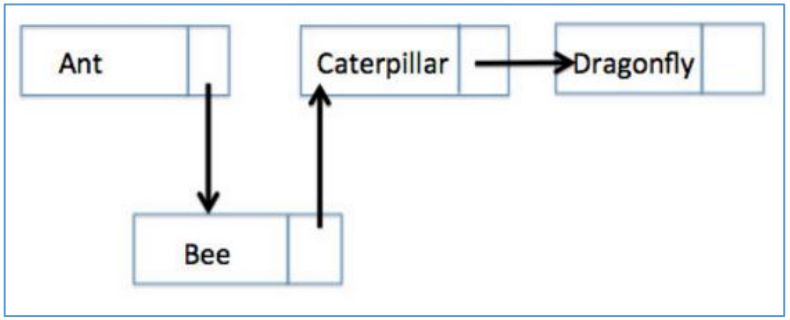
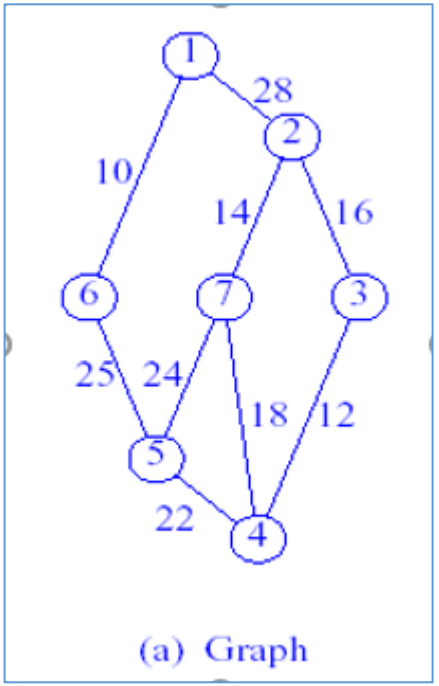
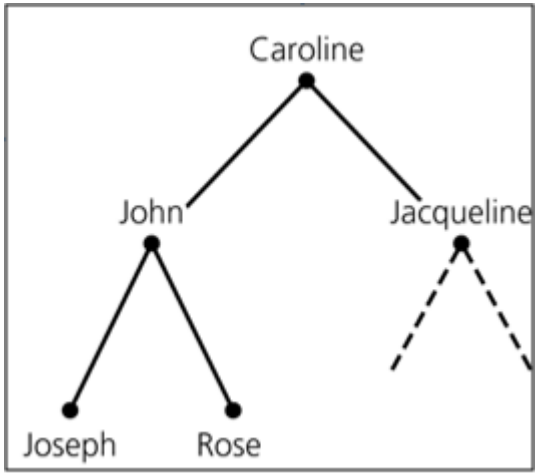
(Ch 25) GraphViz Module

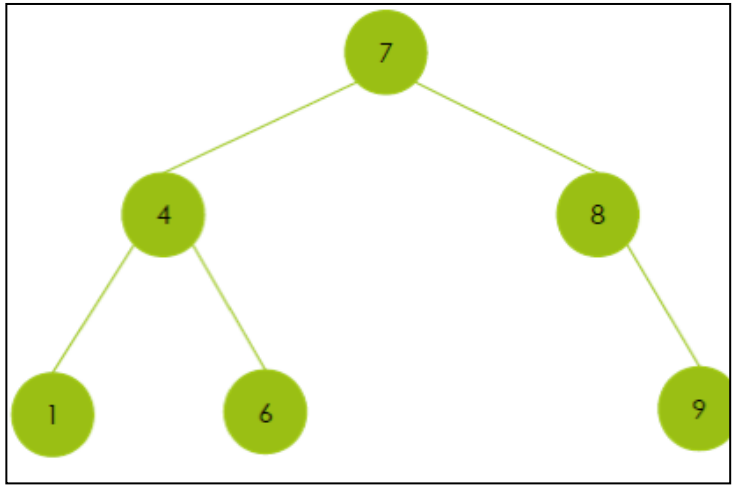
Table of Contents

What If We Want to Draw Data Structures?









** Tkinter?

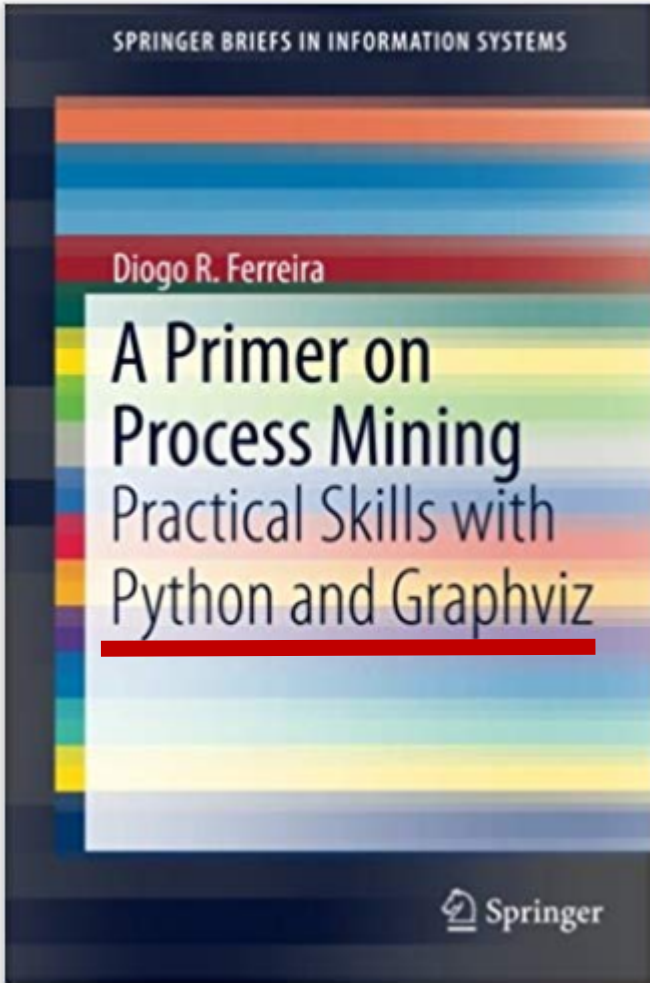
What is Graphviz?

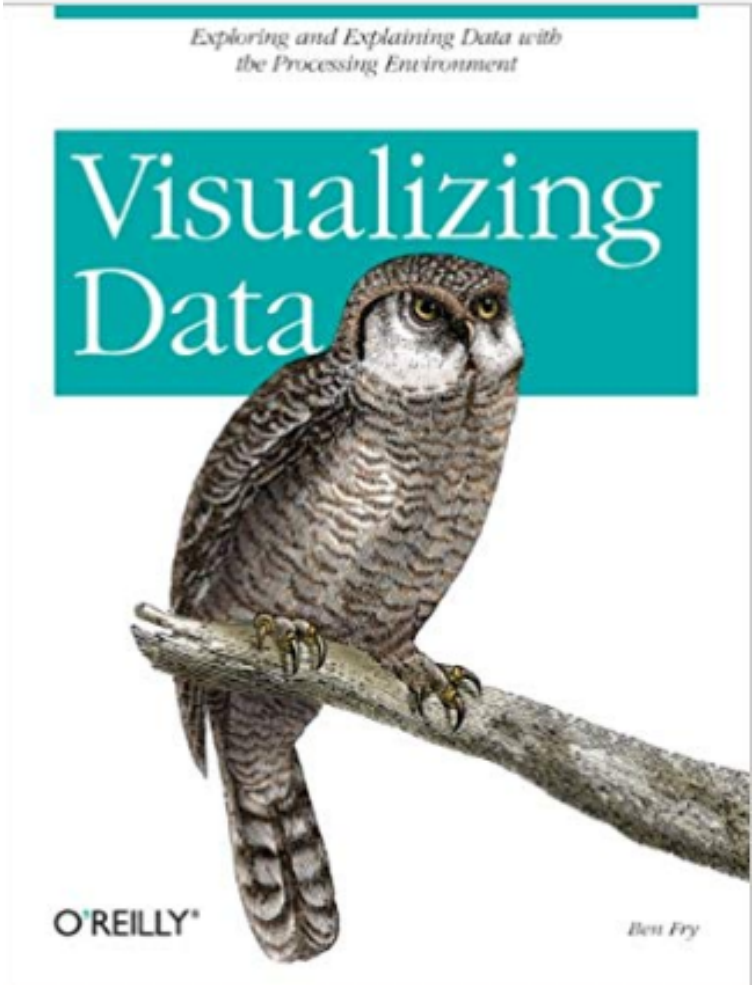
- Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks

- What is Graphviz?
  - Open source graph visualization software
  - www.graphviz.org
  - Originated from AT&T BellLab, 1991 (written in C)
  - Representing structural information as diagrams
    - Eg) Graphs and Networks

- What is Graphviz module in Python?
  - Python Interface to Graphviz
  - Create, edit, and draw graphs using python to access the Graphviz

Books on Python GraphViz





Graphviz DOT Languge

- Graphviz has a graph description language named the DOT language

gv file 생성

```
graph G {
  e
  subgraph clusterA {
    a -- b;
    subgraph clusterC {
      C -- D;
    }
  }
  subgraph clusterB {
    d -- f
  }
  d -- D
  e -- clusterB
  clusterC -- clusterB
}
```

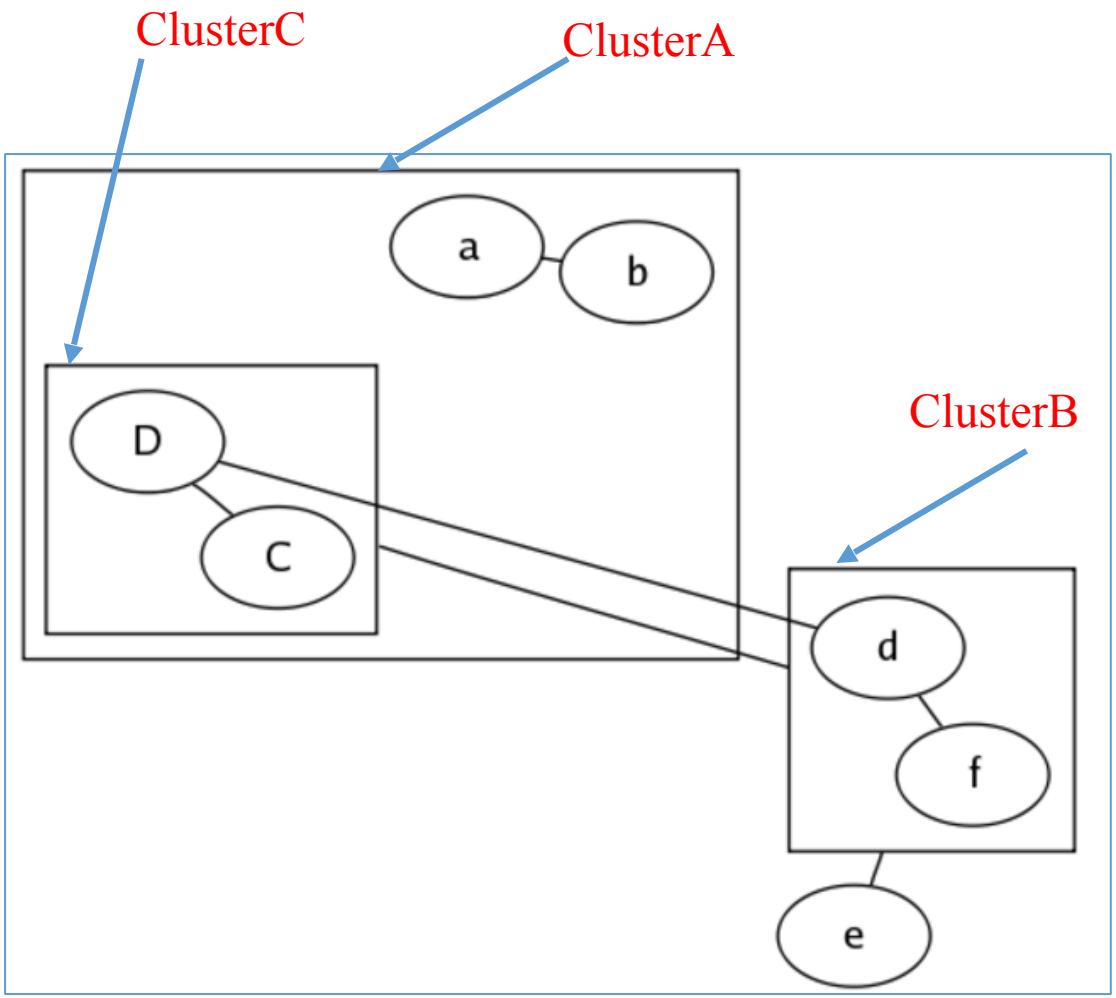
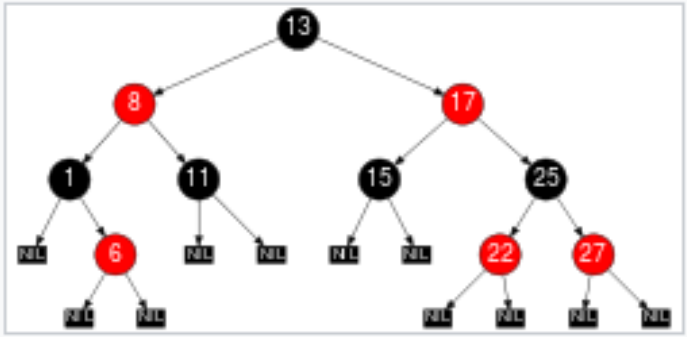


그림 file 생성

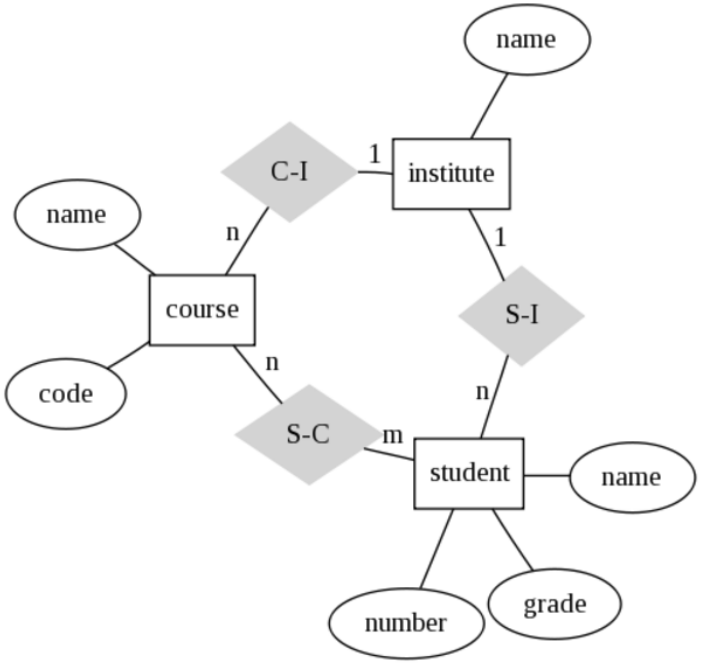
Graphviz Tools

- Graphviz has a set of tools that can generate and/or process DOT files
- Layout Engines: NEATO, FDP, TWOPI, CIRCO, etc….



A Red-Black Tree plotted by Graphviz.

