강의 21: 동적 프로그래밍 III

개요

- 문자열에서의 하위 문제들
- 괄호 묶기
- 편집 거리 문제 (& 최장 거리 공통 부분 시퀀스)
- 가방 싸기 문제
- 유사 다항 시간

Review:

* 동적 프로그램의 5단계

(a) 하위 문제 정의하기 — 하위 문제의 개수 세기
(b) (해법의 일부분을) 추측하기 — 선택의 개수 세기
(c) 하위 문제의 풀이 연관짓기 — 하위 문제당 시간 계산
(d) 재귀+기억 — 총 시간 = 하위 문제당 시간 · 하위 문제의 개수
또는 DP table을 밑에서부터 위로 만들기
하위 문제가 비순환적인지/위상학적 순서 확인하기
(e) 원래 문제의 풀이: = 하위 문제 의 풀이
또는 하위 문제들로 조합하여 풀기 — ⇒ 추가적인 시간 필요

* L20 의 문제(텍스트 정렬, 블랙잭) 는 시퀀스와 관련있다 (단어, 카드의 시퀀스)

* 문자열/시퀀스 x에 대해 유용한 문제들:

$$\left.\begin{array}{l}\text{suffixes } x[i:] \\ \text{prefixes } x[:i]\end{array}\right\}\Theta(|x|) \quad \leftarrow \text{cheaper} \implies \text{use if possible}$$

$$\left.\text{substrings } x[i:j]\right\}\Theta(x^2)$$

괄호 묶기:

A[0] · A[1]···A[n − 1] 를 계산하는 최적의 방법— e.g., 행렬의 곱셈

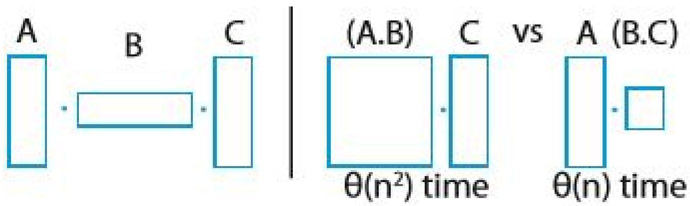


Figure 1:

1. <u>추측</u> = 가장 바깥의 곱셈

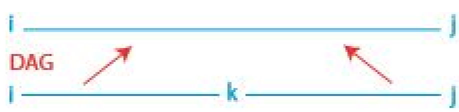
   

   ⇒ 선택의 수 = O(n)

2. <u>하위 문제</u> = <u>prefix & suffix?</u> NO
   = 부분 문자열 A[i:j] 의 비용
   ⇒ 하위 문제의 개수= $\Theta(n^2)$

3. <u>반복</u>:

   - DP[i,j] = min(DP[i,k] + DP[k,j]+ ($A[i]\cdots A[k-1]$) 과 ($A[k]\cdots A[j-1]$) 를 곱하는 데 필요한 비용 (for k in range(i + 1,j))

   - DP[i,i + 1] = 0
     ⇒ 하위 문제당 비용 = O(j −i) = O(n)

4. <u>위상학적 순서</u>: 부분 문자열 크기의 오름차순. 총 시간 = O(n^3)

5. <u>원래 문제</u> = DP[0,n]

   (& 부모 포인터를 사용하여 부모를 복원)

NOTE: 이 DP는 DAG의 최단 경로가 아님! 두 개의 노드에 의존 ==⇒ 경로가 아님!

편집 거리

DNA 비교, 차이, CVS/SVN/..., 철자 확인 (오타), 표절 탐지 등.

주어진 두 개의 문자열 x & y에 대해, x를 y로 바꿀 때 가장 값싼 편집(c 삽입, c 삭제, c → c'로 교체)의 시퀀스는 무엇인가?

- 편집의 비용은 오직 문자 c,c' 에 의존한다
- 예를 들어 DNA에서, C → G 는 흔한 변형이므로 ⇒ 비용이 싸다
- 시퀀스의 비용 = 편집 비용의 합
- 만약 삽입/삭제의 비용이 1, 교체의 비용이 0(c=c') 또는 무한대(c!=c')이라면 이 문제는 최장 길이 공통 부분 시퀀스를 찾는 문제이다
  Note : 부분 시퀀스는 문자열 안에서 연속된 것일 필요는 없다

- for example H I E R O G L Y P H O L O G Y vs. M I C H A E L A N G E L O
  ⇒ HELLO

