Control Flow Graph
(제어 흐름 도표)

200711453 류진렬 200711454 윤병현
200711455 이남섭 200711463 이준하

CLASS B (T5)

Contents

- About CFG(Control Flow Graph)
- About Software Testing
- Kinds of CFG Algorithm
- Statement of Purpose

Definition of CFG(Control Flow Graph)

- 프로그램의 실행과 그 전체의 프로세스가 노테이션으로 표현된 그래프의 형식

- 프로그램의 각각의 세부요소들이 프로그램 내에서 어떻게 사용이 되는지를 분석해야하는 computer science 분야에서 널리 사용되는 기술

- 프로그램 정적/동적 분석(Static/Dynamic Analysis) 방법 중, 가장 많이 쓰이고, 중요한 방법

Essensial components of CFG

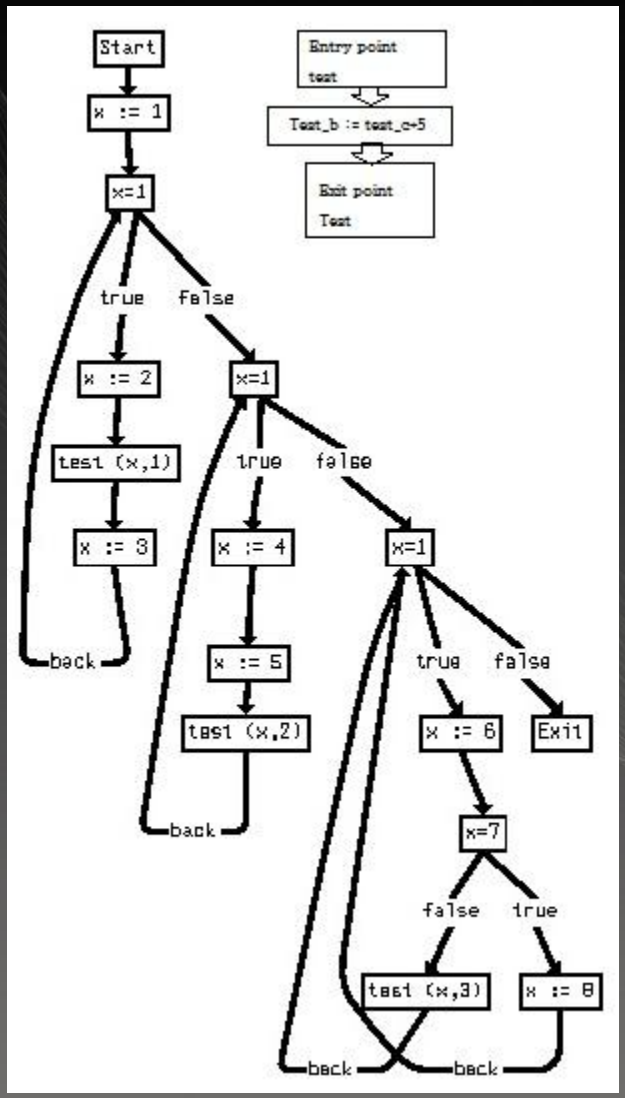
- Basic block： 제어가 시작점으로 들어가서 끝점으로 나올 때까지 정지나 분기의 가능성이 없는 연속적인 문장들의 집합. Program 상에서 일렬로 나타나는 instruction 들의 묶음.
- Node： 프로그램의 각 기본 블록을 노드로 나타낼 수 있다. 각각의 논리적 명령문이라 할 수 있다.
- Edge : 엣지들은 방향성을 가진 직선으로 이 엣지들은 그래프에서 나타나있는 노드(Block) 들이 점핑될때 사용된다.

