Brute-force Algorithm
- 반복을 활용 -

Brute-force

- 개요
  - 답을 찾기 위해, 모든 경우를 전부 확인하거나 수행하는 단순한 방법
    - 주로 반복 기법 활용
    - 컴퓨터의 빠른 성능을 활용
- 예
  - 1-100까지의 합계를 구하는 알고리즘
    - sum = 0 + 1 + 2 + … + 100
    - 반복문으로 구현
  - 9^n을 구하는 알고리즘
    - result = 1 * 9 * 9 * … * 9
  - knapsack 문제를 푸는 알고리즘

Brute-force

- 특성
  - easy to design and implement
  - 광범위한 문제에 적용 가능한 단순한 알고리즘
  - 입력(데이터)의 크기가 작은 경우 효과적이지만, 반대의 경우 계산 오버헤드가 클 수 있음
    - 낮은 성능
  - 더 효율적인 알고리즘을 통해 성능 개선 가능

문제0

- 1부터 자연수 N까지의 합계를 구하시오.

- 의사코드

문제0

- 1부터 자연수 N까지의 합계를 구하시오.

- 의사코드

```
sum = 0
for i in 1-N {
        sum += i
}
```

python - for 반복문

- for 변수 in 리스트
  - 리스트의 각 원소를 변수에 대입하며 반복 실행
  - 리스트의 원소의 개수만큼 for 문이 반복됨

python - range() 함수

- range(start, stop, step)
  - start 부터 stop-1 까지 step 증감
  - range(1, 7 1)
    - [1, 2, 3, 4, 5, 6] 반환
  - range(5, 1, -1)
    - [5, 4, 3, 2] 반환
  - start가 생략되면 0부터 시작
  - step이 생략되면 1씩 증가
  - range(3)
    - [0, 1, 2] 반환

문제0 - 방법1

- 파이썬 코드

python - 함수 정의 및 호출

- 함수 정의 (def 키워드)
  - def func_name(param) :
    - 인자(param)는 여러 개 있을 수 있고 , 로 구분. 없을 수도 있음
    - 인자의 자료형 없음
    - 함수는 여러 값을 반환할 수 있음. 각 값은 , 로 구분함

함수 정의 및 호출

- 함수 호출
  - 함수 이름을 사용하여 호출
  - 인자로는 변수 또는 상수를 전달
  - 변수를 사용하여 return 값을 전달받을 수 있음
  - sum = cal_sum(10)
  - print(cal_sum(10))

문제0 – 방법2

- 파이썬 코드

```
print(cal_sum(10))
print(cal_sum(100))
```

문제1 (최대,최소값)

- N개의 숫자를 저장한 파이썬 리스트에서 가장 작은 값을 출력하시오.
- 의사코드

문제1 (최대,최소값)

- 의사코드

```
min = list[0]
for item in list {
        if (item < min) {
                min = item
        }
}
print(min)
```

파이썬 리스트

- 리스트 정의
  - []를 사용함
  - *mylist = []*
    - 비어 있는 리스트 정의
  - *mylist2 = [0, 1, 2]*
    - 3개의 원소를 갖는 리스트 정의

파이썬 리스트

- 리스트 인덱싱
  - 리스트 이름 뒤에 []를 사용함
  - mylist2[0]
    - 첫번째 원소
  - mylist2[1]
    - 두번째 원소

문제1 (최대,최소값)

- 파이썬 코드

```
...


mylist = [3, 5, 7, 2, 9, 11, 2, 3, 8]
min = get_min(mylist)
print(min)
```

시간 복잡도?

문제2 (약수의 개수)

- 자연수 N의 약수의 개수를 출력하시오.

- 의사코드

문제2 (약수의 개수)

- 의사코드

```
count = 0
for i in (1 ~ num) {
        if (num % i == 0) {
                count++
        }
}
return count
```

문제2 (약수의 개수)

- 파이썬 코드

```
...


print(count_divisors(10))
```

시간 복잡도?

문제3 (선형탐색)

- N개의 숫자를 저장한 파이썬 리스트를 검색하여, 특정 숫자가 발견되면 found, 없으면 not found를 출력하시오.

