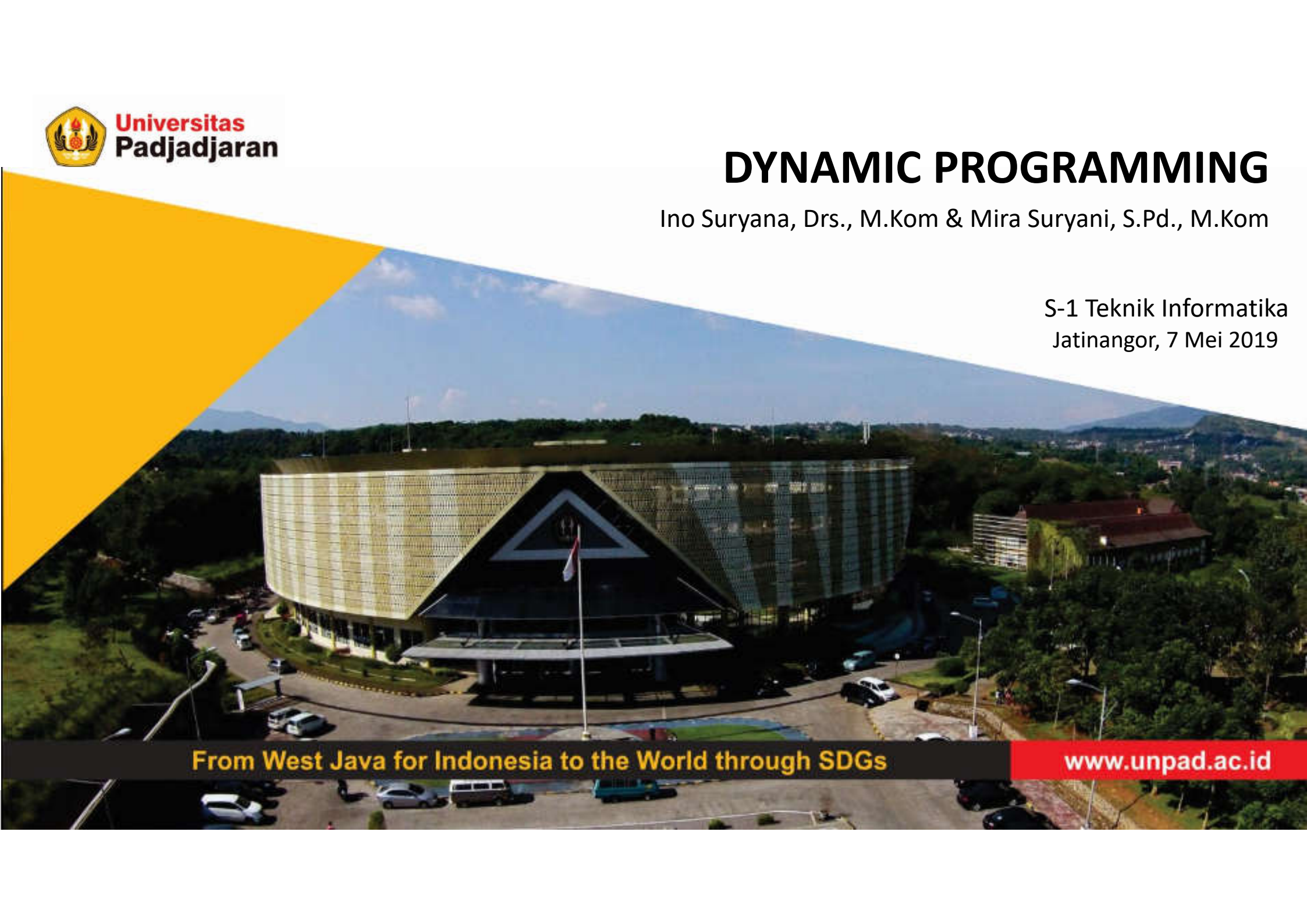


DASAR ALGORITMA GRAF

Ino Suryana, Drs., M.Kom & Mira Suryani, S.Pd., M.Kom

S-1 Teknik Informatika
Jatinangor, 2 April 2019



From West Java for Indonesia to the World through SDGs

www.unpad.ac.id



Pembahasan Materi

1. Definisi Dasar dan Contoh Aplikasi Graf
2. Konektivitas dan Penelusuran Graf
3. Analisis Algoritma Breadth First Search
4. Testing Ke-bipartit-an: Aplikasi BFS



Tujuan Pembelajaran

Setelah mempelajari materi ini diharapkan Anda:

- Memahami konsep graf dan menganalisis kompleksitas dari beberapa jenis graf.



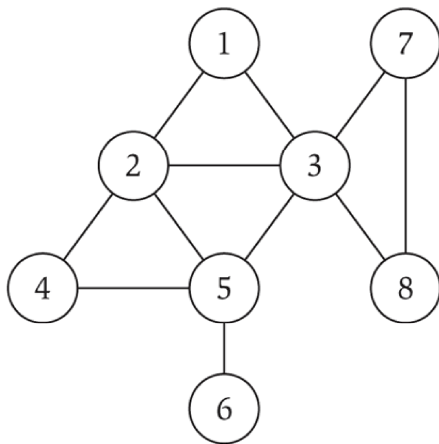
1. Definisi Dasar dan Contoh Aplikasi Graf



Graf Tak Berarah (Undirected Graph)

(Undirected) graph: $G=(V,E)$

- V = sekumpulan node (vertex, simpul, titik, sudut)
- E = sekumpulan edge (garis, tepi)
- Menangkap hubungan berpasangan antar objek.
- Parameter ukuran Graf: $n = |V|$, $m = |E|$



$V = \{1,2,3,4,5,6,7,8\}$

$E = \{(1,2), (1,3), (2,3), (2,4), (2,5), (3,5), (3,7), (3,8), (4,5), (5,6), (7,8)\}$

$n = 8$

$M = 11$



Contoh Implementasi Graf

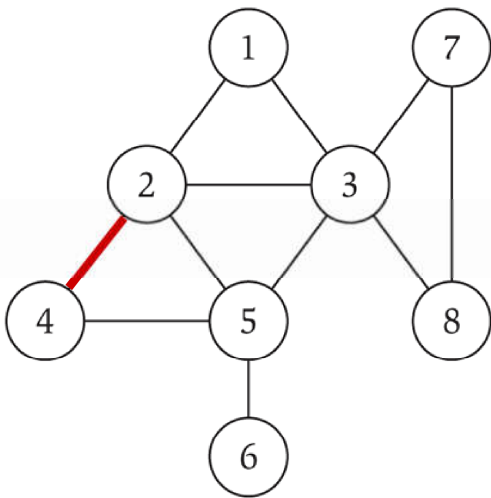
<i>Graph</i>	<i>Nodes</i>	<i>Edges</i>
transportation	street intersections	highways
communication	computers	fiber optic cables
World Wide Web	web pages	hyperlinks
social	people	relationships
food web	species	predator-prey
software systems	functions	function calls
scheduling	tasks	precedence constraints
circuits	gates	wires



Representasi Graf: Adjacency Matrix

Adjacency Matrix: n-ke-n matriks dengan $A_{uv} = 1$ jika (u,v) adalah sebuah garis

- Dua representasi dari setiap sisi
- Ruang berukuran sebesar n^2
- Memeriksa apakah (u, v) edge membutuhkan waktu $\Theta(1)$
- Mengidentifikasi semua tepi membutuhkan $\Theta(n^2)$ waktu



