

Dynamic Programming Basic - 2

DP 기초 - 2

競技プログラミングの鉄則

KPSC Algorithm Study 24/12/26 Thu.

by Haru_101

Dynamic Programming

- 저번엔 DP의 기초를 배워보았으니, 이번엔 다양하게 적용하는 문제를 풀어보겠습니다.

Dynamic Programming

- 다음 문제를 풀어봅시다. (https://atcoder.jp/contests/tessoku-book/tasks/tessoku_book_t)

問題文

文字列 S と T が与えられます。

S と T の共通部分列 (S の部分列かつ T の部分列) のうち、最長のものは何文字かを出力するプログラムを作成してください。

ただし、ある文字列の **部分列** とは、その文字列から順番を変えずに一部の文字を取り出したものを指します。

たとえば `grain` は `programming` の部分列です (4, 5, 6, 9, 10 文字目を取り出す)。



S
 T

制約

- S の文字数は 1 以上 2000 以下
 - T の文字数は 1 以上 2000 以下
 - S, T は英小文字からなる
- 문자열 S, T 가 주어질 때, S 와 T 의 공통부분열(S 의 부분열이면서 T 의 부분열)중에서 가장 긴 문자열의 길이를 출력하는 프로그램을 작성하세요.
 - 이때 문자열의 부분열이라는 것은 그 문자열에서 문자들의 순서를 바꾸지 않고 일부 문자를 뽑아낸 것을 의미합니다.
 예를 들어서 `grain`은 `programming`의 부분열입니다. (4, 5, 6, 9, 10번째 문자)
 - $1 \leq |S|, |T| \leq 2000$, S, T 는 영소문자로만 이루어져있음

Dynamic Programming

- 단순히 생각해 보면, $2^{|S|} \times 2^{|T|}$ 번정도 반복해서 추출해서 길이 비교하면 될거 같아보이지만...
- $2^{2000} \times 2^{2000}$ 은 눈으로 봐도 터질거 같아 보입니다.
- 이를 DP로 푸는 방법을 소개하겠습니다.



Dynamic Programming

- $DP[i][j]$ 를 S 의 $1..i$ 번째 문자로 이루어진 문자열과 T 의 $1..j$ 번째 문자로 이루어진 문자열의 최대 공통부분열의 크기라고 합시다.

		T					
	DP	NULL	s	e	o	u	l
S	NULL	0	0	0	0	0	0
	u	0	0	0	0	0	0
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- 먼저 base로는 각 문자열과 NULL의 공통부분열은 0임을 알 수 있습니다.

		T					
S	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	0	0
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- 그 다음, 각 행의 1열부터 오른쪽으로 이동해가면서 한 행씩 내려가보겠습니다.

		T					
S	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	0	0
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- (1, 1)에서는 $u \neq s$ 이므로 DP[1][1]의 값은 변하지 않습니다.

		<i>T</i>					
<i>S</i>	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	0	0
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- (1, 2)에서는 $u \neq e$ 이므로 DP[1][2]의 값은 변하지 않습니다.

		T					
S	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	0	0
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- (1, 3)에서는 $u \neq o$ 이므로 DP[1][3]의 값은 변하지 않습니다.

		<i>T</i>					
<i>S</i>	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	0	0
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- (1, 4)에서는 $u = u$ 이므로 DP[1][4]의 값이 변합니다. DP[1][4]=1이 됩니다.

		T					
S	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	1	0
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- (1, 5)에서는 $u \neq l$ 이지만, u와 seou / u와 seoul의 공통부분열의 최대크기는 같으므로 DP[1][5]=1이 됩니다.

		<i>T</i>					
<i>S</i>	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	1	1
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- (2, 1)에서는 $l \neq s$ 이므로 $DP[2][1]=0$ 입니다.

		<i>T</i>					
<i>S</i>	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	1	1
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- (2, 2)에서는 $l \neq e$ 이므로 $DP[2][2]=0$ 입니다.

		T					
S	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	1	1
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- (2,3)에서는 $l \neq o$ 이므로 $DP[2][3]=0$ 입니다.