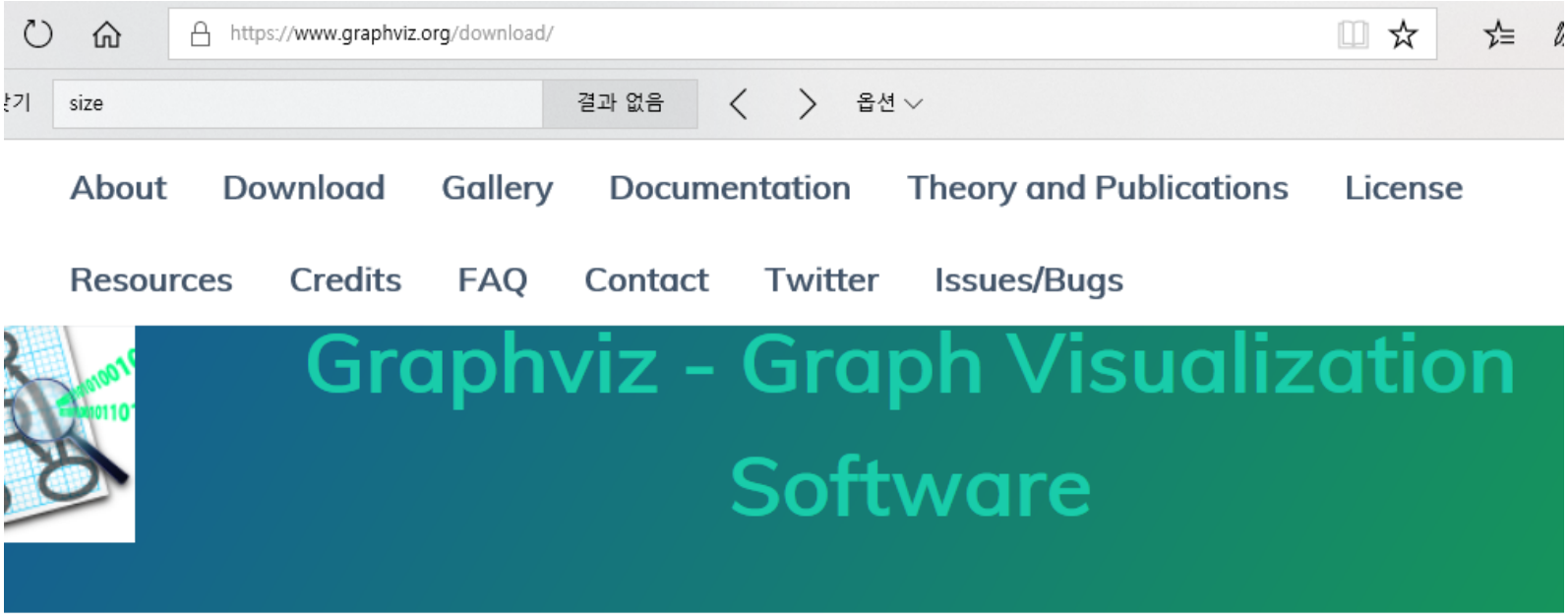
Undirected graph showing adjacency of the 48 contiguous United States



Entity Relation Diagram drawn by NEATO

Graphviz Installation [1/3]

- Graphviz download 링크 접속 (2018-03-06기준)
  - https://www.graphviz.org/download/



Download

Graphviz Installation [2/3]

- OS에 맞는 설치 파일 선택
  - Windows 기준 : Stable 2.38 Windows install packages 클릭

Windows

- Development Windows install packages
- Stable 2.38 Windows install packages
- Cygwin Ports* provides a port of Graphviz to Cygwin.
- WinGraphviz* Win32/COM object (dot/neato library for Visual Basic and ASP).

Mostly correct notes for building Graphviz on Windows can be found here.

Graphviz Installation [3/3]

- 설치파일 다운로드 및 실행

Windows Packages

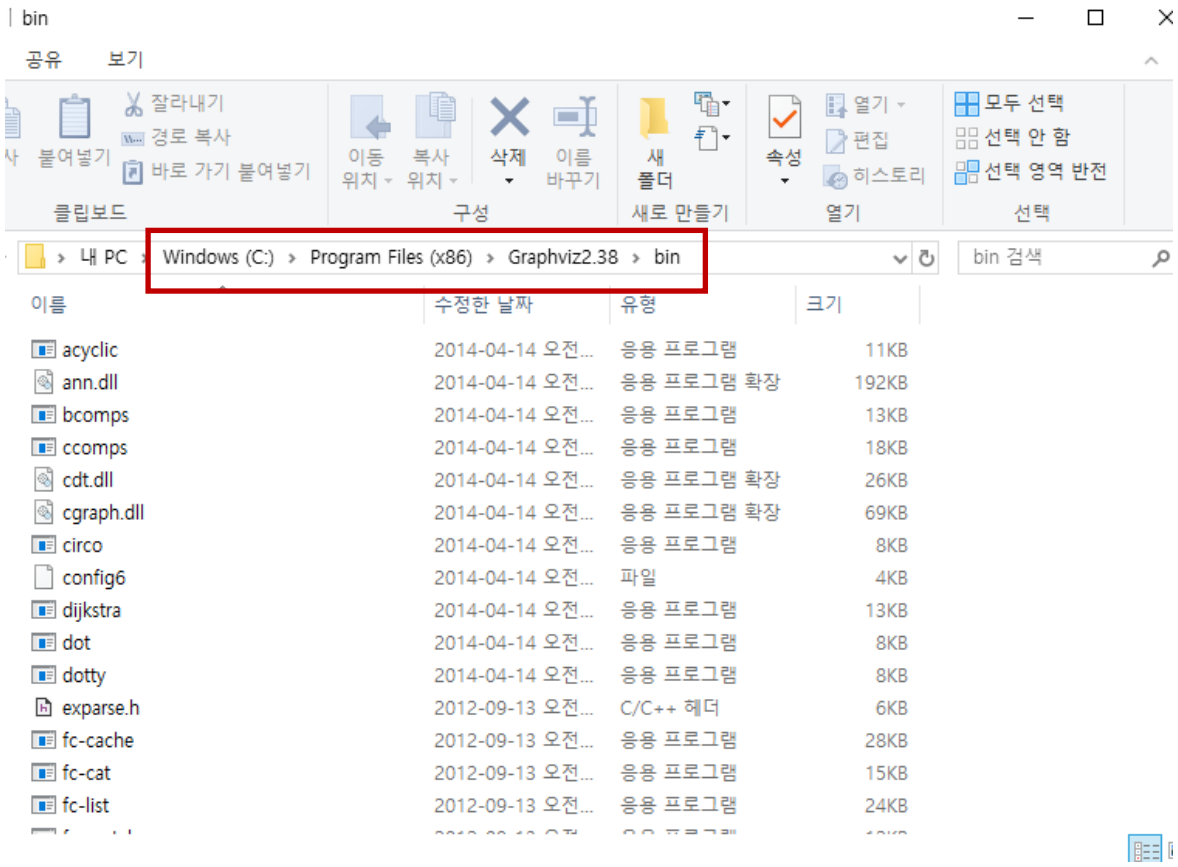
Note: These Visual Studio packages do not alter the PATH variable or access the registry at all. If you wish to use the command-line interface to Graphviz or are using some other program that calls a Graphviz program, you will need to set the PATH variable yourself.

2.38 Stable Release

- graphviz-2.38.msi
- graphviz-2.38.zip

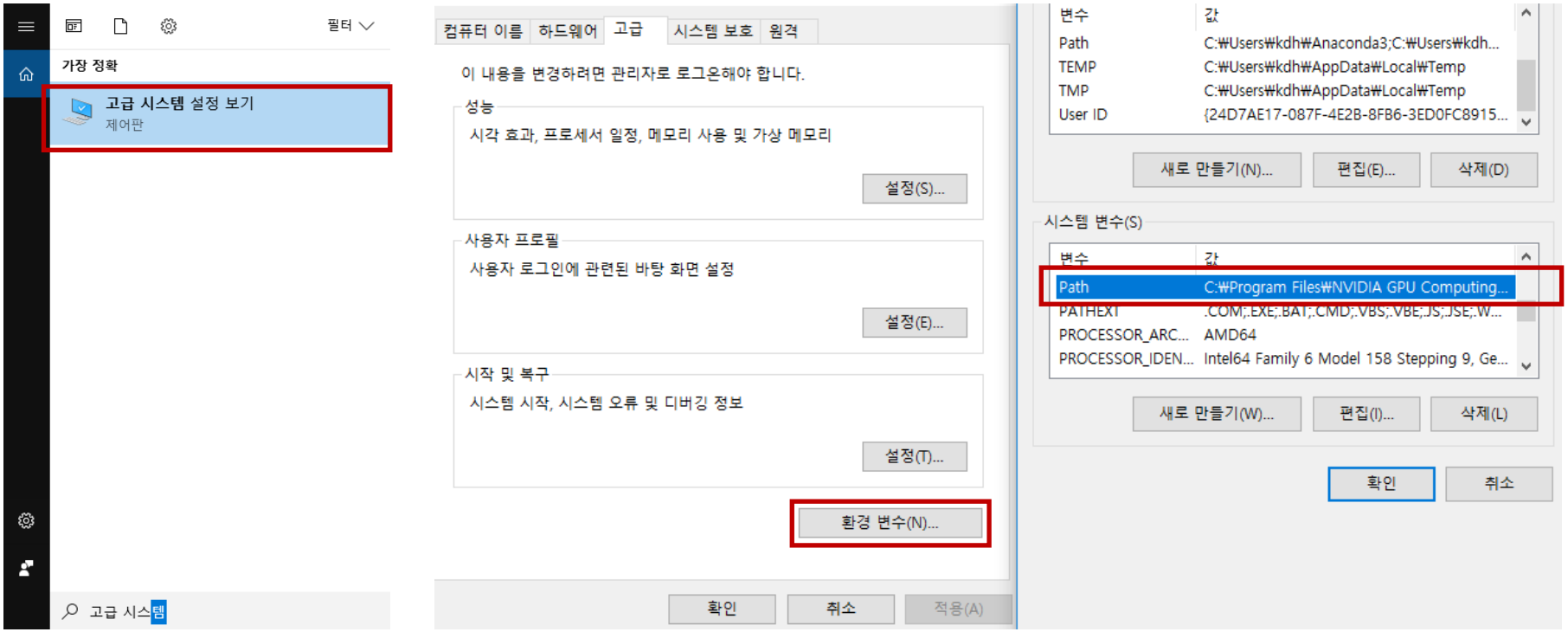
Graphviz 설치를 위한 환경 변수 설정 [1/3]

- 설치완료 후 폴더 경로 복사
  - 기본경로: C:\Program Files (x86)\Graphviz2.38\bin



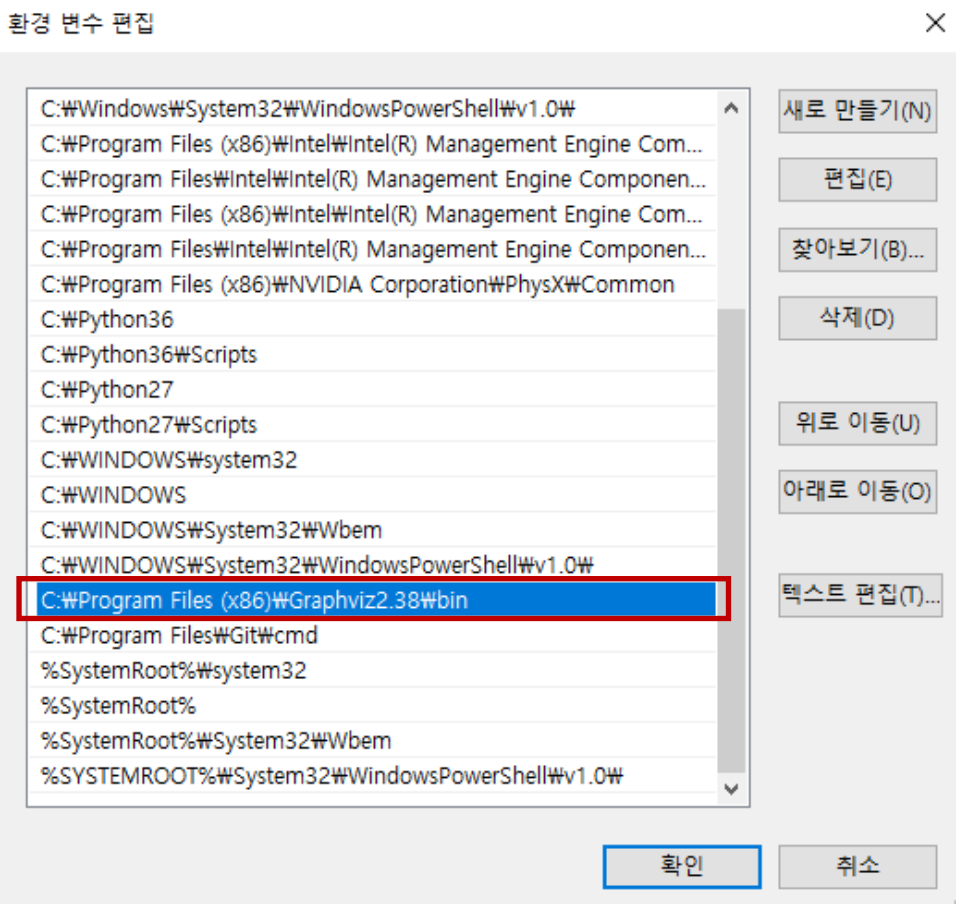
Graphviz 설치를 위한 환경 변수 설정 [2/3]

- 복사한 경로를 시스템 환경변수에 등록
  - Windows검색 – 고급 시스템 설정 보기 – 환경 변수 클릭
  - Path 더블클릭



Graphviz 설치를 위한 환경 변수 설정 [3/3]

- 복사한 경로를 시스템 환경변수에 등록
  - 새로만들기 클릭 – 복사한 경로 추가
  - 확인 후 컴퓨터 재부팅!



Install “graphviz” module in Python

- pip을 사용하여 graphviz 모듈을 파이썬에 설치

```
C:₩>python -m pip install graphviz
```

- 정상적으로 설치되었는지 import 확인

```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import graphviz
>>>
```

Graphviz Quick Example: GraphViz Code [1/4]

- Create a Digraph object:

```
>>> from graphviz import Digraph

>>> dot = Digraph(comment='The Round Table')

>>> dot
<graphviz.dot.Digraph object at 0x...>
```

방향그래프 클래스 생성

- Add nodes and edges:

```
>>> dot.node('A', 'King Arthur')
>>> dot.node('B', 'Sir Bedevere the Wise')
>>> dot.node('L', 'Sir Lancelot the Brave')

>>> dot.edges(['AB', 'AL'])
>>> dot.edge('B', 'L', constraint='false')
```

노드 추가 (name, label)

A→B, A→L 인 edge들 추가

B →L 인 edge 하나 추가

constraint = True: directed edge의 source가 destination 보다 rank가 높게 취급

Graphviz Quick Example: gv file 생성 [2/4]

- Check the generated DOT source code:

```
>>> print(dot.source)
// The Round Table
digraph {
    A [label="King Arthur"]
    B [label="Sir Bedevere the Wise"]
    L [label="Sir Lancelot the Brave"]
    A -> B
    A -> L
    B -> L [constraint=false]
}
```

source에 code가 누적

GV file is a document which includes descriptions about graphs and written using the DOT Language

- Save and render the source code : (with render( ) function)
  - 'file name' +.gv file 생성됨 (dot language format)
  - 'file name' +.gv + .pdf file 생성됨 (default : pdf)

True인 경우, 저장된 그림파일을 새 창으로 불러온다.

```
>>> dot.render('test-output/round-table.gv', view=True)
'test-output/round-table.gv.pdf'
```

dot source → round-table.gv

dot source → round-table.gv.pdf

Graphviz Quick Example: pdf file 생성 [3/4]

```
>>> dot.render('test-output/round-table.gv', view=True)
```

dot source → round-table.gv
dot source → round-table.gv.pdf

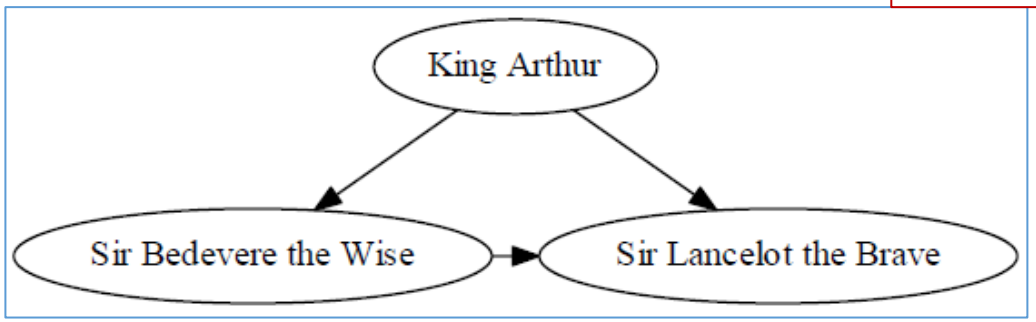
- test-output/round-table.gv
  - Text Editor로 열 수 있음

round-table.gv

```
// The Round Table
digraph {
    A [label="King Arthur"]
    B [label="Sir Bedevere the Wise"]
    L [label="Sir Lancelot the Brave"]
    A -> B
    A -> L
    B -> L [constraint=false]
}
```

