Dynamic Programming

- Basic -

안태진(taejin7824@gmail.com)
GitHub(https://github.com/Taejin1221)
소프트웨어학과 18학번
상명대학교 CodeCure 소프트부장

Contents

지난 과제 풀이

- 1번: 동전 0 (1/2)
  - 특정 돈을 최소 개수로 만들기 위해서 -> 가장 큰 동전부터 최대한 넣자!
    - 이것이 되는 이유 -> A_i는 A_{i-1}의 배수이기 때문
    - why?
      - 오른쪽 예제에서 4,200원을 1로만 만들어보자 -> 1 * 4200
      - 그 1원을 5개 모으면 5원이 됨으로 5원짜리로 만드는 것이 더 적음
        - 5 * 840
      - 그 5원을 2개 모으면 10원이 됨으로 10원짜리로 만드는 것이 더 적음
        - 10 * 420
      - 그 10원을 5개 모으면 50원이 됨으로 50원짜리로 만드는 것이 더 최소 개수
        - 50 * 84
      - …

예제 입력 1 복사

```
10 4200
1
5
10
50
100
500
1000
5000
10000
50000
```

지난 과제 풀이

- 1번: 동전 0 (2/2)
  - 따라서 k보다 작은 가장 큰 동전을 최대한 넣는 것이 가장 적게 넣는 방법임!

  - Code

```cpp
#include <iostream>

using namespace std;

int main(void) {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, k;
    cin >> n >> k;

    int coin[10];
    for (int i = 0; i < n; i++)
        cin >> coin[i];
```

```cpp
    int ans = 0;
    for (int i = n - 1; i >= 0; i--) {
        if (k >= coin[i]) {
            int quotient = k / coin[i]; // 몫
            k -= quotient * coin[i];
            ans += quotient;
        }
    }

    cout << ans << '\n';

    return 0;
}
```

지난 과제 풀이

- 2번: ATM (1/2)
  - 인출하는데 필요한 시간의 합
    - $\sum_{i=0}^{n-1} wait[i] \times (n-i)$
    - e.g., 오른쪽 예제
      - $3\times5 + 1\times4 + 4\times3 + 3\times2 + 2\times1$
    - 따라서 이를 최소한으로 하기 위해선 작은 수에 큰 수를 곱하도록 해야함
    - 오름차순으로 정렬!

예제 입력 1 복사

```
5
3 1 4 3 2
```

지난 과제 풀이

- 2번: ATM (2/2)
  - Code

```cpp
#include <iostream>

#include <algorithm>

using namespace std;

int main(void) {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    cin >> n;

    int time[1'000];
    for (int i = 0; i < n; i++)
        cin >> time[i];
```

```cpp
    sort(time, time + n);

    int ans = 0;
    for (int i = 0; i < n; i++)
        ans += time[i] * (n - i);

    cout << ans << '\n';

    return 0;
}
```

지난 과제 풀이

- 3번: 회의실 배정
  - 최대한 많은 회의를 잡으려면
    - 끝나는 시간 기준으로 정렬, 첫번째 회의부터 잡으며 바로 다음에 잡을 수 있는 회의를 잡자
    - 대신 끝나는 시간이 같다면 시작 시간이 우선으로!
      - 3, (1, 2), (4, 4), (2, 4)로 주어지면 사실 답은 3인데 앞의 두개의 회의만 잡기 때문에 2라고 나옴!
  - Code

```cpp
#include <iostream>

#include <algorithm>

using namespace std;

typedef pair<int, int> pii;

bool compare(const pii& p1, const pii& p2) {
    if (p1.second != p2.second)
        return p1.second < p2.second;
    else
        return p1.first < p2.first;
}

int main(void) {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    cin >> n;

    pii meetings[100'000];
    for (int i = 0; i < n; i++)
        cin >> meetings[i].first >> meetings[i].second;

    sort(meetings, meetings + n, compare);

    int prevEnd = 0, ans = 0;
    for (int i = 0; i < n; i++)
        if (prevEnd <= meetings[i].first)
            ans++, prevEnd = meetings[i].second;

    cout << ans << '\n';

    return 0;
}
```