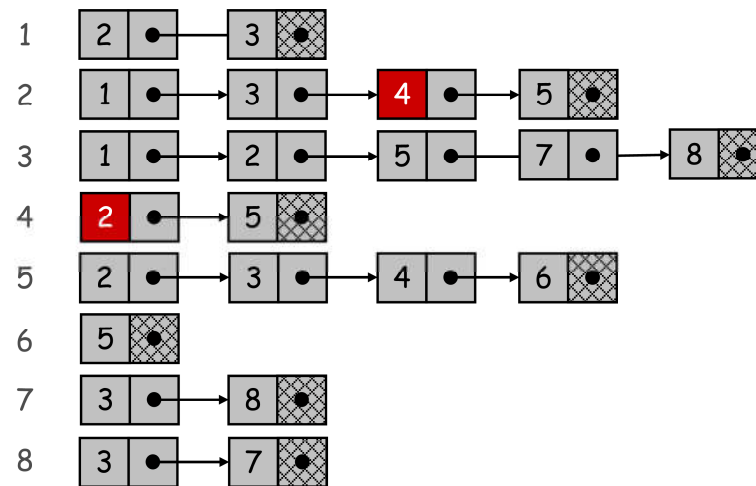
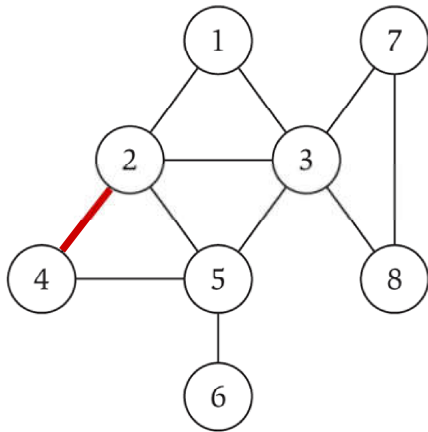
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0



Representasi Graf: Adjacency List

Adjacency List: node diindeks sebagai list

- Dua representasi untuk setiap sisi
- Ukuran ruang $m + n$
- Memeriksa apakah (u, v) edge membutuhkan $O(\text{deg}(u))$.
- Mengidentifikasi semua tepi membutuhkan $\Theta(m + n)$.





Kebutuhan Ruang

Teorema 1.0

Representasi *adjacency matrix* dari graf membutuhkan $O(n^2)$ ruang, sedangkan representasi *adjacency list* hanya membutuhkan $O(m + n)$ ruang.

Karena $m \leq n^2$, batas $O(m + n)$ tidak pernah lebih buruk dari $O(n^2)$; jauh lebih baik ketika graf-nya bersifat *sparse* (jarang), dengan m jauh lebih kecil dari n^2 .

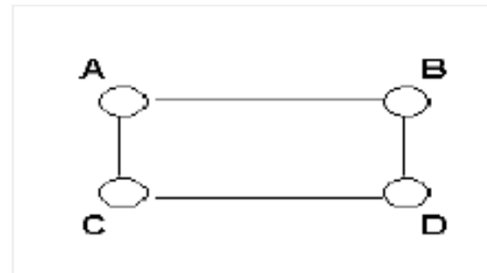


Derajat sebuah Node

- Kita mendefinisikan derajat n_v dari sebuah node v menjadi jumlah edge bersinggungan yang dimilikinya.
- Jumlah derajat dalam graf adalah jumlah yang sering muncul dalam analisis algoritma graf.

Teorema 2.0

$$\sum_{v \in V} n_v = 2m.$$



- $m = |V| = 4$
- Jml Derajat semua node = $2(4) = 8$
- *Jika derajat masing-masing node berjumlah genap disebut **Graf EULER**

Bukti. Setiap sisi $e = (v, w)$ memberikan kontribusi tepat dua kali untuk jumlah: sekali dalam kuantitas n_v dan sekali dalam kuantitas n_w . Oleh karena itu, jumlahnya adalah total dari kontribusi masing-masing sisi, itu adalah $2m$.

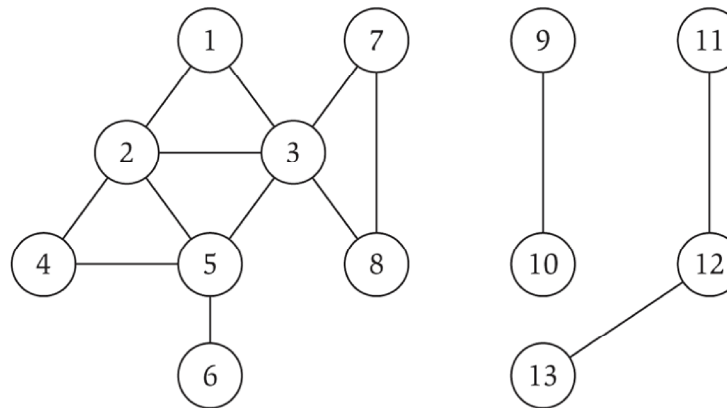


Path dan Konektivitas

Def. Path dalam graf tak berarah $G = (V, E)$ adalah urutan P dari node $v_1, v_2, \dots, v_{k-1}, v_k$ dengan properti yang setiap pasangan berurutan v_i, v_{i+1} digabungkan dengan sebuah edge di E .

Def. Path sederhana jika semua node berbeda.

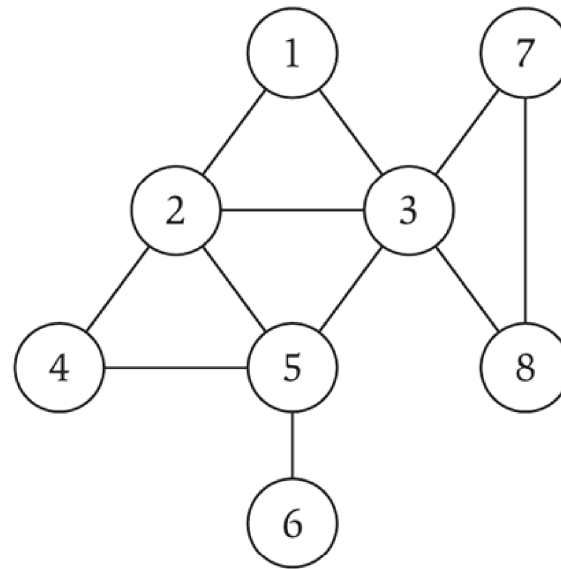
Def. Graf tidak berarah terhubung jika untuk setiap pasangan node u dan v , ada jalur antara u dan v





Cycle (Siklus)

Def. Siklus adalah jalur $v_1, v_2, \dots, v_{k-1}, v_k$ di mana $v_1 = v_k$, $k > 2$, dan node $k-1$ pertama semuanya berbeda.



cycle $C = 1, 2, 4, 5, 3, 1$



Tree (Pohon)

- **Def.** Sebuah grafik tak berarah disebut tree jika ia terhubung dan **tidak mengandung cycle**

Teorema 3.0

G merupakan graf tak berarah dengan node sebanyak n . Dua pernyataan berikut mengimplikasikan pernyataan ketiga.

- G terkoneksi
- G tidak mengandung cycle
- G memiliki edge sejumlah $n-1$

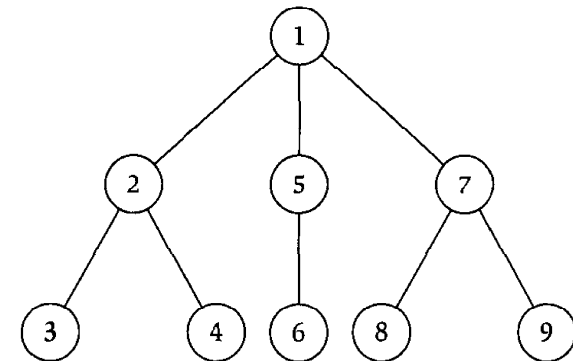
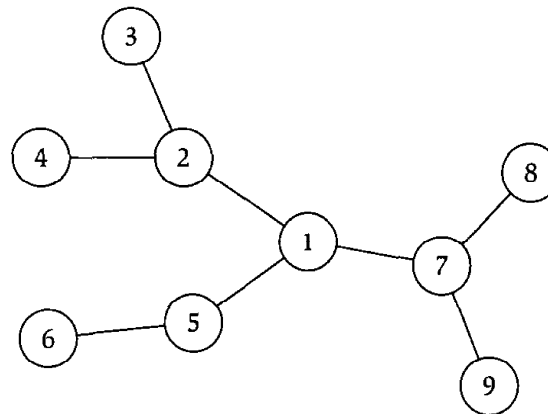


Figure 3.1 Two drawings of the same tree. On the right, the tree is rooted at node 1.