- test-output/round-table.gv.pdf

round-table.gv

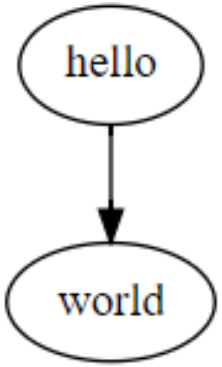


Graphviz Quick Example [4/4]
Interworking with Jupyter Notebook

- Can be displayed directly inside a Jupyter notebook
- Ex) Displaying within the Jupyter notebook

```
In [1]: import graphviz as gv

In [2]: d = gv.Digraph()
        d.edge('hello', 'world')
        d
```

Out [2]:




```
In [ ]:
```

Graphviz Module Function List

1. Graph, Digraph Class

1.1 graphviz.Graph() , graphviz.Digraph()

1.2 attr()

1.3 clear()

1.4 copy()

1.5 edge()

1.6 edges()

1.7 node()

1.8 pipe()

1.9 render()

1.10 save()

1.11 subgraph()

1.12 view()

2. Source Class

2.1 graphviz.source()

2.1 copy()

2.2 pipe()

2.3 render()

2.4 save()

2.5 view()

3. Low-level Functions

3.1 graphviz.render()

3.2 graphviz.pipe()

3.3 graphviz.view()

4. Other

4.1 graphviz.version()

(25) GraphViz Module

Table of Contents

Graphviz Module: Graph, Digraph Class [1/2]

- Graphviz module provides two classes
  - Graph Class: undirected graph
  - Digraph Class: directed graph

```
graph1 = Graph(comment = 'This is a undirected graph')
graph2 = Digraph(comment='This is a directed graph')
```

- Graph and Digraph classes
  - Same API
  - Descriptions in the DOT language

graph1, graph2 object에
node( ), edge( ) 등을 적용

Digraph class – .gv file

```
// This is a directed graph
digraph {
    A [label="King Arthur"]
    B [label="Sir Bedevere the Wise"]
    L [label="Sir Lancelot the Brave"]
    A -> B
    A -> L
    B -> L [constraint=false]
}
```

Graph class – .gv file

```
// This is a undirected graph
graph {
    A [label="King Arthur"]
    B [label="Sir Bedevere the Wise"]
    L [label="Sir Lancelot the Brave"]
    A -- B
    A -- L
    B -- L [constraint=false]
}
```

Graphviz Module: Graph, Digraph Class [2/2]

class **graphviz.Graph**(*name=None, comment=None, filename=None, directory=None, format=None, engine=None, encoding=None, graph_attr=None, node_attr=None, edge_attr=None, body=None, strict=False*)

class **graphviz.Digraph**(*name=None, comment=None, filename=None, directory=None, format=None, engine=None, encoding=None, graph_attr=None, node_attr=None, edge_attr=None, body=None, strict=False*)

Parameters:

- **name** – Graph name used in the source code.
- **comment** – Comment added to the first line of the source.
- **filename** – Filename for saving the source (defaults to *name* + '.gv').
- **directory** – (Sub)directory for source saving and rendering.
- **format** – Rendering output format ('pdf', 'png', ...).
- **engine** – Layout command used ('dot', 'neato', ...).
- **encoding** – Encoding for saving the source.
- **graph_attr** – Mapping of (attribute, value) pairs for the graph.
- **node_attr** – Mapping of (attribute, value) pairs set for all nodes.
- **edge_attr** – Mapping of (attribute, value) pairs set for all edges.
- **body** – Iterable of verbatim lines to add to the graph body.
- **strict** (*bool*) – Rendering should merge multi-edges.

Dictionary

Graphviz Module: Styling Parameters

- Changing the appearance of graph, nodes, and edges
  - Use the **graph_attr**, **node_attr**, and **edge_attr** arguments

```
>>> ps = Digraph(name='pet-shop', node_attr={'shape': 'plaintext'})

>>> ps.node('parrot')
>>> ps.node('dead')
>>> ps.edge('parrot', 'dead')
```

모든 node에 적용

- After creation, they can be edited on the graph object

```
>>> ps.graph_attr['rankdir'] = 'LR'
>>> ps.edge_attr.update(arrowhead='vee', arrowsize='2')
```

가능하면 Left to Right로 lay out

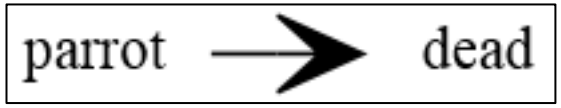
```
>>> ps.render("test_out\pet-shop", view = True)
```

ps source → pet-shop.gv, pet-shop.gv.pdf

```
digraph "pet-shop" {
    graph [rankdir=LR]
    node [shape=plaintext]
    edge [arrowhead=vee arrowsize=2]
    parrot
    dead
    parrot -> dead
}
```

Attribute들

gv 파일



pdf 파일

Graphviz Module: Functions of Graph & Digraph [1/7]

- attr( ): To directly add attribute statements
  - Affecting all following graph, node, or edge items within the same (sub-)graph

attr(*kw=None, _attributes=None, **attrs*)

Add a general or graph/node/edge attribute statement.

Parameters:
- **kw** – Attributes target (None or 'graph', 'node', 'edge').
- **attrs** – Attributes to be set (must be strings, may be empty).

graph_attr
node_attr
edge_attr

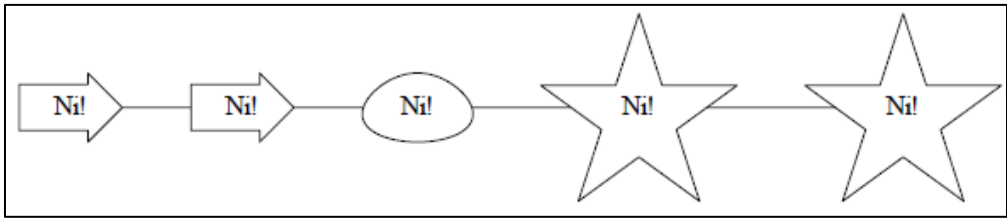
Dictionary를 update 해도 되고

GraphViz Code

```
ni = Graph('ni')

ni.attr('node', shape='rarrow')
ni.node('1', 'Ni!')
ni.node('2', 'Ni!')
ni.node('3', 'Ni!', shape='egg')

ni.attr('node', shape='star')
ni.node('4', 'Ni!')
ni.node('5', 'Ni!')

ni.attr(rankdir='LR')
ni.edges(['12', '23', '34', '45'])
```

Reset shape attr

그림 파일



ni.render("test_out\drawing Ni", view = True)

Graphviz Module: Functions of Graph & Digraph [2/7]

clear(*keep_attrs=False*)

Reset content to an empty body, clear graph/node/egde_attr mappings.

Parameters: **keep_attrs** (*bool*) – preserve graph/node/egde_attr mappings

node(*name, label=None, _attributes=None, **attrs*)

Create a node.

Parameters:

- **name** – Unique identifier for the node inside the source.
- **label** – Caption to be displayed (defaults to the node name).
- **attrs** – Any additional node attributes (must be strings).

Graphviz Module: Functions of Graph & Digraph [3/7]

edge(*tail_name, head_name, label=None, _attributes=None, **attrs*)

Create an edge between two nodes.

Parameters:

- **tail_name** – Start node identifier.
- **head_name** – End node identifier.
- **label** – Caption to be displayed near the edge.
- **attrs** – Any additional edge attributes (must be strings).

edges(*tail_head_iter*)

Create a bunch of edges.

Parameters: **tail_head_iter** – Iterable of (tail_name, head_name) pairs.

Graphviz Module: Functions of Graph & Digraph [4/7]

render(*filename=None, directory=None, view=False, cleanup=False*)

Save the source to file and render with the Graphviz engine.

Parameters:

- **filename** – Filename for saving the source (defaults to *name* + '.gv')
- **directory** – (Sub)directory for source saving and rendering.
- **view** (*bool*) – Open the rendered result with the default application.
- **cleanup** (*bool*) – Delete the source file after rendering.

Returns: The (possibly relative) path of the rendered file.

Raises:

- `graphviz.ExecutableNotFound` – If the Graphviz executable is not found.
- `subprocess.CalledProcessError` – If the exit status is non-zero.
- `RuntimeError` – If viewer opening is requested but not supported.

Graphviz Module: Functions of Graph & Digraph [5/7]

- Subgraph() : Method for adding a subgraph to an instance [1/3]

subgraph(*graph=None, name=None, comment=None, graph_attr=None, node_attr=None, edge_attr=None, body=None*)

Add the current content of the given sole *graph* argument as subgraph or return a context manager returning a new graph instance created with the given (*name*, *comment*, etc.) arguments whose content is added as subgraph when leaving the context manager's `with`-block.

Parameters:

- **graph** – An instance of the same kind (`Graph`, `Digraph`) as the current graph (sole argument in non-with-block use).
- **name** – Subgraph name (with-block use).
- **comment** – Subgraph comment (with-block use).
- **graph_attr** – Subgraph-level attribute-value mapping (with-block use).
- **node_attr** – Node-level attribute-value mapping (with-block use).
- **edge_attr** – Edge-level attribute-value mapping (with-block use).
- **body** – Verbatim lines to add to the subgraph body (with-block use).

Graphviz Module: Functions of Graph & Digraph [6/7]

- Subgraph() [2/3]
  - First usage option, with graph as the only argument :

```
>>> p = Graph(name='parent')
>>> p.edge('spam', 'eggs')

>>> c = Graph(name='child', node_attr={'shape': 'box'})
>>> c.edge('foo', 'bar')

>>> p.subgraph(c)
```

  - Second usage, with a with-block (omitting the graph argument) :

```
>>> p = Graph(name='parent')
>>> p.edge('spam', 'eggs')

>>> with p.subgraph(name='child', node_attr={'shape': 'box'}) as c:
...     c.edge('foo', 'bar')
```

```
>>> p.render("test_out\parent", view = True)
```

Graphviz Module: Functions of Graph & Digraph [7/7]

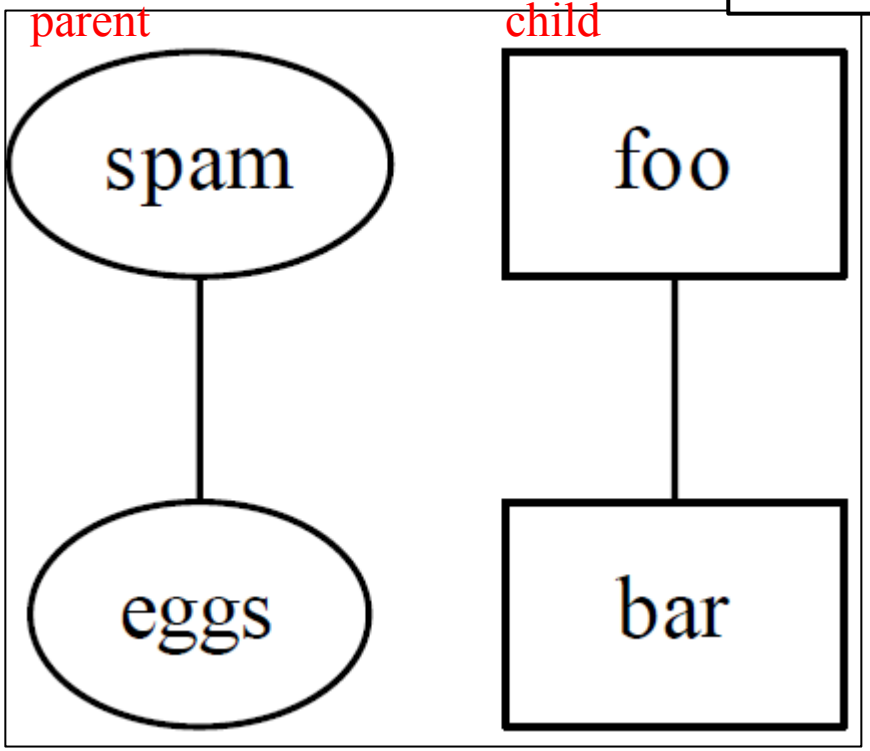
- Subgraph() [3/3]
  - Both produce the same result :

```
>>> p.render("test_out\parent", view = True)
```

gv 파일

```
graph parent {
    spam -- eggs
    subgraph child {
        node [shape=box]
        foo -- bar
    }
}
```

pdf 파일



Graphviz Module: Attributes of Graph & Digraph Class

format

그래프 그림파일의 확장자를 변경할 수 있음 **(default : pdf)**

The output format used for rendering ('pdf', 'png', ...).

방법1

```
dot = Graph(comment='The Round Table', fotmat='jpg')
```

방법2

```
dot.format = 'jpg' # or 'pdf', 'png', 'svg' ...
```

round-table.gv	이미지(jpg) 파일
round-table.gv	Adobe Acrobat Document
round-table.gv	PNG 파일
round-table.gv	SVG 파일

engine

그래프를 그리는 레이아웃 엔진을 변경할 수 있음 **(default : dot)**

The layout commmand used for rendering ('dot', 'neato', ...).

