مروری بر پوشش‌های گراف

محمد تنهایی

پوشش گراف

گراف رایج‌ترین ساختار برای آزمون نرم‌افزار است.

گراف می‌تواند از منابع مختلفی بدست آید:

Control flow graphs

Design structure

FSMs and statecharts

Use cases

آزمون معمولاً به هدف پوشش گراف به شکلی مناسب استفاده می‌شود.

تعریف گراف

A set N of nodes, N is not empty

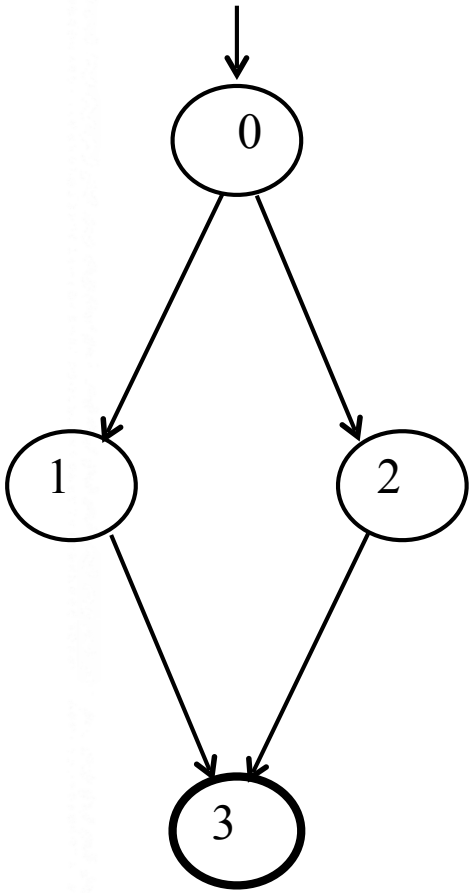
A set N_0 of initial nodes, N_0 is not empty

A set N_f of final nodes, N_f is not empty

A set E of edges, each edge from one node to another

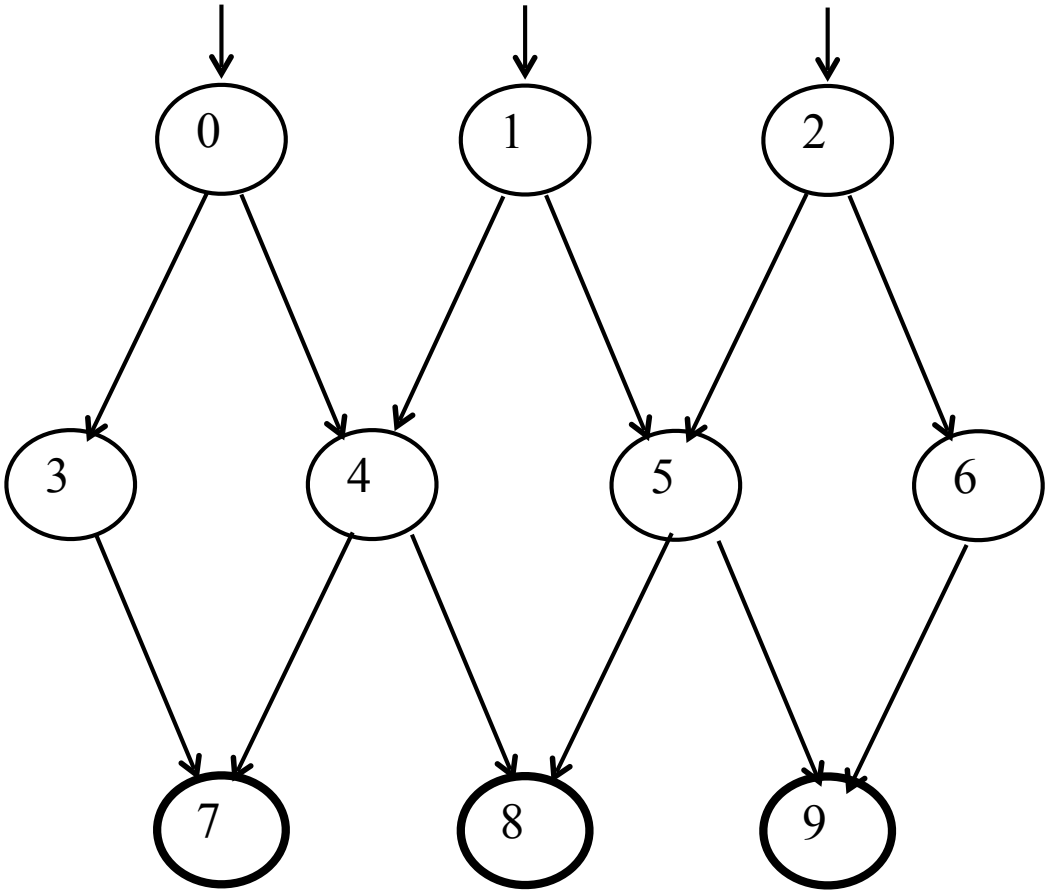
(n_i , n_j), i is predecessor, j is successor

سه مثال از گراف



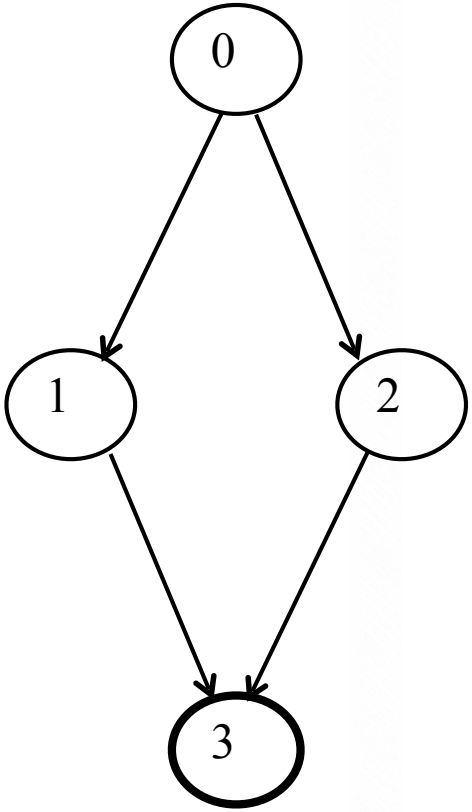
$N_0 = \{ 0 \}$

$N_f = \{ 3 \}$



$N_0 = \{ 0, 1, 2 \}$

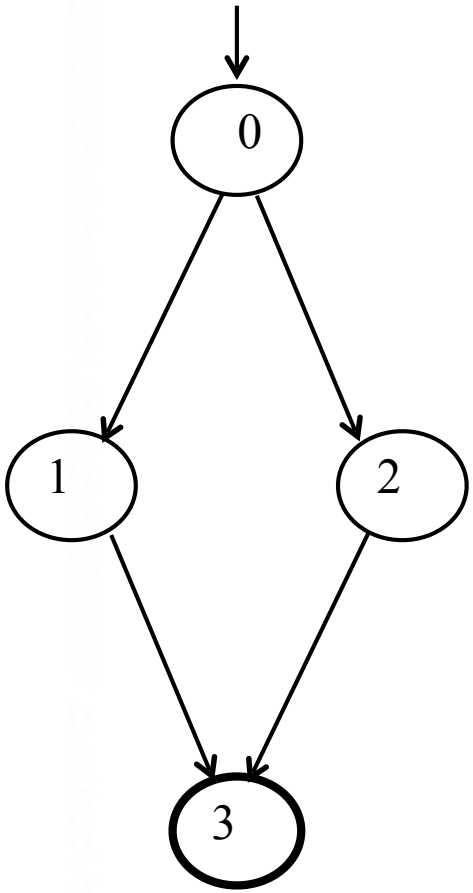
$N_f = \{ 7, 8, 9 \}$



$N_0 = \{ \}$

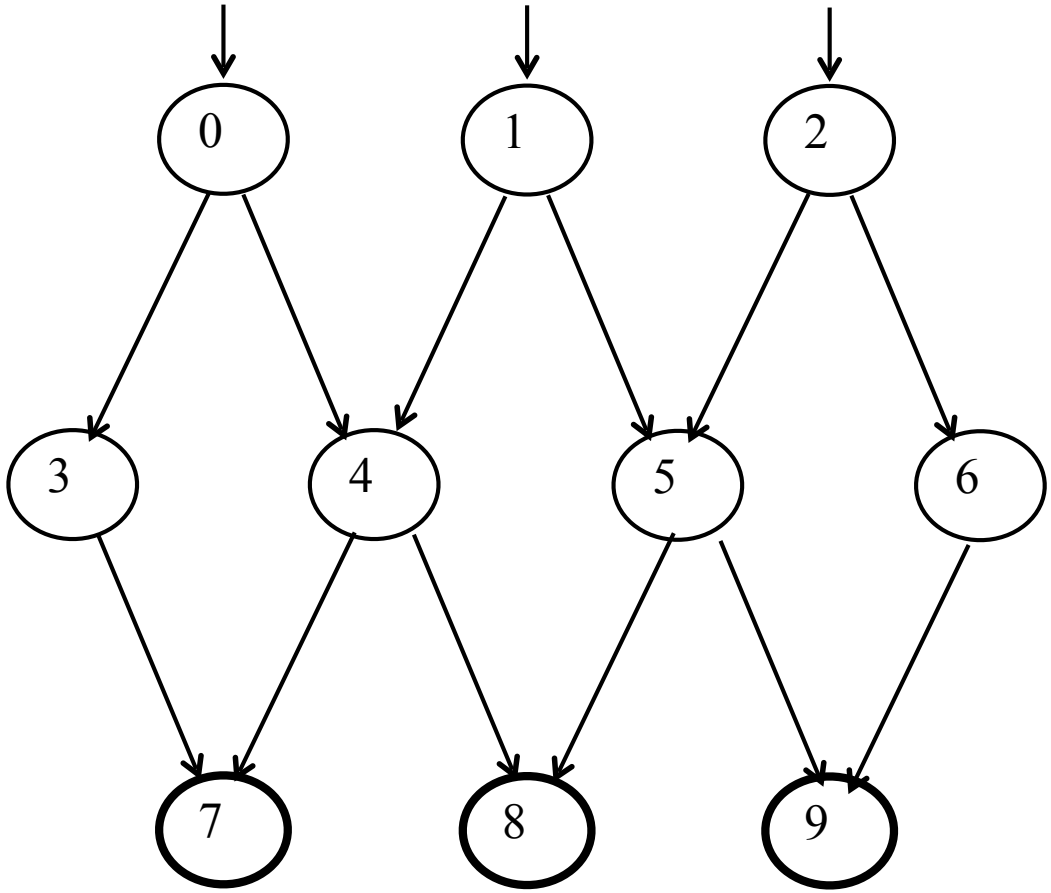
$N_f = \{ 3 \}$

سه مثال از گراف



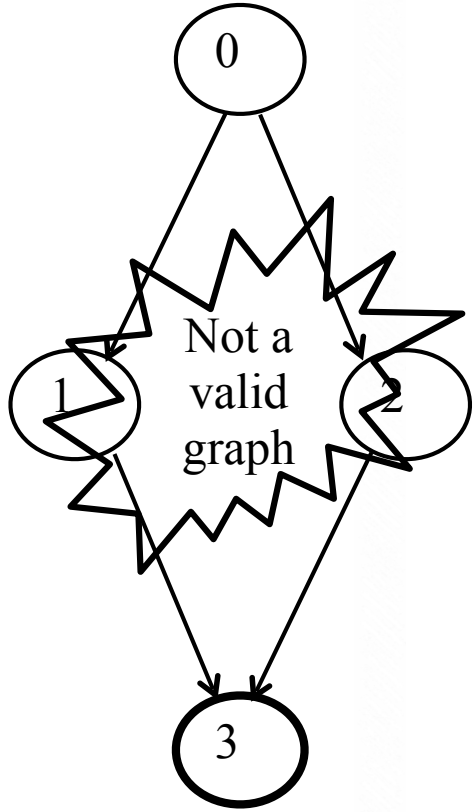
$N_0 = \{ 0 \}$

$N_f = \{ 3 \}$



$N_0 = \{ 0, 1, 2 \}$

$N_f = \{ 7, 8, 9 \}$



$N_0 = \{ \}$

$N_f = \{ 3 \}$

مسیرها در گراف

Path : A sequence of nodes – [n_1, n_2, ..., n_M]

Each pair of nodes is an edge

Length : The number of edges

A single node is a path of length 0

Subpath : A subsequence of nodes in p is a subpath of p

Reach (n) : Subgraph that can be reached from n

مسیرها در گراف

Path : A sequence of nodes – [n_1, n_2, ..., n_M]

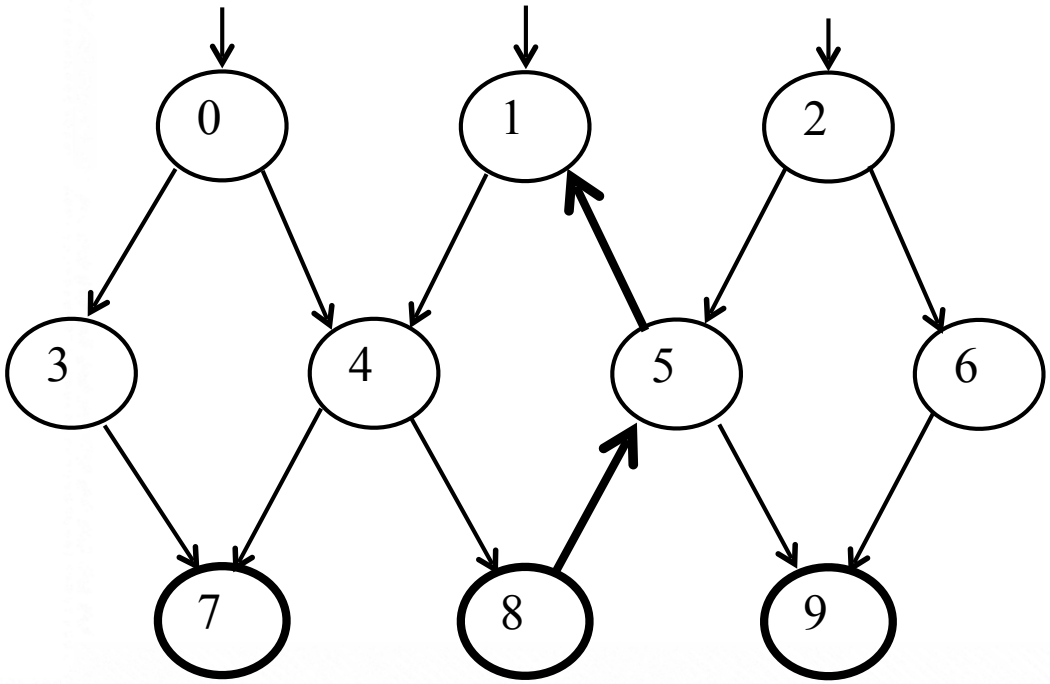
Each pair of nodes is an edge

Length : The number of edges

A single node is a path of length 0

Subpath : A subsequence of nodes in p is a subpath of p

Reach (n) : Subgraph that can be reached from n



مسیرها در گراف

Path : A sequence of nodes – [n_1, n_2, ..., n_M]

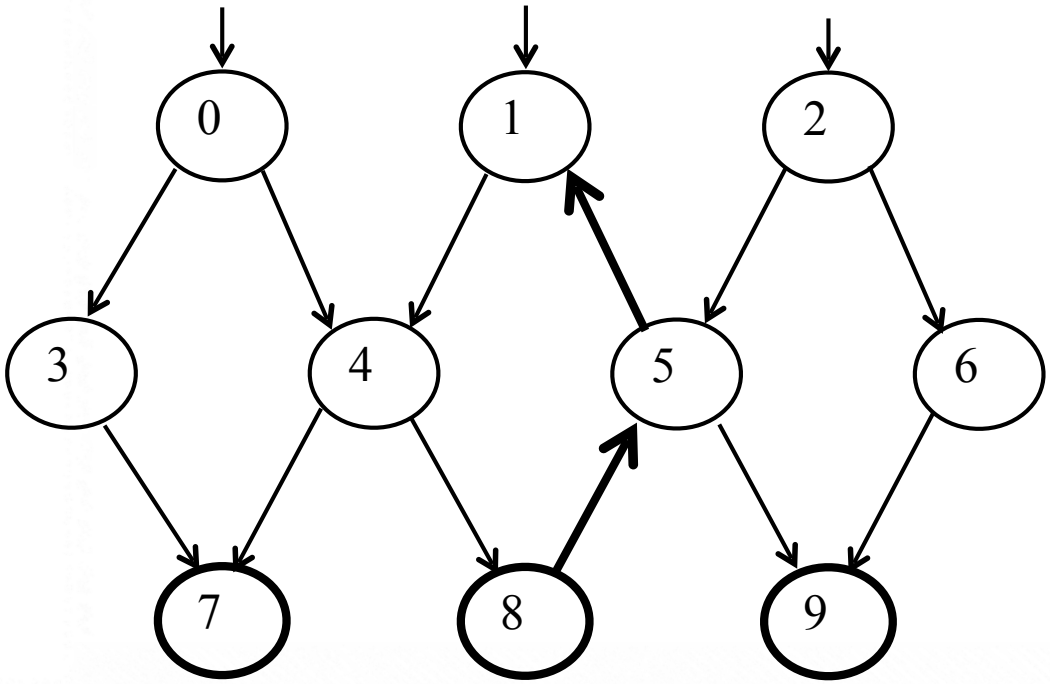
Each pair of nodes is an edge

Length : The number of edges

A single node is a path of length 0

Subpath : A subsequence of nodes in p is a subpath of p

Reach (n) : Subgraph that can be reached from n



Paths

[0, 3, 7]

[1, 4, 8, 5, 1]

[2, 6, 9]

مسیرها در گراف

Path : A sequence of nodes – [n_1, n_2, ..., n_M]

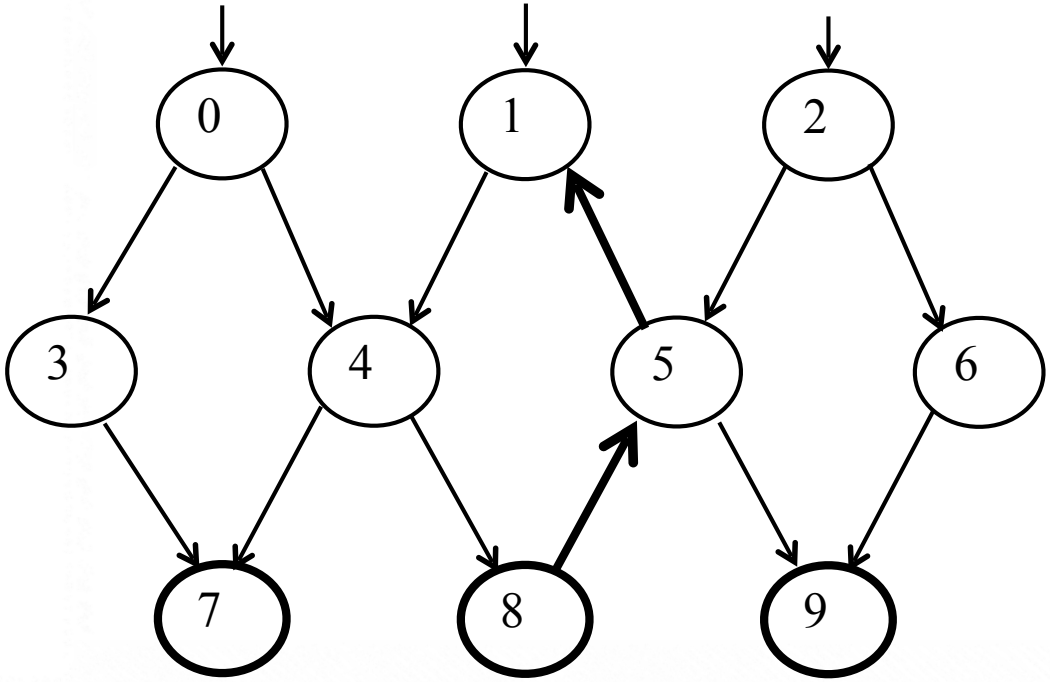
Each pair of nodes is an edge

Length : The number of edges

A single node is a path of length 0

Subpath : A subsequence of nodes in p is a subpath of p

Reach (n) : Subgraph that can be reached from n



Paths

[0, 3, 7]

[1, 4, 8, 5, 1]

[2, 6, 9]

Reach (0) = { 0, 3, 4, 7, 8, 5, 1, 9 }

Reach ({0, 2}) = G

Reach([2,6]) = {2, 6, 9}

Test Paths and SESEs

Test Path : یک مسیر که از گره آغازین شروع و به گره پایانی ختم می‌شود.