Definition of CFG(Control Flow Graph)

Pseudo Code

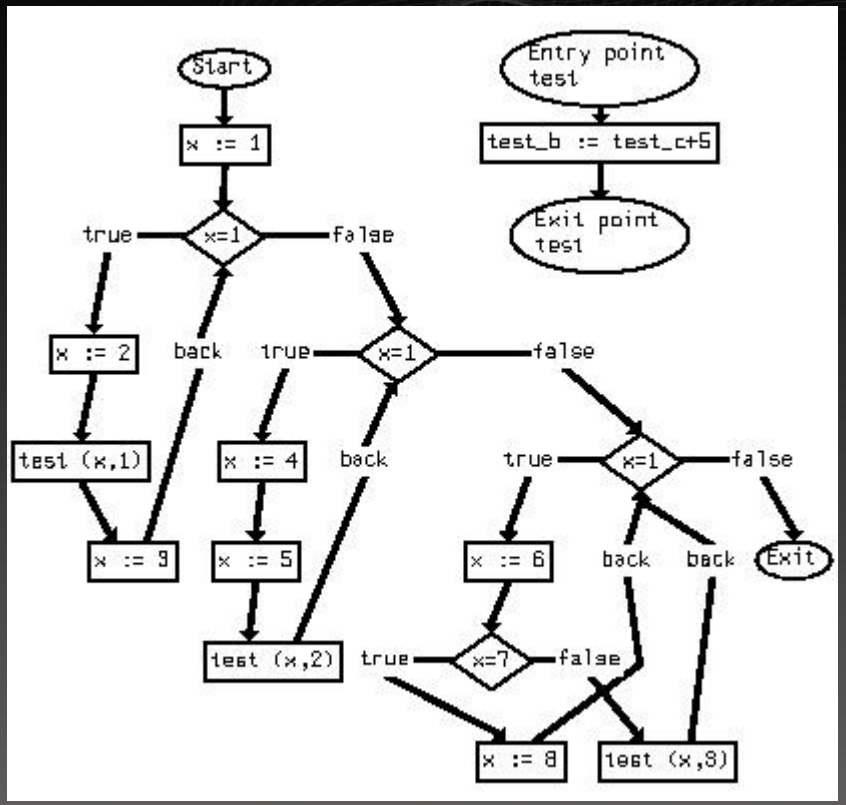
```
PROCEDURE test ( VAR b : INTEGER; c : INTEGER );
BEGIN
        b : = c + 5;
END
BEGIN
        x := 1;
        WHILE ( x = 1) DO
                x := 2;
                test( x, 1 );
                x :=3;
        OD;
        WHILE ( x = 1) DO
                x := 4;
                x := 5;
                test ( x, 2 );
        OD;
        WHILE ( x = 1 ) DO
                x := 6;
                IF ( x = 7 ) THEN x := 8; ELSE test ( x, 3 );
                FI;
        OD;
END
```

