실전 알고리즘 0x09강
다이나믹 프로그래밍

BaaaaaaaaaaaaaaaarkingDog

목차

목차

0x00 다이나믹 프로그래밍 (Dynamic Programming)

- 다이나믹 프로그래밍(=Dynamic Programming, DP라고 줄여서 부르기도 함)은 여러 개의 하위 문제들을 먼저 푼 후 그 결과를 쌓아올려 주어진 문제를 해결하는 알고리즘입니다.
- 쉽게 설명하면, 문제를 해결하기 위한 점화식을 찾아낸 후 점화식의 항을 밑에서부터 차례로 구해나가 답을 알아내는 형태입니다.
- 예를 들어 피보나치 문제를 생각해봅시다. 0x06강에서 재귀함수로 구현했을 경우 N번째 항을 구하는데 $O(1.618^N)$이 필요했지만 DP로 구현하면 $O(N)$에 구할 수 있습니다.

```
int f[20];
f[0] = 1;
f[1] = 1;
int n = 15;
for(int i = 2; i <= n; i++) f[i] = f[i-1]+f[i-2];
```

0x00 다이나믹 프로그래밍 (Dynamic Programming)

- 코딩테스트에 나올 수준의 DP 문제는 일단 점화식만 이끌어놓고 나면 그 뒤는 반복문을 돌면서 배열을 채워나가면 되기 때문에 구현이 굉장히 간단한 편에 속합니다.
- 그러나 다양한 DP 문제를 풀어봤거나 뛰어난 수학적 직관력을 가지고 있지 않은 이상 문제에서 점화식을 이끌어내는 과정이 쉽지 않고, 무엇보다 초보 단계에서는 주어진 문제가 DP로 푸는 문제임을 알아차리지 못할 수도 있습니다.
- DP는 쉬운 문제는 쉽지만 작정하고 어렵게 내면 저세상 난이도의 문제를 만들어낼 수 있습니다.
- 이번 강의에서는 최대한 많은 문제를 다뤄보는데 주력할 것입니다.
- 여러분들도 그룹 내의 문제집에 올려둔 DP 문제들을 풀고 머릿속으로 고민하면서 많은 유형을 학습하고 숙달하는 것이 중요합니다.

0x01 예시 문제 1 : 1로 만들기

- BOJ 1463번 : 1로 만들기를 풀어봅시다.
- 이 문제는 BFS로 해결할 수 있습니다. 그런데 DP로 해결하면 코드가 정말 짧아집니다.
- DP로 해결하기 위해서는 일단 배열에 어떤 값을 넣어야 할지, 또 관계식은 어떻게 될지를 잘 생각해야 합니다.
- 첫 번째 문제이니 같이 한 번 해봅시다.
- `D[i] = i`를 1로 만들기 위해 필요한 연산 사용 횟수의 최솟값 이라고 합시다. `D[1]`은 자명하게 0입니다.
- `k = 2 to N`에 대해 `D[k]`를 어떻게 계산할 수 있을까요?

0x01 예시 문제 1 : 1로 만들기

- `D[12]`(=12를 1로 만들기 위해 필요한 연산의 최소 횟수) 를 계산한다고 쳐봅시다. 할 수 있는 연산은 3가지니
- 1. 3으로 나누거나 (= `D[4]+1`)
- 2. 2로 나누거나 (= `D[6]+1`)
- 3. 1을 빼거나 (= `D[11]+1`)
- `D[12] = min(D[4]+1, D[6]+1, D[11])` 입니다.

0x01 예시 문제 1 : 1로 만들기

- 비슷한 방식으로 생각해보면 `D[k]`는 일단 `D[k-1]+1`이고, `k`가 2로 나누어진다면 `D[k/2]+1`을, 3으로 나누어진다면 `D[k/3]+1`을 추가로 고려해 이들 중에서 최솟값을 택하면 됩니다.
- 정답 코드 : http://boj.kr/dde1252ec5544b99a876baf57cee0571

0x02 예시 문제 2 : 계단 오르기

- BOJ 2579번 : 계단 오르기를 풀어봅시다.
- 만약 계단의 갯수가 그다지 많지 않았으면 백트래킹으로 해결이 가능할텐데 계단의 갯수가 300개니 불가능합니다.
- 이 문제는 DP로 풀 수 있습니다. 배열에 어떤 값을 넣어야 할지, 또 관계식은 어떻게 될지를 알아맞춰보세요.

