STEP16. 복잡한 계산 그래프 (구현 편)

함수 우선 순의 설정 방법

- 순전파 시 '세대 ' 를 설정
- 역전파 시 최근 세대의 함수부터 꺼냄

> 💡 순전파에서 세대 추가
> → 순전파을 하면 모든 변수와 함수의 세대가 설정됨
> 역전파에서 세대 순으로 꺼냄

세대 추가

1. 순전파시 세대 (generation) 추가

```
import numpy as np

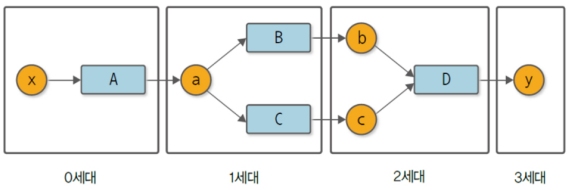
class Variable:
    def __init__(self, data):
        # 입력 데이터가 None이 아닌 경우, 입력 데이터의 타입이 np.ndarray인지 확인
        if data is not None:
            if not isinstance(data, np.ndarray):
                raise TypeError('{}은(는) 지원하지 않습니다.'.format(type(data)))

        # 변수의 데이터를 입력 데이터로 설정
        self.data = data
        # 변수의 기울기 초기화
        self.grad = None
        # 변수를 생성한 함수(연산) 초기화
        self.creator = None
        # 세대 수를 기록하는 변수
        self.generation = 0

    def set_creator(self, func):
        self.creator = func
        # 세대를 기록 (부모 세대 + 1)
        self.generation = func.generation + 1

    ```
```

- Variable 과 Function 클래스에 인스턴스 변수 generation을 추가
- 몇 번째 '세대 ' 의 함수(혹은 변수)인지 나타내는 변수
- generation을 0으로 초기화
- set_creator 메서드 호출될 때 부모 함수의 세대보다 1만큼 큰 값을 설정




```
import numpy as np

class Function:
    def __call__(self, *inputs): # *를 붙여 입력을 리스트가 아닌 인수 목록으로 받음
        # 입력으로 받은 변수들의 데이터를 추출하여 리스트에 저장
        xs = [x.data for x in inputs]

        # forward 메서드를 호출하여 순전파를 수행
        ys = self.forward(*xs) # 별표를 붙여 언팩

        # 튜플이 아닌 경우 추가 지원
        if not isinstance(ys, tuple):
            ys = (ys,)

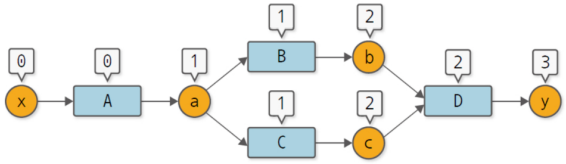
        # 순전파 결과로부터 Variable 객체 생성
        outputs = [Variable(as_array(y)) for y in ys]

        # generation 설정
        self.generation = max([x.generation for x in inputs])

    ```
```

- generation 은 입력 변수와 같은 값으로 설정
- 입력 변수가 둘 이상이라면 가장 큰 generation 수를 선택

세대 순으로 꺼내기



역전파 시에 세대 순으로 꺼냄

- 순전파를 수행하면 모든 변수와 함수에 세대가 설정됨
- A, B, C, D의 세대는 차례로 0, 1, 1, 2 임
- 세대가 설정되면 역전파 때 함수를 올바른 순서대로 꺼낼 수 있음

함수를 세대 순으로 꺼내는 테스트

- 더미 함수를 준비하고 funcs 리스트에 추가
  - 이 리스트에서 세대가 가장 큰 함수 꺼냄

```
>>> class Function:
...     def __init__(self, generation):
...         self.generation = generation
...
>>> generate = [2, 3, 0, 1, 4]
>>> funcs = []
>>> for g in generate:
...     f = Function(g)
...     funcs.append(f)
...
>>> [f.generation for f in funcs]
[2, 3, 0, 1, 4]
```

```
>>> funcs.sort(key=lambda x: x.generation)
>>> [f.generation for f in funcs]
[0, 1, 2, 3, 4]
```

- 리스트의 sort 메서드를 이용하여 generation을 오름차순으로 정렬함

(이 메서드의 인수인 `key=lambda x: x.generation` 은 리스트의 원소를 x라고 했을 때 `x.generation` 을 키로 사용해 정렬하라)

- pop 메서드를 써서 리스트의 끝 원소를 꺼내면 자연스럽게 세대가 가장 큰 함수를 얻을 수 있음 (우선순위 큐를 이용하면 더 효율적임)

Variable 클래스의 backward

```
class Variable:
```
    def backward(self):
        # y.grad = np.array(1.0) 생략을 위한 if문
        if self.grad is None:
            self.grad = np.ones_like(self.data)

        funcs = []
        seen_set = set()

        def add_func(f):
            if f not in seen_set:
                funcs.append(f)
                seen_set.add(f)
                funcs.sort(key=lambda x: x.generation)

        add_func(self.creator)

        while funcs:
            f = funcs.pop() # 함수를 가져온다.
            gys = [output.grad for output in f.outputs]
            gxs = f.backward(*gys) # 함수 f의 역전파 호출 (리스트 언팩)

            # gxs가 튜플이 아니라면 튜플로 변환
            if not isinstance(gxs, tuple):
                gxs = (gxs,)

            # 역전파로 전파되는 미분값을 Variable인스턴스 변수 grad에 저장
            for x, gx in zip(f.inputs, gxs): # gxs와 f.inputs는 대응
                if x.grad is None:
                    x.grad = gx
                else:
                    # x.grad += gx <- 문제 발생 (부록 A)
                    x.grad = x.grad + gx

                if x.creator is not None:
                    add_func(x.creator) # 수정전 : funcs.append(x.creator)

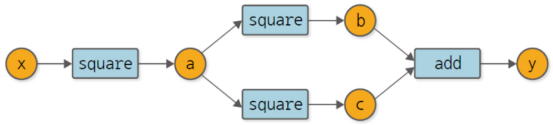
            if x.creator is not None:
                # 하나 앞의 함수를 리스트에 추가한다.
                funcs.append(x.creator)
```

- `add_func` 함수 추가
  - 함수 리스트를 세대 순으로 정렬하는 역할
  - backward 메서드 안에 중첩 함수로 정의

    [ seen_set : func 리스트에 같은 함수 중복 추가를 막기 위해 사용 ]
    - 감싸는 메서드 ( backward 메서드 ) 안에서만 이용
    - 감싸는 메서드 ( backward 메서드 ) 에 정의된 변수 ( funcs과 seen_set ) 를 사용
```

- 함수를 메서드(backward 메서드)에 정의된 순서(funcs와 seen_set)를 사용

동작 확인




```
x = Variable(np.array(2.0))
a = square(x)
y = add(square(a), square(a))
y.backward

print(y.data)
print(x.grad)
```