Intro to Computer Science

Local Laboratory

*** Udacity – Intro to Computer Science**

Introduction

Unit 6: How to Have Infinite Power

Quiz: Long Words

영어에서 가장 긴 단어는 무엇일지 고르시오.

- ❑ honorificabilitudinitatibus
- ❑ antidisestablishmentarianism
- ❑ hippopotomonstrosesquippedaliophobia
- ❑ pneumonoultramicroscopicsilicovolcanoconiosis
- ❑ None of the above

Counter

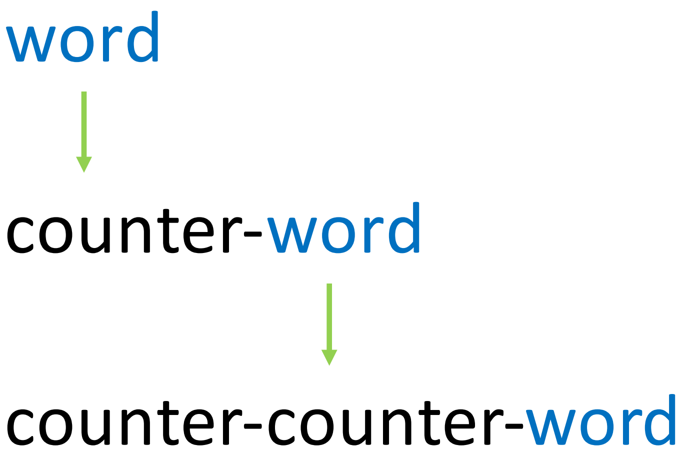
word -> counter-word

intelligence
counter-intelligence
counter-counter-intelligence
counter-counter-counter-intelligence
hippopotomonstrosesquippedaliophobia
fear of long words
counter-hippopotomonstrosesquippedaliophobia
counter-counter-hippopotomonstrosesquippedaliophobia

Quiz: Counter Quiz

오직 아래의 하나의 규칙을 이용하여, word로 시작하여
얼마나 많은 단어들을 만들 수 있는지 고르시오.
규칙 : **word -> conter-word**

- ❑ None
- ❑ 1
- ❑ 2
- ❑ Infinitely Many



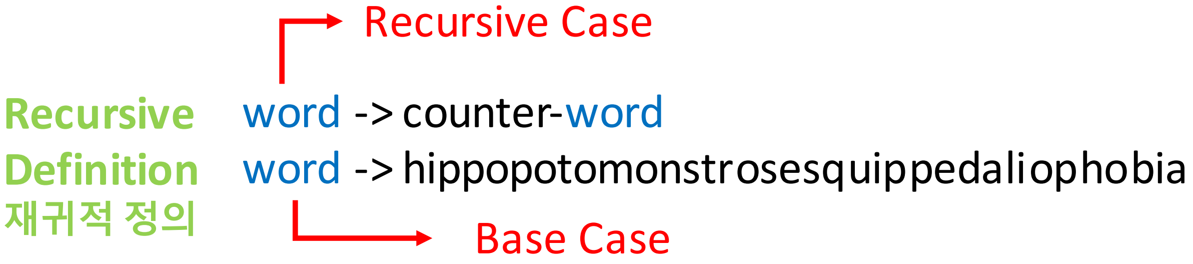
word

counter-word

counter-counter-word

Quiz: Expanding Our Grammar

아래의 2개의 규칙만을 이용하여, word로 시작하여
얼마나 많은 단어들을 만들 수 있는지 고르시오.