지난 과제 풀이

- 4번: 주유소 (1/2)
  - Solution
    - 최소한의 비용으로 가려면 기름이 가장 싼 도시에서 가장 많이 기름을 넣자!
    - 하지만 중간에 기름이 없으면 안됨
      - 따라서 A 도시보다 기름 값이 싼 도시 B까지의 거리만큼 기름을 넣고,
      - B에서 다시 B보다 기름 값이 싼 도시까지의 거리만큼 기름을 넣고
      - 반복…
  - 수식
    - $(\text{B 도시까지 최소 가격}) = \min(prevPrice, price[B]) * distanceTo[B]$
      - $prevPrice$: 지나온 도시들 중 최소 기름 가격
      - $price[a]$: a 도시에서의 기름 가격,
      - $distanceTo[a] = a$ 이전 도시에서 a까지의 거리

지난 과제 풀이

- 4번: 주유소 (2/2)
  - Code

```cpp
#include <iostream>

using namespace std;

typedef long long ll;

int main(void) {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    cin >> n;

    int distanceTo[100'000];
    for (int i = 1; i < n; i++)
        cin >> distanceTo[i];

    int price[100'000];
    for (int i = 0; i < n; i++)
        cin >> price[i];
```

```cpp
    ll prevPrice = price[0], ans = 0;
    for (int i = 1; i < n; i++) {
        ans += prevPrice * distanceTo[i];
        if (prevPrice > price[i])
            prevPrice = price[i];
    }

    cout << ans << '\n';

    return 0;
}
```

지난 과제 풀이

- 5번: Project Teams (1/2)
  - Solution
    - 학생의 코딩 역량의 합을 일정하게 유지하려면
      - 가장 못하는 학생 + 가장 잘하는 학생
      - 두번째로 못하는 학생 + 두번째로 잘하는 학생
      - …
    - 으로 팀을 묶으면 가장 균일하게 유지될 수 있음

    - $S_m = \min\{w(G_i) | \, 1 \le i \le n\}$을 구하는 것이기 때문에 위의 팀 역량 중 최소값을 구하면 됨

지난 과제 풀이

- 5번: Project Teams (2/2)
  - Code

```cpp
#include <iostream>

#include <algorithm>

using namespace std;

int main(void) {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    cin >> n;

    int weight[100'000];
    for (int i = 0; i < 2 * n; i++)
        cin >> weight[i];
```

```cpp
    sort(weight, weight + 2 * n);

    int ans = 1'234'567'890;
    for (int i = 0; i < n; i++)
        ans = min(ans, weight[i] + weight[2 * n - 1 - i]);

    cout << ans << '\n';

    return 0;
}
```

Contents

- 지난 과제 풀이

- Dynamic Programming
  - 재귀의 문제
  - Theory
  - Top-Down vs Bottom-Up
  - Example
  - Tip

