김범수

DP 문제 풀이

DP solution

By POSCAT

피보나치 함수

백준 1003

https://www.acmicpc.net/problem/1003

- 메모이제이션 없이 재귀 만으로 구현한 피보나치 함수를 호출할 때, f(0)과 f(1)이 몇 번 호출되는 지를 묻는 문제입니다.
- 따라서 DP[n] = {f(n)을 호출할 때 f(0)이 호출되는 횟수, f(n)을 호출할 때 f(1)이 호출되는 횟수}로 정의합니다.
- 먼저 초기 상황은 DP[0] = {1, 0}, DP[1] = {0, 1}로 정의할 수 있습니다. 그렇다면 점화식은 DP[n] = DP[n – 1] + DP[n – 2]가 됩니다.
- 따라서 O(n)의 시간복잡도로 문제를 풀 수 있으므로 넉넉하게 제한시간안에 통과할 수 있습니다.

가장 긴 증가하는 부분 수열

백준 11053

https://www.acmicpc.net/problem/11053

- 부분 수열이란 수열이 주어졌을 때, 몇 개의 숫자만 고르되 순서는 바꾸지 않고 골라서 만들 수 있는 수열을 의미합니다.
- 증가하는 수열이란 {1, 3, 6, 8}처럼 뒤로 갈 수록 숫자가 커지는 수열을 증가하는 수열이라 합니다. 즉, 모든 i에 대해 A[i] < A[i+1]을 만족하는 수열 A를 증가하는 수열이라 합니다. 여기서는 {1, 3, 4, 4} 같이 A[i] = A[i+1]인 경우가 존재하면 증가하는 수열로 취급하지 않습니다.
- DP(n)을 n번째 수를 마지막 수로 하는 부분 수열의 최대 길이로 정의하겠습니다. 예를 들어 A = {10, 20, 10, 30, 40, 50}이라는 수열이 주어지는 경우, DP(4)를 구해봅니다. 4번째 원소인 30을 마지막 원소로 하는 부분 수열은 {10, 30}, {20, 30}, {30}, {10, 20, 30}이 가능합니다. 이 중 가장 긴 수열은 {10, 20, 30}이고 그 길이는 3이므로, DP(4) = 3이 됩니다.

가장 긴 증가하는 부분 수열

백준 11053

https://www.acmicpc.net/problem/11053

- n번째 원소를 마지막 원소로 하는 증가하는 부분 수열은 어떻게 만들 수 있을까요? n번째 원소 하나만 가진 수열을 만들 수도 있고, 아니면 n보다 작은 k에 대해 k번째 원소를 마지막 원소로 하는 부분 수열에서 n번째 원소를 추가할 수도 있습니다. 이 때는 A[k] < A[n]이 만족해야 합니다.
- 따라서 DP[n] = max(DP[k] + 1) for k = 0, 1, 2, ⋯, n – 1 and A[k] < A[n]
  라는 점화식을 구할 수 있습니다. 이 때 초기 상태는 DP[0] = 0으로 정의할 수 있습니다. 또한 A[0] = 0이라 가정하면 점화식을 적용할 수 있습니다.
- 원소의 개수를 N이라 하면 답은 max(DP[1], DP[2], ⋯, DP[N])이 됩니다.
- DP[n]을 구하기 위해 O(N)번의 연산을 해야 하고, DP[1]부터 DP[N]까지 N개의 DP값을 계산해야 하므로 시간복잡도는 O(N^2)이 됩니다. N의 최댓값이 1000이므로 넉넉하게 제한시간 안에 통과할 수 있습니다.

1로 만들기

백준 1463

https://www.acmicpc.net/problem/1463

- 세 가지 연산, -1, /2, /3을 최소한으로 사용해서 x를 1로 만드는 데 필요한 최소 횟수를 출력하는 문제입니다.
- DP[n] = n을 1로 만드는 데 필요한 연산의 최소 횟수 라 정의합니다.
- 만약 n에 1을 빼는 연산을 진행한다면 필요한 연산의 최소 횟수는 DP[n-1] + 1이 됩니다.
- 만약에 n이 2의 배수라 2를 나누는 연산을 진행한다면 필요한 연산의 최소 횟수는 DP[n/2] + 1이 됩니다.
- 만약에 n이 3의 배수라 3를 나누는 연산을 진행한다면 필요한 연산의 최소 횟수는 DP[n/3] + 1이 됩니다.
- 이 중 가장 작은 값이 DP[n]이 됩니다. 즉
  DP[n] = min(DP[n-1], DP[n/2] if n % 2 == 0, DP[n/3] if n % 3 == 0) + 1, 초기값 DP[1] = 0
  으로 식을 세울 수 있습니다.
- n의 최대 범위는 백만이고 시간 복잡도는 O(N)이므로 넉넉하게 시간제한을 통과할 수 있습니다.

