

Chapter 04.

동적 계획법 #3

Clip 01 | [9251] LCS

최장 공통 부분 수열의 길이

Clip 02 | [9252] LCS2

최장 공통 부분 수열의 길이와 경로

Clip 03 | [13398] 연속합 2

최대가 되는 연속합과 상태 관리

Clip 04 | [2228] 구간 나누기

구간합과 분할 좌표 고르기

Clip 05 | [15822] Ah-Choo!

최소 오차와 케이스 나누기

Clip 06 | [11066] 파일 합치기

구간합과 최소 비용

Clip 07 | [10942] 팰린드롬?

팰린드롬의 성질과 동적계획법

Clip 08 | [1915] 가장 큰 정사각형

기준점을 이용한 최대 길이 찾기

Ch04. 동적계획법 #3

1. [9251] LCS

BOJ9251: LCS

문제 요약

- Longest Common Subsequence
최장 공통 부분 수열의 길이를 구하는 문제

BOJ9251: LCS

문제 요약

입력 데이터
ACAYKP CAPCAK

출력 데이터
4

BOJ9251: LCS

문제 분석

Brute-force로 생각해보면?

ex) $X = \text{"ABB"}$, $Y = \text{"AABA"}$

1. X와 Y에 대한 모든 Subsequence를 구한다

2. 서로 일치하는 Subsequence중
가장 긴 길이를 찾는다

A
B
AB
BB
ABB

A
AA
AB
BB
BA
AAA
AAB
ABA
AAB
A

BOJ9251: LCS

문제 분석

$X = x_0x_1 \cdots x_{n-1}$ 일 때
Subsequence를 만드는 과정:

x_0 을 고르거나 x_0 을 고르지 않거나
 x_1 을 고르거나 x_1 을 고르지 않거나
...
 x_{n-1} 을 고르거나 x_{n-1} 을 고르지 않거나

$O(2^n)$ 의 시간 복잡도를 가진다

BOJ9251: LCS

문제 분석

- Brute-force로는 문제를 풀이할 수 없다
- 문제에서 주어진 공통 부분수열의 특징을 좀 더 분석해보자
- ex) $X = \text{"BANANA"}$, $Y = \text{"NAYANA"}$

BOJ9251: LCS

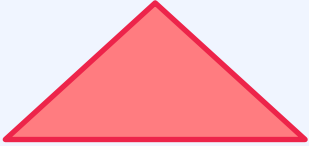

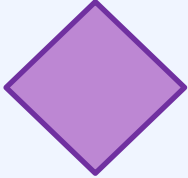
문제 분석

ex) $X = \text{"BANANA"}, Y = \text{"NAYANA"}$
 • $\text{LCS}(X, Y) = \text{"NANA"}$

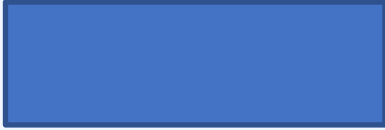
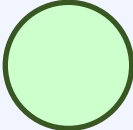
- 만약 LCS에서 ANA를 지우려고 한다면?
- $X' = \text{"BAN"}, Y' = \text{"NAY"}$
- 이 과정을 역으로 다시 생각해보자

BOJ9251: LCS

문제 분석

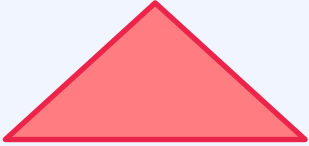

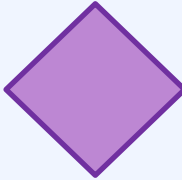
X:  Y:  LCS(X, Y): 

만약 두 X, Y 수열에 같은 수를 추가한다면?

X:   Y:   LCS(X, Y):  

BOJ9251: LCS

문제 분석

X:  Y:  LCS(X, Y): 


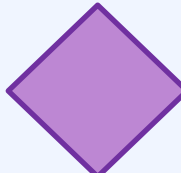
만약 두 X, Y 수열에 다른 수를 추가한다면?

X:   Y:   LCS(X, Y):  ?

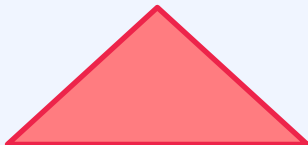


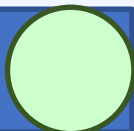
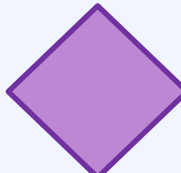

BOJ9251: LCS

문제 분석

case1. X의 마지막 수가 ★ 일수도 있다

X:  ★  Y:  ★ LCS(X, Y):  ★

case2. Y의 마지막 수가  일수도 있다

X:   Y:   ★ LCS(X, Y):  

BOJ9251: LCS

문제 분석

$X_n == Y_m$ 일때

$$LCS(X_n, Y_m) = LCS(X_{n-1}, Y_{m-1}) + 1$$

$X_n \neq Y_m$ 일때

$$\begin{aligned} &LCS(X_n, Y_m) \\ &= \max(LCS(X_{n-1}, Y_m), LCS(X_n, Y_{m-1})) \end{aligned}$$