방법1

```
dot = Graph(comment='The Round Table', engine='neato')
```

방법2

```
dot.engine = 'neato' # or 'dot', 'fdp', 'sfdp', 'twopi', 'circo' ...
```

(25) GraphViz Module

Table of Contents

Graphviz Layout Engines [1/6]

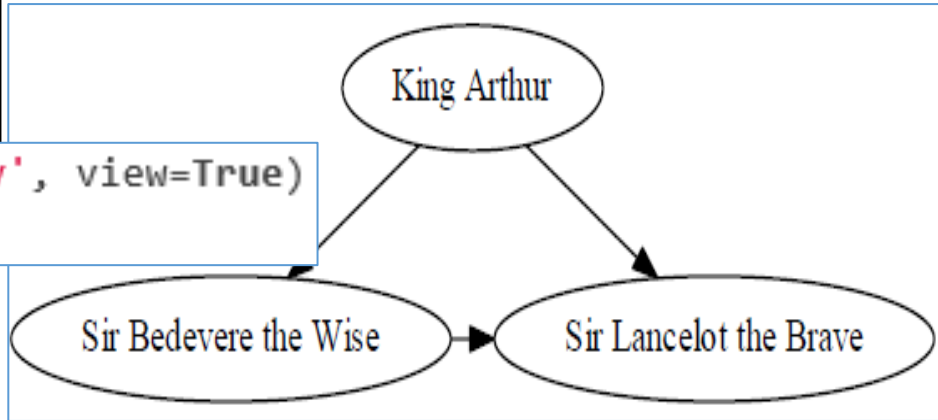
- Dot : “Hierarchical" or layered drawings of directed graphs
  - Default tool to use if edges have directionality
  - Graph 또는 Digraph 클래스 생성 시, engine의 디폴트

```
>>> dot = Digraph(comment = ‘The Round Table’, engine = “dot”)
```

```
>>> from graphviz import Digraph

>>> dot = Digraph(comment='The Round Table')

>>> dot
<graphviz.dot.Digraph object at 0x...>
```

```
>>> dot.node('A', 'King Arthur')
>>> dot.node('B', 'Sir Bedevere the Wise')
>>> dot.node('L', 'Sir Lancelot the Brave')

>>> dot.edges(['AB', 'AL'])
>>> dot.edge('B', 'L', constraint='false')
```

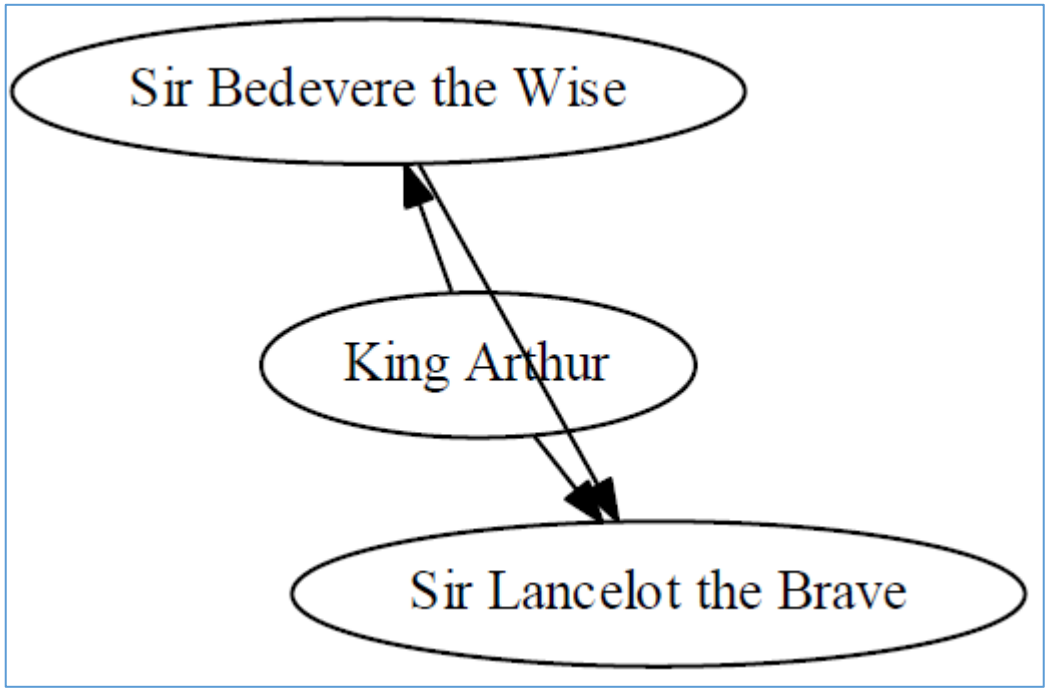
```
>>> dot.render('test-output/round-table.gv', view=True)
'test-output/round-table.gv.pdf'
```



Graphviz Layout Engines [2/6]

- Neato: “Spring model" layouts
  - Default tool to use if the graph is not too large

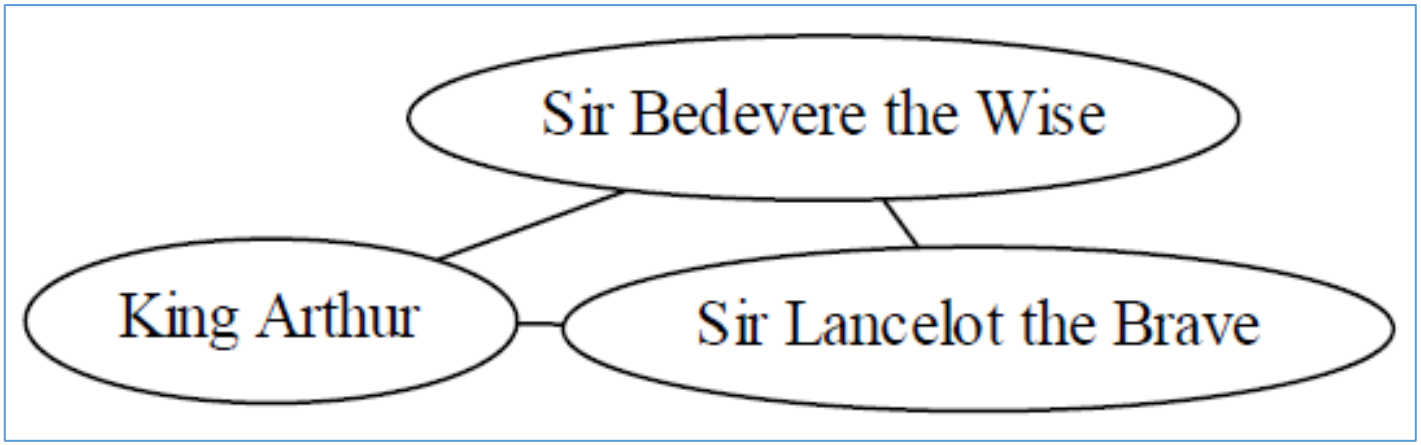
```
>>> dot = Digraph(comment = ‘The Round Table’, engine = “neato”)
```



Graphviz Layout Engines [3/6]

- fdp : “Spring model" layouts similar to those of neato
  - Layout engine for undirected graphs

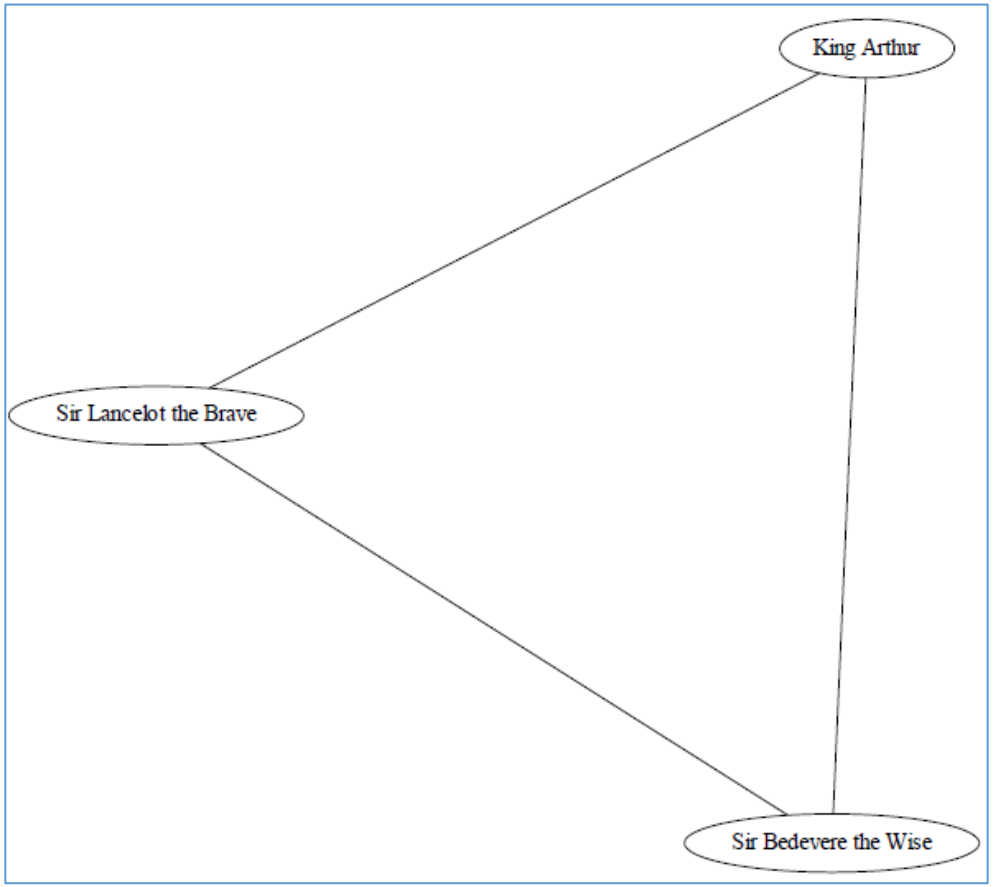
```
>>> dot = Digraph(comment = ‘The Round Table’, engine = “fdp”)
```



Graphviz Layout Engines [4/6]

- sfdp : Layout engine for undirected graphs that scales to very large graphs (Multi-scale version of fdp for the layout of large graphs)

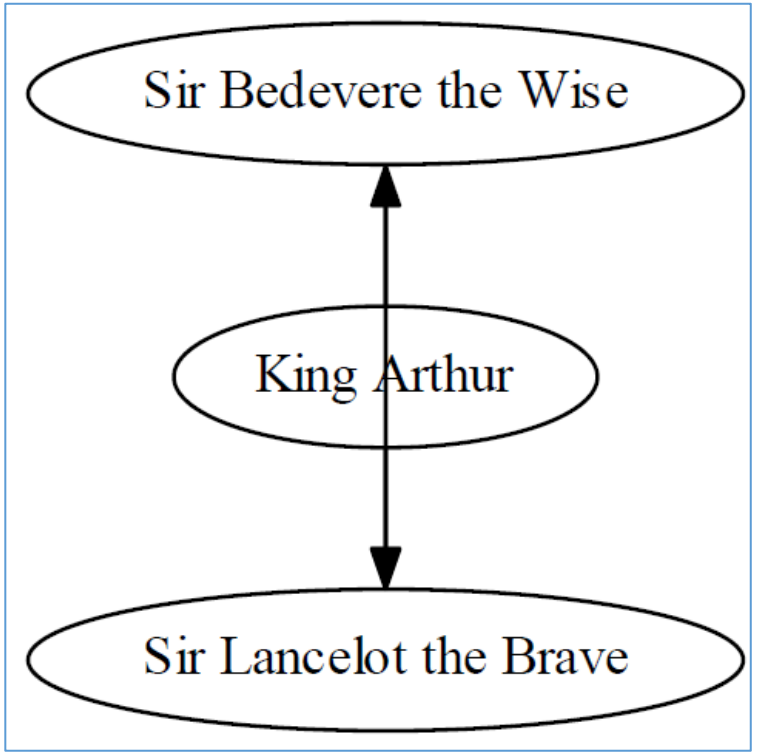
```
>>> dot = Digraph(comment = 'The Round Table', engine = "sfdp")
```



Graphviz Layout Engines [5/6]

- twopi : For radial graph layout, after Graham Wills 97.
  - Nodes are placed on concentric circles depending their distance from a given root node

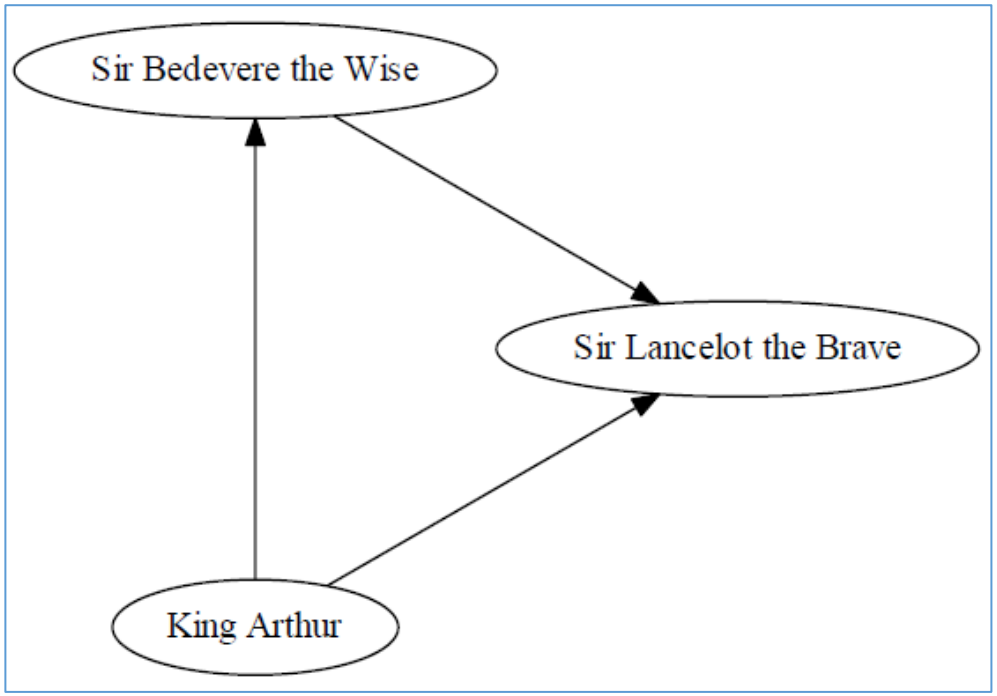
```
>>> dot = Digraph(comment = 'The Round Table', engine = "twopi")
```



Graphviz Layout Engines [6/6]

- circo: For circular graph layouts, after Six and Tollis 99, Kauffman and Wiese 02.
  - Suitable for certain diagrams of multiple cyclic structures, such as certain telecommunications networks

```
>>> dot = Digraph(comment = 'The Round Table', engine = "circo")
```

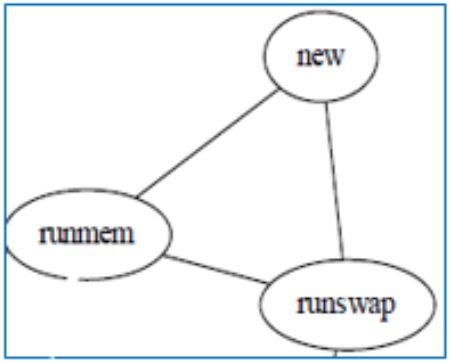


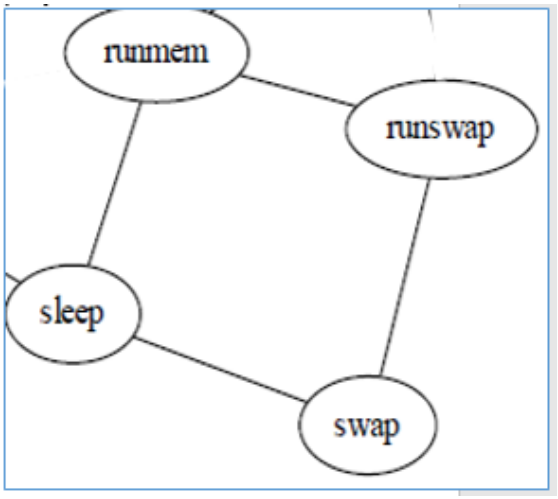
(25) GraphViz Module

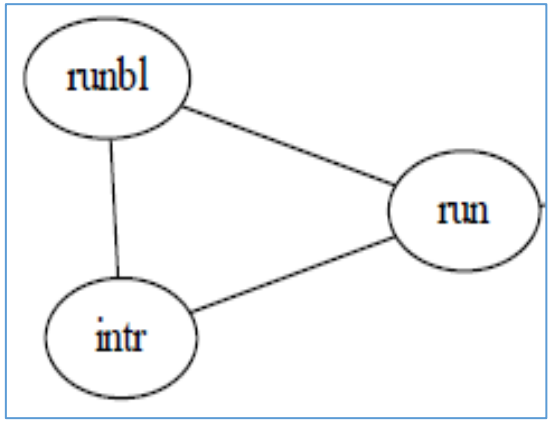
Table of Contents

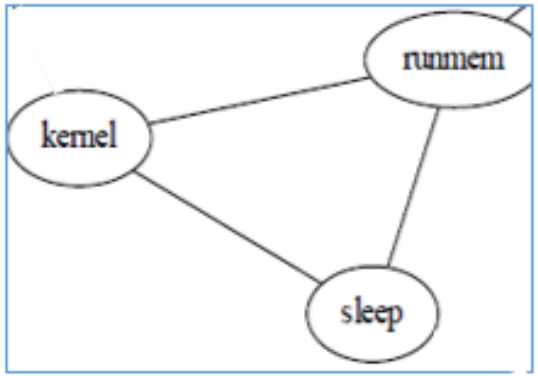
Graphviz – Code Example 1 [1/4]