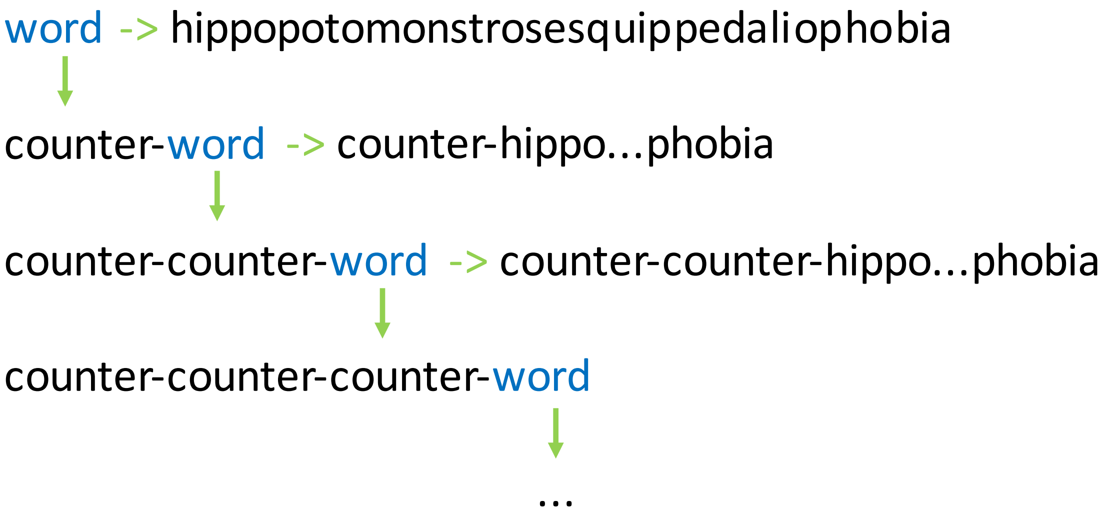
Recursive Case

Recursive Definition
재귀적 정의

word -> counter-word
word -> hippopotomonstrosesquippedaliophobia

Base Case

- ❑ None
- ❑ 1
- ❑ 2
- ❑ Infinitely Many



word -> hippopotomonstrosesquippedaliophobia

↓

counter-word -> counter-hippo...phobia

↓

counter-counter-word -> counter-counter-hippo...phobia

↓

counter-counter-counter-word

↓

...

Recursive Definitions

Two parts:

1. Base case – a starting point
    - Not defined in terms of itself
    - Smallest input – already know the answer

2. Recursive case
    - Defined in terms of “smaller” version of itself

Defining Procedures Recursively

$$\text{factorial}(n) = n * \underbrace{(n-1) * (n-2) * \ldots * 1}_{\text{factorial}(n-1)}$$

Base Case $\text{factorial}(0) = 1$

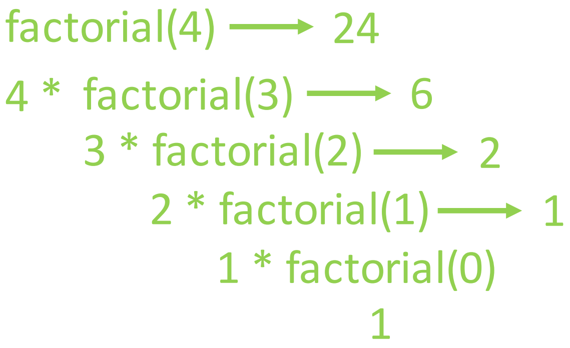
$\underset{n > 0}{\text{factorial}(n)} = n * \underline{\text{factorial}(n-1)}$ Recursive Case

Quiz: Recursive Factorial

0 또는 자연수를 입력받아서, 입력한 숫자에 대한 팩토리얼 값을 계산하여 리턴하는 factorial이라는 프로시져를 정의하시오.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

factorial(0) = 1
factorial(n) = n * factorial(n-1)



factorial(4) → 24
4 * factorial(3) → 6
3 * factorial(2) → 2
2 * factorial(1) → 1
1 * factorial(0)
1

Faster Fibonacci

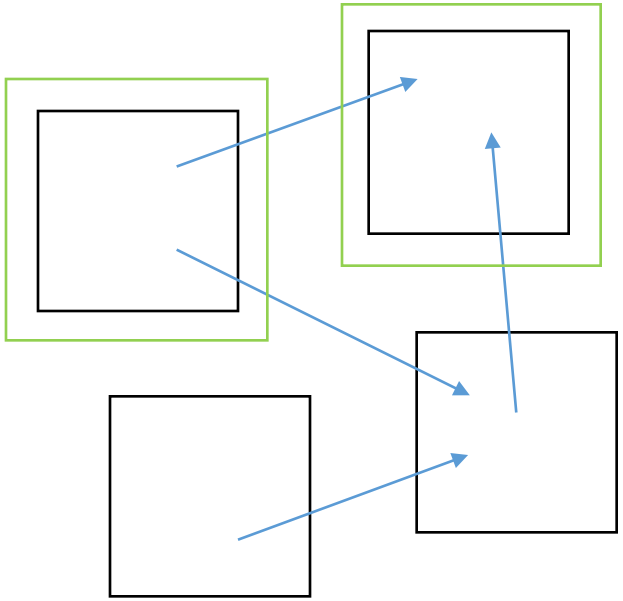
code

```
def fibonacci(n):
    current = 0
    after = 1
    for i in range(0, n):
        current, after = after, current + after

    return current

print(fibonacci(33))
print(fibonacci(36))
print(fibonacci(60))
```

