07강

알고리즘과 자료구조

# 몬테카를로 시뮬레이션

서울과학기술대학교 신일훈 교수

한국방송통신대학교 프리임칼리지

# 학습목표

1. 몬테카를로 시뮬레이션의 개념을 이해한다.
2. 몬테카를로 시뮬레이션을 활용하여 주사위 문제를 해결할 수 있다.
3. 몬테카를로 시뮬레이션을 활용하여 원주율 및 적분 등을 계산할 수 있다.
4. 몬테카를로 시뮬레이션을 활용하여 knapsack 문제를 해결할 수 있다.

01

# 몬테카를로 시뮬레이션 개념

# 1. 몬테카를로 시뮬레이션 개념

## 몬테카를로 시뮬레이션 개념

- 목적: 특정 이벤트가 발생할 확률을 계산
- 방법: randomness에 기반하여 시뮬레이션을 설계.
  시뮬레이션을 여러 번 수행하여 이벤트 발생 횟수를 계수
  prob = counts / tries
- 장점: 충분한 시도를 통해, 수학적으로 계산이 어려운 문제도 해결 가능
  - 포커 게임 확률
  - 생일이 같을 확률
  - 원주율 계산
  - ...

02

# 주사위

# 2. 주사위

- 주사위를 굴렸을 때 10이 나올 확률은?
  - 방법1: 수학적으로 확률을 계산할 수 있다.
    - 이 경우에는 간단함. 하지만 계산이 어려운 문제들도 있음.
  - 방법2: 시뮬레이션을 수행하여 확률 계산
    - (10이 나온 회수) / (전체 시도 회수)
    - 문제는 정확성.
    - 정확성을 높이려면 일단 randomness가 전제되어야 함.
    - 많이 시도해야 함.

# 2. 주사위

- 주사위를 굴렀을 때 10이 나올 확률은?

의사코드

```
def rollDice():
    num = randint(1, 6)     #generate an integer between 1 and 6
    return num
```

# 2. 주사위

- 주사위를 굴렀을 때 10이 나올 확률은?

의사코드

```
def calDiceProb(tries, target) :
    count = 0
    for i in range(tries):
        num = rollDice()
        if (num == target) :
            count += 1
    return (count/tries)
```

# 2. 주사위

- 주사위를 굴렀을 때 10이 나올 확률은?

파이썬 코드

```
import random
def rollDice():
    num = random.randint(1, 6)
    return num
```

# 2. 주사위

- 주사위를 굴렀을 때 10이 나올 확률은?

파이썬 코드

```
def calDiceProb(tries, target) :
    count = 0
    for i in range(tries):
        num = rollDice()
        if (num == target) :
            count += 1
    return (count/tries)
```

# 2. 주사위

- 주사위를 굴렀을 때 10이 나올 확률은?

파이썬 코드

```
print(calDiceProb(100, 1))
print(calDiceProb(1000, 1))
print(calDiceProb(10000, 1))
print(calDiceProb(100000, 1))
```

# 2. 주사위

- 주사위를 굴렀을 때 10이 나올 확률은?

## 파이썬 코드 실행

```
In [109]: runfile('D:/data/재정/주식소스/stock/result/
untitled3.py', wdir='D:/data/재정/주식소스/stock/result')
0.18
0.151
0.17
0.16675
```

03

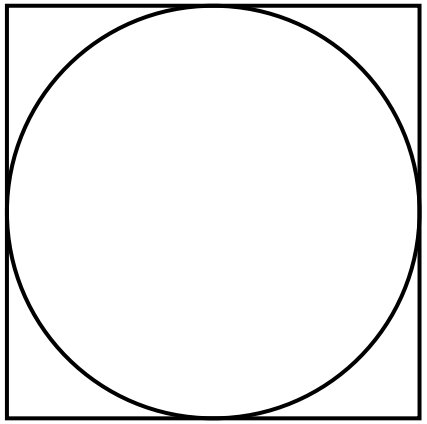
# 원주율 계산

# 3. 원주율 계산

## ❑ 원주율(π)를 계산하시오.

### 아이디어

- 한 변 2인 정사각형과 그 안에 내접한 반지름이 1인 원
- 정사각형 넓이 = 2 * 2 = 4
- 원의 넓이 = π * r * r = π
- 반지름이 1인 원의 넓이를 어떻게 구할까?

