Dynamic Programming

CS202 Lecture Notes

School of Computing
KAIST

What we learn?

- 동적 계획법을 배웁니다.

What we learn?

- 동적 계획법을 배웁니다.
- ... 는 CS202에서는 무리인 것 같고...

What we learn?

- 동적 계획법을 배웁니다.
- ... 는 CS202에서는 무리인 것 같고...
- 동적 계획법의 원리를 보여줄 수 있는 예시들을 소개합니다.
- 동적 계획법은 CS300에서 배울 수 있을 거에요.

What we learn?

- 이번 강의에서 다룰 주제들의 공통점은, **반복되는 계산을 적절한 점화식을 세워서 줄여나가는 방법이라는 것입니다.**
- 점화식을 세우고 이해하는 것이 아주 중요합니다.
- 여러 동적 계획법 문제 중에서 중요하다고 생각되는 예시와 실습 문제를 골라 강의합니다.

Contents

- Prefix Sum Array
- Subset Sum Problem
- Knapsack Problem

Prefix Sum Array

- 어떠한 배열 A의 $[s, e]$ 구간 합 $(\sum_{i=s}^{e} A_i)$ 을 구하는 문제를 생각해 봅시다.
- 예를 들어, 평균을 구하거나..
- 단순한 방법은, 구간이 여러 개 주어지면 느립니다.

Prefix Sum Array

- 어떠한 배열 A의 $[s, e]$ 구간 합 $(\sum_{i=s}^{e} A_i)$ 을 구하는 문제를 생각해 봅시다.
- 예를 들어, 평균을 구하거나..
- 단순한 방법은, 구간이 여러 개 주어지면 느립니다.
- 고등학교 수학 시간에 했던 것 처럼, $S_i = \sum_{j=1}^{i} A_j$ 를 정의해 봅시다.

Prefix Sum Array

- 고등학교 수학 시간에 했던 것 처럼, $S_i = \sum_{j=1}^{i} A_j$ 를 정의해 봅시다.
- 어떠한 배열의 구간 합은 $S_e - S_{s-1}$ **(포함 배제!)**

Prefix Sum Array

- 고등학교 수학 시간에 했던 것 처럼, $S_i = \sum_{j=1}^{i} A_j$ 를 정의해 봅시다.
- 어떠한 배열의 구간 합은 $S_e - S_{s-1}$ **(포함 배제!)**
- S_i는 어떻게 구할까요?
- $S_i = S_{i-1} + A_i$ 라는 점화식을 사용!
- $O(N+Q)$

Prefix Sum Array

- 2차원에서는 어떻게 될까요?
- $S_{i,j} = \sum_{k=1}^{i} \sum_{l=1}^{j} A_{k,l}$

Prefix Sum Array

- 2차원에서는 어떻게 될까요?
- $S_{i,j} = \sum_{k=1}^{i} \sum_{l=1}^{j} A_{k,l}$
- 그렇다면..
- 1. 배열 $S_{i,j}$ 는 어떻게 계산할 것이며
- 2. $S_{i,j}$ 를 알면 2차원 구간 합을 어떻게 계산할 수 있을까요?
- 두 문제의 힌트는 포함 배제입니다. :)

Subset Sum

- 최대 B(g) 를 담을 수 있는 가방이 있습니다.
- n개의 물건이 있고, 각각의 무게는 $w_1, w_2, \cdots, w_n$(g) 입니다.
- 가방에 물건을 담을 수 있는 경우의 수는 몇 개일까요?

Subset Sum

- 최대 B(g) 를 담을 수 있는 가방이 있습니다.
- n개의 물건이 있고, 각각의 무게는 $w_1, w_2, \cdots, w_n$(g) 입니다.
- 가방에 물건을 담을 수 있는 경우의 수는 몇 개일까요?
- 단순하게, 모든 2^n 개의 경우를 시도해 볼 수 있습니다.

Subset Sum

- 하지만 n이 크면 너무 느려요!

Subset Sum

- 하지만 n이 크면 너무 느려요!
- 점화식을 세워 봅시다.

Subset Sum

- 하지만 n이 크면 너무 느려요!
- 점화식을 세워 봅시다.
- $SUM_{i,j} = (w_1, w_2, \cdots, w_i$를 사용해서 가방에 j를 담는 경우의 수)

Subset Sum

- $SUM_{i,j} = (w_1, w_2, \cdots, w_i$ 를 사용해서 가방에 j를 담는 경우의 수)
- 정의는 했는데, 어떻게 구할까요?