- 의사코드

문제3 (선형탐색)

- N개의 숫자를 저장한 파이썬 리스트를 검색하여, 특정 숫자가 발견되면 found, 없으면 not found를 출력하시오.

- 의사코드

```
for item in 리스트 {
        if (item is equal to search) {
                print( “found” )
        }
}
```

불충분!!

문제3 (선형탐색)

- N개의 숫자를 저장한 파이썬 리스트를 검색하여, 특정 숫자가 발견되면 found, 없으면 not found를 출력하시오.

- 의사코드

```
is_found = 0
for item in 리스트 {
        if (item is equal to search) {
                print( “found” )
                is_found = 1
                break
        }
}
if (is_found is equal to 0) {
        print( “not found” )
}
```

문제3 (선형탐색)

- 파이썬 코드

시간 복잡도?

문제3 (선형탐색)

- 파이썬 코드

```
mylist = [3, 5, 7, 2, 9, 11, 2, 3, 8]
search_num(mylist, 2)
search_num(mylist, 15)
```

str() 함수와 문자열 연산

- str()
  - 문자열로 변환
  - str(10)
    - 숫자 10이 문자열 “10” 으로 변환됨
- 문자열 이어 붙이기
  - + 연산자 사용함
  - print(“my” + “name”)
    - myname 출력됨

문제4 (선형탐색)

- N개의 숫자를 저장한 파이썬 리스트를 검색하여, 특정 숫자가 검색하여 발견된 인덱스를 모두 출력하시오. 만약 하나도 없다면 "not found"를 출력하시오.

- 의사코드

문제4 (선형탐색)

- 의사코드

```
result_list is empty
for index in list_indices {
        if (list[index] is equal to search) {
                insert index to result_list
        }
}
if (result_list is empty) {
        print( “not found” )
} else {
        print(result_list)
}
```

python - len() 함수

- len() 함수
  - 인자가 리스트이면 리스트의 크기를 반환
  - 인자가 문자열이면 문자열의 길이를 반환
  - mylist = [1, 2, 3]
  - len(mylist)
    - 3을 반환함

리스트 관련 메서드 – append()

- append(item)
  - 리스트의 메서드로서, 리스트의 마지막에 item을 추가함
  - mylist = [1, 2, 3]
  - mylist.append(4)
    - mylist는 [1, 2, 3, 4]로 바뀜

문제4 (선형탐색) - 방법1

- 파이썬 코드1 (출력까지 처리)

시간 복잡도?

문제4 (선형탐색) – 방법1

- 파이썬 코드1

```
mylist = [3, 5, 7, 2, 9, 11, 2, 3, 8]
get_indices(mylist, 2)
get_indices(mylist, 15)
```

문제4 (선형탐색) – 방법2

- 파이썬 코드2 (출력은 하지 않고 탐색된 결과를 리스트로 반환)

문제4 (선형탐색) – 방법2

- 파이썬 코드2

```
mylist = [3, 5, 7, 2, 9, 11, 2, 3, 8]

result = get_indices(mylist, 2)
if (len(result) == 0):
    print(": Not found")
else:
    print(result)
```

문제5 (선형탐색)

- 길이가 n인 문자열 T와 길이가 m인 문자열 P가 있다. T에서 가장 먼저 나타나는 P의 위치(인덱스)를 출력하시오. 없으면 -1을 출력하시오.

how are you doing?

are

how ow w a ar are

are are are are are

문제5 (선형탐색)

- 의사코드

```
len1 = len(string)
len2 = len(pattern)

for start in 0 ~ (len1-len2)) {
        substring = string[start, start + len2 - 1]
        if (substring equals to pattern)
                return start
}
return (-1)
```

문자열 인덱싱 및 슬라이싱

- 문자열 인덱싱
  - []를 인덱스로 사용함
  - mystr = “seoultech”
  - print(mystr[0])
    - s 출력됨

문자열 인덱싱 및 슬라이싱

- 문자열 슬라이싱
  - [start, stop + 1, step]를 사용함
  - mystr = “seoultech”
  - print(mystr[0:3:1])
    - “seo” 출력

문제5 (선형탐색)

- 파이썬 코드

시간 복잡도?