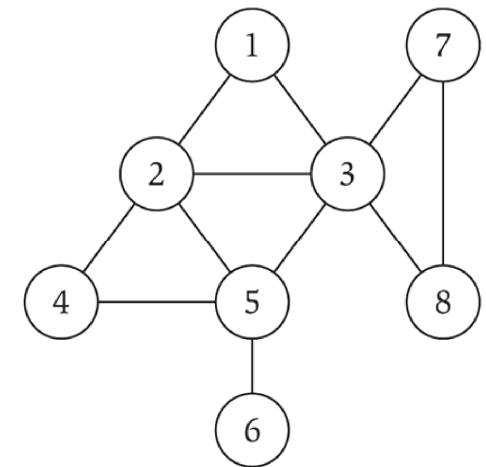
Konektivitas Graf

s-t connectivity problem. Diberikan dua node s dan t, apakah ada path antara s dan t?

s-t shortest path problem. Diberikan dua node s dan t, berapa panjang jalur terpendek antara s dan t?

Applications

- Social networks
- Maze traversal
- Fewest number of hops in a communication network





Breadth First Search

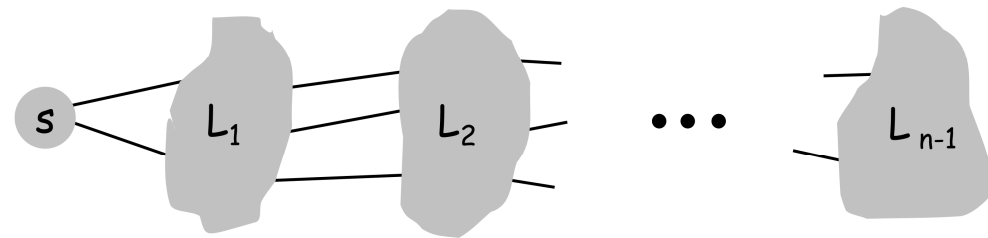


Breadth First Search

Intuisi BFS. Menjelajahi alur keluar dari s ke semua arah yang mungkin, tambahkan node satu "layer" sekaligus.

Algoritma BFS

- $L_0 = \{s\}$
- L_1 = semua tetangga dari L_0
- L_2 = semua node yang tidak termasuk ke dalam L_0 atau L_1 , dan yang mempunyai edge ke sebuah node di L_1
- $L_i + 1$ = semua node yang bukan milik layer sebelumnya, dan yang memiliki edge ke node di L_i



Teorema 4.0

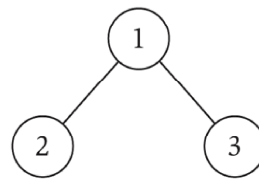
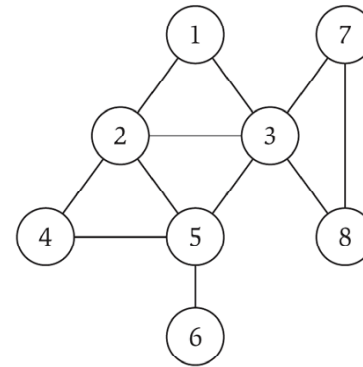
Untuk setiap i , L_i terdiri dari semua node pada jarak tepat ke i dari s . Ada path dari s ke t jika t muncul di beberapa layer



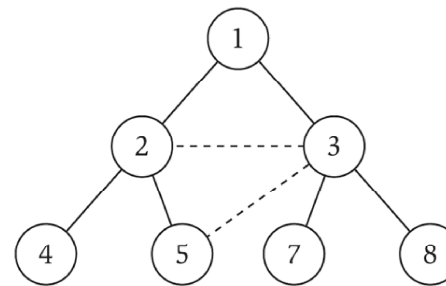
Breadth First Search (2)

Teorema 4.0

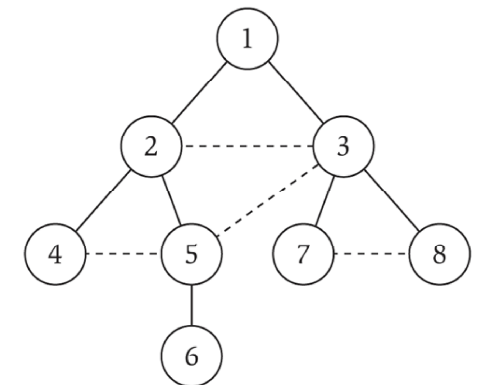
T adalah Tree BFS dari $G = (V, E)$, dan (x, y) adalah sebuah edge di G . Kemudian tingkat x dan y berbeda paling banyak 1



(a)



(b)



(c)

L_0

L_1

L_2

L_3



Implementasi BFS

- Adjacency list adalah representasi struktur data paling ideal untuk BFS
- Algoritma memeriksa setiap ujung yang meninggalkan node satu per satu. Ketika kita memindai edge yang meninggalkan u dan mencapai edge(u, v), kita perlu tahu apakah node v telah ditemukan sebelumnya oleh pencarian.
- Untuk menyederhanakan ini, kita maintain array yang ditemukan dengan panjang n dan mengatur $Discovered[v] = \text{true}$ segera setelah pencarian kita pertama kali melihat v . Algoritma BFS membangun lapisan node L_1, L_2, \dots , di mana L_i adalah set node pada jarak i dari sumber s .
- Untuk mengelola node dalam layer L_i , kami memiliki daftar $L[i]$ untuk setiap $i = 0, 1, 2, \dots$.



BFS(s):

Set Discovered[s] = true and Discovered[v] = false for all other v

Initialize $L[0]$ to consist of the single element s

Set the layer counter $i=0$

Set the current BFS tree $T=\emptyset$

While $L[i]$ is not empty

 Initialize an empty list $L[i+1]$

 For each node $u \in L[i]$

 Consider each edge (u, v) incident to u

 If Discovered[v] = false then

 Set Discovered[v] = true

 Add edge (u, v) to the tree T

 Add v to the list $L[i+1]$

 Endif

 Endfor

 Increment the layer counter i by one

Endwhile



Analisis BFS

Teorema 5.0

Implementasi BFS running dalam waktu $O(m + n)$ jika graf direpresentasikan oleh adjacency list.