Subset Sum

- $SUM_{i,j} = (w_1, w_2, \cdots, w_i$ 를 사용해서 가방에 j를 담는 경우의 수)
- 정의는 했는데, 어떻게 구할까요?
- 1. w_i 를 가방에 넣지 않는다: $w_1, w_2, \cdots, w_{i-1}$ 을 사용해서 j를 담아야 합니다.
- 2. w_i 를 가방에 넣는다: $w_1, w_2, \cdots, w_{i-1}$ 을 사용해서 $(j - w_i)$를 담아야 합니다.

Subset Sum

- $SUM_{i,j} = (w_1, w_2, \cdots, w_i$ 를 사용해서 가방에 jg을 담는 경우의 수)
- 정의는 했는데, 어떻게 구할까요?
- 1. w_i 를 가방에 넣지 않는다: $SUM_{i-1,j}$
- 2. w_i 를 가방에 넣는다: $SUM_{i-1,j-w_i}$

Subset Sum

- $SUM_{i,j} = (w_1, w_2, \cdots, w_i$ 를 사용해서 가방에 jg을 담는 경우의 수)
- 정의는 했는데, 어떻게 구할까요?
- 1. w_i 를 가방에 넣지 않는다: $SUM_{i-1,j}$
- 2. w_i 를 가방에 넣는다: $SUM_{i-1,j-w_i}$
- $SUM_{i,j} = SUM_{i-1,j} + SUM_{i-1,j-w_i}$

Subset Sum

- $SUM_{i,j} = (w_1, w_2, \cdots, w_i$ 를 사용해서 가방에 jg을 담는 경우의 수)
- 정의는 했는데, 어떻게 구할까요?
- 1. w_i 를 가방에 넣지 않는다: $SUM_{i-1,j}$
- 2. w_i 를 가방에 넣는다: $SUM_{i-1,j-w_i}$
- $SUM_{i,j} = SUM_{i-1,j} + SUM_{i-1,j-w_i}$
- 이항 계수를 구하듯이, $SUM_{i-1,*}$ 배열을 채우고, 그 결과를 토대로 $SUM_{i,*}$ 를 구하면 됩니다.

Knapsack

- 최대 B(g) 를 담을 수 있는 가방이 있습니다.
- n개의 물건이 있고, 각각의 무게는 $w_1, w_2, \cdots, w_n$(g) 입니다.

Knapsack

- 최대 $B(\mathrm{g})$ 를 담을 수 있는 가방이 있습니다.
- n개의 물건이 있고, 각각의 무게는 $w_1, w_2, \cdots, w_n(\mathrm{g})$ 입니다.
- 각각의 물건의 가치는 $v_1, v_2, \cdots, v_n$ 원 입니다.
- 당신은 뛰어난 도둑! 물건 가치 합을 최대화해서 가방에 넣고 훔쳐야 합니다.

Knapsack

- 최대 $B(\mathrm{g})$ 를 담을 수 있는 가방이 있습니다.
- n개의 물건이 있고, 각각의 무게는 $w_1, w_2, \cdots, w_n(\mathrm{g})$ 입니다.
- 각각의 물건의 가치는 $v_1, v_2, \cdots, v_n$ 원 입니다.
- 당신은 뛰어난 도둑! 물건 가치 합을 최대화해서 가방에 넣고 훔쳐야 합니다.
- 이것도, 모든 2^n 개의 경우를 시도해 볼 수 있습니다.

Knapsack

- **경우의 수를 셀 수 있으면, 최대화도 할 수 있습니다.**
- 아까와 똑같이 해 봅시다.
- $SUM_{i,j} = (w_1, w_2, \cdots, w_i$ 를 사용해서 가방에 j를 담는 경우의 수)
- $PROFIT_{i,j} = (w_1, w_2, \cdots, w_i$ 를 사용해서 가방에 j를 담았을 때 가능한 최대 이득)

Knapsack

- 1. w_i 를 가방에 넣지 않는다: $PROFIT_{i-1,j}$
- 2. w_i 를 가방에 넣는다: $PROFIT_{i-1,j-w_i} + v_i$

Knapsack

- 1. w_i 를 가방에 넣지 않는다: $PROFIT_{i-1,j}$
- 2. w_i 를 가방에 넣는다: $PROFIT_{i-1,j-w_i} + v_i$
- *SUM* 에서는 합을 취했으니...
- *PROFIT* 에서는 최댓값을 취합니다.

Knapsack

- 1. w_i 를 가방에 넣지 않는다: $PROFIT_{i-1,j}$
- 2. w_i 를 가방에 넣는다: $PROFIT_{i-1,j-w_i} + v_i$
- *SUM* 에서는 합을 취했으니...
- *PROFIT* 에서는 최댓값을 취합니다.
- $PROFIT_{i,j} = max(PROFIT_{i-1,j}, PROFIT_{i-1,j-w_i} + v_i)$