

Intro to Computer Science

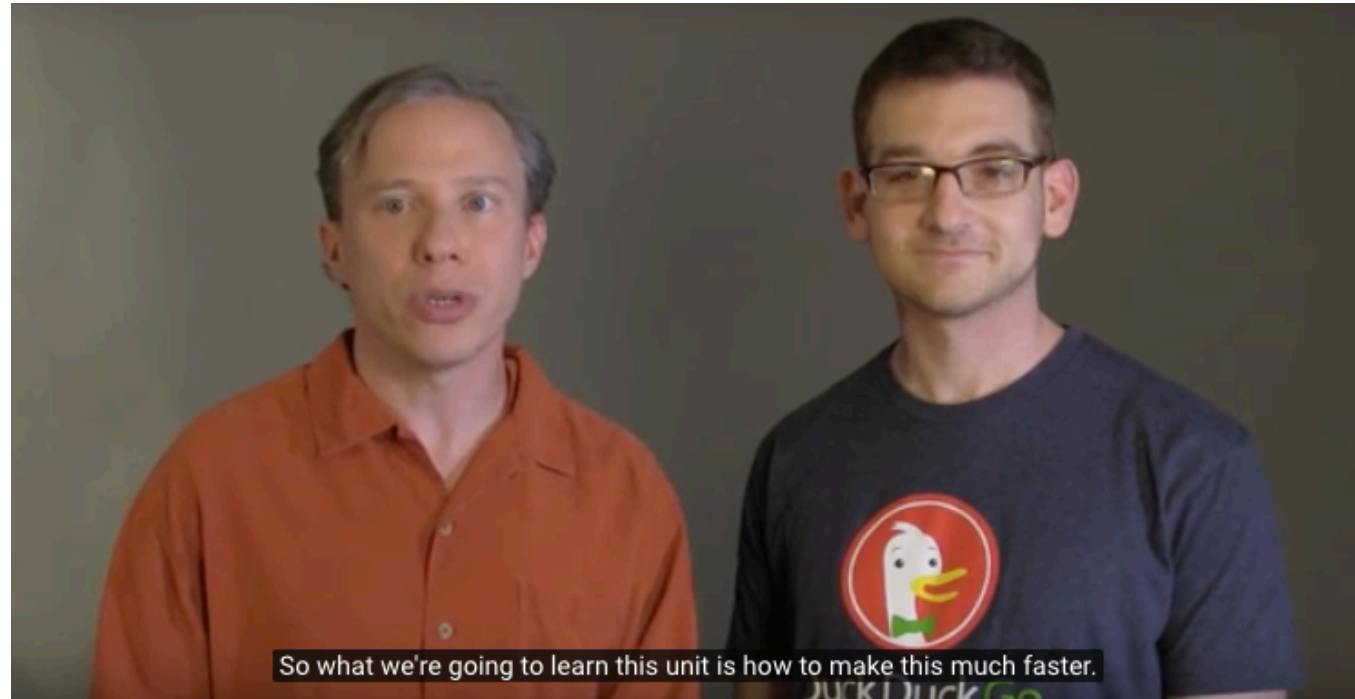
Local Laboratory

*** Udacity – Intro to Computer Science**

Introduction

Unit 5: How Programs Run

Making things Fast



Algorithm Analysis

Procedure

- well-defined sequence of steps that can be executed mechanically

기계적으로 실행될 수 있는
잘 정의된 단계들의 순서

Guaranteed to always finish
and produce correct result

항상 종료되고 정확한 결과를
생산하는 것을 보장함

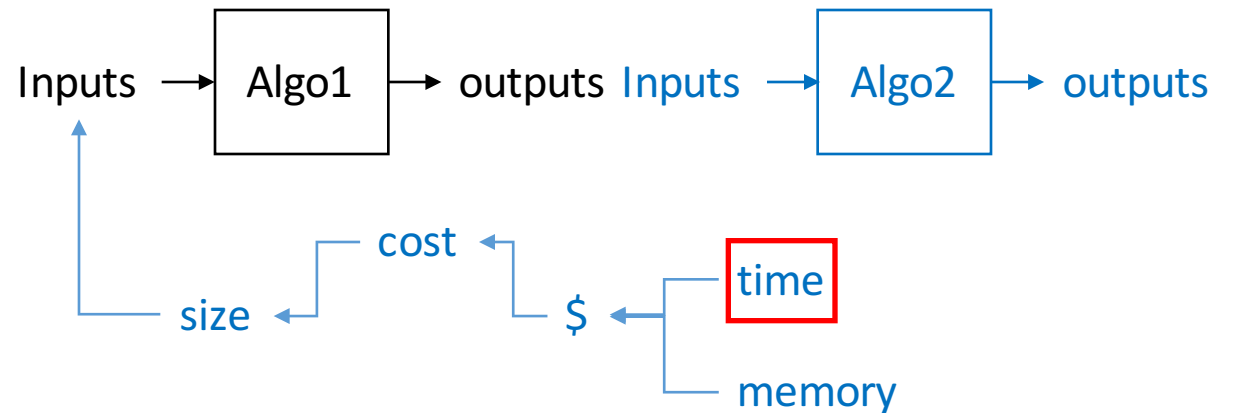
cost



\$25,000



\$10,000



Quiz: Worst Case

worst-case execution time

code

```
def add_to_index(index, keyword, url):  
    for entry in index:  
        if entry[0] == keyword:  
            entry[1].append(url)  
    return  
    index.append([keyword, [url]])  
  
def lookup(index, keyword):  
    for entry in index:  
        if entry[0] == keyword:  
            return entry[1]  
    return []
```

number of times
through the loop
depending on
`len(index)`

어떤 index에 대해 실행 속도 측면에서 최악의 경우(worst-case)를 가지는 keyword를 고르시오.

- ☐ `lookup(index,`
 `first word added)`
- ☐ `lookup(index,`
 `word that is not in index)`
- ☐ `lookup(index,`
 `last word added)`

Quiz: Fast Enough

우리가 만든 lookup 프로시저는 충분히 빠른가?

☐ Yes

☐ index에 얼마나 많은 키워드들이 저장되어있느냐에 따라 다르다.