Bukti

Mudah untuk membuktikan $O(n^2)$:

- Terdapat paling banyak n elemen di $L[i]$
- Setiap node muncul pada paling banyak satu list; untuk loop berjalan $\leq n$ kali
- ketika kita mempertimbangkan node u , ada $\leq n$ edge insiden (u, v) , dan kita menghabiskan $O(1)$ waktu memproses setiap sisi

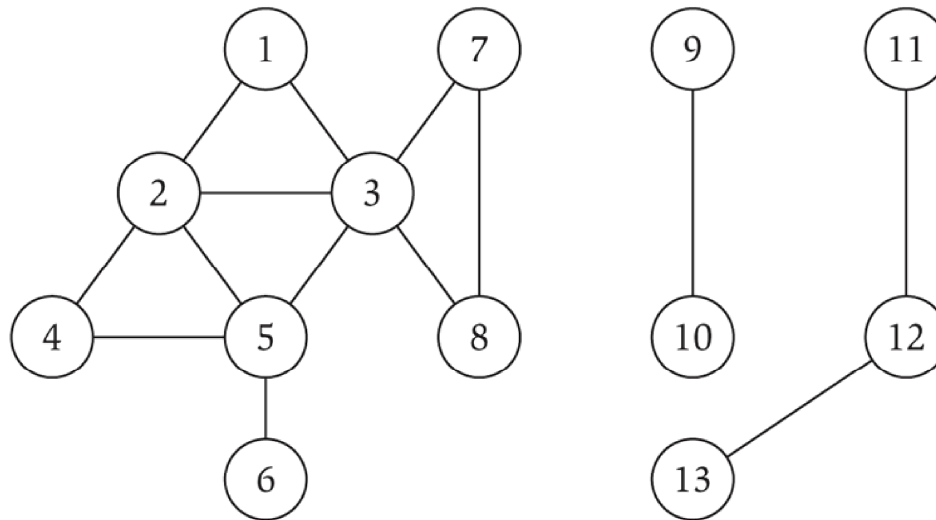
Sebenarnya berjalan dalam waktu $O(m + n)$:

- Ketika kita mempertimbangkan node u , ada tepi $\deg(u)$ tepi insiden (u, v)
- Total waktu pemrosesan edge adalah $\sum_{u \in V} \deg(u) = 2m$ (By Theorem 2.0)
- Kita perlu $O(n)$ waktu tambahan untuk mengatur list dan mengelola array



Connected Component

Connected component. Temukan semua node yang dapat dijangkau dari s



Connected component yang mengandung node 1 = { 1, 2, 3, 4, 5, 6, 7, 8 }

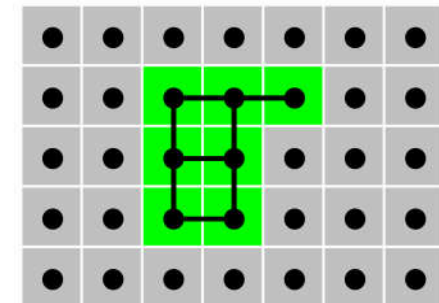
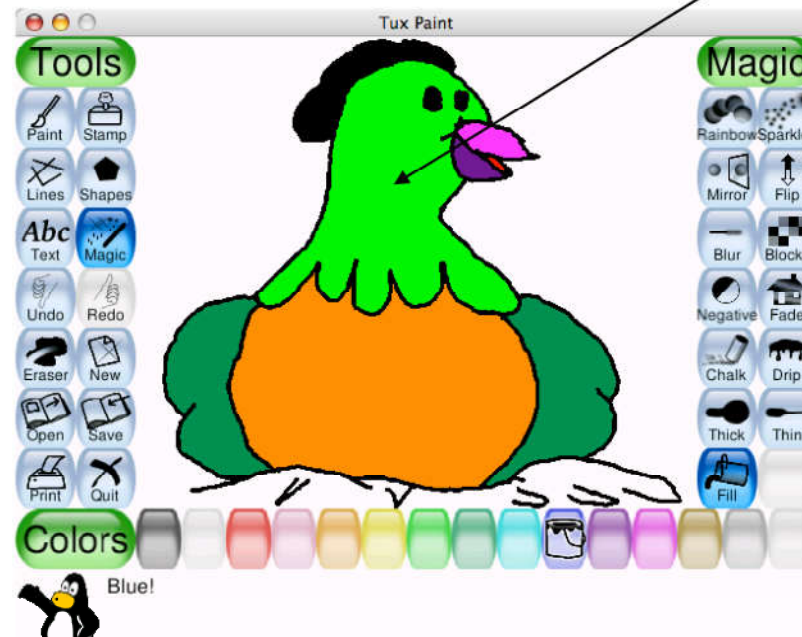


Implementation CC: Flood Fill

Flood Fill. Diberikan piksel hijau limau dalam sebuah gambar, ubah warna seluruh blob piksel tetangga menjadi biru.

- Node: pixel
- Edge: dua piksel hijau limau bertetangga
- Blob: komponen yang terhubung dari piksel hijau limau.

recolor lime green blob to blue



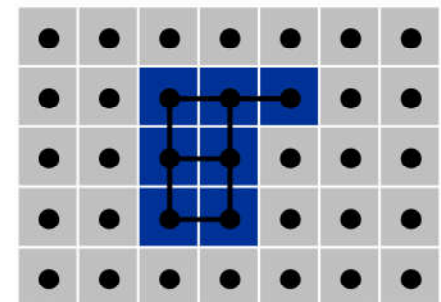


Implementation CC: Flood Fill

Flood Fill. Diberikan piksel hijau limau dalam sebuah gambar, ubah warna seluruh blob piksel tetangga menjadi biru.

- Node: pixel
- Edge: dua piksel hijau limau bertetangga
- Blob: komponen yang terhubung dari piksel hijau limau.

recolor lime green blob to blue





Analisa Connected Component

Connected component. Temukan semua node yang dapat dijangkau dari s

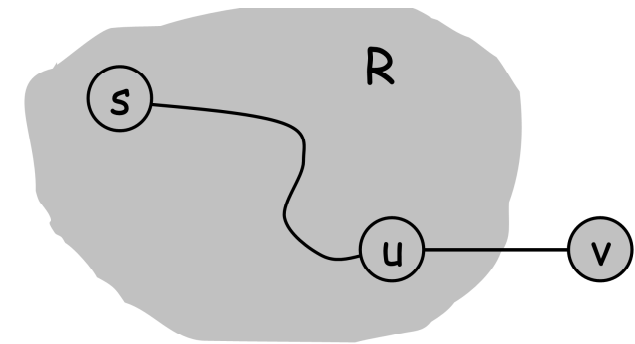
R will consist of nodes to which s has a path

Initially $R = \{s\}$

While there is an edge (u, v) where $u \in R$ and $v \notin R$

 Add v to R

Endwhile



it's safe to add v

Teorema 6.0.

Setelah penghentian, R adalah komponen terhubung yang mengandung s .

- BFS = menjelajah dalam urutan jarak dari s .
- DFS = jelajahi dengan cara yang berbeda.



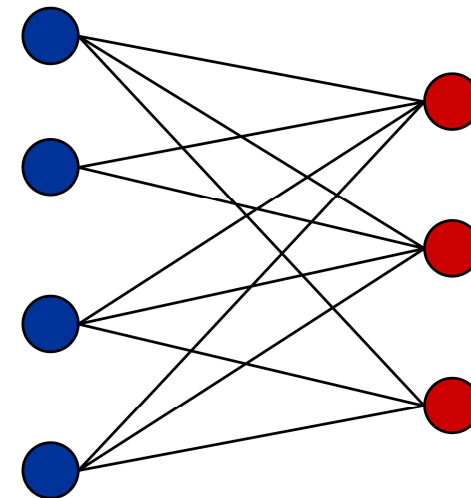
Uji Coba Bipatiteness: Aplikasi dari BFS

Graf Bipartit

Def. Graf tidak berarah $G = (V, E)$ adalah bipartit jika node bisa berwarna **merah** atau **biru** sedemikian rupa sehingga setiap ujung memiliki satu **merah** dan satu ujung **biru**.