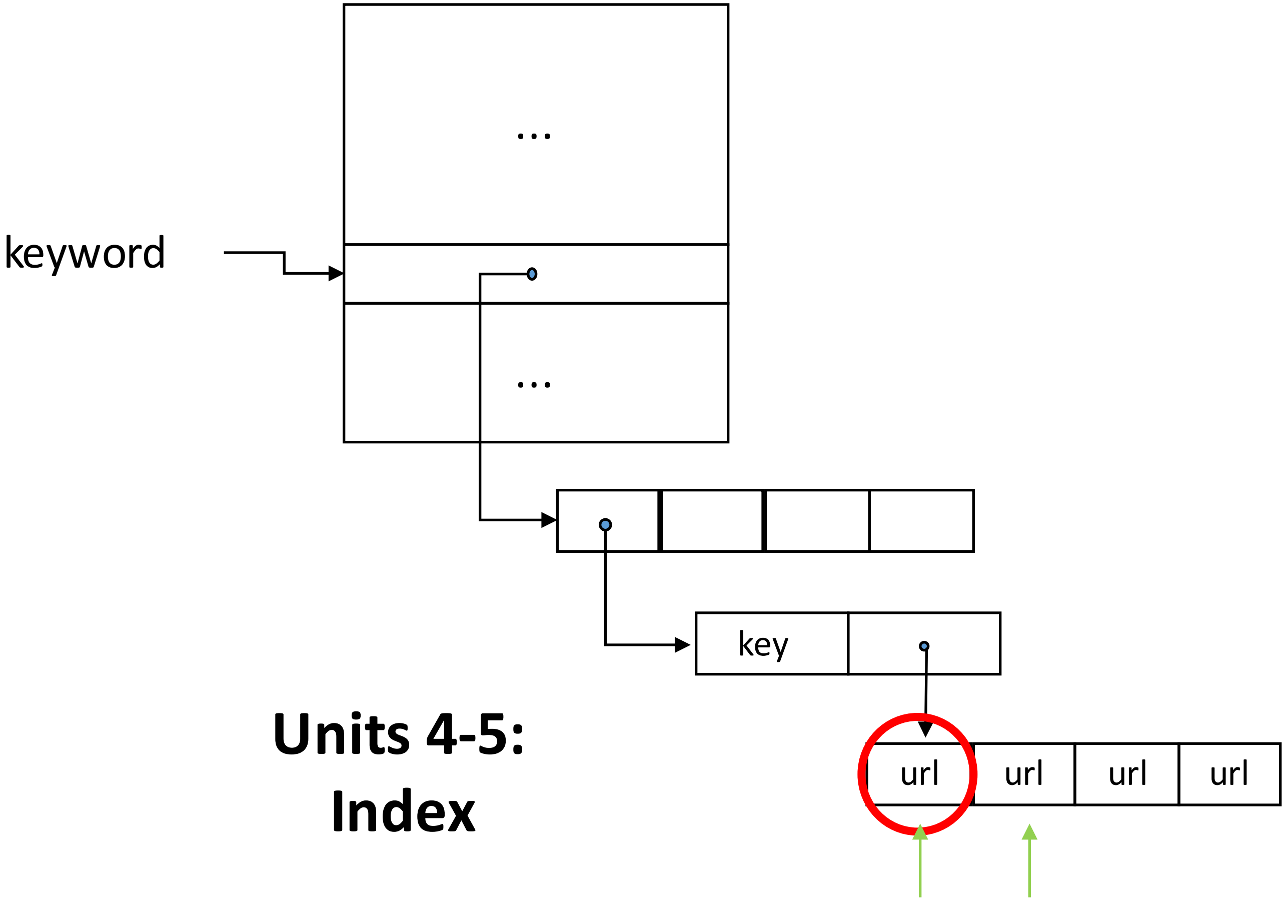
result

```
3524578
14930352
1548008755920
```