		<i>T</i>					
<i>S</i>	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	1	1
	l	0	0	0	0	0	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- 이제 (2, 4)가 문제입니다. $l \neq u$ 이지만, u와 seou의 공통부분열의 최대 크기는 1입니다. 따라서 바로 위의 행 값도 가져와야 합니다.

		T					
S	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	1	1
	l	0	0	0	0	1	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- 즉, $DP[i][j] = \max(DP[i][j-1], DP[i-1][j])$ 가 먼저 들어갑니다.

		<i>T</i>					
<i>S</i>	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	1	1
	l	0	0	0	0	1	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

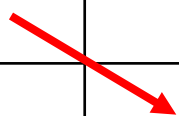
- 그런데, 이렇게만 하면 모든 값이 0으로 채워지기 때문에 뭔가 더하는 작업이 필요합니다.

		T					
S	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	1	1
	l	0	0	0	0	1	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- 바로 $S_i = T_j$ 면 $DP[i-1][j-1]+1$ 을 비교해주면 됩니다.

		<i>T</i>					
<i>S</i>	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	1	1
	l	0	0	0	0	1	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0



Dynamic Programming

- 왜 대각선 방향을 고려하냐면, 단순히 $S_i = T_j$ 라 해서 $\max(DP[i-1][j], DP[i][j-1]) + 1$ 을 하면 아래와 같이 반례가 생깁니다. (sel, hell은 최대가 2인데 3)

		T					
S	DP	NULL	h	e	l	l	o
	NULL	0	0	0	0	0	0
	s	0	0	0	0	0	0
	e	0	0	1	1	1	1
	l	0	0	1	2	→ 3??	
	l	0					
	s	0					

Dynamic Programming

- 따라서 이러한 이유로 $DP[i][j] = \max(DP[i][j-1], DP[i-1][j], DP[i-1][j-1]+1)$ 이 됩니다.

		<i>T</i>					
<i>S</i>	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0	0	0	0	1	1
	l	0	0	0	0	1	0
	s	0	0	0	0	0	0
	a	0	0	0	0	0	0
	n	0	0	0	0	0	0

Dynamic Programming

- 즉, 오른쪽/아래/화살표 방향으로 이동하면서 거친 화살표의 최대 개수를 세는 것과 마찬가지입니다.

		T					
S	DP	NULL	s	e	o	u	l
	NULL	0	0	0	0	0	0
	u	0				1	0
	l	0					2
	s	0	1				
	a	0					
	n	0					2

$u = u$
 $l = l$
 $s = s$

Dynamic Programming

- 교재에 나온 tokyo, kyoto로도 DP테이블을 채워보겠습니다.

T

	DP	NULL	t	o	k	y	o
	NULL	0	0	0	$k = k$	0	0
	k	0			1		
	y	0				2	
S	o	0		$o = o$			3
	t	0	$t = t$	1			
	o	0		$o = o$			3

Red arrows indicate the sequence of matches: $k = k$, $y = y$, $o = o$, $t = t$, $o = o$.

Dynamic Programming

- 코드로 짜면 다음과 같습니다.

```
int dp[2005][2005];
int main() {
    fastio();
    string S, T;
    cin >> S >> T;
    string S_r = ' ' + S;
    string T_r = ' ' + T;
    dp[0][0] = 0;
    for(int i=1; i<=S_r.size(); i++) {
        for(int j=1; j<=T_r.size(); j++) {
            if(S_r[i]==T_r[j]) {
                dp[i][j] = max({dp[i-1][j], dp[i][j-1], dp[i-1][j-1] + 1});
            } else {
                dp[i][j] = max({dp[i-1][j], dp[i][j-1]});
            }
        }
    }
    cout << dp[S.size()][T.size()];
}
```


Dynamic Programming

- 다음 문제를 풀어봅시다. (https://atcoder.jp/contests/tessoku-book/tasks/tessoku_book_u)

問題文

N 個のブロックが並べられており、左から順に $1, 2, \dots, N$ と番号が付けられています。

あなたは、以下の 2 種類の操作を何回か行うことで、すべてのブロックを取り除きたいです。

- 今ある中で **一番左** のブロックを取り除く。
- 今ある中で **一番右** のブロックを取り除く。

ブロック i ($i = 1, 2, \dots, N$) をブロック P_i より先に取り除いた場合、 A_i 点が得られます。

合計得点としてあり得る最大値を出力するプログラムを作成してください。

制約

- $2 \leq N \leq 2000$
- $1 \leq P_i \leq N$
- $P_i \neq i$
- $1 \leq A_i \leq 100$
- 入力はすべて整数