0x02 예시 문제 2 : 계단 오르기

- 우선 서술의 편의를 위해 `i`번째 계단에 적힌 점수를 `S[i]`이라고 하겠습니다.
- **`D[i]` = `i`번째 계단까지 올라섰을 때의 점수 합의 최댓값** 이라고 합시다. `D[1]`은 자명하게 `S[1]`입니다.
- `k = 2 to N`에 대해 `D[k]`를 어떻게 계산할 수 있을까요?
- 문제에서 주어진 예시 (10, 20, 15, 25, 10, 20)을 생각해봅시다.
- `D[2]`는 1번째, 2번째 계단을 다 밟으면 되니 30일 것입니다.
- `D[3]`은 어떤 식으로 구해야할까요? `D[3]`은 연속한 세 계단을 모두 밟아서는 안 된다는 제약 조건으로 인해 20+15 = 35가 답이 됩니다. 그런데 지금 `D[i]`의 값을 가지고는 점화식을 세우고 싶어도 연속한 세 계단을 모두 밟아서는 안 된다는 제약 조건을 점화식에 넣을 수가 없습니다.

0x02 예시 문제 2 : 계단 오르기

- 그렇기에 현재의 `D[i]`는 문제를 풀기에 적절하지 못합니다.
- `D[i][j]` = 현재까지 `j`개의 계단을 연속해서 밟고 `i`번째 계단까지 올라섰을 때의 점수 합의 최댓값, 단 `i`번째 계단은 반드시 밟아야 함 이라고 합시다. `D[1][1]`은 자명하게 `S[1]`이고 `D[1][2]`는 0입니다. 그리고 `j` 는 1 혹은 2입니다. 3 이상이면 연속된 세 개의 계단을 못밟는다는 조건에 위배되기 때문입니다.
- `k = 2 to N`에 대해 `D[k][1], D[k][2]`를 어떻게 계산할 수 있을까요?
- 문제에서 주어진 예시 (10, 20, 15, 25, 10, 20)에서 직접 `D` 테이블을 채워넣는 과정을 손으로 해보세요.

0x02 예시 문제 2 : 계단 오르기

- 예시 : (10, 20, 15, 25, 10, 20)
- `D[2][1]` 은 1번째 계단을 밟지 않았다는 의미이니 20이고 `D[2][2]`는 1번째, 2번째 계단을 다 밟으면 되니 30일 것입니다.
- `D[3][1]`은 2번째 계단을 밟지 않았다는 의미이고 계단은 한 번에 한 계단 혹은 두 계단씩 오를 수 있다는 조건으로 인해 1번째 계단은 반드시 밟아야 합니다. 1번째 계단을 밟을 당시에 점수의 최댓값에 `S[3]`을 더하면 됩니다. 즉 `D[3][1] = max(D[1][1], D[1][2])+S[3] = 25` 입니다.
- `D[3][2]`은 2번째 계단을 밟았다는 의미입니다. 그리고 연속한 세 계단을 모두 밟아서는 안 된다는 제약 조건으로 인해 `D[3][2] = D[2][1]+S[3] = 30` 입니다.

0x02 예시 문제 2 : 계단 오르기

- 이제 점화식이 감이 오나요?
- `D[k][1] = max(D[k-2][1],D[k-2][2])+S[k]`
- `D[k][2] = D[k-1][1]+S[k]`
- 입니다. 그리고 답은 `max(D[N][1], D[N][2])` 입니다.
- 정답 코드 : http://boj.kr/e180e45a1eb64761b9c936d5339b0ff5

0x02 예시 문제 2 : 계단 오르기

- 이 문제를 해결하는 또 다른 DP 테이블을 생각해봅시다. 생각해보면 N번째 계단까지 점수의 최댓값은 곧 밟지 않는 계단의 점수 합의 최솟값을 구하는 문제와 동일합니다.
- 즉 관점을 달리해 밟지 않을 계단을 선택하도록 DP를 짜봅시다.
- `D[i] = i`번째 계단까지 올라가면서 밟지 않을 계단의 합의 최솟값, 단 `i`번째 계단은 반드시 밟지 않을 계단으로 선택해야 함 이라고 합시다. `D[1]`은 자명하게 `S[1]`입니다.
- 계단은 한 번에 한 계단씩 또는 두 계단씩 오를 수 있다는 조건으로 인해 `k`번째 계단을 밟지 않을 계단으로 선택했으면 `k+1`번째 계단을 연달아 선택할 수는 없습니다.
- 연속한 세 계단을 모두 밟을 수는 없다는 조건으로 인해 `k`번째 계단을 밟지 않을 계단으로 선택했으면 `k+2`, `k+3`번째 계단 중 어느 하나는 선택해야 합니다.