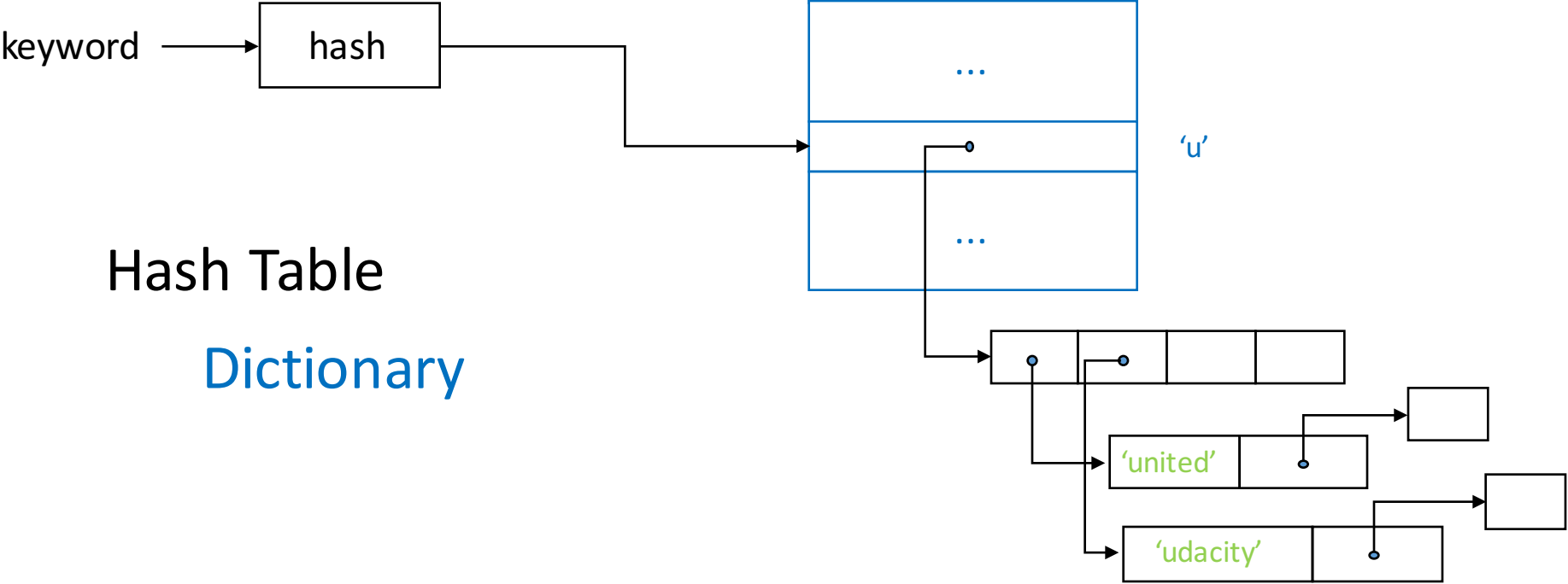
☐ index에 얼마나 많은 URL들이 저장되어 있느냐에 따라 다르다.

☐ lookup을 얼마나 많이 호출하느냐에 따라 다르다.

☐ No

Making Lookup Faster

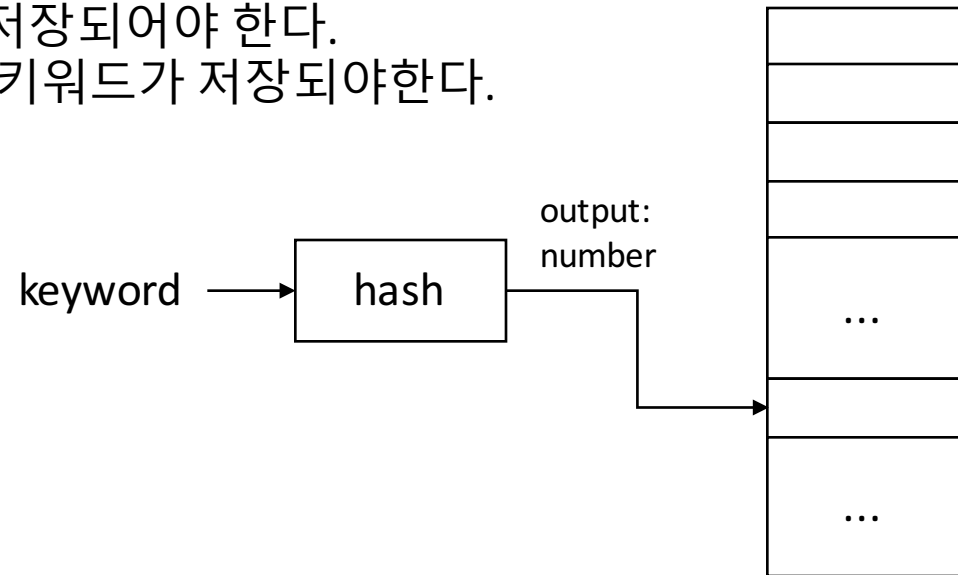
How can we make lookup faster?



Quiz: Hash Table

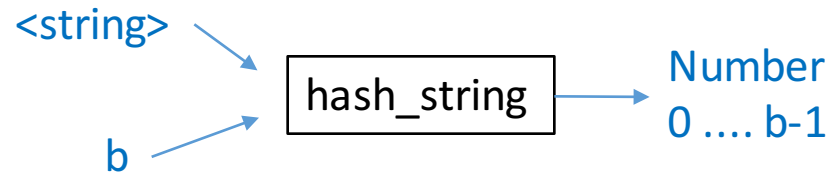
hash function을 구성할 경우, b 개의 bucket과 k 개의 keyword가 있다고 가정하자($k > b$).
hash function이 가져야하는 속성들은 어떤 것인지 고르시오.

- ☐ 0과 $k-1$ 사이의 특정한 숫자를 리턴해야 한다.
- ☐ 0과 $b-1$ 사이의 특정한 숫자를 리턴해야 한다.
- ☐ 0번째 버킷에 약 k/b 개의 키워드를 저장되어야 한다.
- ☐ $b-1$ 번째 버킷에 약 k/b 개의 키워드를 저장되어야 한다.
- ☐ 1번째 버킷보다 0번째 버킷에 더 많은 키워드가 저장되어야 한다.