Aplikasi

- Stable marriage: men = red, women = blue.
- Scheduling: machines = red, jobs = blue.



a bipartite graph



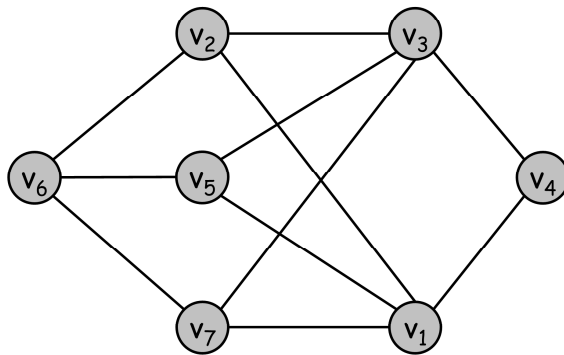
Uji Coba Bipatiteness

Menguji Bipartiteness. Diberi graf G , apakah bipartit?

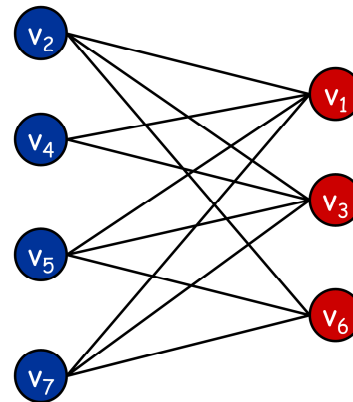
Banyak masalah graf menjadi:

- lebih mudah jika graf yang mendasarinya adalah bipartit (pencocokan)
- *tractable* jika grafik yang mendasarinya adalah bipartit (set independen)

Sebelum mencoba merancang suatu algoritma, kita perlu memahami struktur graf bipartit.



a bipartite graph G



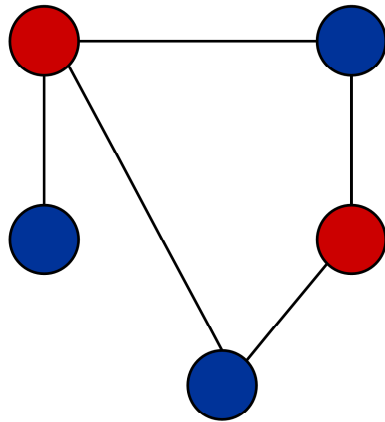
another drawing of G



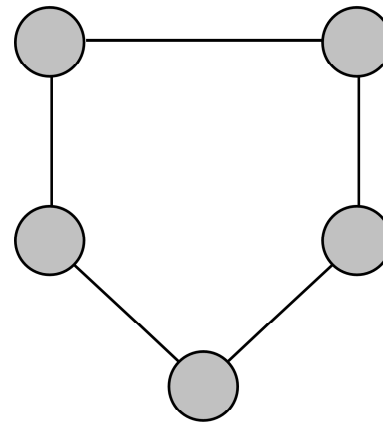
Obstruksi untuk Bipartiteness

Lemma 1.0. Jika graf G adalah bipartit, graf tidak dapat berisi siklus panjang ganjil.

Bukti. Tidak mungkin untuk 2-warna siklus ganjil, biarkan G .



bipartite
(2-colorable)



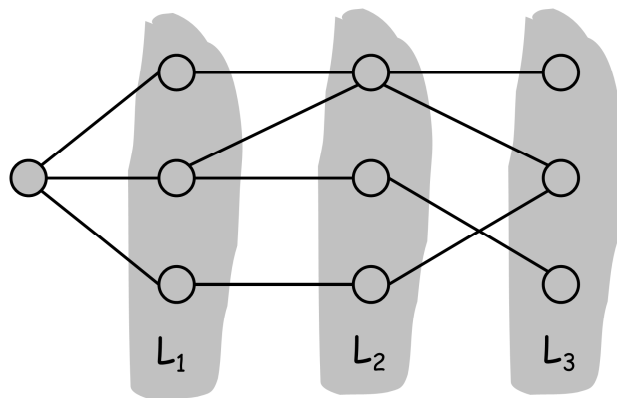
not bipartite
(not 2-colorable)



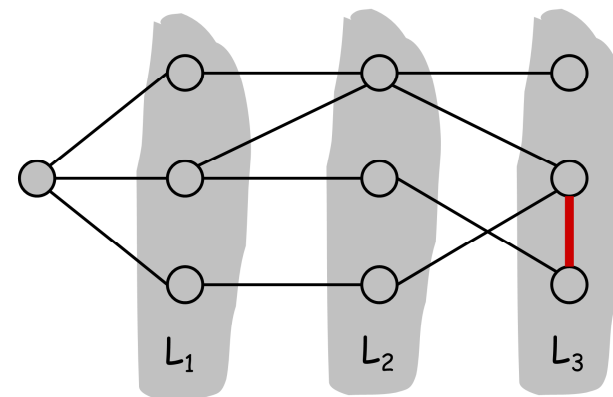
Graf Bipartit

Lemma 2.0 Biarkan G menjadi grafik yang terhubung, dan biarkan L_0, \dots, L_k menjadi lapisan diproduksi oleh BFS mulai dari node s . Secara tepat salah satu dari pernyataan berikut dipenuhi.

- (i) Tidak ada edge G yang bergabung dengan dua node pada lapisan yang sama, dan G adalah bipartit.
- (ii) Edge G bergabung dengan dua node dari lapisan yang sama, dan G berisi siklus panjang ganjil (dan karenanya bukan bipartit).



Case (i)



Case (ii)



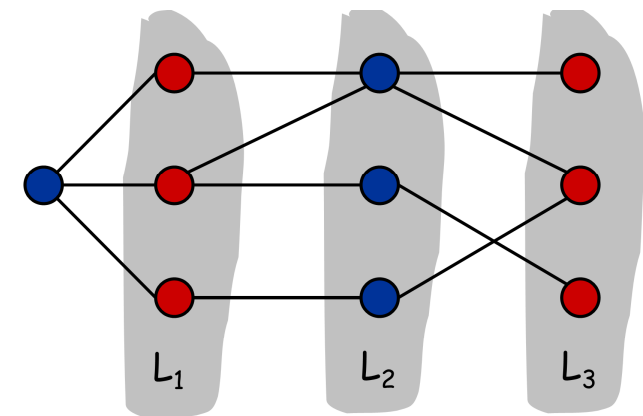
Graf Bipartit

Lemma 2.0 Biarkan G menjadi grafik yang terhubung, dan biarkan L_0, \dots, L_k menjadi lapisan diproduksi oleh BFS mulai dari node s . Secara tepat salah satu dari pernyataan berikut dipenuhi.

- (i) Tidak ada edge G yang bergabung dengan dua node pada lapisan yang sama, dan G adalah bipartit.
- (ii) Edge G bergabung dengan dua node dari lapisan yang sama, dan G berisi siklus panjang ganjil (dan karenanya bukan bipartit).

Bukti (i)

- Misalkan tidak ada tepi yang bergabung dengan dua node di lapisan yang sama.
- Dengan teorema 2.0, ini mengimplikasikan setiap edge bergabung dengan dua node di lapisan yang berdekatan.
- Bipartition: red = node pada layer ganjil, biru = node pada layer genap.



Case (i)



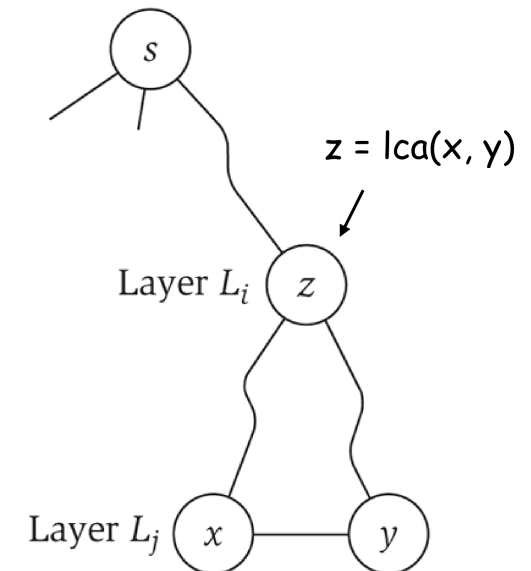
Graf Bipartit

Lemma 2.0 Biarkan G menjadi grafik yang terhubung, dan biarkan L_0, \dots, L_k menjadi lapisan diproduksi oleh BFS mulai dari node s . Secara tepat salah satu dari pernyataan berikut dipenuhi.