0x02 예시 문제 2 : 계단 오르기

- 예시 (10, 20, 15, 25, 10, 20)에서 DP 테이블을 채워봅시다.
- `D[2]`는 자명하게 20입니다.
- `D[3]`은 자명하게 15입니다.
- `D[4]`는 `min(D[1],D[2])+S[4]` 라는거 아시겠나요? 4번째 계단을 제외하고 나머지 계단 중에서 가장 최근에 밟지 않을 계단으로 선택한게 1번째 계단이거나 2번째 계단일테니까요.
- 마찬가지 논리로 생각하면 `D[k] = min(D[k-2],D[k-3])+S[k]` 입니다. 그리고 답은 전체 계단의 합 – `min(D[N-1],D[N-2])` 입니다. 마지막 도착 계단은 반드시 밟아야한다는 조건 때문입니다.
- 정답 코드 : http://boj.kr/7c32cdd30d2741338c88f5965cf214f6

0x03 예시 문제 3 : RGB거리

- BOJ 1149번 : RGB거리를 풀어봅시다.
- 배열에 어떤 값을 넣어야 할지, 또 관계식은 어떻게 될지를 알아맞춰보세요. 풀만한 난이도이니 고민을 충분히 하고 다음 장으로 넘어와주세요.

0x03 예시 문제 3 : RGB거리

- `D[i] = i`번째 집까지 칠할 때 비용의 최솟값 으로 두면 딱히 이끌어낼 수 있는 식이 없습니다.
- `D[i][0] = i`번째 집까지 칠할 때 비용의 최솟값, 단 `i`번째 집은 빨강
- `D[i][1] = i`번째 집까지 칠할 때 비용의 최솟값, 단 `i`번째 집은 초록
- `D[i][2] = i`번째 집까지 칠할 때 비용의 최솟값, 단 `i`번째 집은 파랑
- 으로 두고 나면 식을 세울 수 있을 것 같네요. 식을 생각해보세요.

0x03 예시 문제 3 : RGB거리

- `D[k][0] = min(D[k-1][1],D[k-1][2]) + i`번째 집을 빨강으로 칠하는 비용
- `D[k][1] = min(D[k-1][0],D[k-1][2]) + i`번째 집을 파랑으로 칠하는 비용
- `D[k][2] = min(D[k-1][0],D[k-1][1]) + i`번째 집을 초록으로 칠하는 비용
- 정답은 `max(D[N][0], D[N][1], D[N][2])`입니다.
- 초기값에 무엇을 담을지에 주의해 직접 짜보세요.
- 정답 코드 : http://boj.kr/f40027a5ab3a449e89ac1d4c77738cea

0x04 예시 문제 4 : 2×n 타일링

- BOJ 11726번 : 2×n 타일링을 풀어봅시다.
- 지금 다이나믹 프로그래밍을 배우고 있으니 저 문제가 DP 문제이겠구나 싶긴 하겠지만, 아무 정보 없이 문제를 접했다면 아마 DP 문제라는 것을 눈치채지 못했을 것입니다.
- 조금 어렵지만 직접 종이에 끄적거리면서 한 번 노력해보세요.

0x04 예시 문제 4 : 2×n 타일링

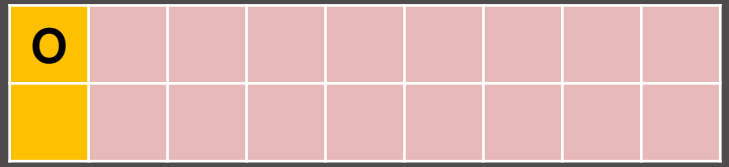
- 늘 그렇듯 D 테이블을 잘 정해야합니다. `D[i] = 2×i` 를 타일링하는 경우의 수 라고 합시다. 이제 점화식을 찾아내봅시다.