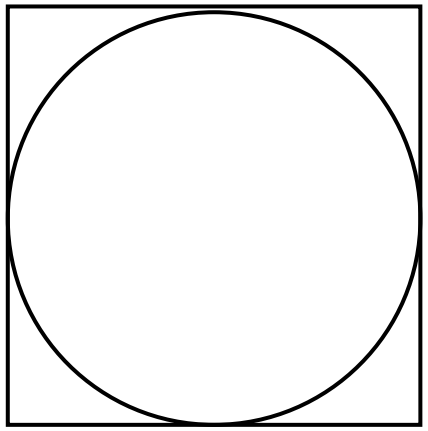


# 3. 원주율 계산

## ❑ 원주율(π)를 계산하시오.

### 아이디어

1. 정사각형의 넓이를 이용하여 원의 넓이를 구하자.



2. 넓이는 점들의 집합이다.
   - 무작위로 굉장히 많은 점을 찍는다면,
     - 정사각형 안의 점의 개수 : 원 안의 점의 개수 = 정사각형의 넓이 : 원의 넓이

=> 원의 넓이(π) = (원 안의 점의 개수 * 정사각형의 넓이) / 정사각형 안의 점의 개수

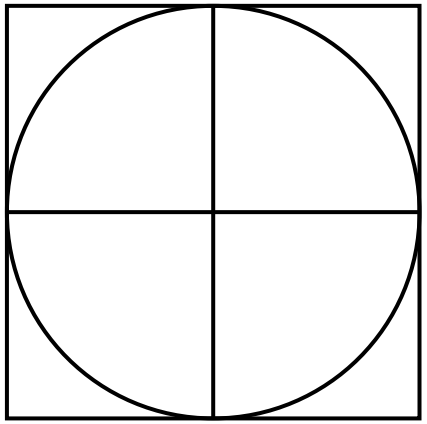
= (원 안의 점의 개수 * 4) / 정사각형 안의 점의 개수

# 3. 원주율 계산

## ❑ 원주율(π)를 계산하시오.

### 아이디어

1. 정사각형의 넓이를 이용하여 원의 넓이를 구하자.



2. 넓이는 점들의 집합이다.
   - 무작위로 굉장히 많은 점을 찍는다면,
     - 정사각형 안의 점의 개수 : 원 안의 점의 개수 = 정사각형의 넓이 : 원의 넓이

=> 원의 넓이(π) = (원 안의 점의 개수 * 4) / 정사각형 안의 점의 개수

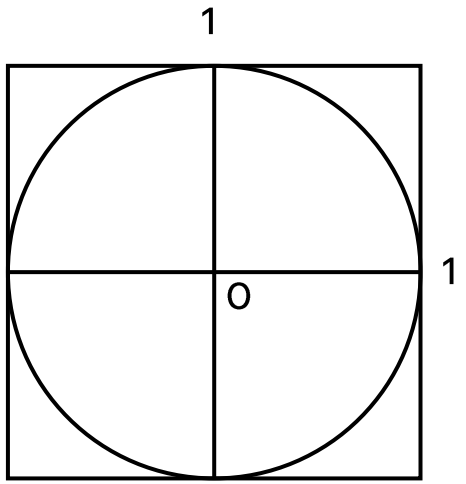
= (부채꼴 안의 점의 개수 * 4) / 작은 정사각형 안의 점의 개수

# 3. 원주율 계산

## ❑ 원주율(π)를 계산하시오.

### 아이디어

1. 정사각형의 넓이를 이용하여 원의 넓이를 구하자.




