수학/자료구조 기초

190513(월) - 전현승

경북대학교 알고리즘 문제해결 동아리 GURI

목차

- 수학 기초
  - GCD, LCM
  - 소수, 에라토스테네스의 체
- 자료구조 기초
  - pair, tuple
  - stack, queue, deque
  - set, map

기초 수학

- 알고리즘 하다가 웬 수학?
- PS/CP(Competitive Programming) 에서 필요한 수학

- GCD, LCM부터 FFT, 키르히호프 정리, 게임이론 등등…
- https://algospot.com/wiki/read/알고리즘_대회에_필요한_수학
- 이렇게 많다

- 오늘은 정말 기초적인 것들만

GCD, LCM

- **GCD** (Greatest Common Divisor) : **최대공약수**
- a와 b의 GCD == a와 b의 공통된 약수 중 가장 큰 수

- 브루트포스한 방법: 2부터 min(a, b)까지 모두 나눠보면 되는가?
  - i가 a의 약수이면서 b의 약수인가? 매번 체크 → O(min(a, b))

```
1 for (int i = 2; i <= min(a, b); i++) {
2     if (a % i == 0 && b % i == 0) {
3         gcd = i;
4     }
5 }
```

GCD, LCM

- 더 좋은 방법 : **유클리드 호제법**
- **GCD(a, b)는 GCD(b, a%b)와 같다.**
- GCD(20, 12) == GCD(12, 8) == GCD(8, 4) == GCD(4, 0) == 4

- 시간복잡도? **O(lg(min(a, b))** → 로그 시간으로 줄었다.

```
1 int gcd(int a, int b) {
2     if (b == 0) {
3         return a;  // b == 0까지 왔을 때, 그 때 a가 GCD
4     } else {
5         return gcd(b, a % b);
6     }
7 }
```

GCD, LCM

- **LCM** (Least Common Multiple) : **최소공배수**
- a와 b의 LCM == a와 b의 공통된 배수 중 가장 작은 수

- GCD를 먼저 구해야 LCM을 구할 수 있다
- LCM(a, b) = (a * b) / GCD(a, b)

소수, 에라토스테네스의 체

- 소수 : 1과 자기 자신으로밖에 나누어지지 않는 **1 이외의 정수**

- 어떠한 수가 소수인지 판별하려면?
- 브루트포스한 방법 : 2부터 n-1까지 모두 나눠보기? → O(N)

```
1 bool isPrime(int n) {
2     for (int i = 2; i <= n - 1; i++) {
3         if (n % i == 0) {
4             return false;
5         }
6     }
7     return true;
8 }
```

소수, 에라토스테네스의 체

- n = a * b라 하면, a와 b 둘 중 하나는 sqrt(n) 보다 작거나 같다.
  - (a = sqrt(n), b = sqrt(n)일 때가 최대)
- n-1까지 안 가고 sqrt(n)까지 체크해보면 된다. → O(sqrt(N))

```
1 bool isPrime(int n) {
2     for (int i = 2; i * i <= n; i++) {
3         if (n % i == 0) {
4             return false;
5         }
6     }
7     return true;
8 }
```

소수, 에라토스테네스의 체

- 특정 수가 소수인지 아닌지 판별은 했는데
- 범위 내의 모든 소수를 구하려면? → **에라토스테네스의 체**

1. 범위 내의 모든 정수를 쓴다.
2. 아직 지워지지 않은 수 중 가장 작은 수를 찾는다.
3. 그 수는 소수가 되고, 그 수의 배수를 모두 지운다.
4. 이를 맨 마지막 수까지 반복

소수, 에라토스테네스의 체

> 일단 1부터 50까지 숫자를 쭉 쓴다.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

소수, 에라토스테네스의 체

> 일단 1을 지우자.(1은 소수도, 합성수도 아니며, 기초수라고 해서 별도로 분류한다.)

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

소수, 에라토스테네스의 체

> 2를 제외한 2의 배수를 지우자.

	2	3		5		7		9	
11		13		15		17		19	
21		23		25		27		29	
31		33		35		37		39	
41		43		45		47		49	

소수, 에라토스테네스의 체

> 4의 배수는 지울 필요 없다.(2의 배수에서 이미 지워졌다.) 2,3 다음으로 남아있는 가장 작은 수, 즉 5의 배수를 5를 제외하고 지우자.

그리고 마지막으로 7의 배수까지 지워제끼면,[1]

	2	3		**5**		**7**			
11		13				17		19	
		23						29	
31						37			
41		43				47			

이런 식으로 남은 것들의 2배수, 3배수,...n배수를 지우다보면 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47이 남는다. 이것이 50미만의 소수이다.