O								

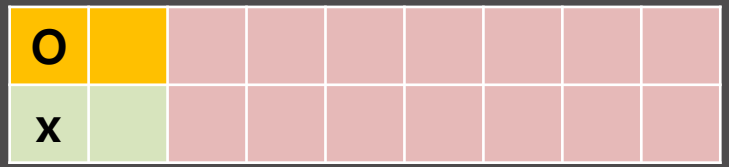
- 다른건 생각하지 말고, 저 O 쳐진 칸을 어떤 타일로 덮을지 생각해보세요.

0x04 예시 문제 4 : 2×n 타일링

- 1. 2×1 타일로 해당 칸을 덮었을 경우



- 남은 2×(n-1) 를 타일링하는 경우의 수만 세면 되겠네요. (`D[8]`)
- 2. 1×2 타일로 해당 칸을 덮었을 경우



- x 쳐진 칸도 자동으로 1×2 타일을 덮을 수 밖에 없게 됩니다. 남은 2×(n-2) 를 타일링하는 경우의 수만 세면 되겠네요. (`D[7]`)

0x04 예시 문제 4 : 2×n 타일링

- 그러므로 `D[k] = D[k-1]+D[k-2]` 입니다.
- 정답 코드 : http://boj.kr/57a8d29e0efb46299ef3c5e609ac9f74
- 다른 타일링 문제들을 풀 때도 특정 칸 하나를 각 타일로 덮었을 때의 경우의 수를 세서 식을 이끌어내면 됩니다.

0x05 예시 문제 5 : 연속합

- BOJ 1912번 : 연속합을 풀어봅시다. 이 문제는 학부 수업시간에서 자주 다루는 문제 중 하나입니다.
- 지금까지 코딩을 한 짬이 있는데 $O(N^3)$ 풀이는 바로 떠올릴 수 있죠?(구간의 시작점, 끝점을 잡아 다 더하는 3중 for문 풀이)

```
int n;
cin >> n;
for(int i = 1; i <= n; i++) cin >> a[i];
int mx = -0x7f7f7f7f; // 이거 0으로 두면 큰일남
for(int st = 1; st <= n; st++){
  for(int en = st; en <= n; en++){
    int tot = 0;
    for(int i = st; i <= en; i++) tot += a[i];
    mx = max(mx,tot);
  }
}
```

0x05 예시 문제 5 : 연속합

- Prefix sum이라는 기법을 사용하면 $O(N^2)$ 까지는 그럭저럭 무난하게 할 수 있습니다. Prefix sum을 사용하기 위해서는 1-indexed인게 편합니다.
- Prefix sum을 처음 본다면 이번 기회에 익혀두세요.

```
int n;
cin >> n;
d[0] = 0;
for(int i = 1; i <= n; i++){
  cin >> a[i];
  d[i] = d[i-1]+a[i]; // d[i] = a[1]+a[2]+...+a[i]
}
int mx = -0x7f7f7f7f; // 이거 0으로 두면 큰일남
for(int st = 1; st <= n; st++){
  for(int en = st; en <= n; en++){
    mx = max(mx,d[en]-d[st-1]);
  }
}
```

0x05 예시 문제 5 : 연속합

- $O(N^2)$ 도 나쁘지 않지만 $N \leq 100{,}000$인 이 문제를 풀기엔 아직 부족합니다.
- DP로 무려 $O(N)$ 에 풀 수 있는 방법을 고민해봅시다.
- 배열에 어떤 값을 넣어야 할지, 또 관계식은 어떻게 될지를 알아맞춰보세요. 조금 어렵지만 스스로 알아맞추면 정말 기쁠테니 해답을 보기 전에 고민을 충분히 많이 해보세요.

0x05 예시 문제 5 : 연속합

- `D[i] = i`번째 항을 마지막으로 사용하는 수열의 합 중 최댓값 으로 두면 딱 느낌이 오나요?
- `D[k] = max(0, D[k-1])+ A[k]` 입니다. `k-1`번째 항을 마지막으로 하는 수열의 합 중 최댓값이 0보다 크면 그 수열에 `A[k]`를 붙이면 되고, 0 이하이면 그냥 `A[k-1]` 을 쓰지 말고 `A[k]`만으로 수열을 만들면 되기 때문입니다.
- 정답 코드 : http://boj.kr/645f1b8f53034a41b486d65a98303bf2

0x06 예시 문제 6 : 가장 긴 증가하는 부분 수열

- BOJ 11053번 : 가장 긴 증가하는 부분 수열을 풀어봅시다.
- `D[i] = i`번째 항을 마지막으로 사용하는 수열 중 최대 길이 로 두면 딱 느낌이 오나요?