- (i) Tidak ada edge G yang bergabung dengan dua node pada lapisan yang sama, dan G adalah bipartit.
- (ii) Edge G bergabung dengan dua node dari lapisan yang sama, dan G berisi siklus panjang ganjil (dan karenanya bukan bipartit).

Bukti (ii)

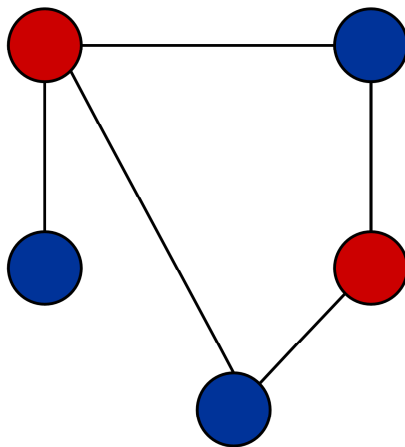
- Misalkan (x, y) adalah tepi dengan x, y di lapisan yang sama L_j .
- $z = \text{lca}(x, y) = \text{lowest common ancestor}$
- L_i menjadi level yang mengandung z
- Pertimbangkan siklus yang mengambil tepi dari x ke y , lalu path dari y ke z , lalu path dari z ke x .
- Panjangnya adalah $1 + \underbrace{(j-i)}_{\text{path from } y \text{ to } z} + \underbrace{(j-i)}_{\text{path from } z \text{ to } x}$, yang ganjil.



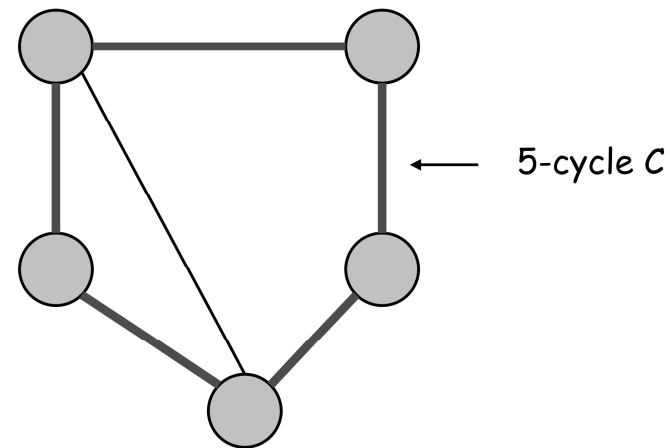


Obstruksi untuk Bipartitness

Corollary 1.0 Graf G adalah bipartit jika tidak mengandung siklus panjang ganjil.



bipartite
(2-colorable)



not bipartite
(not 2-colorable)



Depth First Search



Depth First Search

- Algoritma BFS muncul, khususnya, sebagai cara tertentu mengurutkan node yang kita kunjungi — dalam lapisan berurutan, berdasarkan pada jarak node lain dari s .
- Metode alami lain untuk menemukan node yang dapat dijangkau dari s adalah pendekatan yang mungkin Anda lakukan jika grafik G benar-benar sebuah labirin dari kamar yang saling berhubungan dan kita berjalan-jalan di dalamnya.
- Kita akan mulai dari s dan mencoba edge pertama yang mengarah ke node v . Kita kemudian akan mengikuti edge pertama yang mengarah keluar dari v , dan melanjutkan dengan cara ini sampai kita mencapai "jalan buntu" — sebuah node di mana Anda sudah menjelajahi semua tetangganya.
- Kita kemudian akan mundur sampai kita mencapai node dengan tetangga yang belum dijelajahi, dan melanjutkan dari sana.
- Kami menyebutnya Depth-first search (DFS), karena ini mengeksplorasi G dengan masuk sedalam mungkin dan hanya mundur jika diperlukan.



Depth First Search

- DFS juga merupakan implementasi khusus dari algoritma component-growing generik yang dijelaskan sebelumnya.
- Kami dapat memulai DFS dari titik awal mana pun tetapi mempertahankan pengetahuan global tentang node yang telah dieksplorasi.

DFS(u):

Mark u as "Explored" and add u to R

For each edge (u, v) incident to u

 If v is not marked "Explored" then

 Recursively invoke DFS(v)

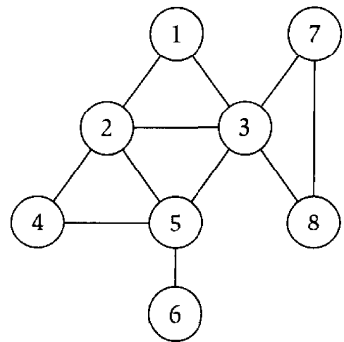
 Endif

Endfor

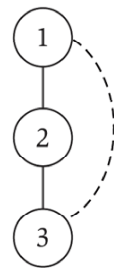


Depth First Search

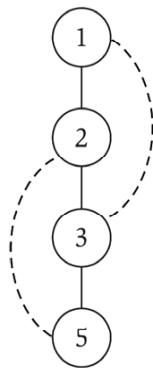
- Untuk menerapkan ini pada problem konektivitas s-t, kita cukup mendeklarasikan semua node pada awalnya untuk tidak dieksplorasi, dan memanggil DFS (s).
- Ada beberapa kesamaan dan beberapa perbedaan mendasar antara DFS dan BFS.
- Kesamaan didasarkan pada fakta bahwa mereka berdua membangun komponen terhubung yang mengandung s, dan bahwa mereka mencapai tingkat efisiensi yang serupa secara kualitatif.
- Sementara DFS akhirnya mengunjungi set node yang sama persis seperti BFS, ia biasanya melakukannya dalam urutan yang sangat berbeda; menyelidiki jalan panjang, berpotensi menjadi sangat jauh dari s, sebelum membuat cadangan untuk mencoba lebih dekat node yang belum dijelajahi.



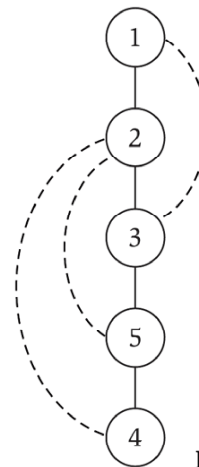
(a)



(b)

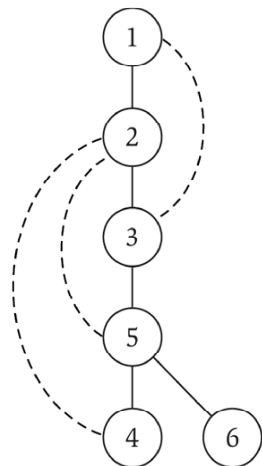


(c)

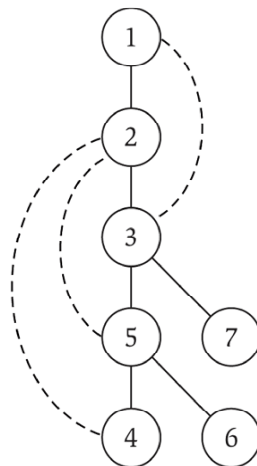


(d)

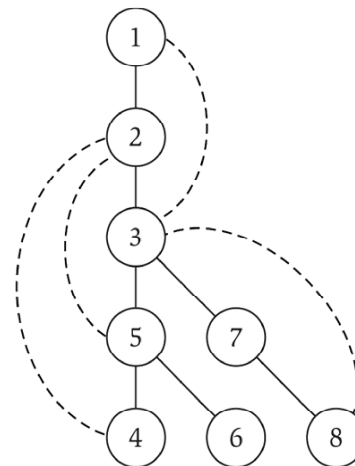
Figure 3.5 The construction of a depth-first search tree T for the graph in Figure 3.2, with (a) through (g) depicting the nodes as they are discovered in sequence. The solid edges are the edges of T ; the dotted edges are edges of G that do not belong to T .