여러 개의 문자열/시퀀스에 대한 문제

- suffix/prefix/부분 문자열 하위 문제를 통합
- 상태의 크기들을 곱함
- O(1) 문자열에 대해 여전히 다항식 형태

편집 거리 DP

(1) 하위 문제: $c(i,j)$ = 편집 거리$(x[i:],y[j:])$ for $0 \le i < |x|, 0 \le j < |y|$
⇒ $\Theta(|x| \cdot |y|)$ 만큼의 하위 문제가 존재
(2) 추측 x를 y로 바꾸기 위해, (3 choices):

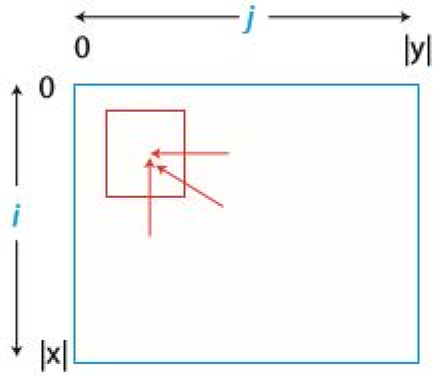
- $x[i]$ 를 삭제하거나
- $y[j]$ 를 삽입하거나
- $x[i]$ 를 $y[j]$로 교체해야 함

(3) 반복: $c(i,j)$ = 아래 중 최대값:

- $x[i]$를 삭제하는 비용 + $c(i+1,j)$ if $i < |x|$,
- $y[j]$를 삽입하는 비용 + $c(i,j+1)$ if $j < |y|$,
- $x[i]$ 를 $y[j]$로 교체하는 비용 + $c(i+1,j+1)$ if $i < |x| \& j < |y|$

기본 단계: $c(|x|,|y|) = 0$
⇒ 하위 문제당 $\Theta(1)$만큼의 시간이 필요

(4) 위상학적 순서: DAG 를 2차원 표로 나타낼 수 있음:



- 밑에서 위로 혹은 오른쪽에서 왼쪽으로
- 마지막 2개의 행/열만 유지하고 있으면 됨

  ⇒ 선형 크기 공간
- 총 시간 = $\Theta(|x| \cdot |y|)$

(5) 원래 문제: c(0,0)

배낭 문제:

크기 S의 가방에 짐을 넣고 싶을 때

- 물건 i 는 정수 형태의크기 si 와 실제 가치 vi를 가지고 있음
- 목표: 가치를 최대로 하면서 총 크기 ≤ S 를 만족하도록, 가져갈 수 있는 물건의 목록 만들기

첫 번째 시도:

1. 하위 문제 = suffix i의 가치: 틀림
2. 추측 = 물건 i를 가져갈지 말지 선택 ⇒ 2가지 선택사항이 있음
3. 반복:
   - $DP[i] = \max(DP[i+1], v_i + DP[i+1]$ if $s_i \leq S$?!)
   - 물건 i를 넣을 수 있는 크기인지 알 수 없음 — 얼마나 공간이 남았는가? 추측이 필요함!

옳은 풀이:

1. 하위 문제 = suffix i:의 가치

   주어진 크기 X의 가방

⇒ 하위 문제의 개수= O(nS) !

3. 반복:
   - $DP[i,X] = \max(DP[i+1,X], v_i + DP[i+1, X - s_i] \text{if } s_i \le X)$
   - $DP[n,X] = 0$

     ⇒하위 문제당 필요한 시간 = O(1)
4. 위상학적 순서: for i in n,...,0: for X in 0,...S

   총 시간 = $O(nS)$
5. 원래 문제= $DP[0,S]$
   (& 부분 집합을 복원하기 위해 부모 포인터 사용)

놀라운 사실: 가져갈 물건의 집합의 모든 경우를 효율적으로 고려함! ...하지만 실제로 빠른가?

다항 시간

다항 시간 = 입력값 크기에 대해 다항식으로 표현가능

- 만약 S가 한 워드에 들어간다면 $\Theta(n)$이라 할 수 있음
- 일반적으로 $O(n \lg S)$.
- S 는 $\lg S$에 대해 지수적임 (다항식이라 할 수 없음)

유사 다항 시간

유사 다항 시간 = 입력값의 크기에 대해 다항식과, 입력으로 받는 숫자들로 표현가능하다 (예시: S, s_i's, v_i's). $\Theta(nS)$ 는 유사 다항 시간이다.

기억할 것:

다항 — 좋음

지수— 나쁨

유사 다항— 보통