- 과제

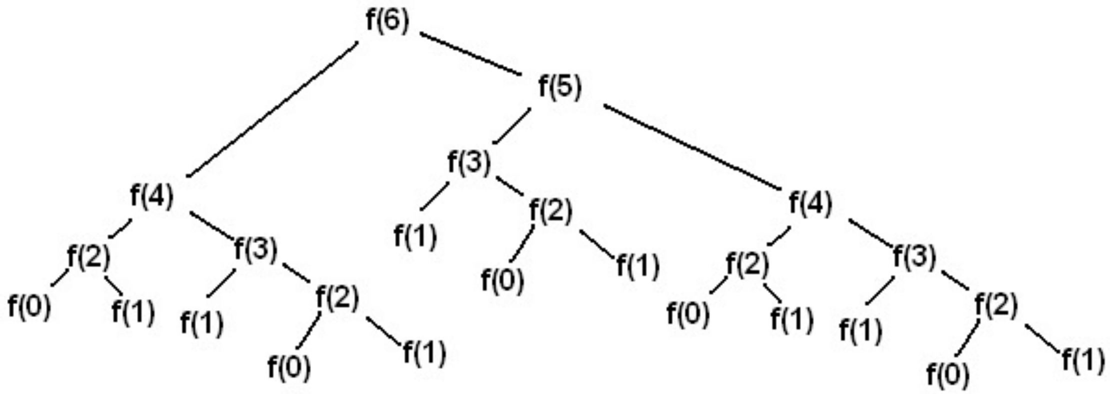
Dynamic Programming

- 재귀의 문제 (1/3)
  - Fibonacci 문제 풀기
    - BOJ 피보나치 수 (BOJ 2747)
    - Fibonacci 수 구하기
      - $fib(n) = \begin{cases} 0 (n = 0) \\ 1 (n = 1) \\ fib(n-1) + fib(n-2) (n \geq 2) \end{cases}$
    - 결과
      - n이 45임에도 불구하고 시간 초과!
      - Why?

```
int fib(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}
```

Dynamic Programming

- 재귀의 문제 (2/3)
  - Fibonacci 문제 풀기
    - 단순 재귀로 구현하면 수많은 반복 호출이 발생!



  - 실제로 각 f(n)이 몇번 호출 되는지 출력
    - n이 10일 때
    - n이 45일 때

```
0 : 34
1 : 55
2 : 34
3 : 21
4 : 13
5 : 8
6 : 5
7 : 3
8 : 2
9 : 1
10 : 1
```

```
0 : 701408733
1 : 1134903170
2 : 701408733
3 : 433494437
4 : 267914296
5 : 165580141
6 : 102334155
7 : 63245986
8 : 39088169
9 : 24157817
10 : 14930352
11 : 9227465
12 : 5702887
13 : 3524578
14 : 2178309
15 : 1346269
```

```
16 : 832040
17 : 514229
18 : 317811
19 : 196418
20 : 121393
21 : 75025
22 : 46368
23 : 28657
24 : 17711
25 : 10946
26 : 6765
27 : 4181
28 : 2584
29 : 1597
30 : 987
```

```
31 : 610
32 : 377
33 : 233
34 : 144
35 : 89
36 : 55
37 : 34
38 : 21
39 : 13
40 : 8
41 : 5
42 : 3
43 : 2
44 : 1
45 : 1
```

⇒ 총 호출 횟수 3,672,623,805!
⇒ Time Complexity = $O(2^n)$
(시간 초과가 안뜨면 이상하다!)