Ranking Web Pages

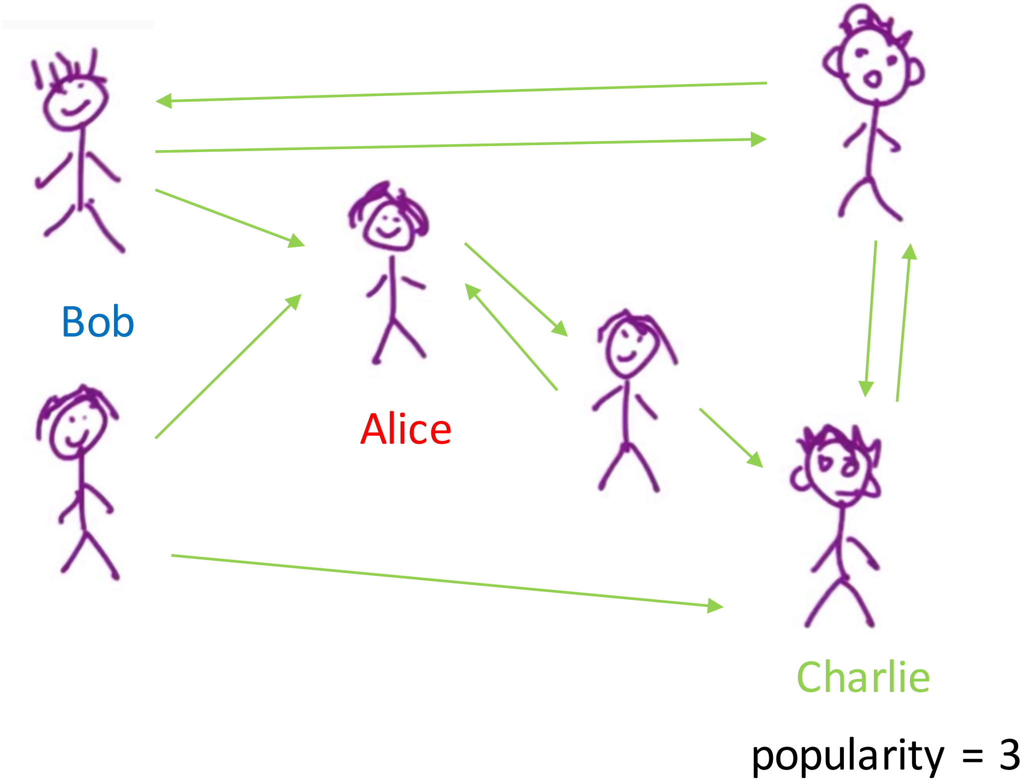


Units 1-3: Crawler



Units 4-5: Index

Popularity



popularity(p) = number of people who are friends with p

↓

$$\text{popularity(p)} = \sum_{\substack{f \in friends \\ of\ p}} popularity(f)$$

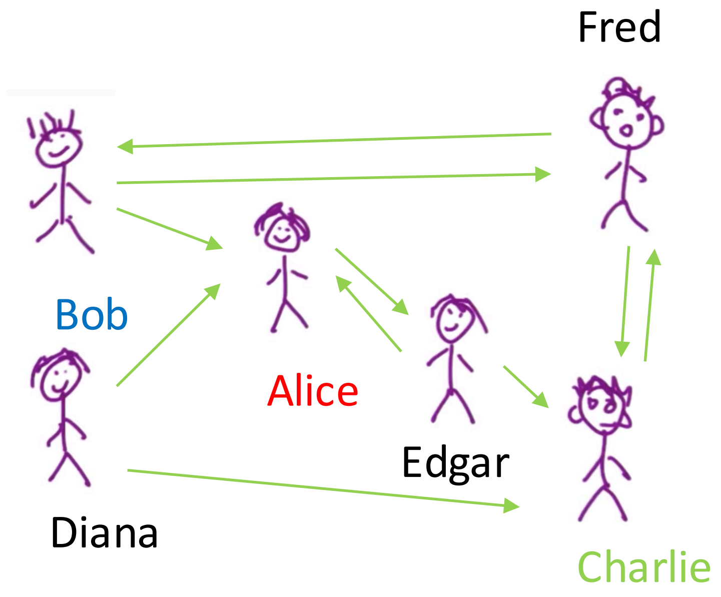
```
def popularity(p):
    score = 0
    for f in friends(p):
        score = score + popularity(f)
    return score
```