Hash Function

Defining a Hash Function



`ord(<one-letter string>)` -> `<Number>`
`chr(<Number>)` -> `<one-letter string>`
`chr(ord('a'))` -> `'a'`

code

```
print(ord('a'))  
print(ord('A'))  
print(ord('B'))  
print(chr(ord('u')))  
print(ord('udacity'))
```

result

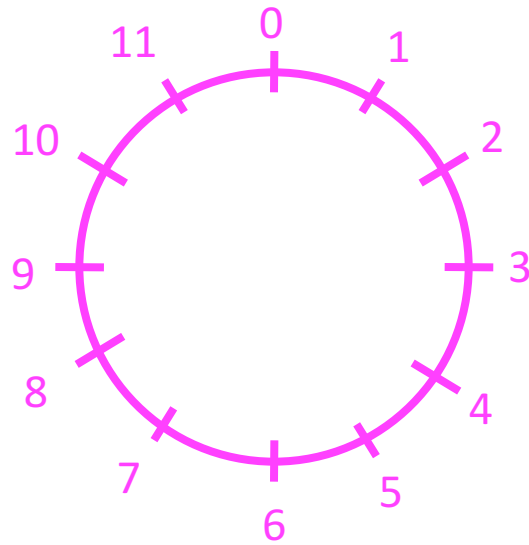
```
97  
65  
66  
u  
TypeError: ord() expected a character, but string of length 7 found
```


Modulus Operator

Modulus Operator

%

<Number> % <Modulus> -> <remainder>



14 % 12 -> 2

Quiz: Modulus Quiz

각각의 표현에 대한 값을 쓰시오.

a) $12\%3$

0

b) $\text{ord}('a') \% \text{ord}('a')$

0

c) $(\text{ord}('z')+3) \% \text{ord}('z')$

3

code

```
print((ord('z') + 3) % ord('z'))  
print(ord('z') + 3 % ord('z'))  
print(ord('z'))
```

result

```
3  
125  
122
```

Quiz: Equivalent Expressions

다음 중 x 가 0과 10 사이의 정수일 때, x 와 같은 것을 고르시오.

- ☐ $x \% 7$
- ☐ $x \% 23$
- ☐ `ord(chr(x))`
- ☐ `chr(ord(x))`

code

```
print(ord(3))
```

result

```
TypeError: ord() expected string of length 1, but int found
```

Quiz: Equivalent Expressions

다음 중 x 가 0과 10 사이의 정수일 때, x 와 같은 것을 고르시오.

- ☐ $x \% 7$
- ☐ $x \% 23$
- ☐ `ord(chr(x))`
- ☐ `chr(ord(x))`

code

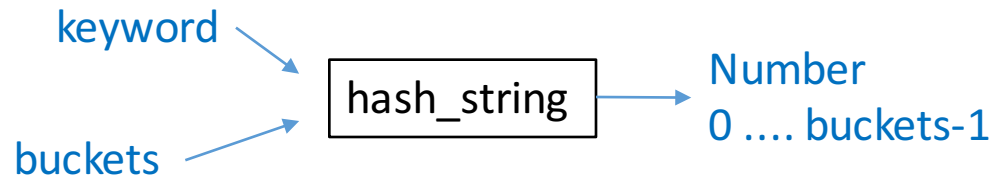
```
print(str(3))  
print(ord(str(3)))  
print(ord('3'))
```

result

```
3  
51  
51
```

Quiz: Bad Hash

Defining a Hash Function



`ord(<one-letter string>)` -> Number
`Number % modulus` -> remainder

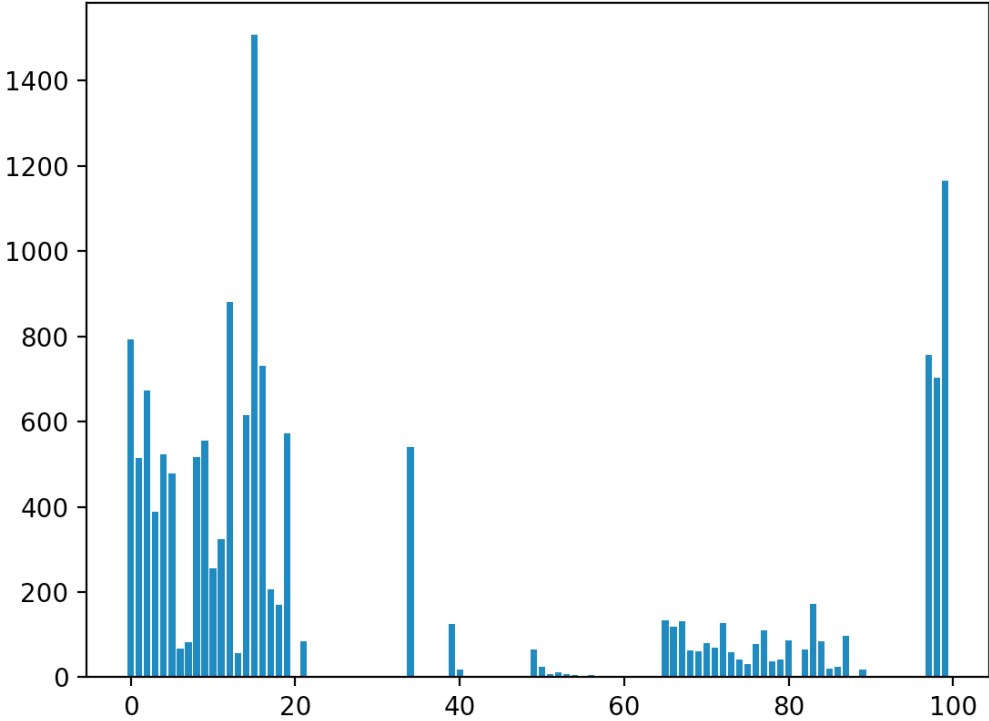
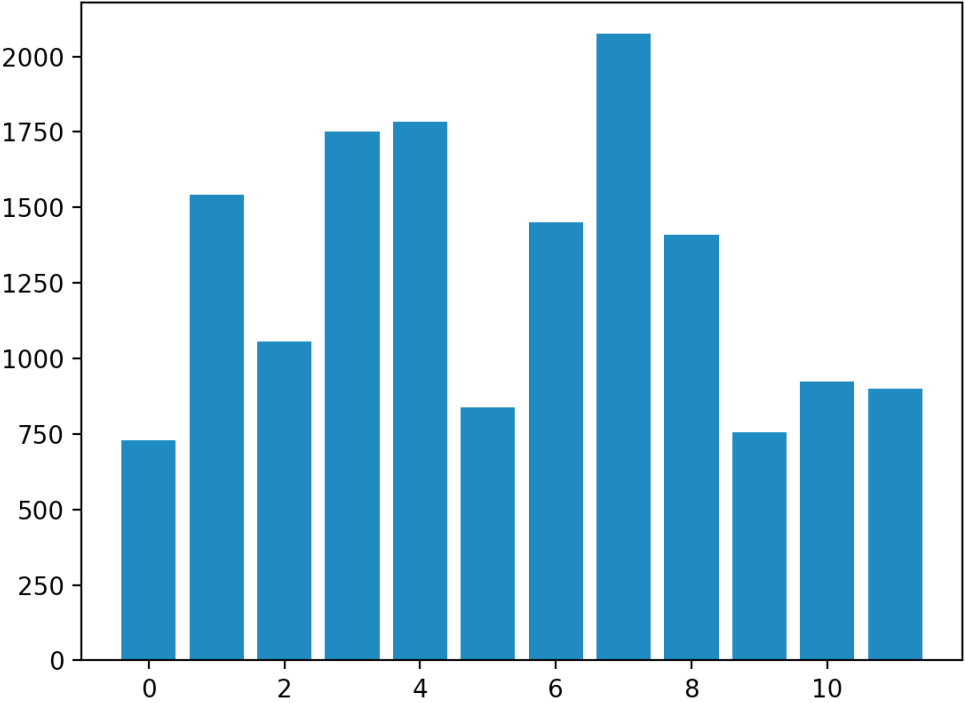
```
def bad_hash_string(keyword, buckets):  
    return ord(keyword[0]) % buckets
```

