# Control Flow Graph and Algorithm

Class A; Team No.9
(문윤주, 이인혁, 곽성훈)

## ▷Statement of Purpose

### ◆ Control Flow Graph

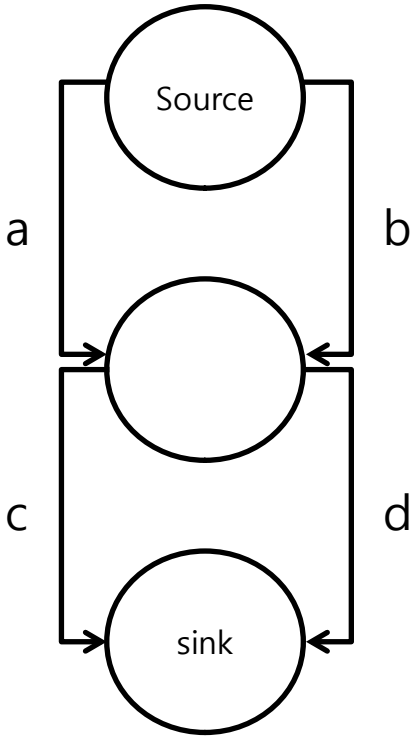
☆ Control Flow Graph는 컴포넌트나 시스템을 통해 실행 될 때 이벤트의 흐름을 표현하는 그림으로서 컴파일러 최적화 정적 분석 도구를 제공합니다.

☆ Control Flow Graph의 특징은 Dominator, Depth, Edges, Loop로 구성되어 있습니다.

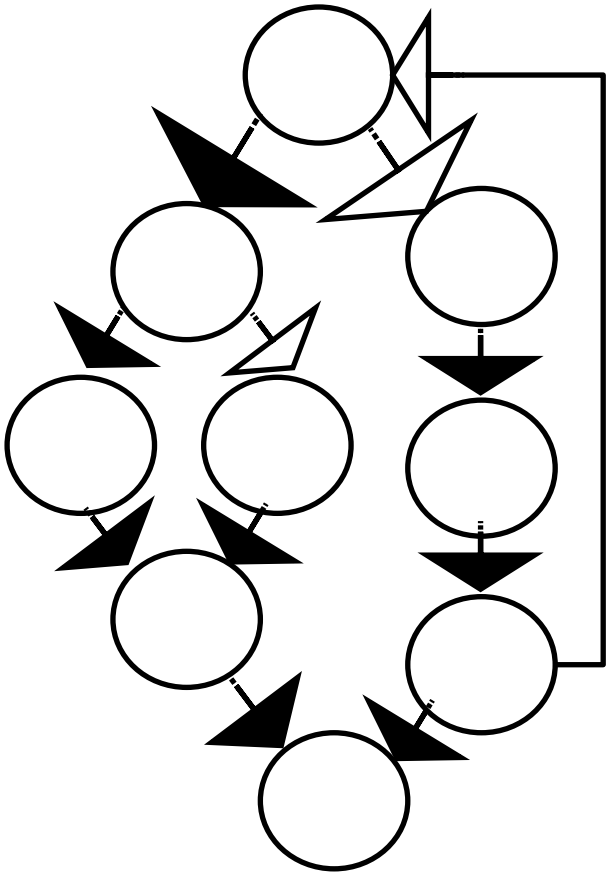
☆ Control Flow Graph에 사용되는 Algorithm에는 Lengauer-Trojan Algorithm, Depth First Order Algorithm 등 각각의 구성에 대한 Algorithm이 여러 가지가 있으며 이 발표에선 Find Dominator Algorithm에 대해 알아봅니다.