Quiz: Good Definitions

```
def popularity(p):
    score = 0
    for f in friends(p):
        score = score + popularity(f)
    return score
```

No base case!

circular definition!



위의 정의는 좋은 recursive defintion(재귀적인 정의)인가?

□ yes □ no

Quiz: Circular Definitions

factorial(0)
-> 1

```
def popularity(p):
    score = 0
    for f in friends(p):
        score = score + popularity(f)
    return score
```

No base case!

circular definition!

popularity('Alice') = 1

$$\text{popularity(p)} = \sum_{\substack{f\, \in friends \\ of\ p}} popularity(f)$$

```
def popularity(p):
    score = 0
    for f in friends(p):
        score = score + popularity(f)
    return score
```

if p == 'Alice':
return 1

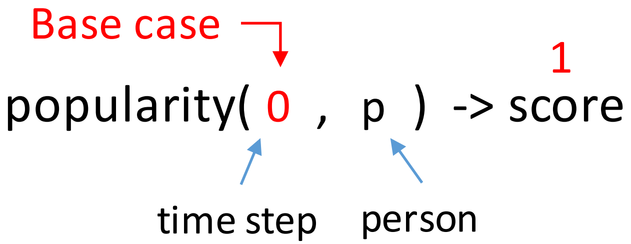
왼쪽과 같은 정의는 잘 동작하겠는가?

- □ Only if everyone is friends with 'Alice'.
- □ Only if no one is friends with 'Alice'.
- □ Only if there is a friendship path from everyone to 'Alice'.
- □ Only if there are no cylces in the graph.
- □ No.

Quiz: Relaxation

Relaxation Algorithm

```
-> start with a guess
   while not done:
       make the guess better
```



Base case

popularity(0 , p) -> score

1

time step

person

$$popularity(t,p) = \sum_{\substack{f \in friends \\ of\ p}} popularity(t-1,f)$$

```
def popularity(t, p):
    if t == 0:
        return 1
    else:
        score = 0
        for f in friends(p):
            score = score + popularity(t-1, f)
        return score
```

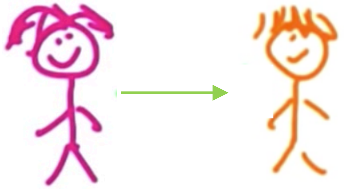
위의 정의는 좋은 recursive definition인가?

- □ Yes
- □ Only if people can’t be friend themselves.
- □ Only if everyone has at least one friend.
- □ Only if everyone is more popular than ‘Alice’

