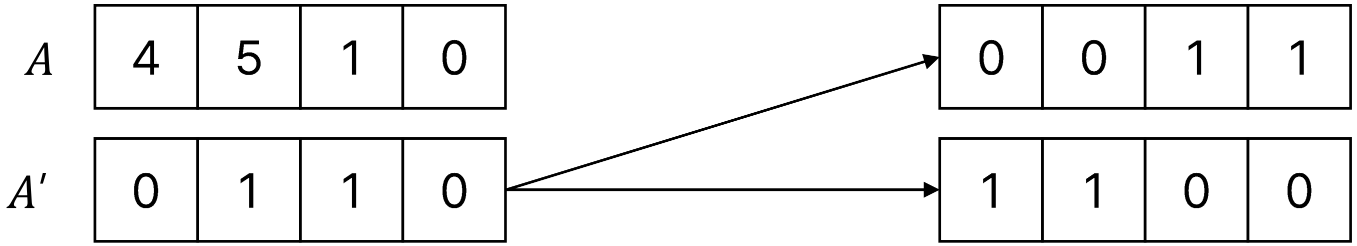
2주차 연습문제 풀이

피하자 BOJ 25379

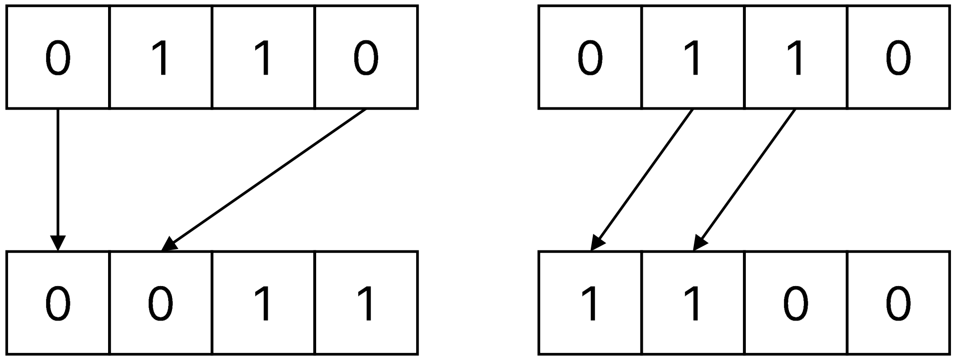
- 배열에 홀수와 짝수가 모두 존재하는 경우 짝수는 짝수끼리, 홀수는 홀수끼리 붙어 있어야 한다는 사실은 자명함
- 따라서 짝수면 0, 홀수면 1로 바꾸어도 동일한 문제



피하자 BOJ 25379

- 가능한 경우는 두가지, 각 경우마다 0 또는 1이 갈 자리는 정해져 있음
- 교환 한번으로 0 또는 1을 왼쪽으로 한 칸 이동할 수 있음
- 답이 32비트 정수 범위를 넘어갈 수 있으므로 주의

```cpp
ll idx0 = 0, idx1 = 0, ans0 = 0, ans1 = 0;
for (int i = 0; i < n; i++) {
    if (a[i] & 1) ans1 += i - idx1++;
    else ans0 += i - idx0++;
}
cout << min(ans0, ans1) << '\n';
```



질문?

다이나믹 프로그래밍

다이나믹 프로그래밍

- 엄청나게 많은 문제들이 DP로 풀림

- 경우의 수를 구하는 문제
- 최적해를 구하는 문제

다이나믹 프로그래밍

- **전체문제**를 **부분문제**로 나눠서 생각하는 방법 – 점화식
- 중복 연산을 피하는 방법 – 메모이제이션

- 문제를 통해 알아보자

피보나치 수 2 BOJ 2748

$$F_n = \begin{cases} 0 & (n = 0) \\ 1 & (n = 1) \\ F_{n-1} + F_{n-2} & (n \geq 2) \end{cases}$$

피보나치 수 2 BOJ 2748

- 재귀적으로 작성

```
ll fibo(int n) {
    if (n <= 1) return n;
    return fibo(n - 1) + fibo(n - 2);
}
```