مسیر آزمون، نمایان‌کننده اجرای موارد آزمون است:

برخی از مسیرهای آزمون را می‌توان توسط چند آزمون اجرا کرد.

برخی از مسیرهای آزمون را نمی‌توان توسط هیچ آزمونی اجرا کرد.

SESE graphs : تمامی مسیرها از یک گره شروع و در یک گره دیگر تمام می‌شوند.

Single-entry, single-exit

N0 and Nf have exactly one node

Test Paths and SESEs

Test Path : یک مسیر که از گره آغازین شروع و به گره پایانی ختم می‌شود.

مسیر آزمون، نمایان‌کننده اجرای موارد آزمون است:

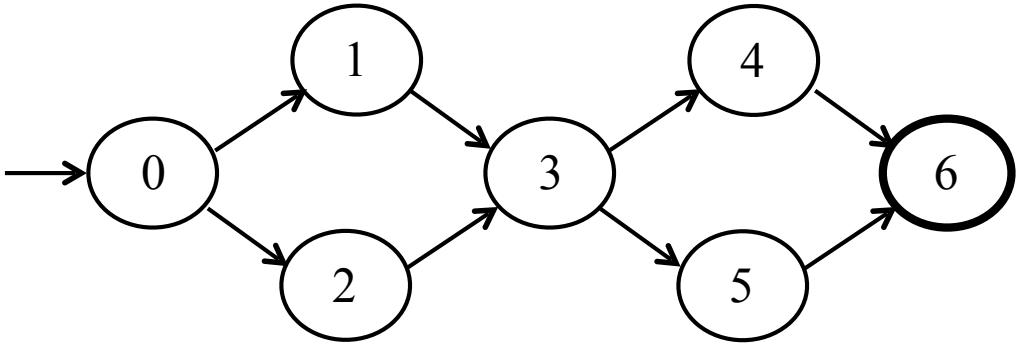
برخی از مسیرهای آزمون را می‌توان توسط چند آزمون اجرا کرد.

برخی از مسیرهای آزمون را نمی‌توان توسط هیچ آزمونی اجرا کرد.

SESE graphs : تمامی مسیرها از یک گره شروع و در یک گره دیگر تمام می‌شوند.

Single-entry, single-exit

N0 and Nf have exactly one node



Test Paths and SESEs

Test Path : یک مسیر که از گره آغازین شروع و به گره پایانی ختم می‌شود.

مسیر آزمون، نمایان‌کننده اجرای موارد آزمون است:

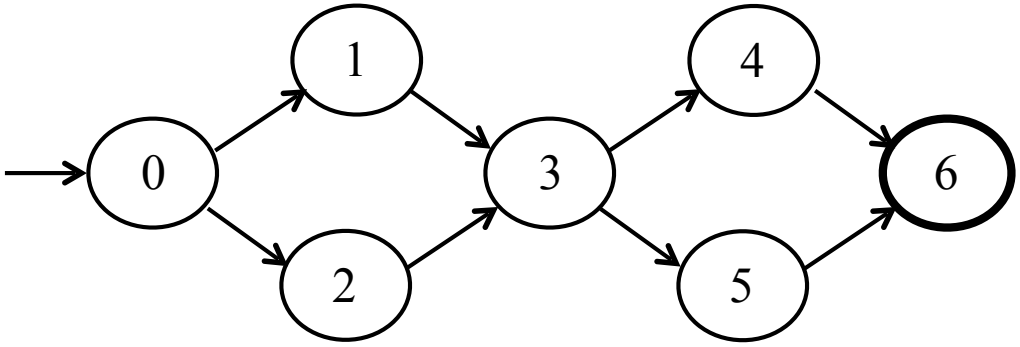
برخی از مسیرهای آزمون را می‌توان توسط چند آزمون اجرا کرد.

برخی از مسیرهای آزمون را نمی‌توان توسط هیچ آزمونی اجرا کرد.

SESE graphs : تمامی مسیرها از یک گره شروع و در یک گره دیگر تمام می‌شوند.

Single-entry, single-exit

N0 and Nf have exactly one node



Double-diamond graph

Four test paths

[0, 1, 3, 4, 6]

[0, 1, 3, 5, 6]

[0, 2, 3, 4, 6]

[0, 2, 3, 5, 6]

Visiting and Touring

Visit : A test path p *visits* node n if n is in p

A test path p *visits* edge e if e is in p

Tour : A test path p *tours* subpath q if q is a subpath of p

Path [0, 1, 3, 4, 6]

Visits nodes 0, 1, 3, 4, 6

Visits edges (0, 1), (1, 3), (3, 4), (4, 6)

Tours subpaths (0, 1, 3), (1, 3, 4), (3, 4, 6), (0, 1, 3, 4), (1, 3, 4, 6)

آزمون و مسیر آزمون

path (t) : The test path executed by test t

path (*T*) : The set of test paths executed by the set of tests T

Each test executes one and only one test path

A location in a graph (node or edge) can be reached from another location if there is a sequence of edges from the first location to the second

Syntactic reach : A subpath exists in the graph

Semantic reach : A test exists that can execute that subpath

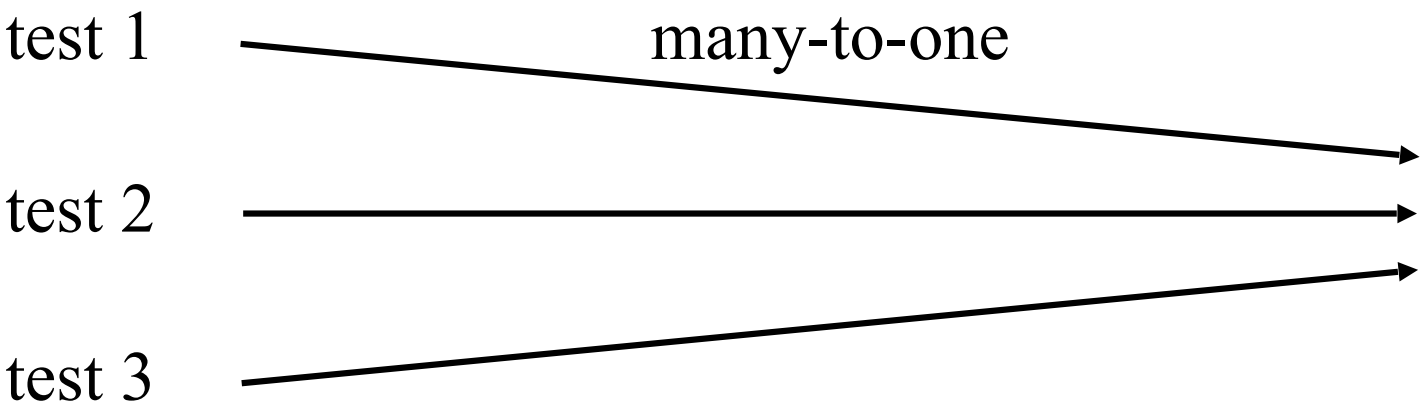
آزمون و مسیرهای آزمون

test 1

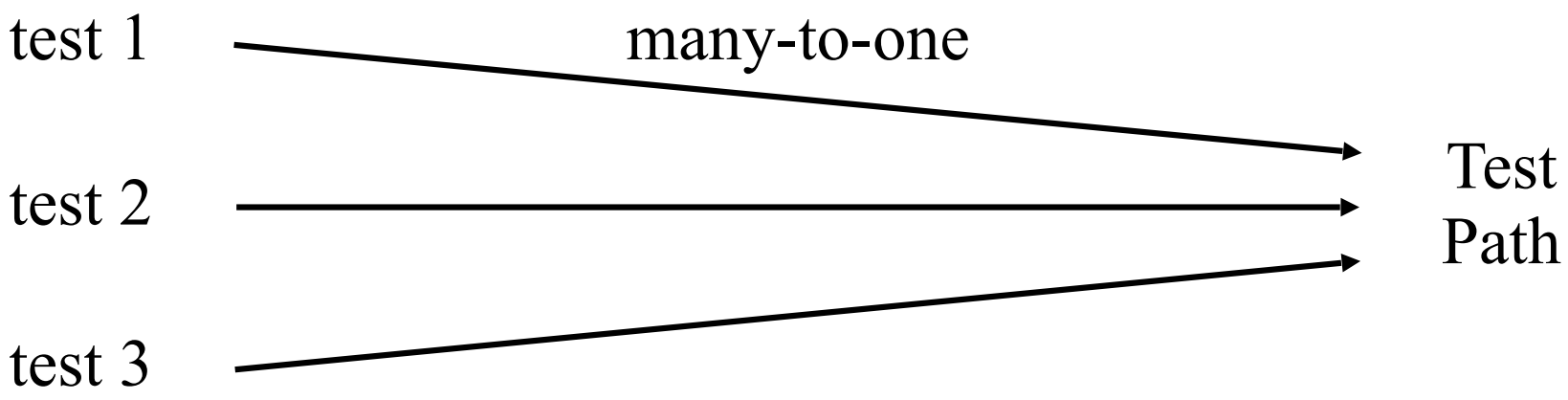
test 2

test 3

آزمون و مسیرهای آزمون

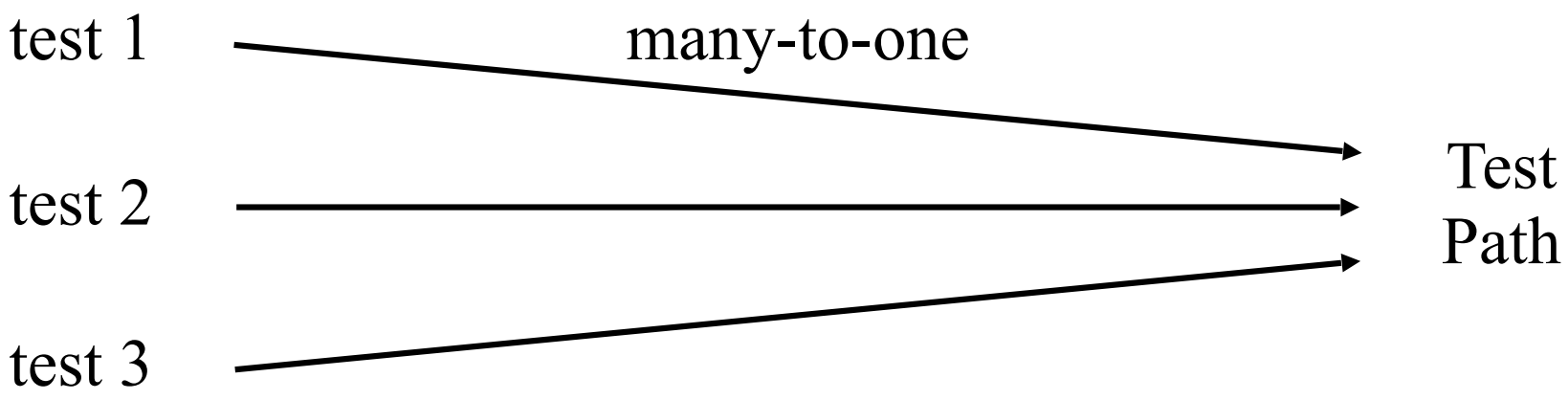


آزمون و مسیرهای آزمون



Deterministic software – a test always executes the same test path

آزمون و مسیرهای آزمون



Deterministic software – a test always executes the same test path

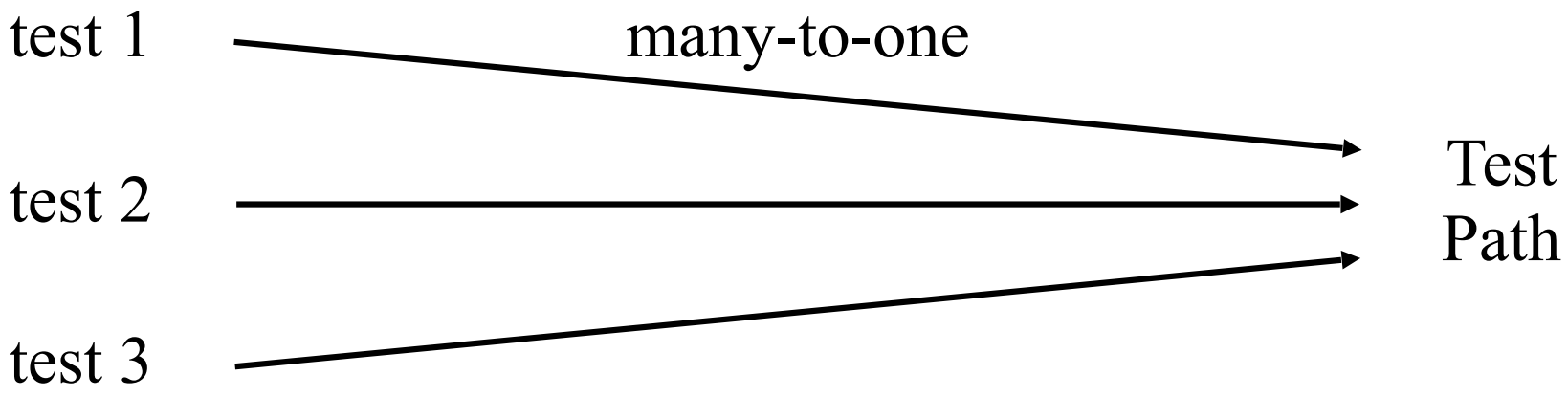
---

test 1

test 2

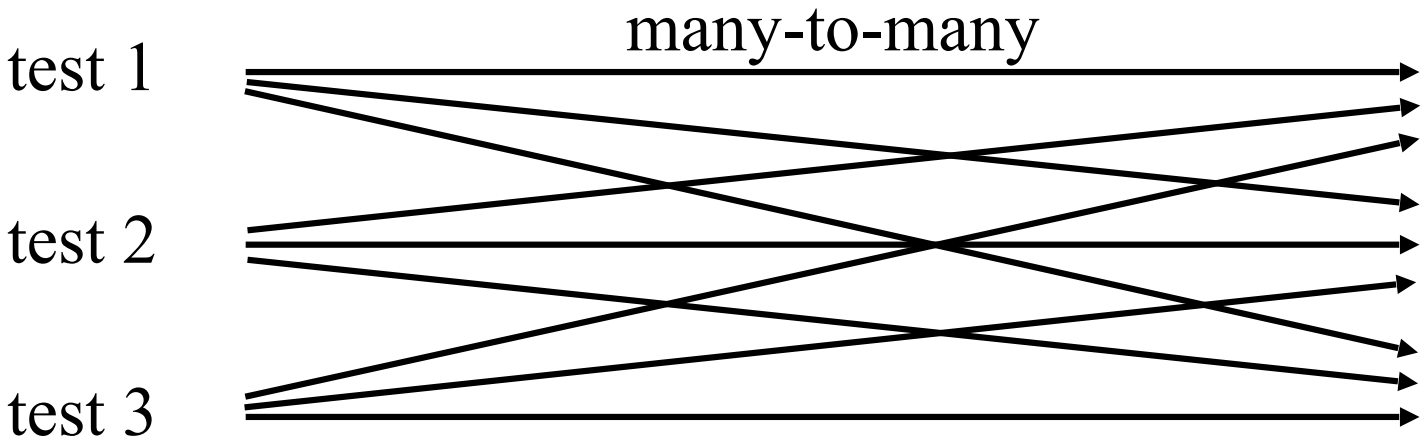
test 3

آزمون و مسیرهای آزمون

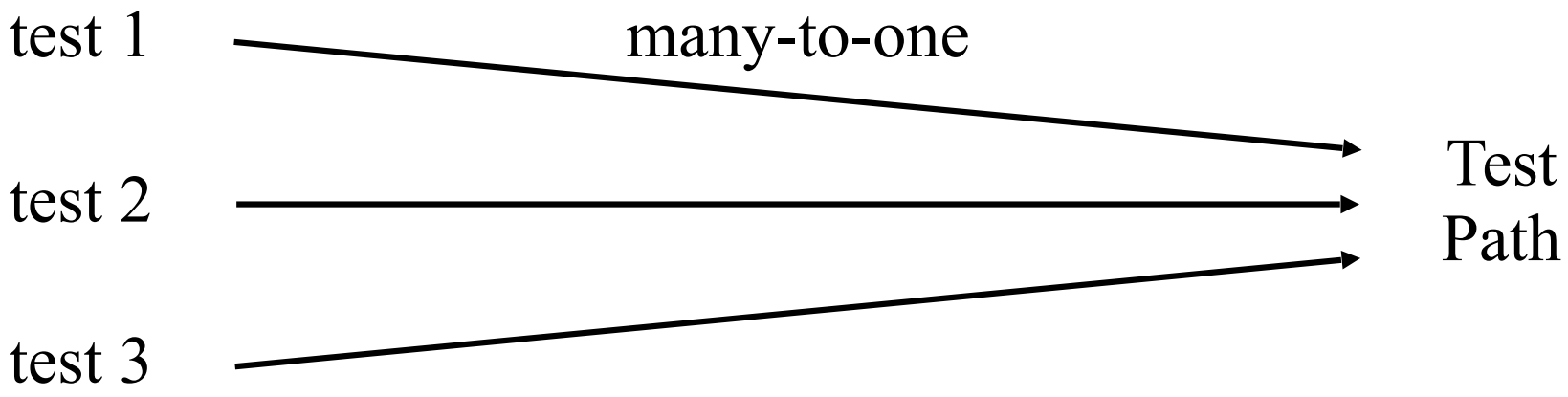


Deterministic software – a test always executes the same test path

---

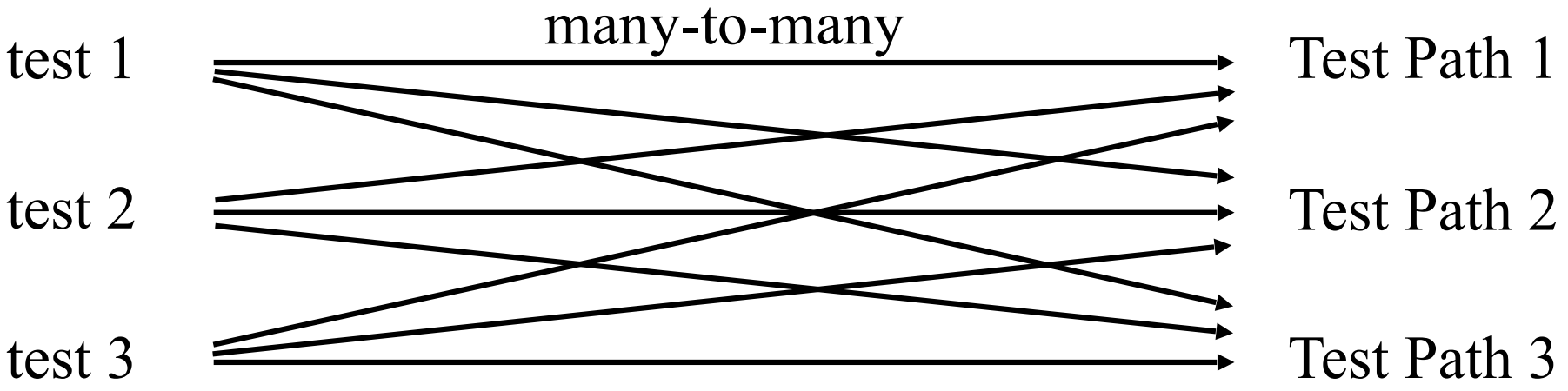


آزمون و مسیرهای آزمون



Deterministic software – a test always executes the same test path

---



Non-deterministic software – a test can execute different test paths

آزمون و پوشش گراف

- **استفاده ما از گراف در آزمون به صورت زیر است:**
  - **ایجاد یک مدل از نرم‌افزار به صورت گراف**
  - **تعریف معیارهایی برای پوشش برخی گره‌ها و یال‌ها و زیرمسیرها**