문제5 (선형탐색)

- 파이썬 코드

```
string = "how are you doing?"
pattern = "are"
print(find_pattern(string, pattern))
```

문제6 (선택정렬)

- N개의 숫자를 저장한 파이썬 리스트를 내림차순으로 정렬하시오.

- 아이디어
  - 가장 큰 값을 찾아서 원래 리스트에서 제거하고 새로운 리스트에 추가.
  - 이를 원래 리스트가 빌 때까지 반복

문제6 (선택정렬)

- 의사코드 예시

```
result is empty
while (list is not empty) {
        find max_number in list
        remove max_number from list
        append max_number to result
}
return result
```

리스트 관련 메서드 – pop()

- pop(index)
  - 리스트의 메서드로서, 기본적으로 리스트의 마지막 item을 반환함 (리스트가 수정됨)
  - mylist = [1, 2, 3]
  - print(mylist.pop())
    - 3 출력됨
    - mylist는 [1, 2]로 바뀜
  - print(mylist.pop(0))
    - 1 출력됨
    - mylist는 [2]로 바뀜

문제6 (선택정렬)

- 파이썬 코드 예시

시간 복잡도?

문제6 (선택정렬)

- 파이썬 코드 예시

시간 복잡도?

문제6 (선택정렬)

- 파이썬 코드 예시

```
mylist = [3, 5, 7, 2, 9, 11, 2, 3, 8]
result = selection_sort(mylist)
print(result)
```

문제7 (최근접 쌍)

- 2차원 평면 상에 n개의 점이 있다. 가장 인접한 쌍의 거리를 구하라.

- 아이디어
  - 모든 두 점 간의 거리를 구하여 최소값을 찾으라.

문제7 (최근접 쌍)

- 의사코드

```
-- 모든 점의 좌표가 리스트로 전달 --

min = INF
len = length(point_array)
for idx1 in 0 ~ len-2 {
        for idx2 in idx1+1 ~ len-1 {
                dist = cal_dist(point_array[idx1], point_array[idx2])
                if (min > dist)
                        min = dist
        }
}
return min
```

파이썬 – “infinite”

- 최소값 구할 때 유용하게 사용할 수 있는 “inf”
  - 방법1
    - import math
    - min1 = math.inf
  - 방법2
    - min2 = float(“inf”)
  - 테스트
    - print(min1, min2)
      - 모두 inf를 값으로 출력함.
  - 값이 무한이므로 최소값 구할 때 min의 초기값으로 활용

파이썬 – 제곱근 구하기

- 제곱근 구하기
- 방법1
  - import math
  - res1 = math.sqrt(4)
- 방법2
  - res2 = 4 ** 0.5
- 테스트
  - print(res1, res2)
    - 모두 2.0을 값으로 출력함.

파이썬 – tuple

- 리스트와 유사하게 여러 원소값을 저장
- ()를 사용함
- 개별 원소를 접근하는 방법은 리스트와 동일함
  - [] 사용.
- tuple은 수정 불가함.
  - mytuple = (3, 4, 5)
  - print(mytuple[0])
    - 3 출력됨
  - mytuple[0] = 5
    - 불가. 값을 수정할 수 없음.

문제7 (최근접 쌍)

- 파이썬 코드

```
import math
```

문제7 (최근접 쌍)

- 파이썬 코드

시간 복잡도?

문제7 (최근접 쌍)

- 파이썬 코드

```
point_list = [(2,3), (3,5), (8,10), (11,-1)]
print(closest_pair(point_list))
```

문제8 (knapsack)

- 각각 무게가 $weight_i$, 가치가 val_i인 n개의 물건들을 배낭에 넣으려고 한다. 배낭에 넣을 수 있는 최대 무게는 W이다. 배낭에 넣은 물건들의 가치를 최대로 하는 물건의 조합을 구하시오.

- 아이디어
  - 모든 경우의 수에 대해 가치와 무게를 모두 구하자.
  - 일단 최대 무게를 초과하는 경우는 버린다.
  - 최대 무게 이하인 경우들 중에서 가장 가치가 높은 경우를 선택한다.