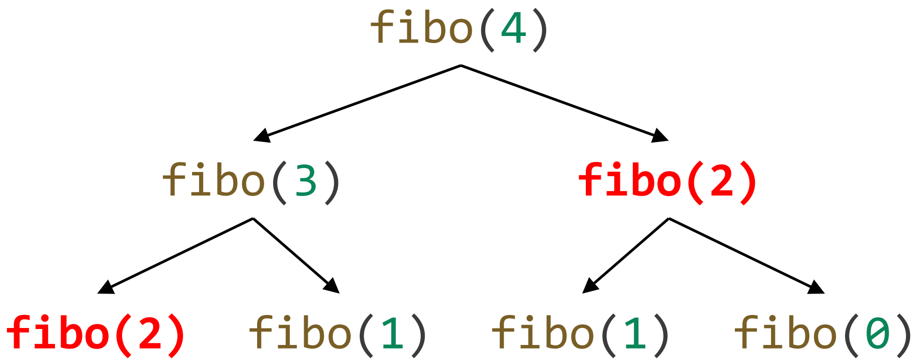
$$T(n) \in O(1.62^n)$$

시간 초과

피보나치 수 2 BOJ 2748

- 무엇이 문제일까?

```
ll fibo(int n) {
    if (n <= 1) return n;
    return fibo(n - 1) + fibo(n - 2);
}
```



- 이미 답을 구한 부분문제의 답을 다시 구하고 있음
- 한 번 구한 부분문제의 답은 바뀌지 않음 (F_2의 값은 항상 1임)
- 답을 구하면 저장했다가 필요할 때 저장한 값을 사용하면 어떨까?

피보나치 수 2 BOJ 2748

- 다음과 같이 값을 저장하여 중복 연산을 피하는 방법을 메모이제이션(memoization)이라고 함

```
bool vis[91];
ll dp[91];
ll fibo(int n) {
    if (n <= 1) return n;
    if (vis[n]) return dp[n];
    dp[n] = fibo(n - 1) + fibo(n - 2);
    vis[n] = true;
    return dp[n];
}
```


맞았습니다!!

Top-down과 Bottom-up

- 전체문제를 부분문제로 나누어서 재귀적으로 구하면 Top-down 방식,
  가장 작은 부분문제부터 채워넣으면 Bottom-up 방식이라고 부름

```cpp
bool vis[91];
ll dp[91];
ll fibo(int n) {
    if (n <= 1) return n;
    if (vis[n]) return dp[n];
    dp[n] = fibo(n - 1) + fibo(n - 2);
    vis[n] = true;
    return dp[n];
}
```

Top-down 방식

```cpp
ll dp[91];
ll fibo(int n) {
    dp[0] = 0; dp[1] = 1;
    for (int i = 2; i <= n; i++)
        dp[i] = dp[i - 1] + dp[i - 2];
    return dp[n];
}
```

Bottom-up 방식

Top-down과 Bottom-up

Top-down	Bottom-up
점화식이 복잡할 때 생각하기 편함	점화식이 복잡하면 코드로 구현하기 어려움
재귀 호출로 인해 실행속도가 느려짐	재귀 호출이 없어서 빠르게 작동함
점화식이 간단할 때 코딩량이 비교적 많아짐	점화식이 간단할 때 빠르게 코딩하기 유용함

질문?

구간 합 구하기 4 BOJ 11659

구간 합 구하기 4 BOJ 11659

- 문제에 나와있는 그대로 구현하면 $O(NM)$으로 시간 초과
- 고등학교에서 배웠던 수열의 합을 떠올려 보자

- $a_n = S_n - S_{n-1}$

그렇다면 $a_i + a_{i+1} + \cdots + a_{j-1} + a_j$는?

구간 합 구하기 4 BOJ 11659

- $a_i + a_{i+1} + \cdots + a_{j-1} + a_j = S_j - S_{i-1}$

처음에 S_i의 값들을 미리 저장해 놓는다면?

$O(N + M)$ 시간에 해결 가능

	[1]	[2]	[3]	[4]	[5]
a	5	4	3	2	1
S	5	9	12	14	15

질문?