آزمون و پوشش گراف

- **استفاده ما از گراف در آزمون به صورت زیر است:**
  - **ایجاد یک مدل از نرم‌افزار به صورت گراف**
  - **تعریف معیارهایی برای پوشش برخی گره‌ها و یال‌ها و زیرمسیرها**

- Test Requirements (TR) : Describe properties of test paths
- Test Criterion : Rules that define test requirements
- Satisfaction : *Given a set TR of test requirements for a graph criterion C, a test set T satisfies C on graph G if and only if for every test requirement tr in TR, there is a test path p in path(T) such that p meets tr.*

آزمون و پوشش گراف

- **استفاده ما از گراف در آزمون به صورت زیر است:**
  - **ایجاد یک مدل از نرم‌افزار به صورت گراف**
  - **تعریف معیارهایی برای پوشش برخی گره‌ها و یال‌ها و زیرمسیرها**

- Test Requirements (TR) : Describe properties of test paths
- Test Criterion : Rules that define test requirements
- Satisfaction : *Given a set TR of test requirements for a graph criterion C, a test set T satisfies C on graph G if and only if for every test requirement tr in TR, there is a test path p in path(T) such that p meets tr.*
- Structural Coverage Criteria : Defined on a graph just in terms of nodes and edges
- Data Flow Coverage Criteria : Requires a graph to be annotated with references to variables

گره و پوشش یال

- اولین و ساده‌ترین معیار روی گراف، پوشش هر گره و هر یال گراف است.

گره و پوشش یال

- اولین و ساده‌ترین معیار روی گراف، پوشش هر گره و هر یال گراف است.

Node Coverage (NC) : Test set T satisfies node coverage on graph G iff for every syntactically reachable node n in N, there is some path p in $path(T)$ such that p visits n.

گره و پوشش یال

- **اولین و ساده‌ترین معیار روی گراف، پوشش هر گره و هر یال گراف است.**

Node Coverage (NC) : Test set T satisfies node coverage on graph G iff for every syntactically reachable node n in N, there is some path p in $path(T)$ such that p visits n.

- این تعریف اندکی شما رو گیج می‌کنه!! تعریف قشنگ‌تر به صورت زیر هست:

گره و پوشش یال

- **اولین و ساده‌ترین معیار روی گراف، پوشش هر گره و هر یال گراف است.**

Node Coverage (NC) : Test set T satisfies node coverage on graph G iff for every syntactically reachable node n in N, there is some path p in $path(T)$ such that p visits n.

- این تعریف اندکی شما رو گیج می‌کنه!! تعریف قشنگ‌تر به صورت زیر هست:

Node Coverage (NC) : TR contains each reachable node in G.

گره و پوشش یال

- پوشش همه یال‌ها اندکی قوی‌تر از پوشش گره‌ها است.

گره و پوشش یال

- پوشش همه یال‌ها اندکی قوی‌تر از پوشش گره‌ها است.

Edge Coverage (EC) : TR contains each reachable path of length up to 1, inclusive, in G.

گره و پوشش یال

- پوشش همه یال‌ها اندکی قوی‌تر از پوشش گره‌ها است.

Edge Coverage (EC) : TR contains each reachable path of length up to 1, inclusive, in G.

- The “length up to 1” allows for graphs with one node and no edges

گره و پوشش یال

- پوشش همه یال‌ها اندکی قوی‌تر از پوشش گره‌ها است.

Edge Coverage (EC) : TR contains each reachable path of length up to 1, inclusive, in G.

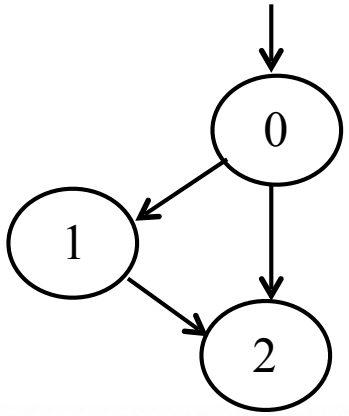
- The “length up to 1” allows for graphs with one node and no edges
- NC and EC are only different when there is an edge and another subpath between a pair of nodes (as in an “if-else” statement)

گره و پوشش یال

- پوشش همه یال‌ها اندکی قوی‌تر از پوشش گره‌ها است.

Edge Coverage (EC) : TR contains each reachable path of length up to 1, inclusive, in G.

- The “length up to 1” allows for graphs with one node and no edges
- NC and EC are only different when there is an edge and another subpath between a pair of nodes (as in an “if-else” statement)

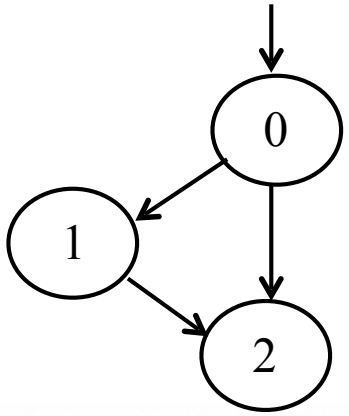


گره و پوشش یال

- پوشش همه یال‌ها اندکی قوی‌تر از پوشش گره‌ها است.

Edge Coverage (EC) : TR contains each reachable path of length up to 1, inclusive, in G.

- The “length up to 1” allows for graphs with one node and no edges
- NC and EC are only different when there is an edge and another subpath between a pair of nodes (as in an “if-else” statement)



Node Coverage : TR = { 0, 1, 2 }
Test Path = [0, 1, 2]

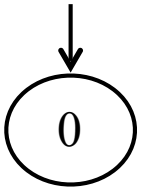
Edge Coverage : TR = { (0,1), (0, 2), (1, 2) }
Test Paths = [0, 1, 2] [0, 2]

مسیر به طول ۱ و ۰

- گراف با تنها یک گره، هیچ یالی ندارد.

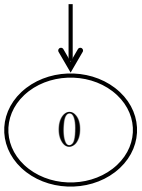
مسیر به طول ۰

- گراف با تنها یک گره، هیچ یالی ندارد.



مسیر به طول ۰

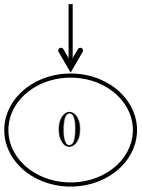
- گراف با تنها یک گره، هیچ یالی ندارد.



- It may be boring, but formally, Edge Coverage needs to require Node Coverage on this graph

مسیر به طول ۱و۰

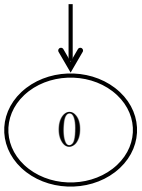
- گراف با تنها یک گره، هیچ یالی ندارد.



- It may be boring, but formally, Edge Coverage needs to require Node Coverage on this graph
- Otherwise, Edge Coverage will not subsume Node Coverage
  - So we define “length up to 1” instead of simply “length 1”

مسیر به طول ۱ و ۰

- گراف با تنها یک گره، هیچ یالی ندارد.

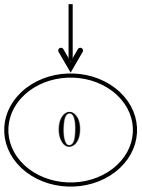


- It may be boring, but formally, Edge Coverage needs to require Node Coverage on this graph
- Otherwise, Edge Coverage will not subsume Node Coverage
  - So we define “length up to 1” instead of simply “length 1”

- We have the same issue with graphs that only have one edge – for Edge Pair Coverage …

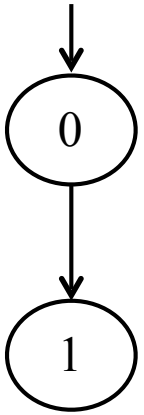
مسیر به طول ۱ و ۰

- گراف با تنها یک گره، هیچ یالی ندارد.



- It may be boring, but formally, Edge Coverage needs to require Node Coverage on this graph
- Otherwise, Edge Coverage will not subsume Node Coverage
  - So we define "length up to 1" instead of simply "length 1"

- We have the same issue with graphs that only have one edge – for Edge Pair Coverage …



پوشش چند یاله

- **Edge-pair coverage requires pairs of edges, or subpaths of length 2**

پوشش چند یاله

- **Edge-pair coverage requires pairs of edges, or subpaths of length 2**

Edge–Pair Coverage (EPC) : TR contains each reachable path of length up to 2, inclusive, in G.

پوشش چند یاله

- **Edge-pair coverage requires pairs of edges, or subpaths of length 2**

Edge–Pair Coverage (EPC) : TR contains each reachable path of length up to 2, inclusive, in G.

- The “length up to 2” is used to include graphs that have less than 2 edges

پوشش چند یاله

- **Edge-pair coverage requires pairs of edges, or subpaths of length 2**

Edge–Pair Coverage (EPC) : TR contains each reachable path of length up to 2, inclusive, in G.

- The “length up to 2” is used to include graphs that have less than 2 edges
- The logical extension is to require all paths …

پوشش چند یاله

- **Edge-pair coverage requires pairs of edges, or subpaths of length 2**

Edge-Pair Coverage (EPC) : TR contains each reachable path of length up to 2, inclusive, in G.

- The "length up to 2" is used to include graphs that have less than 2 edges
- The logical extension is to require all paths …

Complete Path Coverage (CPC) : TR contains all paths in G.

پوشش چند یاله

- **Edge-pair coverage requires pairs of edges, or subpaths of length 2**

Edge–Pair Coverage (EPC) : TR contains each reachable path of length up to 2, inclusive, in G.

- The "length up to 2" is used to include graphs that have less than 2 edges
- The logical extension is to require all paths …

Complete Path Coverage (CPC) : TR contains all paths in G.

- Unfortunately, this is impossible if the graph has a loop, so a weak compromise is to make the tester decide which paths:

پوشش چند یاله

- **Edge-pair coverage requires pairs of edges, or subpaths of length 2**

Edge–Pair Coverage (EPC) : TR contains each reachable path of length up to 2, inclusive, in G.

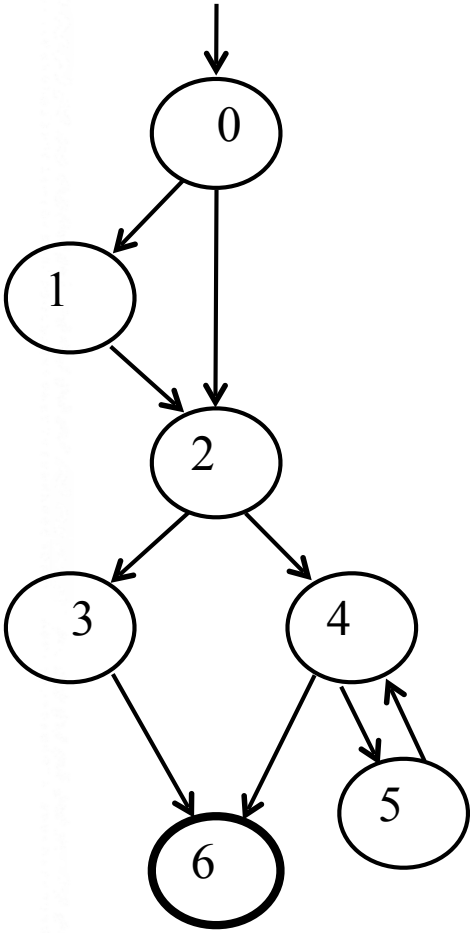
- The "length up to 2" is used to include graphs that have less than 2 edges
- The logical extension is to require all paths …

Complete Path Coverage (CPC) : TR contains all paths in G.

- Unfortunately, this is impossible if the graph has a loop, so a weak compromise is to make the tester decide which paths:

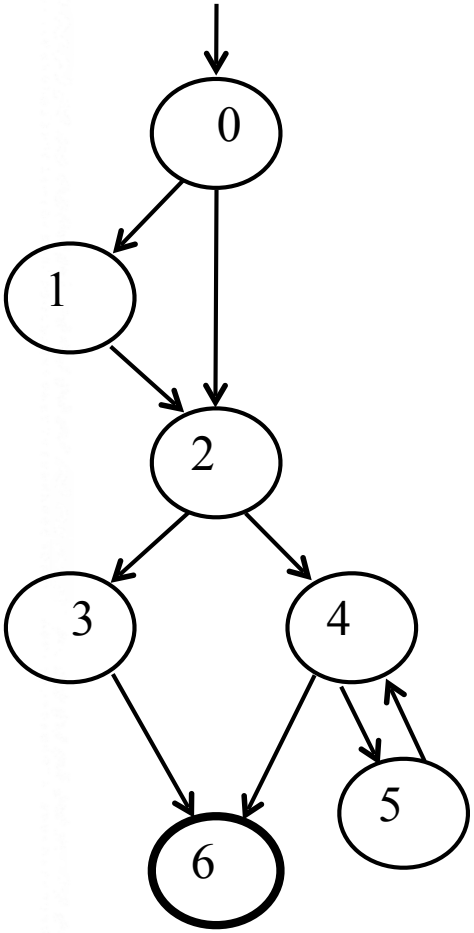
Specified Path Coverage (SPC) : TR contains a set S of test paths, where S is supplied as a parameter.

مثال از پوشش ساختار

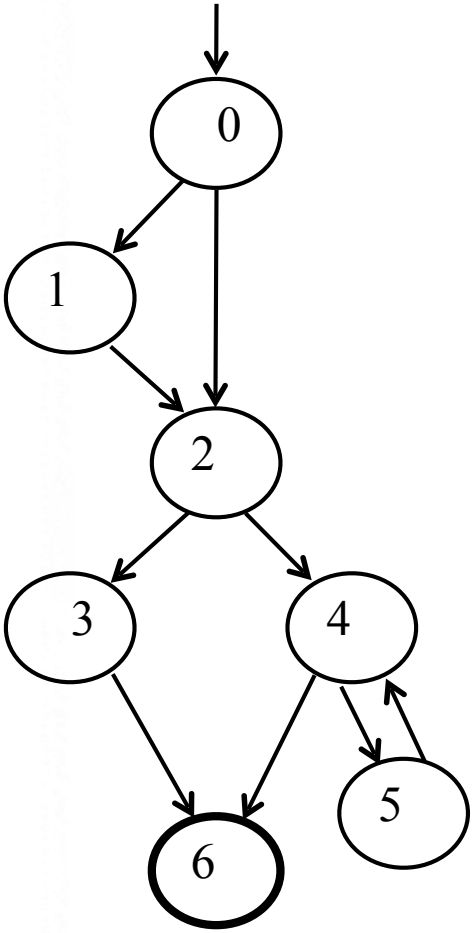


مثال از پوشش ساختار

Node Coverage
TR = { 0, 1, 2, 3, 4, 5, 6 }
Test Paths: [0, 1, 2, 3, 6] [0, 1, 2, 4, 5, 4, 6]



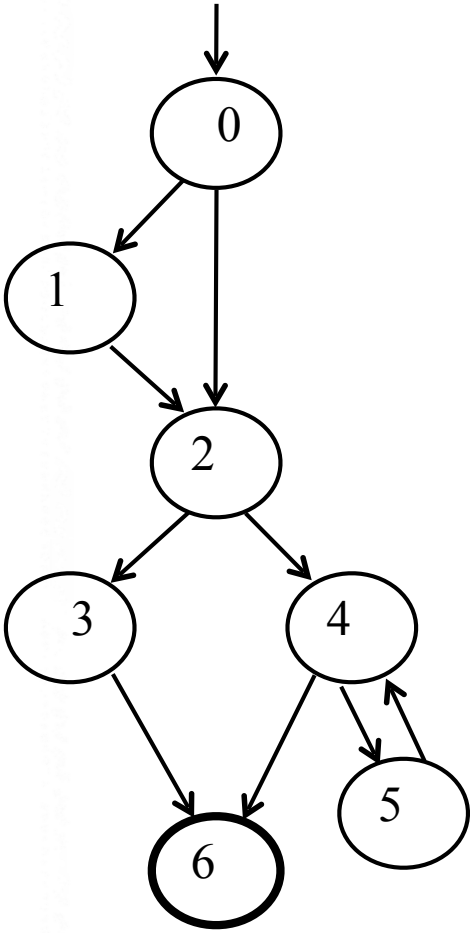
مثال از پوشش ساختار



Node Coverage
TR = { 0, 1, 2, 3, 4, 5, 6 } Test Paths: [0, 1, 2, 3, 6] [0, 1, 2, 4, 5, 4, 6]

Edge Coverage
TR = { (0,1), (0,2), (1,2), (2,3), (2,4), (3,6), (4,5), (4,6), (5,4) } Test Paths: [0, 1, 2, 3, 6] [0, 2, 4, 5, 4, 6]

مثال از پوشش ساختار



Node Coverage

TR = { 0, 1, 2, 3, 4, 5, 6 }

Test Paths: [0, 1, 2, 3, 6] [0, 1, 2, 4, 5, 4, 6]

Edge Coverage

TR = { (0,1), (0,2), (1,2), (2,3), (2,4), (3,6), (4,5), (4,6), (5,4) }

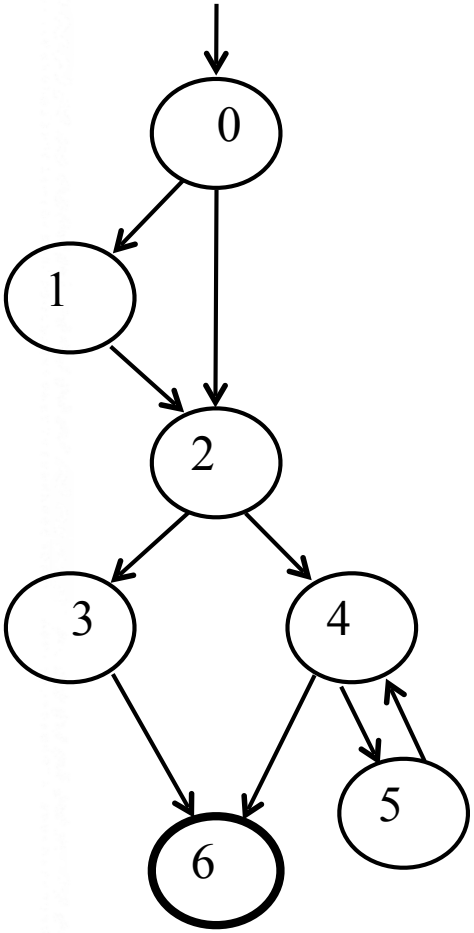
Test Paths: [0, 1, 2, 3, 6] [0, 2, 4, 5, 4, 6]

Edge-Pair Coverage

TR = { [0,1,2], [0,2,3], [0,2,4], [1,2,3], [1,2,4], [2,3,6],
[2,4,5], [2,4,6], [4,5,4], [5,4,5], [5,4,6] }

Test Paths: [0, 1, 2, 3, 6] [0, 1, 2, 4, 6] [0, 2, 3, 6]
[0, 2, 4, 5, 4, 5, 4, 6]

مثال از پوشش ساختار



Node Coverage

TR = { 0, 1, 2, 3, 4, 5, 6 }

Test Paths: [0, 1, 2, 3, 6] [0, 1, 2, 4, 5, 4, 6]

Edge Coverage

TR = { (0,1), (0,2), (1,2), (2,3), (2,4), (3,6), (4,5), (4,6), (5,4) }

Test Paths: [0, 1, 2, 3, 6] [0, 2, 4, 5, 4, 6]

Edge-Pair Coverage

TR = { [0,1,2], [0,2,3], [0,2,4], [1,2,3], [1,2,4], [2,3,6],
[2,4,5], [2,4,6], [4,5,4], [5,4,5], [5,4,6] }

Test Paths: [0, 1, 2, 3, 6] [0, 1, 2, 4, 6] [0, 2, 3, 6]
[0, 2, 4, 5, 4, 5, 4, 6]

Complete Path Coverage

Test Paths: [0, 1, 2, 3, 6] [0, 1, 2, 4, 6] [0, 1, 2, 4, 5, 4, 6] [0, 1, 2, 4, 5, 4, 5, 4, 6] [0, 1, 2, 4, 5, 4, 5, 4, 5, 4, 6] …

وجود حلقه در گراف

اگر یک گراف، دارای حلقه باشد، تعداد مسیرها بی‌نهایت می‌شود.