BOJ9251: LCS

구현

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0					
A	0						
Y	0						
A	0						
N	0						
A	0						

BOJ9251: LCS

구현

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0				
A	0						
Y	0						
A	0						
N	0						
A	0						

BOJ9251: LCS

구현

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1			
A	0						
Y	0						
A	0						
N	0						
A	0						

BOJ9251: LCS

구현

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1		
A	0						
Y	0						
A	0						
N	0						
A	0						

BOJ9251: LCS

구현

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	
A	0						
Y	0						
A	0						
N	0						
A	0						

BOJ9251: LCS

구현

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0						
Y	0						
A	0						
N	0						
A	0						

BOJ9251: LCS

구현

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

BOJ9251: LCS

구현

```
for(int i = 1; i <= s1.length; i++) {  
    for(int j = 1; j <= s2.length; j++) {  
        if(s1[i - 1] == s2[j - 1]) {  
            dp[i][j] = dp[i - 1][j - 1] + 1;  
        }  
        else dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);  
    }  
}
```

BOJ9251: LCS

구현

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

Ch04. 동적계획법 #3

2. [9252] LCS2

BOJ9252: LCS2

문제 요약

- Longest Common Subsequence
최장 공통 부분 수열의 길이를 구하는 문제
- 그리고 그 수열도 함께 출력해야 한다 <- NEW

BOJ9252: LCS2

문제 분석

- 이전 클립에서 아래 점화식을 이용해 LCS의 길이를 구했다
- 원본 수열은 어떻게 알아낼 수 있을까?

$X_n == Y_m$ 일때

$$LCS(X_n, Y_m) = LCS(X_{n-1}, Y_{m-1}) + 1$$

$X_n \neq Y_m$ 일때

$$LCS(X_n, Y_m) = \max(LCS(X_{n-1}, Y_m), LCS(X_n, Y_{m-1}))$$

BOJ9252: LCS2

문제 분석

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

BOJ9252: LCS2

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

- 수열의 각 수가 같으면?
왼쪽 대각선 위에서 값을 가져왔다
- 서로 다르면 위/왼쪽
중 큰 값을 가져왔다

BOJ9252: LCS2

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

- 수열의 각 수가 같으면?
왼쪽 대각선으로 탐색한다
- 서로 다르면?
- 왼쪽과 위 중에 큰값으로 탐색한다

BOJ9252: LCS2

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

- 수열의 각 수가 같으면?
왼쪽 대각선으로 탐색한다
- 서로 다르면?
- 왼쪽과 위 중에 큰값으로 탐색한다

BOJ9252: LCS2

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

- 수열의 각 수가 같으면?
왼쪽 대각선으로 탐색한다
- 서로 다르다면?
- 왼쪽과 위 중에 큰값으로 탐색한다

BOJ9252: LCS2

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

- 수열의 각 수가 같으면?
왼쪽 대각선으로 탐색한다
- 서로 다르다면?
- 왼쪽과 위 중에 큰값으로 탐색한다

BOJ9252: LCS2

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

- 수열의 각 수가 같으면?
왼쪽 대각선으로 탐색한다
- 서로 다르다면?
- 왼쪽과 위 중에 큰값으로 탐색한다

BOJ9252: LCS2

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	2	2	2	2
Y	0	0	1	2	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

- 수열의 각 수가 같으면?
왼쪽 대각선으로 탐색한다
- 서로 다르다면?
- 왼쪽과 위 중에 큰값으로 탐색한다

BOJ9252: LCS2

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

- 수열의 각 수가 같으면?
왼쪽 대각선으로 탐색한다
- 서로 다르면?
- 왼쪽과 위 중에 큰값으로 탐색한다

BOJ9252: LCS2

	∅	B	A	N	A	N	A
∅	0	0	0	0	0	0	0
N	0	0	0	1	1	1	1
A	0	0	1	1	2	2	2
Y	0	0	1	1	2	2	2
A	0	0	1	1	2	2	3
N	0	0	1	2	2	3	3
A	0	0	1	2	3	3	4

- 왼쪽 대각선 위로 탐색?
- 서로 같은 수
- 해당 수를 배열에 담고 역순으로 출력한다

Ch04. 동적계획법 #3

3. [13398] 연속합 2

BOJ13398: 연속합 2

문제 요약

- N개의 정수로 이루어진 임의의 수열
- 연속된 수를 선택해서 가장 큰 합을 골라야 함
 - 단 수는 한 개 이상 선택
- 수열에서 수를 한 개 제거할 수 있다
 - 제거하지 않아도 된다

BOJ13398: 연속합 2

문제 분석

- 수열에서 수를 제거하는 조건이 문제를 어렵게 한다
- 우선 수를 제거하지 않고 연속합을 구해보자
 - [BOJ1912] 연속합

BOJ13398: 연속합 2

문제 분석

- 문제 요구사항: 연속된 수의 합 중 최대
 - $d[i] = \{i\}$ 번째 수까지 연속된 수의 최대 합
- 만약 이 문제에 음수가 존재하지 않는다면?
 - 무조건 $\{0\}$ 번째부터 $\{i\}$ 번째까지 모두 더하는게 크다
- 음수가 누적합에 어떤 영향을 끼치는 걸까?