파도반 수열

백준 9461

https://www.acmicpc.net/problem/9461

- 그림을 관찰하면 P(n) = P(n – 1) + P(n – 5)가 성립함을 알 수 있습니다.
- 따라서 피보나치 수열을 구하는 것과 똑같은 방식으로 구현하면 P(n)을 구할 수 있습니다.
- 초기항은 1, 1, 1, 2, 2 가 됩니다. 그림을 통해 알 수 있습니다.
- P(n)을 구하는 데는 O(1)의 시간이 걸리고, 총 P(1)부터 P(n)까지 모두 구해야 하므로 시간복잡도는 O(n)입니다. n은 최대 100이므로 충분히 제한 시간안에 통과합니다.

포도주 시식

백준 2156

https://www.acmicpc.net/problem/2156

- 이 문제는 n개의 포도주의 양이 1번부터 n번까지 주어졌을 때 최대한 많은 포도주를 마시면 되는 문제입니다. 하지만 연속된 3개의 포도주는 마실 수 없습니다. 예를 들어 3번, 4번, 5번 포도주를 모두 마실 수는 없습니다.
- 먼저 k번째 포도주의 양을 A[k]라 하겠습니다. 그리고, DP(n, m)을 “1번 포도주부터 n번 포도주까지 마실 지 말 지 결정했고, 지금까지 m번 연속으로 포도주를 마셨을 때, 마실 수 있는 포도주의 최대량” 으로 정의합니다.
- 예를 들어 DP(5, 1)은 1번 포도주부터 5번 포도주까지 마실 지 말지를 결정했고 지금까지 1번 연속으로 포도주를 마셨다는 것을 의미합니다. 즉 4번 포도주는 마시지 않았고, 5번 포도주는 마신 상태입니다. 1 ~ 3번 포도주는 마시든 마시지 않았든 지금까지 1번 연속으로 포도주를 마셨다는 사실은 변하지 않으므로 알 필요가 없습니다. 따라서 DP(5, 1)은 4번 포도주를 마시지 않고, 5번 포도주를 마신 상황에서 내가 마실 수 있는 포도주의 최대량을 의미합니다.
- 먼저 DP(n, 0)을 계산해봅시다. 0번 연속으로 포도주를 마시지 않았다는 것은 곧 n번째 포도주를 마시지 않았음을 의미합니다. 따라서 n-1번째 포도주까지 마실 수 있는 포도주의 최댓값이 DP(n, 0)의 값이 됩니다.

포도주 시식

백준 2156

https://www.acmicpc.net/problem/2156

- 초기항은 DP(0, 0) = 0, DP(0, 1) = DP(0, 2) = infinite로 정의할 수 있습니다.
- 그렇다면 정답은 DP(n, 0), DP(n, 1), DP(n, 2) 중 최댓값으로 정의됩니다.
- DP(n, m)을 구하는 데, O(1)의 시간 안에 계산할 수 있고, 총 3n개의 DP(n, m)을 계산해야 하므로, 시간 복잡도는 O(n)이 됩니다.
- n의 범위는 10000 이하이므로 충분히 제한시간을 통과할 수 있습니다.

파일 합치기

11066

https://www.acmicpc.net/problem/11066

- 이 문제는 K개의 파일을 최대한 적은 비용으로 모두 합치는 문제입니다. 총 K개의 파일이 연속적으로 존재하고, i번째 파일의 크기를 C[i]라고 합시다. 파일 합치기는 이웃한 파일만 합칠 수 있고, 이 때 합치는 비용은 합칠 두 파일의 크기의 합입니다.
- 예를 들어 [40, 30, 30, 50]의 4개의 파일이 존재할 때
  [40, 30, 30, 50] ==(비용:70)==> [70, 30, 50] ==(비용:80)==> [70, 80] ==(비용:150)==> [150]
  의 과정으로 총 70+80+150=300의 비용으로 모든 파일을 합칠 수 있습니다. 300보다 더 작은 비용으로 합치는 것은 불가능합니다. 따라서 이 경우 모든 파일을 합치는 비용의 최솟값은 300이 됩니다.

