강의 19: 동적 프로그래밍 I: 메모이제이션, 피보나치 수, 최단 경로, 추측

강의 개요

- 메모이제이션과 하위 문제
- 예제
    - 피보나치 수
    - 최단 경로
- 추측 & 방향성 비순환 그래프 관점

동적 프로그래밍 (DP)

어렵지만 간단한 큰 개념

- 강력한 알고리즘 디자인 기법
- 보기에는 지수 시간이 걸리는 많은 문제도 DP를 사용하면 (사용해야지만) 다항 시간에 풀 수 있음
- 특히 최적화 문제 (최소화/최대화)에 쓰임 (최단 경로 문제)

* DP ≈ "세심한 무차별 대입법"
* DP ≈ 재귀+ 재사용

역사

리처드 E. 벨만(1920-1984)
리처드 벨만은 1979년에 IEEE 명예 메달을 받았습니다. "RAND에서 연구라는 단어에 병적인 두려움과 증오를 느끼는 국방 장관 밑에서 일 할 때 벨만은 수학 연구를 하고 있다는 사실을 숨기기 위하여 '동적 프로그래밍'이라는 단어를 만들어냈습니다. 국회의원도 반대하기 어렵고 난해하게 들리지 않는 '동적 프로그래밍'이라는 단어를 골랐습니다 [존 러스트 2006]

피보나치 수

$$F_1 = F_2 = 1; \quad F_n = F_{n-1} + F_{n-2}$$

목표: F_n 계산

단순 알고리즘

재귀 정의 따르기

$\underline{\text{fib}}(n)$:

if $n \leq 2$: return $f = 1$

else: return $f = \text{fib}(n-1) + \text{fib}(n-2)$

$\implies T(n) = T(n-1) + T(n-2) + O(1) \geq F_n \approx \varphi^n$

$\geq 2T(n-2) + O(1) \geq 2^{n/2}$

EXPONENTIAL — BAD!

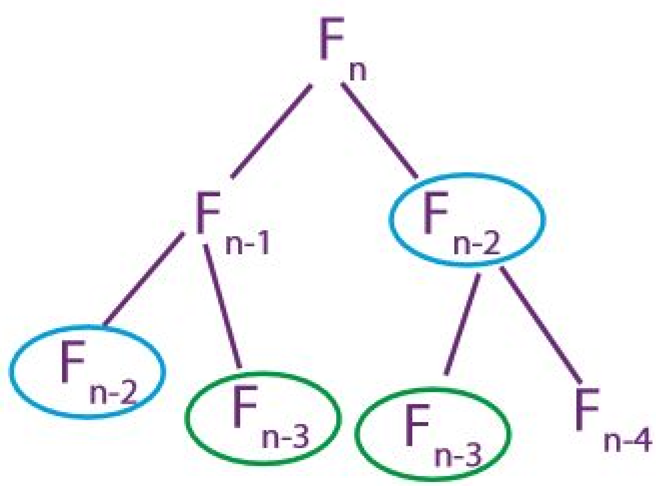


그림 1: 단순 피보나치 알고리즘

메모이제이션 DP 알고리즘

기억하기

memo = { }

fib(n):

if n in memo: return memo[n]

else: if $n \leq 2$: $f = 1$

else: $f =$ fib$(n-1)$ + fib$(n-2)$

memo[n] = f

return f

- ⇒ 모든 k에 대해 fib(k)는 처음 호출될 때만 재귀 호출을 한다
- ⇒ n개의 메모이제이션 되지 않은 호출만이 존재한다: k = n, n − 1,..., 1
- 메모이제이션 된 호출은 거의 공짜이다 (Θ(1) time)
- ⇒ 재귀를 무시한다면 호출당 Θ(1) 시간이 걸린다

다항 시간 소요 — 좋은 알고리즘

* DP ≈ 재귀 + 메모이제이션

- <u>메모이제이션</u> (기억하기) & 문제를 푸는데 도움이 되는 하위 문제 해의 재사용
  - 피보나치 수의 하위 문제는 $F_1, F_2, ..., F_n$ 이다

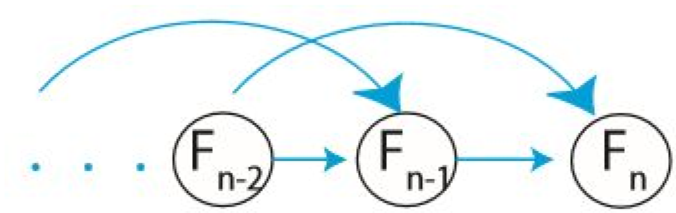
* ⇒ 소요 시간 = 하위 문제 개수 · 하위 문제당 소요 시간

- 피보나치 수: 하위 문제 개수는 n이고, 하위 문제당 소요 시간은 Θ(1) = Θ(n)이다 (재귀를 무시한다).