<Simple Version>

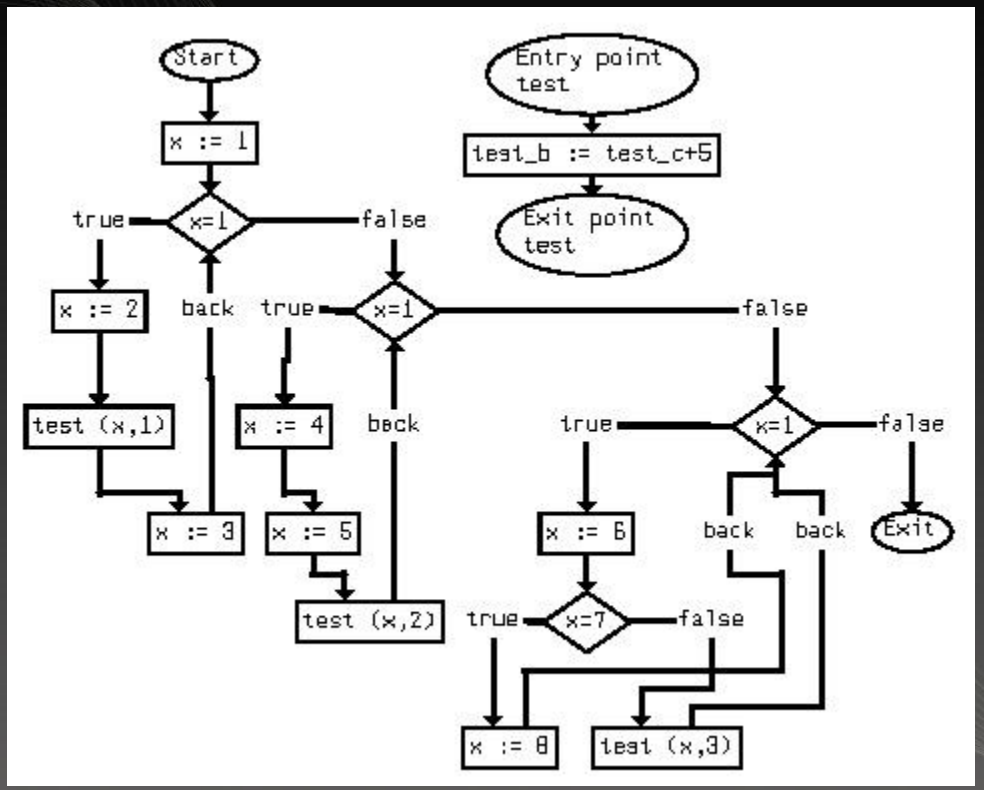


Necessities of CFG

<CFG with Rhomboid and Near Edges>



<CFG with Manhattan Edges>



Necessities of CFG

- 프로그램의 흐름 해석, 파악 용이
- 테스트 및 디버깅 시 사용
- 프로그램의 최적화 및 효율성 향상을 위한 수단
- 테스팅 분야에서 필요로 하는 정보

Definition of Software Testing

- 응용 프로그램 또는 시스템의 동작과 성능, 안정성이 요구하는 수준을 만족하는지를 판단하기 위해 결함을 발견하는 메커니즘.
- 디버깅과는 다르게 오류를 찾아내는 작업.
- **제어흐름 정보(Control Flow Graph)가 정확할수록 정확한 분석 결과를 바탕으로 테스팅 결과를 구할 수 있다.**

Necessities of Testing

- 남아 있는 결함 발견, 예방
- 사용자 및 비즈니스의 요구 충족 확인
- 비즈니스 리스크를 감소시키는 정보에 근거한 조언 제공
- 품질 수준에 대한 자신감 획득과 정보 제공
- 논리적 설계의 구현 검증

Kinds of CFG Algorithms

- Edges recognition Algorithm
- Basic block construction Algorithm
- Positioning Algorithm
  1. BFS Positioning
  2. Loop Positioning
  3. Hierarchical Positioning
- Routing Algorithm
  1. Bezier Routing
  2. Manhattan Routing

Edges recognition Algorithm

- 프로그램을 통해 모든 가능한 경로를 탐색, 하나의 경로를 따라가며 갈라지는 다른경로에 대한 정보들을 쌓아두고 이 정보들을 저장.
- 이 알고리즘은 이미 분석한 지점을 다시 방문하지 않게 하기 위해 프로그램 내의 어떤 state가 어느 지점에서 실행, 분석되었는지를 저장하고 다시 반복하지 않도록 함.
- 이 알고리즘에 대한 결과는 CFG edge들의 집합

Basic block construction Algorithm

- Edge Recognition Algorithm을 통해 생성된 edge들과 연관이 되는 정보를 갖는 Basic block 형성 알고리즘
- 기본적으로 2개의 Basic Block (Start,Exit)을 형성
- 알고리즘을 통해 프로그램의 instruction들을 Block 단위로 나누어 저장
- 새로운 CFG edge들이 생성 될 때마다, 반드시 이 알고리즘에 의해 생성된 CFG Node들과 관련되어져 저장

