시간 복잡도와 언어별 유의사항

최백준 choi@startlink.io

효율성

효율성

Algorithm

- 알고리즘 문제를 해결하는 어떤 코드를 작성했을 때, 이 프로그램이 얼마나 효율적인지 알고 싶다.
- 다음 중 무엇이 가장 중요할까?

1. 수행 시간
2. 사용한 메모리
3. 코드의 길이

효율성

Algorithm

- 수행 시간이 가장 중요하다.

효율성

Algorithm

- 어떤 프로그램을 작성했는데, 시간이 30일이 걸리면 정말로 30일동안 실행시켜야 한다.
- 어떤 프로그램을 작성했는데, 메모리가 64GB 필요한데, 메모리가 부족하면 램을 구매하면 된다.
- 이런 이유 때문에 문제를 해결할 때는 시간이 중요하다.

효율성

Algorithm

- 앞에서 얘기한 것 보다 더 많은 항목이 있고, 상황에 따라 무엇이 중요한지는 다르다
- 앞 페이지에서 얘기한 것은 알고리즘 문제 해결에 한정지은 것이다.

문제의 크기

문제의 크기

Algorithm

- 개발 상황에서 접하게 되는 상황은 문제를 해결하는 것이고
- 항상 문제의 크기가 존재한다.
- 예시 1) 쇼핑몰 장바구니 물건의 개수
- 예시 2) 게임 동시 접속자의 수
- 이러한 문제의 크기를 보통 N이라고 하고, 문제의 크기 N에 따라 걸리는 시간이 다르다.

문제의 크기

Algorithm

- 웹 사이트를 만드는 경우에
- 10명이 동시 접속하는 사이트를 만드는 것과 10만명이 동시 접속하는 사이트를 만드는 방법은
- 매우 큰 차이가 있다. 또, 10만명이 동시 접속하는 사이트를 만드는 방법이 더 어렵다.
- 문제를 해결할 때도 문제의 크기에 따라 알맞은 방법을 선택하는 것이 좋다.
- 대부분의 문제는 가장 빠른 방법이 정해져 있지만, 가장 빠른 방법이 너무 어려운 경우일 수도 있어, 그 방법보다는 상대적으로 느린 (하지만 문제는 해결할 수 있음) 방법을 이용하기도 한다.

문제의 크기

Algorithm

- 이러한 이유 때문에 문제를 해결할 때는 문제의 크기를 먼저 보고 방법을 생각해야 한다.

시간 복잡도

시간 복잡도

Time Complexity

- 시간 복잡도를 이용하면 작성한 코드가 시간이 대략 얼마나 걸릴지 예상할 수 있다.
- 표기법으로 대문자 O를 사용한다. (다양한 시간 복잡도가 많지만, 보통 Big-O만 사용한다)
- 영어로는 Big O Notation
- 입력의 크기 N에 대해서 시간이 얼마나 걸릴지 나타내는 방법
- 즉, 최악의 경우에 시간이 얼마나 걸릴지 알 수 있다.

시간 복잡도

Time Complexity

- 총 N명의 사람이 식당에 방문했다.
- 식당에는 메뉴가 M개 있고, 메뉴판이 1개 있다.
- 사람 1명이 메뉴판을 읽는데 걸리는 시간은 O(M)이다.
- 주문한 모든 메뉴는 동시에 나왔고, 각 사람 i가 식사를 하는데 걸리는 시간은 A_i이다.
- 각 사람이 계산을 하는데 걸리는 시간은 O(P) 이다.
- 각 사람이 메뉴판에 있는 모든 메뉴를 읽는 시간 복잡도 = O(NM)
- 모든 사람이 식사를 마치는데 걸리는 시간 = $O(\max(A_i))$
- 식사를 모두 마친 다음 한 줄로 서서 각자 계산을 하는 시간 복잡도 = O(NP)

시간 복잡도

Time Complexity

- 시간 복잡도는 소스를 보고 계산할 수도 있고, 소스를 작성하기 전에 먼저 계산해볼 수 있다.
- 문제를 풀기 전에 먼저 생각한 방법의 시간 복잡도를 계산해보고 이게 시간 안에 수행될 것 같은 경우에만 구현하는 것이 좋다.