- 정답 코드 : http://boj.kr/29cfe3ff46f249d18658ce00f6fb013a
- 참고로 이 문제는 Segment Tree나 Binary Search를 이용해 O(*NlgN*)로도 해결할 수 있습니다. 먼 훗날에 공부해보세요!

0x07 예시 문제 7 : 동전 1

- BOJ 2293번 : 동전 1을 풀어봅시다.
- N이 작다면 $O(2^N)$에 전수조사를 할 수 있을텐데 여의치가 않네요.
- `D[i][j] = i`번째 동전까지를 사용해 가치의 합이 `j`가 되게끔 하는 경우의 수 로 두면 느낌이 오나요?
- 예를 들어 현재 3번째 동전을 보고 있고 3번째 동전의 가치가 10일 때, `D[3][25]`은 얼마일까요?

0x07 예시 문제 7 : 동전 1

- 1. 3번째 동전을 0개 사용하는 경우 : 2번째 동전까지 써서 합 25를 만들기 = `D[2][25]`
- 2. 3번째 동전을 1개 사용하는 경우 : 2번째 동전까지 써서 합 15를 만들기 = `D[2][15]`
- 3. 3번째 동전을 2개 사용하는 경우 : 2번째 동전까지 써서 합 5를 만들기 = `D[2][5]`
- 즉 `D[3][25] = D[2][25]+D[2][15]+D[2][5]` 입니다.
- 그런데 이렇게 구현하면 `D[i][j]`를 채우기 위해 `j/A[i]` 개의 항의 합이 필요하고 모든 동전의 가치가 1일때를 생각해보면 $O(NK^2)$에 동작해서 시간초과가 발생합니다.

0x07 예시 문제 7 : 동전 1

- 관점을 조금 바꿔, 동전을 0개/1개/2개/ … 사용했는지를 가지고 DP식을 만드는게 아니라 `i`번째 동전을 사용했다/안했다로 DP식을 만들어야 합니다.
- 1. 3번째 동전을 사용하지 않은 경우 : 2번째 동전까지 써서 합 25를 만들기 = `D[2][25]`
- 2. 3번째 동전을 사용한 경우 : 3번째 동전까지 써서 합 15를 만들기 = `D[3][15]`
- 즉 `D[i][j] = D[i-1][j]+D[i][j-A[i]]`입니다. (`A[i]`는 `i`번째 동전의 무게, 그리고 `j < A[i]` 일 경우 `i`번째까지의 동전으로 `j-A[i]`원을 만드는 것이 불가능하므로 뒤의 항은 무시하도록 해야합니다.)
- 그런데 생각할 점이 한 가지 더 있습니다. 메모리 제한이 4MB이기 때문에 `D[100][10001]`을 잡으면 메모리 초과가 발생합니다.

0x07 예시 문제 7 : 동전 1

- 다시 DP식을 봅시다. `D[i][j]` 는 `D[i+1][..]`을 구할 때에만 필요하지 `D[i+2][..]`, `D[i+3][..]` .. 를 구할 때에는 더 이상 쓸모가 없습니다.
- 그러므로 `D1[k+1]`, `D2[k+1]` 2개만 가지고 처리를 해줄 수가 있습니다. 정답 코드를 참고하세요.
- 정답 코드 : http://boj.kr/08720c2231ff4011a6d07834e697c6b5
- 동전 관련 문제는 코딩 테스트에서 자주 나오는 문제입니다.
- 코딩 테스트 수준에서는 이런 식으로 메모리를 아끼는 기법까지 요구하지 않지만, 알아둬서 나쁠건 없습니다.

강의 정리

- DP의 개념을 배우고 다양한 DP 문제들에 대해 풀어봤습니다.
- 7문제를 같이 다루긴 했지만 7문제로는 부족합니다. 정말 수많은 DP 문제들이 있고 많이 풀면 풀수록 문제해결능력을 올릴 수 있으니 계속 풀어보세요.