알고리즘 과제

- 교차와 사이클이 없는 최대 깊이 무향 그래프 -

담당 교수	서영건 교수님
작성자	도우진
학과	컴퓨터과학과
학번	2015010925
작성일자	2019.10.22.

[목차]

1. 프로젝트 개요

1) 프로젝트 제목

교차와 사이클이 없는 최대 깊이 그래프를 구현한다.

2) 소스코드

Github URL: https://github.com/NeoMindStd/DrawGraph

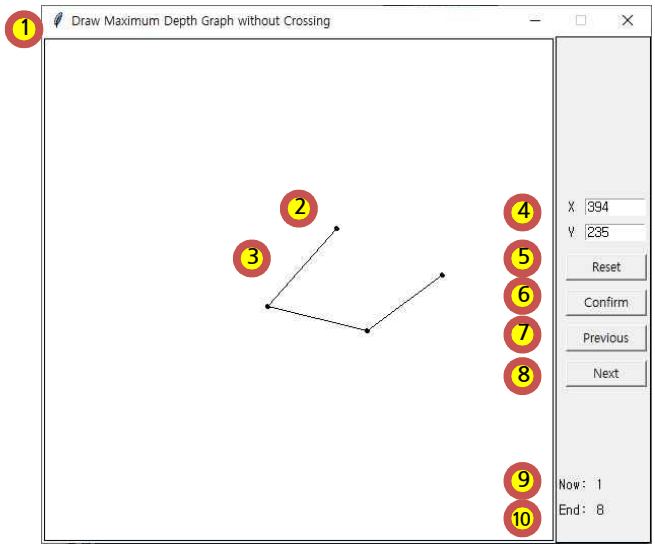
3) 개발환경 및 타겟 플랫폼

Development O/S Target Platform	Windows10
Language	Python 3.6.5
Version Control	Git & Github

4) 개발 의도와 목적

교차와 사이클이 없는 최대 깊이 그래프를 구현한다.
즉, (그래프의 깊이) = (점의 개수)-1 이다.

5) 프로그램 설명 및 조작법



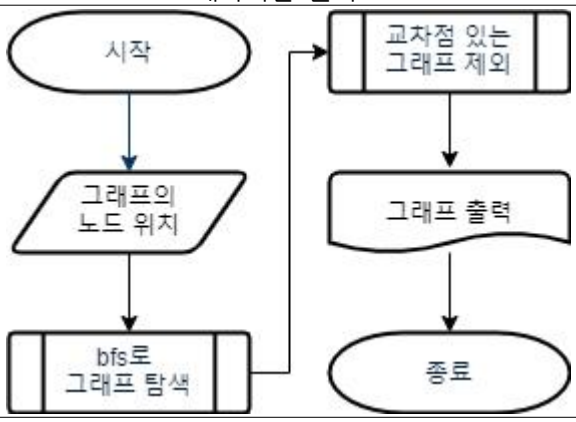
① 프로그램의 제목표시줄이다.
② 캔버스 영역이다. Confirm 전에는
마우스 왼쪽클릭으로 점을 찍을 수 있다.
③ 프로그램이 계산 후 그린 경로이다.
④ 마지막으로 점을 찍은 x좌표와 y좌표이다.
⑤ 초기화버튼이다. 캔버스, 계산 데이터를 초기화한다.
⑥ 찍은 점을 토대로 계산 후 첫 그래프를 보여준다.
⑦ 이전 그래프를 보는 버튼이다.
⑧ 다음 그래프를 보는 버튼이다.
⑨ 현재 그래프가 몇 번째인지 나타낸다.
⑩ 총 몇 개의 그래프가 있는지 나타낸다.

2. 내부 구현

1) 알고리즘

(1) 알고리즘(함수) 구조

대략적인 순서도



함수 설명

```
def showFrame():
    root = Tk()
    root.title("Draw Maximum Depth Graph without Crossing")
    root.geometry("600x500")
    root.resizable(False, False)

    # 이벤트 리스너
    def onClick(pos):
        global ovalParams, showMode
        if showMode:
            return
        ovalParams.append([pos.x-2, pos.y-2, pos.x+2, pos.y+2, "black"])
        canvas.create_oval(ovalParams[-1][0], ovalParams[-1][1], ovalParams[-1][2], ovalParams[-1][3], fill=ovalParams[-1][4])
        textPosX.delete(1.0, "end-1c")
        textPosX.insert(INSERT, f"{pos.x}")
        textPosY.delete(1.0, "end-1c")
        textPosY.insert(INSERT, f"{pos.y}")
        print("position: ", pos.x, pos.y)
        print(textPosX.index(END))
        pts.append((pos.x, pos.y))


    def onResetTap():
        global pts, result, idx, ovalParams, showMode
        pts, result, idx, ovalParams, showMode = [], [], -1, [], False
        textPosX.delete(1.0, "end-1c")
        textPosY.delete(1.0, "end-1c")
        textCurrentIdx.delete(1.0, "end-1c")
        textEndIdx.delete(1.0, "end-1c")
        canvas.delete("all")

    def onConfirmTap():
        global result, idx, showMode
        try:
            if idx == -1 and not result:
                result, showMode = bfsGraph(pts), True
                print("필터링 전: %d개" %len(result))
                result = list(filter(lambda x: not isCrossGraph(x), result))
                print("필터링 후: %d개" %len(result))
                idx+=1
                canvas.create_line(result[idx])
                textCurrentIdx.insert(INSERT, idx+1)
                textEndIdx.insert(INSERT, len(result))
                print(*result[idx])
            else:
```

```
                pass
        except:
            print("***점을 2개 이상 찍어주세요***")
            btnReset.invoke()

    def onPreviousTap():
        global idx, ovalParams
        if not result or idx < 1:
            pass
            return
        canvas.delete("all")
        for ovalParam in ovalParams:
            canvas.create_oval(ovalParam[0], ovalParam[1], ovalParam[2], ovalParam[3], fill=ovalParam[4])
        idx-=1
        canvas.create_line(result[idx])
        textCurrentIdx.delete(1.0, "end-1c")
        textCurrentIdx.insert(INSERT, idx+1)
        print(*result[idx])

    def onNextTap():
        global idx
        if not result:
            btnConfirm.invoke()
            return
        if idx < len(result)-1:
            canvas.delete("all")
            for ovalParam in ovalParams:
                canvas.create_oval(ovalParam[0], ovalParam[1], ovalParam[2], ovalParam[3], fill=ovalParam[4])
            idx+=1
            canvas.create_line(result[idx])
            textCurrentIdx.delete(1.0, "end-1c")
            textCurrentIdx.insert(INSERT, idx+1)
            print(*result[idx])
        else:
            pass

    # 왼쪽 레이아웃
    canvas = Canvas(root, width=500, height=500,
                            bg="white", relief="solid", bd=1)
    canvas.bind("<Button-1>", onClick)
    canvas.pack(side="left")

    # 오른쪽 레이아웃
    frameButtonBox = Frame(root, width=100, height=500, relief="solid", bd=1)
    frameButtonBox.pack(side="right")

    # X
    textX=Text(frameButtonBox, bg="#f0f0f0", width=1, height=1, relief="flat")
    textX.insert(INSERT,"X")
    textX.place(relx=0.1, rely=0.32)

    # x좌표
    textPosX = Text(frameButtonBox, width=8, height=1)
    textPosX.place(relx=0.3, rely=0.32)

    # Y
    textY=Text(frameButtonBox, bg="#f0f0f0", width=1, height=1, relief="flat")
    textY.insert(INSERT,"Y")
    textY.place(relx=0.1, rely=0.37)

    # y좌표
    textPosY = Text(frameButtonBox, width=8, height=1)
    textPosY.place(relx=0.3, rely=0.37)

    # Reset 버튼
    btnReset = Button(frameButtonBox, text="Reset", command=onResetTap,
                            overrelief="solid", width=10)
    btnReset.place(relx=0.1, rely=0.43)

    # Confirm 버튼
    btnConfirm = Button(frameButtonBox, text="Confirm", command=onConfirmTap,
                            overrelief="solid", width=10)
    btnConfirm.place(relx=0.1, rely=0.50)
```

```
    # Previous 버튼
    btnPrevious = Button(frameButtonBox, text="Previous", command=onPreviousTap,
                         overrelief="solid", width=10)
    btnPrevious.place(relx=0.1, rely=0.57)

    # Next 버튼
    btnNext = Button(frameButtonBox, text="Next", command=onNextTap,
                         overrelief="solid", width=10)
    btnNext.place(relx=0.1, rely=0.64)

    # Now
    textNow=Text(frameButtonBox, bg="#f0f0f0", width=4, height=1, relief="flat")
    textNow.insert(INSERT,"Now:")
    textNow.place(relx=0.0, rely=0.87)

    # current idx
    textCurrentIdx=Text(frameButtonBox, bg="#f0f0f0", width=6, height=1, relief="flat")
    textCurrentIdx.place(relx=0.4, rely=0.87)

    # End
    textEnd=Text(frameButtonBox, bg="#f0f0f0", width=4, height=1, relief="flat")
    textEnd.insert(INSERT,"End:")
    textEnd.place(relx=0.0, rely=0.92)

    # end idx
    textEndIdx=Text(frameButtonBox, bg="#f0f0f0", width=6, height=1, relief="flat")
    textEndIdx.place(relx=0.4, rely=0.92)

    root.mainloop()
```