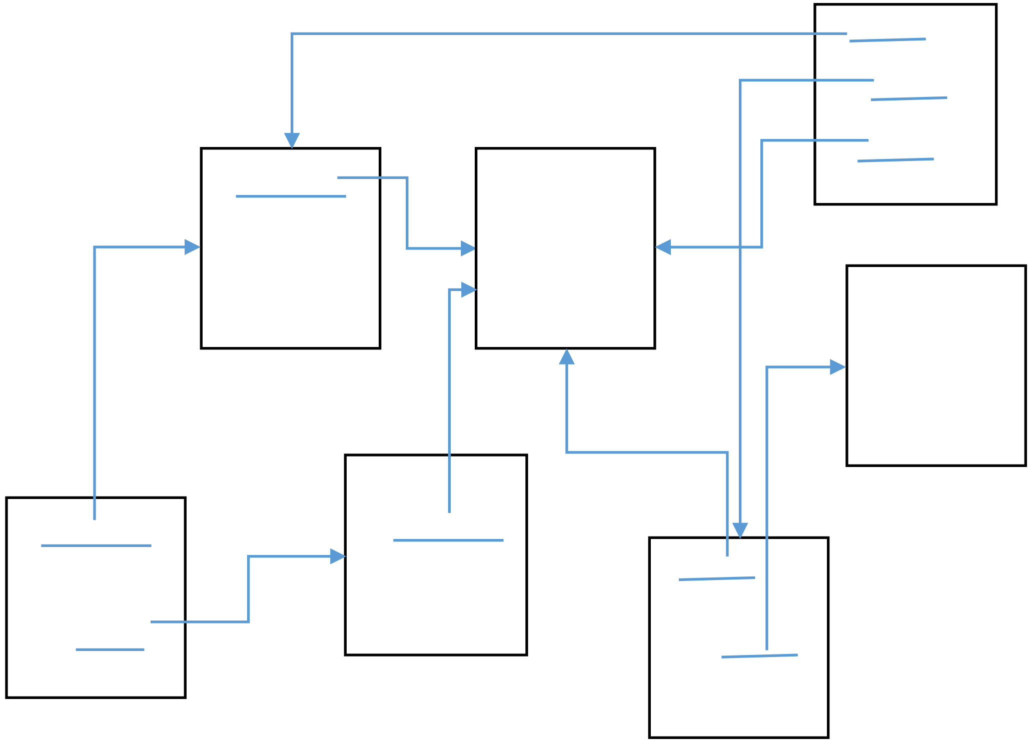
Page Rank

Ranking Web Pages

aka. Page Rank



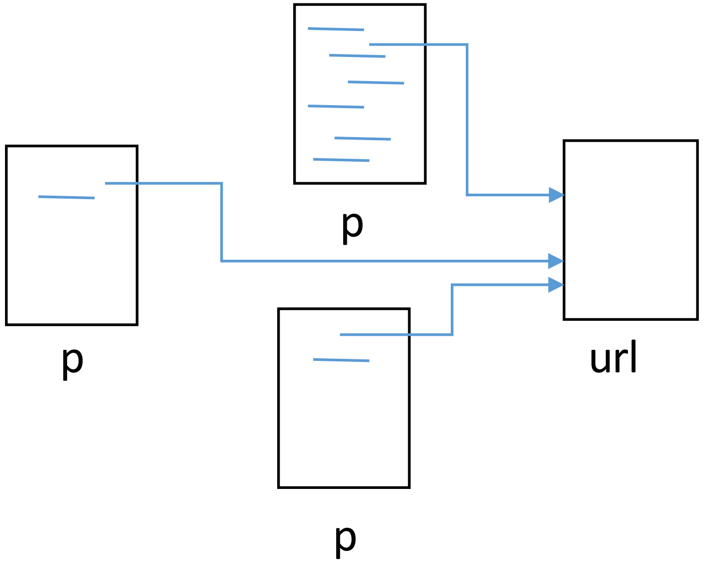
Freinds



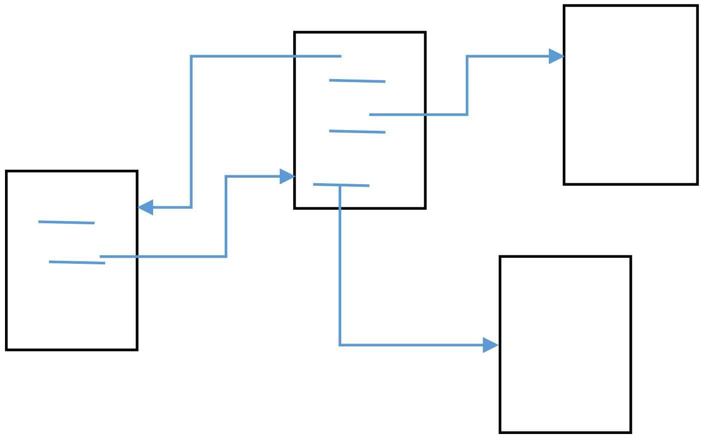
Page Rank

rank(0, url) -> 1

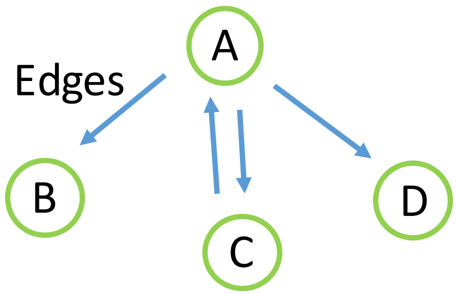
rank(t, url) -> $\sum_{p \in \text{inlinks[url]}}$ rank(t-1, p) **/ outlinks[p]**