파일 합치기

11066

https://www.acmicpc.net/problem/11066

- 이 문제에서는 DP(a, b)를 다음과 같이 정의합니다.
  DP(a, b) = a번째 파일부터 b번째 파일까지 합치는 데 필요한 비용의 최솟값
- 초기 상태는 DP(a, a) = 0이 됩니다. 즉 파일 1개만 존재하는 상황이 초기 상태로 정의됩니다.
- a~b번째 파일을 하나로 합치기 직전의 상황을 생각해봅시다. a번째 파일부터 k번째 파일까지 하나의 파일로 합쳐져 있고, k+1번째 파일부터 b번째 파일까지 하나의 파일로 합쳐져 있을 것입니다. 따라서 이 때 파일을 합치는 비용은 (a번째 파일부터 k번째 파일까지 하나의 파일로 합치는 비용) + (k+1번째 파일부터 b번째 파일까지 하나의 파일로 합치는 비용) + (a번 파일부터 b번 파일까지 용량의 합) 으로 계산할 수 있습니다. 이 때 k는 a이상 b미만의 값이므로, 모든 k에 대해 위 식을 계산하고, 그 중 최솟값이 DP(a, b)가 됩니다. 따라서 점화식은 아래와 같이 나타낼 수 있습니다.

$$DP(a,b) = \min(DP(a,k) + DP(k+1,b) + C[a] + C[a+1] + \cdots + C[b])\ for\ a \le k < b$$

파일 합치기

11066

https://www.acmicpc.net/problem/11066

- 파일의 개수를 K라 하면, DP(a, b)를 구하기 위해서는 DP(a, k), DP(k + 1, b), C(a), … C(b)까지 계산해야 하므로, O(K)의 시간이 필요하고, 이를 모든 k에 대해서 계산해야 하므로 최대 O(K^2)의 시간이 필요합니다.
- 모든 a, b에 대해 DP(a, b)를 계산해야 하므로 시간복잡도는 O(K^2 * K^2) = O(K^4)가 됩니다.
- K는 최대 500이므로, K^4 = 6천억이 되어 시간 초과가 발생합니다. 그렇다면 시간을 어떻게 줄일 수 있을까요?
- DP(a, k) + DP(k + 1, b) + C(a) + … + C(b)부분에서 C(a) + … +C(b)를 보다 빠르게 구하는 방법을 알아봅시다.

파일 합치기

11066

https://www.acmicpc.net/problem/11066

- S[i] = C[1] + C[2] + ⋯ + C[i]라 정의합니다. 그렇다면, S[i] = (C[1] + C[i-1]) + C[i] = S[i-1] + C[i]의 점화식을 도출할 수 있습니다. 이를 이용해서 S[i]를 모든 i에 대해 미리 구해둘 수 있습니다.
- S[i]를 미리 구해두었다면, C[a] + ⋯ + C[b]는 쉽게 구할 수 있습니다.
- C[a] + ⋯ C[b] = (C[1] + ⋯ + C[b]) – (C[1] + ⋯ + C[a-1]) = S[b] = S[a-1]로 계산할 수 있습니다. (S[0] = 0으로 정의합니다.)
- 그렇다면 점화식은 아래와 같이 바뀝니다.

$$DP(a,b) = \min(DP(a,k) + DP(k+1,b) + S[b] - S[a-1])\ for\ a \le k < b$$

- 그러면 시간복잡도는 O(K^3)으로 줄어들게 됩니다. 따라서 500^3 = 1억2500만으로 구현만 잘 한다면 제한시간안에 통과할 수 있습니다. (저의 경우에는 428ms가 소요되었습니다.)

마치며

- DP 소스 코드는 아래의 주소에서 확인할 수 있습니다.
  https://github.com/qjatn0120/algorithm_open_seminar/tree/master/1. DP/source_code

POSCAT

Thank you :-)