showFrame() 함수는 파이썬 내장 GUI프로그래밍 api를 이용하여 사용자가 정점을 찍고 프로그램이 경로를 그려 그래프를 완성할 캔버스, 현재 찍은 정점의 위치를 나타내는 텍스트박스, 사용자와 다양한 상호작용을 할 Reset, Confirm, Previous, Next버튼, 현재 그래프가 몇 번째인지 나타내주고, 총 그래프의 개수가 몇 개인지 나타내주는 텍스트박스를 화면에 표시한다.

showFrame()함수 안의 onClick(pos) 함수는 마우스 좌클릭 이벤트 콜백 메소드로서, 사용자가 찍은 좌표를 바탕으로 캔버스에 점을 그리고 텍스트박스에 좌표를 표시하는 역할을 한다.
@param tkinter.Event pos 마우스 좌클릭 이벤트를 받아 클릭 위치를 포함하는 변수

showFrame()함수 안의 onResetTap() 함수는 사용자가 프로그램을 재실행하지 않아도 새로운 점을 찍어 새로운 그래프를 볼 수 있게 관련된 데이터를 리셋해주는 함수다.

showFrame()함수 안의 onConfirmTap() 함수는 사용자가 찍은 정점을 바탕으로 경로의 개수가 정점의 개수 -1 개인 모든 그래프를 받아오고, 해당 그래프를 교차하지 않는 그래프만 남도록 필터링한 후 캔버스에 첫번째 그래프를 그리는 함수다.

showFrame()함수 안의 onPreviousTap() 함수는 이전 그래프를 캔버스에 다시 그리는 함수다.

showFrame()함수 안의 onNextTap() 함수는 다음 그래프를 캔버스에 그리는 함수다.

```
def bfsGraph(points):
    n = len(points)
    result = []
    queue = deque([])
    enQ = queue.append
    deQ = queue.popleft
    for point in points:
        queue.append([point])
        while queue:
            graph = deQ()
            if len(graph) < n:
                for point in points:
                    if point not in graph:
                        enQ(graph+[point])
            else:
```

```
                new = list(reversed(graph))
                flag = True
                for i in range(len(result)):
                    cnt = 0
                    for j in range(n):
                        if result[i][j] != new[j]:
                            break
                        cnt += 1
                    if cnt == n:
                        flag = False
                        break
                if flag:
                    result.append(graph)

    return result
```

점의 리스트를 파라미터로 넘겨받아 해당 점을 잇는 경로의 개수가 정점의 개수 -1개인 모든 그래프를 리턴하는 함수다. 모든 그래프를 구할 때는 큐를 이용한 BFS를 사용했으며, 점이 1, 2, 3, 4가 있다고 할 때, 1-2-3-4와 4-3-2-1은 방향을 고려하지 않기 때문에 같은 그래프이므로 중복으로 판단하여 이미 같은 그래프가 결과에 있을 경우, 새로운 그래프는 결과에 포함시키지 않는다.
@param list points 그릴 그래프의 정점 좌표정보를 담은 tuple로 구성된 list.
@return list 탐색한 그래프의 리스트. 그래프는 튜플로 표현된 점을 경로 순서대로 나열한 형태이다.

```
def ccw(pos1, pos2, pos3):
    x1, x2, x3 = pos1[0], pos2[0], pos3[0]
    y1, y2, y3 = pos1[1], pos2[1], pos3[1]
    tmp = x1*y2+x2*y3+x3*y1 - y1*x2-y2*x3-y3*x1
    if tmp > 0:
        return 1
    elif tmp < 0:
        return -1
    else:
        return 0
```

세 점의 위치를 나타내는 튜플 세 개를 파라미터로 받아 성분이 주어진 벡터의 외적을 이용하여 두 점이 이루는 직선에 대해 한 점의 위치가 어디인지 판별해주는 함수다.
@param tuple pos1 위치정보를 판별하는 기준 선분의 한 점의 좌표를 담은 tuple
@param tuple pos2 위치정보를 판별하는 기준 선분의 다른 한 점의 좌표를 담은 tuple
@param tuple pos3 위치정보를 판별할 점의 좌표를 담은 tuple
@return int 선분에 대해 점이 ccw방향이면 1, 선분의 직선상의 점이면 0, cw방향이면 -1

```
def isCross(a, b, c, d):
    abc, abd, cda, cdb = ccw(a, b, c), ccw(a, b, d), ccw(c, d, a), ccw(c, d, b)
    ab, cd = abc*abd, cda*cdb
    if ab == 0 and cd == 0 and (sum([abc**2, abd**2, cda**2, cdb**2])==0  or (a!=c and a!=d and b!=c and
b!=d)):
        if a > b:
            a, b = b, a
        if c > d:
            c, d = d, c
        return c<b and a<d
    return (ab<=0 and cd<=0) and not (a==c or a==d or b==c or b==d)
```

교차여부를 판별하고 싶은 두 선분의 양 끝점 좌표 4개를 차례대로 파라미터로 넘겨받아 ccw()를 이용하여 선분과 한 점의 위치관계(cw, ccw, 선분에포함)를 받아와 선분의 교차 여부를 판별해주는 함수다.
@param tuple a 교차여부를 검사할 선분 1의 점 1의 좌표정보를 가진 tuple
@param tuple b 교차여부를 검사할 선분 1의 점 2의 좌표정보를 가진 tuple
@param tuple c 교차여부를 검사할 선분 2의 점 1의 좌표정보를 가진 tuple
@param tuple d 교차여부를 검사할 선분 2의 점 2의 좌표정보를 가진 tuple
@return bool 선분이 교차하면 True, 그렇지 않으면 False

```
def isCrossGraph(graph):
    for i in range(len(graph)-1):
        for j in range(len(graph)-1):
            if i != j:
                if isCross(graph[i], graph[i+1], graph[j], graph[j+1]):
                    return True
    return False
```

넘겨받은 그래프를 반복하며 그래프의 선분 양 끝점 좌표를 isCross()를 이용하여 교차여부를 검증하여 하나라도

교차하면 True를 리턴하고 그렇지 않으면 False를 리턴하는 함수다.

@param list graph 교차여부를 판별할 그래프. 튜플로 표현된 점을 경로 순서대로 나열한 형태이다.

@return bool 교차하는 선분이 하나라도 있으면 True, 그렇지 않으면 False

```
def testDriver(function, params):
    if function == bfsGraph:
        result = bfsGraph(params)
        n = len(result)
        idx = [i for i in range(len(result))]
        for i in range(len(result)):
            result[i] = [idx[i], *result[i]]
        print("₩n".join(map(str, result)))
    elif function == isCrossGraph:
        print(f"{params}: {isCrossGraph(params)}")
```

다른 모듈들을 테스트해보는 함수다. 테스트 할 모듈과 파라미터를 넘겨받아 실행 후, 적절한 값을 출력해준다.

@param function function 테스트 할 모듈

@param type params 테스트 할 모듈의 파라미터.

(2) 알고리즘 설명 및 프로그램 증명

bfs로 그래프를 완전 탐색하는 방법, 교차여부를 판별하는 방법에 대해 설명 및 증명한다.

bfs로 그래프를 완전 탐색하는 방법

우선, bfs란 Breadth-First Search, 즉 너비 우선 탐색을 뜻한다. 완전탐색은 출발지에서 목적지까지 이르는 모든 경로를 탐색하는 것이다. 알고리즘은 다음과 같다.