- N 개의 블록이 나열되어 있고, 왼쪽부터 $1, 2, \dots, N$ 까지 번호가 붙어있습니다.
- 아래 2종류의 행동을 여러번 해서 모든 블록을 제거해야합니다.
 - 지금 있는 블록중에 가장 왼쪽에 있는 블록을 제거한다.
 - 지금 있는 블록중에 가장 오른쪽에 있는 블록을 제거한다.
- 블록 i 를 블록 P_i 보다 먼저 제거한 경우 A_i 점을 얻습니다. 총 득점수의 최대치를 출력하는 프로그램을 작성하세요.

Dynamic Programming

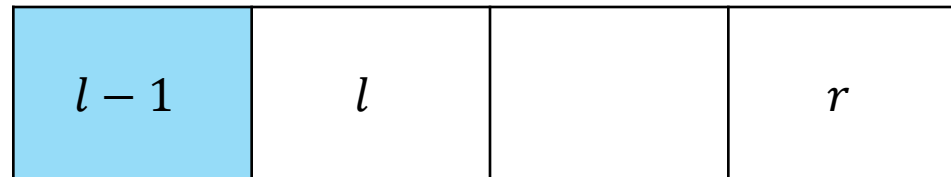
- 현재 블록의 상태를 l, r 과 같은 변수로 표현할 수 있습니다.
- 즉, 현재 남아있는 블록이 $l, l + 1, l + 2, \dots, r - 2, r - 1, r$ 번임을 의미합니다.
- 이를 이용해서 DP 테이블을 구성할 수 있습니다.

Dynamic Programming

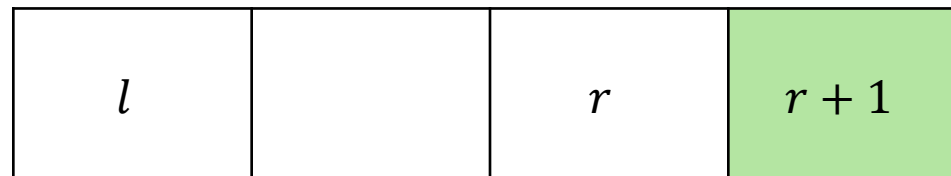
- 먼저 DP를 다음과 같이 정의할 수 있습니다.
- $DP[l][r]$: 현재 블록이 $l \sim r$ 까지 남아있을 때 최대점수
- 우리가 구해야하는 것은 $DP[1][1], DP[2][2], \dots, DP[N][N]$ 의 최댓값이 됩니다.
- 왜냐하면, 마지막 블록의 경우 다른 블록들보다 먼저 제거하지 못하기 때문입니다.
- 또한 초기에는 점수가 0이므로 $DP[1][N]=0$ 입니다.

Dynamic Programming

- DP에 관한 식은 다음과 같이 정의할 수 있을 것 같습니다.
- 왼쪽 블록을 제거하는 경우 : $DP[l][r] = DP[l-1][r] + \text{score}$



- 오른쪽 블록을 제거하는 경우 : $DP[l][r] = DP[l][r+1] + \text{score}$



Dynamic Programming

- score의 값은 아래와 같이 결정됩니다.
- 왼쪽 블록을 제거하는 경우에는 $l \leq P_{l-1} \leq r$ 의 경우 $score = A_{l-1}$, 아니면 $score = 0$
- 오른쪽 블록을 제거하는 경우에는 $l \leq P_{r+1} \leq r$ 의 경우 $score = A_{r+1}$, 아니면 $score = 0$

Dynamic Programming

- 이제 DP 테이블을 채워봅시다.
- 초기 조건에서 DP[1][N]은 0이므로 0을 써줍시다.

