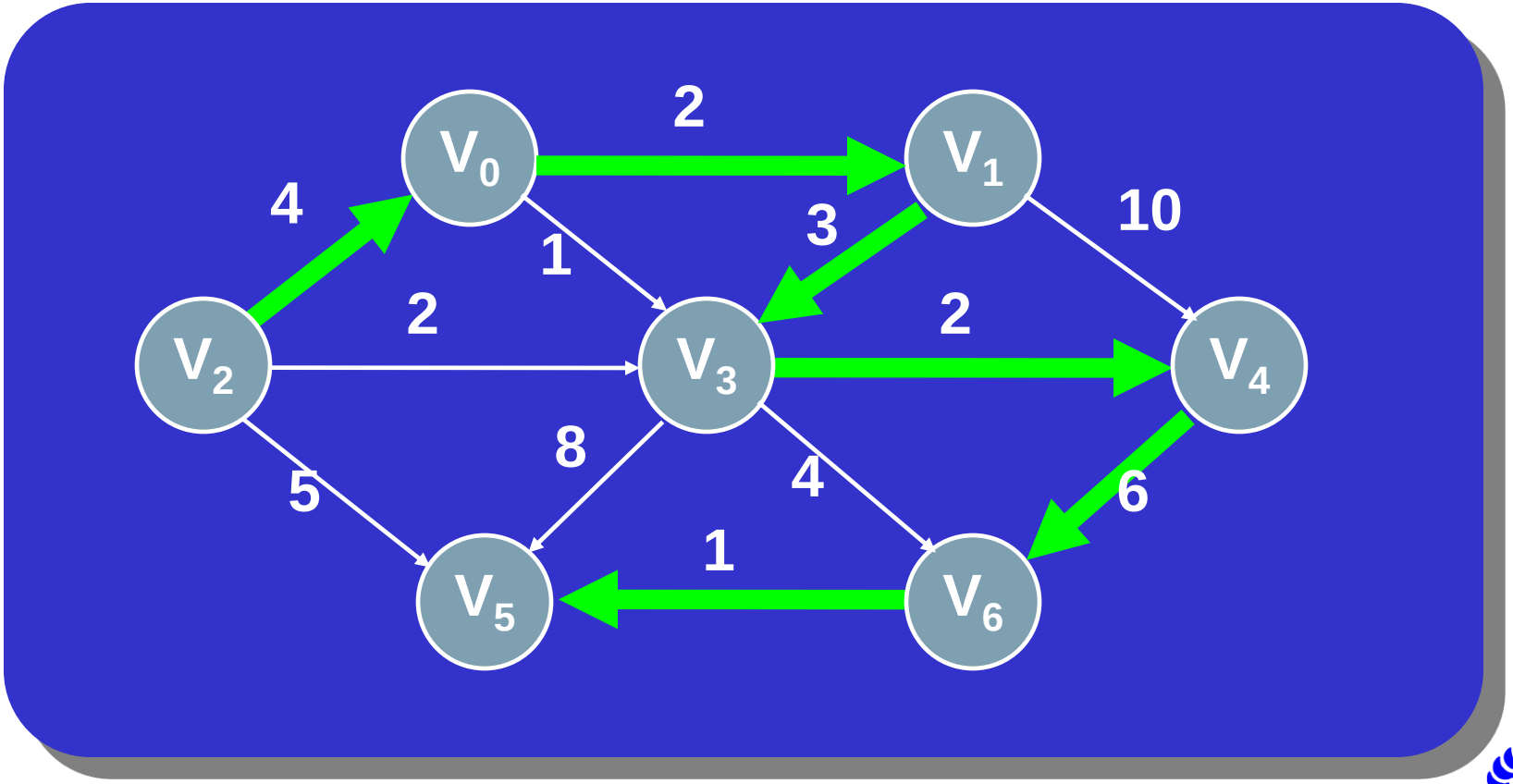
Topological Sorting



Topological Sorting



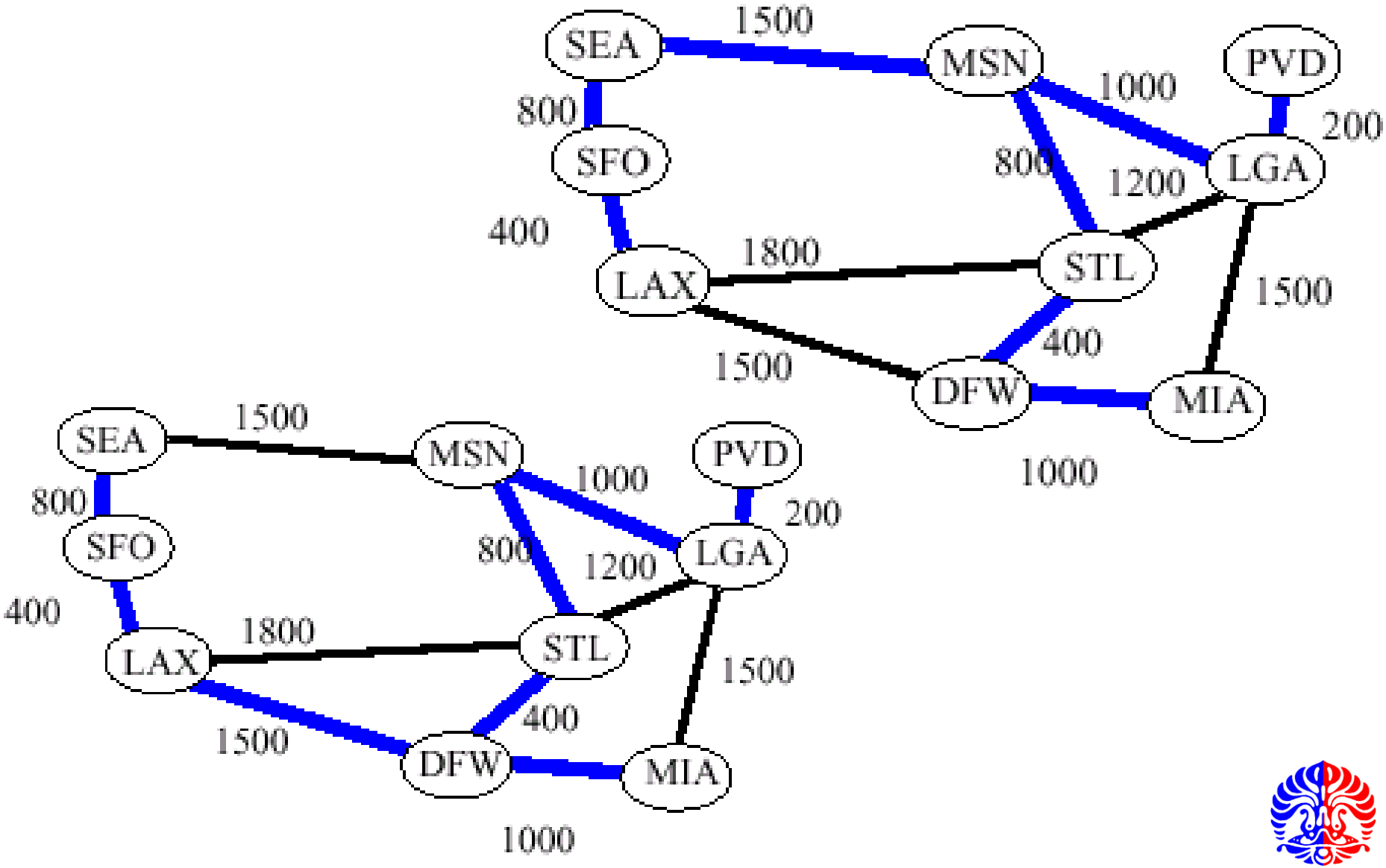
Topological Sorting



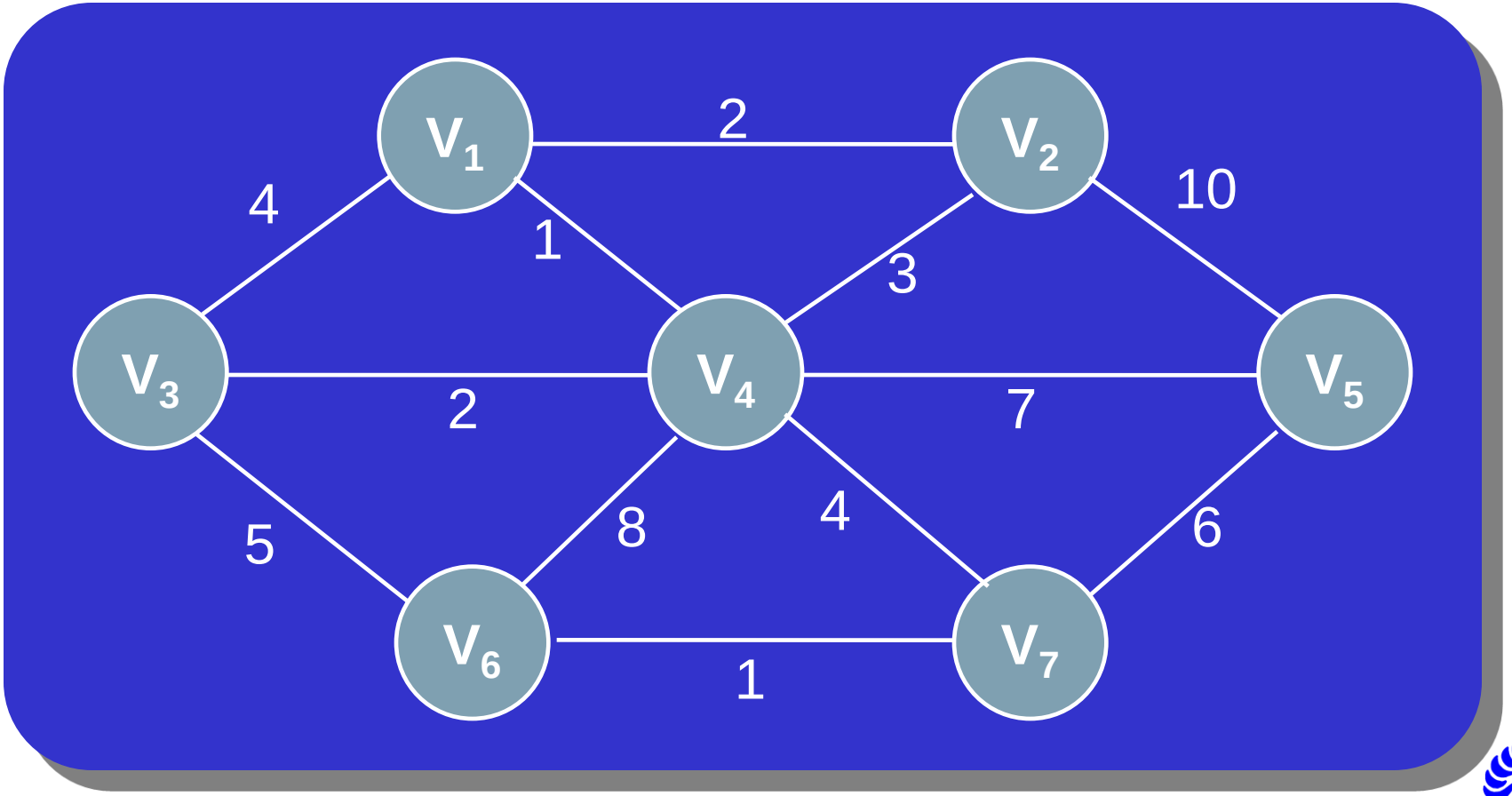
Minimum Spanning Tree (MST)

- Adalah sebuah struktur *tree* yang terbentuk dari graph, dimana sisi-sisi yang menghubungkan setiap simpul memiliki nilai total paling kecil.
- Nilai total dari sebuah spanning tree, adalah jumlah total bobot tiap sisi dalam *tree* tersebut.
- Penerapan:
  - Mencari jumlah biaya kabel paling minimum untuk menghubungkan sebuah kelompok perumahan atau perkotaan.
  - Mencari biaya minimum terendah untuk menghubungkan jaringan komputer.
  - Mencari biaya produksi total terendah untuk pengerjaan proyek.