~~Page Rank~~ Local Rank



Graph



Directed Graph

{ url: [pages it links to], ... }

{ 'A': ['B', 'C', 'D'],
'B': [ ],
'C': ['A'],
'D': [ ] }

Local Rank

Building the Link Graph

crawl_web(seed) -> index , graph

```
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []
    index = {}
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
            content = get_page(page)
            add_page_to_index(index, page, content)
            union(tocrawl, get_all_links(content))
            crawled.append(page)
    return index
```

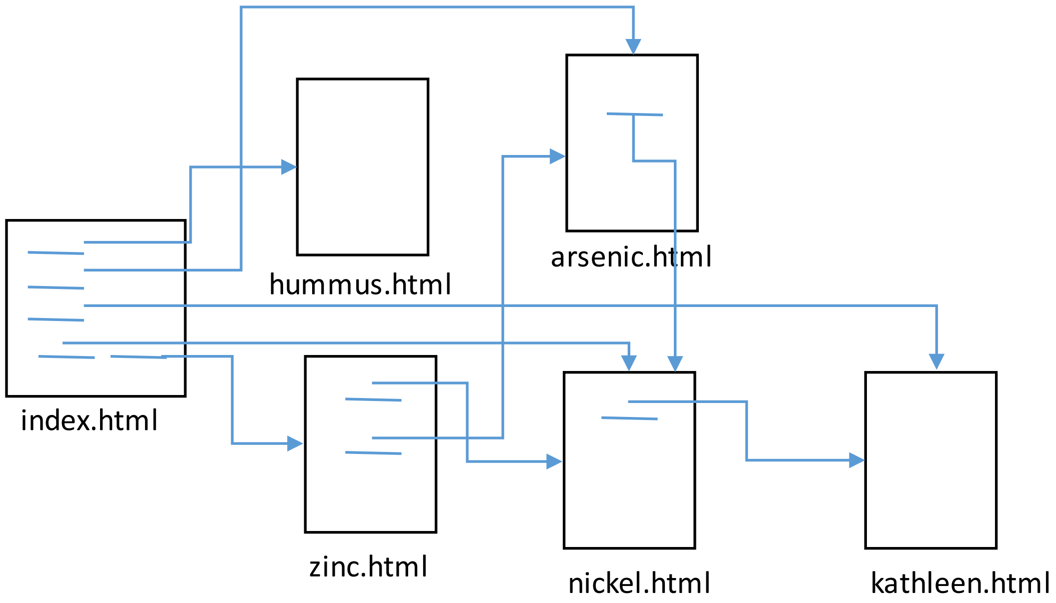
```
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []
    index = {}
    graph = {}
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
            content = get_page(page)
            add_page_to_index(index, page, content)
            outlinks = get_all_links(content)

            union(tocrawl, outlinks)
            crawled.append(page)

    return index, graph
```

Quiz: Implementing Local Rank

crawl_web 프로시저를 기존에 index만 리턴하는 것 대신 index와 graph를 리턴하도록 수정하시오.
그래프는 아래와 같은 entry들을 가지는 Dictionary 타입이어야 한다.

url: [url, url, url]

page pages that link to target



Implementing Local Rank

code

```
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []
    index = {}
    graph = {}
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
            content = get_page(page)
            add_page_to_index(index, page, content)
            outlinks = get_all_links(content)
            graph[page] = outlinks
            union(tocrawl, outlinks)
            crawled.append(page)

    return index, graph
```

result

```
/Users/lastland/anaconda3/bin/python
/Users/lastland/PycharmProjects/cs101_9/5_implementing_
urank.py['http://udacity.com/cs101x/urank/hummus.html',
'http://udacity.com/cs101x/urank/arsenic.html',
'http://udacity.com/cs101x/urank/kathleen.html',
'http://udacity.com/cs101x/urank/nickel.html',
'http://udacity.com/cs101x/urank/zinc.html']
```