# Control Flow Graph

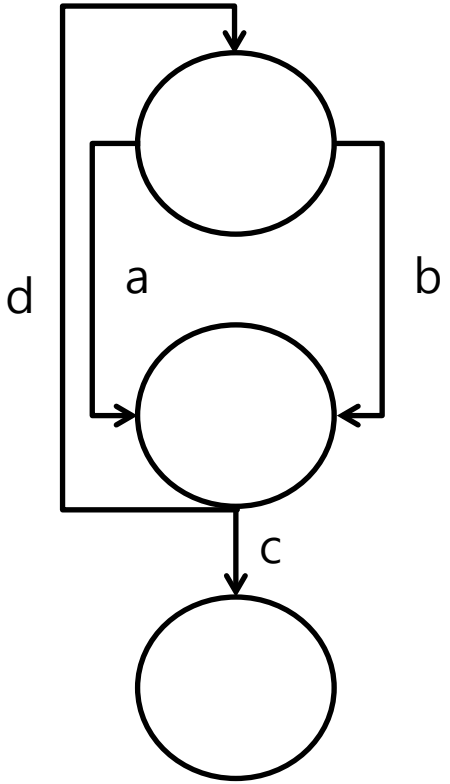
## ▷ Examples




<Simplified CFG>



<Marked edge CFG>

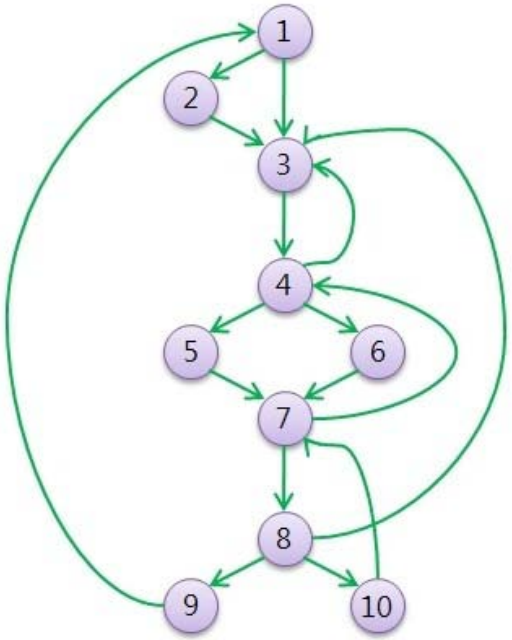



<Looping CFG>

# Control Flow Graph

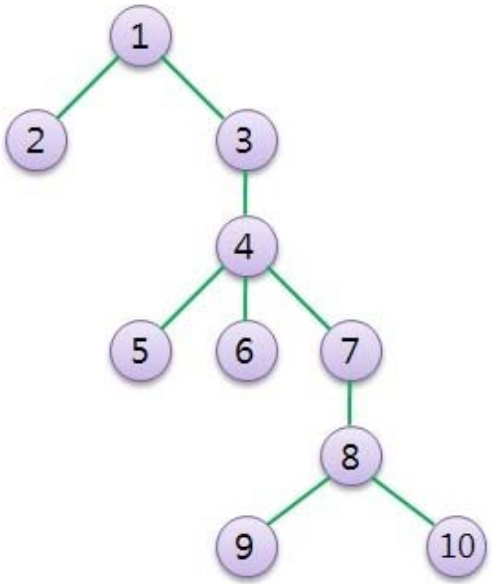
## ▷ Dominator

시작 Node에서 어떤 Node N에 이르는 모든 경로가 Node A를 통과하는 경우, Node A는 Node N을 Dominates(지배한다)라고 하고, "A dom N"이라고 표현합니다. 모든 Node는 자기 자신을 지배하며, 자신의 밑에 직접 닿는 Node만을 지배합니다. =(Root Node는 Tree 전체를 지배한다.)




<그림1>

그림 1과 같은 Tree가 있을 때
지배 관계는 그림 2와 같습니다.

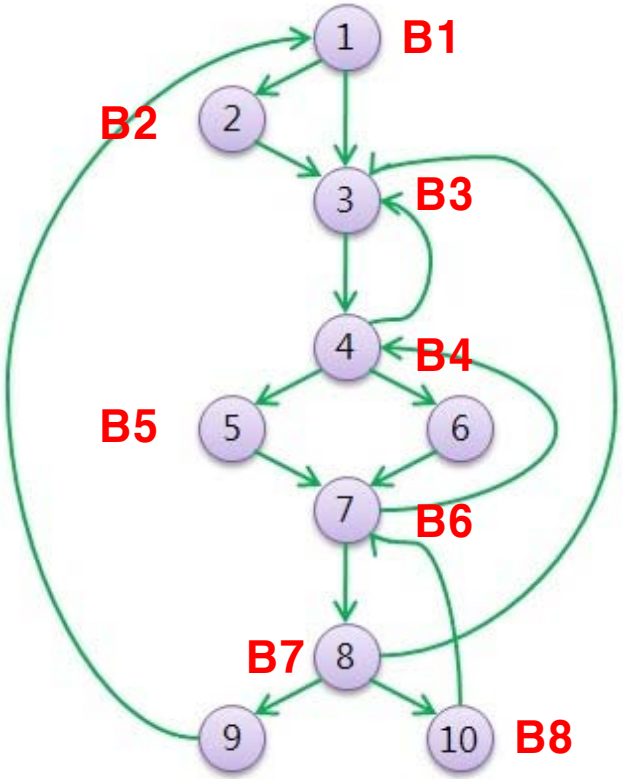



<그림2>

# Control Flow Graph

## ▷Depth

Control Flow Graph에는 Depth(깊이)라는 특징이 있습니다.
이는 Root Node를 깊이 1로 하고 그 밑의 단계 Node를 깊이 2로 하는 방식입니다.
그 표현방법으로는 일반적으로 "B1, B2...Bn" 이라고 표현합니다.