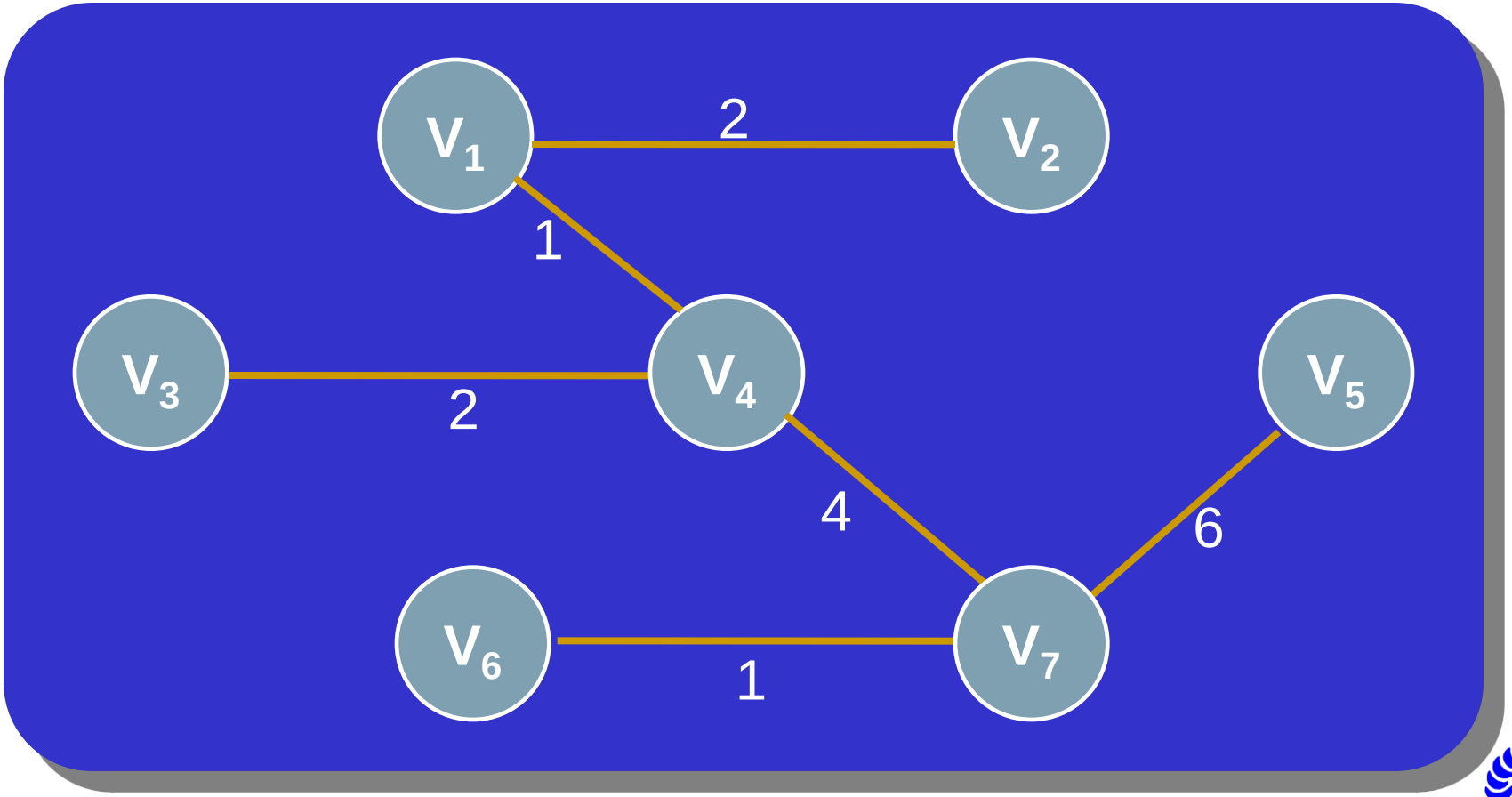
Minimum Spanning Tree (MST)



Minimum Spanning Tree: a graph



Minimum Spanning Tree

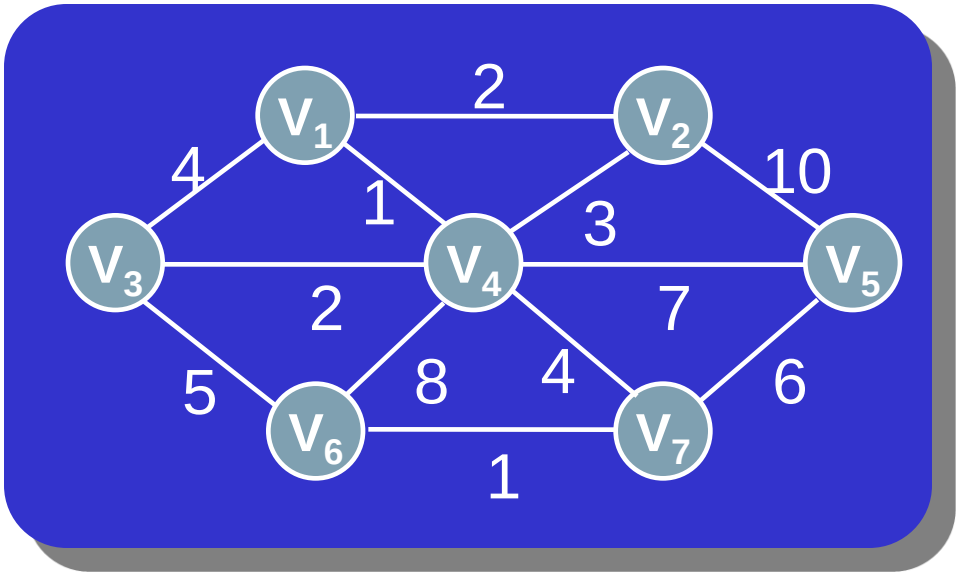


Prim's Algorithm

- mulai dari sebuah simpul
- bangun tree dengan menambahkan sebuah sisi/busur satu persatu.
  - secara berulang pilih sisi terkecil yang dapat menyambung tree.
- greedy algorithms:
  - Pilihan langkah diambil berdasarkan pilihan terbaik secara local tanpa memperhatikan pengaruhnya secara global.