2. 넓이는 점들의 집합이다.
   - 무작위로 굉장히 많은 점을 찍는다면,
     - 정사각형 안의 점의 개수 : 원 안의 점의 개수 = 정사각형의 넓이 : 원의 넓이

=> 원의 넓이(π) = (원 안의 점의 개수 * 4) / 정사각형 안의 점의 개수

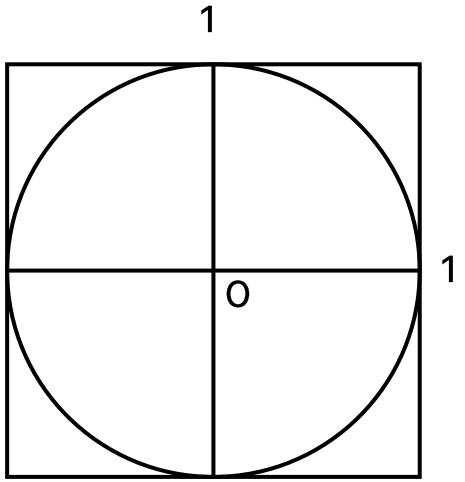
= (부채꼴 안의 점의 개수 * 4) / 작은 정사각형 안의 점의 개수

# 3. 원주율 계산

## ❑ 원주율(π)를 계산하시오.

### 아이디어

- 무작위로 점을 찍는 것을 시뮬레이션으로 표현?
  - 0~1 사이의 난수 발생 => 점의 x좌표
  - 0~1 사이의 난수 발생 => 점의 y좌표




- 점이 부채꼴 안에 있는지를 어떻게 판단?
  - 원점에서 점까지의 거리가 1이하이면 부채꼴 안에 있음.

# 3. 원주율 계산

- 원주율(π)를 계산하시오.

의사코드

```
def calPI(tries):
    insideCnt = 0
    for i in range(tries):
        x = random(0, 1)  # generate a random number between 0~1
        y = random(0, 1)
        dist = sqrt(x*x + y*y)    # calculate the distance from the origin
        if (dist <= 1):
            insideCnt += 1
    return (4*insideCnt/tries)
```

# 3. 원주율 계산

- 원주율(π)를 계산하시오.

파이썬 코드

```
import random
def calPI(tries = 10000):
    insideCnt = 0
    for i in range(tries):
        x = random.random()
        y = random.random()
```

# 3. 원주율 계산

- 원주율(π)를 계산하시오.

**파이썬 코드**

```
import random
def calPI(tries = 10000):
    insideCnt = 0
    for i in range(tries):
        x = random.random()
        y = random.random()
        dist = (x*x + y*y)**0.5
        if (dist <= 1):
            insideCnt += 1
    return (4*insideCnt/tries)
```

# 3. 원주율 계산

- 원주율(π)를 계산하시오.

파이썬 코드

```
print(calPI(10))
print(calPI(100))
print(calPI(1000))
print(calPI(10000))
print(calPI(100000))
print(calPI(1000000))
```

# 3. 원주율 계산

- 원주율(π)를 계산하시오.

## 파이썬 코드 실행

```
In [113]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/untitled1.py',
wdir='D:/data/lecture/파이썬으로배우는자료구조와알고리즘/code/
알고리즘')
2.8
2.92
3.104
3.128
3.15008
3.14192
```