Quiz: Bad Hash

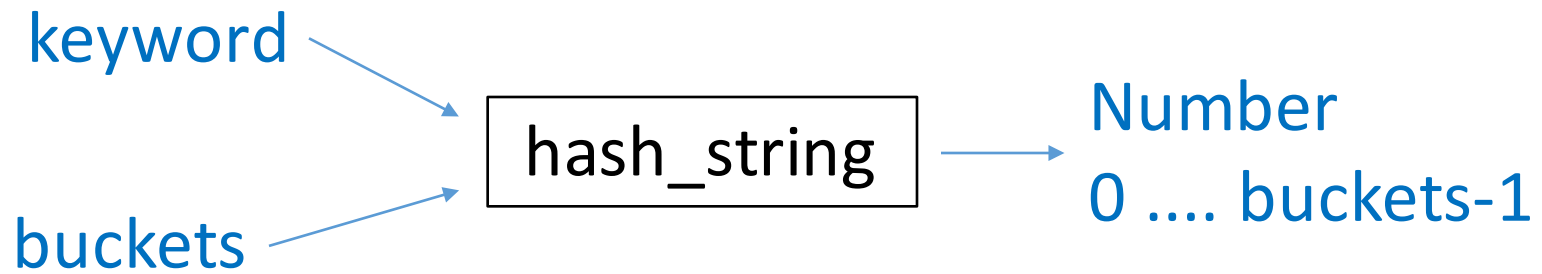
bad_hash_string은 왜 Bad hash function인가?

- ☐ 계산 시간이 너무 오래 걸린다.
- ☐ 한 입력에 대해서 에러를 발생시킨다.
- ☐ 영어 단어와 같이 분산된 단어의 경우,
특정 버킷에 많은 단어들이 담기게 된다.
- ☐ 버킷의 개수가 많아지면,
특정 버킷에는 단어들이 담기지 않게 된다.

Bad Hash



Better Hash Functions

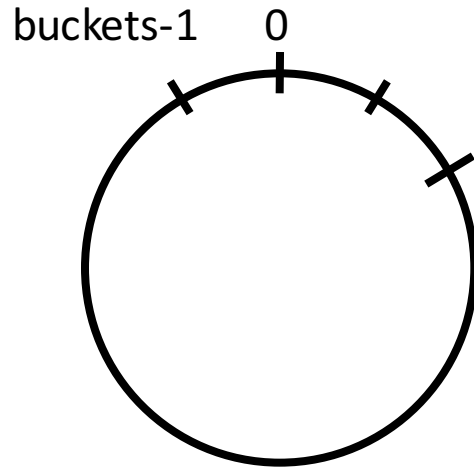


```
p = ['a', 'b', ... ]  
for e in p:  
    <block>
```

```
s = "abcd"  
for c in s:  
    <block>
```


Quiz: Better Hash Functions

keyword(문자열)와 bucket의 개수(숫자)를 입력으로 하여, 그 keyword에 대한 bucket 번호를 리턴하는 hash_string 이라는 함수를 정의하시오.



hash_string('a', 12) = 1
ord('a') = 97
 $12 * 8 + 1$ $13 * 7 + 6$

hash_string('b', 12) = 2
hash_string('a', 13) = 6

hash_string('au', 12) -> 10
ord('a') = 97
ord('u') = 117

$214 \% 12 = 10$

hash_string('udacity', 12) -> 11

Better Hash Functions

code

```
def hash_string(keyword, buckets):  
    h = 0  
    for c in keyword:  
        h = (h + ord(c)) % buckets  
    return h  
  
print(hash_string('a', 12))  
print(hash_string('', 12))  
print(hash_string('udacity', 12))  
print(hash_string('udacity', 1000))
```

result

```
1  
0  
11  
755
```

Testing Hash Functions

code

```
words = get_page('http://www.gutenberg.org/cache/epub/1661/pg1661.txt').split()
counts = test_hash_function(bad_hash_string, words, 12)
print(counts)
counts = test_hash_function(hash_string, words, 12)
print(counts)
counts = test_hash_function(hash_string, words, 100)
print(counts)
```

result

```
[730, 1541, 1055, 1752, 1784, 839, 1452, 2074, 1409, 754, 924, 899]
```

```
[1368, 1267, 1271, 1282, 1283, 1245, 1206, 1229, 1283, 1230, 1235, 1314]
```

```
[137, 127, 118, 137, 129, 149, 117, 126, 112, 128, 142, 131, 151, 129, 150, 124, 157, 144, 151, 150, 137, 105, 151, 144, 141,
153, 140, 185, 144, 154, 154, 163, 192, 158, 164, 190, 153, 177, 162, 175, 172, 166, 179, 165, 186, 167, 173, 144, 174, 166,
154, 164, 177, 178, 163, 171, 187, 162, 160, 181, 166, 161, 135, 154, 169, 156, 150, 147, 154, 164, 125, 173, 156, 165, 145,
150, 150, 145, 148, 152, 148, 148, 161, 140, 188, 150, 150, 122, 167, 123, 142, 133, 136, 132, 126, 142, 151, 135, 152, 162]
```