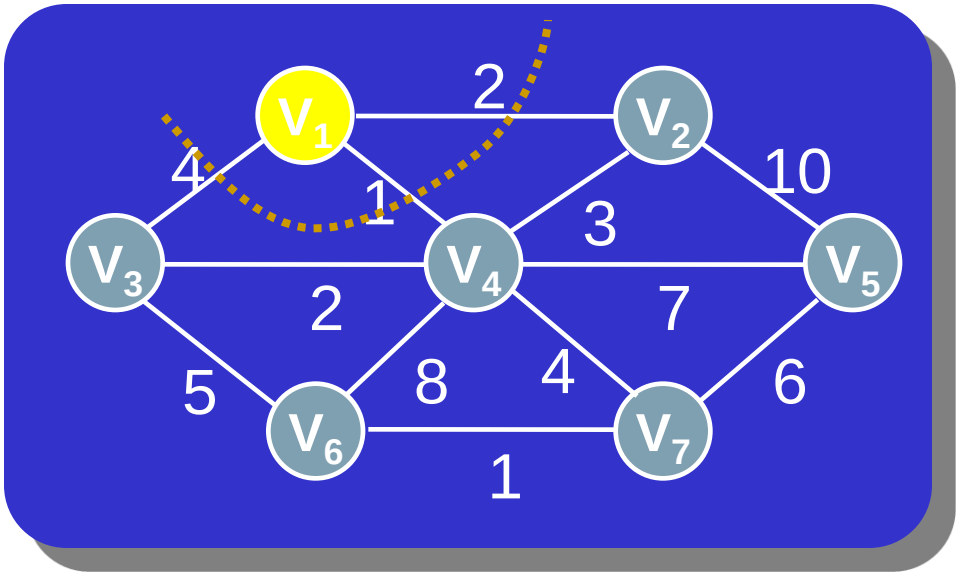
Prim's Algorithm

V	known	d_V	p_V
V_1	0	0	0
V_2	0	∞	0
V_3	0	∞	0
V_4	0	∞	0
V_5	0	∞	0
V_6	0	∞	0
V_7	0	∞	0



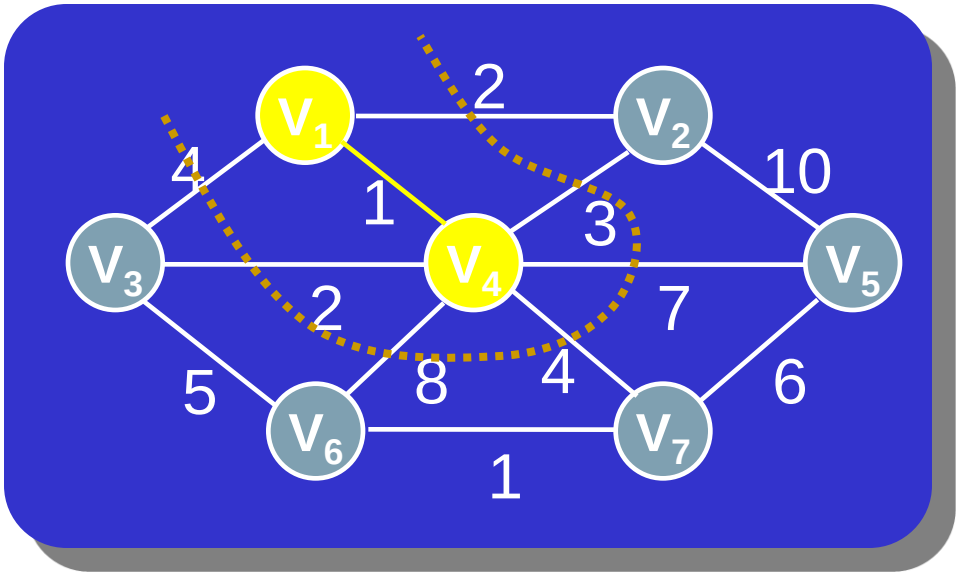
Prim's Algorithm

v	known	d_v	p_v
V_1	1	0	0
V_2	0	2	V_1
V_3	0	4	V_1
V_4	0	1	V_1
V_5	0	∞	0
V_6	0	∞	0
V_7	0	∞	0



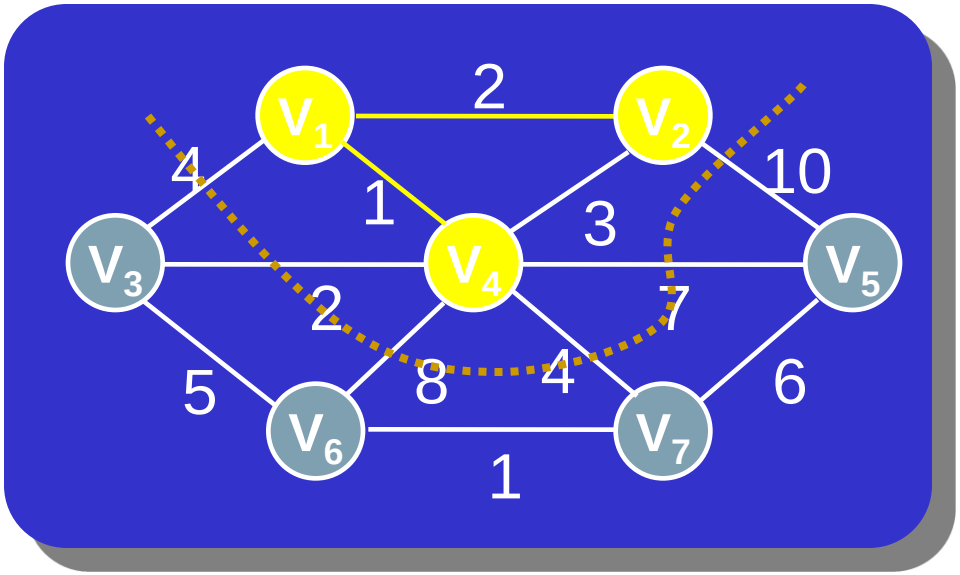
Prim's Algorithm

V	known	d_v	p_v
V_1	1	0	0
V_2	0	2	V_1
V_3	0	2	V_4
V_4	1	1	V_1
V_5	0	7	V_4
V_6	0	8	V_4
V_7	0	4	V_4