BOJ13398: 연속합 2

문제 분석

[illegible]

BOJ13398: 연속합 2

문제 분석

수열

10

-4

3

1

5

6

-35

12

21

-1

누적합

10

6

0

0

0

0

0

0

0

0

BOJ13398: 연속합 2

문제 분석

수열

10	-4	3	1	5	6	-35	12	21	-1
----	----	---	---	---	---	-----	----	----	----

누적합

10	6	9	10	15	21	0	0	0	0
----	---	---	----	----	----	---	---	---	---

BOJ13398: 연속합 2

문제 분석

수열	10	-4	3	1	5	6	-35	12	21	-1
누적합	10	6	9	10	15	21	-14	0	0	0

여태까지 구한 누적합보다 큰 음수를 만나면?
구하던 누적합을 포기하고, 0부터 시작 하는게 낫다

BOJ13398: 연속합 2

문제 분석

수열	10	-4	3	1	5	6	-35	12	21	-1
누적합	10	6	9	10	15	21	0	0	0	0

여태까지 구한 누적합보다 큰 음수를 만나면?
구하던 누적합을 포기하고, 0부터 시작 하는게 낫다

BOJ13398: 연속합 2

문제 분석

수열

10	-4	3	1	5	6	-35	12	21	-1
----	----	---	---	---	---	-----	----	----	----

누적합

10	6	9	10	15	21	0	12	34	33
----	---	---	----	----	----	---	----	----	----

여태까지 구한 누적합보다 큰 음수를 만나면?
구하던 누적합을 포기하고, 0부터 시작 하는게 낫다

BOJ13398: 연속합 2

문제 분석

- 문제 요구사항: 연속된 수의 합 중 최대
 - $d[i] = \{i\}$ 번째 수까지 연속된 수의 최대 합
- 따라서 점화식은 누적합을 $d[]$ 에 계산하다가 음수가 되어버리면 포기하고, 0으로 리셋 한다
 - $d[i] = \max(d[i - 1], 0) + a[i];$

BOJ13398: 연속합 2

문제 분석

문제 요구사항: 연속된 수의 합 중 최대

문제	결과	메모리	시간	언어
1912	맞았습니다!!	108672 KB	808 ms	Java 11 / 수정

- 따라서 점화식은 누적합을 $d[]$ 에 계산하다가 음수가 되어버리면 포기하고, 0으로 리셋 한다
 - $d[i] = \max(d[i - 1], 0) + a[i];$

BOJ13398: 연속합 2

문제 분석

- 이제 다시 원래 문제로 돌아와서 생각을 해보자
- 이전의 연속합의 조건에서 1개의 수를 고르지 않는다는 선택지가 추가로 주어진다
- 동적배열에 추가 상태를 부여해서, 수를 고르는 경우를 표현해보자

BOJ13398: 연속합 2

문제 분석

- 문제 요구사항: 연속된 수의 합 중 최대
 - $d[0][*]$ = 수를 0개 삭제하고 구한 최대 연속 합
 - $d[1][*]$ = 수를 1개 삭제하고 구한 최대 연속 합
- $d[0][*]$ 는 이전과 같이 음수인지 판단하고 0으로 리셋 하면 된다
- $d[1][*]$ 은 어떻게 식을 유도할 수 있을까?

BOJ13398: 연속합 2

문제 분석

- $d[1][*]$ 은 어떻게 식을 유도할 수 있을까?
 - case 1. 이미 수를 삭제한 경우
누적합을 포기($d[0][i]$) 하지 않는 이상 $a[i]$ 를 안고 가야한다
$$d[1][i] = d[1][i-1] + a[i]$$
 - case 2. 아직 수를 삭제할 수 있는 경우
 $a[i]$ 번째 수를 고르지 않고 넘어갈 수 있다
$$d[1][i] = d[0][i-1]$$

$$\rightarrow d[i][1] = \max(d[1][i-1] + a[i], d[0][i-1])$$

BOJ13398: 연속합 2

구현

```
d[0][0] = a[0];  
int ans = d[0][0];  
for(int i = 1; i < n; i++) {  
    d[0][i] = Math.max(d[0][i - 1], 0) + a[i];  
    d[1][i] = Math.max(d[0][i - 1], d[1][i - 1] + a[i]);  
    ans = Math.max(ans, Math.max(d[0][i], d[1][i]));  
}
```

Ch04. 동적계획법 #3

4. [2228] 구간 나누기

BOJ2228: 구간 나누기

문제 요약

- N개의 정수로 이루어진 1차원 배열
 - $(1 \leq N \leq 100)$
- M개의 구간을 선택하고, 구간의 합이 최대가 되도록
 - 구간의 조건
 - 연속된 수로 이루어 져야함
 - 서로 다른 구간끼리 겹치거나 인접하면 안됨
 - M개의 구간이 정확하게 있어야 함