소수, 에라토스테네스의 체

```
// 1부터 100까지 소수 구하기
bool erased[101] = {false, };
for (int i = 2; i <= 100; i++) {
    if (!erased[i]) {  // 아직 안 지워졌으면 그 수는 소수
        cout << i << '\n';
        for (int j = i; j <= n; j += i) {
            erased[j] = true;
        }
    }
}
```

- 시간복잡도 : O(N * lg(lg(N)) (거의 선형 시간)

자료구조

- 대부분의 언어에서 자료구조 구현체를 지원한다
  - C++ : STL, Python : collections, …
- 따라서 직접 구현할 필요 없음

- 뭘로 구현되어 있고, 어떻게 작동하는지 정도만 기억하고
- 대회나 문제풀이 때 빠르게 응용할 줄 알면 됨

- 자료구조 자체에 대한 자세한 설명은 생략. 사용법 위주로 설명
- C++ STL 컨테이너 기준으로 설명

pair, tuple

- pair : 2개의 데이터를 묶어서 관리할 수 있는 자료형

```
#include <utility>  // pair 사용 시 utility 헤더 필수
using namespace std;

pair<int, int> p;  // pair<자료형1, 자료형2> 이름; 형식으로 선언 (자료형 동일할 필요 X)

p = make_pair(4, 6);  // (4, 6)을 p1에 저장
cout << p.first << ‘ ’ << p.second;
// .first, .second로 원소 접근 가능 (여기서는 “4 6” 출력)

p.first = 46; p.second = 100; // 직접 값 수정 가능
```

pair, tuple

- 비교 연산 시 first로 먼저 결정, first가 동일할 시 second로 결정

```
pair<int, int> p[5];  // pair<int, int>형 5개짜리 배열 선언

p[0] = make_pair(100, 1);
p[1] = make_pair(10, 120);
p[2] = make_pair(5, 120);
p[3] = make_pair(5, 1);
p[4] = make_pair(50, -100);
sort(p, p + 5);  // 기본 비교연산으로 pair<int, int>형 배열을 정렬하면?
for (int i = 0; i < 5; i++) cout << p[i].first << ‘,’ << p[i].second << ‘ ’;
// 출력 : 5,1 5,120 10,120 50,-100 100,1
```

pair, tuple

- Tuple : 2개 이상의 데이터를 묶을 수 있다

```
#include <tuple>  // tuple 헤더 필수
using namespace std;

tuple<int, string, double> tp;
tp = make_tuple(46, “Best gardener”, 4.6);
int first = get<0>(tp);
string second = get<1>(tp);
double third = get<2>(tp);

tie(alt1, alt2, alt3) = tp;
```

stack, queue, deque

- 스택 (Stack) : 구멍 막힌 통 (First In Last Out)

```
#include <stack>  // stack 사용 시 헤더 include 필수
using namespace std;

stack<int> st;  // stack<자료형> 이름; 형식으로 선언

st.push(1);  // 스택에 1 추가
st.push(2);  // 스택에 2 추가
int a = st.top();  // 스택의 제일 위에 있는 값 반환
st.pop();  // 스택의 제일 위에 있는 값 삭제
cout << st.size();  // 스택의 크기 (데이터 수) 반환
if (st.empty()) cout << “스택이 비었다!”;  // 스택이 비었으면 true, 아니면 false
```

stack, queue, deque

- 큐 (Queue) : 구멍 뚫린 통 (First In First Out)

```
#include <queue>  // queue 사용 시 헤더 include 필수
using namespace std;

queue<int> q;  // queue<자료형> 이름; 형식으로 선언

q.push(1);  // 큐에 1 추가
q.push(2);  // 큐에 2 추가
int a = q.front();  // 큐의 제일 앞에 있는 값 반환
q.pop();  // 큐의 제일 앞에 있는 값 삭제
cout << q.size();  // 큐의 크기 (데이터 수) 반환
if (q.empty()) cout << “큐가 비었다!”;  // 큐가 비었으면 true, 아니면 false
```

stack, queue, deque

- 덱 (Deque = Double-ended Queue)
- 말 그대로 양쪽에서 push, pop 모두 가능한 큐

- deque<int> dq;
- dq.push_front(1);
- dq.push_back(2);
- dq.pop_front();
- dq.pop_back();
- dq.front();
- dq.back();
- 다른 건 stack, queue랑 거의 동일

set, map

- 양이 너무 많아져서 여기서 끊어야 할 듯
- 대충 뭐고 어떻게 쓰는지 정도만 알아보고
- 응용문제는 언젠가 있을 자료구조 2 스터디 때 같이 풀어보는 걸로

set, map

- set : 말 그대로 집합. 원소 중복 안 되는 정렬된 배열
- 본질은 균형 이진 탐색 트리
- 원소 삽입, 삭제, 접근 등 O(lgN)

set, map

```
#include <set>  // set 헤더 필수
using namespace std;

set<int> s;  // int형 set 선언

s.insert(10);  // 자료 삽입
s.insert(10);
s.insert(50);
s.insert(20);
s.insert(50);
s.insert(30);
// s의 원소 : [10, 20, 50]
```

set, map

- map : 해시 테이블 (key, value로 이루어진 연관 배열)
  - Python의 딕셔너리와 동일
- 본질은 균형 이진 탐색 트리
- 원소 삽입, 삭제, 접근 등 O(lgN)

set, map

```
#include <map>  // map 헤더 필수
using namespace std;

map<string, int> mp;  // key를 string으로, value를 int로 하는 map 선언

mp[“Lunasa”] = 1  // 자료 삽입
mp[“Merlin”] = 2
mp[“Lyrica”] = 3

cout << mp[“Merlin”];  // 출력 : 2
```

참고

- https://opentutorials.org/course/1685/9533
- https://namu.wiki/w/에라토스테네스의 체

끝