- 10개 node로 구성된 Directed Graph를 그리려고 한다
- Node는 업무Process를 표현하며 관련 node들과 edge들로 형성한다
- 업무의 이름들은: runswap, runmen, sleep, swap, numbl, intr, run, new, kernel, sleep
- (runswap, runmen, sleep, swap) 4개의 업무들은 밀접하게 4각형 형성
- (numbl, intr, run) 3개의 업무들은 밀접하게 삼각형을 형성
- (new, runswap, runmen) 3개의 업무들은 밀접하게 삼각형을 형성
- (kernel, sleep, runmen) 3개의 업무들은 밀접하게 삼각형을 형성
- (kernel, zombie) 2개의 업무에 edge가 형성
- (kernel, run) 2개의 업무에 edge가 형성

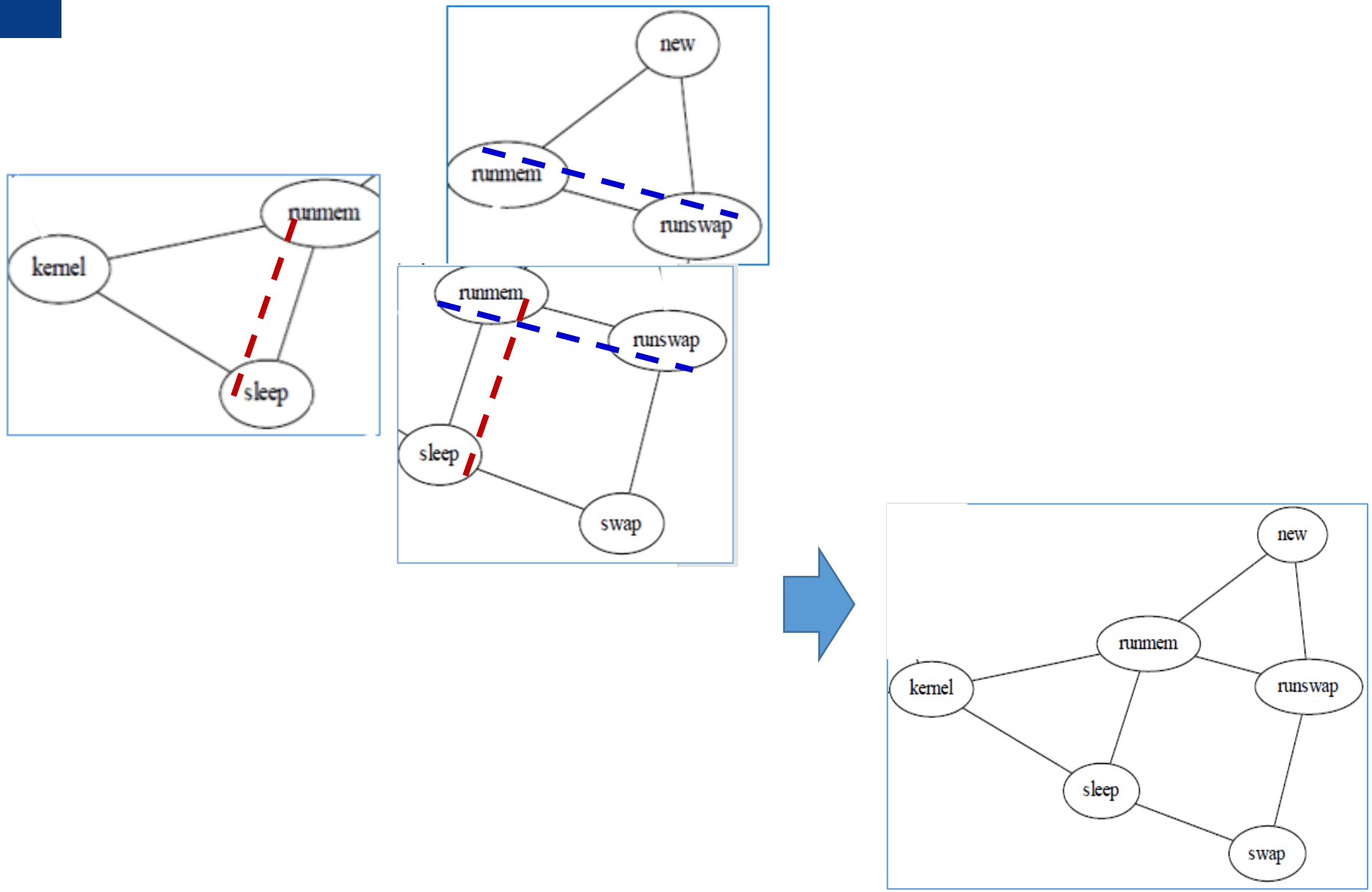








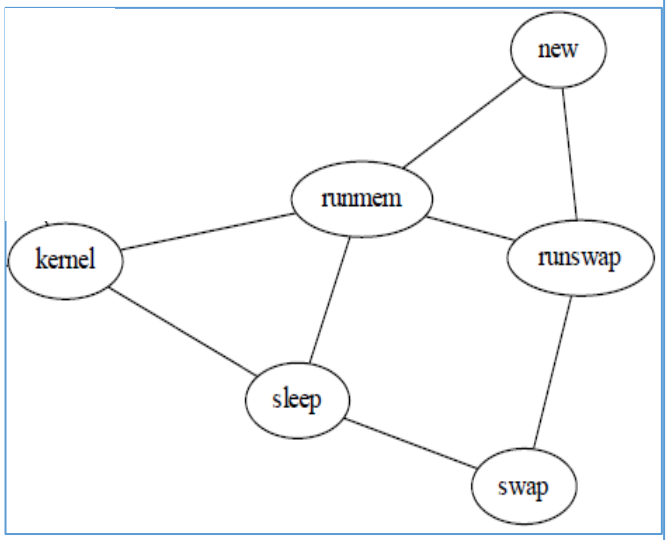
Interim Processes



Graphviz – Code Example 1 [2/4]

GraphViz Code

```
from graphviz import Graph

g = Graph('G', filename='gviz/process.gv', engine='sfdp')

g.edge('run', 'intr')
g.edge('intr', 'runbl')
g.edge('runbl', 'run')
g.edge('run', 'kernel')
g.edge('kernel', 'zombie')
g.edge('kernel', 'sleep')
g.edge('kernel', 'runmem')
g.edge('sleep', 'swap')
g.edge('swap', 'runswap')
g.edge('runswap', 'new')
g.edge('runswap', 'runmem')
g.edge('new', 'runmem')
g.edge('sleep', 'runmem')

g.render(view=True)
```



g → process.gv

g → process.gv.pdf

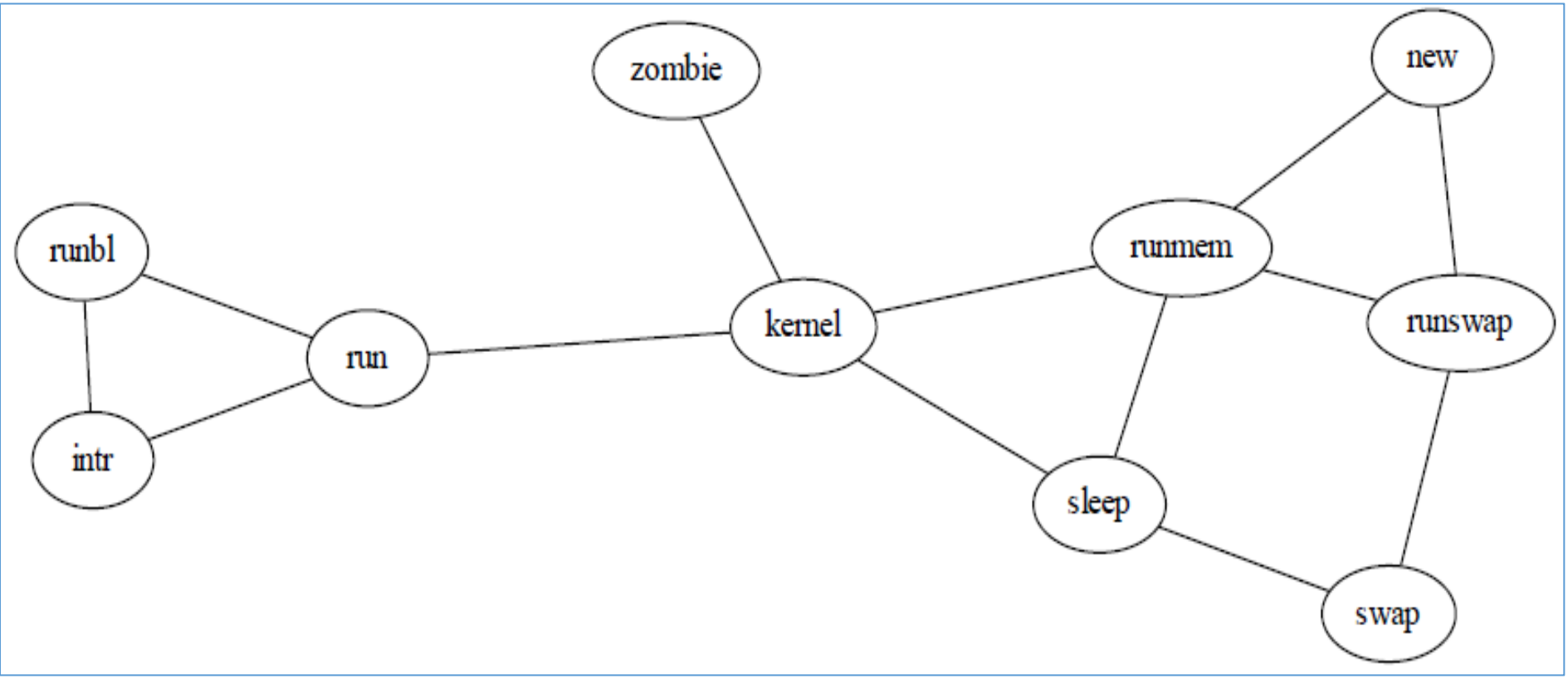
Graphviz – Code Example 1 [3/4]

process.gv 생성

```
graph G {
    run -- intr
    intr -- runbl
    runbl -- run
    run -- kernel
    kernel -- zombie
    kernel -- sleep
    kernel -- runmem
    sleep -- swap
    swap -- runswap
    runswap -- new
    runswap -- runmem
    new -- runmem
    sleep -- runmem
}
```

Graphviz – Code Example 1 [4/4]

process.gv.pdf생성

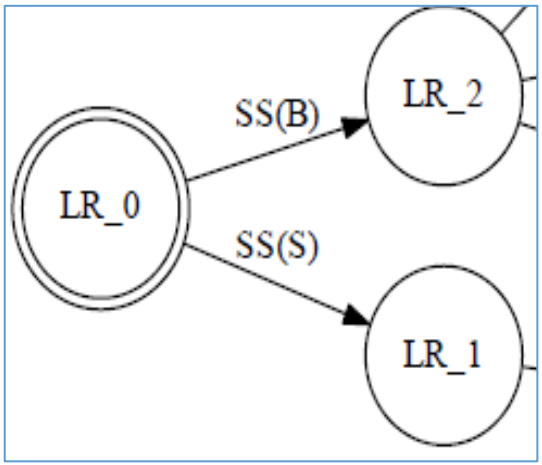


Graphviz – Code Example 2 [1/4]

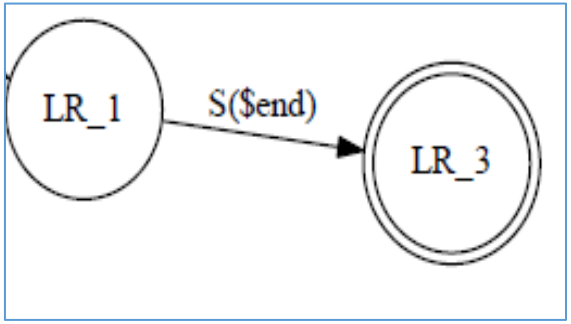
- 9개 node로 구성된 Finite State Machine을 그리려고 한다
- Double Circle Node 는 4개이고, Label은 { LR_0, LR_3, LR_4, LR_8 }
- Single Circle Node 는 5개이고, Label은 { LR_1, LR_2, LR_5, LR_6, LR_7 }
- LR_0 에서 LR_1 로 상태변화가 있으려면 SS(S) 의 event가 있어야 한다
- LR_0 에서 LR_2 로 상태변화가 있으려면 SS(B) 의 event가 있어야 한다
- LR_1 에서 LR_3 로 상태변화가 있으려면 S($end) 의 event가 있어야 한다
- LR_2 에서 LR_4 로 상태변화가 있으려면 S(A) 의 event가 있어야 한다
- LR_2 에서 LR_5 로 상태변화가 있으려면 SS(a) 의 event가 있어야 한다
- LR_2 에서 LR_6 로 상태변화가 있으려면 SS(b) 의 event가 있어야 한다
- LR_6 에서 S(b) 의 event가 발생하면 LR_6 로 상태가 유지되다
- LR_6 에서 LR_5 로 상태변화가 있으려면 S(a) 의 event가 있어야 한다
- LR_5 에서 S(a) 의 event가 발생하면 LR_5 로 상태가 유지되다
- LR_5 에서 LR_7 로 상태변화가 있으려면 S(b) 의 event가 있어야 한다
- LR_7 에서 LR_5 로 상태변화가 있으려면 S(a) 의 event가 있어야 한다
- LR_7 에서 LR_8 로 상태변화가 있으려면 S(b) 의 event가 있어야 한다
- LR_8 에서 LR_5 로 상태변화가 있으려면 S(a) 의 event가 있어야 한다
- LR_8 에서 LR_6 로 상태변화가 있으려면 S(b) 의 event가 있어야 한다

Interim Processes

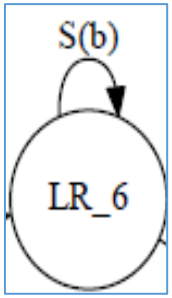
- Double Circle Node 는 4개이고, Label은 { LR_0, LR_3, LR_4, LR_8 }
- Single Circle Node 는 5개이고, Label은 { LR_1, LR_2, LR_5, LR_6, LR_7 }



- LR_0 에서 LR_1 로 상태변화가 있으려면 SS(S) 의 event가 있어야 한다
- LR_0 에서 LR_2 로 상태변화가 있으려면 SS(B) 의 event가 있어야 한다



- LR_1 에서 LR_3 로 상태변화가 있으려면 S($end) 의 event가 있어야 한다



- LR_6 에서 S(b) 의 event가 발생하면 LR_6 로 상태가 유지되다

Graphviz – Code Example 2 [1/3]