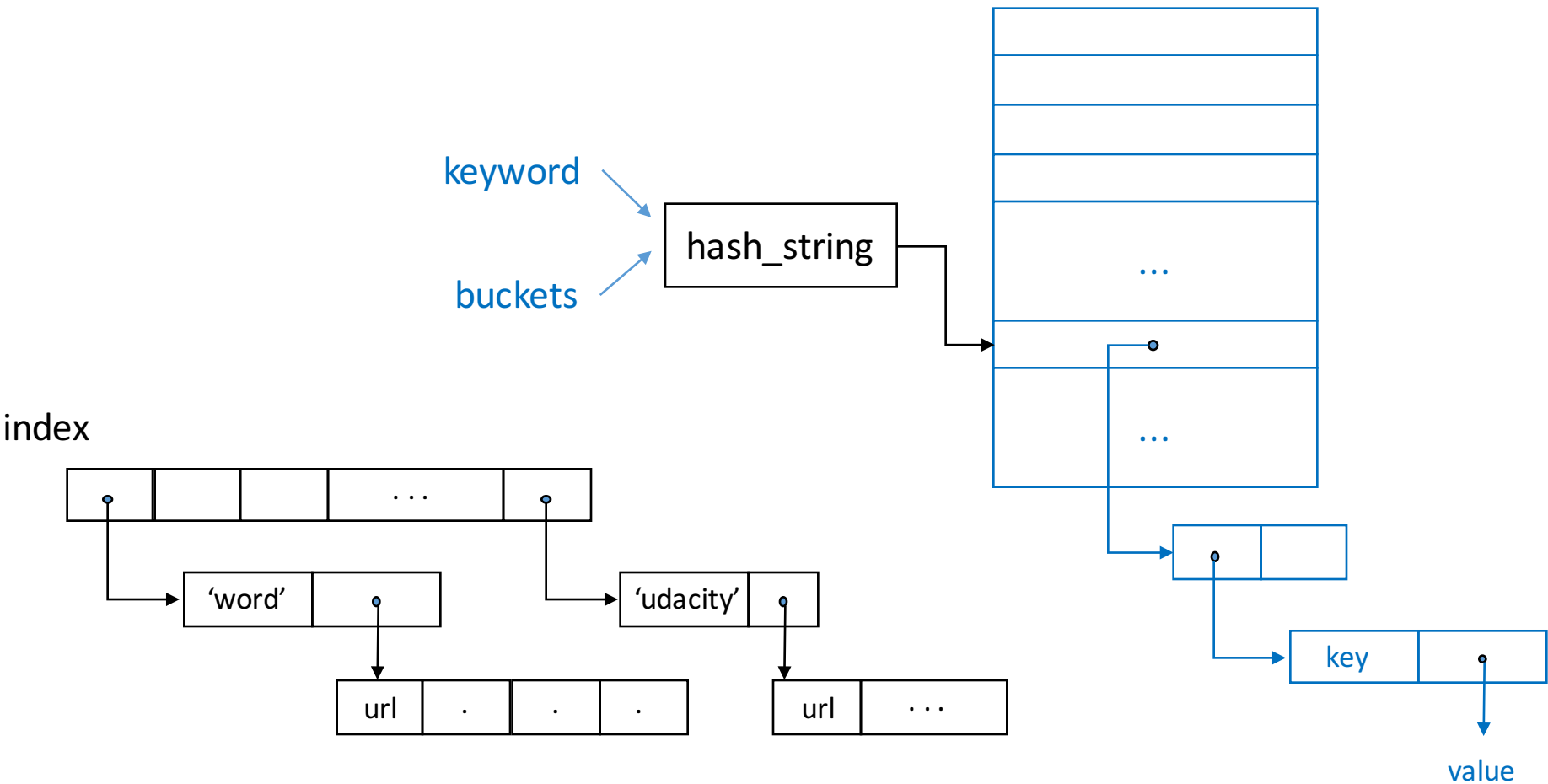
Quiz: Keywords and Buckets

우리의 hash 함수가 bucket에 완벽하게 고르게 key들을 분배했다고 가정할 때,
아래 보기 중 keyword를 lookup하는 시간이 본질적으로 변하지 않는 경우를 모두 고르시오.

각 버킷당 담긴 키워드 수 : k/b

- ☐ 키워드 수가 두배로 늘어나고, 버킷의 숫자는 그대로일 경우
- ☐ 키워드 수는 그대로이고, 버킷의 수는 두배가 되었을 경우
- ☐ 키워드 수도 두배로 늘어나고, 버킷의 수도 두배로 늘어나는 경우
- ☐ 키워드 수는 반으로 줄어듦, 버킷의 수는 그대로일 경우
- ☐ 키워드 수도 반으로 줄어듦, 버킷의 수도 반으로 줄어듦 경우

Quiz: Implementing Hash Tables



Implementing Hash Tables

hash table index를 사용하기 위해서 다음 중 어떤 자료 구조가 적당할지 고르시오.

☐ [[<word>, [<url>, ...]], ...]

☐ [[<word>, [[<url>, ...], [<url>, ...]], ...]