1. 빈 큐에 루트노드로 선정되지 않은 노드 중 루트노드를 선택하여 삽입한다.
2. 큐에서 제거한 그래프의 마지막 정점에서 방문하지 않은 모든 정점에 대해 각각 방문하며 큐에 추가한다.
3. 만약 방문하지 않은 정점이 없다면, 결과를 저장한다.
4. 2~3을 큐가 빌 때까지 반복한다.
5. 모든 노드를 루트노드로 사용할 때 까지 1~4를 반복한다.

이 과정을 거치면 모든 정점에서 출발하는 사이클이 없는 (경로의 개수)=(정점의 개수 - 1)인 그래프가 완전 탐색된다. 다만, 무향 그래프이므로 정점 a에서 출발하여 정점 e에 도착하는 그래프와 정점 e에서 출발하여 정점 a에 도착하는 그래프는 같으므로, 그래프의 순서를 거꾸로 한 결과가 있는 경우는 저장하지 않는다.

그래프의 교차여부를 판별하는 방법

정점이 n개이고 경로가 1, 2, …, n-1까지 있는 그래프의 교차 여부를 판별하는 알고리즘은 다음과 같다.

1. 경로를 하나 i로 선택한다.
2. 다른 또 하나의 경로를 j로 선택한다.
3. 선택된 경로끼리 교차되는지 여부를 검사한다.
4. 만약 교차한다면, 교차하는 그래프로 판정하고 종료한다.
5. 교차하지 않는다면, 다른 모든 경로를 j로 선택할 때 까지 2~4를 반복한다.
6. i로 모든 경로를 선택할 때 까지 1~5를 반복한다.

무향그래프이므로 경로는 곧 선분이다. 선분의 교차 판별은 아래에서 설명한다.

그래프의 교차여부를 판별하는 방법 - 정점을 공유하는 경로(선분)의 교차 판별

함수를 다음과 같이 정의한다.

$$ccw(x, y, z) \begin{cases} -1 & (\text{정점 x-y-z로 향하는 방향이 시계방향인 경우}) \\ 0 & (\text{정점 x-y-z로 향하는 경로가 한 직선상에 있을 때}) \\ 1 & (\text{정점 x-y-z로 향하는 방향이 반시계방향인 경우}) \end{cases}$$

ccw 함수에 대한 설명은 아래에서 한다.

선분 $\overline{ab}$ 와 선분 $\overline{bc}$ 가 있을 때, abc=ccw(a, b, c)와 abd=ccw(a, b, d), cda=ccw(c, d, a)와 cdb=ccw(c, d, b)를 이용하여 검사한다. 우선, 직선의 경우라면 abc와 abd의 곱이 0이하인 경우 직선 $\overline{ab}$ 와 직선 $\overline{bc}$ 는 교차한다.

이에 대한 증명은 다음과 같다. 단, ab = abc*abd, cd = cda*cdb라고 가정한다.

abc와 abd는 각 -1, 0, 1의 값을 갖는다.

1. abc 또는 abd가 0인 경우

직선인 경우, abc가 0이라면 직선 $\overline{ab}$ 위에 정점 c가 위치한다. 마찬가지로, abd가 0이라면 직선 $\overline{ab}$ 위에 정점 d가 위치한다. 그래프로 생각하지 않고 일반적인 직선으로 생각한다면, 직선위의 특정 정점이 다른 직선 위에 오게 되므로 교차라고 할 수 있다. 이 때, ab = 0이다.
∵ abc=0이라면, 0*abd=0. abd=0이라면, abc*0=0

2. abc=abd≠0인 경우

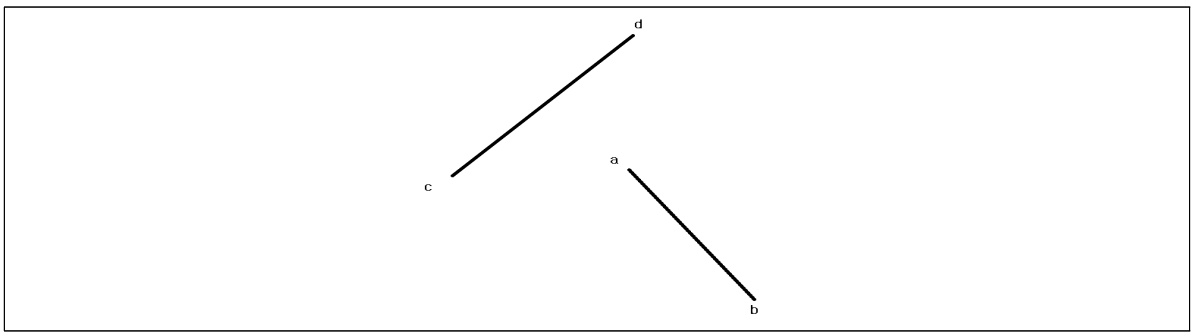
abc가 -1이고 abd가 -1이거나 abc가 1이고 abd가 1이라면, 직선 $\overline{ab}$ 에 대해 한쪽 방향에 두 정점이 모두 위치한다. 이 경우 직선 $\overline{ab}$ 에 대해 선분 $\overline{cd}$ 는 교차하지 않는다. 이 때, ab=1이다.
∵ abc=-1, abd=-1이라면, -1*-1=1, abc=1, abd=1이라면, 1*1=1

3. 0≠abc≠abd≠0인 경우

abc가 -1이고 abd가 1이거나 abc가 1이고 abd가 -1이라면, 직선 $\overline{ab}$ 에 대해 두 정점이 위치한 방향은 반대쪽이다. 이 경우 직선 $\overline{ab}$ 에 대해 선분 $\overline{cd}$ 는 교차하게 된다. 이 때, ab =-1이다.
∵ abc=-1, abd=1이라면, -1*1=-1, abc=1, abd=-1이라면, 1*-1=-1

직선임을 가정했을 때, 위의 1, 2, 3 경우를 통해 abc*abd ≤ 0인 경우 abc와 abd는 교차한다.
단, 우리가 고려하는 $\overline{ab}$ 와 $\overline{bc}$ 는 직선이 아닌 선분이다. 이 경우, 고려해야 할 것이 몇 가지 더 생긴다.

위 3의 경우 직선인 경우에는 교차하지만, 선분의 경우 교차하지 않는 아래와 같은 경우가 생긴다.



이 경우를 위해 cda와 cdb를 추가로 고려해준다. ab = abc*abd ≤ 0 이지만, cd = cda*cdb = 1이 된다.
즉, ab ≤ 0 이고, cd ≤ 0인 경우에만 교차할 수 있다.

하나 더 고려해야 할 것이 있다. 위 1의 경우에도 아래와 같은 경우가 있다.

선분 $\overline{ab}$ 와 선분 $\overline{bc}$ 는 분명히 같은 직선상의 선분이지만, 분명히 교차하지 않는다.
이 경우는 ab=0이고, cd=0이다. 이 경우에는 각 점의 위치를 이용하여 교차여부를 판별한다. 즉, 선분 $\overline{ab}$ 의 연장인 선분 위의 두 정점 x, y를 연결한 선분 $\overline{xy}$ 에서, 정점 a와 정점 b 중 정점 x에 가까운 정점을 정점 α, 정점 y에 가까운 정점을 정점 β라 하고, 정점 c와 정점 d 중 정점 x에 가까운 정점을 ɣ, 정점 y에 가까운 정점을 δ라 했을 때, α 가 δ보다 y에 멀거나 같은 위치이고, ɣ가 β보다 x에 멀거나 같은 위치인 경우에는 두 선분 사이의 교점이 1개 이상 생긴다.

이제 선분의 교차에서 모든 경우를 판별할 수 있게 되었지만, 우리는 노드와 경로로 구성된 그래프에서의 교차가 있는지 없는지를 판단해야 한다. 경로 ab와 bc가 같은 정점 b를 공유한다고 해서 ab와 bc가 무조건 교차하지는 않는다는 것이다.

이 경우, "α 가 δ보다 y에 멀거나 같은 위치이고, ɣ가 β보다 x에 멀거나 같은 위치인 경우에는 두 선분 사이의 교점이 1개 이상 생긴다." 는 명제가 거짓이 된다. 교점이 생겨도 되므로 해당 명제는
"α 가 δ보다 y에 먼 위치이고, ɣ가 β보다 x에 먼 위치인 경우에는 두 선분은 교차한다." 로 수정한다.
이때, 편의상 선분 $\overline{xy}$ 상의 임의의 정점 ψ보다 임의의 정점 ω가 y에 가까운 경우, ψ<ω로, x에 가까운 경우 ψ>ω로 표기하기로 한다.

