Data structure [A03]

김종규, PhD

2017-03-20

Outline

- Important Big-*O*'s
- Proving Big-O
- Examples of abstract data types

Sorting algorithm

```
  def test_sort(x)
1   n = len(x)
2   for j in range(n):
3     for i in range(n-1):
4       if x[i] > x[i+1]:
5         (x[i+1],x[i])=(x[i],x[i+1])
```

$\longrightarrow$ Bubble sort

- 비교는 몇 번? $\longrightarrow n^2 \longrightarrow$ The most complex operation

Interesting operation

```
k++    # k = k + 1
```

Complexity

- What is the value of `k`?

```
k = 0
i = 0
while i < 92:
  k++   # k = k + 1
  i++   # i = i + 1
j = 0
```

⟶ Sequential operations

Python

```
k = 0
i = 0
while i < 92:
    k = k + 1
    i = i + 1
print(k)
```

C

```
int main()
{
        int i, k;
        i = 0;
        k = 0;
        while (i < 92) {
                k++;
                i++;
        }
        printf("%d\n", k);

        getchar();
    return 0;
}
```

Java

```
public static void main(String [] args)
{
        int i, k;
        i = 0;
        k = 0;
        while (i < 92) {
                k++;
                i++;
        }
        System.out.println(k);

        getchar();
    return 0;
}
```

Complexity

- What is the value of `k`?

```
k = 0
i = 0
while i < 92:
  k++  # k = k + 1
  i++  # i = i + 1
j = 0
while j < 44:
  k++
  j++
print(k)
```

⟶ Sequential operations

Complexity

- What is the value of `k`?

```
k = 0
i = 0
while i < m:
  k++
  i++
j = 0
while j < n:
  k++
  j++
print(k)
```

$\longrightarrow$ $O(m+n)$

Complexity

- What is the value of `k`?

```
k = 0
i = 0
while i < m:
  k++
  j = 0
  while j < n:
    k++
    j++
  i++
print(k)
```

$\longrightarrow$ Nested loop

$\longrightarrow$ $O(m \times n)$

Complexity

- What is the value of `k`?

```
k = 0
i = 0
while i < n:
  k++
  j = 0
  while j < n:
    k++
    j++
  i++
print(k)
```

$\longrightarrow$ Nested loop

$\longrightarrow$ $O(n \times n) = O(n^2)$

Complexity

- What is value of `k`?

```
def f(n):
    global k
    if n == 0:
        k += 1
        return 1
    else:
        return f(n-1)+f(n-1)
k = 0
f(10)
print(k)
```

Complexity

- What is value of `k`?

```
k = 0
i = 1
while i < n:
  k++
  i = i * 2
```

$\longrightarrow$ $O(\log_2 n)$

Complexity

- What is value of `k`?

```
k = 0
i = n
while i > 0:
  k++
  i = i / 2
```

$\longrightarrow$ $O(\log_2 n)$

Importatn points on Big-O

- $O(f(n)) \in O(g(n))$ and $O(g(n)) \notin O(f(n)) \longrightarrow f(n)$ does not grow faster than $g(n)$
- Complexity is usually denoted by $T(n)$
- We could identify the complexity of a part of a code
- By inspecting that if it's sequential operation or nested operation, we could add or multiply the complexity
- If an index grows or diminishes exponentially, its complexity is $O(\log_2 n)$
- When a complexity is $T(n) = 2T(n)$, its complexity is $O(2^n)$

Algebra of Big-O

- $f(n) + g(n) \longrightarrow O(f(n) + g(n))$
- $f(n) \cdot g(n) \longrightarrow O(f(n) \cdot g(n))$
- $O(cf(n)) = O(f(n)$ where c is a constant
- $f(n) \geq g(n) \longrightarrow f(n) + g(n) \in O(f(n))$
- $O(\log_2 n)$, a.k.a $O(\lg n)$ is very fast

Examples

- $O(n)$

```
for i in range(10):
  for j in range(n):
    k++
```

Examples

- $O(n^2)$

```
for i in range(n):
  for j in range(n):
    k++
```

Examples

- $O(n^3)$

```
for i in range(n):
  for j in range(n):
    for m in range(n):
      k++
```

Examples

- $O(n^3)$

```
for i in range(n):
  for j in range(n):
    for m in range(n):
      k++

for i in range(n):
  for j in range(n):
    k++
```

Example

- $O(\lg n)$

```
k = 0
i = 1
while i < n:
  i = i * 2
  k++
```

Example

- $O(n \lg n)$

```
k = 0
i = 1
while i < n:
  i = i * 2
  for j in range(n):
    k++
```