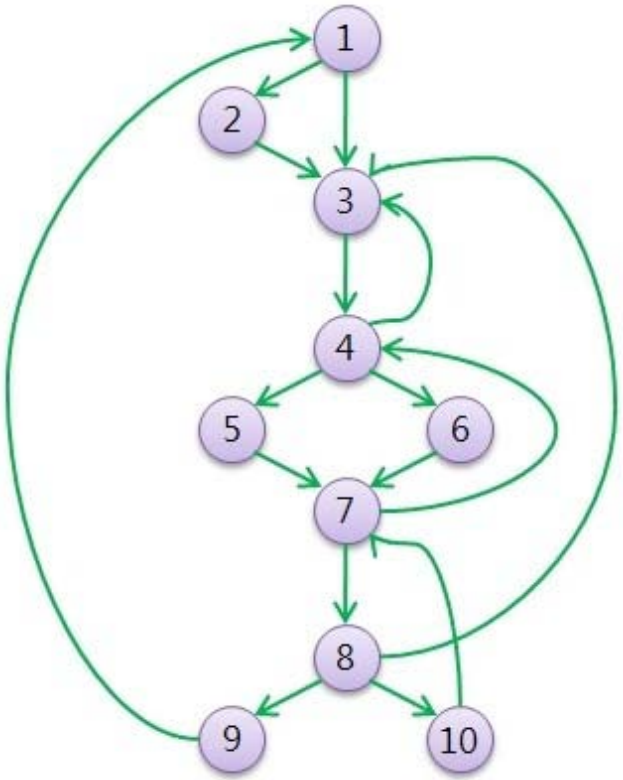



<그림1>

그림 1과 같은 Tree가 있을 때 Depth는 8입니다.

# Control Flow Graph

## ▷Edges




<그림1>

Control Flow Graph에는 3가지의 Edge가 있습니다.

1. 어떤 Node A로부터 Tree상에서 A의 자손이 되는 Node(A 자신은 제외)로 연결되는 Edge를 Advancing Edges(순행 간선)라고 합니다.

2. 어떤 Node A로부터 Tree상에서 A의 조상이 되는 Node(A 자신도 포함)로 연결되는 Edge를 Retreating Edges(역행 간선)라고 합니다. Dominator가 되는 Node로 가는 Edge는 Back Edge(역 간선)라고 합니다.

3. 어떤 A와 B가 서로 부모 자식 관계가 아닌 A→B Edge가 있습니다. 이러한 Edge를 Cross Edges(교차 간선)라 합니다. 새로운 Node를 그릴 때 왼 편에서 오른편으로 그린다고 하면, Cross Edges는 왼편에서 오른편이 아닌, 오른편에서 왼편으로 가게 되는 특징을 가집니다.

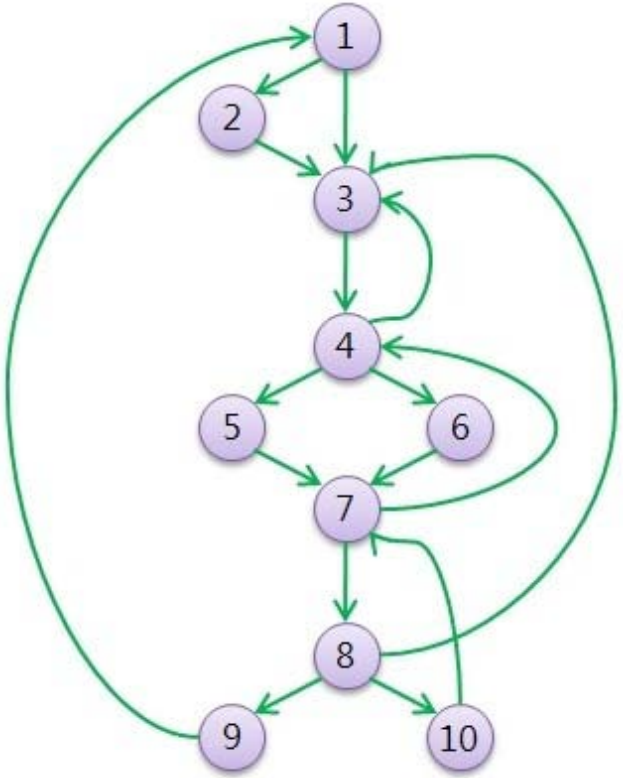
그림 1과 같은 Tree가 있을 때

- Advancing Edges : 모든 Node
- Retreating Edges : 4→3, 7→4, 8→3, 10→7, 9→1
- Cross Edges : 2→3, 5→7

# Control Flow Graph

## ▷Loop

Control Flow Graph의 Loop에는 for Loop, while loop, repeat loop 등이 있으며, Node의 이동에는 go to 명령이 가능하기도 합니다.
모든 Loop에는 Loop Header가 존재하고 Back Edge가 존재합니다.