상향식 DP 알고리즘

```
fib = {}                                            Θ(n)
for k in [1, 2, ..., n]:
    if k ≤ 2: f = 1                            Θ(1)
    else: f = fib[k − 1] + fib[k − 2]
    fib[k] = f
return fib[n]
```

- **메모이제이션 DP와 계산이 정확히 일치한다 (재귀를 "펼친다")**
- **일반적으로 하위 문제 종속 방향성 비순환 그래프의 위상 정렬**



- **재귀가 없기 때문에 사실상 더 빠르다**
- **분석이 더 쉽다**
- **공간을 아낄수 있다: 최근 두 피보나치 수만 기억한다 ⟹ Θ(1)**

[참고: 피보나치 수 계산은 다른 방법을 사용한 O(lg n)-시간 알고리즘이 존재한다]

최단 경로

- **재귀 공식:**
  $\delta(s, v) = \min\{w(u, v) + \delta(s, u)\ (u, v) \in E\}$

- **메모이제이션 DP 알고리즘: 순환이 있다면 무한 시간이 걸림!**
  어떤 경우에는 음수 순환 처리 필요

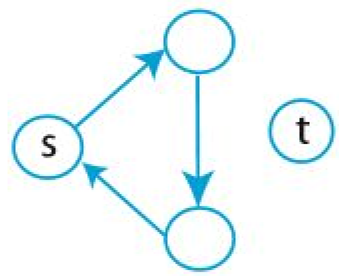


Figure 2: Shortest Paths

- 방향성 비순환 그래프에서는 $O(V + E)$시간에 실행
  실질적으로 깊이 우선 탐색/위상 정렬 + 벨만-포드 한 단계가 하나의 재귀로 압축

* 하위 문제 종속은 비순환이어야 함

- 하위 문제가 늘어난다면 순환 종속을 없앨 수 있다:
  $\delta_k(s, v)$ = k개 이하 간선을 사용한 s에서 v까지의 최단 경로

- 재귀:

$$\begin{aligned}
\delta_k(s,v) &= \min\{\delta_{k-1}(s,u) + w(u,v) \,|\, (u,v) \in E\} \\
\delta_0(s,v) &= \infty \text{ for } s \neq v \text{ (base case)} \\
\delta_k(s,s) &= 0 \text{ for any } k \text{ (base case, if no negative cycles)}
\end{aligned}$$

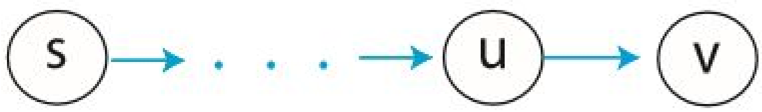
- 목표: $\delta(s, v) = \delta_{|V|-1}(s, v)$ (음수 순환이 없는 경우)
- 메모이제이션
- time: $\underbrace{\#\ \text{subproblems}}_{|V|\cdot|V|} \cdot \underbrace{\text{time/subproblem}}_{O(v)} = 0(V^3)$
- 실제로는 $\delta_k(s, v)$에 대해 Θ(진입차수(v))
- $\implies \text{time} = \Theta(V \sum_{v \in V} \text{indegree}(V)) = \Theta(VE)$

벨만-포드 알고리즘!

추측

재귀 디자인 방법

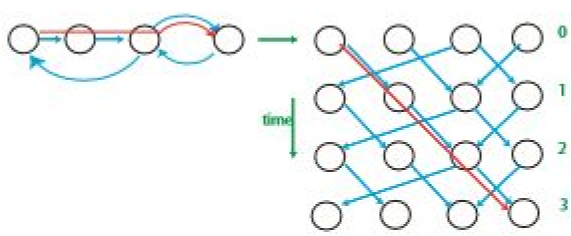
- s에서 v까지의 최단 거리 계산



- 경로에서 마지막 간선? 알 수 없음
- (u, v)라고 추측
- path is $\underbrace{\text{shortest } s \to u \text{ path}}_{\text{by optimal substructure}} + \text{edge } (u, v)$
- cost is $\underbrace{\delta_{k-1}(s, u)}_{\text{another subproblem}} + w(u, v)$
- 최선의 추측을 찾기 위해 모든 (|V|) 값들을 시도해보고 최적을 찾는다
- * 비결: 하위 문제당 적은 (다항) 가능한 추측 — 하위 문제당 시간을 결정하는 요인이다

* DP ≈ 재귀 + 메모이제이션 + 추측

DAG view



- 시간을 나타내도록 그래프를 복제하는것과 비슷하다
- 그래프의 최단 경로를 방향성 비순환 그래프의 최단 경로로 변환

* DP ≈ 몇 방향성 비순환 그래프의 최단 경로

6.006 Introduction to Algorithms
Fall 2011