GraphViz Code

Default engine (dot)

```
from graphviz import Digraph

f = Digraph('finite_state_machine', filename='gviz/fsm.gv')
f.attr(rankdir='LR', size='8,5')

f.attr('node', shape='doublecircle')
f.node('LR_0')
f.node('LR_3')
f.node('LR_4')
f.node('LR_8')

f.attr('node', shape='circle')
f.edge('LR_0', 'LR_2', label='SS(B)')
f.edge('LR_0', 'LR_1', label='SS(S)')
f.edge('LR_1', 'LR_3', label='S($end)')
f.edge('LR_2', 'LR_6', label='SS(b)')
f.edge('LR_2', 'LR_5', label='SS(a)')
f.edge('LR_2', 'LR_4', label='S(A)')
f.edge('LR_5', 'LR_7', label='S(b)')
f.edge('LR_5', 'LR_5', label='S(a)')
f.edge('LR_6', 'LR_6', label='S(b)')
f.edge('LR_6', 'LR_5', label='S(a)')
f.edge('LR_7', 'LR_8', label='S(b)')
f.edge('LR_7', 'LR_5', label='S(a)')
f.edge('LR_8', 'LR_6', label='S(b)')
f.edge('LR_8', 'LR_5', label='S(a)')

f.view()
```

f → fsm.gv

f → fsm.gv.pdf

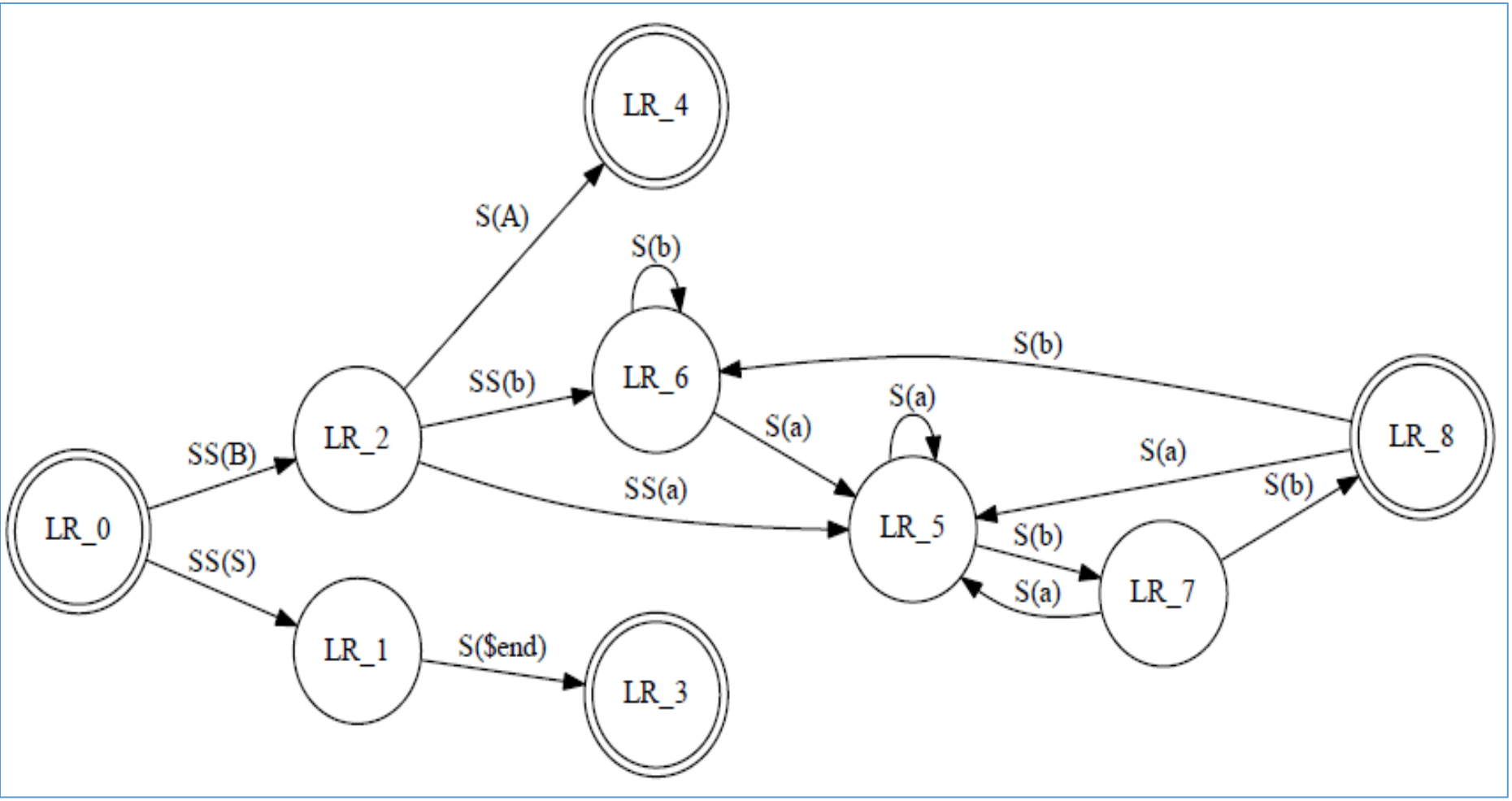
Graphviz – Code Example 2 [2/3]

fsm.gv 생성

```
digraph finite_state_machine {
    rankdir=LR size="8,5"
    node [shape=doublecircle]
    LR_0
    LR_3
    LR_4
    LR_8
    node [shape=circle]
    LR_0 -> LR_2 [label="SS(B)"]
    LR_0 -> LR_1 [label="SS(S)"]
    LR_1 -> LR_3 [label="S($end)"]
    LR_2 -> LR_6 [label="SS(b)"]
    LR_2 -> LR_5 [label="SS(a)"]
    LR_2 -> LR_4 [label="S(A)"]
    LR_5 -> LR_7 [label="S(b)"]
    LR_5 -> LR_5 [label="S(a)"]
    LR_6 -> LR_6 [label="S(b)"]
    LR_6 -> LR_5 [label="S(a)"]
    LR_7 -> LR_8 [label="S(b)"]
    LR_7 -> LR_5 [label="S(a)"]
    LR_8 -> LR_6 [label="S(b)"]
    LR_8 -> LR_5 [label="S(a)"]
}
```

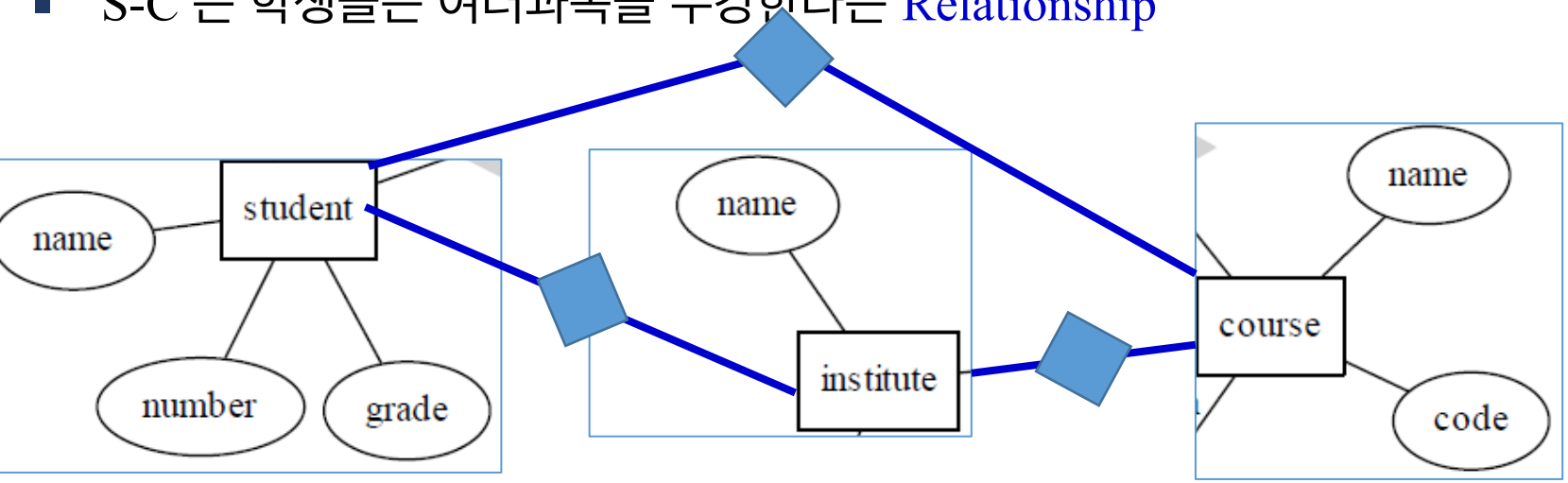
Graphviz – Code Example 2 [3/3]

fsm.gv.pdf 생성



Graphviz – Code Example 3 [1/4]

- 아래의 내용을 반영한 Entity-Relationship Diagram을 그리려고 한다
- Institute, Course, Student 3개의 Entity가 있다
- Institute Entity 는 {name} attribute 를 가진다
- Course Entity 는 {name, code} attribute 를 가진다
- Student Entity 는 {name, number, grade} attribute 를 가진다
- S-I 는 1개 Institute에 여러명의 학생이 다닌다는 Relationship
- C-I 는 1개 Institute에 여러과목이 강의된다는 Relationship
- S-C 는 학생들은 여러과목을 수강한다는 Relationship



Graphviz – Code Example 3 [2/4]

GraphViz Code

```
from graphviz import Graph
e = Graph('ER', filename='gviz/er.gv', engine='neato')

e.attr('node', shape='box')
e.node('course')
e.node('institute')
e.node('student')

e.attr('node', shape='ellipse')
e.node('name0', label='name')
e.node('name1', label='name')
e.node('name2', label='name')
e.node('code')
e.node('grade')
e.node('number')

e.attr('node', shape='diamond', style='filled', color='lightgrey')
e.node('C-I')
e.node('S-C')
e.node('S-I')

e.edge('name0', 'course')
e.edge('code', 'course')
e.edge('course', 'C-I', label='n', len='1.00')
e.edge('C-I', 'institute', label='1', len='1.00')
e.edge('institute', 'name1')
e.edge('institute', 'S-I', label='1', len='1.00')
e.edge('S-I', 'student', label='n', len='1.00')
e.edge('student', 'grade')
e.edge('student', 'name2')
e.edge('student', 'number')
e.edge('student', 'S-C', label='m', len='1.00')
e.edge('S-C', 'course', label='n', len='1.00')

e.attr(label=r'\n\nEntity Relation Diagram\ndrawn by NEATO')
e.attr(fontsize='20')
e.view()
```

e → er.gv

e → er.gv.pdf

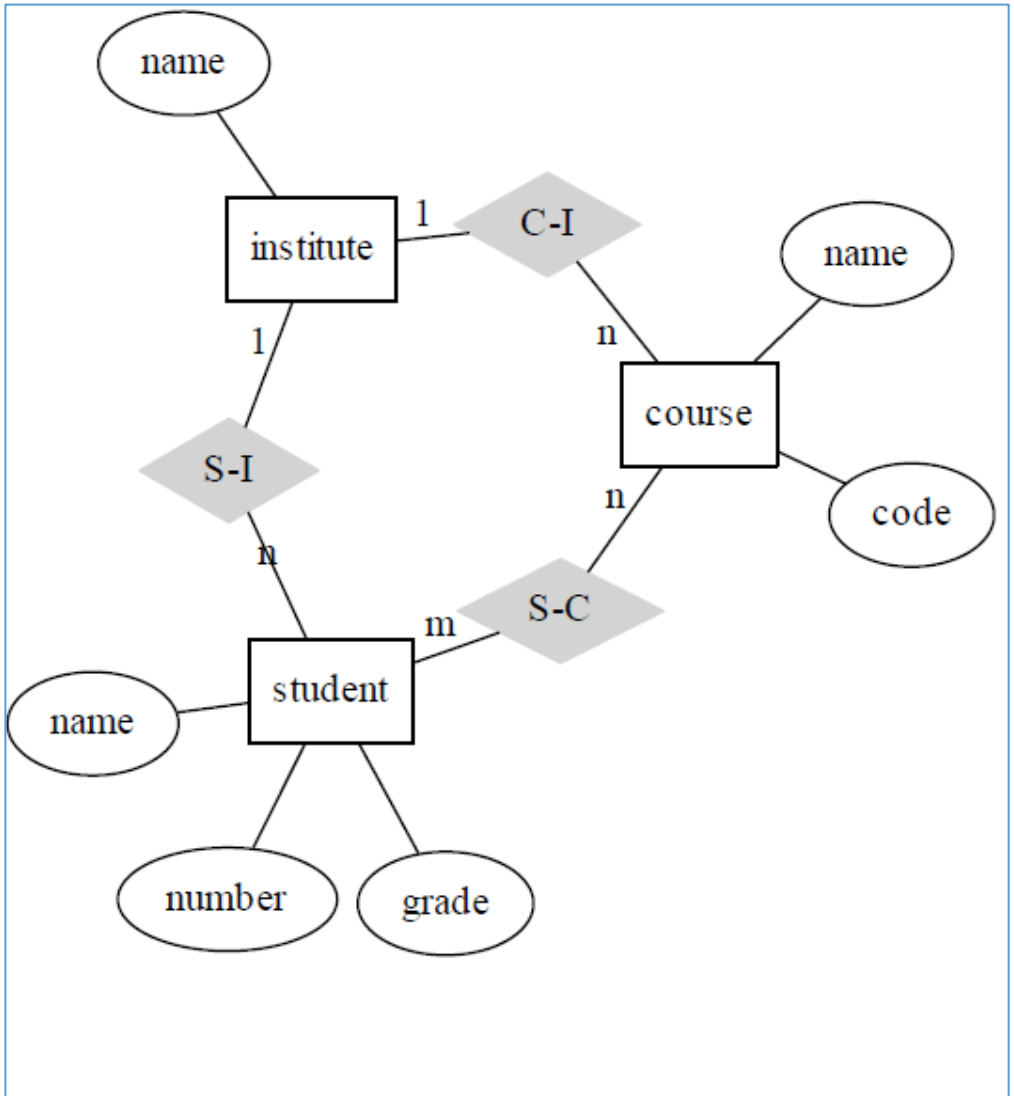
Graphviz – Code Example 3 [3/4]

er.gv 생성

```
graph ER {
    node [shape=box]
    course
    institute
    student
    node [shape=ellipse]
    name0 [label=name]
    name1 [label=name]
    name2 [label=name]
    code
    grade
    number
    node [color=lightgrey shape=diamond style=filled]
    "C-I"
    "S-C"
    "S-I"
    name0 -- course
    code -- course
    course -- "C-I" [label=n len=1.00]
    "C-I" -- institute [label=1 len=1.00]
    institute -- name1
    institute -- "S-I" [label=1 len=1.00]
    "S-I" -- student [label=n len=1.00]
    student -- grade
    student -- name2
    student -- number
    student -- "S-C" [label=m len=1.00]
    "S-C" -- course [label=n len=1.00]
    label="\n\nEntity Relation Diagram\ndrawn by NEATO"
    fontsize=20
}
```

Graphviz – Code Example 3 [4/4]

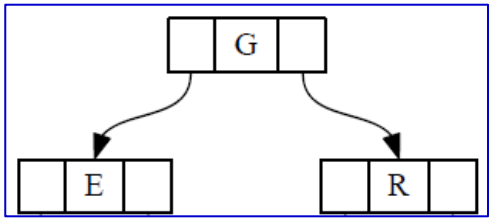
er.gv.pdf 생성

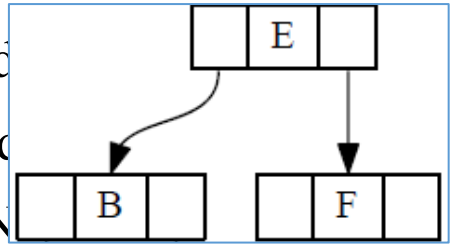


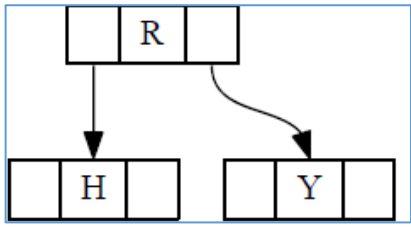
Entity Relation Diagram
drawn by NEATO

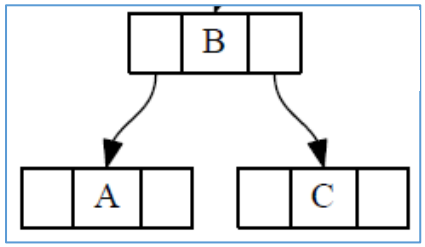
Graphviz – Code Example 4 [1/4]

- 아래의 내용을 반영한 Binary Tree 를 그리려고 한다
- 모든 Node의 구성:
  - lst : pointer to left subtree
  - label
  - rst: pointer to right subtree
- Root Node 는 label G
- Node G: lst ➔ Node E, rst ➔ Node R
- Node E: lst ➔ Node B, rst ➔ Nod
- Node R: lst ➔ Node H, rst ➔ Nod
- Node B: lst ➔ Node A, rst ➔ N