또, 다른 모든 경우에 대해서도 한 점은 공유를 하되, 그 점이 각 경로의 양 끝단 점인 경우를 고려해야 한다. 즉, "ab ≤ 0 이고, cd ≤ 0인 경우에만 교차할 수 있다." 라는 명제도 거짓이 된다. 이는 즉, "ab ≤ 0 이고, cd ≤ 0이고 경로를 구성하는 양 끝 정점은 서로 공유하지 않는 경우에는 교차한다." 로 수정해야 한다. 이렇게 모든 경우를 고려하여 두 경로가 교차하는지 아닌지의 여부를 판별할 수 있다.
이를 정리하면, 함수 isCross(a, b, c, d)는 다음과 같이 정의한다.

isCross(a, b, c, d) {
- True ((조건5이고 조건4) 이거나 (조건5가 아니고 (조건6이고 조건7이 아닌 경우)))
- False ((조건5이고 조건4가 아닌 경우) 이거나 (조건5가 아니고 (조건6이고 조건7이 아닌 경우))가 아닌 경우)

단, 각 조건은 다음과 같다.
조건1: ab=0 이고 cd=0
조건2: abc=abd=cda=cdb=0
조건3: a≠c 이고 a≠d 이고 b≠c 이고 b≠d
조건4: ɣ<β 이고 α<δ
조건5: 조건1 이고 (조건2 이거나 조건3) 이고 조건4
조건6: ab ≤ 0 이고 cd ≤ 0
조건7: a=c 이거나 a=d 이거나 b=c 이거나 b=d

그래프의 교차여부를 판별하는 방법 - 선분과 한 점의 위치관계 판별

선분과 점의 위치관계 판별은 세 점을 순서대로 진행하는 방향이 어떤가로 구할 수 있다.
즉, 세 점의 위치관계 판별로 대신 할 수 있다.
이는 세 점의 위치정보를 세 좌표를 순서대로 진행하는 두 경로를 벡터로 취급하여 외적한 결과를 이용해 알 수 있다. 임의의 벡터 $A=(x_A, y_A)$와 $B(x_B, y_B)$가 있다고 할 때, 두 벡터의 벡터 곱 $A\times B = (x_Ay_B, y_Ax_B, 0) = |A|*|B|*\sin\theta*N = -B\times A$이다. 이 때, N은 두 벡터 A, B에 수직인 방향을 갖는 단위벡터이다. 세 점 $a=(x_1, y_1)$, $b=(x_2, y_2)$, $c=(x_3, y_3)$를 이용해 벡터를 구해보면, 임의의 두 점 p에서 q로 향하는 벡터를 →pq라고 할 때, $\rightarrow ab=(x_2-x_1, y_2-y_1)$, $\rightarrow ac=(x_3-x_1, y_3-y_1)$ 가 된다. 외적을 이용해야하므로 2차원 벡터를 3차원 벡터로 표현하면, $\rightarrow ab=(x_2-x_1, y_2-y_1, 0)$, $\rightarrow ac=(x_3-x_1, y_3-y_1, 0)$ 으로 표현할 수 있고, 이 벡터의 외적 $\rightarrow ab\times\rightarrow ac=(0, 0, (x_2-x_1)(y_3-y_1)-(y_2-y_1)(x_3-x_1))$이다. 이 때, 이 벡터의 z방향은 $(x_2-x_1)(y_3-y_1)-(y_2-y_1)(x_3-x_1) = x_2y_3-x_2y_1-x_1y_3+x_1y_1 - (y_2x_3-y_2x_1-y_1x_3+y_1x_1) = x_1y_2+x_2y_3+x_3y_1 - (y_1x_2+y_2x_3+y_3x_1)$가 된다. 오른손 법칙에 의해 이 값이 양수인지 음수인지에 따라 법선벡터의 방향이 반대가 되고, 이 값이 0인 경우 동일한 직선상에 위치하므로 선분 $\overline{ab}$ 에 대한 점c의 위치관계를 알 수 있다. $result=x_1y_2+x_2y_3+x_3y_1-(y_1x_2+y_2x_3+y_3x_1)$라 가정하여 ccw(a, b, c)를 다음과 같이 정의한다.

ccw(a, b, c) {
- 1 (단, result > 0)
- 0 (단, result = 0)
- -1 (단, result < 0)

2) 자료구조 및 변수 설명

파라미터로 사용한 변수의 경우 '2-1)-(1) 알고리즘(함수) 구조' 참고

자료구조	list	타입에 상관없이 자료를 담을 수 있는 순서가 있으며 변경 가능한 파이썬 기본 자료구조
	tuple	타입에 상관없이 자료를 담을 수 있는 순서가 있으며 변경 불가능한 파이썬 기본 자료구조
	deque, queue	큐는 선입선출구조로 가장 먼저 들어온 데이터가 가장 먼저 나가는 자료구조이

		다. 큐의 경우엔 파이썬에서 기본으로 제공하는 것이 없어 collections 라이브러리를 import 하여 deque(double-ended queue) 자료구조를 이용했다. 리스트를 사용할 수도 있으나, 리스트의 경우 배열과 마찬가지로 원소의 중간에 삽입이나 삭제가 일어날 경우 인덱스를 다시 계산하는 연산이 들어가 매우 비효율적이 된다.
변수		

전역변수

result	계산된 그래프를 저장하는 리스트
pts	사용자가 찍은 정점의 좌표를 저장하는 리스트
idx	현재 표시하는 그래프가 몇 번째인지 저장하는 정수
ovalParams	정점을 찍는 함수의 실질매개변수들의 리스트
showMode	현재 그리기 모드인지 그래프 표시 모드인지 저장하는 부울 대수

showFrame() 지역변수(GUI 오브젝트)

root	프로그램 윈도우
canvas	왼쪽 레이아웃
frameButtonBox	오른쪽 레이아웃
textX	'X'를 표시하는 텍스트
textPosX	x좌표를 표시하는 텍스트
textY	'Y'를 표시하는 텍스트
textPosY	y좌표를 표시하는 텍스트
btnReset	Reset 버튼
btnConfirm	Confirm 버튼
btnPrevious	Previous 버튼
btnNext	Next 버튼
textNow	'Now'를 표시하는 텍스트
textCurrentIdx	현재 그래프가 몇 번째인지 표시하는 텍스트
textEnd	'End'를 표시하는 텍스트
textEndIdx	그래프의 개수가 몇 개인지 표시하는 텍스트

bfsGraph(points) 지역변수

n	정점의 개수를 담는 정수
result	계산한 그래프를 담을 리스트
enQ	deque 클래스의 인스턴스인 queue의 append 함수
deQ	deque 클래스의 인스턴스인 queue의 popleft 함수
point	points의 원소인 좌표정보를 갖는 튜플
new	계산한 그래프를 result에 삽입 전, 중복 여부를 검사하기 위한 리스트
flag	계산한 그래프가 중복이 아니면 True를 갖는 부울 대수

ccw(pos1, pos2, pos3) 지역변수

x1, x2, x3	각각 pos1, pos2, pos3의 x좌표를 담는 정수
y1, y2, y3	각각 pos1, pos2, pos3의 y좌표를 담는 정수
tmp	계산 결과를 담는 정수