<그림1>

그림 1과 같은 Tree가 있을 때 Loop는 10, 9, 8, 7, 4에서 발생하며 각각의 Node는 Loop Header가 되고 Back Edge를 존재하게 합니다.

## ▷Statement of Purpose

### ◆ Find Dominator Algorithm

☆ Find Dominator Algorithm은 각각의 Node들이 어떤 Dominator를 가지고 있는 지 알기 위해 개발 되었습니다.

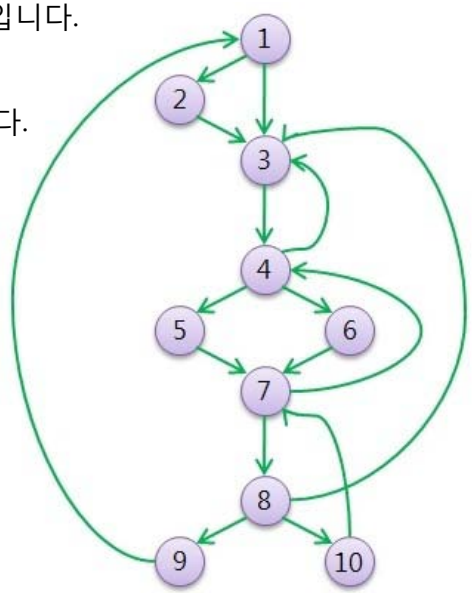
☆ Find Dominator Algorithm은 순방향으로 Tree를 돌고 각각의 Node별로 Dominator를 정의합니다.

☆ Loop는 위에서 아래로 한 번, 아래에서 위로 한 번 시행합니다.

## ▷Algorithm

# Find Dominator Algorithm

Input : Node Set이 N, Edge Set이 E, 시작 Node가 ENTRY인 Flow Graph G
Output : N의 모든 Node n에 대해 n의 Dominator Set D(n)
Method : 아래에 기술된 내역에 따라 정의되는 Data Flow Problem의 Solution을 구합니다.
Basic Block은 Flow Graph의 Node가 되고, N의 모든 Node n에 대해 D(n) = OUT[n]입니다.
Domain : N의 Power Set / Direction : 순방향




for Loop를 통해 작동하고 for Loop가 Flow Graph의 Node를 번호순으로 방문한다고 가정합니다.
D(n)이 OUT[n] Node Set이라고 할 때,
1은 시작 Node이므로 Line(1)에 의해서 D(1)에는 {1}이 저장됩니다.
Node 2의 선행 Node는 1개 뿐이므로 D(2) = {2} ∪ D(1)입니다.
따라서 D(2)는 Set {1, 2}가 됩니다.
다음엔 선행 Node가 1, 2, 4, 8인 Node 3의 차례입니다.
모든 내부 Node는 전체 집합 N으로 초기화되어 있으므로 아래의 식이 성립합니다.

D(3) = {3} ∪ ({1} ∩ {1, 2} ∩ {1, 2, ... , 10} ∩ {1, 2, ... , 10}) = {1, 3}

나머지 값들을 구해서 다시 적어보자면, 아래와 같습니다.
D(1) = {1}
D(2) = {1, 2}
D(3) = {1, 3}
D(4) = {1, 3, 4}
D(5) = {1, 3, 4, 5}
D(6) = {1, 3, 4, 6}
D(7) = {1, 3, 4, 7}
D(8) = {1, 3, 4, 7, 8}
D(9) = {1, 3, 4, 7, 8, 9}
D(10) = {1, 3, 4, 7, 8, 10}

Reverse Loop의 경우에도 값은 변하지 않습니다.
Reverse Loop의 예) 7의 경우를 보면, 5와 6, 그리고 10이 들어갑니다. 5는 {1, 3, 4, 5}, 6은 {1, 3, 4, 6}, 그리고 10은 {1, 3, 4, 7, 8, 10}으로 교집합을 한 값은 {1, 3, 4}가 되고, 자기 자신인 7이 합집합되어 {1, 3, 4, 7}로 유지됩니다. 즉, 2번째 반복으로 값이 불변하므로 이로써 종료됩니다.