Dynamic Programming

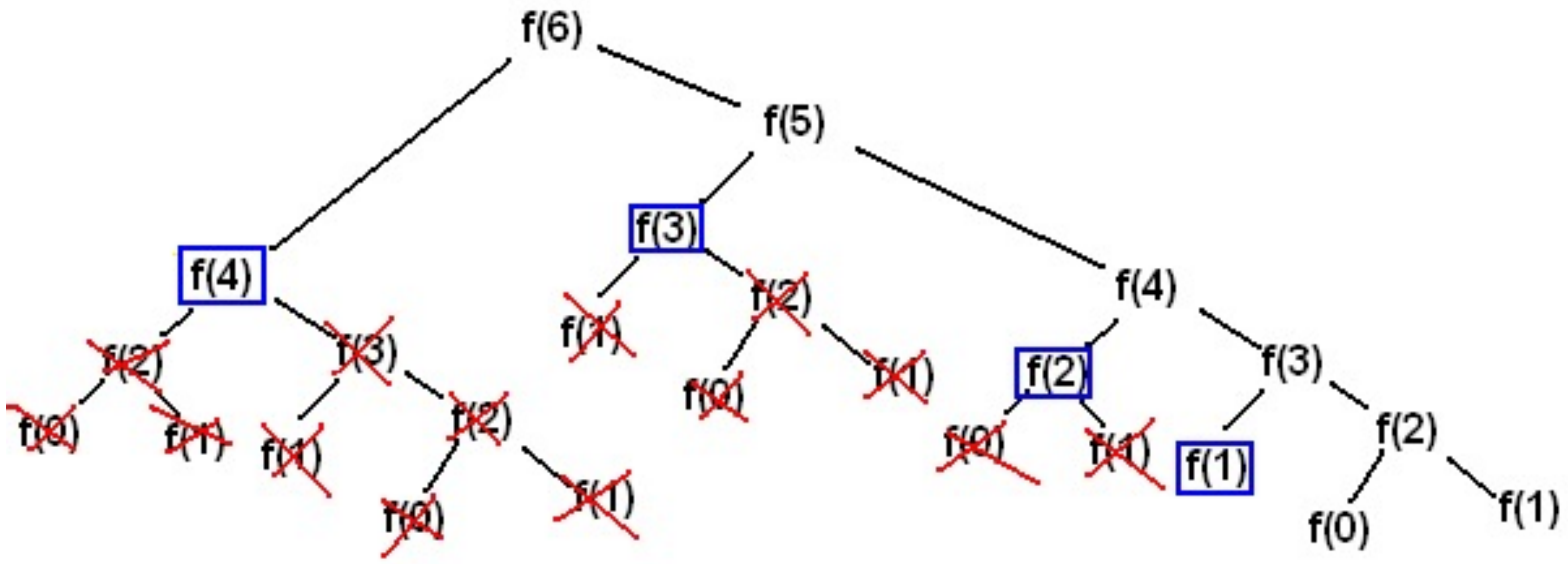
- 재귀의 문제 (3/3)
  - 문제점
    - fib(n)을 구할 땐 0 ~ (n - 1) 까지의 fibonacci 수를 알지 못함
    - 따라서 0 ~ (n – 1)까지의 fibonacci를 구해야하는 건 틀리지 않음
    - 하지만 이미 구한 값을 또 구함
    - 이것이 문제점!
  - 해결법
    - 이미 구한 값들을 다시 재사용할 순 없을까?
      - => 메모리(배열)에 저장하자!

Contents

- 지난 과제 풀이
- Dynamic Programming
    - 재귀의 문제
    - Theory
    - Top-Down vs Bottom-Up
    - Example
    - Tip
- 과제

Dynamic Programming

- Theory (1/6)
  - 동적 계획법, 동적 프로그래밍 (DP, Dynamic Programming)
    - 재귀 함수의 반복 호출을 줄이기 위해 이미 구한 값들을 메모리에 저장해 놓는 방법
    - 왜 Dynamic Programming이냐고?
      - 그니까? 왤까? 나도 궁금함



Dynamic Programming

- Theory (2/6)
  - Fibonacci 다시 풀기

  - Pseudo Code

```
function Fib(int n):
    if n을 구하지 않았다면
        Fib(n - 1) + Fib(n - 2)를 호출
        호출 한 뒤 값 저장

    return 저장한 피보나치 n 값
```

Dynamic Programming

- Theory (3/6)
  - Fibonacci Code 1 (C++)

```cpp
#include <iostream>

#define NOT_FIND -1

using namespace std;

int memo[46];

int fib(int n) {
    if (memo[n] == NOT_FIND)
        memo[n] = fib(n - 1) + fib(n - 2);

    return memo[n];
}
```

```cpp
int main(void) {
    int n;
    cin >> n;

    fill(memo, memo + 46, NOT_FIND);
    memo[0] = 0;
    memo[1] = 1;

    cout << fib(n) << '\n';

    return 0;
}
```

Dynamic Programming

- Theory (4/6)
  - Fibonacci Code 1 (Python3)

```python
NOT_FIND = -1

def fib(n):
    if (memo[n] == NOT_FIND):
        memo[n] = fib(n - 1) + fib(n - 2)

    return memo[n]

n = int(input())

memo = [-1 for i in range(46)]
memo[0] = 0;
memo[1] = 1;

print(fib(n))
```