BOJ2228: 구간 나누기

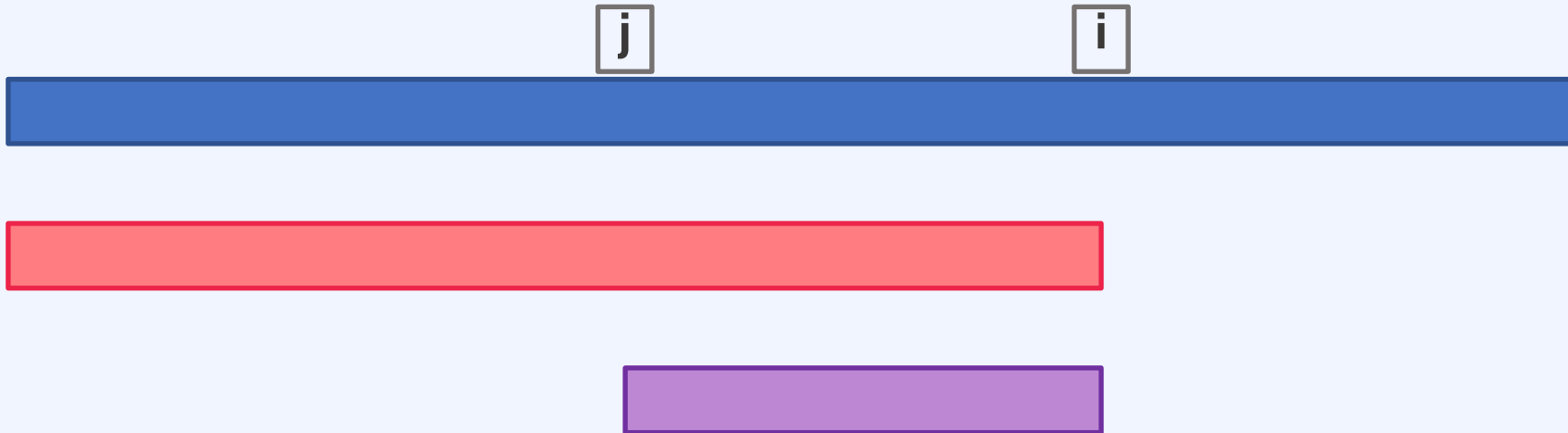
문제 분석

- 문제 요구사항: 구간의 속한 수의 최대 합
- 동적배열을 정의하기 전에, 연속된 구간에서 빠르게 합을 구하는 방법을 알아보자
 - 다른 유형의 문제에서도 자주 사용되는 방법이다

BOJ2228: 구간 나누기

문제 분석

- $[1, i]$ 구간의 합을 구하는 방법은 너무나도 쉽다

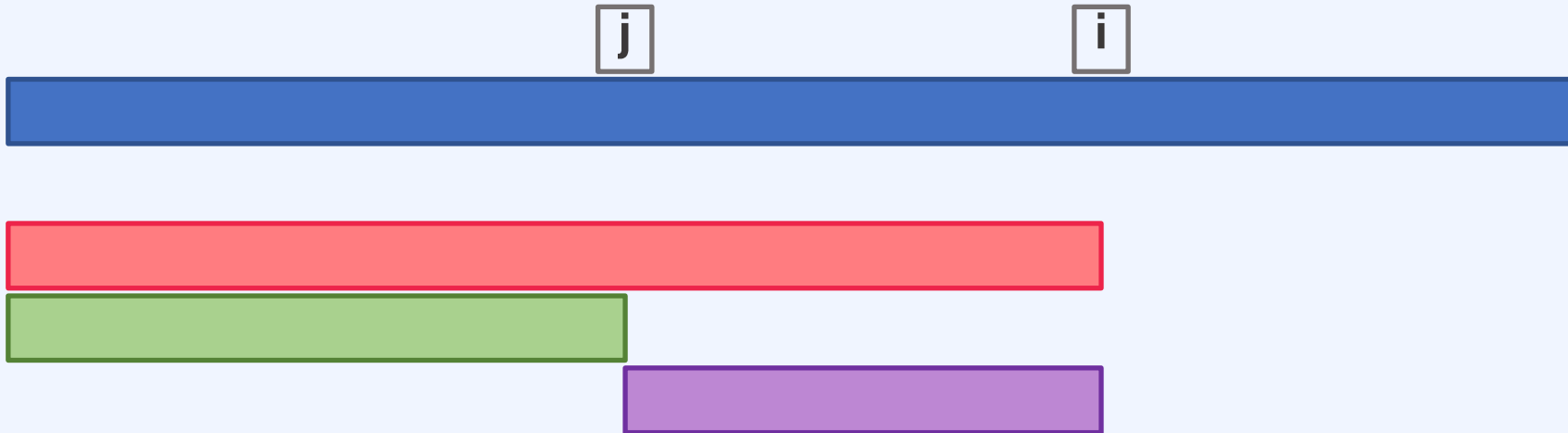


- $[j, i]$ 구간의 합을 구하는 방법은?