☐ [[[<word>, [<url>, ...]], [<word>, [<url>, ...]], ...]



bucket

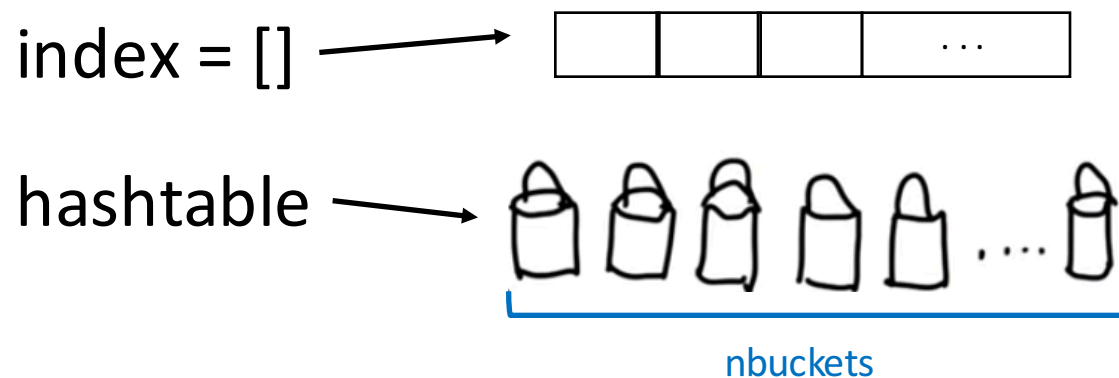
entry

☐ [[[<word>, <word>, ...], [<url>, ...]], ...]

☐ [[[<word>, [[<url>, ...], [<url>, ...]], ...], ...]

Quiz: Empty Hash Table

Creating an Empty Hash Table



숫자 `nbuckets`을 입력 받아서 `nbuckets`만큼의 빈 bucket을 가지는 hash table을 출력하는 `make_hashtable` 이라는 프로시저를 정의하시오.

Empty Hash Table

code

```
def make_hashtable(nbuckets):  
    i = 0  
    table = []  
    while i < nbuckets:  
        table.append([])  
        i = i + 1  
    return table  
  
print(make_hashtable(3))
```

result

```
[[], [], []]
```

for e in <collection>:

<block> List, String

range(<start>, <stop>)

[<start>, <start>+1, ... , <stop> - 1]

range(0, 10) -> [0, 1, 2, ... , 9]

```
def make_hashtable(nbuckets):
```

```
    i = 0
```

```
    table = []
```

```
    while i < nbuckets:
```

```
        table.append([])
```

```
        i = i + 1
```

```
    return table
```

```
    table = []
```

```
    for unused in range(0, nbuckets):
```

```
        table.append([])
```

```
    return table
```


Empty Hash Table

code

```
def make_hashtable(nbuckets):  
    table = []  
    for unused in range(0, nbuckets):  
        table.append([])  
    return table  
  
print(make_hashtable(3))
```

result

```
[[], [], []]
```

code

```
def make_hashtable_NOT(nbuckets):  
    return [[]] * nbuckets  
print(make_hashtable_NOT(3))  
table = make_hashtable_NOT(3)  
table[1].append(['udacity', 'http://udacity.com'])  
print(table[1])  
print(table[0])
```

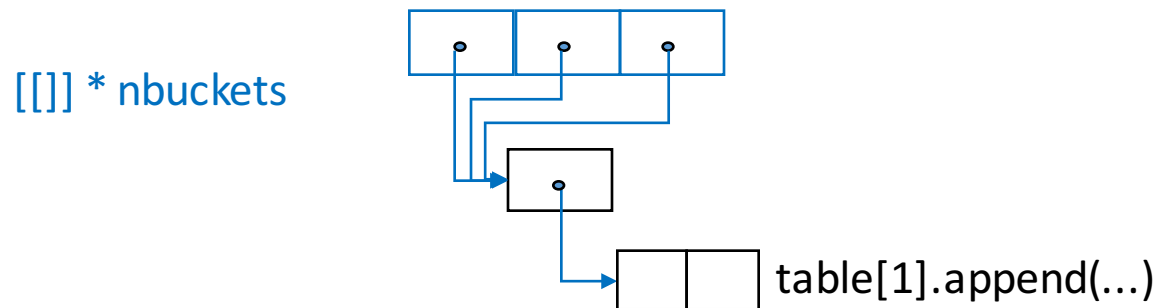
result

```
[[], [], []]  
[['udacity', 'http://udacity.com']]  
[['udacity', 'http://udacity.com']]
```

Quiz: The Hard Way

`[[[]] * nbuckets` 는 왜 빈 hash table을 만들기 위해 사용할 수 없는가?

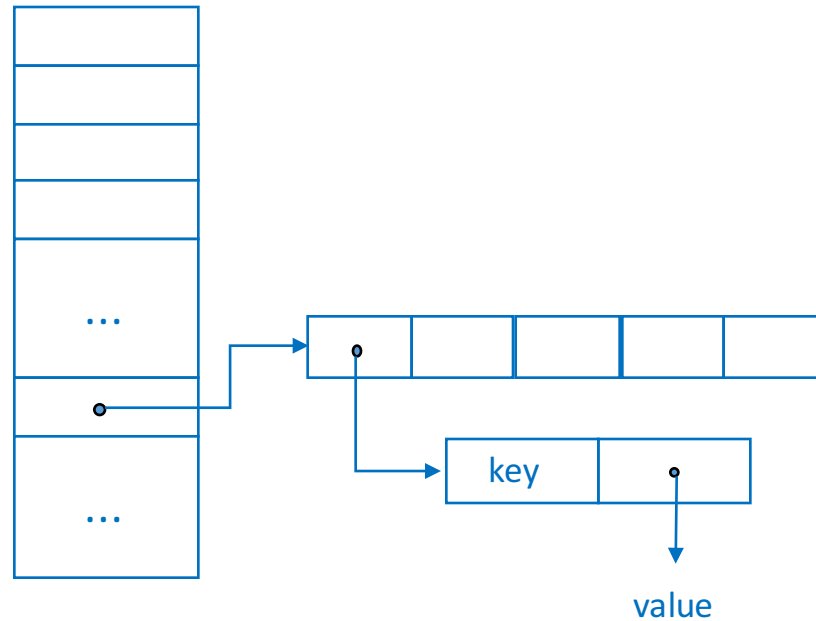
- ☐ Because it is too easy and we like doing things the hard way.
- ☐ Because each element in the output refers to the same empty list.
- ☐ Because `*` for lists means something different than it does for strings.