가장 긴 증가하는 부분 수열 BOJ 11053

- 부분 수열?

가장 긴 증가하는 부분 수열 BOJ 11053

- dp_i를 A_i까지 봤을 때 A_i를 포함하는 LIS의 길이로 정의
- LIS는 가장 긴 증가하는 부분 수열을 뜻함

- 인덱스는 0부터 시작한다고 가정

$$\mathrm{dp}_i = \max\left(\left\{\mathrm{dp}_j + 1 \mid 0 \le j < i \text{ and } A_j < A_i\right\} \cup \{1\}\right)$$

가장 긴 증가하는 부분 수열 BOJ 11053

- dp 배열을 구한 뒤 최댓값을 출력하면 됨
- dp 배열의 가장 마지막 값을 출력하면 오답

A	10	20	10	30	20	50
dp	1	2	1	3	2	4

가장 긴 증가하는 부분 수열 BOJ 11053

- 이 풀이의 시간복잡도는 $O(n^2)$

더 빠른 풀이는 없을까?

다이나믹 프로그래밍

- 상태를 잘 정의하는 것이 중요
- 현재 상태를 이전 상태들로 표현하는 방법을 생각
- 잘 풀리지 않거나 제한 시간 안에 풀기 힘들다는 생각이 들면 다른 방법으로 상태를 표현하는 방법이 없는지 고민

코테 수준의 DP 문제들은 많이 풀어서 유형을 익히면 대부분 풀 수 있음

자주 등장하는 상태 정의

- i번 인덱스를 선택했을 때 최댓값/최솟값/경우의 수
- i번 인덱스까지만 고려했을 때 최댓값/최솟값/경우의 수
- i번 인덱스까지만 고려했을 때 j를 만들 수 있는 최댓값/최솟값/경우의 수
- i번 인덱스까지 중에서 j개를 골랐을 때 최댓값/최솟값/경우의 수

질문?

LCS BOJ 9251

- 최장 공통 부분 수열

ACAYKP
CAPCAK

LCS BOJ 9251

- 문자열 A와 문자열 B의 LCS를 구하는 문제

상태를 어떻게 정의할까?

LCS BOJ 9251

- A의 i번째 문자까지, B의 j번째 문자까지 봤을 때 정답을 $\mathrm{dp}_{i,j}$로 정의

$$\mathrm{dp}_{i,j} = \begin{cases} \mathrm{dp}_{i-1,j-1} + 1 & (A_i = B_j) \\ \mathrm{dp}_{i-1,j}\text{와 } \mathrm{dp}_{i,j-1} \text{ 중 작지 않은 값} & (A_i \neq B_j) \end{cases}$$

LCS BOJ 9251

	A	C	A	Y	K	P
C	0	1	1	1	1	1
A	1	1	2	2	2	2
P	1	1	2	2	2	3
C	1	2	2	2	2	3
A	1	2	3	3	3	3
K	1	2	3	3	4	4

LCS BOJ 9251

- 문자열 A의 길이를 N, 문자열 B의 길이를 M으로 정의
- 이 풀이의 시간복잡도는 $O(NM)$

더 빠른 풀이는 없을까?

이 문제는 LCS의 길이만 구하면 되는데, LCS를 직접 구할 수는 없을까?

질문?

평범한 배낭 BOJ 12865

- Knapsack 문제라고 부름

평범한 배낭 BOJ 12865

- 상태 정의를 어떻게 할까?

dp_i를 i번째 물건까지 고려했을 때 넣을 수 있는 물건들의 가치합의 최댓값으로 정의

평범한 배낭 BOJ 12865

- dp_i를 i번째 물건까지 고려했을 때 넣을 수 있는 물건들의 가치합의 최댓값으로 정의

점화식을 생각해보자

평범한 배낭 BOJ 12865

- dp_i를 i번째 물건까지 고려했을 때 넣을 수 있는 물건들의 가치합의 최댓값으로 정의

$$\mathrm{dp}_i = \mathrm{dp}_{i-1} + V_i$$

그렇다면 답은 dp_N임을 알 수 있다

평범한 배낭 BOJ 12865

- dp_i를 i번째 물건까지 고려했을 때 넣을 수 있는 물건들의 가치합의 최댓값으로 정의

$$\mathrm{dp}_i = \mathrm{dp}_{i-1} + V_i$$

그렇다면 답은 dp_N임을 알 수 있다

과연 그럴까?