Abstract data type

```
  def test_sort(x)
1   n = len(x)
2   for j in range(n):
3     for i in range(n-1):
4       if x[i] > x[i+1]:
5         (x[i+1],x[i])=(x[i],x[i+1])
```

$\longrightarrow$ Bubble sort

- 비교는 몇 번? $\longrightarrow$ Interesting operation of a data structure

Abstract data type: Queue

그림: Queue

Abstract data type: Stack

그림: Queue

Queue

- Enqueue: 새로운 작업을 요청
- Dequeue: 쌓여있는 작업을 처리
- Empty: 남은 작업이 있는지

Stack

- Push: 새로 쌓아넣기
- Pop: 맨 위에서 꺼내기
- Empty: 남은 것이 있는지?

Data structure: Queue

- Circular queue

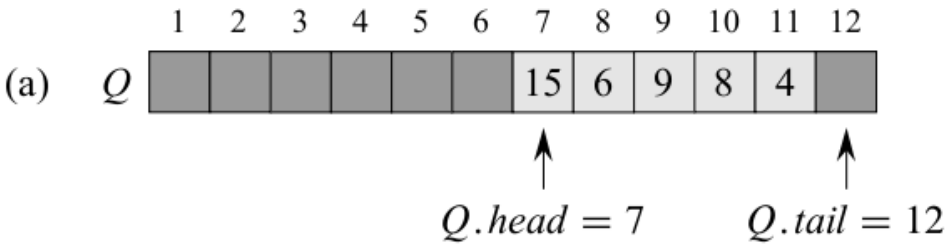

그림: Circular queue

Data structure: Queue

- Circular queue

```
ENQUEUE(Q, x)
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

```
DEQUEUE(Q)
1  x = Q[Q.head]
2  if Q.head == Q.length
3        Q.head = 1
4  else Q.head = Q.head + 1
5  return x
```

Data structure: Stack

- Array based stack

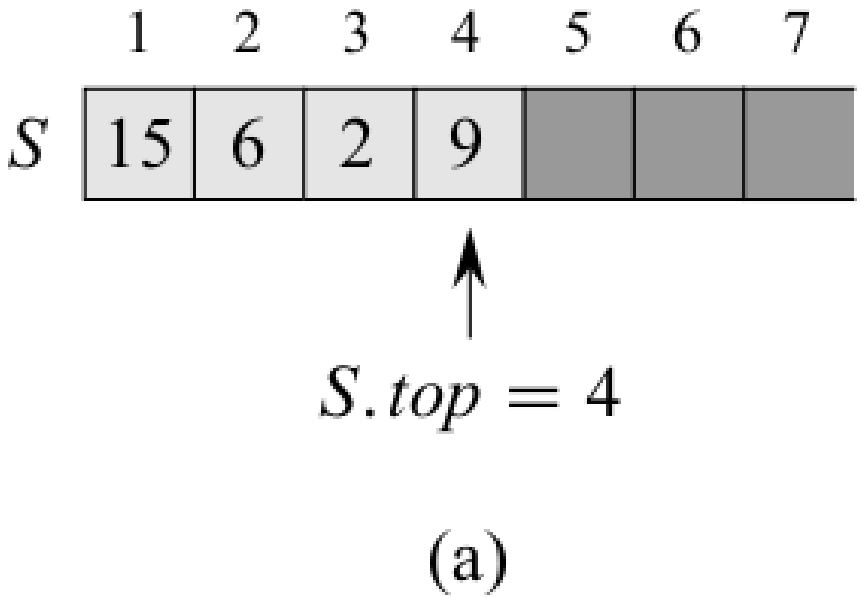

그림: Stack

Data structure: Stack

- Array based stack

```
STACK-EMPTY(S)
1  if S.top == 0
2       return TRUE
3  else return FALSE

PUSH(S, x)
1  S.top = S.top + 1
2  S[S.top] = x

POP(S)
1  if STACK-EMPTY(S)
2       error "underflow"
3  else S.top = S.top − 1
4       return S[S.top + 1]
```

Example problem

- Circular queue 에서 empty() 는 어떻게 정의할 수 있을까?
- 크기 n 인 array 로 circular queue 를 만들었을 때 최대로 넣을 수 있는 element 의 갯수는?
- Stack 을 이용하여 queue 를 구현할 수 있을까?

Further reading

- 이번 주: Chapter 10
- 다음 주: Chapter 6, 7, 12

Wrap-up

- Worst case complexity of insertion sort is $O(n^2)$
- Linear combination of Big-O functions is dominated by the fastest growing function
- Stacks and queues are the most fundamental abstract data types in computer science
- By limiting the maximum elements to put, we could implement the data structure of stacks and queues