1 状態を $O(\sqrt{n})$ 個にまとめる DP

愚直には以下のような遷移をする DP を考える．

Algorithm 1: 愚直な DP

for $i \in \{1, \ldots, K\}$ **do**
  for $j \in \{1, \ldots, N\}$ **do**
    $dp[i][j] \leftarrow \sum_{k=1}^{\lfloor N/j \rfloor} dp[i-1][k]$

$\sum$ の部分は累積和を用意することで $O(1)$ で計算可能としても，全体では $O(KN)$ 時間と $O(N)$ 要素ぶんの空間が必要となる[*1]．

ここで，$\sum$ の上限である $\lfloor N/j \rfloor$ が等しい j たちについては $dp[i][j]$ の値は同じになることがわかる．$j < \sqrt{N}$ である j は $O(\sqrt{N})$ 個しかなく，$j \geqslant \sqrt{N}$ である j に対して $\lfloor N/j \rfloor$ は $O(\sqrt{N})$ 個しかないことから，状態の個数を $O(\sqrt{N})$ 個にまとめられることがわかる．また，以下のようにしてその状態たちを求めることができる．この処理のあと

Algorithm 2: まとめる状態を求める

$S \leftarrow \{0\}$
$i \leftarrow 1$
while $i \times i \leqslant N$ **do**
  $S \leftarrow S \cup \{i, \lfloor N/i \rfloor\}$
  $i \leftarrow i + 1$

$$S = \{a_0, a_1, \ldots, a_{M-1}\},\ \text{where } a_0 = 0 < a_1 < \cdots < a_{M-1}$$

とすると，各 j について $k \in (a_{j-1}, a_j]$ は同じ状態としてまとめられる．まとめられた状態から別の状態に値を渡すとき，まとめた個数を掛ける必要があるので，次のようになる．以下で，$dp[i][j]$ は各 i に関して $(a_{j-1}, a_j]$ でまとめた状態の値を管理する．時間計算量は $O(KM) = O(K\sqrt{N})$．

Algorithm 3: 状態をまとめたあとの DP

for $i \in \{1, \ldots, K\}$ **do**
  for $j \in \{1, \ldots, M-1\}$ **do**
    $dp[i][j] \leftarrow \left(\sum_{k=1}^{M-j} dp[i-1][k] \right) \times (a_j - a_{j-1})$

[*1] 前のテーブルと今のテーブルだけ確保するやつを必要に応じて使う．

```
int main() {
  size_t N, K;
  scanf("%zu %zu", &N, &K);

  std::vector<size_t> a{0};
  for (size_t i = 1; i * i <= N; ++i) {
    a.push_back(i);
    a.push_back(N/i);
  }
  std::sort(a.begin(), a.end());
  a.erase(std::unique(a.begin(), a.end()), a.end());

  size_t M = a.size();
  std::vector<std::vector<intmax_t>> dp(K, std::vector<intmax_t>(M));
  for (size_t j = 1; j < M; ++j) {
    dp[0][j] = a[j] - a[j-1];
    (dp[0][j] += dp[0][j-1]) %= mod;
  }

  for (size_t i = 1; i < K; ++i) {
    for (size_t j = 1; j < M; ++j) {
      dp[i][j] = dp[i-1][M-j] * (a[j]-a[j-1]) % mod;
      (dp[i][j] += dp[i][j-1]) %= mod;
    }
  }

  printf("%jd\n", dp[K-1][M-1]);
}
```