r

	DP	1	2	3	4
l	1				0
	2				
	3				
	4				

Dynamic Programming

- 이제 블록을 왼쪽에서 하나 제거하는 거랑 오른쪽에서 제거하는 거를 계산해봅시다.
- $(P_4, A_4) = (1, 10), (P_1, A_1) = (4, 20)$

		r			
		1	2	3	4
l	1			10 ←	0
	2				↓ 20
	3				
	4				

Dynamic Programming

- 이제 블록을 왼쪽에서 하나 제거하는 거랑 오른쪽에서 제거하는 거를 계산해봅시다.
- $(P_3, A_3) = (2, 40), (P_1, A_1) = (4, 20)$

		r			
		1	2	3	4
l	1		50	10	0
	2			10	20
	3				
	4				

Dynamic Programming

- 이제 블록을 왼쪽에서 하나 제거하는 거랑 오른쪽에서 제거하는 거를 계산해봅시다.
- $(P_2, A_2) = (3, 30), (P_1, A_1) = (4, 20)$

		r			
		1	2	3	4
l	1	50 ←	50 ↓	10	0
	2		50	10	20
	3				
	4				

Dynamic Programming

- 이제 블록을 왼쪽에서 하나 제거하는 거랑 오른쪽에서 제거하는 거를 계산해봅시다.
- $(P_4, A_4) = (1, 10), (P_2, A_2) = (3, 30)$

		r			
		1	2	3	4
l	1	50	50	10	0
	2		50	20 ←	20
	3				↓ 50
	4				

Dynamic Programming

- 이제 블록을 왼쪽에서 하나 제거하는 거랑 오른쪽에서 제거하는 거를 계산해봅시다.
- $(P_3, A_3) = (2, 40), (P_2, A_2) = (3, 30)$

		r			
		1	2	3	4
l	1	50	50	10	0
	2		60 ←	20 ↓	20
	3			50	50
	4				

Dynamic Programming

- 이제 블록을 왼쪽에서 하나 제거하는 거랑 오른쪽에서 제거하는 거를 계산해봅시다.
- $(P_4, A_4) = (1, 10), (P_3, A_3) = (2, 40)$

		r			
		1	2	3	4
l	1	50	50	10	0
	2		60	20	20
	3			50	50
	4				50

Dynamic Programming

- 정답은 대각성분의 최댓값이 됩니다.

		r				
		DP	1	2	3	4
l	1		50	50	10	0
	2			60	20	20
	3				50	50
	4					50

Dynamic Programming

- 이를 코드로 짜면 다음과 같습니다.

```
int P[2005];
int A[2005];
int dp[2005][2005];
int main() {
    fastio();
    int N;
    cin >> N;
    for(int i=1; i<=N; i++) {
        cin >> P[i] >> A[i];
    }
    int score_L = 0;
    int score_R = 0;
    for(int l=1; l<=N; l++) {
        for(int r=N; r>l; r--) {
            if(l <= P[r] && P[r] <= r-1) {
                score_R = A[r];
            } else {
                score_R = 0;
            }
            if(l+1 <= P[l] && P[l] <= r) {
                score_L = A[l];
            } else {
                score_L = 0;
            }
            dp[l][r-1] = max(dp[l][r-1], dp[l][r] + score_R);
            dp[l+1][r] = max(dp[l+1][r], dp[l][r] + score_L);
        }
    }
    int ans = 0;
    for(int i=1; i<=N; i++) {
        ans = max(ans, dp[i][i]);
    }
    cout << ans;
}
```

Dynamic Programming

- 다음 문제를 풀어봅시다. (https://atcoder.jp/contests/tessoku-book/tasks/tessoku_book_w)

問題文

情報商店では N 種類の品物を扱っています。それぞれ 1 から N までの番号が付けられています。
この店では、いくつかの指定された品物を無料で買えるクーポン券が配布されています。

太郎君は M 枚のクーポン券を持っています。

クーポン券 i ($i = 1, 2, \dots, M$) の情報は以下の通りです。

- $A_{i,j} = 1$ のとき：品物 j は無料で買える対象に含まれている。
- $A_{i,j} = 0$ のとき：品物 j は無料で買える対象に含まれていない。