بنابراین، معیار CPC امکان‌پذیر نیست.

SPC نیز با توجه به سلیقه‌ای بودن و تفاوت بین سلایق آزمونگرها مناسب نیست.

تلاش به جهت مواجهه با حلقه‌ها:

1970s : Execute cycles once ([4, 5, 4] in previous example, informal)

1980s : Execute each loop, exactly once (formalized)

1990s : Execute loops 0 times, once, more than once (informal description)

2000s : Prime paths

مسیرهای ساده و مسیرهای اصلی

Simple Path : *A path from node ni to nj is simple if no node appears more than once, except possibly the first and last nodes are the same*

- No internal loops
- Includes all other subpaths
- A loop is a simple path

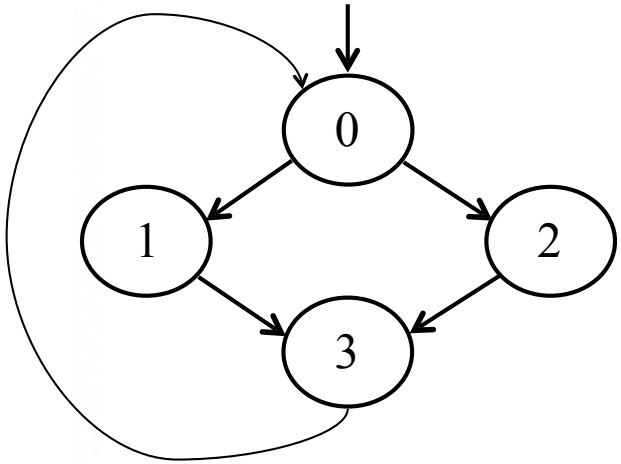
Prime Path : *A simple path that does not appear as a **proper subpath** of any other simple path*

مسیرهای ساده و مسیرهای اصلی

Simple Path : *A path from node ni to nj is simple if no node appears more than once, except possibly the first and last nodes are the same*

- No internal loops
- Includes all other subpaths
- A loop is a simple path

Prime Path : *A simple path that does not appear as a **proper subpath** of any other simple path*

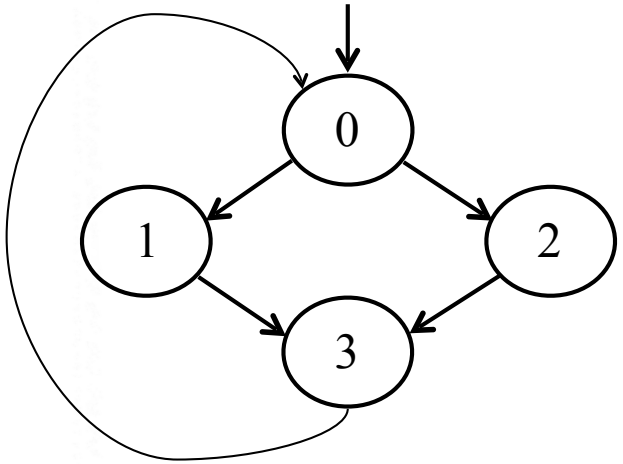


مسیرهای ساده و مسیرهای اصلی

Simple Path : *A path from node ni to nj is simple if no node appears more than once, except possibly the first and last nodes are the same*

- No internal loops
- Includes all other subpaths
- A loop is a simple path

Prime Path : *A simple path that does not appear as a **proper subpath** of any other simple path*



Simple Paths : [0, 1, 3, 0], [0, 2, 3, 0], [1, 3, 0, 1], [2, 3, 0, 2], [3, 0, 1, 3], [3, 0, 2, 3], [1, 3, 0, 2], [2, 3, 0, 1], [0, 1, 3], [0, 2, 3], [1, 3, 0], [2, 3, 0], [3, 0, 1], [3, 0, 2], [0, 1], [0, 2], [1, 3], [2, 3], [3, 0], [0], [1], [2], [3]

Prime Paths : [0, 1, 3, 0], [0, 2, 3, 0], [1, 3, 0, 1], [2, 3, 0, 2], [3, 0, 1, 3], [3, 0, 2, 3], [1, 3, 0, 2], [2, 3, 0, 1]

پوشش مسیرهای اصلی

یک روش هوشمندانه به جهت اجرای حلقه‌ها و عدم اجرای آن‌ها در تعداد محدود آزمون

تمامی مسیرهای به طول ۰، ۱ و ... را تور می‌کند.

بر این اساس، تمامی روش‌های همه گره‌ها، همه یال‌ها و جفت ـ یال را در بر دارد.

پوشش مسیرهای اصلی

Prime Path Coverage (PPC) : TR contains each prime path in G.

یک روش هوشمندانه به جهت اجرای حلقه‌ها و عدم اجرای آن‌ها در تعداد محدود آزمون

تمامی مسیرهای به طول ۰، ۱ و ... را تور می‌کند.

بر این اساس، تمامی روش‌های همه گره‌ها، همه یال‌ها و جفت‌ـیال را در بر دارد.

Round Trips

Round-Trip Path : یک مسیر اصلی که شروع و پایان آن مسیر روی یک گره قرار دارد.

این معیار گره‌ها و یال‌هایی که روی یک round trip نیستند را **از قلم می‌اندازد**.

بر این اساس، این معیار دربرگیرنده معیارهای جفت‌ـیال، هر یال و هر گره نیست.

Round Trips

Simple Round Trip Coverage (SRTC) : TR contains at least one round-trip path for each reachable node in G that begins and ends a round-trip path.

Round-Trip Path : یک مسیر اصلی که شروع و پایان آن مسیر روی یک گره قرار دارد.

این معیار گره‌ها و یال‌هایی که روی یک round trip نیستند را **از قلم می‌اندازد**.

بر این اساس، این معیار دربرگیرنده معیارهای جفت-یال، هر یال و هر گره نیست.

Round Trips

Simple Round Trip Coverage (SRTC) : TR contains at least one round-trip path for each reachable node in G that begins and ends a round-trip path.

Complete Round Trip Coverage (CRTC) : TR contains all round-trip paths for each reachable node in G.

Round-Trip Path : یک مسیر اصلی که شروع و پایان آن مسیر روی یک گره قرار دارد.

این معیار گره‌ها و یال‌هایی که روی یک round trip نیستند را **از قلم می‌اندازد**.

بر این اساس، این معیار دربرگیرنده معیارهای جفت-یال، هر یال و هر گره نیست.

مثال از مسیر اولیه

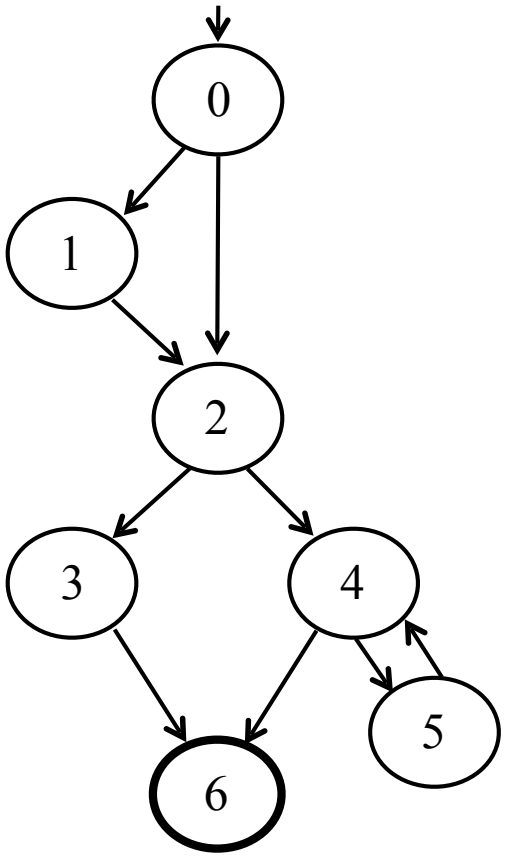
مثال قبل شامل ۳۸ مسیر ساده است.

از این تعداد تنها ۹ عدد مسیر اصلی هستند.

مثال از مسیر اولیه

مثال قبل شامل ۳۸ مسیر ساده است.

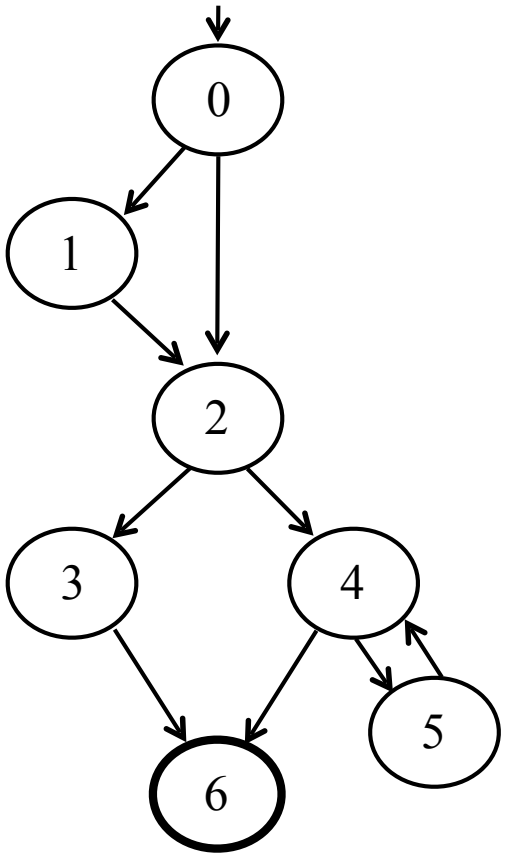
از این تعداد تنها ۹ عدد مسیر اصلی هستند.



مثال از مسیر اولیه

مثال قبل شامل ۳۸ مسیر ساده است.

از این تعداد تنها ۹ عدد مسیر اصلی هستند.

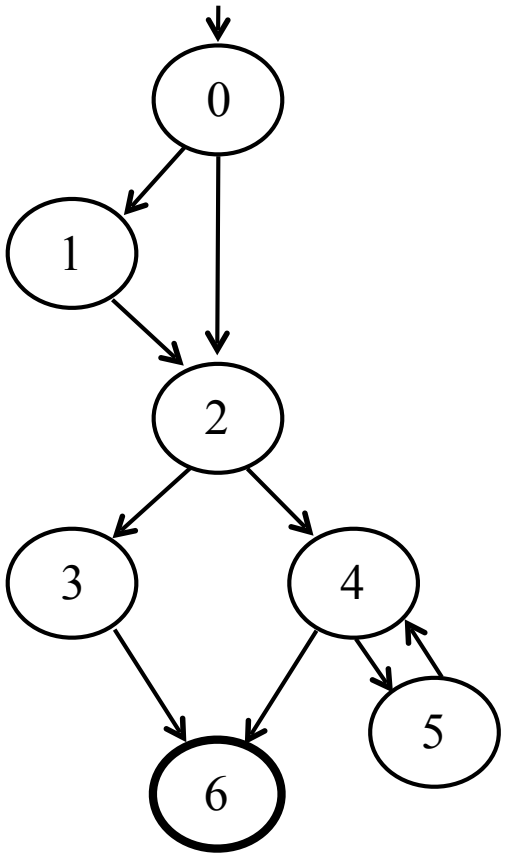


Prime Paths
[0, 1, 2, 3, 6]
[0, 1, 2, 4, 5]
[0, 1, 2, 4, 6]
[0, 2, 3, 6]
[0, 2, 4, 5]
[0, 2, 4, 6]
[5, 4, 6]
[4, 5, 4]
[5, 4, 5]

مثال از مسیر اولیه

مثال قبل شامل ۳۸ مسیر ساده است.

از این تعداد تنها ۹ عدد مسیر اصلی هستند.



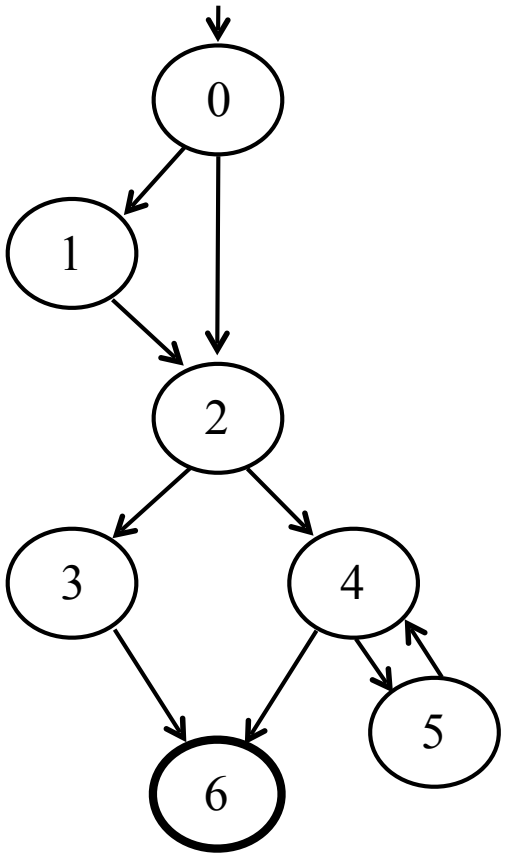
Prime Paths
[0, 1, 2, 3, 6]
[0, 1, 2, 4, 5]
[0, 1, 2, 4, 6]
[0, 2, 3, 6]
[0, 2, 4, 5]
[0, 2, 4, 6]
[5, 4, 6]
[4, 5, 4]
[5, 4, 5]

Execute loop 0 times

مثال از مسیر اولیه

مثال قبل شامل ۳۸ مسیر ساده است.

از این تعداد تنها ۹ عدد مسیر اصلی هستند.



Prime Paths
[0, 1, 2, 3, 6]
[0, 1, 2, 4, 5]
[0, 1, 2, 4, 6]
[0, 2, 3, 6]
[0, 2, 4, 5]
[0, 2, 4, 6]
[5, 4, 6]
[4, 5, 4]
[5, 4, 5]

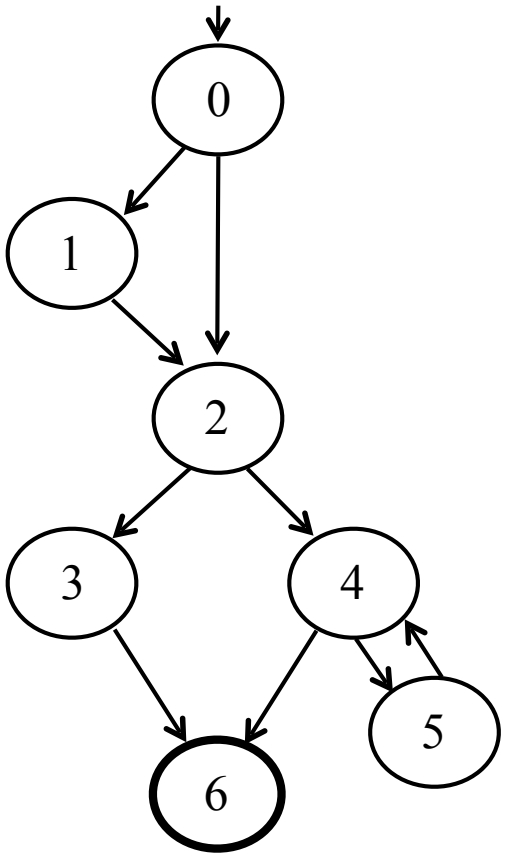
Execute loop 0 times

Execute loop once

مثال از مسیر اولیه

مثال قبل شامل ۳۸ مسیر ساده است.

از این تعداد تنها ۹ عدد مسیر اصلی هستند.