Dynamic Programming

- Theory (5/6)
  - Fibonacci Code 2 (C++)

```cpp
#include <iostream>

using namespace std;

int main(void) {
    int n;
    cin >> n;

    int dp[46] = { 0, 1, };
    for (int i = 2; i <= n; i++)
        dp[i] = dp[i - 1] + dp[i - 2];

    cout << dp[n] << '\n';

    return 0;
}
```

Dynamic Programming

- Theory (6/6)
  - Fibonacci Code 2 (Python3)

```
n = int(input())

dp = [0 for i in range(46)]
dp[0], dp[1] = 0, 1

for i in range(2, n + 1):
    dp[i] = dp[i - 1] + dp[i - 2]

print(dp[n])
```

Contents

- 지난 과제 풀이

- Dynamic Programming
    - 재귀의 문제
    - Theory
    - Top-Down vs Bottom-Up
    - Example
    - Tip

- 과제

Dynamic Programming

- Top-Down vs Bottom-Up
  - Top-Down Approach
    - Memoization라고도 함
    - 큰 값에서 필요한 값들을 구해가며 DP table을 채워가는 방식
  - Bottom-Up Approach
    - 작은 값에서 큰 값으로 순차적으로 나아가며 DP table을 채워가는 방식
  - 더 쉬운 걸로 하시길!
    - Bottom-Up은 빠르고, Top-Down은 직관적이다!
    - 하지만 두 가지 방법 다 알고있는 것이 도움 됨!
    - 모든 문제를 두 가지 방법으로 풀어보자!

Contents

- 지난 과제 풀이
- Dynamic Programming
  - 재귀의 문제
  - Theory
  - Top-Down vs Bottom-Up
  - Example
  - Tip
- 과제

Dynamic Programming

- Example 1
  - 피보나치 수열 구하기
    - BOJ 15624
  - Solution
    - 이전과 동일하게 풀자!

문제

피보나치 수는 0과 1로 시작한다. 0번째 피보나치 수는 0이고, 1번째 피보나치 수는 1이다. 그 다음 2번째 부터는 바로 앞 두 피보나치 수의 합이 된다.

이를 식으로 써보면 $F_n = F_{n-1} + F_{n-2}$ ($n \geq 2$)가 된다.

n=17일때 까지 피보나치 수를 써보면 다음과 같다.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597

n이 주어졌을 때, n번째 피보나치 수를 구하는 프로그램을 작성하시오.

입력

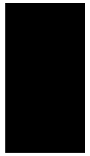
첫째 줄에 n이 주어진다. n은 1,000,000보다 작거나 같은 자연수 또는 0이다.

출력

첫째 줄에 n번째 피보나치 수를 1,000,000,007으로 나눈 나머지를 출력한다.

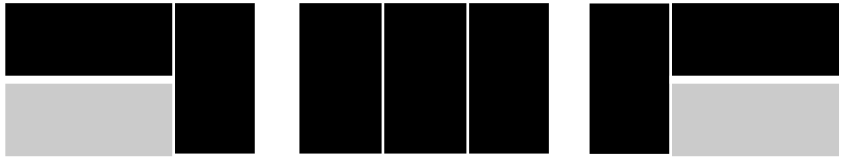
Dynamic Programming

- Example 2 (1/2)
  - 2xn 타일링
    - [BOJ 11726]
  - Solution
    - n = 1



    - n = 2

    - n = 3



문제

2×n 크기의 직사각형을 1×2, 2×1 타일로 채우는 방법의 수를 구하는 프로그램을 작성하시오.

아래 그림은 2×5 크기의 직사각형을 채운 한 가지 방법의 예이다.

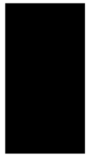
입력

첫째 줄에 n이 주어진다. (1 ≤ n ≤ 1,000)

출력

첫째 줄에 2×n 크기의 직사각형을 채우는 방법의 수를 10,007로 나눈 나머지를 출력한다.