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

```
int sum = 0;
for (int i=1; i<=N; i++) {
    sum += i;
}
```

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

```
int sum = 0;
for (int i=1; i<=N; i++) {
    sum += i;
}
```

- 시간 복잡도: O(N)

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

```
int sum = 0;
for (int i=1; i<=N; i++) {
    for (int j=1; j<=N; j++) {
        if (i == j) {
            sum += j;
        }
    }
}
```

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

```
int sum = 0;
for (int i=1; i<=N; i++) {
    for (int j=1; j<=N; j++) {
        if (i == j) {
            sum += j;
        }
    }
}
```

- 시간 복잡도: $O(N^2)$

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

```
int sum = 0;
sum = N*(N+1)/2;
```

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

```
int sum = 0;
sum = N*(N+1)/2;
```

- 시간 복잡도: O(1)

시간 복잡도

Time Complexity

- 대표적인 시간 복잡도는 아래와 같다.
- O(1)
- O(lgN)
- O(N)
- O(NlgN)
- O(N^2)
- O(N^3)
- O(2^N)
- O(N!)

시간 복잡도

Time Complexity

- 시간 복잡도 안에 가장 큰 입력 범위를 넣었을 때, 1억이 1초정도이다.
- 이 값은 대략적인 값으로, 실제로 구현해보면 1억을 조금 넘어도 1초 이내에 수행이 가능하다.

시간 복잡도

Time Complexity

- 1초가 걸리는 입력의 크기
- O(1)
- O(lgN)
- O(N) : 1억
- O(NlgN) : 5백만
- O(N^2) : 1만
- O(N^3) : 500
- O(2^N) : 20
- O(N!) : 10

시간 복잡도 계산

Time Complexity

- Big O Notation 에서 상수는 버린다.
- O(3N^2) = O(N^2)
- O(1/2 N^2) = O(N^2)
- O(5) = O(1)
- 두 가지 항이 있을 때, 변수가 같으면 큰 것만 빼고 다 버린다.
- O(N^2 + N) = O(N^2)
- O(N^2 + NlgN) = O(N^2)
- 두가지 항이 있는데 변수가 다르면 놔둔다.
- O(N^2 + M)

메모리

사용한 메모리

Memory

- 메모리 제한은 보통 넉넉하기 때문에, 걱정할 필요가 없다.
- 대략적으로 얼마나 공간을 사용할지 예상할 수는 있다.

사용한 메모리

Memory

- 보통 가장 많은 공간을 사용하는 것은 보통 배열이다.
- 배열이 사용한 공간: 배열의 크기 × 자료형의 크기 B
- `int a[10000];`
- `int a[100000];`
- `int a[1000000];`
- `int a[1000][1000];`
- `int a[10000][10000];`
- `int a[100000][100000];`

사용한 메모리

Memory

- 보통 가장 많은 공간을 사용하는 것은 보통 배열이다.
- 배열이 사용한 공간: 배열의 크기 × 자료형의 크기 B
- `int a[10000];` → 10000×4B
- `int a[100000];` → 100000×4B
- `int a[1000000];` → 1000000×4B
- `int a[1000][1000];` → 1000×1000×4B
- `int a[10000][10000];` → 10000×10000×4B
- `int a[100000][100000];` → 100000×100000×4B

사용한 메모리

Memory

- 보통 가장 많은 공간을 사용하는 것은 보통 배열이다.
- 배열이 사용한 공간: 배열의 크기 × 자료형의 크기 B
- `int a[10000];` → `10000×4B = 40,000B`
- `int a[100000];` → `100000×4B = 400,000B`
- `int a[1000000];` → `1000000×4B = 4,000,000B`
- `int a[1000][1000];` → `1000×1000×4B = 4,000,000B`
- `int a[10000][10000];` → `10000×10000×4B = 400,000,000B`
- `int a[100000][100000];` → `100000×100000×4B = 40,000,000,000B`