BOJ2228: 구간 나누기

문제 분석

- $[1, i]$ 구간의 합을 구하는 방법은 너무나도 쉽다



$$\text{sum}(j, i) = \text{sum}(1, i) - \text{sum}(1, j - 1)$$

BOJ2228: 구간 나누기

문제 분석

- 문제 요구사항: 구간의 속한 수의 최대 합
- 누적합 배열을 만들고, 구간 합 식을 함수로 작성하면 간단하고 빠르게 연속된 구간의 합을 구할 수 있다
- 이제 구간합을 자유롭게 사용할 수 있다고 가정하고 동적 배열을 고민해 보자

BOJ2228: 구간 나누기

문제 분석

- 문제 요구사항: 구간의 속한 수의 최대 합
 $d[n][m] = n$ 번째 수까지 m 개의 구간으로 나누어 구한 최대 합
- case1. n 번째 수를 아예 고르지 않는 경우
 - $d[n][m] = d[n-1][m]$

BOJ2228: 구간 나누기

문제 분석

- case2. 신규 구간을 $[i, n]$ 으로 할당한 경우
이때 $\{i\}$ 가 가능한 경우를 모두 확인해 봐야 한다



그렇다면 이전 구간은 어떻게 되는걸까?

BOJ2228: 구간 나누기

문제 분석

- case2. 신규 구간을 $[i, n]$ 으로 할당한 경우
그렇다면 이전 구간은 어떻게 되는 걸까?
- 구간 사이는 한 칸 이상 떨어져야 하므로 파란색 구간을
좌표로 나타내면 $[1, i-2]$ 로 표현할 수 있다



- 동적배열의 정의
 - $d[n][m]$ = n 번째 수까지 m 개의 구간으로 나누어 구한 최대 합

BOJ2228: 구간 나누기

문제 분석

- case2. 신규 구간을 $[i, n]$ 으로 할당한 경우
그렇다면 이전 구간은 어떻게 되는 걸까?
- 구간 사이는 한 칸 이상 떨어져야 하므로 파란색 구간을
좌표로 나타내면 $[1, i-2]$ 로 표현할 수 있다



- $d[i-2][m-1] = \{i-2\}$ 번째 수까지 $\{m-1\}$ 개의 구간으로
나누어 구한 최대 합
왜 $\{m-1\}$ 개 구간일까? → 신규구간 1개를 더하면 m 개

BOJ2228: 구간 나누기

문제 분석

- 정리하면 $d[n][m]$ 은 아래 case중에 최댓값 이다
- case1:
 $d[n-1][m]$
- case2:
 $(i = [1, n]) \quad d[i-2][m-1] + \text{sum}(i, n)$
- 단, $d[i-2][m-1]$ 가 계산되어 있을 거라는 보장은 없다
 - 재귀함수 + memoization으로 구현하면 된다

BOJ2228: 구간 나누기

구현

```
for(int i = 1; i <= n; i++) {  
    a[i] = sc.nextInt();  
    s[i] = s[i - 1] + a[i];  
}
```

```
static int sum(int[] s, int i, int j) {  
    return s[j] - s[i - 1];  
}
```

빠른 구간합을 위한 함수

BOJ2228: 구간 나누기

구현

```
static int calc(int n, int m) {  
    if(m == 0) return 0;  
    if(n <= 0) return -INF;  
    if(d[n][m] != -1) return d[n][m];  
    d[n][m] = calc(n - 1, m);  
    for(int i = 1; i <= n; i++) {  
        int temp = calc(i - 2, m - 1) + sum(s, i, n);  
        if(d[n][m] < temp) d[n][m] = temp;  
    }  
    return d[n][m];  
}
```

top down으로 내려가며 계산

case 1

case 2

Ch04. 동적계획법 #3

5. [15822] Ah-Choo!

BOJ15822: Ah-Choo!

문제 요약

- 길이가 N인 파형이 2개 주어진다
- 두개의 파형에 대해 DTW 기법으로 유사도를 측정한다
(기법의 내용은 문제에서 설명)
- 각 소리의 파형의 높이를 $X[]$ $Y[]$ 로 정의할 때,
최소오차 구하기

BOJ15822: Ah-Choo!

문제 분석

- $X[i]$ 와 $Y[j]$ 를 대응 시켰을 때 만들어지는 최소 거리를 $d[i][j]$ 로 정의해보자
- 어떤 유형이 만들어질까?



BOJ15822: Ah-Choo!

문제 분석

- case1.
 - $X[i]$ 와 $Y[i]$ 가 $X[i-1]$, $Y[j-1]$,
그리고 이전에 영역에 대응되지 않는 경우



BOJ15822: Ah-Choo!

문제 분석

- case2.
 - $X[i-1], Y[j]$ 에 대응되는 경우



BOJ15822: Ah-Choo!