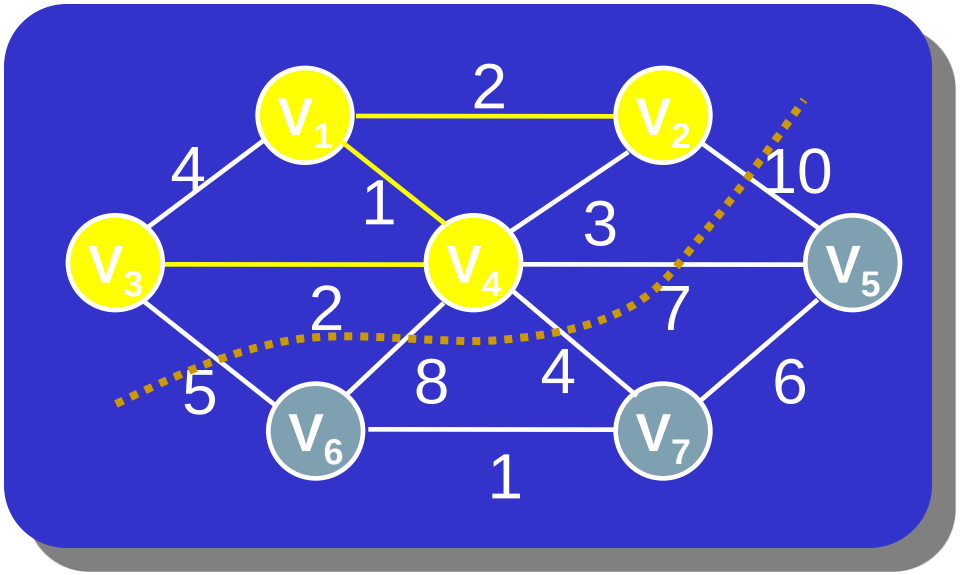
Prim's Algorithm

V	known	d_v	p_v
V_1	1	0	0
V_2	1	2	V_1
V_3	0	2	V_4
V_4	1	1	V_1
V_5	0	7	V_4
V_6	0	8	V_4
V_7	0	4	V_4



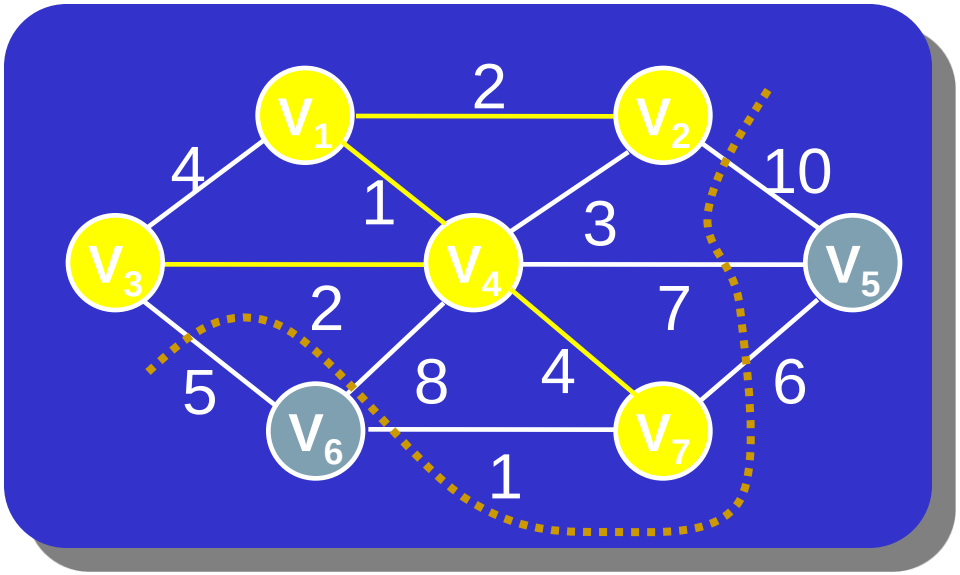
Prim's Algorithm

v	known	d_v	p_v
V_1	1	0	0
V_2	1	2	V_1
V_3	1	2	V_4
V_4	1	1	V_1
V_5	0	7	V_4
V_6	0	5	V_3
V_7	0	4	V_4



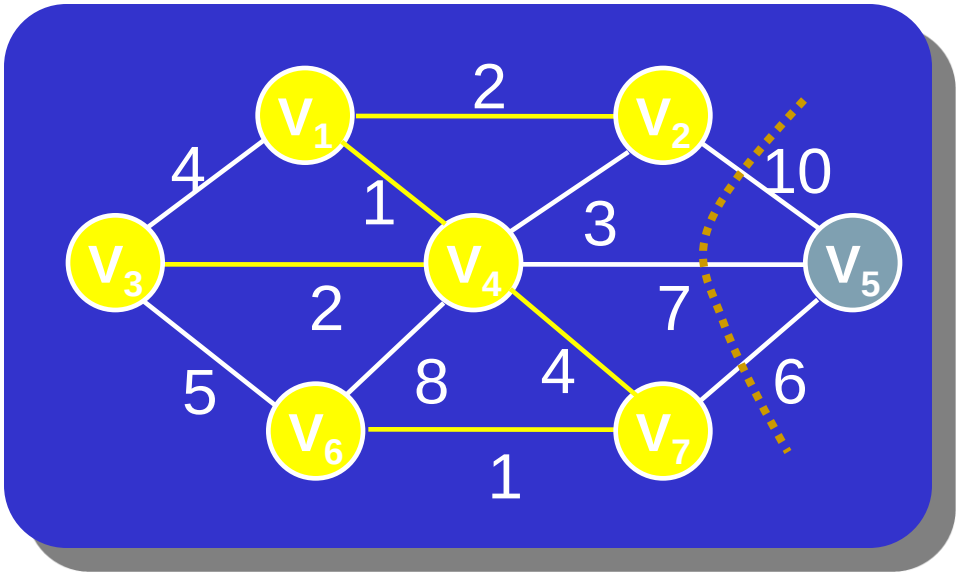
Prim's Algorithm

v	known	d_v	p_v
V_1	1	0	0
V_2	1	2	V_1
V_3	1	2	V_4
V_4	1	1	V_1
V_5	0	6	V_7
V_6	0	1	V_7
V_7	1	4	V_4



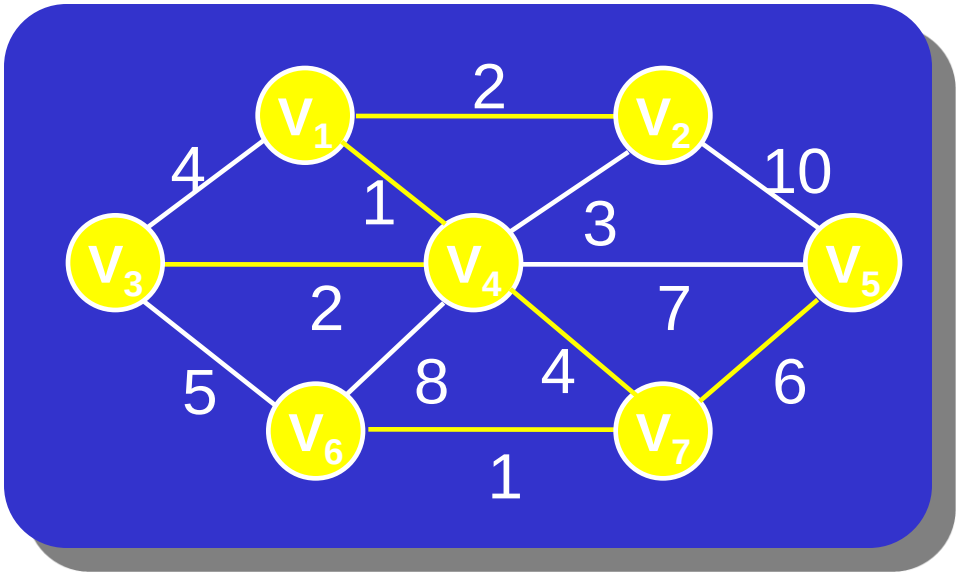
Prim's Algorithm

V	known	d_v	p_v
V_1	1	0	0
V_2	1	2	V_1
V_3	1	2	V_4
V_4	1	1	V_1
V_5	0	6	V_7
V_6	1	1	V_7
V_7	1	4	V_4