사용한 메모리

Memory

- 보통 가장 많은 공간을 사용하는 것은 보통 배열이다.
- 배열이 사용한 공간: 배열의 크기 × 자료형의 크기 B
- `int a[10000];` → 10000×4B = 40,000B = 39.06KB
- `int a[100000];` → 100000×4B = 400,000B = 390.62KB
- `int a[1000000];` → 1000000×4B = 4,000,000B = 3.814MB
- `int a[1000][1000];` → 1000×1000×4B = 4,000,000B 3.814MB
- `int a[10000][10000];` → 10000×10000×4B = 400,000,000B = 381.469MB
- `int a[100000][100000];` → 100000×100000×4B = 40,000,000,000B = 37.25GB

사용한 메모리

Memory

- 보통 배열의 크기가 크면 시간 초과를 받는 경우가 많다.
- 불필요한 공간이 없다면, 대부분 메모리 제한은 알아서 지켜진다.

입/출력

C 입출력

C

- C의 경우에는 scanf/printf를 사용할 수 있다.

C++ 입출력

C++

- C++의 경우에는 scanf/printf, cin/cout을 사용할 수 있다.
- cin/cout은 scanf/printf보다 느리기 때문에, 입/출력이 많은 문제의 경우에는 scanf/printf를 사용하는 것이 좋다.
- cin/cout의 경우 아래 세 줄을 추가하면 scanf/printf만큼 빨라진다.

```
ios_base::sync_with_stdio(false);
cin.tie(nullptr);
cout.tie(nullptr);
```

- 이 경우에는 cin/cout와 scanf/printf을 섞어 쓰면 안된다. 즉, cin/cout만 써야 한다.
- endl 대신에 '\n'을 사용한다.

Java 입출력

Java

- Java는 입력은 Scanner, 출력은 System.out을 사용한다.

```
Scanner sc = new Scanner(System.in);
```

- 입력이 많은 경우에는 속도가 느리기 때문에, BufferedReader를 사용한다.

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

- 출력이 많은 경우에는 StringBuilder를 사용해서 한 문자열로 만들어서 출력을 한 번만 사용하거나
- BufferedWriter를 사용한다.

Python 입출력

Python

- Python은 입력은 `input`, 출력은 `print`를 사용한다.
- 입력이 많은 경우에는 속도가 느리기 때문에, `sys.stdin.readline()`를 사용한다.
- 출력이 많은 경우에는 한 문자열로 만들어서 출력을 한 번만 사용한다.

입력 속도 비교

Input

- 10,000이하의 자연수 10,000,000개가 적힌 파일을 입력받는데 걸리는 시간.
- 출처: https://www.acmicpc.net/blog/view/56

언어	입력 방법	평균 (초)
C++17	`ios_base::sync_with_stdio(false); cin.tie(NULL);`	0.5938
Java	`BufferedReader, Integer.parseInt`	0.6585
C11	`scanf`	0.9206
C++17	`cin`	2.1742
Python 3	`int(sys.stdin.readline())`	4.4394
Java	`Scanner`	4.8448
Python 3	`int(input())`	12.4443

출력 속도 비교

Output

- 1부터 10,000,000까지 자연수를 한 줄에 하나씩 출력하는 시간
- 출처: https://www.acmicpc.net/blog/view/57

출력 속도 비교

Output

언어	출력 방법	평균 (초)
C++17	`ios_base::sync_with_stdio(false); cout << i << '\n';`	0.8270
C++17	`printf("%d\n",i);`	0.8614
Java	`BufferedWriter, bf.write(i + "\n");`	0.9581
Java	`StringBuilder, System.out.println(sb);`	1.1881
Python 3	`print('\n'.join(map(str,range(1,n+1))))`	2.3312
Java	`Scanner`	4.8448
Python 3	`for i in range(1,n+1): print(i)`	5.8186
C++17	`cout << i << endl;`	11.5322
Java	`System.out.println(i);`	30.0130

A+B

https://www.acmicpc.net/problem/1000

- 두 수를 입력받고 A+B를 출력하는 문제

A+B

https://www.acmicpc.net/problem/1000

- 소스: http://codeplus.codes/bb13daa70e14414c90f12e2f92a92c08

A+B - 3

https://www.acmicpc.net/problem/10950

- 테스트 케이스 형식으로 주어지는 경우에는
- 각각을 독립적인 문제로 생각하고 풀면 된다.
- 전체 테스트 케이스를 입력 받은 다음에, 풀지 않아도 된다.