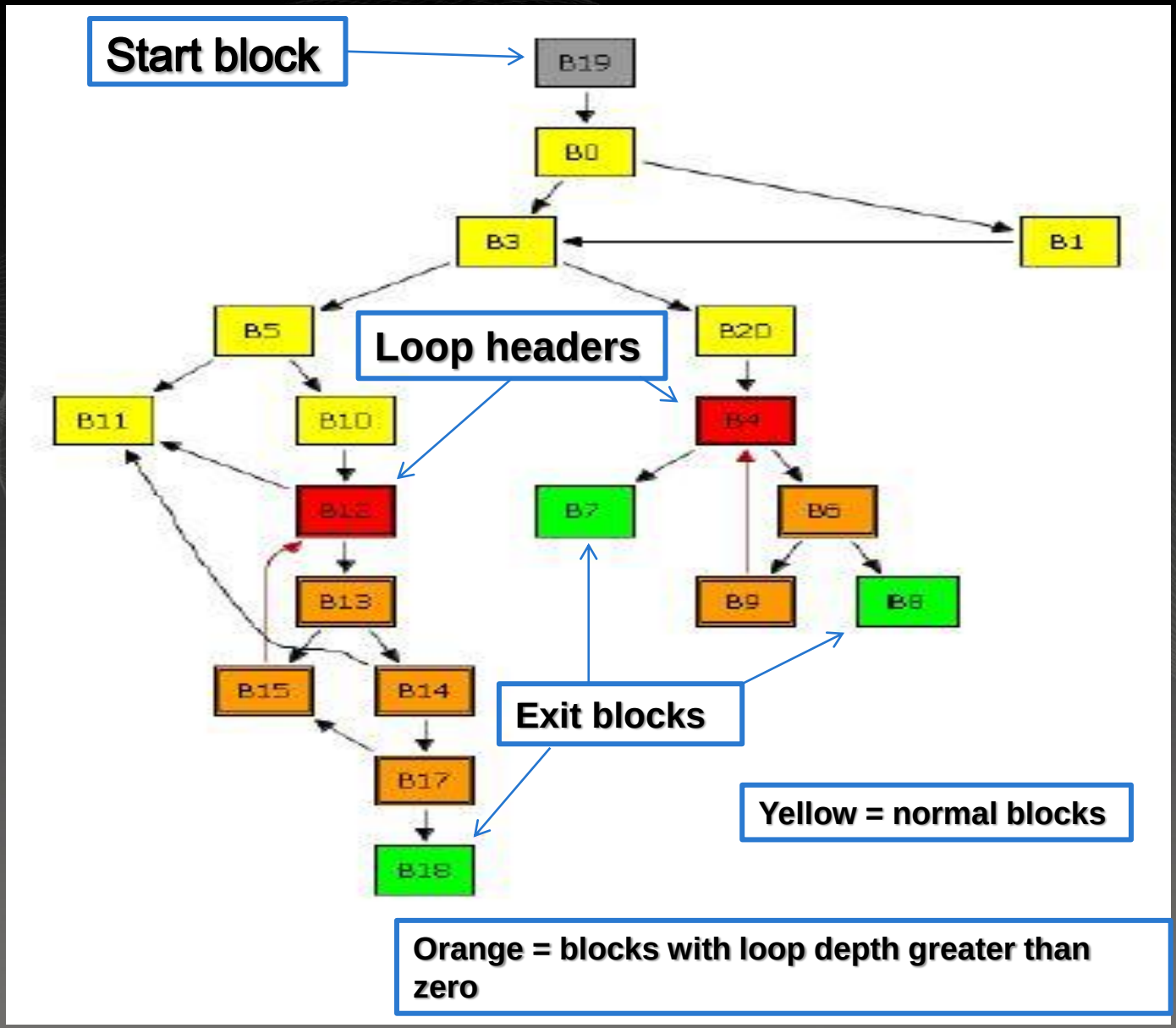
Positioning Algorithm

- **CFG**에 표현될 모든 노드들에 특별한 값(**x** 좌표값, **y** 좌표값)을 부여하는 알고리즘

1. BFS Positioning Algorithm

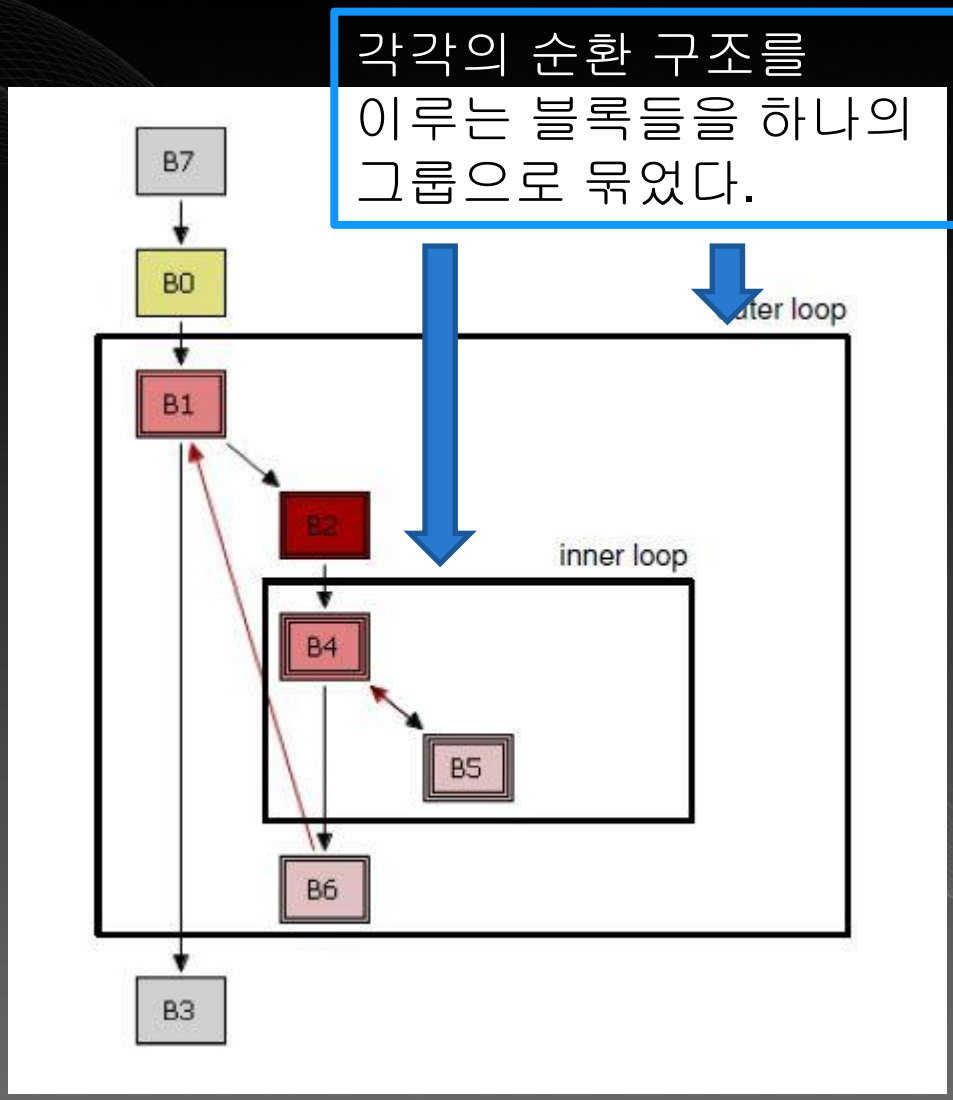
- 각 노드들을 모두 순회하고, 트리의 형태로 **CFG**가 형성될 수 있도록 각각의 노드들에 **x,y**좌표를 부여한다.
- 트리의 형태로 **CFG**를 형성시키기 위해 각 노드들은 레벨 정보를 갖고 있다.
- 이 각 레벨 정보는 그 노드의 루트노드와 자식노드와의 관계로부터 정해진다.
- 루트노드에서부터 시작해서 제일 밑의 자식노드까지의 높이가 **y**좌표를 정하는 데 사용된다.
- 순회할 때 **backedge**를 무시하여 트리형태로 변경 후 순회한다.
- 같은 레벨 값을 가지고 있는 **Node**들의 갯수가 **X**좌표를 정하는 데 사용된다.