Prime Paths
[0, 1, 2, 3, 6]
[0, 1, 2, 4, 5]
[0, 1, 2, 4, 6]
[0, 2, 3, 6]
[0, 2, 4, 5]
[0, 2, 4, 6]
[5, 4, 6]
[4, 5, 4]
[5, 4, 5]

Execute loop 0 times

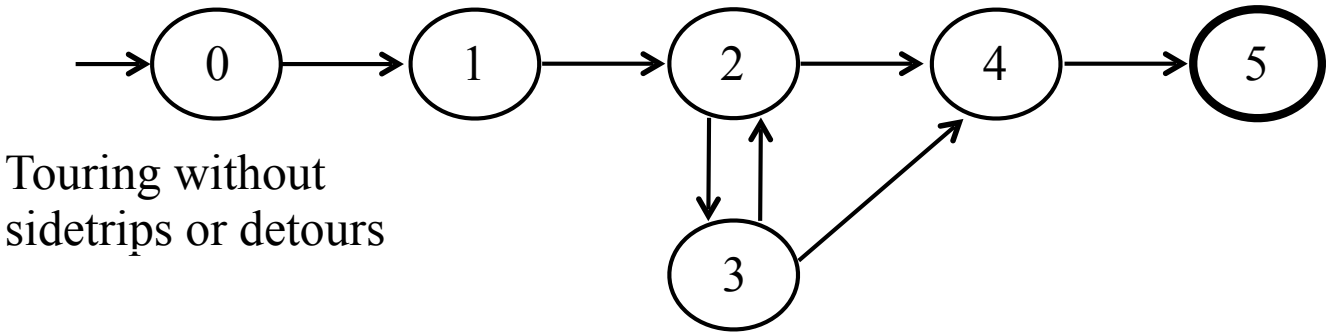
Execute loop once

Execute loop more than once

Touring, Sidetrips and Detours

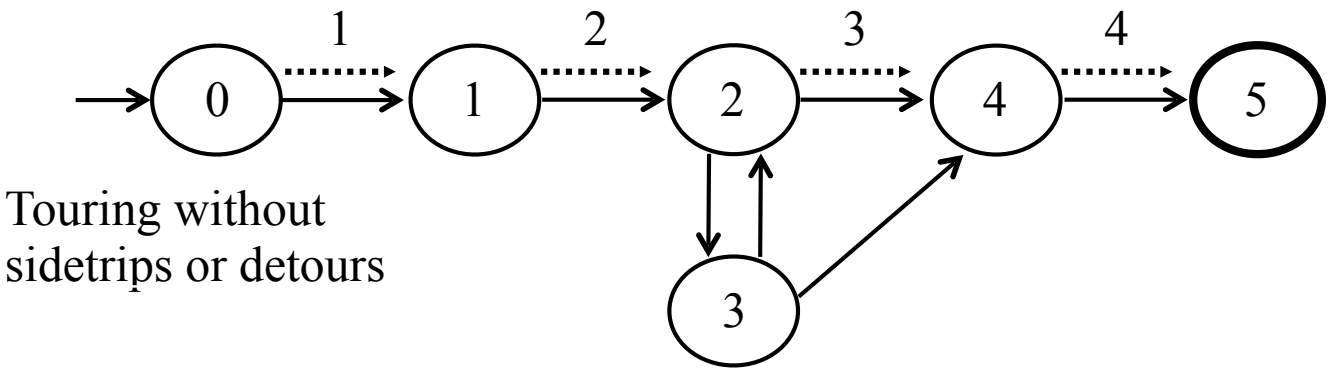
- Prime paths do not have internal loops … test paths might
- **Tour** : *A test path p tours subpath q if q is a subpath of p*
- **Tour With Sidetrips** : *A test path p tours subpath q with sidetrips iff every edge in q is also in p in the same order*
  - The tour can include a sidetrip, as long as it comes back to the same node
- **Tour With Detours** : *A test path p tours subpath q with detours iff every node in q is also in p in the same order*
  - The tour can include a detour from node *ni*, as long as it comes back to the prime path at a successor of *ni*

Sidetrips and Detours Example



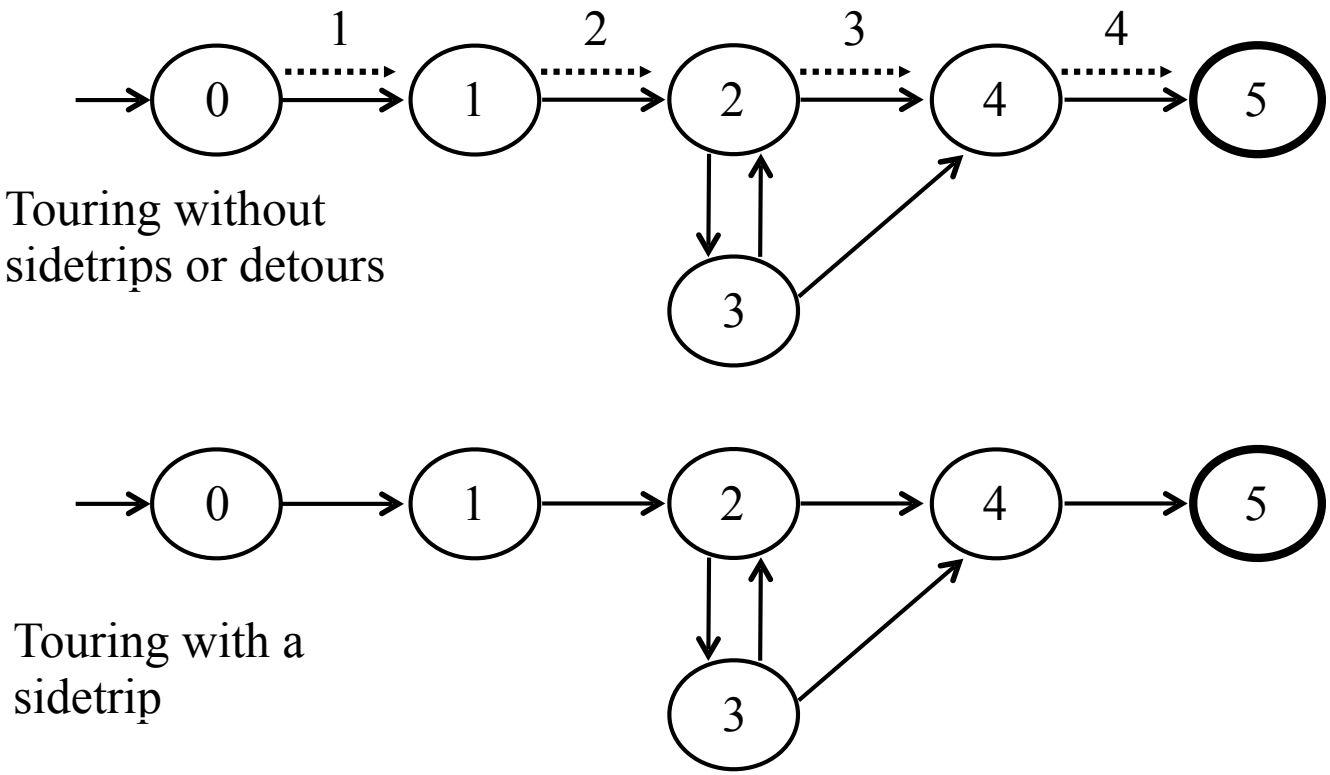
Touring without
sidetrips or detours

Sidetrips and Detours Example

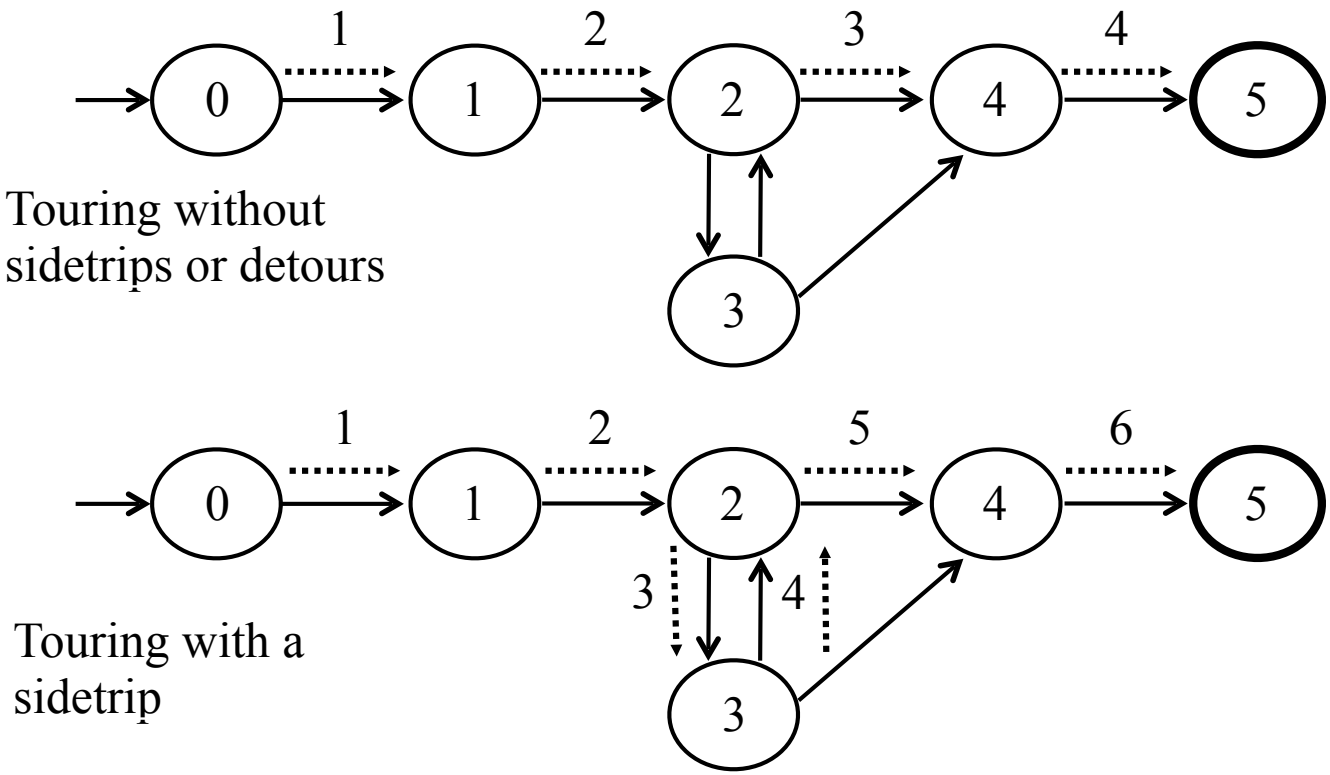


Touring without
sidetrips or detours

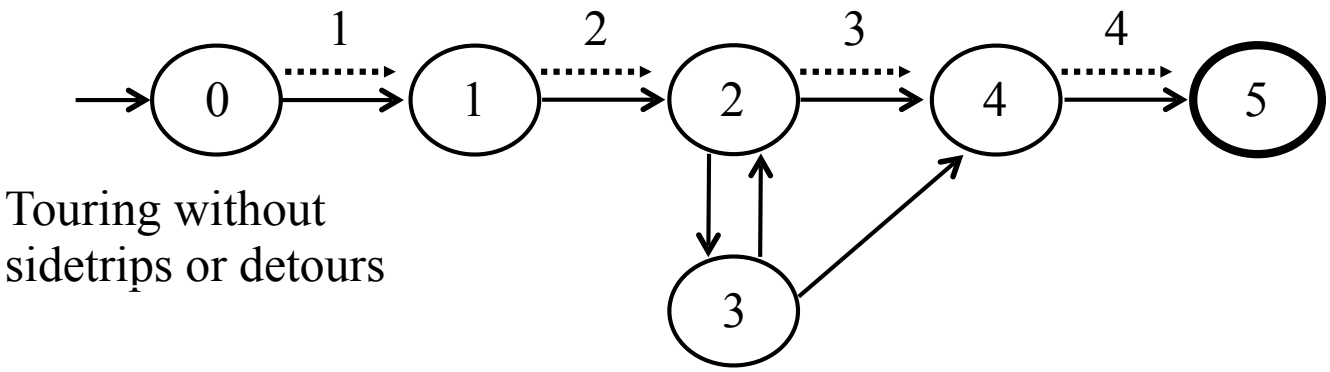
Sidetrips and Detours Example



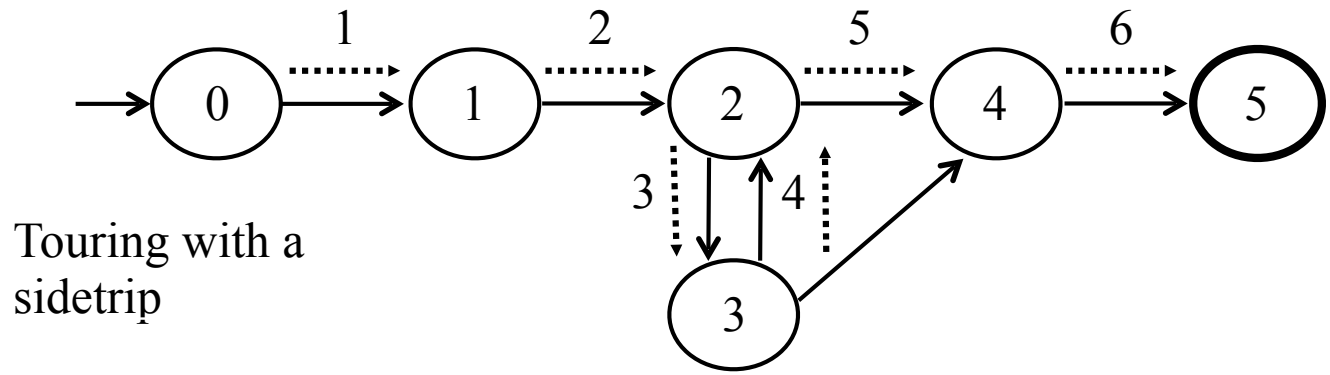
Sidetrips and Detours Example



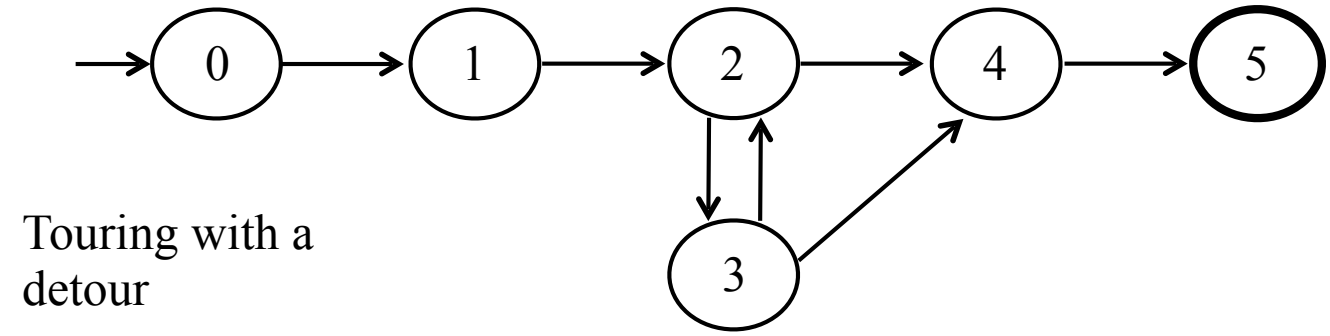
Sidetrips and Detours Example



Touring without sidetrips or detours

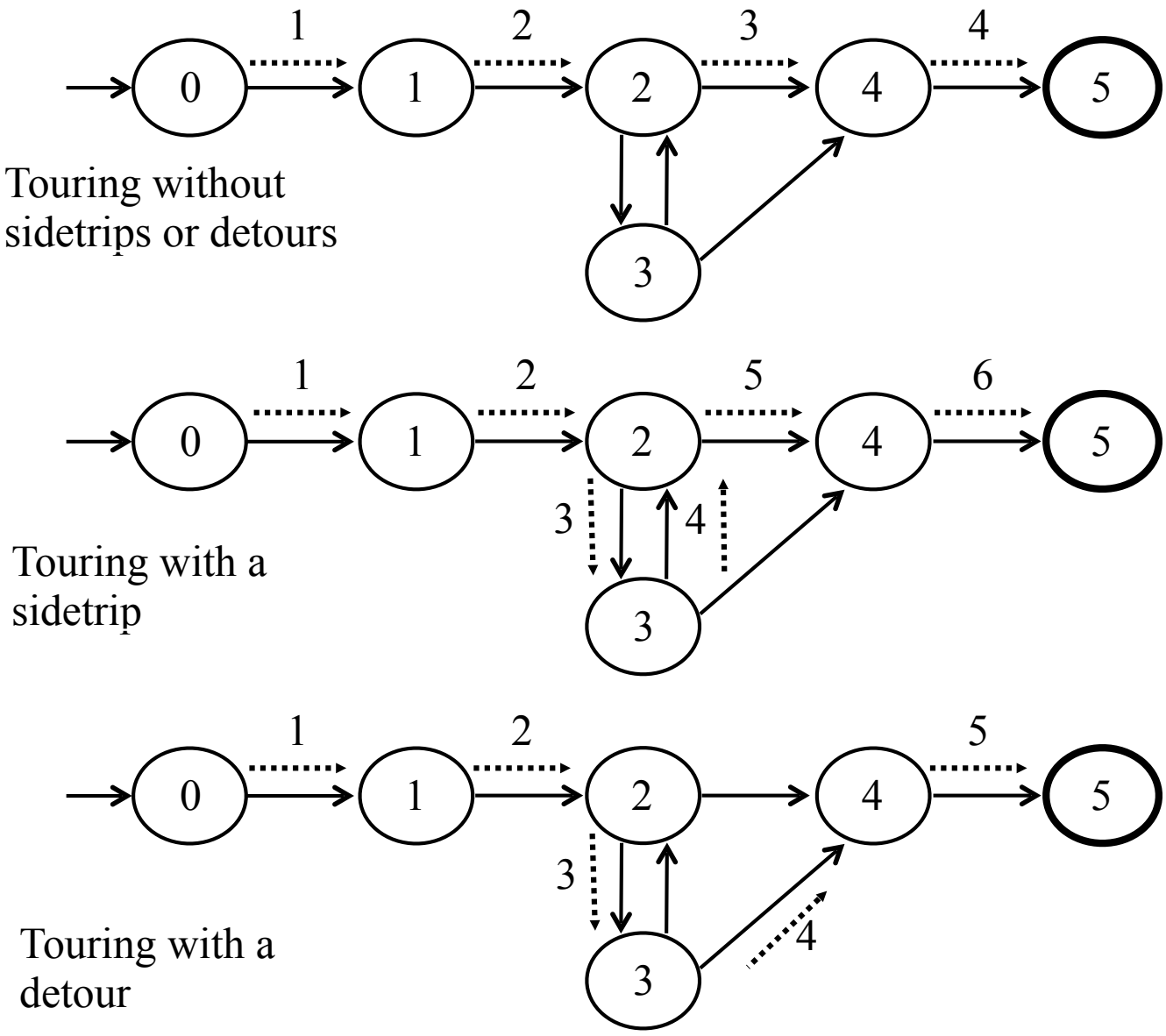


Touring with a sidetrip



Touring with a detour

Sidetrips and Detours Example



نیازمندی‌های آزمون غیرعملی

- **یک نیازمندی غیرعملی، نیازمندی‌ای است که قابل ارضا نباشد:**
  - کدهای مرده
  - مسیرهایی که منجر به تناقض می‌شوند. مثلا برای طی مسیر باید x>0 و X<0 باشد.
- بسیاری از معیارهای آزمون، منجر به ایجاد نیازمندی‌های غیرعملی می‌شوند.
- معمولا تعیین عملی بودن همه نیازمندی‌ها غیرقابل تصمیم‌گیری است.
- زمانی که sidetrips مجاز نباشد، نیازمندی‌های غیرعملی زیادی به وجود می‌آید.
- در هر حال، اجازه دادن به sidetrips سبب ضعیف شدن معیار آزمون می‌شود.

نیازمندی‌های آزمون غیرعملی

- **یک نیازمندی غیرعملی، نیازمندی‌ای است که قابل ارضا نباشد:**
  - کدهای مرده
  - مسیرهایی که منجر به تناقض می‌شوند. مثلا برای طی مسیر باید x>0 و X<0 باشد.
- بسیاری از معیارهای آزمون، منجر به ایجاد نیازمندی‌های غیرعملی می‌شوند.
- معمولا تعیین عملی بودن همه نیازمندی‌ها غیرقابل تصمیم‌گیری است.
- زمانی که sidetrips مجاز نباشد، نیازمندی‌های غیرعملی زیادی به وجود می‌آید.
- در هر حال، اجازه دادن به sidetrips سبب ضعیف شدن معیار آزمون می‌شود.