最小何枚のクーポン券を使うことで、 N 種類すべての品物を買うことができますか。

制約

- $1 \leq N \leq 10$
- $1 \leq M \leq 100$
- $0 \leq A_{i,j} \leq 1$
- 入力はすべて整数

- 상점에는 N 종류의 상품을 취급하고 있고, 각 상품은 $1, \dots, N$ 의 번호가 붙어 있습니다.
- 이 상점에서는 몇 개의 상품을 무료로 살 수 있는 쿠폰을 배포하고 있습니다.
- 당신은 M 장의 쿠폰을 갖고 있고, 쿠폰 i 의 정보는 다음과 같습니다.
 - $A_{i,j} = 1$ 이면 상품 j 를 무료로 살 수 있다.
 - $A_{i,j} = 0$ 이면 상품 j 를 무료로 살 수 없다.
- 최소 몇 장의 쿠폰을 사용해서 N 종류의 상품을 모두 무료로 살 수 있는지 출력하는 프로그램을 작성하세요.

Dynamic Programming

- 이번에 배울 DP는 비트 DP라고 부르는 기법입니다.
- 비트 DP는 이진수 연산을 통해서 DP 테이블에 접근하는 방법입니다.
- $A = 11100011_2, B = 00001100_2$ 일때 $A \text{ OR } B$ 의 값을 구해봅시다.

$$\begin{array}{r}
 \\
 11100011_2 \\
 OR 00001100_2 \\
 \hline
 11101111_2
 \end{array}$$

Dynamic Programming

- 이진수의 특성에 따라 $11100011_2 = 2^7 + 2^6 + 2^5 + 2^1 + 2^0$ 입니다.
- 지수를 살펴보면, 위 이진수는 0, 1, 5, 6, 7번째 값이 1임을 나타낸다는 것을 알 수 있습니다.
- 이를 문제에 적용시켜봅시다.
- 현재 구매한 물품을 11100011_2 로 나타냈을때, 2, 3번째 물건을 구매한다면 11100011_2 에 00001100_2 를 OR 연산 취해서 구매하고 난 후의 정보를 11101111_2 로 나타낼 수 있게 됩니다.
- 11101111_2 를 정수표현으로 다시 나타낸다면 $DP[~~][11101111_2]$ 식으로 DP 테이블에 접근할 수 있습니다.

```
3 4
0 0 1
0 1 0
1 0 0
1 1 0
```

Dynamic Programming

- 이제 DP 테이블을 채워봅시다.
- 먼저 base로 아무 물품을 사지 않는 경우에 사용할 수 있는 쿠폰의 최솟값은 0입니다.

DP	0	1	2	3	4	5	6	7
0	0	INF	INF	INF	INF	INF	INF	INF
1	INF	INF	INF	INF	INF	INF	INF	INF
2	INF	INF	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF	INF	INF	INF

```
3 4
0 0 1
0 1 0
1 0 0
1 1 0
```

Dynamic Programming

- 그 다음, 1번 쿠폰을 사용하지 않는 경우부터, 위에서 값을 가져오면 됩니다.
- 만약 쿠폰을 사용하는 경우 $DP[1][000_2 \text{ OR } 001_2] = \min(DP[0][000_2 \text{ OR } 001_2], DP[0][000_2] + 1)$ 입니다.

DP	0	1	2	3	4	5	6	7
0	0	INF	INF	INF	INF	INF	INF	INF
1	0	1	INF	INF	INF	INF	INF	INF
2	INF	INF	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF	INF	INF	INF

入力例 1 [Copy](#)

```
3 4
0 0 1
0 1 0
1 0 0
1 1 0
```

Dynamic Programming

- 그 다음, 2번 쿠폰을 사용하지 않는 경우부터, 위에서 값을 가져오면 됩니다.

DP	0	1	2	3	4	5	6	7
0	0	INF	INF	INF	INF	INF	INF	INF
1	0	1	INF	INF	INF	INF	INF	INF
2	0	1	INF	INF	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF	INF	INF	INF

入力例 1 Copy

```
3 4
0 0 1
0 1 0
1 0 0
1 1 0
```

Dynamic Programming

- 그 다음, 2번 쿠폰을 사용할때에는 이전 행에서 INF 값이 아닌 경우 010_2 를 OR 을 취해 DP 테이블을 채우면 됩니다.