Positioning Algorithm



Loop Positioning Algorithm

- Loop를 이루는 블록들을 하나로 묶어 grouping하고, Loop(순환) 구조에 중점을 두어 블록들의 위치정보를 setting 해주는 알고리즘이기 때문에 Loop가 이루어 지는 부분을 표현 함에 있어서 최적화된 알고리즘.
- Grouping된 Loop는 하나의 블록으로 볼수 있다.
- Modified BFS를 이용하여 블록의 y-좌표를 할당한다.



Hierarchical Positioning Algorithm

- -각 단계별 노드들에 대해 층높이를 선택하여 그 층에 맞게 **x, y** 좌표에 위치시켜주는 알고리즘.

1. Break Loop 단계

➢ 이 알고리즘은 순환이 있게 되면 계층구조를 나타내는 데 꼬일 수 있다. 따라서 **backedge**를 제거하여 **loop**부분이 생기지 않도록 해준다.

2. Layer Assigment 단계

➢ 각 노드들에 층을 할당해주는 단계이다. 변형된 **BFS** 방법을 통해 각 노드들의 층을 정하여 그 층에 할당해준다.

3. Adding Dummy Vertices 단계

➢ 어떤 두 노드가 서로 연결되어 있는데 이 두 노드가 **2**층 이상 차이가 날 경우 연결된 두 노드의 층 사이사이마다 더미노드를 추가하여 이 두노드를 연결해주는 단계이다.

4. Crossing Reduction 단계

➢ **edge**들이 서로 겹치지 않게 각 층에 있는 노드들(더미노드포함)의 **x**좌표를 재설정해 주는 단계이다.

Routing Algorithm

- 두 개의 노드 사이에 **edge**를 효율적으로 그리는 방법

1. Initial Routing(첫번째 길찾기)

➢초기 루팅에서는 **B0**와 **B1** 간에 단순한 **straight** 라인을 그린다.

2. Evade Obstacle(장애물 피하기)

➢**Node**들 간 교차점이 많이 발생할 경우, 새로운 **bend** 포인트를 설정하고 교차점을 없애준다.

➢**Node**간 연결된 **Edge**가 다른 노드를 가로지를 경우, 추가적인 중간점을 설정해주고, 그 문제를 해결해야한다.

3. Simplify Polygon (다각형으로의 간략화)

➢연속되지 않은 모든 포인트들을 확인 후 삭제시킨다.

4. Create Curve(곡선으로 만들기)

➢단순화된 다각형의 **Edge**를 바탕으로 **Curve**를 만들어주는 수식을 이용, 부드러운 곡선형으로 만든다.

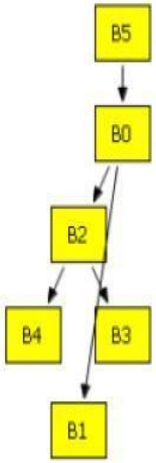


Figure 6.1: Initial routing.

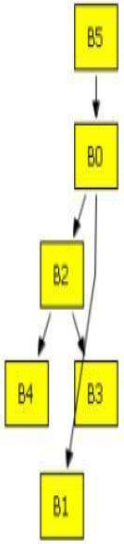


Figure 6.2: After first iteration step.

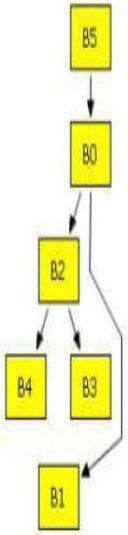


Figure 6.3: After second iteration step.