Quiz: Finding Buckets

Finding the Right Bucket

lookup(htable, word)
add(htable, word, value)



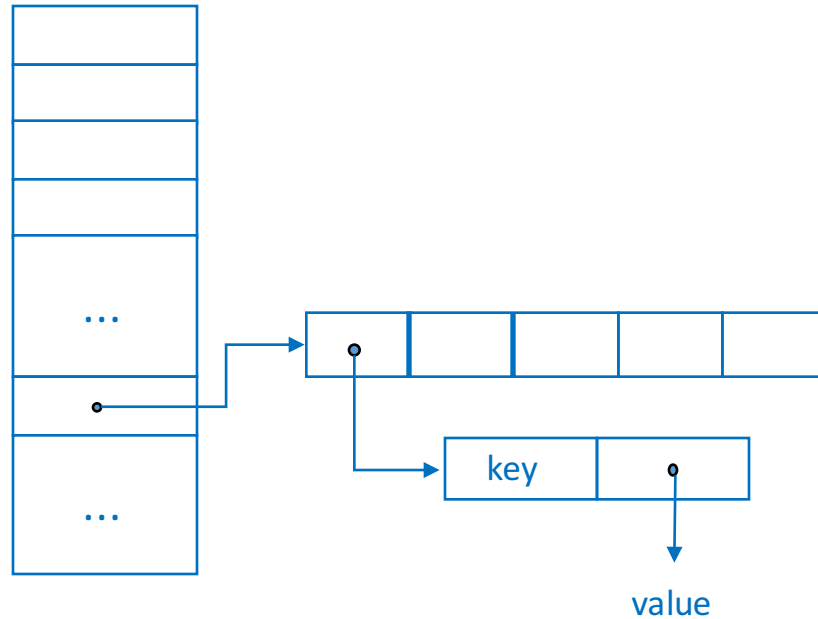
hashtable과 keyword, 두개의 입력을 받아서 keyword에 해당하는 bucket을 출력하는 `hashtable_get_bucket` 이라는 프로시저를 정의하시오.(힌트: 한줄)

hash_string(keyword, nbuckets) -> number, index of bucket
size

Quiz: Adding Keywords

Finding the Right Bucket

`add(htable, word, value)`



hashtable에 key와, key와 연관된 value를 추가하는
`hashtable_add(htable, key, value)` 라는 프로시저를 정의하시오.

```
def hashtable_add(htable, key, value):  
    bucket = hashtable_get_bucket(htable, key)  
    bucket.append([key, value])
```

Adding Keywords

code

```
table = make_hashtable(3)
hashtable_add(table, 'udacity', 23)
print(table)
print(hashtable_get_bucket(table, 'udacity'))
hashtable_add(table, 'audacity', 17)
hashtable_add(table, 'budacity', 19)
print(table)
hashtable_add(table, 'wudacity', 28)
print(table)
```

result

```
[[], [], [['udacity', 23]]]
[['udacity', 23]]
[[['audacity', 17]], [['budacity', 19]], [['udacity', 23]]]
[[['audacity', 17]], [['budacity', 19], ['wudacity', 28]], [['udacity', 23]]]
```

code

```
hashtable_add(table, 'udacity', 27)
print(hashtable_get_bucket(table, 'udacity'))
```

result

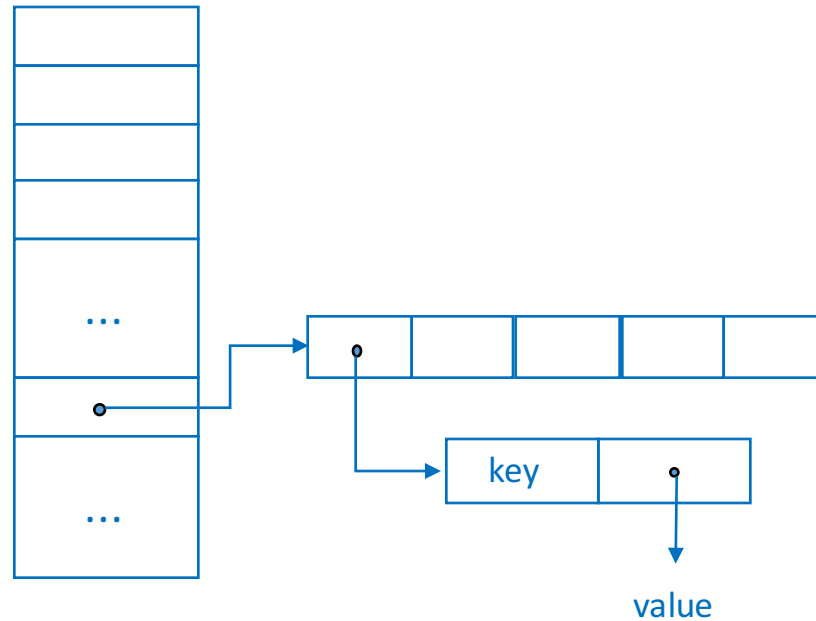
```
[['udacity', 23], ['udacity', 27]]
```

Quiz: Lookup

Finding the Right Bucket

lookup(htable, word)

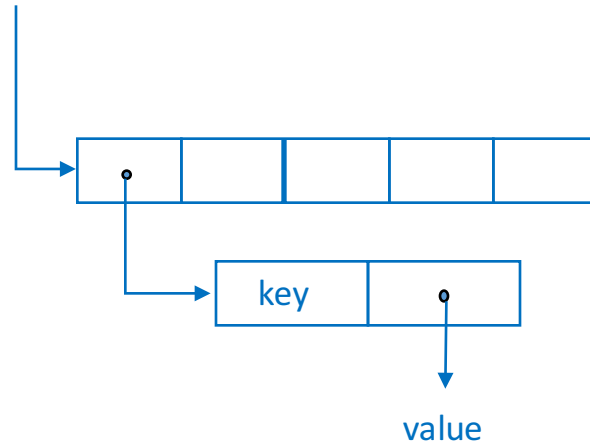
└─ value



hashtable과 key(문자열), 두개의 입력을 받아서, 입력받은 key와 연관된 value를 출력하는 `hashtable_lookup(htable, key)` 라는 프로시저를 정의하시오.
만약 key가 table에 없다면, `None`을 출력하시오.