DP	0	1	2	3	4	5	6	7
0	0	INF	INF	INF	INF	INF	INF	INF
1	0	1	INF	INF	INF	INF	INF	INF
2	0	1	1	2	INF	INF	INF	INF
3	INF	INF	INF	INF	INF	INF	INF	INF
4	INF	INF	INF	INF	INF	INF	INF	INF

```
3 4
0 0 1
0 1 0
1 0 0
1 1 0
```

Dynamic Programming

- 그 다음, 3번 쿠폰을 사용하지 않는 경우부터, 위에서 값을 가져오면 됩니다.

DP	0	1	2	3	4	5	6	7
0	0	INF	INF	INF	INF	INF	INF	INF
1	0	1	INF	INF	INF	INF	INF	INF
2	0	1	1	2	INF	INF	INF	INF
3	0	1	1	2	INF	INF	INF	INF
4	INF	INF	INF	INF	INF	INF	INF	INF

入力例 1 Copy

```
3 4
0 0 1
0 1 0
1 0 0
1 1 0
```

Dynamic Programming

- 그 다음, 3번 쿠폰을 사용할때에는 이전 행에서 INF 값이 아닌 경우 100_2 를 OR 을 취해 DP 테이블을 채우면 됩니다.

DP	0	1	2	3	4	5	6	7
0	0	INF	INF	INF	INF	INF	INF	INF
1	0	1	INF	INF	INF	INF	INF	INF
2	0	1	1	2	INF	INF	INF	INF
3	0	1	1	2	1	2	2	3
4	INF	INF	INF	INF	INF	INF	INF	INF

入力例 1 [Copy](#)

```
3 4
0 0 1
0 1 0
1 0 0
1 1 0
```

Dynamic Programming

- 그 다음, 4번 쿠폰을 사용하지 않는 경우부터, 위에서 값을 가져오면 됩니다.

DP	0	1	2	3	4	5	6	7
0	0	INF	INF	INF	INF	INF	INF	INF
1	0	1	INF	INF	INF	INF	INF	INF
2	0	1	1	2	INF	INF	INF	INF
3	0 ↓	1 ↓	1 ↓	2 ↓	1 ↓	2 ↓	2 ↓	3 ↓
4	0	1	1	2	1	2	2	3

Dynamic Programming

入力例 1 [Copy](#)

```
3 4
0 0 1
0 1 0
1 0 0
1 1 0
```

- 그 다음, 4번 쿠폰을 사용할때에는 이전 행에서 INF 값이 아닌 경우 110_2 를 OR 을 취해 DP 테이블을 채우면 됩니다.

DP	0	1	2	3	4	5	6	7
0	0	INF	INF	INF	INF	INF	INF	INF
1	0	1	INF	INF	INF	INF	INF	INF
2	0	1	1	2	INF	INF	INF	INF
3	0	1	1	2	1	2	2	3
4	0	1	1	2	1	2	1	2

Dynamic Programming

入力例 1 [Copy](#)

```
3 4
0 0 1
0 1 0
1 0 0
1 1 0
```

- 우리가 원하는 것은 모든 물품을 공짜로 사는 것이니, 111_2 열의 최소값을 취하면 됩니다.

DP	0	1	2	3	4	5	6	7
0	0	INF	INF	INF	INF	INF	INF	INF
1	0	1	INF	INF	INF	INF	INF	INF
2	0	1	1	2	INF	INF	INF	INF
3	0	1	1	2	1	2	2	3
4	0	1	1	2	1	2	1	2

Dynamic Programming

- 이를 코드로 짜면 다음과 같습니다.

```
int dp[105][(1 << 10)+5];
int arr[105];
int main() {
    fastio();
    int N, M;
    cin >> N >> M;
    for(int i=1; i<=M; i++) {
        int k=1;
        for(int j=0; j<N; j++) {
            int x;
            cin >> x;
            if(x==1) {
                arr[i]+=k;
            }
            k*=2;
        }
    }
    for(int i=0; i<=M; i++) {
        for(int j=0; j<=((1<<10)-1); j++) {
            dp[i][j] = 100000;
        }
    }
    dp[0][0] = 0;
    for(int i=1; i<=M; i++) {
        for(int j=0; j<=((1<<N)-1); j++) {
            dp[i][j] = min(dp[i][j], dp[i-1][j]);
            dp[i][j | arr[i]] = min(dp[i][j | arr[i]], dp[i-1][j] + 1);
        }
    }
    int ans = 100000;
    for(int i=1; i<=M; i++) {
        ans = min(ans, dp[i][(1 << N)-1]);
    }
    if(ans != 100000) {
        cout << ans;
    } else {
        cout << -1;
    }
}
```