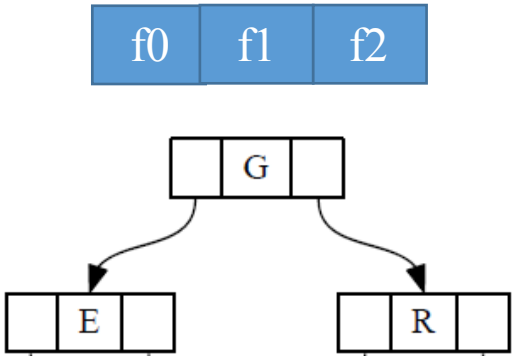




Graphviz – Code Example 4 [2/4]

GrapViz Code

```
from graphviz import Digraph

g = Digraph('g', filename='gviz/btree.gv', node_attr={'shape': 'record', 'height': '.1'})

g.node('node0', '<f0> |<f1> G|<f2> ')
g.node('node1', '<f0> |<f1> E|<f2> ')
g.node('node2', '<f0> |<f1> B|<f2> ')
g.node('node3', '<f0> |<f1> F|<f2> ')
g.node('node4', '<f0> |<f1> R|<f2> ')
g.node('node5', '<f0> |<f1> H|<f2> ')
g.node('node6', '<f0> |<f1> Y|<f2> ')
g.node('node7', '<f0> |<f1> A|<f2> ')
g.node('node8', '<f0> |<f1> C|<f2> ')

g.edge('node0:f2', 'node4:f1')
g.edge('node0:f0', 'node1:f1')
g.edge('node1:f0', 'node2:f1')
g.edge('node1:f2', 'node3:f1')
g.edge('node2:f2', 'node8:f1')
g.edge('node2:f0', 'node7:f1')
g.edge('node4:f2', 'node6:f1')
g.edge('node4:f0', 'node5:f1')

g.view()
```



g → btree.gv

g → btree.gv.pdf

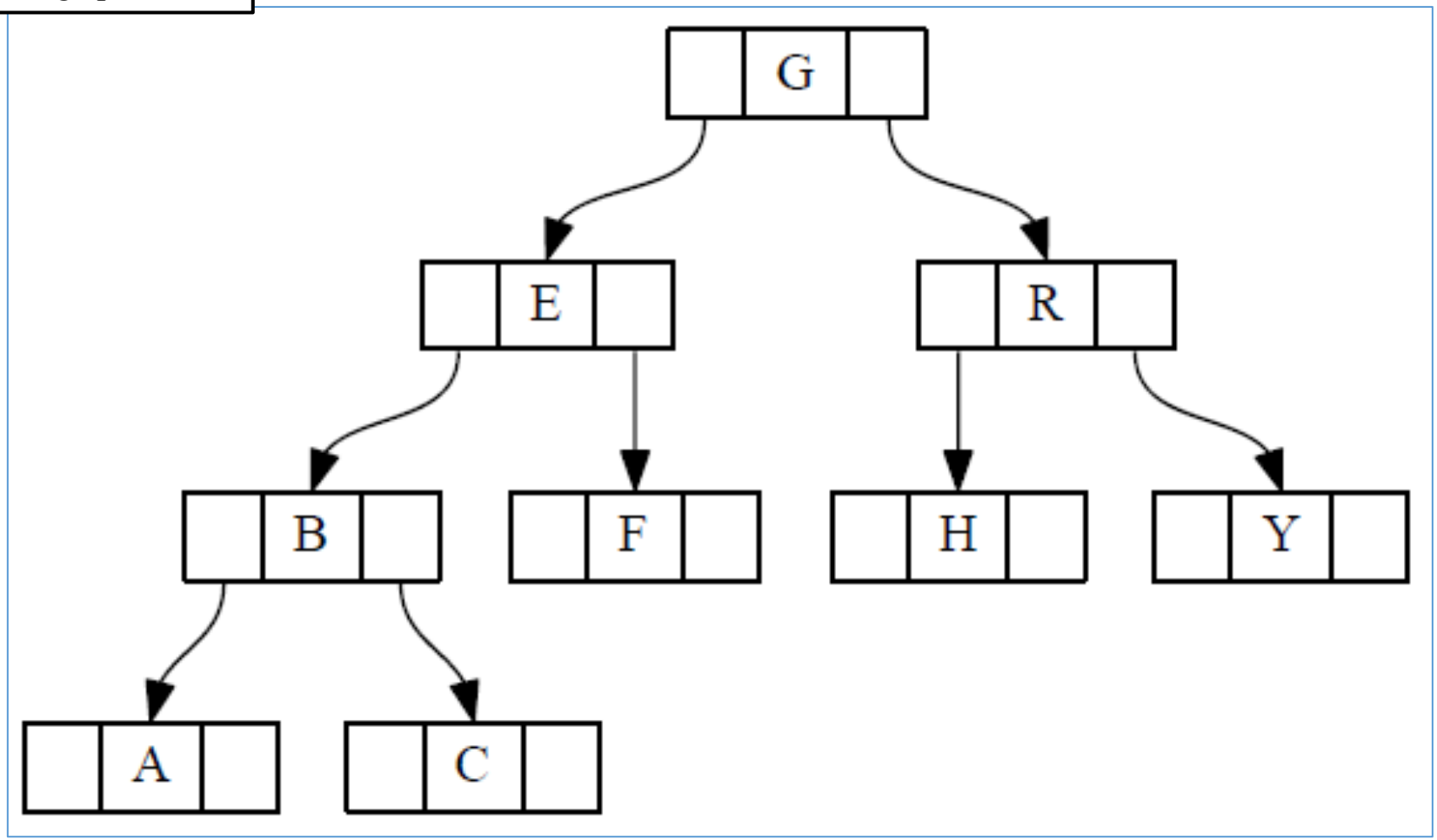
Graphviz – Code Example 4 [3/4]

btree.gv 생성

```
digraph g {
    node [height=.1 shape=record]
    node0 [label="<f0> |<f1> G|<f2> "]
    node1 [label="<f0> |<f1> E|<f2> "]
    node2 [label="<f0> |<f1> B|<f2> "]
    node3 [label="<f0> |<f1> F|<f2> "]
    node4 [label="<f0> |<f1> R|<f2> "]
    node5 [label="<f0> |<f1> H|<f2> "]
    node6 [label="<f0> |<f1> Y|<f2> "]
    node7 [label="<f0> |<f1> A|<f2> "]
    node8 [label="<f0> |<f1> C|<f2> "]
    node0:f2 -> node4:f1
    node0:f0 -> node1:f1
    node1:f0 -> node2:f1
    node1:f2 -> node3:f1
    node2:f2 -> node8:f1
    node2:f0 -> node7:f1
    node4:f2 -> node6:f1
    node4:f0 -> node5:f1
}
```

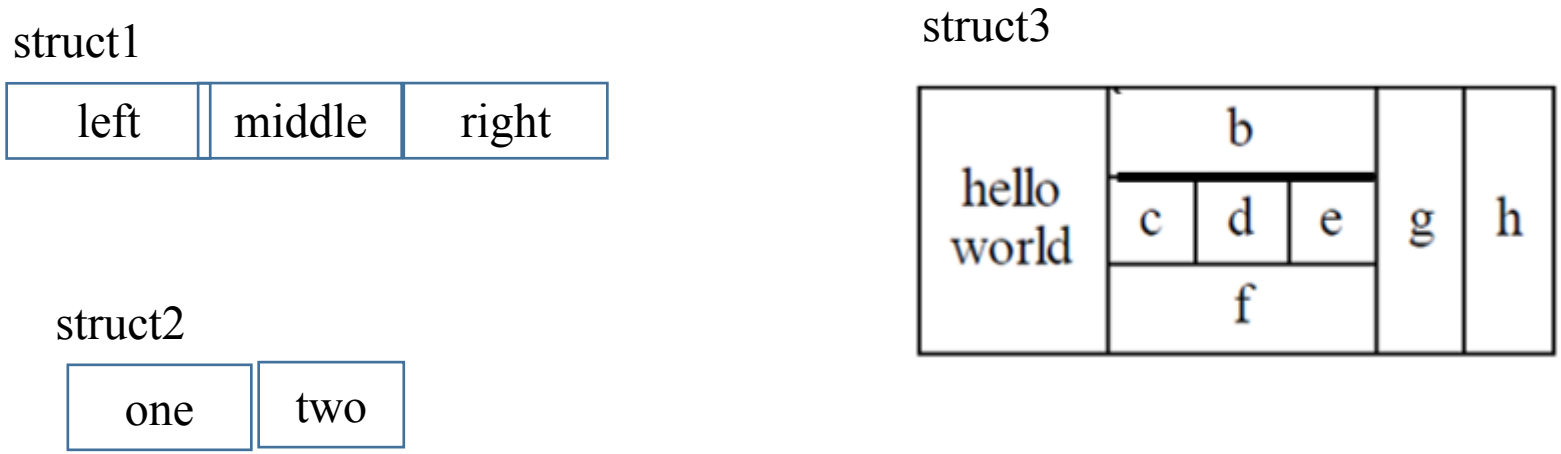
Graphviz – Code Example 4 [4/4]

btree.gv.pdf 생성



Graphviz – Code Example 5 [1/3]

- 아래의 내용을 반영한 Tree 를 그리려고 한다
- Node는 총 3개이며, 아래 그림과 같다
- Struct1 node의 middle 에서 struct2를 pointing 한다
- Struct1 node의 right 에서 struct3의 (c,d,e) 부분을 pointing 한다



Graphviz – Code Example 5 [2/3]

GraphViz Code

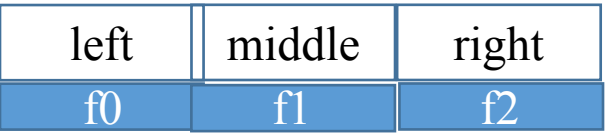
Default engine (dot)

```
from graphviz import Digraph

s = Digraph('structs', filename='gviz/structs_revisited.gv', node_attr={'shape': 'record'})

s.node('struct1', '<f0> left|<f1> middle|<f2> right')
s.node('struct2', '<f0> one|<f1> two')
s.node('struct3', r'hello\nworld |{ b |{c|<here> d|e}| f}| g | h')

s.edges([('struct1:f1', 'struct2:f0'), ('struct1:f2', 'struct3:here')])

s.view()
```

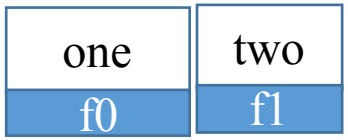
s → structs_revisted.gv

s → structs_revisted.gv.pdf

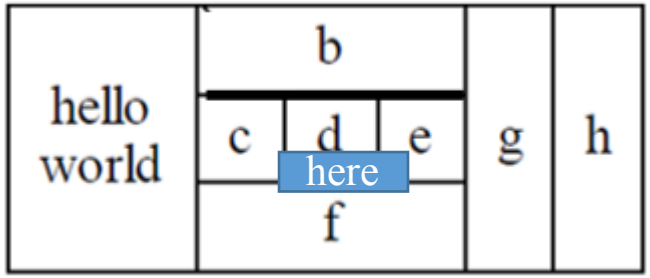
struct1



struct2



struct3

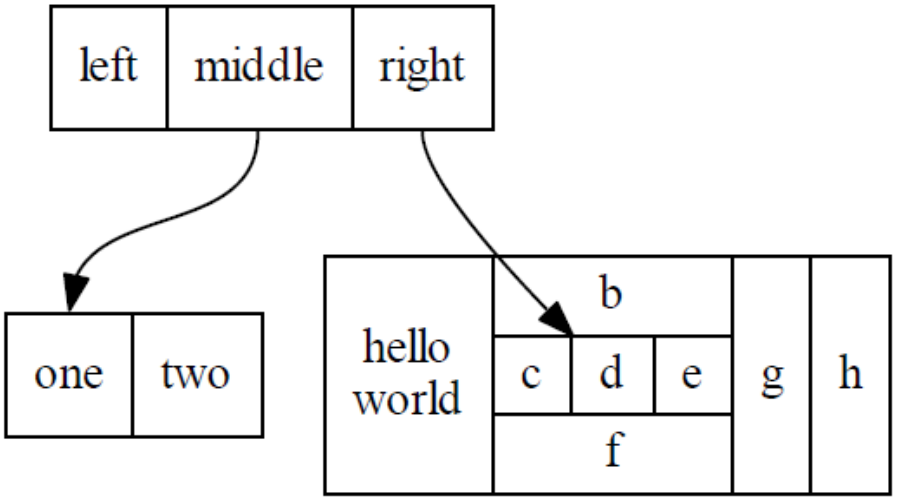


Graphviz – Code Example 5 [3/3]

structs_revisted.gv 생성

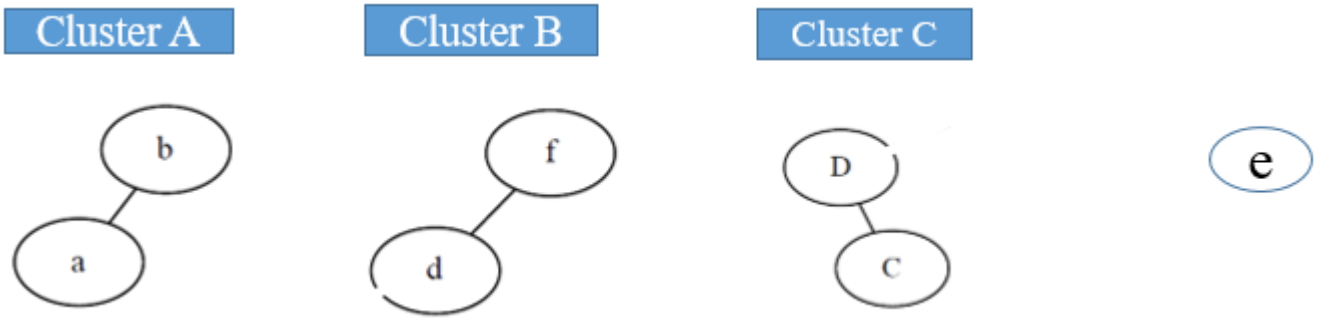
```
digraph structs {
    node [shape=record]
    struct1 [label="<f0> left|<f1> middle|<f2> right"]
    struct2 [label="<f0> one|<f1> two"]
    struct3 [label="hello\nworld |{ b |{c|<here> d|e}| f}| g | h"]
    struct1:f1 -> struct2:f0
    struct1:f2 -> struct3:here
}
```

structs_revisted.gv.pdf 생성



Graphviz – Code Example 6 [1/4]

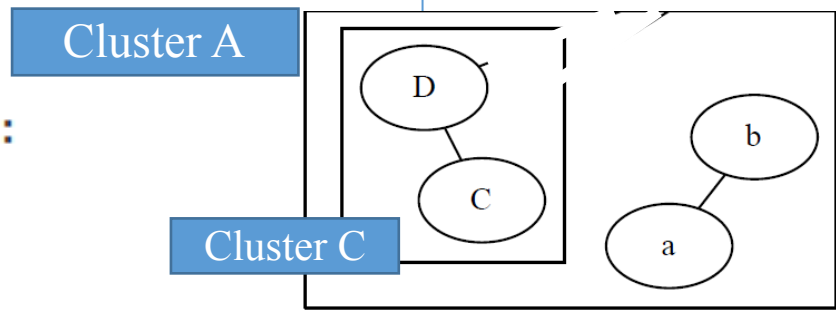
- 아래의 내용을 반영한 Graph 를 그리려고 한다
- Cluster는 총 3개와 single node 1개가 아래 그림과 같다
- Cluster A가 Cluster C를 포함한다
- Cluster C 에서 Cluster B로 edge가 있다
- Node e에서 Cluster B로 edge가 있다
- Node d 에서 Node D로 edge가 있다



Graphviz – Code Example 6 [2/4]

GraphViz Code

```
from graphviz import Graph

g = Graph('G', filename='gviz/fdpclust.gv', engine='fdp')

g.node('e')

with g.subgraph(name='clusterA') as a:
    a.edge('a', 'b')
    with a.subgraph(name='clusterC') as c:
        c.edge('C', 'D')

with g.subgraph(name='clusterB') as b:
    b.edge('d', 'f')

g.edge('d', 'D')
g.edge('e', 'clusterB')
g.edge('clusterC', 'clusterB')

g.view()
```



