학과:사이버보안

학번:1871058

이름: LI YE

학습주제: 비트마스크(비트 연산), 다이나믹 프로그래밍(동적 계획법)

비트마스크(Beat Mask) 문제

비트 마스크의 개념

컴퓨터의 최소 연산 단위는 bit입니다. bit는 이진수를 나타내기 위해 0과 1로만 이루어져있는데, 우리는 비트 연산을 통해 약간 Tricky하게 문제를 빠르게 해결할 수 있습니다.

1.비트 연산은 삽입, 삭제, 조회가 빠르다.

2.코드가 간결해진다. 뒤에서 살펴볼 예정이다.

3.정수 표현으로 DP 문제를 해결할 수 있다.

비트 연산

1) AND 연산(&)

대응하는 숫자가 모두 1이면 1 반환:

bin(0b1010011010 & 0b1101101100) # 0b1000001000

2) OR 연산(|)

대응하는 숫자에서 1이 한 개라도 있으면 1 반환 (only 1)

bin(0b1010011010 | 0b1101101100) # 0b1111111110

3) XOR 연산(^)

대응하는 숫자가 서로 다르면 1 반환

bin(0b1010011010 ^ 0b1101101100) # 0b111110110

4)NOT 연산(~)

비트값이 1이면 0으로, 0이면 1로 바꿈

bin(~0b0010 << 2) # 0b1101

5)SHIFT 연산 (>>, <<)

a << b는 a의 비트를 b칸 만큼 왼쪽으로 밀어 내는 것이고,

a >> b는 a의 비트를 b칸 만큼 오른쪽으로 밀어내는 것 입니다.

bin(0b0010 << 2) # 0b1000

bin(0b1100 >> 2) # 0b11

활용

1)원소 추가

n = 3

print(bin(0b0010 | (1 << n))) # 0b1010

2)원소 삭제

n = 3

print(bin(0b1010 & ~(1 << n))) # 0b10

3)원소 조회

n = 3

print(bin(0b1010 & (1 << n))) # 0b1000

4) 원소 역전

n = 3

print(bin(0b1010 ^ (1 << n))) # 0b10

연습

백준(12813) 이진수 연산

틀린 답안:

A = int(input(), 2)

B = int(input(), 2)

print(bin(A & B)[2:].zfill(100000)) # 10000 문자열 되돌리기

print( bin(A | B)[2:].zfill(100000))

print(bin(A ^ B)[2:].zfill(100000))

print(bin(~A)[2:].zfill(100000))

print(bin(~B)[2:].zfill(100000))

 ~ 연산 결과는 -(입력값 + 1)이다. (1을 더하고 음수로 변환)

정답:

A = int(input(), 2) # 입력받은 2진수를 int형 정수로 변환

B = int(input(), 2)

mask = 2 \*\* 100000 - 1

print(bin(A & B)[2:].zfill(100000))

print( bin(A | B)[2:].zfill(100000))

print(bin(A ^ B)[2:].zfill(100000))

print(bin(A ^ mask)[2:].zfill(100000))

print(bin(B ^ mask)[2:].zfill(100000))

2.백준(11723)집합

생각:

파이썬 에서는 set()이라는 함수를 사용해서 집합을 표현할 수 있습니다. 이 문제를 list자료형으로 해결하기보다 set자료형으로 해결하는 것이 시간 복잡도에 더욱 좋습니다.

풀이:

![텍스트, 영수증이(가) 표시된 사진

자동 생성된 설명]()![테이블이(가) 표시된 사진

자동 생성된 설명]()

3.백준(23253)자료구조

![텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명]()텍스트, 시계이(가) 표시된 사진

자동 생성된 설명![텍스트, 시계이(가) 표시된 사진

자동 생성된 설명]()

다이나믹 프로그래밍(동적 계획법)

다이나믹 프로그래밍은 메모리 공간을 약간 더 사용해서 연산 속도를 비약적으로 증가시키는 방법이다. 다이나믹 프로그래밍은 다음과 같은 조건을 만족할 때 사용할 수 있다.