Dynamic Programming

- 다음 문제를 풀어봅시다. (https://atcoder.jp/contests/tessoku-book/tasks/tessoku_book_x)

問題文

配列 $A = [A_1, A_2, \dots, A_N]$ の最長増加部分列の長さを求めてください。

制約

- $1 \leq N \leq 100000$
- $1 \leq A_i \leq 500000$
- 入力はすべて整数

- 배열 $A = [A_1, A_2, \dots, A_N]$ 의 *최장증가부분열의 길이를 구하는 프로그램을 작성하세요.
- *최장증가부분열 : $x_1 < x_2 < \dots < x_k$ 이면서 $L_{x_1} < L_{x_2} < L_{x_3} < \dots < L_{x_k}$ 인 L

Dynamic Programming

- 이 문제는 이분탐색과 DP를 섞어서 쓰는 문제입니다.
- $DP[i]$ 를 i 번째 값까지의 최장증가부분열의 크기라고 합시다.
- 먼저 베이스로, $DP[i]$ 는 최소 1입니다. (자기 자신)
- 그 외의 경우, 이전 $j < i$ 인 A_j 값들과 비교하면서 $A_j < A_i$ 면 $DP[i] = \max(DP[i], DP[j]+1)$ 입니다.
- 하지만, 이는 시간 복잡도가 $O(n^2)$ 이기 때문에 어려워 보입니다.

Dynamic Programming

- 그럼, 이전 A_j 를 어떻게 빠르게 가져올 수 있을까요?
- 직접 A 에서 찾지 말고, 현재 값보다 작은 값의 DP값들 중에서 가장 큰 값에서 1을 더하면 됩니다.
- $DP[k]$ 가 1~ k 까지 인덱스에서의 최장증가부분열의 크기이므로, $L[i]$ 에는 $DP[k]=i$ 이면서 그러한 A_k 중 최솟값을 담는 배열이라고 합시다.
- 그러면 어떤 A_i 에 대해서 $L[j] < A_i$ 이면서 가장 큰 j 가 이전 DP값의 최대가 되고, 이 값에 1을 더하면 $DP[A_i]$ 를 구할 수 있습니다.
- 말로 하면 어려우니, 그림으로 살펴봅시다.

Dynamic Programming

- $A = [2, 3, 1, 6, 4, 5]$ 라고 하고, 오름차순으로 요소들을 저장할 배열을 L 이라고 합시다.
- 먼저, $DP[1]=1$ 이고, $L = []$ 이기 때문에, $A_1 = 2$ 는 그냥 L 에 넣어줍니다.

A	2	3	1	6	4	5
L	2					
DP	1					

Dynamic Programming

- 그 다음, $A_2 = 3$ 의 DP값은 기본적으로 $DP[2]=1$ 이고, A_2 값보다 작은 L 에서 찾습니다.
- $L_1 = 2 < A_2$ 이므로, $DP[2]=(\text{화살표가 가리키는 인덱스} + 1)=2$ 이 됩니다.
- 이 값이 2보다 크므로 L_2 를 $A_2 = 3$ 으로 업데이트 하면 됩니다.
- 즉, $L_2 = 3$, 3이 길이가 2인 최장증가부분열의 가장 오른쪽 요소임을 뜻합니다.