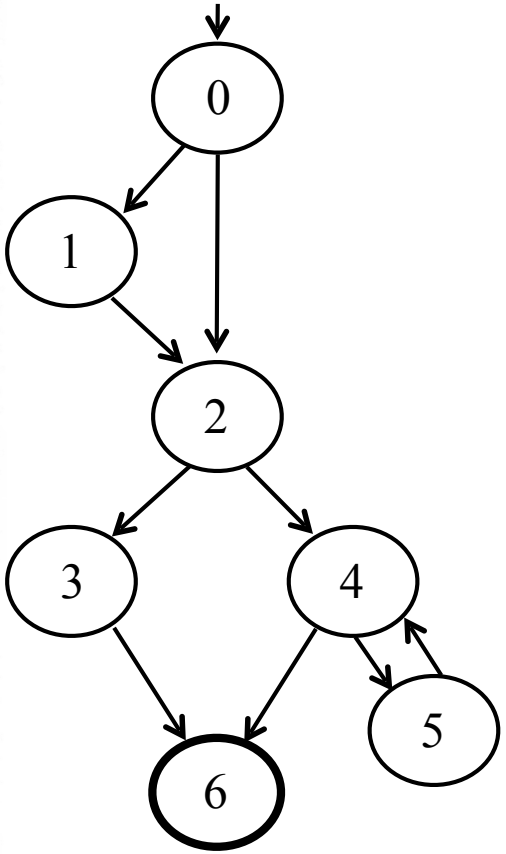
Practical recommendation – Best Effort Touring

Satisfy as many test requirements as possible without sidetrips

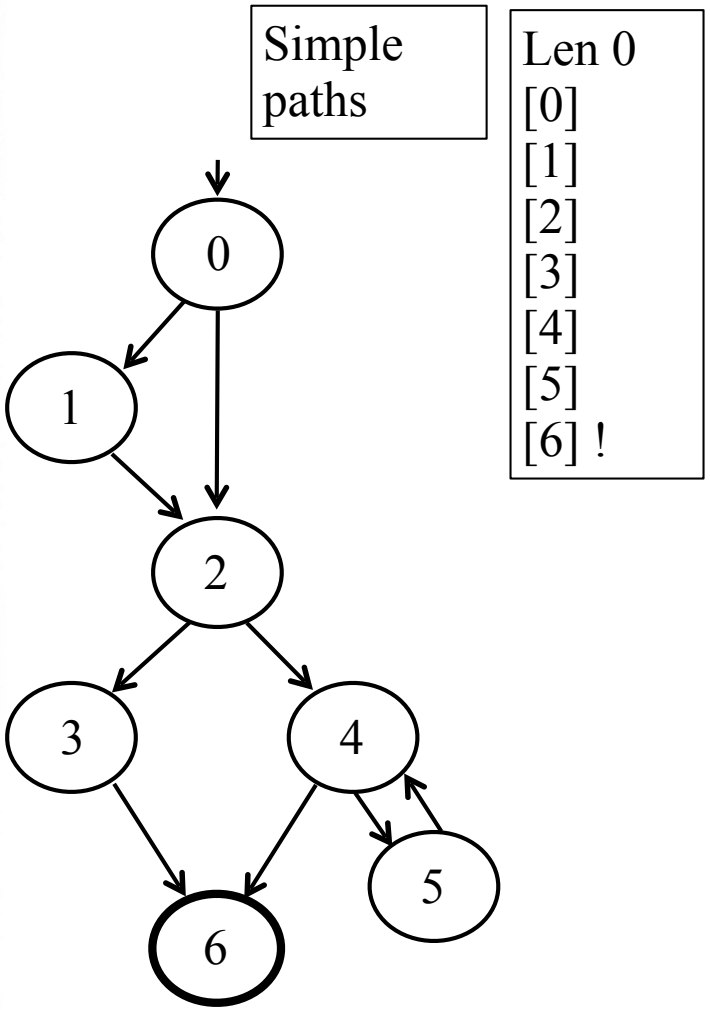
Allow sidetrips to try to satisfy unsatisfied test requirements

مثال از مسیرهای ساده و اصلی

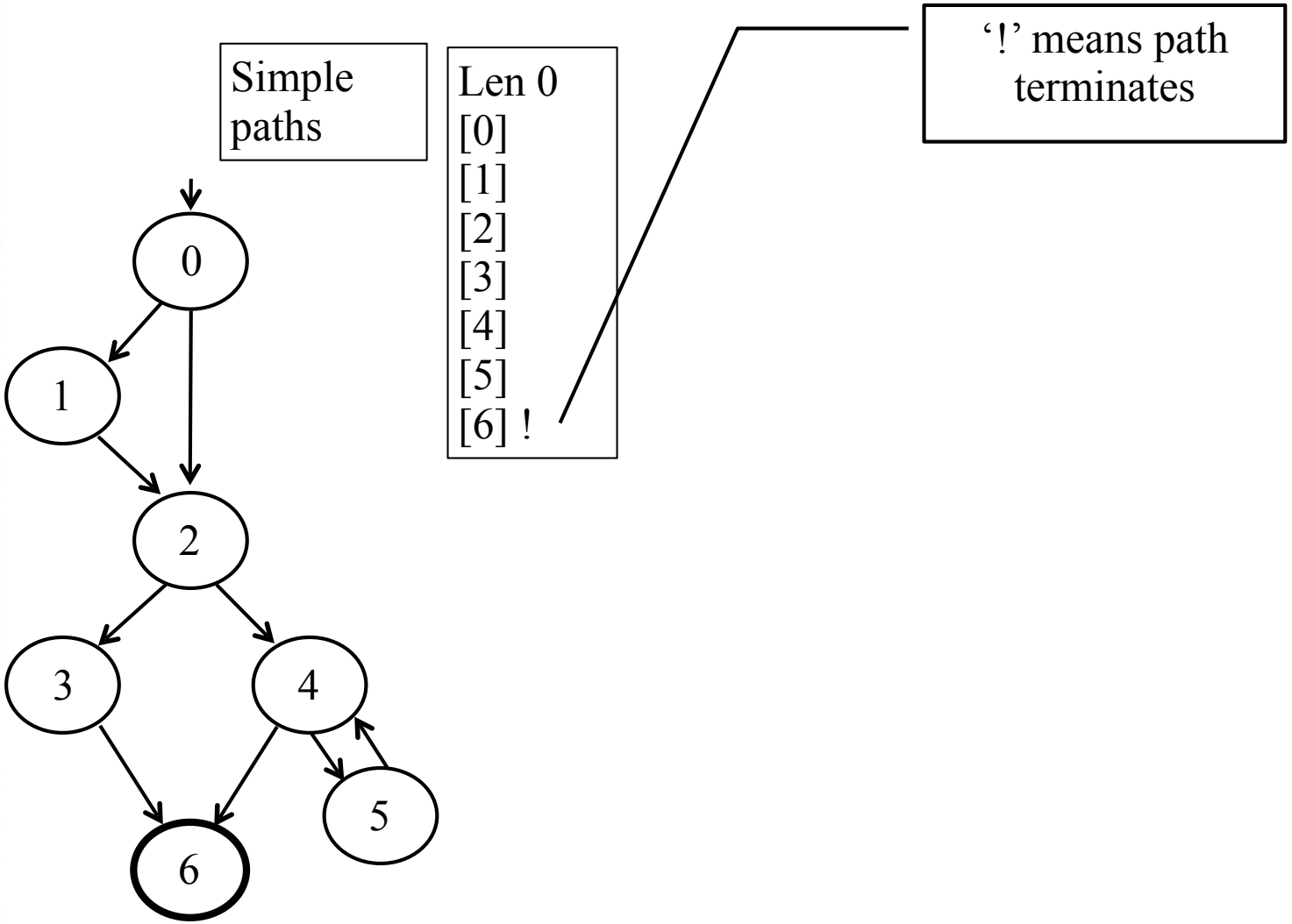
مثال از مسیرهای ساده و اصلی



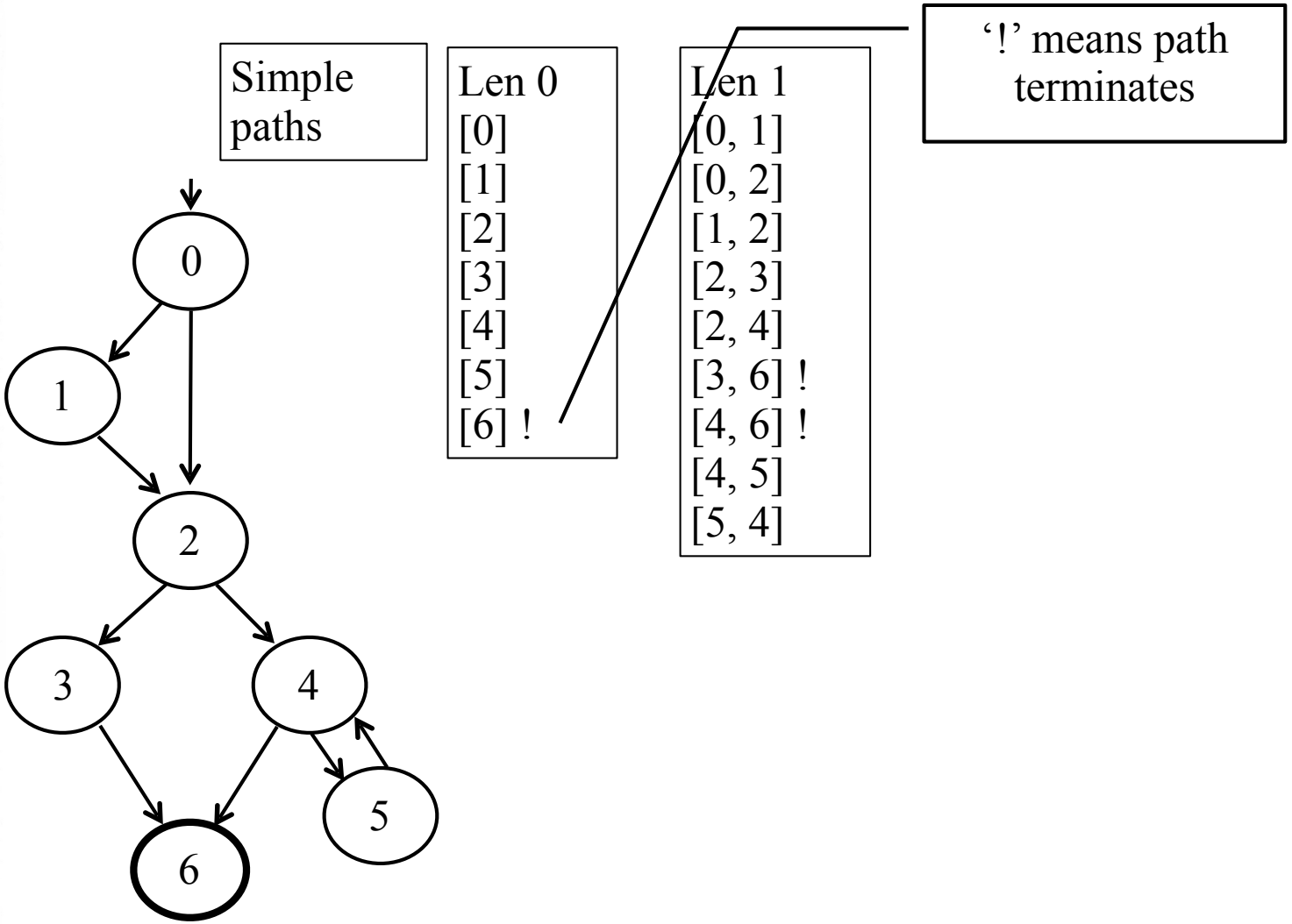
مثال از مسیرهای ساده و اصلی



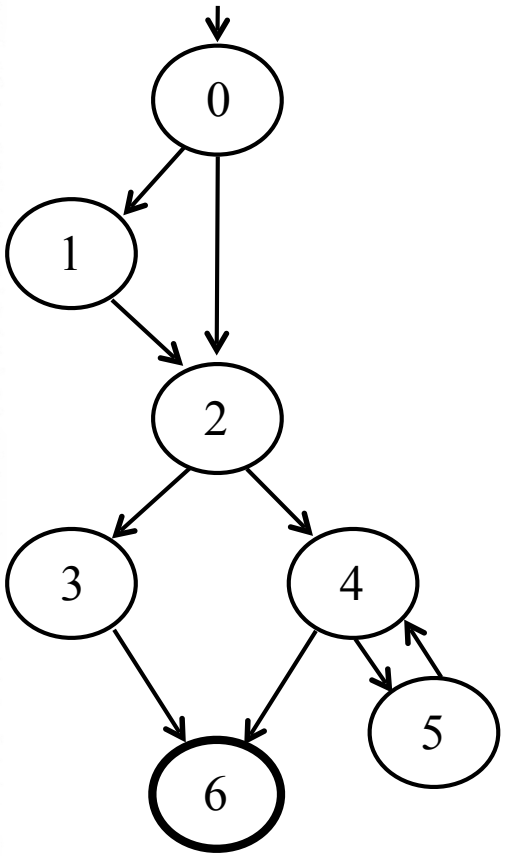
مثال از مسیرهای ساده و اصلی



مثال از مسیرهای ساده و اصلی



مثال از مسیرهای ساده و اصلی



Simple paths

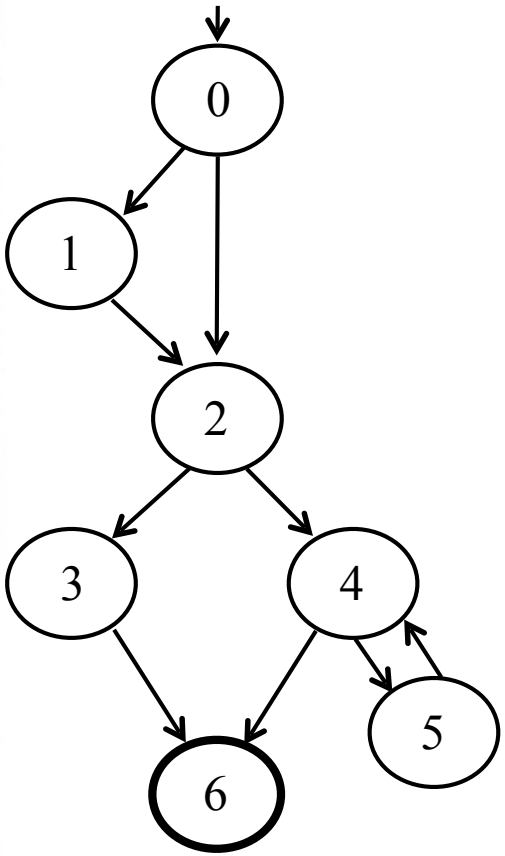
Len 0
[0]
[1]
[2]
[3]
[4]
[5]
[6] !

Len 1
[0, 1]
[0, 2]
[1, 2]
[2, 3]
[2, 4]
[3, 6] !
[4, 6] !
[4, 5]
[5, 4]

‘!’ means path terminates

Len 2
[0, 1, 2]
[0, 2, 3]
[0, 2, 4]
[1, 2, 3]
[1, 2, 4]
[2, 3, 6] !
[2, 4, 6] !
[2, 4, 5] !
[4, 5, 4] *
[5, 4, 6] !
[5, 4, 5] *

مثال از مسیرهای ساده و اصلی



Simple paths

Len 0
[0]
[1]
[2]
[3]
[4]
[5]
[6] !

Len 1
[0, 1]
[0, 2]
[1, 2]
[2, 3]
[2, 4]
[3, 6] !
[4, 6] !
[4, 5]
[5, 4]

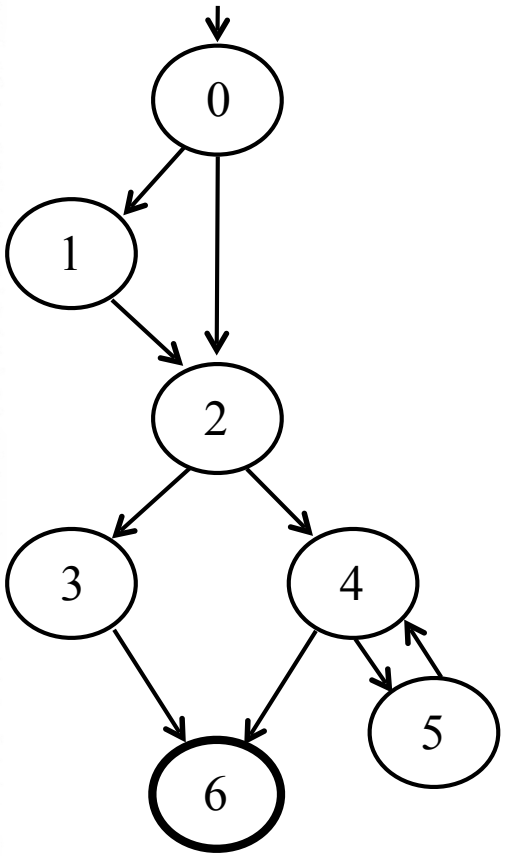
‘!’ means path terminates

Len 2
[0, 1, 2]
[0, 2, 3]
[0, 2, 4]
[1, 2, 3]
[1, 2, 4]
[2, 3, 6] !
[2, 4, 6] !
[2, 4, 5] !
[4, 5, 4] *
[5, 4, 6] !
[5, 4, 5] *

‘*’ means path cycles

مثال از مسیرهای ساده و اصلی

Simple paths



Len 0
[0]
[1]
[2]
[3]
[4]
[5]
[6] !

Len 1
[0, 1]
[0, 2]
[1, 2]
[2, 3]
[2, 4]
[3, 6] !
[4, 6] !
[4, 5]
[5, 4]

‘!’ means path terminates

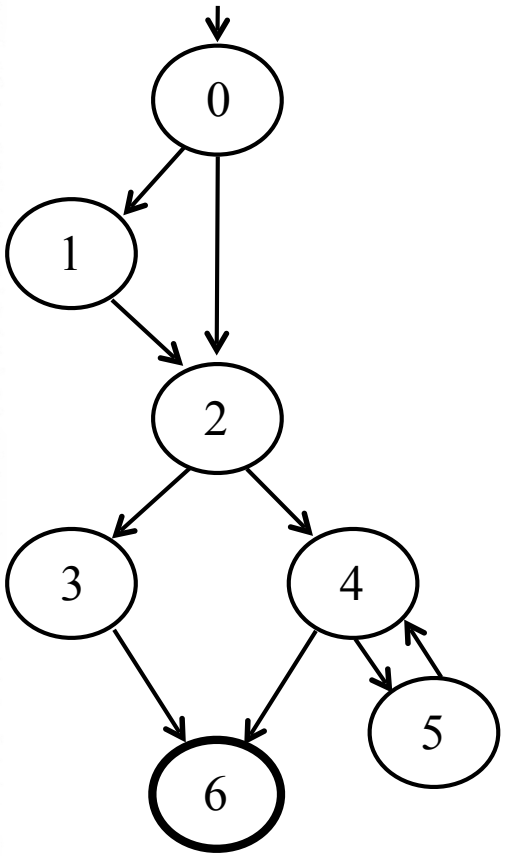
Len 2
[0, 1, 2]
[0, 2, 3]
[0, 2, 4]
[1, 2, 3]
[1, 2, 4]
[2, 3, 6] !
[2, 4, 6] !
[2, 4, 5] !
[4, 5, 4] *
[5, 4, 6] !
[5, 4, 5] *

Len 3
[0, 1, 2, 3]
[0, 1, 2, 4]
[0, 2, 3, 6] !
[0, 2, 4, 6] !
[0, 2, 4, 5] !
[1, 2, 3, 6] !
[1, 2, 4, 5] !
[1, 2, 4, 6] !