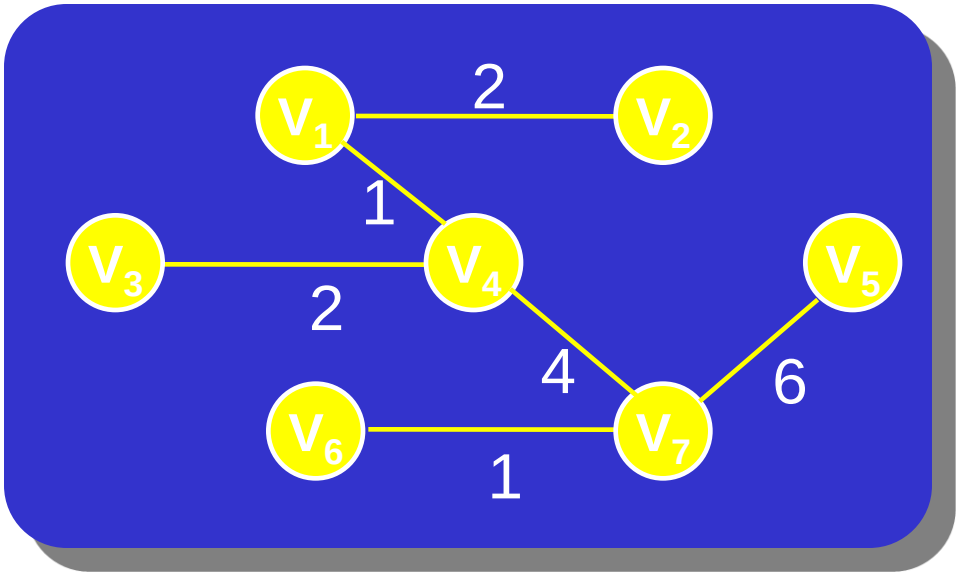
Prim's Algorithm

V	known	d_v	p_v
V_1	1	0	0
V_2	1	2	V_1
V_3	1	2	V_4
V_4	1	1	V_1
V_5	1	6	V_7
V_6	1	1	V_7
V_7	1	4	V_4



Prim's Algorithm

v	known	d_v	p_v
v_1	1	0	0
v_2	1	2	v_1
v_3	1	2	v_4
v_4	1	1	v_1
v_5	1	6	v_7
v_6	1	1	v_7
v_7	1	4	v_4

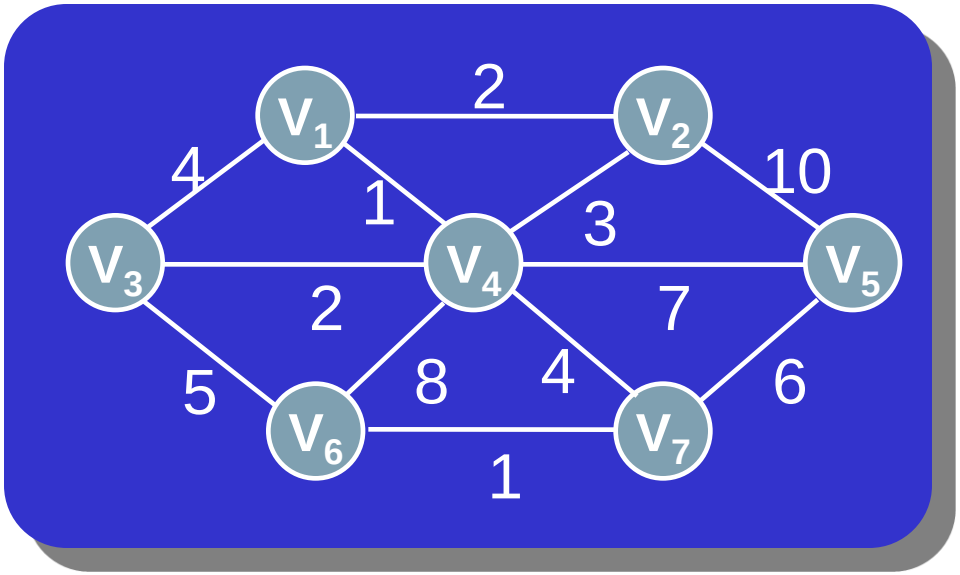


Kruskal's Algorithm

- Tambahkan sebuah sisi terkecil pada tiap iterasi. Seluruh sisi dapat di urutkan dulu berdasarkan bobot-nya.
- Sisi dapat ditambahkan *hanya jika* tidak menimbulkan cycle, atau tidak menuju simpul yang sudah dikunjungi.