A	2	3	1	6	4	5
L	2	3				
DP	1	2				

Dynamic Programming

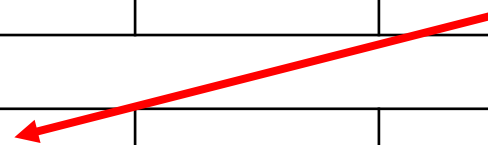
- 그 다음, $A_3 = 1$ 의 DP값은 기본적으로 $DP[3]=1$ 이고, 이 값보다 작은 L 에서 찾습니다.
- 그런 값이 L 에 존재하지 않습니다. 따라서 $DP[3]$ 의 값은 1로 그대로 저장됩니다.
- $DP[k]=1$ 인 k 중에서 가장 작은 A_k 는 1이므로, $L_1 = 1$ 이 됩니다.

A	2	3	1	6	4	5
L	1	3				
DP	1	2	1			

Dynamic Programming

- 그 다음, $A_4 = 6$ 의 DP값은 기본적으로 $DP[4]=1$ 이고, 이 값보다 작은 L 에서 찾습니다.
- $L_2 = 3 < A_4$ 이므로, $DP[4]=(\text{화살표가 가리키는 인덱스} + 1)=3$ 이 됩니다.
- $DP[k]=3$ 인 k 중에서 가장 작은 A_k 는 현재 6이므로 $L_3 = 6$ 이 됩니다.

A	2	3	1	6	4	5
L	1	3	6			
DP	1	2	1	3		



Dynamic Programming


- 그 다음, $A_5 = 4$ 의 DP값은 기본적으로 $DP[5]=10$ 이고, A_5 보다 작은 값을 L 에서 찾습니다.
- $L_2 = 3 < A_5$ 이므로, $DP[5]=(\text{화살표가 가리키는 인덱스} + 1)=3$ 이 됩니다.
- $DP[k]=3$ 인 k 중에서 가장 작은 A_k 는 현재 4이므로 $L_3 = 4$ 가 됩니다.

A	2	3	1	6	4	5
L	1	3	4			
DP	1	2	1	3	3	

Dynamic Programming

- 그 다음, $A_6 = 5$ 의 DP값은 기본적으로 $DP[6]=10$ 이고, A_6 보다 작은 값을 L 에서 찾습니다.
- $L_3 = 4 < A_6$ 이므로, $DP[6]=(\text{화살표가 가리키는 인덱스} + 1)=4$ 가 됩니다.
- $DP[k]=4$ 인 k 중에서 가장 작은 A_k 는 현재 5이므로 $L_4 = 5$ 가 됩니다.

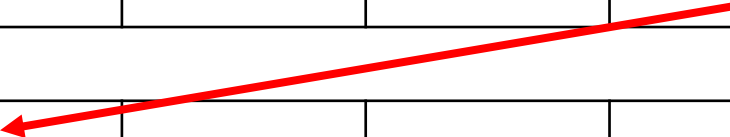
A	2	3	1	6	4	5
L	1	3	4	5		
DP	1	2	1	3	3	4



Dynamic Programming


- L 은 오름차순으로 정렬되기 때문에 lower_bound와 같은 함수를 사용해서 인덱스를 찾으면 됩니다.
- 이렇게 되면 $O(n^2)$ 에서 시간이 많이 절약되기 때문에 문제를 잘 해결할 수 있게 됩니다.

A	2	3	1	6	4	5
L	1	3	4	5		
DP	1	2	1	3	3	4



Dynamic Programming

- 최장증가부분열의 길이를 구하는 방법은 L 의 길이를 출력하거나, DP의 max값을 출력하면 됩니다.
- 만약 최장증가부분열의 요소들을 출력하고자 한다면, A 의 가장 오른쪽에서부터 시작해서 A 및 DP값이 감소하는 순으로 출력하면 됩니다.



A	2	3	1	6	4	5
L	1	3	4	5		
DP	1	2	1	3	3	4

Dynamic Programming

- 이를 코드로 짜면 다음과 같습니다.

```
int arr[100005];
int L[100005];
int dp[100005];
int main() {
    fastio();
    int N;
    cin >> N;
    int len = 0;
    for(int i=0; i<N; i++) {
        cin >> arr[i];
        int idx = lower_bound(L+1, L+len+1, arr[i]) - L;
        dp[i] = idx;
        L[dp[i]] = arr[i];
        if(dp[i] > len) len++;
    }
    cout << len;
}
```

Dynamic Programming Basic - 2

끝.

다음주에는 Mathematics Basic - 1로 찾아뵙겠습니다.