문제 분석

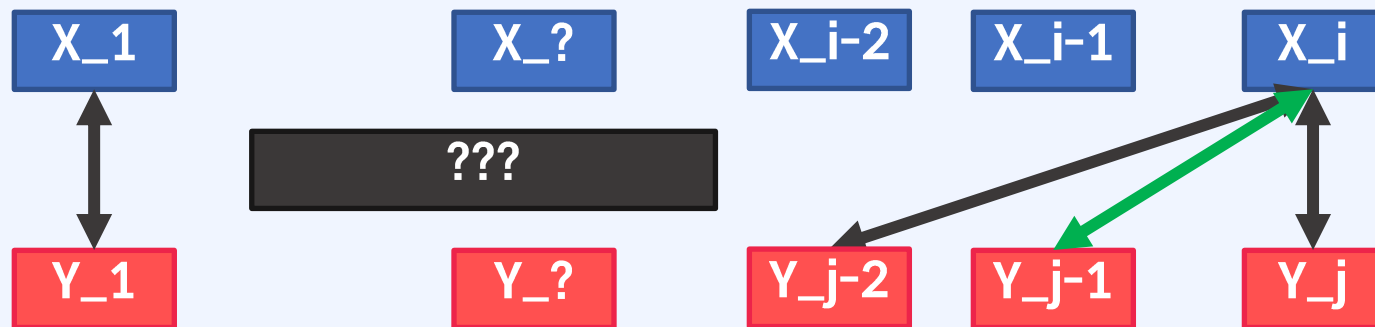
- case3.
 - $X[i], Y[j-1]$ 에 대응되는 경우



BOJ15822: Ah-Choo!

문제 분석

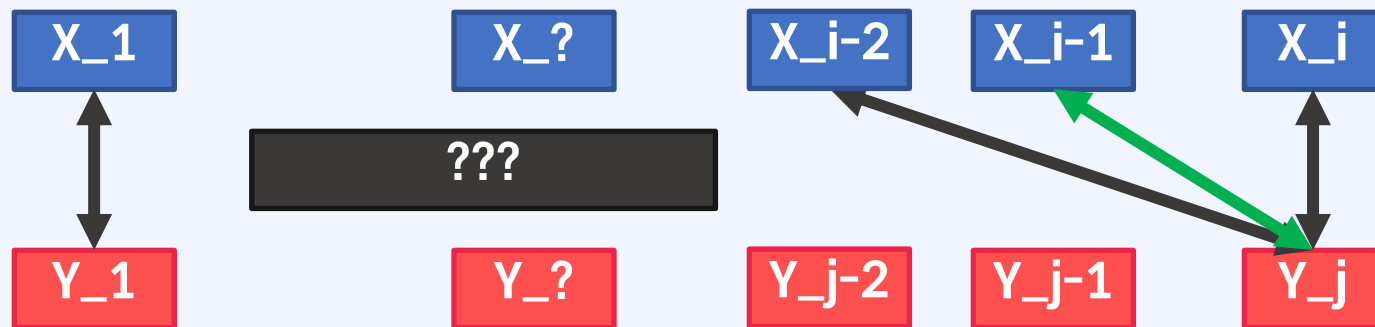
- $X[i], Y[j-2]$ 에 대응되는 경우?
 - $Y[j-1]$ 은 $X[i]$ 에 대응될 수밖에 없다
 - case3에서 이미 계산된다



BOJ15822: Ah-Choo!

문제 분석

- $X[i-2], Y[j]$ 에 대응되는 경우?
 - $X[i-1]$ 은 $Y[j]$ 에 대응될 수밖에 없다
 - case2에서 이미 계산된다



BOJ15822: Ah-Choo!

문제 분석

- 따라서 아래의 3가지 경우만 top-down으로 계산하면 된다
- $d[i][j] = \min3(d[i-1][j-1], d[i-1][j], d[i][j-1]) + \text{diff}^2$

BOJ15822: Ah-Choo!

구현

```
static int diff(int a, int b) {  
    return (a - b) * (a - b);  
}  
  
static int min3(int a, int b, int c) {  
    return Math.min(a, Math.min(b, c));  
}
```

차이의 제곱을 구하는 함수와, 3개 정수의 최소값을 구하는 함수 구현

BOJ15822: Ah-Choo!

구현

```
d[0][0] = 0;
for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) {
        d[i][j] = min3(d[i-1][j-1], d[i-1][j], d[i][j-1]) + diff(sound1[i], sound2[j]);
    }
}
System.out.println(d[n][n]);
```

케이스 정리만 잘 하면, 간단한 점화식이 나온다

Ch04. 동적계획법 #3

6. [11066] 파일 합치기

BOJ11066: 파일 합치기

문제 요약

- T개의 테스트 케이스, K개의 소설 파일
- 파일을 합치는 동안, 합친 용량만큼 임시 파일이 생성된다
- 임시 파일의 합이 최소가 되도록 합치고, 비용을 출력

BOJ11066: 파일 합치기

문제 요약

입력 데이터
1
4
40 30 30 50

출력 데이터
300

BOJ11066: 파일 합치기

문제 요약

	40	30	30	50
70	40	30	30	50
150	40	30	30	50
300	40	30	30	50

BOJ11066: 파일 합치기

문제 분석

- 문제 요구사항: 임시파일의 합이 최소가 되도록
 - $d[s][e] = [s, e]$ 구간의 합이 최소가 되도록 하는 크기
- 임시파일의 크기는 해당 영역의 구간 합이므로 Clip4에서 사용한 빠른 구간 합 함수를 미리 생성