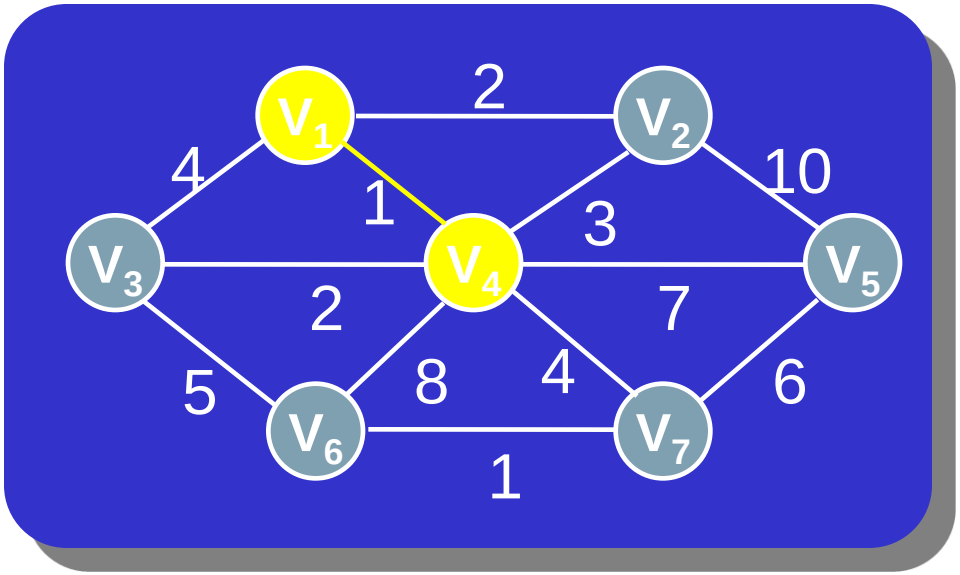
Kruskal's Algorithm

Edge	Weight	Action
(V_1, V_4)	1	-
(V_6, V_7)	1	-
(V_1, V_2)	2	-
(V_3, V_4)	2	-
(V_2, V_4)	3	-
(V_1, V_3)	4	-
(V_4, V_7)	4	-
(V_3, V_6)	5	-
(V_5, V_7)	6	-



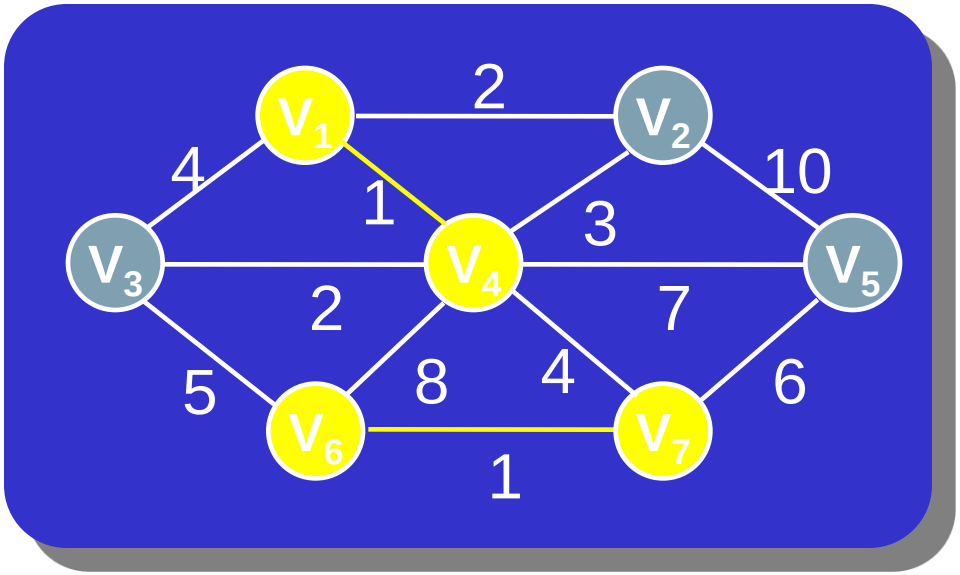
Kruskal's Algorithm

Edge	Weight	Action
(V_1, V_4)	1	A
(V_6, V_7)	1	-
(V_1, V_2)	2	-
(V_3, V_4)	2	-
(V_2, V_4)	3	-
(V_1, V_3)	4	-
(V_4, V_7)	4	-
(V_3, V_6)	5	-
(V_5, V_7)	6	-



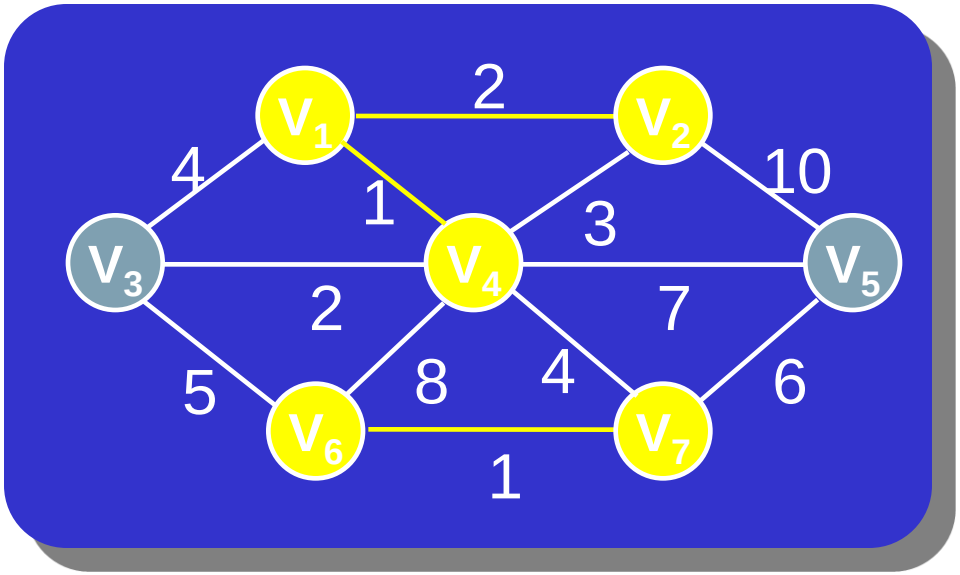
Kruskal's Algorithm

Edge	Weight	Action
(V_1, V_4)	1	A
(V_6, V_7)	1	A
(V_1, V_2)	2	-
(V_3, V_4)	2	-
(V_2, V_4)	3	-
(V_1, V_3)	4	-
(V_4, V_7)	4	-
(V_3, V_6)	5	-
(V_5, V_7)	6	-