‘*’ means path cycles

مثال از مسیرهای ساده و اصلی

Simple paths



Len 0
[0]
[1]
[2]
[3]
[4]
[5]
[6] !

Len 1
[0, 1]
[0, 2]
[1, 2]
[2, 3]
[2, 4]
[3, 6] !
[4, 6] !
[4, 5]
[5, 4]

‘!’ means path terminates

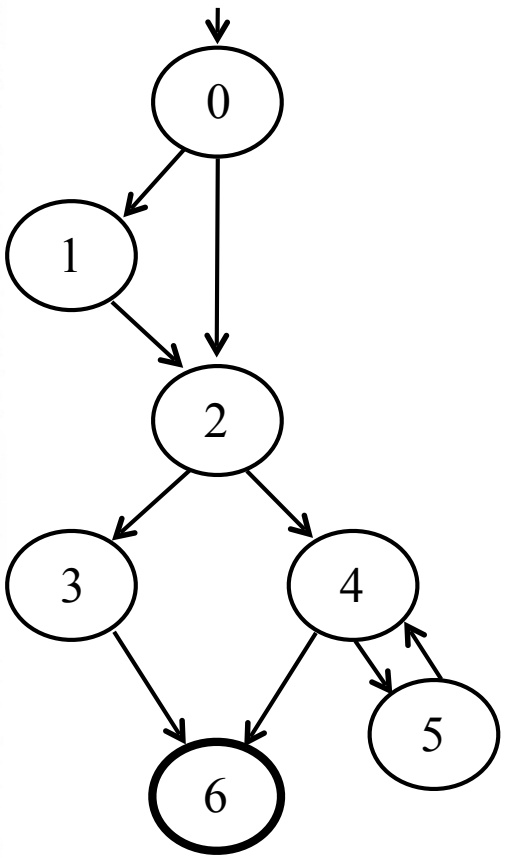
Len 2
[0, 1, 2]
[0, 2, 3]
[0, 2, 4]
[1, 2, 3]
[1, 2, 4]
[2, 3, 6] !
[2, 4, 6] !
[2, 4, 5] !
[4, 5, 4] *
[5, 4, 6] !
[5, 4, 5] *

Len 3
[0, 1, 2, 3]
[0, 1, 2, 4]
[0, 2, 3, 6] !
[0, 2, 4, 6] !
[0, 2, 4, 5] !
[1, 2, 3, 6] !
[1, 2, 4, 5] !
[1, 2, 4, 6] !

‘*’ means path cycles

Len 4
[0, 1, 2, 3, 6] !
[0, 1, 2, 4, 6] !
[0, 1, 2, 4, 5] !

مثال از مسیرهای ساده و اصلی



Simple paths

Len 0
[0]
[1]
[2]
[3]
[4]
[5]
[6] !

Len 1
[0, 1]
[0, 2]
[1, 2]
[2, 3]
[2, 4]
[3, 6] !
[4, 6] !
[4, 5]
[5, 4]

‘!’ means path terminates

Len 2
[0, 1, 2]
[0, 2, 3]
[0, 2, 4]
[1, 2, 3]
[1, 2, 4]
[2, 3, 6] !
[2, 4, 6] !
[2, 4, 5] !
[4, 5, 4] *
[5, 4, 6] !
[5, 4, 5] *

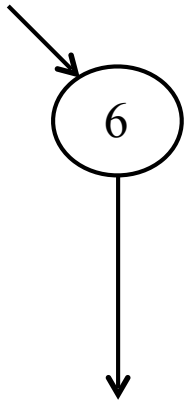
Len 3
[0, 1, 2, 3]
[0, 1, 2, 4]
[0, 2, 3, 6] !
[0, 2, 4, 6] !
[0, 2, 4, 5] !
[1, 2, 3, 6] !
[1, 2, 4, 5] !
[1, 2, 4, 6] !

‘*’ means path cycles

Len 4
[0, 1, 2, 3, 6] !
[0, 1, 2, 4, 6] !
[0, 1, 2, 4, 5] !

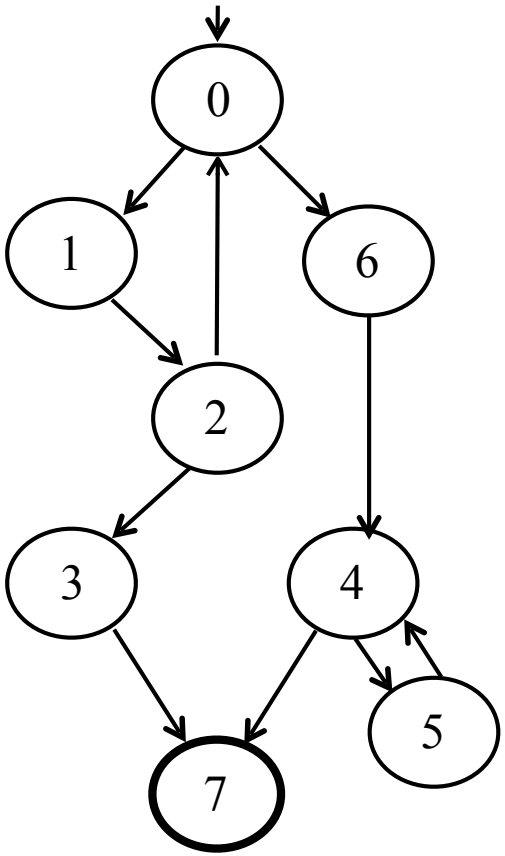
Prime Paths

کوییز دوم



مسیرهای اصلی را در گراف بالا تعیین نمایید

کوییز دوم



مسیرهای اصلی را در گراف بالا تعیین نمایید

مسیرهای اصلی - مثال دوم

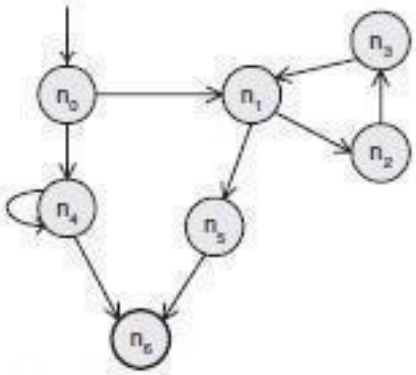


Figure 2.10. An example for prime test paths.

Simple paths of length 0 (7):

1) [0]
2) [1]
3) [2]
4) [3]
5) [4]
6) [5]
7) [6] !

مسیرهای اصلی - مثال دوم

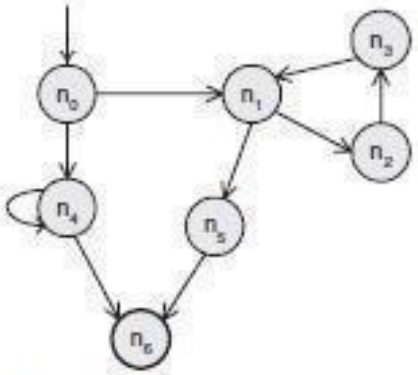


Figure 2.10. An example for prime test paths.

Simple paths of length 0 (7):

1) [0]
2) [1]
3) [2]
4) [3]
5) [4]
6) [5]
7) [6] !

Simple paths of length 1 (9):

8) [0, 1]
9) [0, 4]
10) [1, 2]
11) [1, 5]
12) [2, 3]
13) [3, 1]
14) [4, 4] *
15) [4, 6] !
16) [5, 6] !

مسیرهای اصلی - مثال دوم

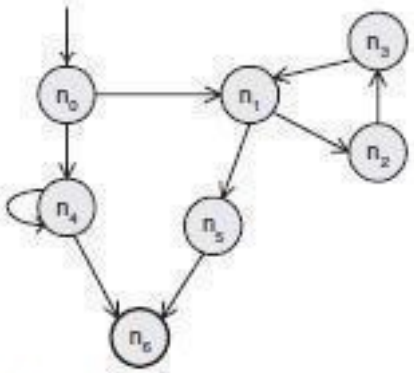


Figure 2.10. An example for prime test paths.

Simple paths of length 0 (7):

1) [0]
2) [1]
3) [2]
4) [3]
5) [4]
6) [5]
7) [6] !

Simple paths of length 1 (9):

8) [0, 1]
9) [0, 4]
10) [1, 2]
11) [1, 5]
12) [2, 3]
13) [3, 1]
14) [4, 4] *
15) [4, 6] !
16) [5, 6] !

Simple paths of length 2 (8):

17) [0, 1, 2]
18) [0, 1, 5]
19) [0, 4, 6] !
20) [1, 2, 3]
21) [1, 5, 6] !
22) [2, 3, 1]
23) [3, 1, 2]
24) [3, 1, 5]

مسیرهای اصلی - مثال دوم

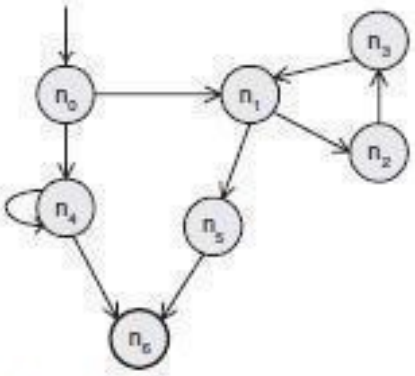


Figure 2.10. An example for prime test paths.

Simple paths of length 0 (7):

1) [0]
2) [1]
3) [2]
4) [3]
5) [4]
6) [5]
7) [6] !

Simple paths of length 1 (9):

8) [0, 1]
9) [0, 4]
10) [1, 2]
11) [1, 5]
12) [2, 3]
13) [3, 1]
14) [4, 4] *
15) [4, 6] !
16) [5, 6] !

Simple paths of length 2 (8):

17) [0, 1, 2]
18) [0, 1, 5]
19) [0, 4, 6] !
20) [1, 2, 3]
21) [1, 5, 6] !
22) [2, 3, 1]
23) [3, 1, 2]
24) [3, 1, 5]

Simple paths of length 3 (7):

25) [0, 1, 2, 3] !
26) [0, 1, 5, 6] !
27) [1, 2, 3, 1] *
28) [2, 3, 1, 2] *
29) [2, 3, 1, 5]
30) [3, 1, 2, 3] *
31) [3, 1, 5, 6] !

مسیرهای اصلی - مثال دوم

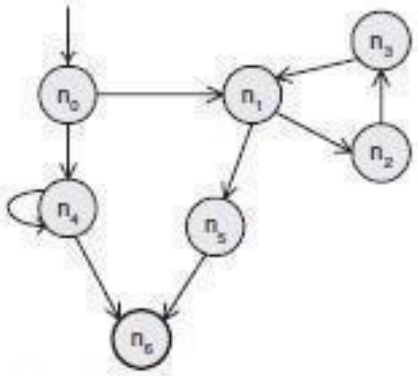


Figure 2.10. An example for prime test paths.

Simple paths of length 0 (7):

1) [0]
2) [1]
3) [2]
4) [3]
5) [4]
6) [5]
7) [6] !

Simple paths of length 1 (9):

8) [0, 1]
9) [0, 4]
10) [1, 2]
11) [1, 5]
12) [2, 3]
13) [3, 1]
14) [4, 4] *
15) [4, 6] !
16) [5, 6] !

Simple paths of length 2 (8):

17) [0, 1, 2]
18) [0, 1, 5]
19) [0, 4, 6] !
20) [1, 2, 3]
21) [1, 5, 6] !
22) [2, 3, 1]
23) [3, 1, 2]
24) [3, 1, 5]

Simple paths of length 3 (7):

25) [0, 1, 2, 3] !
26) [0, 1, 5, 6] !
27) [1, 2, 3, 1] *
28) [2, 3, 1, 2] *
29) [2, 3, 1, 5]
30) [3, 1, 2, 3] *
31) [3, 1, 5, 6] !

Prime paths of length 4 (1):

32) [2, 3, 1, 5, 6]!

14) [4, 4] *
19) [0, 4, 6] !
25) [0, 1, 2, 3] !
26) [0, 1, 5, 6] !
27) [1, 2, 3, 1] *
28) [2, 3, 1, 2] *
30) [3, 1, 2, 3] *
32) [2, 3, 1, 5, 6]!

مسیرهای اصلی - مثال دوم

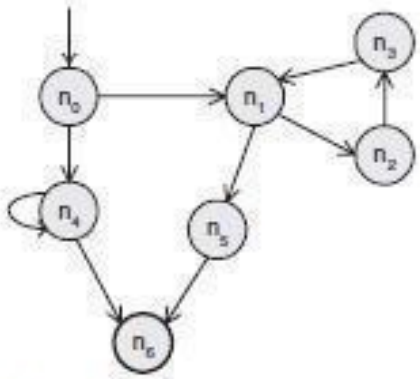


Figure 2.10. An example for prime test paths.

Simple paths of length 0 (7):

1) [0]
2) [1]
3) [2]
4) [3]
5) [4]
6) [5]
7) [6] !

Simple paths of length 1 (9):

8) [0, 1]
9) [0, 4]
10) [1, 2]
11) [1, 5]
12) [2, 3]
13) [3, 1]
14) [4, 4] *
15) [4, 6] !
16) [5, 6] !

Simple paths of length 2 (8):

17) [0, 1, 2]
18) [0, 1, 5]
19) [0, 4, 6] !
20) [1, 2, 3]
21) [1, 5, 6] !
22) [2, 3, 1]
23) [3, 1, 2]
24) [3, 1, 5]

Simple paths of length 3 (7):

25) [0, 1, 2, 3] !
26) [0, 1, 5, 6] !
27) [1, 2, 3, 1] *
28) [2, 3, 1, 2] *
29) [2, 3, 1, 5]
30) [3, 1, 2, 3] *
31) [3, 1, 5, 6] !

Prime paths of length 4 (1):

32) [2, 3, 1, 5, 6]!

14) [4, 4] *
19) [0, 4, 6] !
25) [0, 1, 2, 3] !
26) [0, 1, 5, 6] !
27) [1, 2, 3, 1] *
28) [2, 3, 1, 2] *
30) [3, 1, 2, 3] *
32) [2, 3, 1, 5, 6]!

The complete set of test paths is then:

1) [0, 1, 2, 3, 1, 5, 6]
2) [0, 1, 2, 3, 1, 2, 3, 1, 5, 6]
3) [0, 1, 5, 6]
4) [0, 4, 6]
5) [0, 4, 4, 6]

پایان

پوشش گراف مبتنی بر جریان داده

محمد تنهایی

معیار مبتنی بر جریان داده

هدف: اطمینان از محاسبه و استفاده درست از داده‌ها

Definition (def) : جایی که یک متغییر مقداردهی می‌شود. یا به عبارتی مقدار آن در حافظه قرار می‌گیرد.

Use : جایی که مقدار یک متغییر مورد استفاده و دسترسی قرار می‌گیرد.

def (n) or def (e) : مجموعه‌ای از متغیرهای که در گره n و یا یال e تعریف شده‌اند.

use (n) or use (e) : مجموعه‌ای از متغیرها که در گره n و یا یال e مورد استفاده قرار گرفته‌اند.

معیار مبتنی بر جریان داده

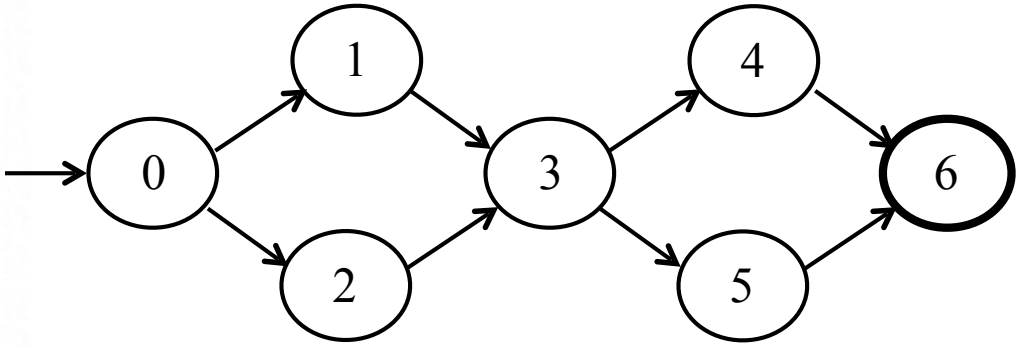
هدف: اطمینان از محاسبه و استفاده درست از داده‌ها

Definition (def) : جایی که یک متغییر مقداردهی می‌شود. یا به عبارتی مقدار آن در حافظه قرار می‌گیرد.

Use : جایی که مقدار یک متغییر مورد استفاده و دسترسی قرار می‌گیرد.

def (n) or def (e) : مجموعه‌ای از متغیرهای که در گره n و یا یال e تعریف شده‌اند.

use (n) or use (e) : مجموعه‌ای از متغیرها که در گره n و یا یال e مورد استفاده قرار گرفته‌اند.



معیار مبتنی بر جریان داده

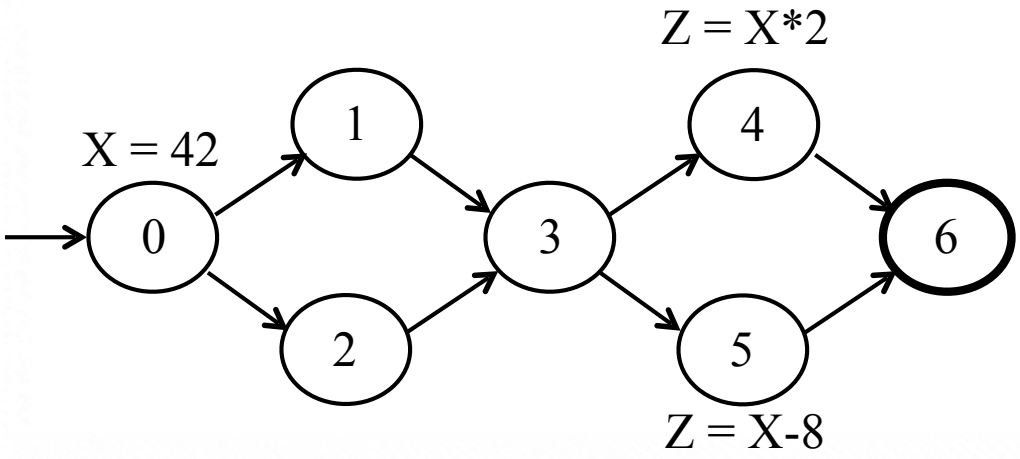
هدف: اطمینان از محاسبه و استفاده درست از داده‌ها

Definition (def) : جایی که یک متغییر مقداردهی می‌شود. یا به عبارتی مقدار آن در حافظه قرار می‌گیرد.