Dynamic Programming

- Example 2 (2/2)
  - Solution
    - n번째 타일은
      - n-1번째 타일을 만들고  를 붙이거나
      - n-2번째 타일을 만들고  를 붙이면 됨
    - 따라서 $\mathrm{Tiling(n)} = \begin{cases} 1(n=1) \\ 2(n=2) \\ Tiling(n-1) + Tiling(n-2)(n \geq 3) \end{cases}$

Dynamic Programming

- Example 3
  - 동전 2
    - BOJ 2294

  - Solution
    - 15원 최소 개수로 만들기 위해
      - 1원 + 14원을 만드는 최소 개수
      - 5원 + 10원을 만드는 최소 개수
      - 12원 + 3원을 만드는 최소 개수

    - $getMinCoin(n) = \min([getMinCoin(n - coin)\ for\ coin\ in\ coin\_list]) + 1$
      - $coin\_list$: 동전 가치들의 list(array)
      - $coin$: 한 동전의 가치(원)

문제

n가지 종류의 동전이 있다. 이 동전들을 적당히 사용해서, 그 가치의 합이 k원이 되도록 하고 싶다. 그러면서 동전의 개수가 최소가 되도록 하려고 한다. 각각의 동전은 몇 개라도 사용할 수 있다.

사용한 동전의 구성이 같은데, 순서만 다른 것은 같은 경우이다.

입력

첫째 줄에 n, k가 주어진다. (1 ≤ n ≤ 100, 1 ≤ k ≤ 10,000) 다음 n개의 줄에는 각각의 동전의 가치가 주어진다. 동전의 가치는 100,000보다 작거나 같은 자연수이다. 가치가 같은 동전이 여러 번 주어질 수도 있다.

출력

첫째 줄에 사용한 동전의 최소 개수를 출력한다. 불가능한 경우에는 -1을 출력한다.

Contents

- 지난 과제 풀이
- Dynamic Programming
  - 재귀의 문제
  - Theory
  - Top-Down vs Bottom-Up
  - Example
  - Tip
- 과제

Dynamic Programming

- Tip (1/2)
  - DP 문제를 접근할 때 많이 하는 실수 or 어렵다고 생각하는 이유
    - 재귀적인 구조를 안찾고, 그냥 규칙을 찾으려고 함
      - 1 ~ 10까지 구해보고, 어! 이거 그냥 func(n) = func(n – 1) + func(n – 2)이네!
      - 물론 DP에 훈련이 된다면 나중에는 이렇게 찾게되지만, 훈련이 되지 않으면 좋은 풀이가 아님

  - 최종적으로 구하고자 하는 목적을 정의하고 함수로 만들어 재귀적인 구조를 찾기
    - e.g., fib(n), tiling(n), stair(n)

Dynamic Programming

- Tip (2/2)
  - DP가 나온 이유가 재귀적인 호출의 반복을 줄이기 위함
    - 따라서 DP 문제면 무조건 재귀적인 구조이고,
      그 재귀적인 구조가 반복 호출을 부름 (재귀적 구조를 찾고 DP를 적용하자!)
    - 이런 재귀 함수의 특징
      - fac(n) = n * fac(n – 1) 처럼 재귀 호출을 1개를 부르지 않음, 2개, 3개를 부름
      - 그렇기 때문에 중복이 생기고, 비효율적이 되는 것!

Contents

- 지난 과제 풀이
- Dynamic Programming
  - 재귀의 문제
  - Theory
  - Top-Down vs Bottom-Up
  - Example
  - Tip
- 과제

과제

- 3주차 과제
  - [링크]
- 과제가 좀 많음
  - 8문제 ㄸ
  - 근데 사실상 3문제는 다 알려줬기 때문에 5문제임 ㅎㅎ
- 마지막 문제 (계단 오르기)는 어려울 수도…
  - DP 테이블을 꼭 1차원으로 만들지 않아도 된다!
  - 상태에 따라 2차원으로 만들 수도 있고, 3차원으로 만들 수도 있다!
- 화이팅!

감사합니다!