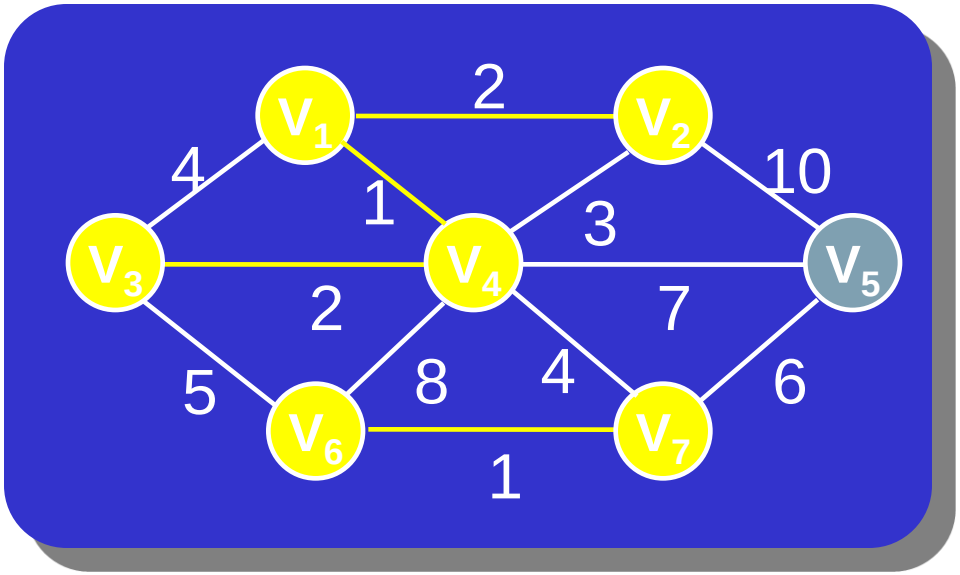
Kruskal's Algorithm

Edge	Weight	Action
(V_1, V_4)	1	A
(V_6, V_7)	1	A
(V_1, V_2)	2	A
(V_3, V_4)	2	-
(V_2, V_4)	3	-
(V_1, V_3)	4	-
(V_4, V_7)	4	-
(V_3, V_6)	5	-
(V_5, V_7)	6	-



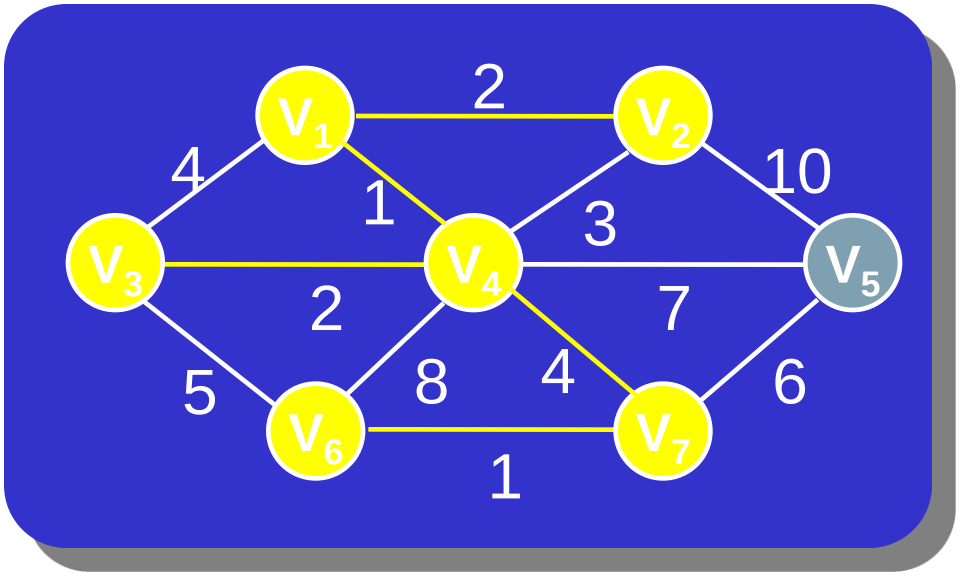
Kruskal's Algorithm

Edge	Weight	Action
(V_1, V_4)	1	A
(V_6, V_7)	1	A
(V_1, V_2)	2	A
(V_3, V_4)	2	A
(V_2, V_4)	3	-
(V_1, V_3)	4	-
(V_4, V_7)	4	-
(V_3, V_6)	5	-
(V_5, V_7)	6	-



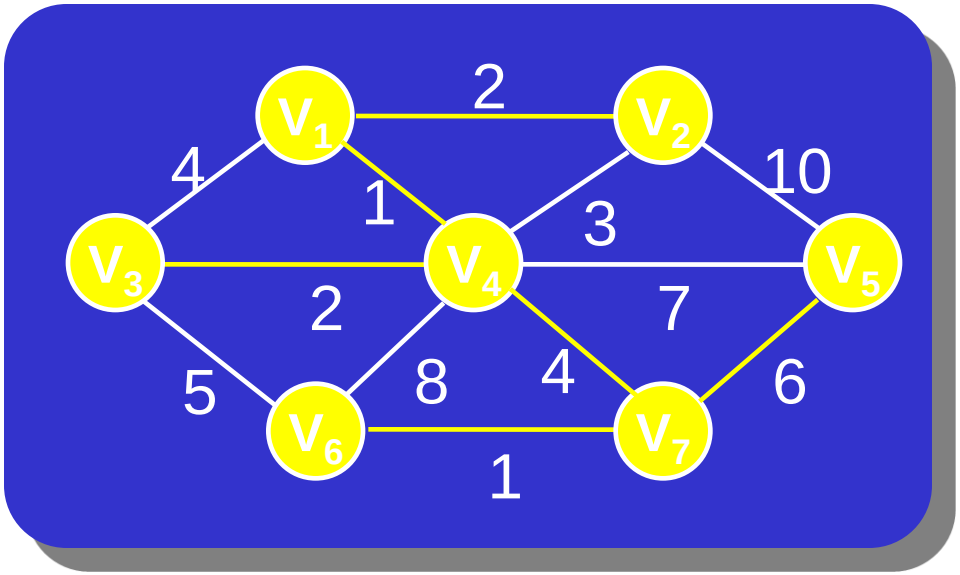
Kruskal's Algorithm

Edge	Weight	Action
(V_1, V_4)	1	A
(V_6, V_7)	1	A
(V_1, V_2)	2	A
(V_3, V_4)	2	A
(V_2, V_4)	3	R
(V_1, V_3)	4	R
(V_4, V_7)	4	A
(V_3, V_6)	5	-
(V_5, V_7)	6	-



Kruskal's Algorithm

Edge	Weight	Action
(V1, V4)	1	A
(V6, V7)	1	A
(V1, V2)	2	A
(V3, V4)	2	A
(V2, V4)	3	R
(V1, V3)	4	R
(V4, V7)	4	A
(V3, V6)	5	R
(V5, V7)	6	A



Kruskal's Algorithm

Edge	Weight	Action
(V1, V4)	1	A
(V6, V7)	1	A
(V1, V2)	2	A
(V3, V4)	2	A
(V2, V4)	3	R
(V1, V3)	4	R
(V4, V7)	4	A
(V3, V6)	5	R
(V5, V7)	6	A