BOJ11066: 파일 합치기

문제 분석

- 길이가 {length}인 파일을 구하려면?
 - 길이가 $[1, \text{length})$ 인 합치는 비용 정보가 필요하다
- 파일은 단계적으로 합쳐야 한다
 - 길이가 1인 파일은 0으로 초기화
 - ($\text{length} == 2$)인 파일부터 점점 커지도록

BOJ11066: 파일 합치기

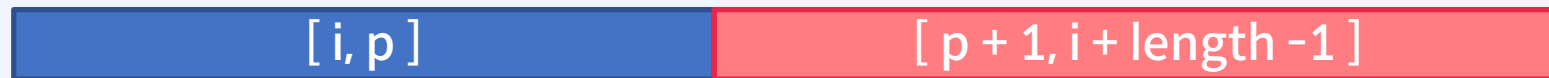
문제 분석

- $d[s][e] = [s, e]$ 구간의 합이 최소가 되도록 하는 크기
- 시작 좌표는 반복문을 돌며 1부터 순회하면 된다
- 끝 좌표는 앞서 $\{length\}$ 만큼의 정보를 확보해야 하므로 $\{시작좌표\} + \{length - 1\}$ 로 제한하여 정보를 획득한다

BOJ11066: 파일 합치기

문제 분석

- 앞서 풀이한 구간 나누기 문제와 유사하게 접근할 수 있다



- 두개의 파일을 합칠 때, 선택하는 위치에 따라 비용이 다르게 나올 수 있으므로, 반복문을 돌며 찾는다

BOJ11066: 파일 합치기

구현

```
for (int i = 1; i <= k; i++) {  
    for (int j = 1; j <= k; j++) {  
        d[i][j] = INF;  
    }  
    d[i][i] = 0;  
}
```

```
static int prefixSum(int i, int j) {  
    return sum[j] - sum[i - 1];  
}
```

초기화:
임시 파일의 크기 정보를
모두 INF로 채우기
단, 파일 1개([i][i]) 구간
은 다음 계산을 위해 0으
로 세팅

빠른 구간 합 계산을 위해
누적 합 배열을 사용

BOJ11066: 파일 합치기

구현

```
for (int length = 2; length <= k; length++) {  
    for (int i = 1; i <= k - length; i++) {  
        int end = i + length - 1;  
        if(end > k) break;  
        for (int p = i; p < end; p++) {  
            d[i][end] = Math.min(d[i][end], d[i][p] + d[p + 1][end] + prefixSum(i, end));  
        }  
    }  
}
```

길이 2인 파일부터 임시파일 정보 만들기

구간: $[i, i + \text{length} - 1]$

어느 위치에서 파일을 분할 하느냐에 따라 임시파일의 크기가 달라진다
{p} 변수를 이용해 분할하는 위치를 조정

Ch04. 동적계획법 #3

7. [10942] 팰린드롬?

BOJ10942: 팰린드롬?

문제 요약

- 자연수 N개를 나열, M개의 질문
- S번째 수부터 E번째 수까지 팰린드롬에 대한 true/false
- $(1 \leq N \leq 2,000)$ $(1 \leq M \leq 1,000,000)$
수는 100,000 보다 작거나 같음

BOJ10942: 팰린드롬?

문제 분석

- 만약 $[S, E]$ 구간의 수가 팰린드롬 관계라면?
 - 앞 뒤로 수를 하나씩 제거해도 팰린드롬이다
- 반대로 생각하면 $[S + 1, E - 1]$ 가 팰린드롬이고 $\{S\}$ 번째 수와 $\{E\}$ 번째 수가 같다면?
 - $[S, E]$ 는 팰린드롬이다

BOJ10942: 팰린드롬?

문제 분석

- top-down 접근
 - 만약 $[start]$ 와 $[end]$ 에 같은 수가 있다면
 - $[start + 1]$, $[end - 1]$ 가 같은 수인지 확인해 본다
 - 재귀 호출 중에 다른 수가 나오면? 팰린드롬이 아님
 - $start \geq end$ 까지 모두 같은 수라면, 팰린드롬

BOJ10942: 팰린드롬?

문제 분석

- 질문의 수가 1,000,000 개
- 최초로 입력 받은 수열은 중간에 변동되지 않는다
- 따라서 (s, e) 를 재귀 호출 중에 구한적이 있다면?
 - 배열에 결과를 저장하고, 다시 호출되면 재사용한다
- 주의: 출력결과도 많으므로, 버퍼에 담았다가 한번에 출력

BOJ10942: 팰린드롬?

구현

```
int n = sc.nextInt();
for(int i = 1; i <= n; i++) {
    arr[i] = sc.nextInt();
    for(int j = 1; j <= n; j++) {
        dp[i][j] = -1;
    }
}
```

연산 결과 저장할 dp배열
초기화

BOJ10942: 팰린드롬?

