알고리즘
문제 해결 시작하기, 완전탐색

Nexters

알고리즘 관련 사이트

- acmicpc.net
- algospot.com
- train.usaco.org
- topcoder.com
- euler.synap.co.kr
- algospot.com

문제 해결 과정

- 문제를 읽고 이해하기
- 재정의와 추상화
- 계획 세우기
- 계획 검증하기
- 계획 수행하기
- 회고하기

-> 문제를 풀지 못할 때 너무 매달려있지 않는다.

체계적인 접근을 위한 질문들

- 비슷한 문제를 풀어 본적이 있을까
- 단순한 방법에서 시작 할 수 있을까
- 문제를 푸는 과정을 수식화 할 수 있을까
- 문제를 단순화 할 수 있을까
- 그림으로 그려 볼 수 있을까
- 수식으로 표현할 수 있을까
- 문제를 분해 할 수 있을까
- 뒤에서부터 생각해서 문제를 풀 수 있을까
- 순서를 강제 할 수 있을까
- 특정형태의 답만 고려 할 수 있을까

코딩과 디버깅

- 좋은 코드를 짜기 위한 원칙
- 간결한 코드 작성하기
- 적극적으로 코드 재사용하기
- 표준 라이브러리 공부하기 (문자열, 동적 배열, 스택, 큐, 리스트, 사전)
- 항상 같은 형태로 프로그램을 작성하기
- 일관적이고 명료한 명명법 사용하기
- 자료를 정규화해서 저장하기
- 코드와 데이터 분리하기

자주 하는 실수 1

- 산술 오버플로
- 배열 범위 밖의 원소 접근
- 일관되지 않은 범위 표현 방식

  a[n] 0<= i <n
- Off-by-one 오류
- 컴파일러가 잡아주지 못하는 상수 오타
- 스택 오버플로
- 다차원 배열 인덱스 순서 바꿔 쓰기

자주 하는 실수 2

- 잘못된 비교 함수 작성
- 비반사성: a<a 는 항상 거짓
- 비대칭성: a<b 가 참이면 b<a는 거짓
- 전이성: a<b가 참이고 b<c가 참이면 a<c
- 상등관계의 전이성: a<b와 b<a 가 모두 거짓이면 a 와 b는 같은 값으로 간주 a와b가 같고 b와 c가 같으면 a와 c가 같음

자주 하는 실수 3

- 최소, 최대 예외 잘못 다루기

```
bool isPrime(int n) {
 if(n%2 ==0) return false;
 for(int i =0; i < n; ++i)
    if(n % i == 0)
        return false;
 return true;
}
```

자주 하는 실수 4

- 연산자 우선순위
  if(b&1 == 0)
  if(b& (1==0))
- 너무 느린 입출력 방식
- 변수 초기화 문제

디버깅과 테스트

- 작은 입력에 대해 제대로 실행되나 확인하기
- 프로그램 계산 중간결과 출력하기

변수 범위의 이해

- 오버플로 피해가기
- 자료형의 프로모션

```
unsigned char a = 17;
short b = -18;
int c = 2;
unsigned int d = 0;
cout << (a+b)*c+d ;
```

✓Standard Conversions Link
https://msdn.microsoft.com/en-us/library/aetzh118.aspx

실수 자료형 이해

- 실수 연산의 어려움
- 실수와 근사값
- 실수의 이진법 표기
- 부동 소수점 표기
- 실수 비교하기

1. 비교한 실수의 크기들에 비례한 오차 한도를 설정한다
2. 상대 오차를 이용한다
3. 실수 연산 아예 안 하기

시간복잡도 분석

- **Θ** 표기 – 점근적으로 정확한 한계

- **Θ**(g(n))= { f(n): 모든 n >= n0 에 대해 0<=c1(gn)<=f(n)<= c2g(n) 인 양의 상수 c1, c2, n0 이 존재한다}

- **O** 표기 – 점근적 상한

- **O**(g(n))= { f(n): 모든 n >= n0 에 대해 0<=f(n)<= cg(n) 인 양의 상수 c, n0 이 존재한다}

- **Ω** 표기 – 점근적 하한

- **Ω**(g(n))= { f(n): 모든 n >= n0 에 대해 0<=cg(n)<=f(n) 인 양의 상수 c, n0 이 존재한다}

완전탐색

- Brute–force 모든 경우의 수를 나열 하여 답을 찾는 방법을 의미
- 순열 또는 조합 같은 형태의 문제에서 많이 사용한다.
- 재귀 호출의 형태로 문제를 해결 할 수 있다.

문제 (Picnic)

- 학생들과 가능한 싹이 주어졌을 때 짝을 지어 줄 수 있는 방법의 수를 출력
- 중복에 유의하여 짝을 지을 수 있는 모든 경우의 수를 체크 한다.

✓문제

https://algospot.com/judge/problem/read/PICNIC

✓풀이코드

https://github.com/Nexters/algorithmStudy/blob/master/seokjoong/Chapter06/Picnic.cpp

문제 (BoardCover)

- 게임 판이 주어졌을 때 게임 판을 덮을 수 있는 모든 방법을 찾아내는 것

- 빈 공간을 순회하며 가능한 모든 모양으로 게임 판을 모두 덮는 방법을 탐색한다.

✓문제

https://algospot.com/judge/problem/read/BOARDCOVER

✓풀이코드

https://github.com/Nexters/algorithmStudy/blob/master/seokjoong/Chapter06/Boardcover.cpp

문제(CLOCKSYNC)

- 연결되어있는 시계들을 모두 12시에 맞출 수 있는 최소의 스위치 누름 횟수를 구함

- 재귀 호출을 이용해 모든 스위치를 한번 씩 누르는 함수를 작성하고 맞출 수 있는 최소의 횟수를 구한다.

✓문제

https://algospot.com/judge/problem/read/CLOCKSYNC

✓풀이코드

https://github.com/Nexters/algorithmStudy/blob/master/seokjoong/Chapter06/Clocksync.cpp

참고자료

- 알고리즘 문제 해결전략
- Introduction to algorithms
- http://algospot.com