Lookup

```
def hashtable_lookup(htable, key):  
    bucket = hashtable_get_bucket(htable, key)  
    for entry in bucket:  
        if entry[0] == key:  
            return entry[1]  
    return None
```



code

```
table = make_hashtable(3)  
hashtable_add(table, 'udacity', 23)  
hashtable_add(table, 'audacity', 17)  
hashtable_add(table, 'budacity', 19)  
hashtable_add(table, 'wudacity', 28)  
  
print(hashtable_lookup(table, 'udacity'))  
hashtable_add(table, 'udacity', 27)  
  
print(hashtable_lookup(table, 'udacity'))  
print(hashtable_get_bucket(table, 'udacity'))
```

result

```
23  
23  
[['udacity', 23], ['udacity', 27]]
```

Quiz: Update

key와 연관된 value를 출력하는

`hashtable_update(htable, key, value)` 라는 프로시저를 정의하시오.

만약 key가 이미 table에 있다면, value를 입력받은 새로운 value로 변경하시오.

그렇지 않다면, 입력받은 key와 value에 대한 새로운 entry를 추가하시오.

```
def hashtable_update(htable, key, value):  
    bucket = hashtable_get_bucket(htable, key)  
    for entry in bucket:  
        if entry[0] == key:  
            entry[1] = value  
            return  
    bucket.append([key, value])
```


Quiz: Update

code

```
table = make_hashtable(3)
hashtable_add(table, 'udacity', 23)
hashtable_add(table, 'audacity', 17)
hashtable_add(table, 'budacity', 19)
hashtable_add(table, 'wudacity', 28)

print(hashtable_lookup(table, 'udacity'))
hashtable_update(table, 'udacity', 27)
print(hashtable_lookup(table, 'udacity'))
print(hashtable_get_bucket(table, 'udacity'))
```

result

```
23
27
[['udacity', 27]]
```

Dictionaries

String

'hello'

sequence of
characters

immutable

s[i]

i th character in s

~~s[i] = 'x'~~

List

['alpha', 23]

list of

elements

mutable

p[i]

i th element of p

p[i] = u

replace value of

i th element with u

Dictionary

{ 'hydrogen': 1,
 'helium': 2 }

set of <key, value> pairs

mutable

d[k] ← key

value associated with k in d

d[k] = v

update k -> v

Using Dictionaries

code

```
elements = {'hydrogen': 1, 'helium': 2, 'carbon': 6}
print(elements)
print(elements['hydrogen'])
print(elements['carbon'])
print(elements['lithium'])
```

result

```
{'hydrogen': 1, 'helium': 2, 'carbon': 6}
1
6
KeyError: 'lithium'
```

code

```
elements['lithium'] = 3
elements['nitrogen'] = 8

print(elements['nitrogen'])
elements['nitrogen'] = 7
print(elements['nitrogen'])
```

result

```
8
7
```

Quiz: Population

세계에서 가장 큰 도시들에 대한 정보를 제공하는 `population` 이라는 Dictionary를 정의하시오.
key는 도시의 이름(문자열)으로 하고,
연관된 value는 백만단위(millions)의 인구수로 하시오.

Shanghai	17.8
Istanbul	13.3
Karachi	13.0
Mumbai	12.5

code

```
population = {}  
population['Shanghai'] = 17.8  
population['Istanbul'] = 13.3  
population['Karachi'] = 13.0  
population['Mumbai'] = 12.5  
population['Charlottesville'] = 0.043  
  
print(population['Shanghai'])  
print(population['Charlottesville'])
```

result

```
17.8  
0.043
```

A Noble Gas

code

```
elements = {}  
elements['H'] = {'name': 'Hydrogen', 'number': 1, 'weight': 1.00794}  
elements['He'] = {'name': 'Helium', 'number': 2, 'weight': 4.002602,  
                  'noble gas': True}  
print(elements['H'])  
print(elements['H']['name'])  
print(elements['H']['weight'])  
print(elements['He']['noble gas'])  
print(elements['H']['noble gas'])
```

result

```
{'name': 'Hydrogen', 'number': 1, 'weight': 1.00794}  
Hydrogen  
1.00794  
True  
KeyError: 'noble gas'
```

Quiz: Modifying the Search Engine

list index를 Dictionary index로 변경할 때,
아래 우리가 만든 검색엔진의 프로시저 중에 변경되어야 하는 함수를 고르시오.

☐ get_all_links

☐ add_to_index

☐ crawl_web

☐ lookup

☐ add_page_to_index

```
def get_all_links(page):
    links = []
    while True:
        url, endpos = get_next_target(page)
        if url:
            links.append(url)
            page = page[endpos:]
        else:
            break
    return links
```

```
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 add_page_to_index(index, url, content):
    words = content.split()
    for word in words:
        add_to_index(index, word, url)
```

```
def add_to_index(index, keyword, url):
    for entry in index:
        if entry[0] == keyword:
            entry[1].append(url)
            return
    # not found, add new keyword to index
    index.append([keyword, [url]])
```

```
def lookup(index, keyword):
    for entry in index:
        if entry[0] == keyword:
            return entry[1]
    return None
```


Modifying the Search Engine

```
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []
index = [] 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 add_page_to_index(index, url, content):
    words = content.split()
    for word in words:
        add_to_index(index, word, url)
```

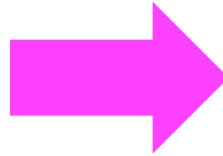
```
def add_to_index(index, keyword, url):
    for entry in index:
        if entry[0] == keyword:
            entry[1].append(url)
    return
# not found, add new keyword to index
index.append([keyword, [url]])
```

```
def add_to_index(index, keyword, url):
    if keyword in index:
        index[keyword].append(url)
    else:
        # not found, add new keyword to index
        index[keyword] = [url]
```



Quiz: Changing Lookup

```
def lookup(index, keyword):  
    for entry in index:  
        if entry[0] == keyword:  
            return entry[1]  
    return None
```



```
def lookup(index, keyword):  
    if keyword in index:  
        return index[keyword]  
    else:  
        return None
```