구현

```
static int isPalindrome(int start, int end) {  
    if(start >= end) return 1;  
    if(dp[start][end] != -1)  
        return dp[start][end];
```

만약 이전에 연산한 결과가
들어있다면, 탐색 없이 재사용

```
    if(arr[start] != arr[end]) return dp[start][end] = 0;  
    else dp[start][end] = isPalindrome(start + 1, end - 1);  
    return dp[start][end];  
}
```

글자가 다르면 팰린드롬이 아님
같으면 길이를 앞 뒤로 줄여가며 판단

Ch04. 동적계획법 #3

8. [1915] 가장 큰 정사각형

BOJ1915: 가장 큰 정사각형

문제 요약

- $n * m$ 의 배열에 0 or 1 이 숫자로 입력됨
- 1로 이루어진 가장 큰 정사각형의 넓이를 구하기
- ($1 \leq n, m \leq 1,000$)

BOJ1915: 가장 큰 정사각형

문제 분석

- 문제의 요구사항은 정사각형의 넓이이다
 - 넓이는 (길이)*(길이)로 계산 가능하므로
동적 배열은 정사각형의 한 변의 길이로 두고 판단해도 된다
- 기준점을 하나 잡고 생각을 해보자 (우측 하단)
- $d[i][j] = (i, j)$ 를 우측 하단으로 하는 정사각형의 최대 길이

BOJ1915: 가장 큰 정사각형

문제 분석

- 회색: 0, 흰색: 1
- $d[i][j] = (i, j)$ 를 우측 하단으로 하는 정사각형의 최대 길이
- 길이를 어떻게 구할 수 있을까?

			(i, j)

BOJ1915: 가장 큰 정사각형

문제 분석

- (i, j) 에 0이 들어있으면, 정사각형이 성립되지 않는다
 - 0으로 세팅
- (i, j) 에 1이 들어 있다면?

			(i-1, j)
			(i, j)

BOJ1915: 가장 큰 정사각형

문제 분석

- (i, j) 에 0이 들어있으면, 정사각형이 성립되지 않는다
 - 0으로 세팅
- (i, j) 에 1이 들어 있다면?

BOJ1915: 가장 큰 정사각형

문제 분석

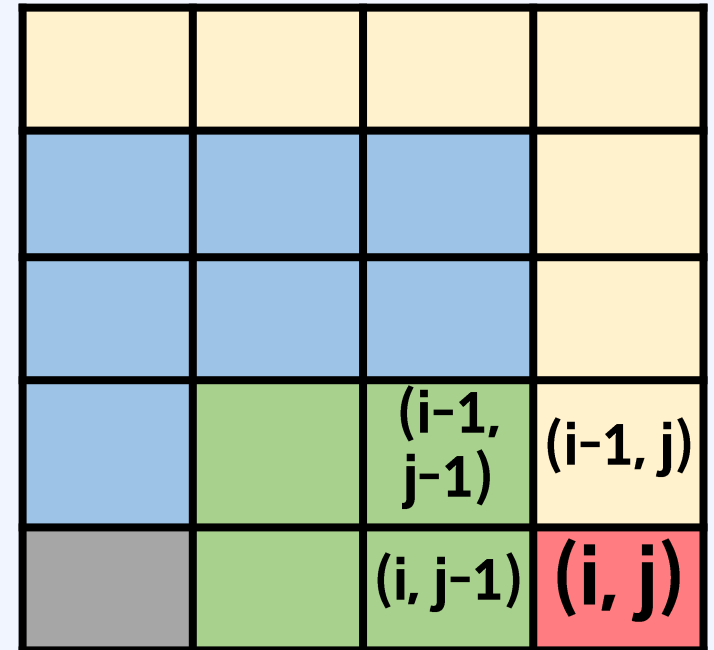
- (i, j) 에 0이 들어있으면, 정사각형이 성립되지 않는다
 - 0으로 세팅
- (i, j) 에 1이 들어 있다면?
- 좌, 상, 좌상 좌표에서 만들 수 있는 {최대 정사각형의 길이 중 최소} + 1

		(i-1, j-1)	(i-1, j)
		(i, j-1)	(i, j)

BOJ1915: 가장 큰 정사각형

문제 분석

- 좌, 상, 좌상 좌표에서 만들 수 있는 {최대 정사각형의 길이 중 최소} + 1
- 정사각형을 만들려면, (i, j) 좌표에서 같은 길이만큼 떨어져 있어야 한다



BOJ1915: 가장 큰 정사각형

문제 분석

- 좌, 상, 좌상 좌표에서 만들 수 있는 {최대 정사각형의 길이 중 최소} + 1
- $d[i][j] = \min(d[i][j-1], d[i-1][j], d[i-1][j-1]) + 1$
- $answer = \max(answer, d[i][j])$

		(i-1, j-1)	(i-1, j)
		(i, j-1)	(i, j)

BOJ1915: 가장 큰 정사각형

구현

```
for(int i = 1; i <= n; i++) {  
    for(int j = 1; j <= m; j++) {  
        if(arr[i][j] == 1) {  
            dp[i][j] = Math.min(dp[i - 1][j - 1], Math.min(dp[i][j - 1], dp[i - 1][j])) + 1;  
            max = Math.max(max, dp[i][j]);  
        }  
    }  
}
```

좌, 상, 좌상 좌표에서 만들 수 있는 {최대 정사각형의 길이 중 최소} + 1