The subgraph name needs to begin with 'cluster' (all lowercase).
So that Graphviz recognizes it as a special cluster subgraph

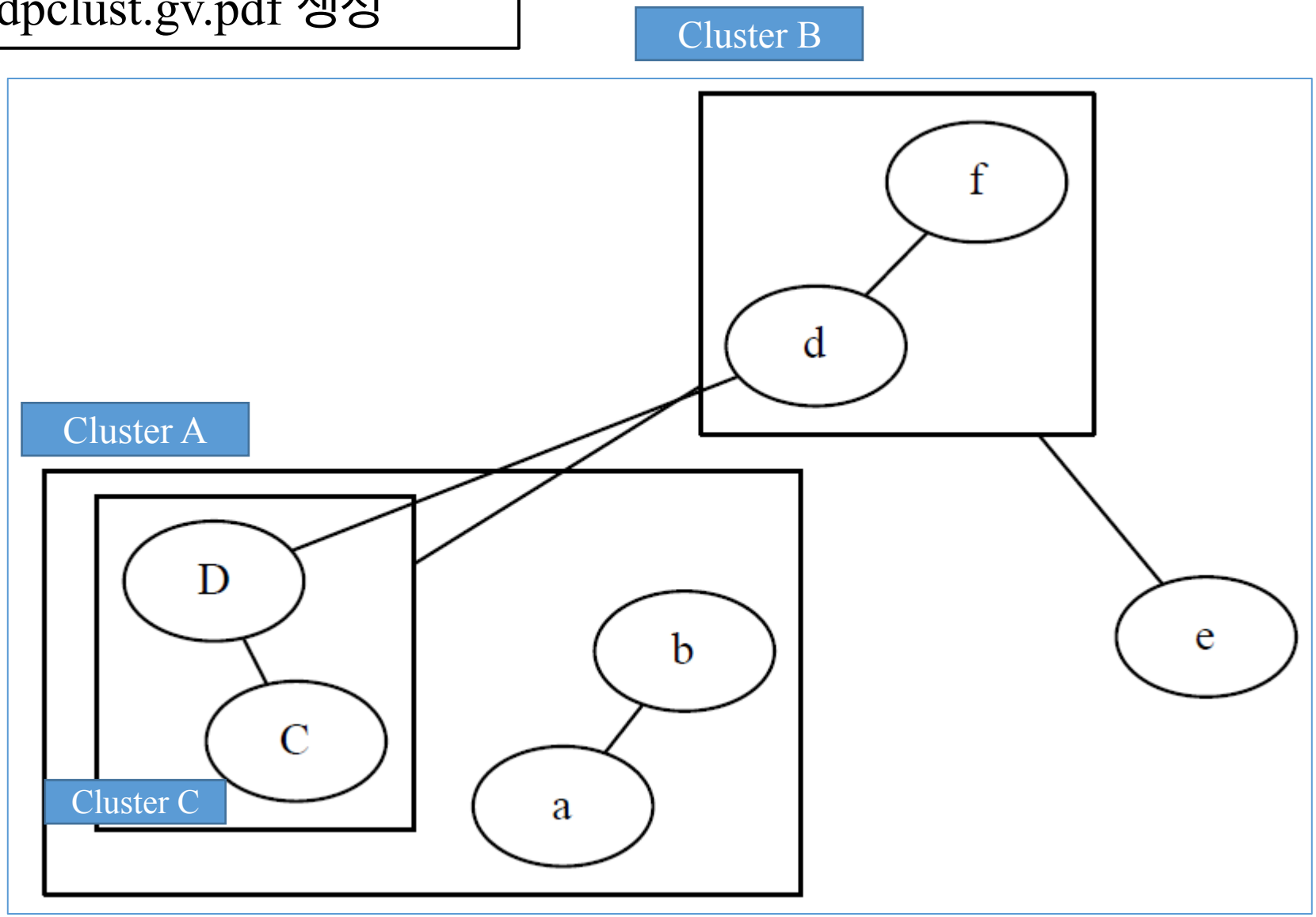
Graphviz – Code Example 6 [3/4]

fdpclust.gv 생성

```
graph G {
    e
    subgraph clusterA {
        a -- b
        subgraph clusterC {
            C -- D
        }
    }
    subgraph clusterB {
        d -- f
    }
    d -- D
    e -- clusterB
    clusterC -- clusterB
}
```

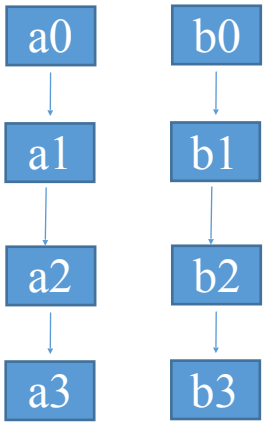
Graphviz – Code Example 6 [4/4]

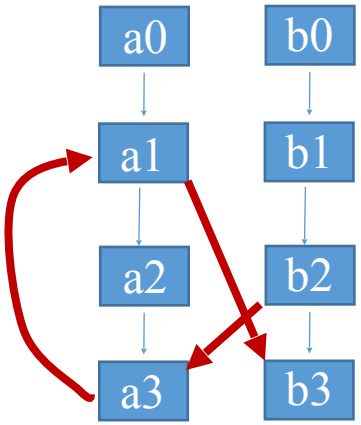
fdpclust.gv.pdf 생성



Graphviz – Code Example 7 [1/4]

- 아래의 내용을 반영한 Graph 를 그리려고 한다
- Process 1 & Process 2가 병렬로 진행된다
  - Process 1은 a0, a1, a2, a3의 step으로 구성
  - Process 2는 b0, b1, b2, b3의 step으로 구성
- Start node는 diamond로 그린다
- End node는 rectangle로 그린다
- Process1의 a1 step에서 process 2의 b3 step으로 directed edge가 있다
- Process1의 a3 step에서 process 1의 a0 step으로 directed edge가 있다
- Process2의 b2 step에서 process 1의 a3 step으로 directed edge가 있다



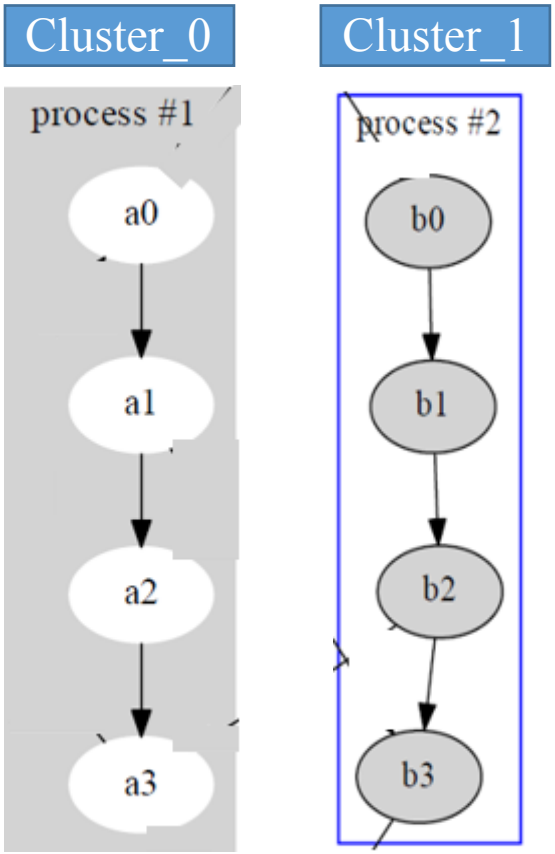


Graphviz – Code Example 7 [2/4]

```python
from graphviz import Digraph
g = Digraph('G', filename='gviz/cluster.gv')

with g.subgraph(name='cluster_0') as c:
    c.attr(style='filled')
    c.attr(color='lightgrey')
    c.node_attr.update(style='filled', color='white')
    c.edges([('a0', 'a1'), ('a1', 'a2'), ('a2', 'a3')])
    c.attr(label='process #1')

with g.subgraph(name='cluster_1') as c:
    c.node_attr.update(style='filled')
    c.edges([('b0', 'b1'), ('b1', 'b2'), ('b2', 'b3')])
    c.attr(label='process #2')
    c.attr(color='blue')

g.edge('start', 'a0')
g.edge('start', 'b0')
g.edge('a1', 'b3')
g.edge('b2', 'a3')
g.edge('a3', 'a0')
g.edge('a3', 'end')
g.edge('b3', 'end')

g.node('start', shape='Mdiamond')
g.node('end', shape='Msquare')

g.view()
```

GraphViz Code

Default engine (dot)



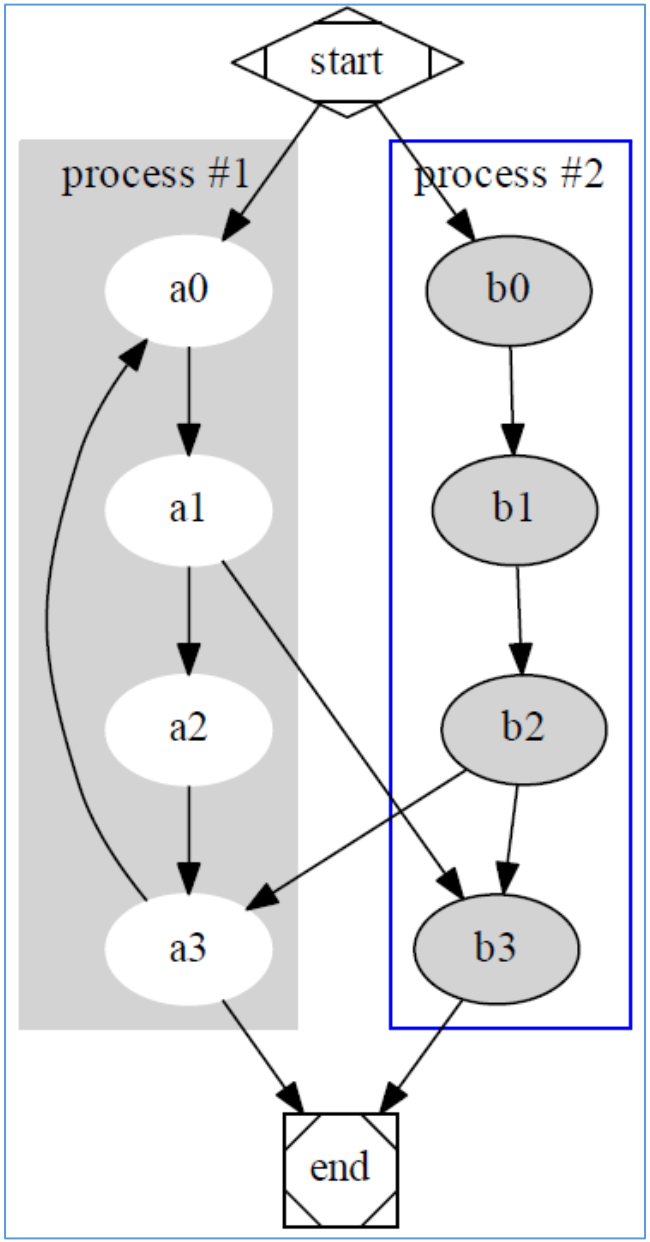
Graphviz – Code Example 7 [3/4]

cluster.gv 생성

```
digraph G {
    subgraph cluster_0 {
        node [color=white style=filled]
        style=filled
        color=lightgrey
        a0 -> a1
        a1 -> a2
        a2 -> a3
        label="process #1"
    }
    subgraph cluster_1 {
        node [style=filled]
        b0 -> b1
        b1 -> b2
        b2 -> b3
        label="process #2"
        color=blue
    }
    start -> a0
    start -> b0
    a1 -> b3
    b2 -> a3
    a3 -> a0
    a3 -> end
    b3 -> end
    start [shape=Mdiamond]
    end [shape=Msquare]
}
```

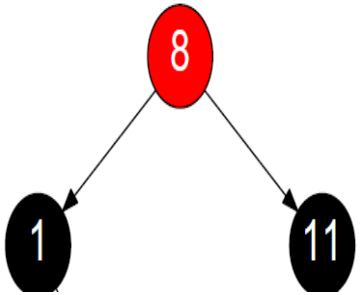
Graphviz – Code Example 7 [4/4]

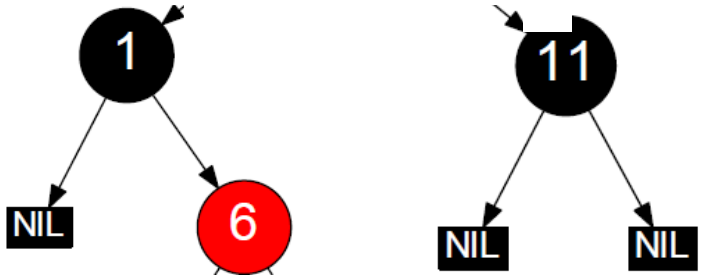
cluster.gv.pdf 생성



Graphviz Code Example 8 : Red Black Tree [1/4]

- 아래의 내용을 반영한 Binary Tree 를 그리려고 한다
- 모든 Circle Node의 구성: (Red or Black의 color)
  - Lst : pointer to left subtree
  - Label
  - Rst: pointer to right subtree
- Root Node 는 Label 8
- Node 8 (Red): lst ➔ Node 1, rst ➔ Node 11
- Node 1 (Black): lst ➔ NIL, rst ➔ Node 6
- Node 11 (Black): lst ➔ NIL, rst ➔ NIL
- Node 6 (Red): lst ➔ NIL, rst ➔ NIL
- Node NIL (Black) 은 작은 rectangle





Graphviz Code Example 8 : Red Black Tree [2/4]

Code

```
from graphviz import Digraph

g = Digraph(comment='Red-Black Tree', engine='dot')
g.attr('graph', ratio='0.8')

g.attr('node', style='filled', fillcolor='red', shape='circle', width='.6',
        fontname='Helvetica', fontweight='bold', fontcolor='white', fontsize='24', fixedsize='true')
g.node('8');

g.attr('node', style='filled', fillcolor='black', shape='circle', width='.6',
        fontname='Helvetica', fontweight='bold', fontcolor='white', fontsize='24', fixedsize='true')
g.node('1'); g.node('11');

g.attr('node', fillcolor='black', shape='record', label="NIL", width='0.4', height='.25', fontsize='16')
g.node('n1'); g.node('n2'); g.node('n3'); g.node('n4'); g.node('n5');

g.attr('node', style='filled', fillcolor='red', shape='circle', label=r"\N", width='.6',
        fontname='Helvetica', fontweight='bold', fontcolor='white', fontsize='24', fixedsize='true')
g.node('6')

g.edge('8', '1');  g.edge('8', '11')
g.edge('1', 'n1', weight='2'); g.edge('1', '6')
g.edge('6', 'n2'); g.edge('6', 'n3')
g.edge('11', 'n4', weight='2'); g.edge('11', 'n5', weight='3')

g.render('gviz/Red-Black Tree', view=True)
```

Graphviz Code Example 8 : Red Black Tree [3/4]

gv 파일

```
// Red-Black Tree
digraph {
    graph [ratio=0.8]
    node [fillcolor=red fixedsize=true fontcolor=white fontname=Helvetica fontsize=24 fontweight=bold shape=circle style=filled width=.6]
    8
    node [fillcolor=black fixedsize=true fontcolor=white fontname=Helvetica fontsize=24 fontweight=bold shape=circle style=filled width=.6]
    1
    11
    node [fillcolor=black fontsize=16 height=.25 label=NIL shape=record width=0.4]
    n1
    n2
    n3
    n4
    n5
    node [fillcolor=red fixedsize=true fontcolor=white fontname=Helvetica fontsize=24 fontweight=bold label="\N" shape=circle style=filled width=.6]
    6
    8 -> 1
    8 -> 11
    1 -> n1 [weight=2]
    1 -> 6
    6 -> n2
    6 -> n3
    11 -> n4 [weight=2]
    11 -> n5 [weight=3]
}
```

Graphviz Code Example 8 : Red Black Tree [4/4]

그림 파일