isCross(a, b, c, d) 지역변수

abc, abd, cda, cdb	각각의 ccw() 호출결과를 담는 정수
ab	abc*abd를 담는 정수
cd	cda*cdb를 담는 정수

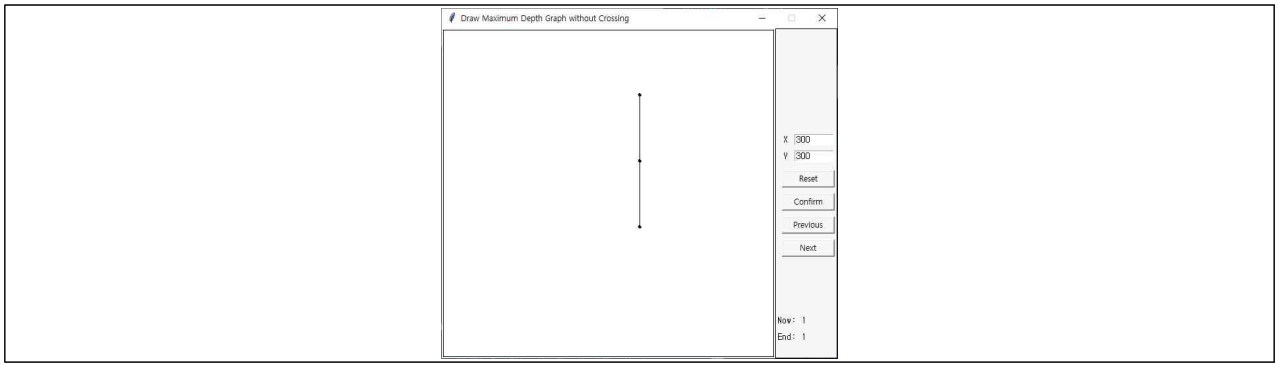
isCrossGraph(graph) 지역변수

i	교차여부를 판별할 선분 1을 가리킬 인덱스를 담는 정수
j	교차여부를 판별할 선분 2를 가리킬 인덱스를 담는 정수

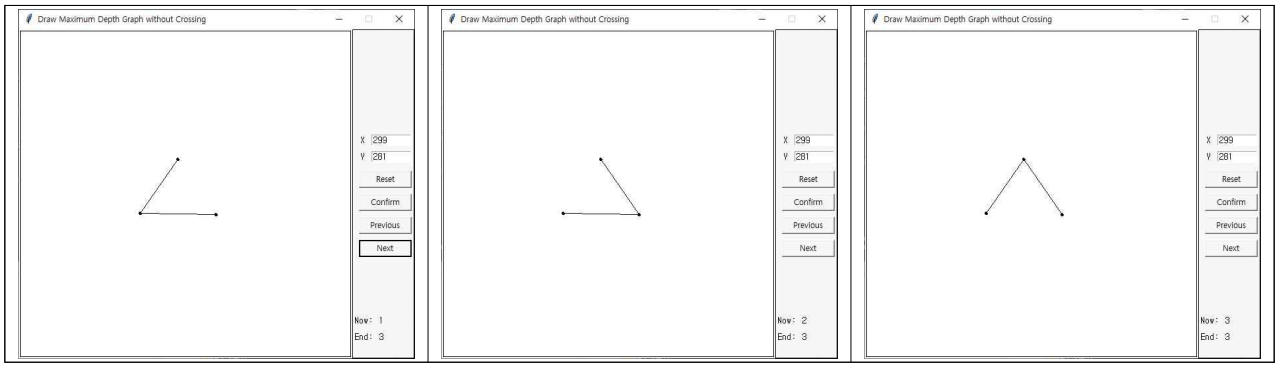
3. 실행 결과

1) 점의 개수: 3

(1) 일직선: 1개의 그래프

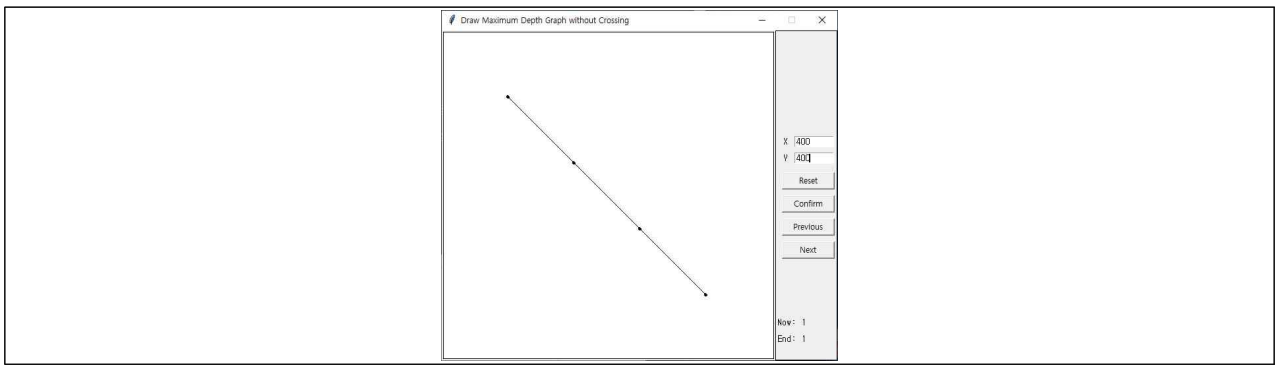


(2) 삼각형모양: 3개의 그래프



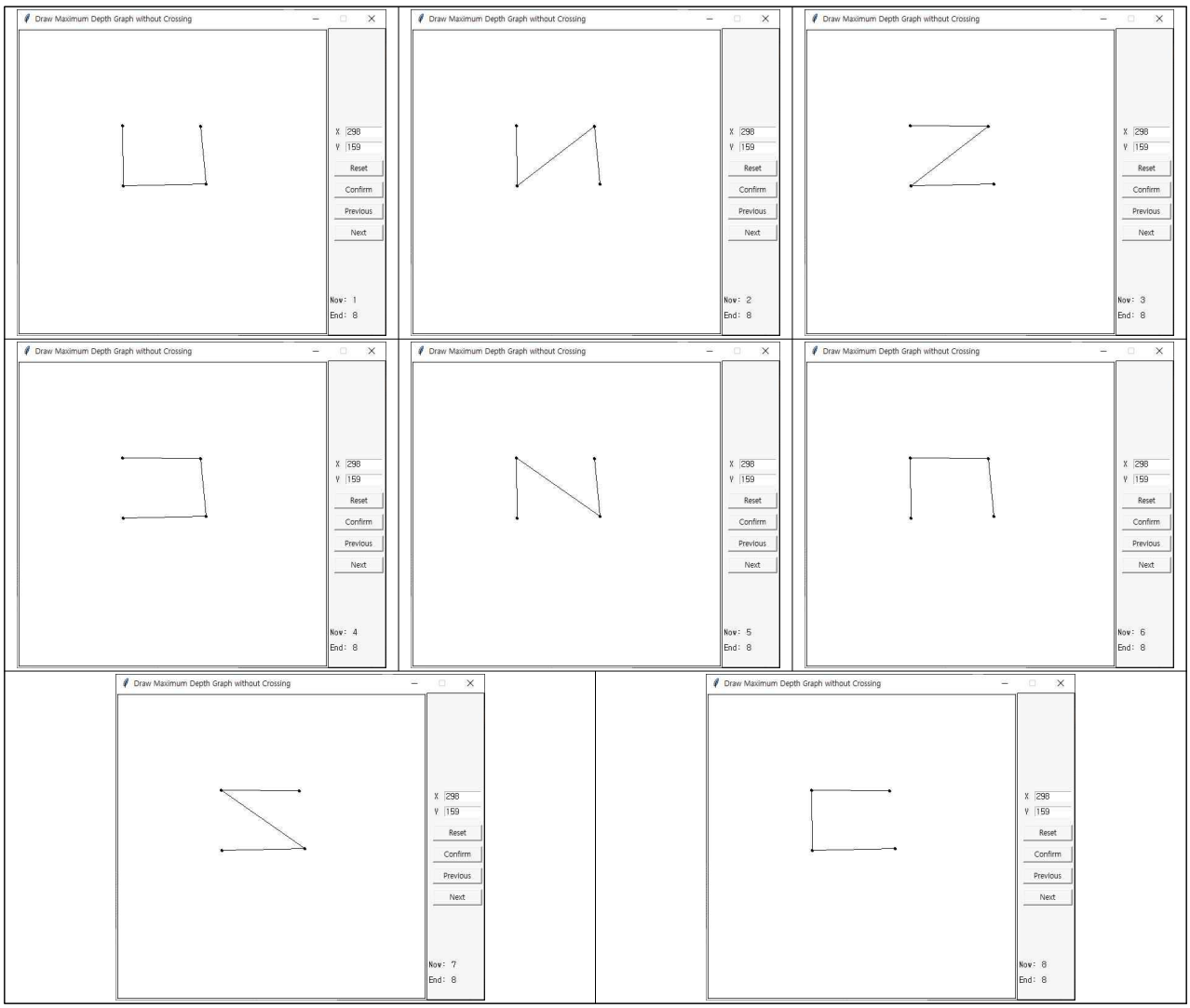
2) 점의 개수: 4

(1) 일직선: 1개의 그래프



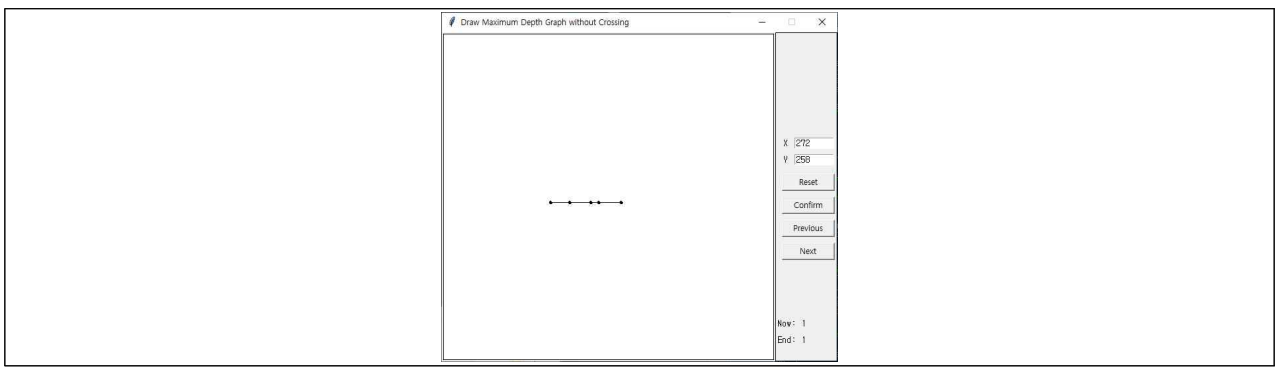