04

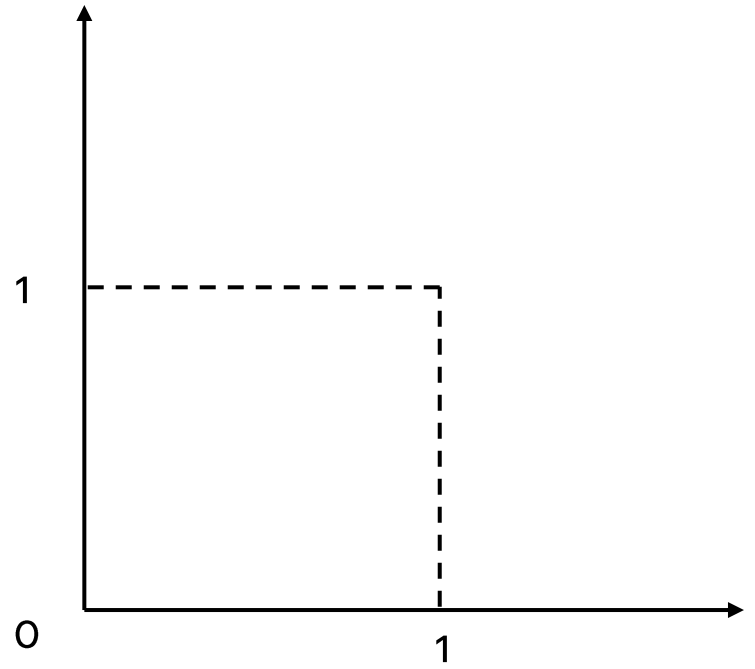
# 적분값 구하기

# 4. 적분값 구하기

- y = x * x 을 0부터 1까지 적분한 값을 계산하시오.

## 아이디어

- 빗금친 넓이가 구하고자 하는 적분값이다.

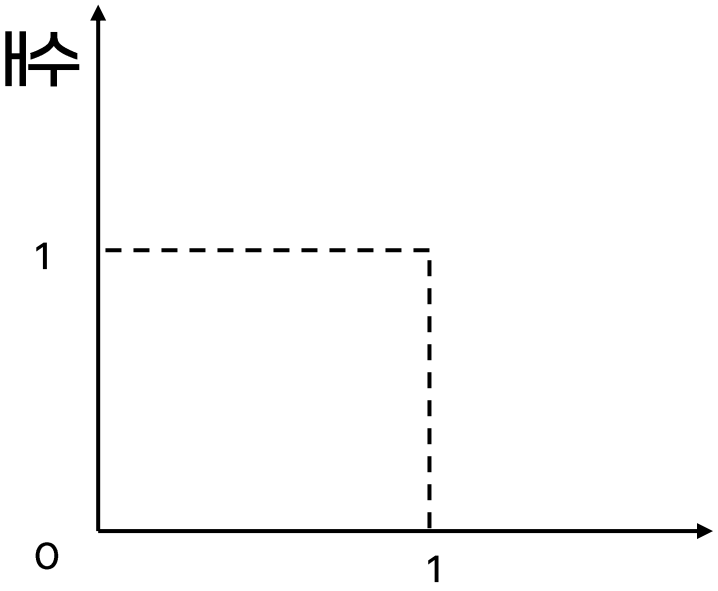


# 4. 적분값 구하기

- y = x * x 을 0부터 1까지 적분한 값을 계산하시오.

## 아이디어

- 정사각형의넓이 : 빗금넓이 = 정사각형안의점의개수 : 빗금안의점의개수
- 빗금넓이 = 빗금안의점의개수 / 정사각형안의점의개수

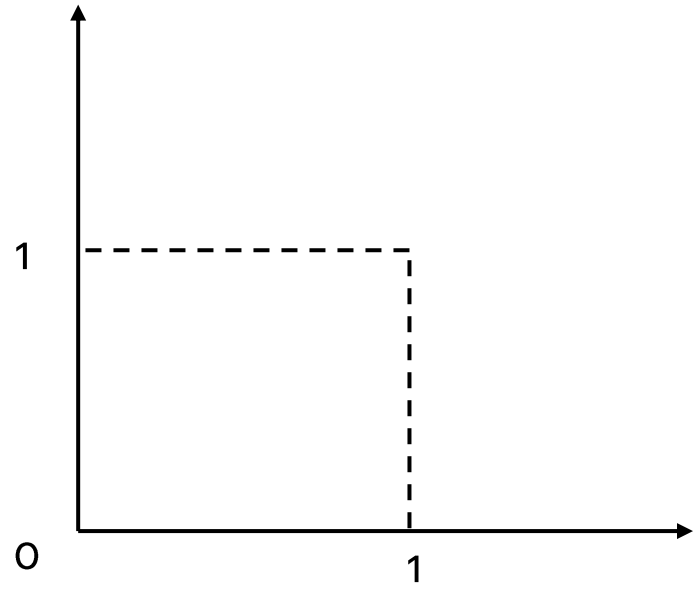


# 4. 적분값 구하기

- y = x * x 을 0부터 1까지 적분한 값을 계산하시오.