  - 모든 경우를 어떻게 표현할까?

문제8 (knapsack)

- 아이디어 (3개의 물건으로 가정)
  - 모든 경우를 어떻게 표현할까?
  - 총 경우의 수
    - 2^3-1 (모두 선택하지 않는 경우는 제외함)
      - XXO, XOX, XOO, OXX, OXO, OOX, OOO
  - 어떻게 반복문으로 설계할까?

문제8 (knapsack)

- 아이디어 (3개의 물건으로 가정)
  - 어떻게 반복문으로 설계할까?
    - 물건을 선택하지 않은 경우를 0으로 표현하고, 물건을 선택한 경우를 1로 표현하면
      - XX0, X0X, X00, 0XX, 0X0, 00X, 000 => 001, 010, 011, 100, 101, 110, 111
      - => 이진수로 해석하면 => 1 ~ (2^3-1)
      - => for 반복문 등을 1 ~ (2^3-1) 까지 반복하면 됨.
      - 각 비트가 0이면 물건을 선택하지 않았고 1은 물건을 선택했다고 해석하여 무게와 가치를 계산!!
      - 가령 3은 011 이므로 첫째 물건 선택, 둘째 물건 선택, 셋째 물건 제외한 경우를 의미.

문제8 (knapsack)

- 의사코드 (물건들의 가치와 무게를 각각 리스트(values, weights)로 전달받는다고 가정)

```
max_case = 0
max_value = 0
total_count = power(2, n) - 1
for case in (1 ~ total_count) {
        value = 0, weight = 0
        for bitnum in (0 ~ n-1) {
                if (bitnum bit of case is 1) {
                        // 해당 물건 선택
                        value += values[bitnum]
                        weight += weight[bitnum]
                }
        }
        if (value > max_value and weight <= max_weight) {
                max_case = case
                max_value = value
        }
}
return (case, max_value)
```

Class 정의

- 클래스는 멤버 변수와 메서드로 구성됨
  - 변수는 데이터를 저장하고, 메서드는 인터페이스 역할을 하는 함수임
  - 가령 리스트 클래스의 경우, *append()*, *pop()* 등은 리스트에 속한 메서드
  - 생성자는 클래스 객체가 생성될 때 호출되는 특수 메서드
    - 일반적으로 멤버 변수 초기화 역할
    - *__init__(self, …)*

Class 정의

- 클래스 정의
  - class Myclass(object) :
    - ()안에는 Myclass가 상속하는 클래스를 기술
    - object는 파이썬에서 최상위 기본 클래스
- 생성자
  - __init__(self, …) :
    - 모든 메서드의 첫번째 인자는 self, 이후 나머지 필요한 인자를 기술함
  - self.name = name
    - self는 현재 대상 클래스 객체를 의미함.. 따라서 self.name은 현재 객체의 멤버 변수 name을 의미

문제8 (knapsack)

- 파이썬 코드 – 1단계: item 클래스 정의
  - Item 클래스는 각각의 아이템을 표현
  - 아이템의 이름, 가치, 무게를 저장
  - 생성자만 있으며 여타 아이템 관련 메서드는 없음.

```
class Item(object):
    def __init__(self, name, value, weight):
        self.name = name
        self.value = value
        self.weight = weight
```

문제8 (knapsack)

- 파이썬 코드 – 2단계: Knapsack 클래스 정의
  - 멤버변수
    - 아이템들을 저장한 items 리스트. self.items[]
    - 허용 가능한 무게의 최대값. self.max_weight
    - 가치를 최대로 하는 조합. self.opt_case
    - 가치를 최대로 할 때의 가치. self.max_value
  - 메소드
    - 생성자
    - printItems(): 현재 생성된 모든 item들의 정보를 출력

문제8 (knapsack)

- 파이썬 코드 – 2단계: Knapsack 클래스 정의

```
class Knapsack(object):
    def __init__(self, names, values, weights, max_weight):
        self.items = []
        self.max_weight = max_weight
        self.max_value = 0
        self.opt_case = 0

        for i in range(len(names)) :
            item = Item(names[i], values[i], weights[i])          # Item 객체 생성
            self.items.append(item)
```