1. 최적 부분 구조: 큰 문제를 작은 문제를 나눌 수 있다. 이러한 작은 문제의 답을 모아 큰 문제를 해결할 수 있다.

2. 중복된 하위 문제: 동일한 작은 문제를 반복적으로 해결해야 한다.

위 조건을 만족하는 대표적인 문제가 피보나치 수열 문제이다. 피보나치 수열은 다음과 같은 점화식을 만족하는 수열이다.

An = An-1 + An-2, A1 = 1, A2 = 1

피보나치 수열

다이나믹 프로그래밍으로 해결할 수 있는 대표적인 예시가 바로 피보나치 수열이다.

피보나치 수열은 현재 항을 계산하기 위해 이전 두항의 합을 구해야한다. 이를 표현하기 위해 다음과 같은 점화식을 사용한다.

F1 = F2 = 1

Fn = Fn-1 + Fn-2 (n∈{3,4,…})

1.백준(1463) 1로 만들기

생각

이 문제는 동적 계획법을 이용해 풀어야 합니다. 동적 계획법은 가능성을 모두 두고 그 안에 최솟값을 찾을 수 있습니다. 만약 조건문으로 풀게 되면 10 - > 5 -> 4 -> 2 -> 1 이지만, 10은 10 -> 9 -> 3 - > 1 , n/3보다 n-1을 먼저 해줘야 최소값이 나오게 된다.

식:dp(N) = min ( dp(N//3) +1, dp(N//2)+1, dp(N-1)+1)

f(i)=min(f(i//3)+1, f(i//2)+1, f(i)+1)

풀기:

n = int(input())

dp = [0 for i in range(n+1)] # d에 계산된 값을 저장. 1번째 수는 사실 d[1]이 아니고 d[2]이기 때문에, 계산하기 편하게 d[1]을 1번째 인 것 만들기

for i in range(2, n+1):

dp[i] = dp[i-1] + 1

if i%2 == 0 and dp[i] > dp[i//2] + 1 :

dp[i] = dp[i//2]+1

if i%3 == 0 and dp[i] > dp[i//3] + 1 : # 1을 더하는 것은 d는 결과가 아닌 계산한 횟수를 저장하는 것 이기 때문이다. d[i]에는 더하지 않는 이유는 이미 1을 뺄 때 1을 더해준 이력이 있기 때문이다.

dp[i] = dp[i//3] + 1

print(dp[n])

출력결과:



2.개미 전사

\*입력 조건

- 첫째줄에 식량창고의 개수 N이 주어진다.(3<=x<=100)

- 둘째줄에 공백으로 구분되어 각 식량창고에 저장된 식량의 개수 K가 주어진다.(0<=K<=1,000)

\*출력 조건

- 첫째줄에 개미 전사가 얻을 수 있는 식량의 최댓값을 출력하시오

\*입력예시

4

1 3 1 5

\*출력예시

8

해결방법:

반복적인 문제이기 때문에 동적 계획법으로 해결합니다.

**F[i] = max(F[j]) + 1 (i=1~n,j=1~i-1 있기 때문에**

f(i)=max(f(i-1), f(i-2)+k(i)) 얻다.

풀이:

![텍스트이(가) 표시된 사진

자동 생성된 설명]()![텍스트, 시계이(가) 표시된 사진

자동 생성된 설명]()

느낌 점

예전에는 비트마스크가 복잡하다고 생각했는데, 이진법이나 십진법 전환에 대한 부분은 기억하기 어려워서 활용할 수 없고 문제를 푸는 데 시간이 좀 오래 걸렸는데, 이런 부분을 좀 더 강화해야 될 것 같습니다.동적 계획법에 관해서는 이 방면을 깊이 이해하면 어려움이 있습니다.브루트 포스 이나 일반적인 욕심적 알고리즘과는 다라고 일반 깊이 검색보다 더 우수입니다. 국부 최적분해 유도에서 전체 최적분해로 가는 과정입니다. 국부 최적해 유도에서 전체 최적해로 가는 과정입니다.먼저 상태를 설정하고 상태를 변화한 방정식을 정확하게 써야 합니다.최적자구조의 존재를 정확하게 판단합니다.