## 아이디어

- 점을 찍는 행위
  - 0~1 사이의 두 개의 난수 발생 => (x, y) 좌표
- 빗금 영역 안에 있는 지의 판단
  - if (y <= x * x) => 빗금 영역 안에 있음.



# 4. 적분값 구하기

- y = x * x 을 0부터 1까지 적분한 값을 계산하시오.

의사코드

```
def callntegral(tries):
    insideCnt = 0
    for i in range(tries):
        x = random(0, 1)      # generate a random number between 0~1
        y = random(0, 1)
        if (y <= x * x):
            insideCnt += 1
    return (insideCnt/tries)
```

# 4. 적분값 구하기

- y = x * x 을 0부터 1까지 적분한 값을 계산하시오.

파이썬코드

```
import random
def callntegral(tries = 10000):
    insideCnt = 0
    for i in range(tries):
        x = random.random()
        y = random.random()
        if (y <= x * x):
            insideCnt += 1
    return (insideCnt/tries)
```

# 4. 적분값 구하기

- y = x * x 을 0부터 1까지 적분한 값을 계산하시오.

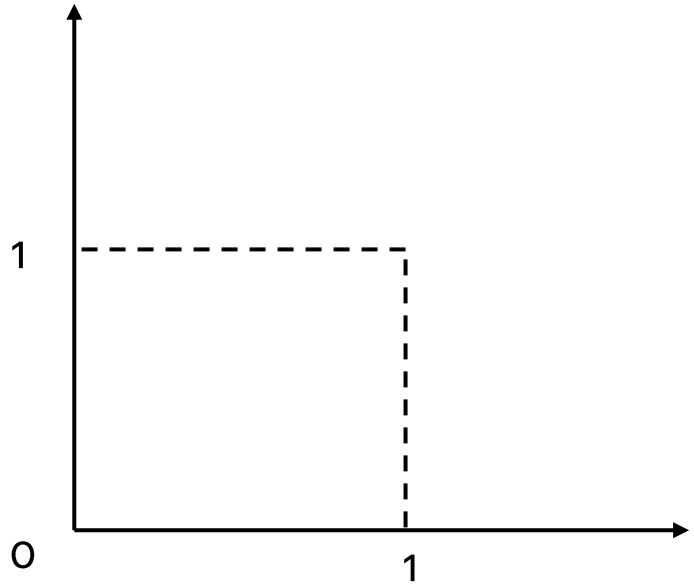
파이썬코드

```
print(callntegral(10))
print(callntegral(100))
print(callntegral(1000))
print(callntegral(10000))
print(callntegral(100000))
print(callntegral(1000000))
```

# 4. 적분값 구하기

## 파이썬 코드 실행

```
0.3
0.26
0.321
0.3338
0.33658
0.333659
```



# 05

# Knapsack

# 5. Knapsack

- 배낭에 넣을 수 있는 최대 무게: W, N개의 물건 (value , weight)
  배낭에 넣을 수 있으면서 가치의 총합을 최대로 하는 물건의 조합을 구하시오.

## 아이디어

names = ['A', 'B', 'C', 'D', 'E'], values = [10, 30, 20, 14, 23]

weights = [5, 8, 3, 7, 9], max_weight = 20

# 5. Knapsack

## 아이디어

names = ['A', 'B', 'C', 'D', 'E'], values = [10, 30, 20, 14, 23]
weights = [5, 8, 3, 7, 9], max_weight = 20

- 각 아이템의 포함 여부를 임의로 결정 (using random 함수)
  - 단, 아이템을 포함했을 시, 최대 무게를 초과하면 해당 아이템은 포함하지 않음

- 여러 번 시도해서 그 중 최적의 케이스를 최적값으로 간주하자.