문제8 (knapsack)

- 파이썬 코드 – 2단계: Knapsack 클래스 정의

```
def printItems(self):
    for item in self.items:
        print(item.name, item.value, item.weight)
```

문제8 (knapsack)

- 파이썬 코드 – 3단계: Knapsack 클래스 객체 생성하여 아이템들 출력 테스트

```
names = ['0', '1', '2', '3', '4']
values = [10, 30, 20, 14, 23]
weights = [5, 8, 3, 7, 9]
max_weight = 20

knapsack = Knapsack(names, values, weights, max_weight)
knapsack.printItems()
```

문제8 (knapsack)

- 파이썬 코드 – 4단계: Knapsack 클래스에 최대 가치 구하는 메소드 추가

```
def findOptCase(self):
    self.max_value = 0
    item_count = len(self.items)
    case_count = 2**item_count - 1
```

문제8 (knapsack)

- 파이썬 코드 – 4단계: Knapsack 클래스에 최대 가치 구하는 메소드 추가

```
def findOptCase(self):
    ...
    for case in range(1, case_count+1): # 각 케이스에 대해 반복
        value = 0
        weight = 0
        for bitnum in range(item_count): # 해당 케이스에 대해 가치와 무게 계산
            target_bit_num = 2**bitnum
            if (case & target_bit_num == target_bit_num):   # bitnum에 해당하는 비트 자리가 1
                item = self.items[bitnum]
                value += item.value
                weight += item.weight
```

문제8 (knapsack)

- 파이썬 코드 – 4단계: Knapsack 클래스에 최대 가치 구하는 메소드 추가

```
def findOptCase(self):
    ...
    for case in range(1, case_count+1):
        ...

        if (value > self.max_value and weight <= self.max_weight):
            self.opt_case = case
            self.max_value = value

    chosen_items = self.decodeCase(self.opt_case)
    return (chosen_items, self.max_value)
```

문제8 (knapsack)

- 파이썬 코드 – 4단계: Knapsack 클래스에 최대 가치 구하는 메소드 추가

```
def decodeCase(self, case):
    item_count = len(self.items)
    chosen_items = []

    for index in range(item_count):
        target_bit_num = 2**index
        if (case & target_bit_num == target_bit_num):  # bitnum에 해당하는 비트 자리가 1
            item = self.items[index]
            chosen_items.append(item.name)

    return chosen_items
```

문제8 (knapsack)

- 파이썬 코드 – 5단계: Knapsack 최대 가치 구하는 테스트 코드

```
names = ['0', '1', '2', '3', '4']
values = [10, 30, 20, 14, 23]
weights = [5, 8, 3, 7, 9]
max_weight = 20

knapsack = Knapsack(names, values, weights, max_weight)
(chosen_items, max_value) = knapsack.findOptCase()

print(chosen_items, max_value)
```

Summary

- 반복 구조를 활용한 brute-force 알고리즘
  - 최소/최대값 찾기
  - 선형탐색
  - 선택정렬
  - 최근접 쌍
  - knapsack
  - …

homework candidate

- 최근접 쌍 문제에서, 최근접 쌍의 거리와 각 좌표를 함께 출력하도록 프로그램을 작성하시오.

homework candidate

- knapsack 문제에서, 무게 제한 조건을 만족하는 모든 경우에 대해 선택된 아이템들과 가치를 출력하는 메서드를 추가하시오.
  - def calValuesAllCases(self):

homework candidate

- knapsack 문제에서, random 함수를 이용하여 임의의 개수의 아이템을 생성하는 함수를 추가하라.
  - createItems(item_count, max_value, max_weight)
    - item_count 개수 만큼의 item들을 생성.
    - 이들의 value와 weight는 1-max_value, 1-max_weight 범위의 임의의 자연수 (random 라이브러리 사용)
  - 10개의 아이템, 20개의 아이템, 50개의 아이템을 생성하고 최적의 해를 구하는 수행 시간을 각각 측정