(2) 삼각형모양 및 내부의 점: 12개의 그래프

(3) 사각형모양: 8개의 그래프



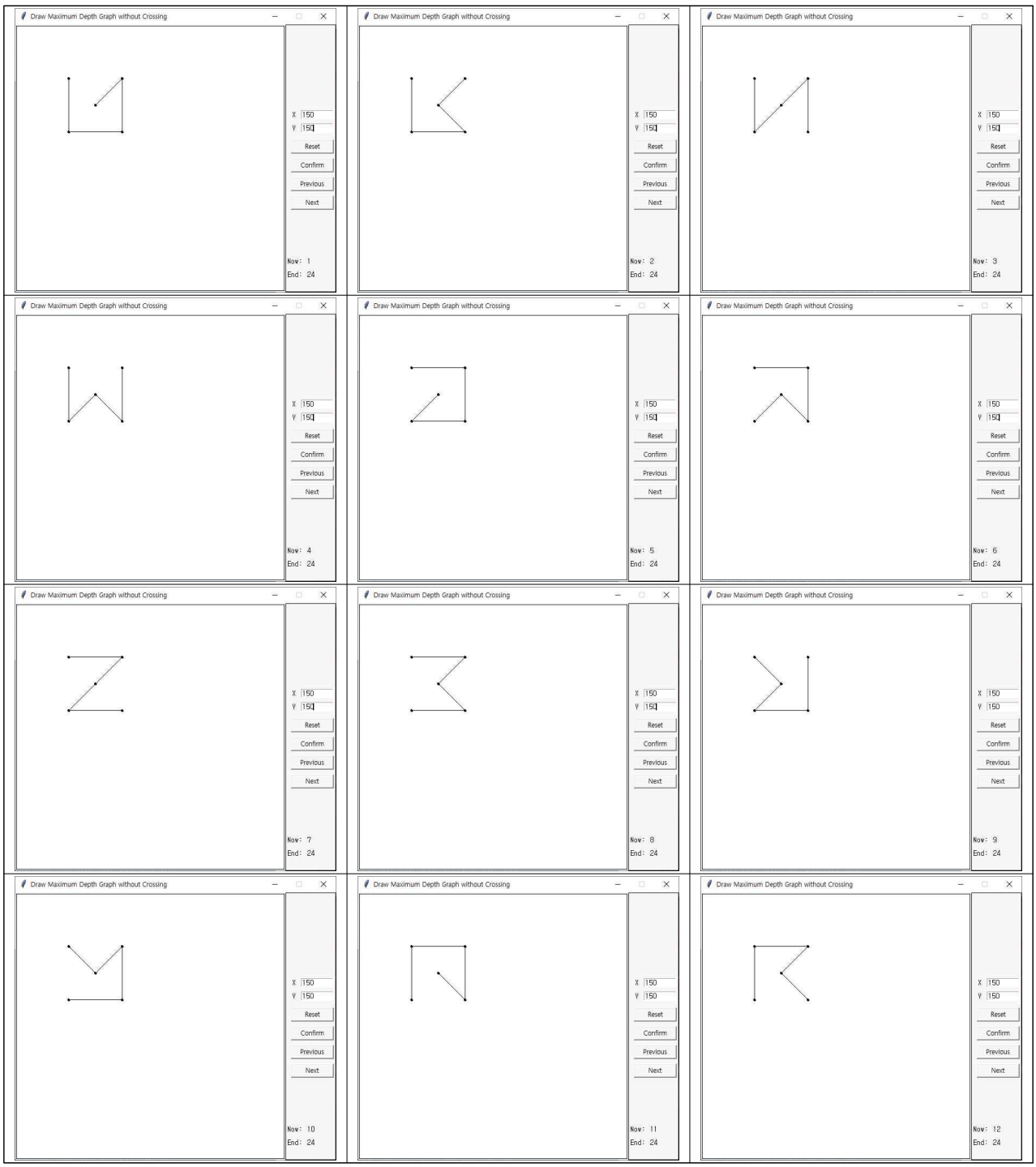
3) 점의 개수: 5

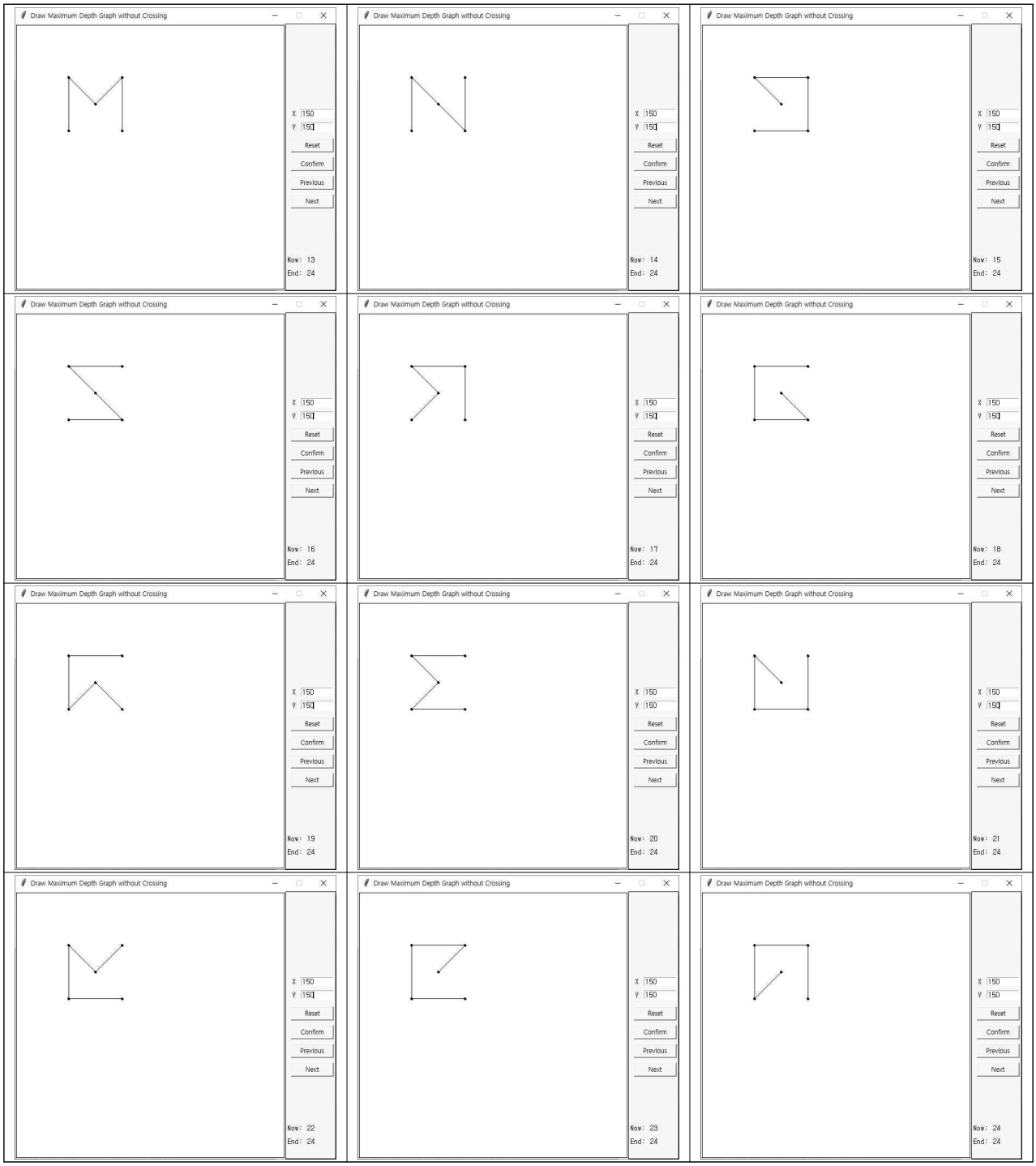
(1) 일직선: 1개의 그래프



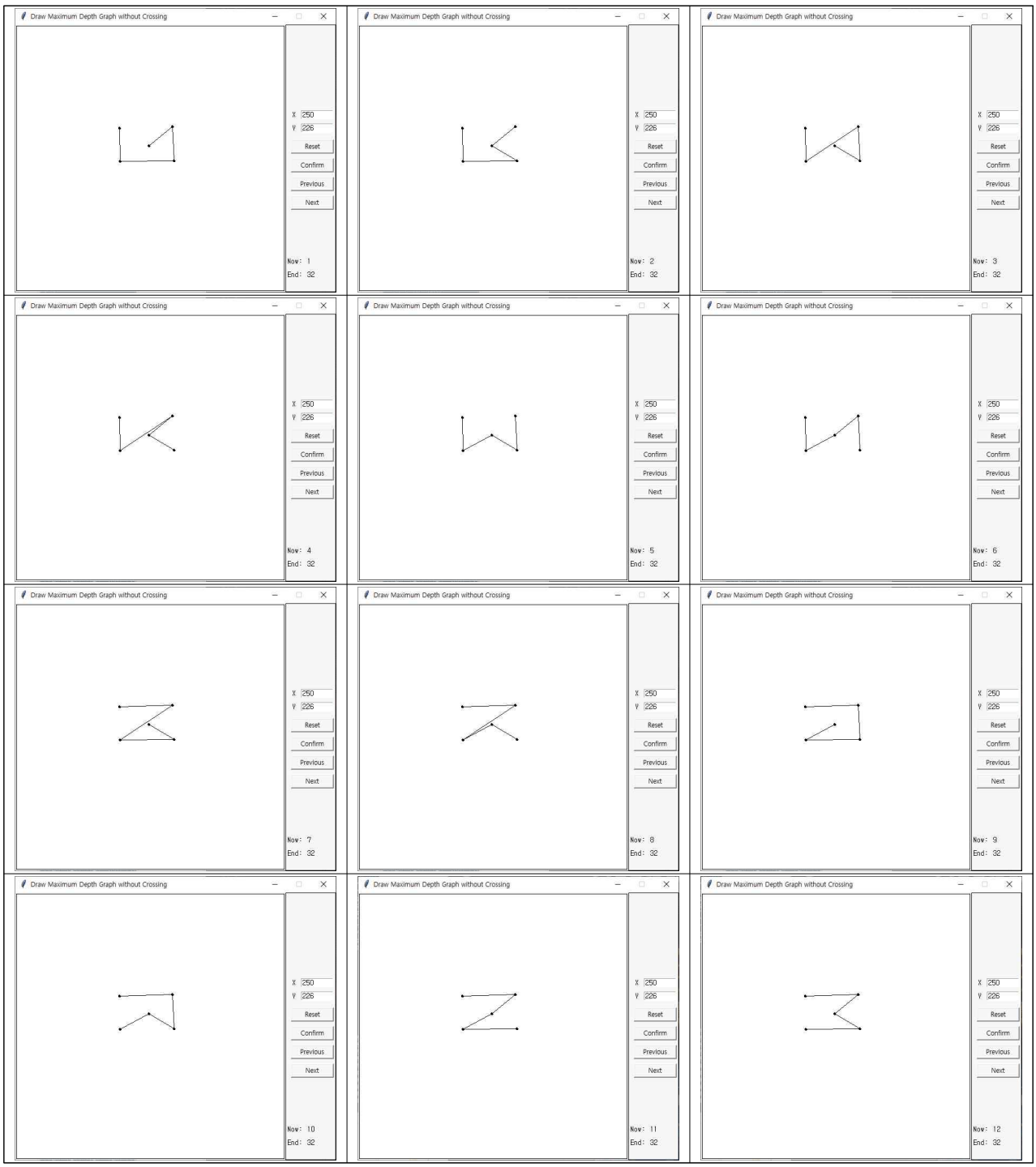
(2) 일직선과 점: 10개의 그래프

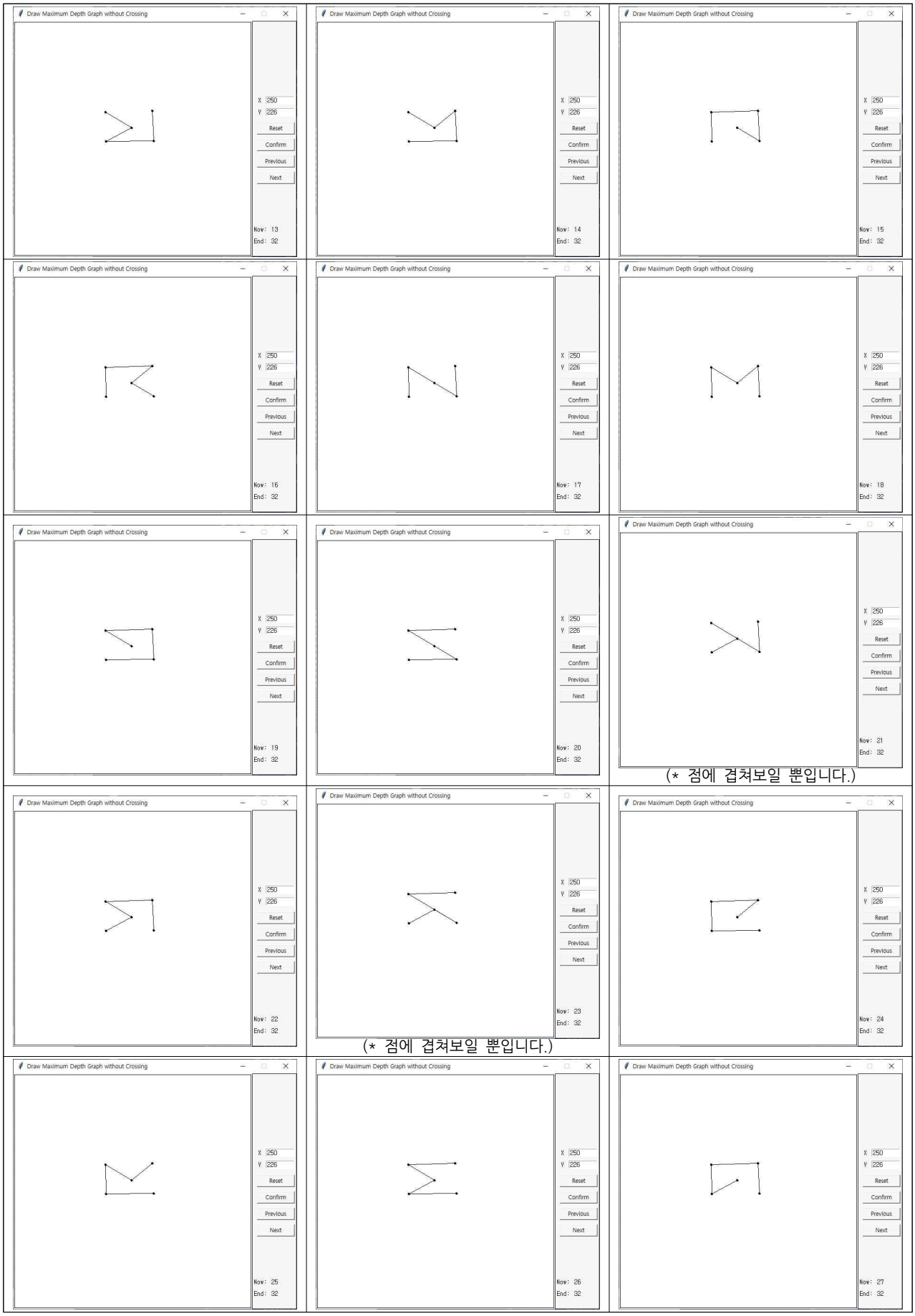
(3-1) 정사각형모양 및 중앙의 점: 24개의 그래프




Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 13
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 14
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 15
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 16
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 17
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 18
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 19
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 20
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 21
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 22
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 23
End: 24
Draw Maximum Depth Graph without Crossing
X 150
Y 150
Reset
Confirm
Previous
Next
Now: 24
End: 24