(e)



(f)



(g)



Depth First Search

Teorema 7.0. T menjadi tree depth first search, misalkan x dan y menjadi node di T , dan biarkan (x, y) menjadi edge G yang bukan edge T . Lalu salah satu dari x atau y adalah ancestor lain.

Bukti. Misalkan (x, y) adalah edge G yang bukan merupakan edge T , dan anggaplah tanpa kehilangan keumuman bahwa x dicapai terlebih dahulu oleh algoritma DFS. Ketika edge (x, y) diperiksa selama pelaksanaan DFS (x), itu tidak ditambahkan ke T karena y ditandai "Eksplorasi." Karena y tidak ditandai "Eksplorasi" ketika DFS (x) pertama kali dipanggil, itu adalah node yang ditemukan antara pemanggilan dan akhir dari proses rekursif DFS (x). Karena itu, y adalah turunan dari x .



Himpunan Semua Connected Component

- Adakah komponen terhubung yang terkait dengan setiap node dalam grafik. Apa hubungan antara komponen-komponen ini?

Teorema 8.0

Untuk setiap dua node dalam graf, mereka terhubung komponennya bisa identik atau terputus-putus.



Implementasi DFS dengan Stack

DFS(s):

Initialize S to be a stack with one element s

While S is not empty

Take a node u from S

If Explored[u] = false then

Set Explored[u] = true

For each edge (u, v) incident to u

Add v to the stack S

Endfor

Endif

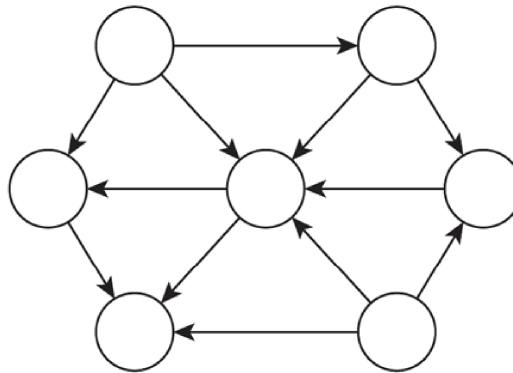
Endwhile



Konektivitas pada Graf Berarah

Grafik berarah. $G = (V, E)$

- Edge (u, v) berpindah dari node u ke node v .



Contoh. Grafik web - poin hyperlink dari satu halaman web ke halaman lainnya.

- Arah grafik sangat penting.
- Mesin pencari web modern mengeksploitasi struktur hyperlink untuk memeringkat web halaman menurut kepentingan.



Pencarian di Graf

Directed reachability. Diberikan node s , temukan semua node yang dapat dijangkau dari s .

Directed s-t shortest path problem. Diberikan dua node s dan t , berapa panjang jalur terpendek antara s dan t ?

Pencarian graf. BFS meluas secara alami ke graf berarah.

Web Crawler. Mulai dari halaman web s . Temukan semua halaman web yang ditautkan dari s , baik secara langsung maupun tidak langsung.



Konektivitas Kuat

Def. Node u dan v **saling dapat dijangkau (mutually reachable)** jika ada jalur dari u ke v dan juga jalur dari v ke u .

Def. Graf terhubung dengan kuat jika setiap pasangan node dapat dijangkau satu sama lain.

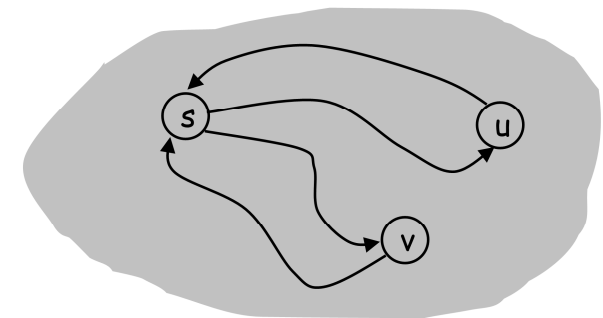
Lemma 3.0. s adalah sebuah node. G sangat terhubung (**strongly connected**) jika setiap node dapat dijangkau dari s , dan s dapat dijangkau dari setiap node.

Pf. \Rightarrow Mengikuti definisi.

Pf. \Leftarrow Jalur dari u ke v : menyatukan jalur $u-s$ dengan jalur $s-v$.

Jalur dari v ke u : menyatukan jalur $v-s$ dengan jalur $s-u$.

↖
ok if paths overlap



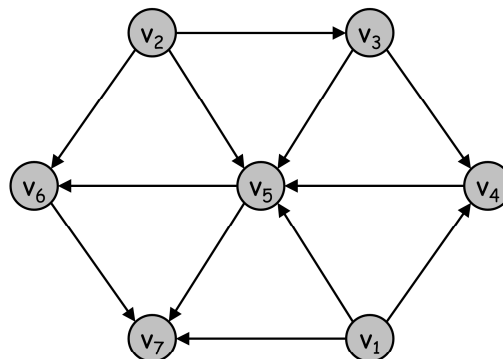


Directed Acyclic Graph (DAGs)

Def. DAG adalah graf terarah yang tidak mengandung siklus terarah.

Ex. Batasan Precedence: edge (v_i, v_j) berarti v_i harus mendahului v_j .

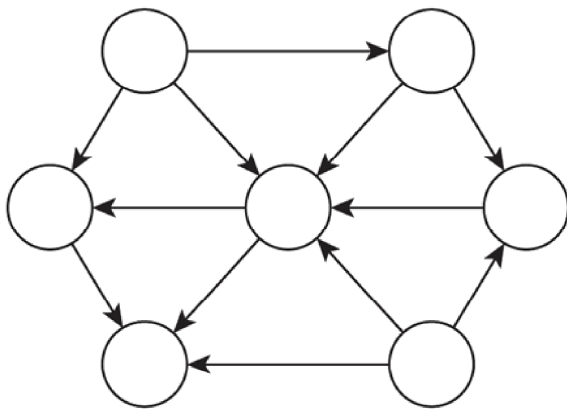
Def. Urutan topologi dari grafik berarah $G = (V, E)$ adalah urutan dari node-nya sebagai v_1, v_2, \dots, v_n sehingga untuk setiap sisi (v_i, v_j) kita memiliki $i < j$.



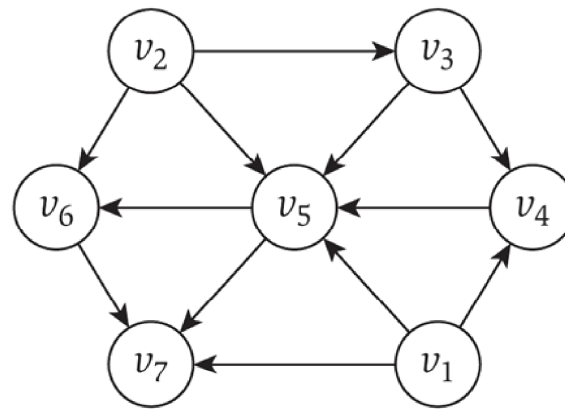
a DAG



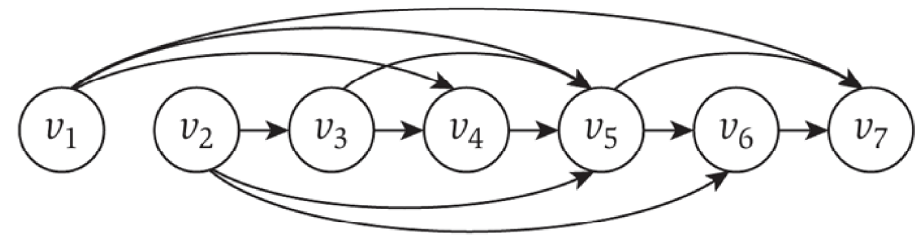
In a topological ordering, all edges point from left to right.