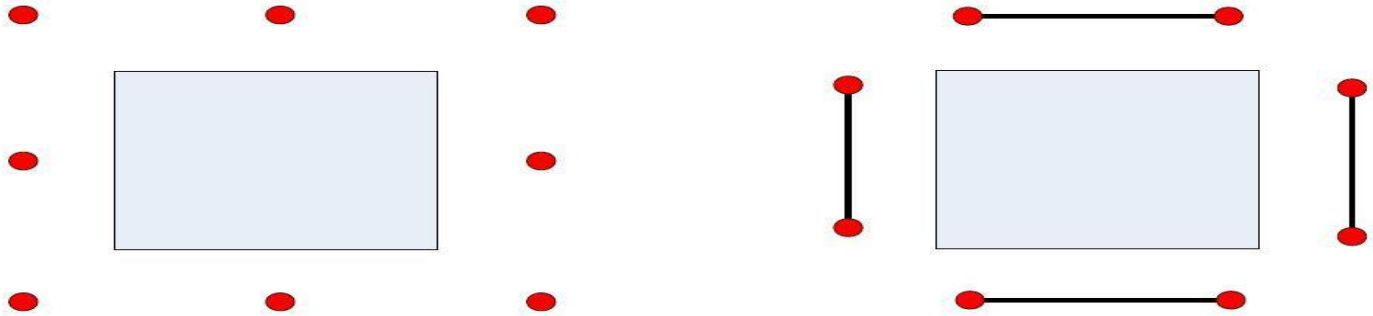


Figure 6.4: All possible evasion points.

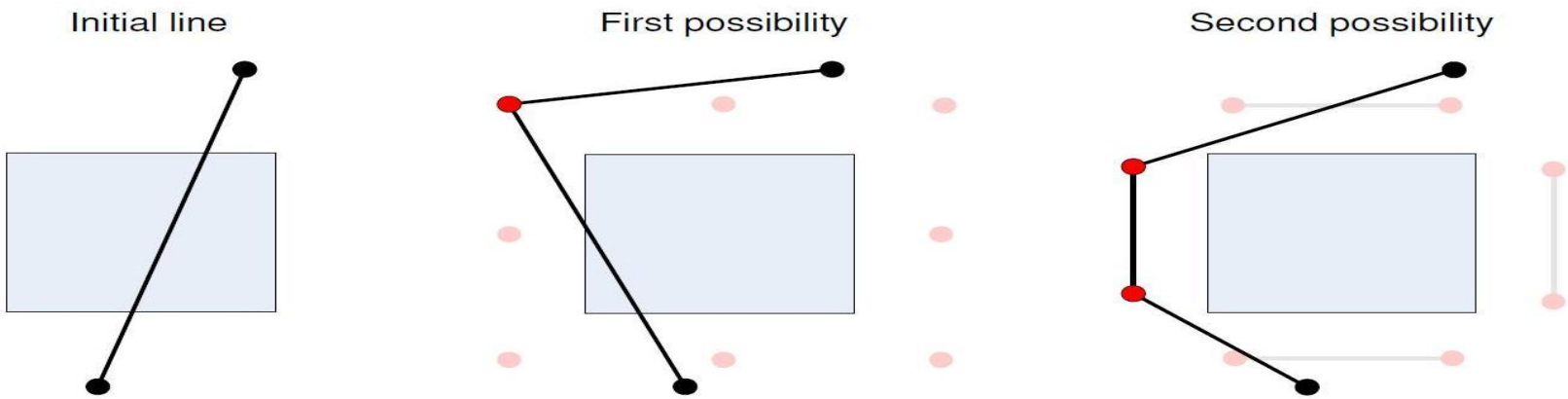


Figure 6.5: Two examples for adding intermediate points.

Statement of Purpose

- 소스코드를 입력받으면 **cfg**를 그려주는 프로그램이다.
- **Edge recognition** 알고리즘과 **Basic Block Construction** 알고리즘을 이용하여 **edge**와 **node**들을 만든다.
- **BFS positioning** 알고리즘을 통해 이차원평면 **x, y**에 노드들을 위치시키고, **Bezier routing** 알고리즘을 사용하여 **edge**들을 나타내어 **cfg**를 표현한다.
- 모든 노드들을 순회한다.
- 순회하는 동안에는 **backedge**에 대해 고려하지 않는다.
- 그려진 **edge**들이 교차되어지고 복잡해지는 것을 **Bezier routing** 알고리즘을 통해서 방지한다.
- 오직 **cfg**를 그리는 기능에 대해서만 고려한다.