# 5. Knapsack

## 아이디어

- try1 : 가치의 총합을 계산하고 최대값을 이 값으로 초기화.
  선택된 아이템들을 솔루션으로 기억.
- try2 : 가치의 총합을 계산하고 최대값과 비교하여, 최대값보다 가치의 총합이 크면,
  가치의 총합을 최대값으로 저장
  선택된 아이템들을 솔루션으로 기억.
- 충분히 반복
- 최대값과 선택된 아이템 반환

# 5. Knapsack

## 의사코드

```
def findBestCaseMontecarlo(items, max_weight, tries):
    max_value = 0
    best_chosen_items = []
    best_weight = 0
    for i in range(tries):
        (value, weight, chosen_items) = select_items(items, max_weight)
        if (value > max_value) :
            max_value = value
            best_weight = weight
            best_chosen_items = list(chosen_items)
    return (max_value, best_weight, best_chosen_items)
```

# 5. Knapsack

## 의사코드

```
def select_items(items, max_weight):
    value = 0; weight = 0
    chosen_items = []
    for item in items:
        if (weight + item.weight <= max_weight) :
            roll = randint(1, 100)
            if (roll % 2 == 0):
                chosen_items.append(item.name)
                weight += item.weight
                value += item.value
    return (value, weight, chosen_item)
```

# 5. Knapsack

## 파이썬코드

```
import random

class Item(object):

    def __init__(self, name, value, weight):

        self.name = name

        self.value = value

        self.weight = weight
```

# 5. Knapsack

## 파이썬코드

```
class Knapsack(object):
    def __init__(self, names, values, weights, max_weight):          #아이템들을생성함
        self.items = []
        self.max_weight = max_weight
        for i in range(len(names)) :
            item = Item(names[i], values[i], weights[i])
            self.items.append(item)
```

# 5. Knapsack

## 파이썬코드

```
class Knapsack(object):
    def findBestCaseMontecarlo(self, tries=10000):
        max_value = 0
        best_chosen_items = []
        best_weight = 0
```

# 5. Knapsack

## 파이썬코드

```
for i in range(tries):
        (value, weight, chosen_items) = self.select_items()
        if (value > max_value) :
            max_value = value
            best_weight = weight
            best_chosen_items = list(chosen_items)
 return (max_value, best_weight, best_chosen_items)
```

# 5. Knapsack

## 파이썬코드

```
class Knapsack(object):
    ...
    def select_items(self):
        value = 0; weight = 0
        chosen_items = []
```

# 5. Knapsack

## 파이썬코드

```
for item in self.items:
    if (weight + item.weight <= self.max_weight) :
        roll = random.randint(1, 100)
        if (roll % 2 == 0):
            chosen_items.append(item.name)
            weight += item.weight
            value += item.value
return (value, weight, chosen_items)
```

# 5. Knapsack

## 파이썬코드

```
names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
values = [10, 30, 20, 14, 23, 11, 15, 18]
weights = [5, 8, 3, 7, 9, 2, 6, 1]
max_weight = 20
knapsack = Knapsack(names, values, weights, max_weight)
(value, weight, items) = knapsack.findBestCaseMontecarlo(100)
print(value, weight, items)
```

# 5. Knapsack

## 파이썬 코드 실행

```
In [114]: runfile('D:/data/
lecture/
파이썬으로배우는자료구조와알고리즘/
code/알고리즘/untitled1.py',
wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/
code/알고리즘')
94 20 ['B', 'C', 'F', 'G', 'H']
```

# 정리하기

- 몬테카를로 시뮬레이션의 개념
- 다양한 주사위 문제 해결
- 원주율 및 적분 계산
- knapsack 문제 해결

08강

다음 시간 안내▶▶▶

# 자료구조 개요 및 연결리스트