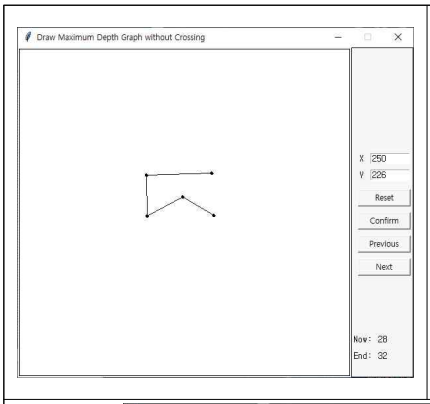
(3-2) 사각형모양 및 내부의 점: 32개의 그래프

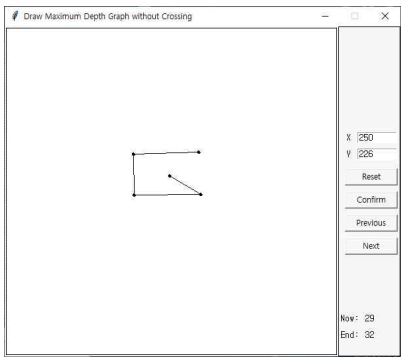


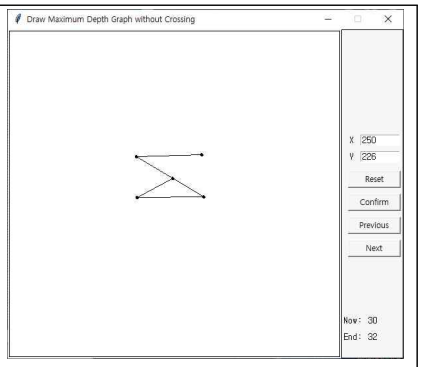


(* 점에 겹쳐보일 뿐입니다.)

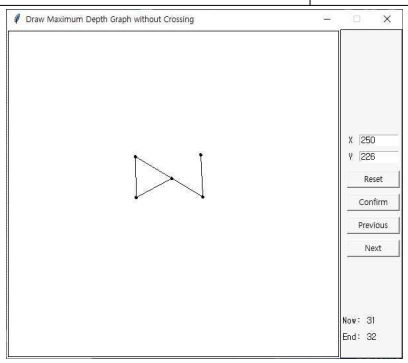
(* 점에 겹쳐보일 뿐입니다.)



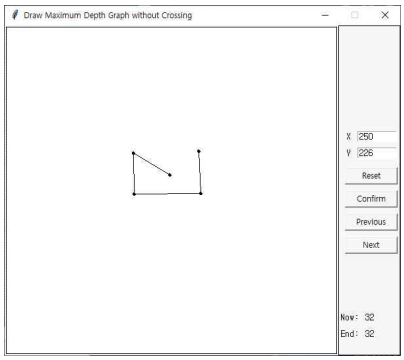




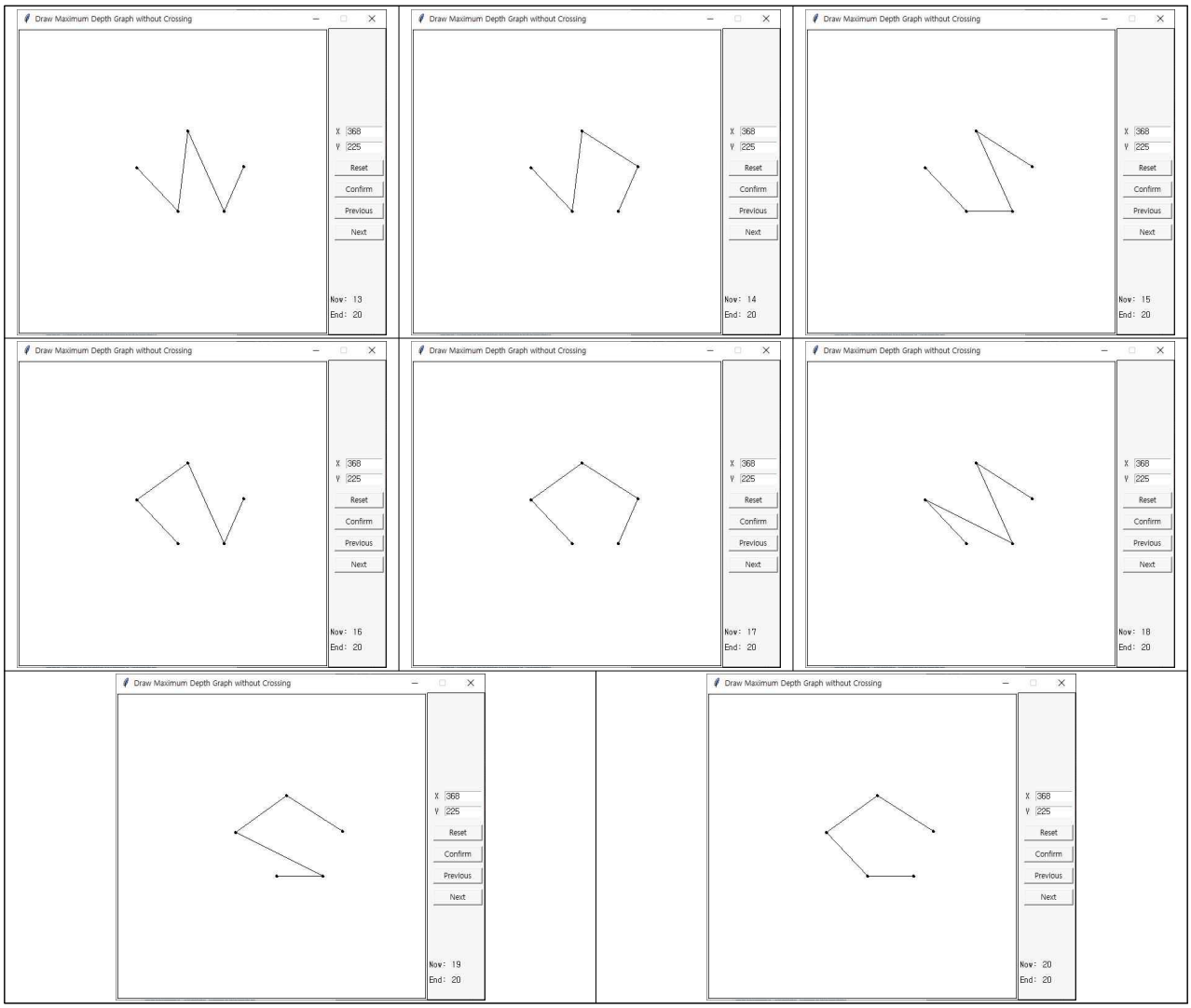
(* 점에 겹쳐보일 뿐입니다.)



(* 점에 겹쳐보일 뿐입니다.)



(4) 오각형모양: 20개의 그래프


Draw Maximum Depth Graph without Crossing
X 368
Y 225
Reset
Confirm
Previous
Next
Now: 13
End: 20
Draw Maximum Depth Graph without Crossing
X 368
Y 225
Reset
Confirm
Previous
Next
Now: 14
End: 20
Draw Maximum Depth Graph without Crossing
X 368
Y 225
Reset
Confirm
Previous
Next
Now: 15
End: 20
Draw Maximum Depth Graph without Crossing
X 368
Y 225
Reset
Confirm
Previous
Next
Now: 16
End: 20
Draw Maximum Depth Graph without Crossing
X 368
Y 225
Reset
Confirm
Previous
Next
Now: 17
End: 20
Draw Maximum Depth Graph without Crossing
X 368
Y 225
Reset
Confirm
Previous
Next
Now: 18
End: 20
Draw Maximum Depth Graph without Crossing
X 368
Y 225
Reset
Confirm
Previous
Next
Now: 19
End: 20
Draw Maximum Depth Graph without Crossing
X 368
Y 225
Reset
Confirm
Previous
Next
Now: 20
End: 20