평범한 배낭 BOJ 12865

- dp_i를 i번째 물건까지 고려했을 때 넣을 수 있는 물건들의 가치합의 최댓값으로 정의

현재 구한 것만으로는 무게가 K를 초과했는지 알 수 없음

dp를 다시 정의해보자

평범한 배낭 BOJ 12865

- $\mathrm{dp}_{i,j}$를 i번째 물건까지 고려했을 때 무게가 j 이하가 되도록 넣을 수 있는 물건들의 가치합의 최댓값으로 정의

$$\mathrm{dp}_{i,j} = \begin{cases} \mathrm{dp}_{i-1,j} & (W_i > j) \\ \mathrm{dp}_{i-1,j}\text{와 } \mathrm{dp}_{i-1,j-W_i} + V_i \text{ 중 작지 않은 값} & (W_i \leq j) \end{cases}$$

평범한 배낭 BOJ 12865

- $\mathrm{dp}_{i,j}$를 i번째 물건까지 고려했을 때 무게가 **정확히** j가 되도록 넣을 수 있는 물건들의 가치합의 최댓값으로 정의해도 점화식은 동일하게 나오는데, 무엇이 다를까?

$$\mathrm{dp}_{i,j} = \begin{cases} \mathrm{dp}_{i-1,j} & (W_i > j) \\ \mathrm{dp}_{i-1,j}\text{와 } \mathrm{dp}_{i-1,j-W_i} + V_i \text{ 중 작지 않은 값} & (W_i \le j) \end{cases}$$

질문?

선형 점화식

다음을 만족하는 양의 정수 k가 존재하면

$$\mathrm{dp}_n = \sum_{i=1}^{k} (c_i \times \mathrm{dp}_{n-i})$$

위 점화식은 선형 점화식이라고 한다. (단, c_i는 상수, $c_k \neq 0$)

- $F_n = F_{n-1} + F_{n-2}$
- $P_n = P_{n-1} + P_{n-5}$

모든 선형 점화식은 행렬로 나타낼 수 있다

$$
\begin{aligned}
F_n &= 1 \times F_{n-1} + 1 \times F_{n-2} \\
F_{n-1} &= 1 \times F_{n-1} + 0 \times F_{n-2}
\end{aligned}
$$

$$
\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix}
$$

$$
\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}
$$

분할 정복을 이용한 행렬 거듭제곱

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{13} = \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{1} \times \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{4} \times \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{8}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{2} = \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{1} \times \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{1}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{4} = \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{2} \times \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{2}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{8} = \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{4} \times \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}^{4}$$

분할 정복을 이용한 거듭제곱

```
ll fexp(ll a, ll b, ll p) {
    ll res = 1;
    for (; b > 0; b >>= 1) {
        if (b & 1) (res *= a) %= p;
        (a *= a) %= p;
    }
    return res;
}
```

$a^b \bmod p$를 $O(\log b)$ 시간에 계산하는 코드

분할 정복을 이용한 행렬 거듭제곱

- 같은 방식으로 구현하면 됨

행렬 곱셈을 할 줄 모른다면?

행렬 곱셈 BOJ 2740

질문?

연습문제

1149 : RGB거리
1463 : 1로 만들기
1520 : 내리막 길
1915 : 가장 큰 정사각형
1958 : LCS 3
2133 : 타일 채우기
2225 : 합분해
2248 : 이진수 찾기
2293 : 동전 1
2294 : 동전 2

연습문제

2579 : 계단 오르기
2618 : 경찰차
5626 : 제단
9184 : 신나는 함수 실행
9252 : LCS 2
10830 : 행렬 제곱
11660 : 구간 합 구하기 5
24464 : 득수 밥 먹이기
30460 : 스위치
30859 : M. S. I. S.