A+B - 3

https://www.acmicpc.net/problem/10950

```
int t;
int a[100],b[100];
scanf("%d",&t);
for (int i=0; i<t; i++) {
    scanf("%d %d",&a[i],&b[i]);
}

for (int i=0; i<t; i++) {
    printf("%d\n",a[i]+b[i]);
}
```

- 이렇게 입력을 다 받고, 모아서 하나씩 출력하지 않고

A+B - 3

https://www.acmicpc.net/problem/10950

```
int t;
int a,b;
scanf("%d",&t);
while (t--) {
	scanf("%d %d",&a,&b);
	printf("%d\n",a+b);
}
```

- 이렇게 하나 하나 입력받고 풀면 된다.
- 앞 페이지의 방법 처럼 구현하면, T개수를 모를때 배열의 크기를 정하기 어렵다.
- 또, 전체 입력이 매우 큰 경우에 매우 큰 크기의 배열을 필요하게 된다.

A+B - 3

https://www.acmicpc.net/problem/10950

- 소스: http://codeplus.codes/653258a2021c43748e78dc909111326c

A+B - 4

https://www.acmicpc.net/problem/10951

- 이 문제 처럼 입력이 몇 개인지 주어지지 않은 경우에는
- 입력을 EOF까지 받으면 된다.
- C: `while (scanf("%d %d",&a,&b) == 2)`
- C++: `while (cin >> a >> b)`
- Java: `while (sc.hasNextInt())`
- Python: `try`, `except`를 이용
- `scanf`의 리턴값은 성공적으로 입력받은 변수의 개수다.

A+B - 4

https://www.acmicpc.net/problem/10951

- 소스: http://codeplus.codes/e7d7f1860b5545aa834c39f715268a0a

언어별 유의사항

시간 초과

Time Limit Exceeded

- `strlen(s)`의 시간 복잡도는 O(s의 길이) 이기 때문에
- `for (int i=0; i<strlen(s); i++) {`
- s의 길이를 N이라고 한다면 $O(N^2)$과 같다.
- C++ `string`의 `.length()` 와 Java `String`의 `.length()`, Python의 `len`은 O(1) 이다.

시간 초과

Time Limit Exceeded

- Python으로 문제를 해결하는 경우에는 각 연산의 시간 복잡도를 알고 있어야 한다.

```
a = list(range(1, 10000001))
a = a + [1000001]
a.append(1000001)
a = a + [1,2,3]
a.extend([1, 2, 3])
a += [4, 5, 6]

if 10 in a:
    print(1)

print(len(a))
```

시간 초과

Time Limit Exceeded

- Python으로 문제를 해결하는 경우에는 각 연산의 시간 복잡도를 알고 있어야 한다.

```
a = list(range(1, 10000001))
a = a + [1000001] # O(N)
a.append(1000001) # O(1)
a = a + [1,2,3] # O(N+K)
a.extend([1, 2, 3]) # O(K)
a += [4, 5, 6] #O(K)

if 10 in a: # O(N)
    print(1)

print(len(a)) # O(1)
```

시간 초과

Time Limit Exceeded


```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s;
    int n = 1000000;
    for (int i=0; i<n; i++) {
        s += "A";
    }
    return 0;
}
```

시간 초과

Time Limit Exceeded


```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s;
    int n = 1000000;
    for (int i=0; i<n; i++) {
        s = s + "A";
    }
    return 0;
}
```

시간 초과

Time Limit Exceeded


```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        String s = "";
        int n = 1000000;
        for (int i=0; i<n; i++) {
            s += "A";
        }
    }
}
```

시간 초과

Time Limit Exceeded

- C++에서 `string` 의 += 연산은 O(K) 이다.
- C++에서 `string` 의 + 연산은 O(N+K) 이다.
- Java에서 `String`의 += 연산은 O(N+K) 이다.
- Java의 경우 `StringBuilder`를 이용해야 한다.