(a)



(b)



(c)

Figure 3.7 (a) A directed acyclic graph. (b) The same DAG with a topological ordering, specified by the labels on each node. (c) A different drawing of the same DAG, arranged so as to emphasize the topological ordering.



Prasyarat Utama Topological Ordering

Prasyarat utama. Edge (v_i, v_j) berarti tugas v_i harus terjadi sebelum v_j .

Contoh Aplikasi

- Graf prasyarat kursus: kursus v_i harus diambil sebelum v_j .
- Kompilasi modul: modul v_i harus dikompilasi sebelum modul v_j .
- Tahapan pekerjaan komputasi: output dari job v_i diperlukan untuk menjadi input pada pekerjaan v_j .

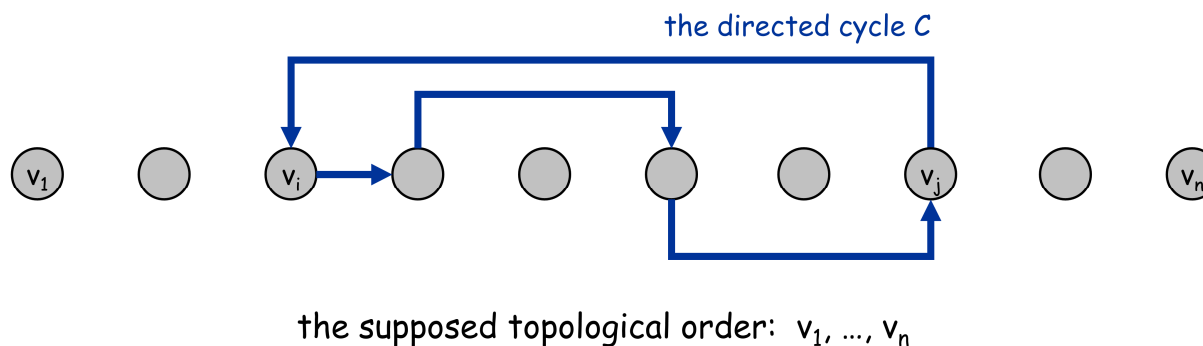


Directed Acyclic Graphs

Lemma 4.0 Jika G memiliki sebuah topological order, maka G adalah DAG.

Pf. (dengan kontradiksi)

- Misalkan G memiliki topological order v_1, \dots, v_n dan G memiliki cycle berarah C . Mari kita lihat yang terjadi
- Biarkan v_i menjadi node dengan indeks terendah pada C , dan biarkan v_j menjadi node pada C tepat sebelum v_i ; dengan demikian (v_j, v_i) adalah edge.
- Dengan pilihan kita terhadap i , kita mempunyai $i < j$
- Disisi lain, karena (v_j, v_i) adalah edge dan v_i, \dots, v_n adalah sebuah topological order, kita harus mempunyai $j < i$, **maka kontradiksi**.





Directed Acyclic Graphs

Lemma 4.0. Jika G memiliki sebuah topological order, maka G adalah DAG.

Critical Thinking

Q. Apakah setiap DAG memiliki sebuah topological order?

Q. Jika iya, bagaimana mengkomputasinya?

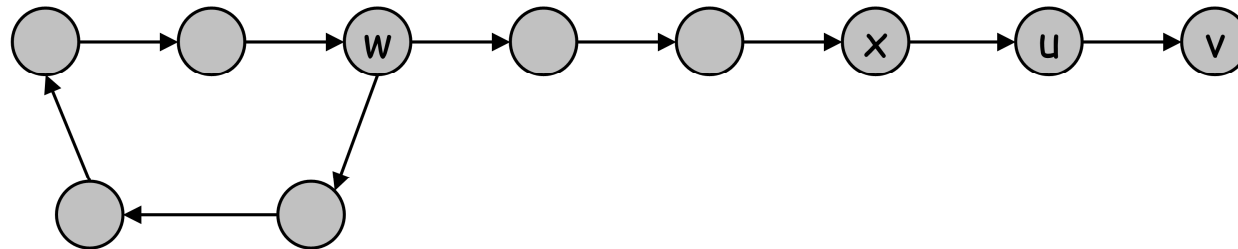


Directed Acyclic Graphs

Lemma 5.0. Jika G adalah DAG, maka G memiliki node tanpa edge yang masuk.

Pf. (dengan kontradiksi)

- Misalkan G adalah DAG dan setiap node memiliki setidaknya satu tepi yang masuk. Mari lihat apa yang terjadi.
- Pilih sembarang node v , dan mulailah mengikuti edge mundur dari v . Karena v memiliki setidaknya satu edge masuk (u, v) kita dapat berjalan mundur ke u .
- Kemudian, karena kita memiliki setidaknya satu tepi yang masuk (x, u) , kita dapat berjalan mundur ke x .
- Ulangi sampai kita mengunjungi sebuah node, katakanlah node w , dua kali.
- C menunjukkan urutan node yang ditemui di antara kunjungan berturut-turut ke w . C adalah sebuah siklus.





Directed Acyclic Graphs

Lemma 4.0. Jika G memiliki sebuah topological order, maka G adalah DAG.

Pf. (dengan induksi ke dalam n)

- Base case: true jika $n = 1$.
- Diberikan DAG dengan node $n > 1$, temukan sebuah node v dengan tidak ada edge yang masuk
- $G - \{v\}$ adalah sebuah DAG, karena menghapus v tidak dapat membuat siklus.
- Dengan hipotesis induktif, $G - \{v\}$ mempunyai sebuah topological ordering.
- Tempatkan v pertama di topological ordering, kemudian tambahkan nodes dari $G - \{v\}$ ke dalam topological order. Ini valid karena v tidak mempunyai arah edge yang masuk

To compute a topological ordering of G :

Find a node v with no incoming edges and order it first

Delete v from G

Recursively compute a topological ordering of $G - \{v\}$

and append this order after v



Topological Sorting Algorithm: Running Time

Teorema 6.0 Algoritma menemukan sebuah topological ordering memiliki running time $O(m+n)$

Pf.

- Kelola informasi berikut:
 - $\text{count}[w]$ = semua jumlah edge yang memiliki arah masuk
 - S = set node yang tidak memiliki arah edge masuk
- Inisialisasi: $O(m + n)$ melalui proses traversal ke seluruh graf
- Update: to delete v
 - remove v dari S
 - decrement $\text{count}[w]$ untuk semua edge dari v ke w , dan menambahkan w ke S jika $\text{count}[w]$ mencapai 0
 - membutuhkan $O(1)$ per edge



Topological Ordering

Dalam 1 DAG, topological ordering yang dihasilkan bisa lebih dari 1 untuk 1 sumber s. Hal ini berdasarkan slide 48 dan 49, sehingga topological ordering dari 1 DAG bisa banyak selama memenuhi prasyarat **lemma 4.0** dan **lemma 5.0**.



**ANY
QUESTIONS?**



Sesi Materi Berakhir
TERIMA KASIH