Use : جایی که مقدار یک متغییر مورد استفاده و دسترسی قرار می‌گیرد.

def (n) or def (e) : مجموعه‌ای از متغیرهای که در گره n و یا یال e تعریف شده‌اند.

use (n) or use (e) : مجموعه‌ای از متغیرها که در گره n و یا یال e مورد استفاده قرار گرفته‌اند.



معیار مبتنی بر جریان داده

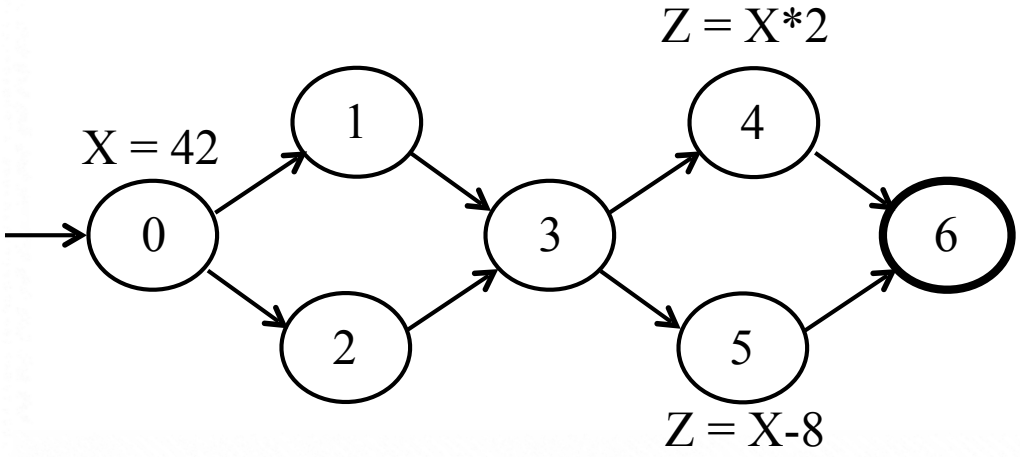
هدف: اطمینان از محاسبه و استفاده درست از داده‌ها

Definition (def) : جایی که یک متغییر مقداردهی می‌شود. یا به عبارتی مقدار آن در حافظه قرار می‌گیرد.

Use : جایی که مقدار یک متغییر مورد استفاده و دسترسی قرار می‌گیرد.

def (n) or def (e) : مجموعه‌ای از متغیرهای که در گره n و یا یال e تعریف شده‌اند.

use (n) or use (e) : مجموعه‌ای از متغیرها که در گره n و یا یال e مورد استفاده قرار گرفته‌اند.



Defs: def (0) = {X}
def (4) = {Z}
def (5) = {Z}
Uses: use (4) = {X}
use (5) = {X}

DU Pairs and DU Paths

DU Pairs and DU Paths

DU pair : A pair of locations (l_i, l_j) such that a variable v is defined at l_i and used at l_j

DU Pairs and DU Paths

DU pair : A pair of locations (l_i, l_j) such that a variable v is defined at l_i and used at l_j

Def-clear : A path from l_i to l_j is *def-clear* with respect to variable v if v is not given another value on any of the nodes or edges in the path

DU Pairs and DU Paths

DU pair : A pair of locations (l_i, l_j) such that a variable v is defined at l_i and used at l_j

Def-clear : A path from l_i to l_j is *def-clear* with respect to variable v if v is not given another value on any of the nodes or edges in the path

Reach : If there is a def-clear path from l_i to l_j with respect to v, the def of v at l_i reaches the use at l_j

DU Pairs and DU Paths

DU pair : A pair of locations (l_i, l_j) such that a variable v is defined at l_i and used at l_j

Def-clear : A path from l_i to l_j is *def-clear* with respect to variable v if v is not given another value on any of the nodes or edges in the path

Reach : If there is a def-clear path from l_i to l_j with respect to v, the def of v at l_i reaches the use at l_j

du-path : A simple subpath that is def-clear with respect to v from a def of v to a use of v

DU Pairs and DU Paths

DU pair : A pair of locations (l_i, l_j) such that a variable v is defined at l_i and used at l_j

Def-clear : A path from l_i to l_j is *def-clear* with respect to variable v if v is not given another value on any of the nodes or edges in the path

Reach : If there is a def-clear path from l_i to l_j with respect to v, the def of v at l_i reaches the use at l_j

du-path : A simple subpath that is def-clear with respect to v from a def of v to a use of v

du (n_i, n_j, v) – the set of du-paths from n_i to n_j

DU Pairs and DU Paths

DU pair : A pair of locations (l_i, l_j) such that a variable v is defined at l_i and used at l_j

Def-clear : A path from l_i to l_j is *def-clear* with respect to variable v if v is not given another value on any of the nodes or edges in the path

Reach : If there is a def-clear path from l_i to l_j with respect to v, the def of v at l_i reaches the use at l_j

du-path : A simple subpath that is def-clear with respect to v from a def of v to a use of v

du (n_i, n_j, v) – the set of du-paths from n_i to n_j

du (n_i, v) – the set of du-paths that start at n_i

Touring DU-Paths

A test path p *du-tours* subpath d with respect to v if p tours d and the subpath taken is def-clear with respect to v

Sidetrips can be used, just as with previous touring

Three criteria

- Use every def
- Get to every use
- Follow all du-paths

معیارهای مبتنی بر جریان داده

- مطمین می‌شویم که هر تعریف به یک استفاده می‌رسد.

معیارهای مبتنی بر جریان داده

- مطمین می‌شویم که هر تعریف به یک استفاده می‌رسد.

All-defs coverage (ADC) : For each set of du-paths $S = du(n, v)$, TR contains at least one path d in S.

معیارهای مبتنی بر جریان داده

- مطمین می‌شویم که هر تعریف به یک استفاده می‌رسد.

All-defs coverage (ADC) : For each set of du-paths $S = du(n, v)$, TR contains at least one path d in S.

- هر تعریف به تمامی استفاده‌ها ختم می‌شود.

معیارهای مبتنی بر جریان داده

- مطمین می‌شویم که هر تعریف به یک استفاده می‌رسد.

All-defs coverage (ADC) : For each set of du-paths $S = du\ (n, v)$, TR contains at least one path d in S.

- هر تعریف به تمامی استفاده‌ها ختم می‌شود.

All-uses coverage (AUC) : For each set of du-paths to uses $S = reach\ (n_i, v)$, TR contains at least one path d in S.

معیارهای مبتنی بر جریان داده

- مطمین می‌شویم که هر تعریف به یک استفاده می‌رسد.

All-defs coverage (ADC) : For each set of du-paths $S = du\ (n, v)$, TR contains at least one path d in S.

- هر تعریف به تمامی استفاده‌ها ختم می‌شود.

All-uses coverage (AUC) : For each set of du-paths to uses $S = reach\ (n_i, v)$, TR contains at least one path d in S.

- در نهایت، تمامی تعریف‌ها به تمامی استفاده‌ها ختم شود. (همه مسیرهای بین این دو)

معیارهای مبتنی بر جریان داده

- مطمین می‌شویم که هر تعریف به یک استفاده می‌رسد.

All-defs coverage (ADC) : For each set of du-paths $S = du\ (n, v)$, TR contains at least one path d in S.

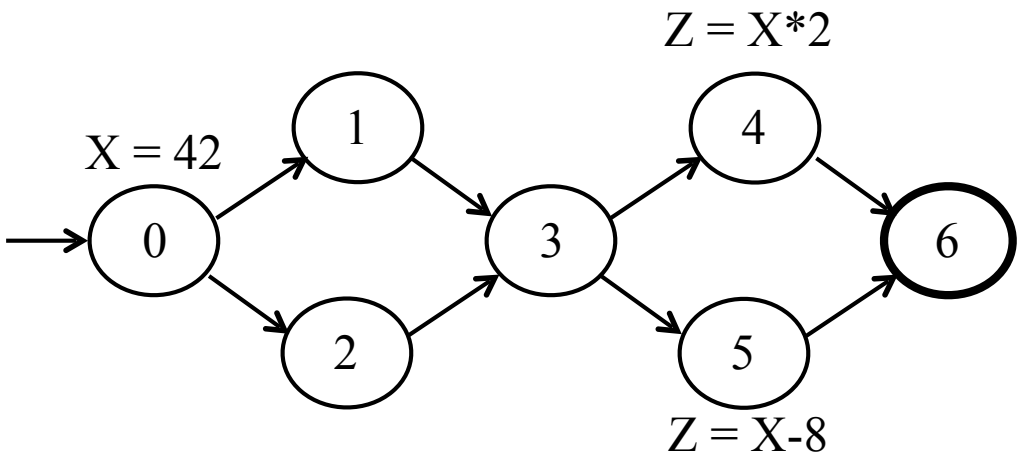
- هر تعریف به تمامی استفاده‌ها ختم می‌شود.

All-uses coverage (AUC) : For each set of du-paths to uses $S = reach\ (n_i, v)$, TR contains at least one path d in S.

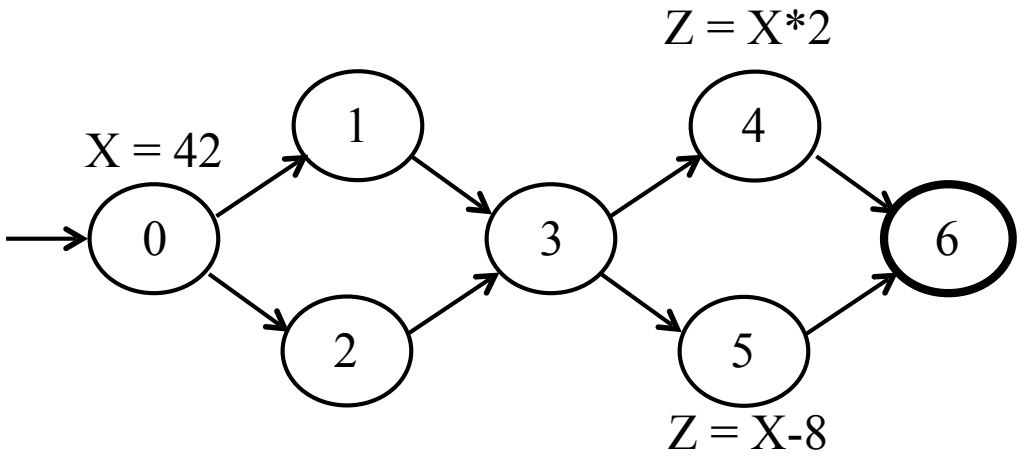
- در نهایت، تمامی تعریف‌ها به تمامی استفاده‌ها ختم شود. (همه مسیرهای بین این دو)

All-du-paths coverage (ADUPC) : For each set $S = du\ (ni, nj, v)$, TR contains every path d in S.

مثالی از آزمون مبتنی بر جریان داده

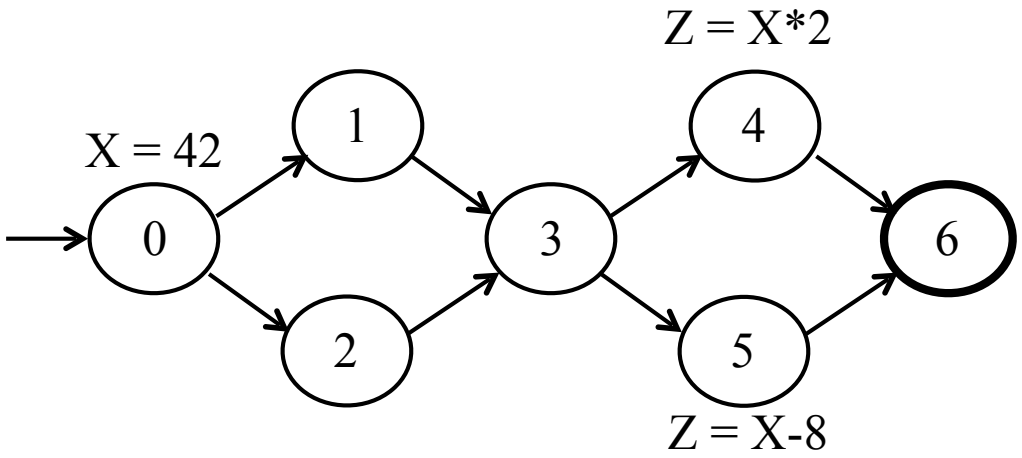


مثالی از آزمون مبتنی بر جریان داده



All-defs for *X*
[0, 1, 3, 4]

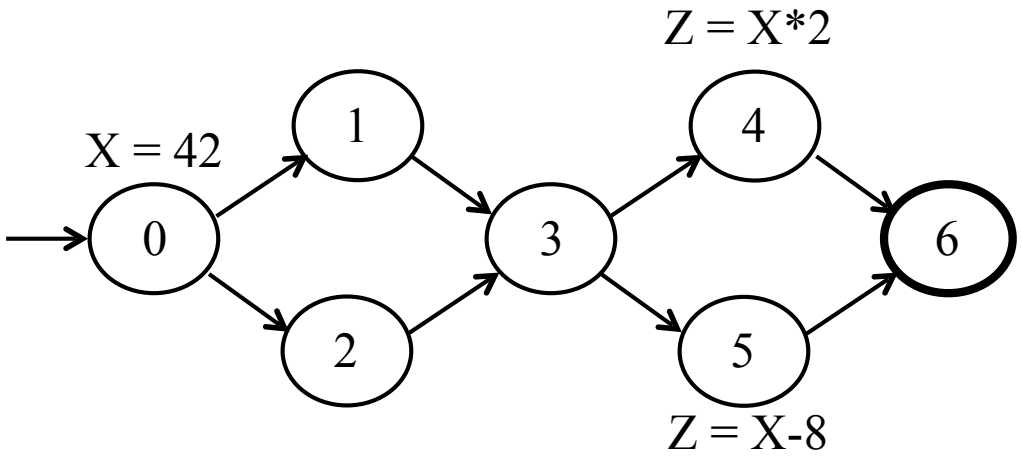
مثالی از آزمون مبتنی بر جریان داده



All-defs for *X*
[0, 1, 3, 4]

All-uses for *X*
[0, 1, 3, 4]
[0, 1, 3, 5]

مثالی از آزمون مبتنی بر جریان داده



All-defs for *X*
[0, 1, 3, 4]

All-uses for *X*
[0, 1, 3, 4]
[0, 1, 3, 5]

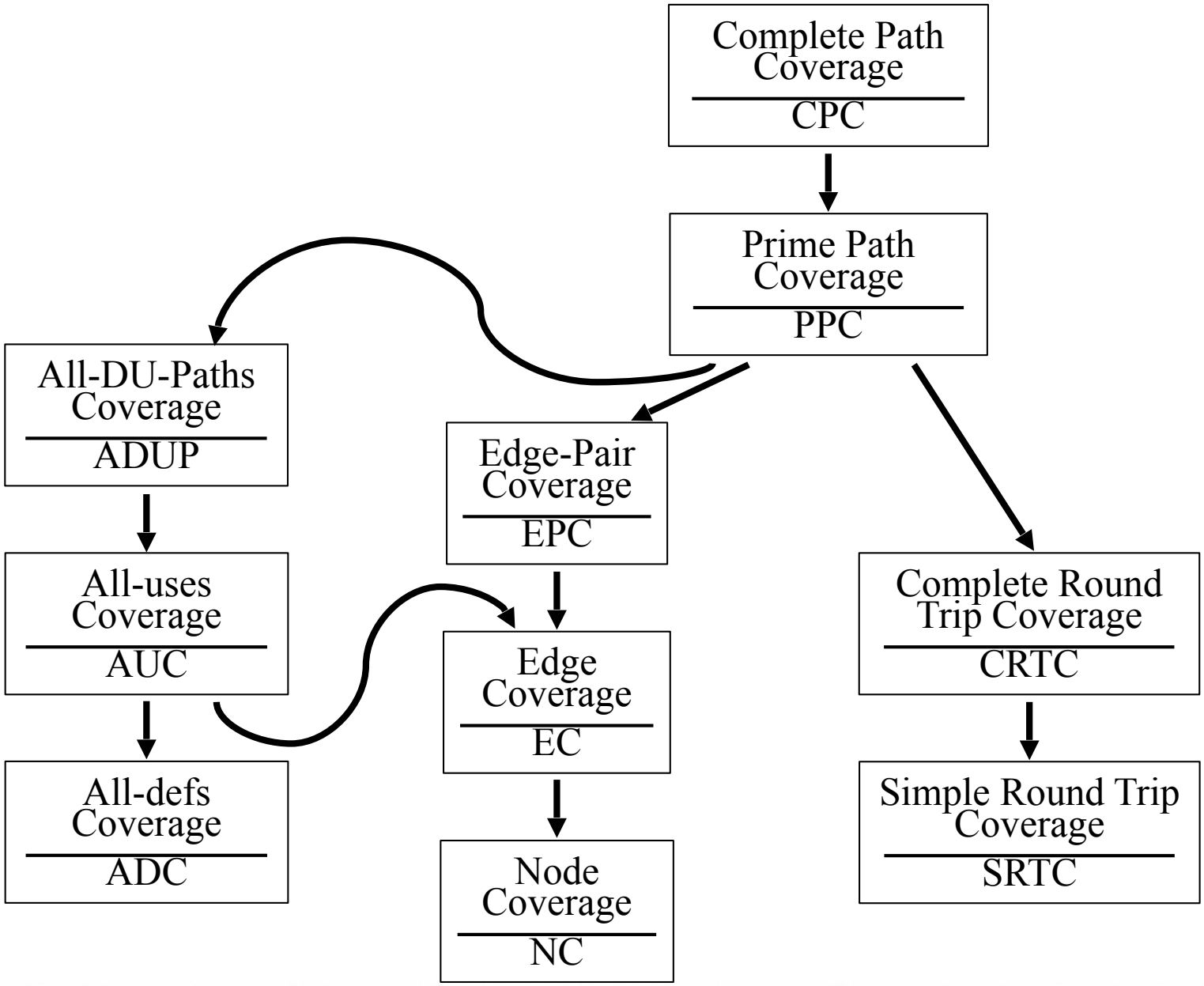
All-du-paths for *X*
[0, 1, 3, 4]
[0, 2, 3, 4]
[0, 1, 3, 5]
[0, 2, 3, 5]

Example: TestPat

```
public int pat (char[] subject, char[] pattern)
{
// Post: if pattern is not a substring of subject,
    return -1
//       else return (zero-based) index where
    the pattern (first)
//       starts in subject
   final int NOTFOUND = -1;
   int  iSub = 0, rtnIndex = NOTFOUND;
   boolean isPat  = false;
   int subjectLen = subject.length;
   int patternLen = pattern.length;
while (isPat == false && iSub + patternLen - 1 <
    subjectLen)
   {
     if (subject [iSub] == pattern [0])
     {
       rtnIndex = iSub; // Starting at zero
       isPat = true;
       for (int iPat = 1; iPat < patternLen; iPat ++)
       {
         if (subject[iSub + iPat] != pattern[iPat])
         {
           rtnIndex = NOTFOUND;
           isPat = false;
           break;  // out of for loop
         }
       }
     }
     iSub ++;
   }
   return (rtnIndex);
}
}
```

Graph Coverage Criteria Subsumption

Graph Coverage Criteria Subsumption



پایان پوشش‌های گراف