강의 20: 동적 프로그래밍 II

강의 개요

- 간단한 5개 단계
- 글 정렬
- 정보를 모두 알고 있는 블랙잭
- 부모 포인터

개요

* DP ≈ "세심한 무차별 대입법"

* DP ≈ 추측 + 재귀 + 메모이제이션

* DP ≈ 문제를 합리적인 개수의 하위 문제로 나누고 해를 비순환적으로 연관 짓는다. 보통 해의 일부를 추측한다.

* $\text{time} = \#\text{ subproblems} \times \underbrace{\text{time/subproblem}}_{\substack{\text{treating recursive calls as } O(1) \\ \text{(usually mainly guessing)}}}$

  - 근본적으로 분할 상환
  - 각 하위 문제를 한번만 센다; 메모이제이션을 통해 처음 이후에는 O(1) 시간이 걸린다

* DP ≈ DAG에서의 최단 경로

동적 프로그래밍의 간단한 5 단계

1. 하위 문제를 정의한다 — 하위 문제의 개수를 센다
2. 해의 일부를 추측한다 — 선택지의 개수를 센다
3. 하위 문제의 해를 연관짓는다 — 하위 문제당 시간을 계산한다
4. 재귀와 메모이제이션 — 시간 = 하위 문제당 시간 · 하위 문제의 개수
   혹은 DP 테이블을 상향식으로 만든다
   하위 문제가 비순환이고 위상 순서가 있다는걸 확인한다
5. 기존 문제를 푼다 - 하위 문제와 같거나
   하위 문제의 해를 합쳐서 푼다 ⇒ 추가 시간이 걸림

Examples:	Fibonacci	Shortest Paths
subprobs:	F_k for $1 \le k \le n$	$\delta_k(s,v)$ for $v \in V$, $0 \le k < \lvert V \rvert$ $= \min s \to v$ path using $\le k$ edges
# subprobs:	n	V^2
guess:	nothing	edge into v (if any)
# choices:	1	$\text{indegree}(v) + 1$
recurrence:	$F_k = F_{k-1} + F_{k-2}$	$\delta_k(s,v) = \min\{\delta_{k-1}(s,u) + w(u,v) \mid (u,v) \in E\}$
time/subpr:	$\Theta(1)$	$\Theta(1 + \text{indegree}(v))$
topo. order:	for $k = 1, \ldots, n$	for $k = 0, 1, \ldots \lvert V \rvert - 1$ for $v \in V$
total time:	$\Theta(n)$	$\Theta(VE)$ + $\Theta(V^2)$ unless efficient about indeg. 0
orig. prob.:	F_n	$\delta_{\lvert V \rvert - 1}(s,v)$ for $v \in V$
extra time:	$\Theta(1)$	$\Theta(V)$

글 정렬

글을 "좋은" 줄로 나눈다

- MS Word/Open Office에서 쓰이는 알고리즘: 첫 줄에 가능한 많은 단어를 넣고 반복한다
- 좋지 않은 줄을 만들 수 있음

그림 1: 좋은 정렬 vs 나쁜 정렬

• 단어$[i : j]$에 대해 나쁨을 정의한다
총 길이가 페이지 너비보다 크면 ∞, 아니면 (페이지 너비 - 총 길이) ^{3}

• 목표: 단어를 나눠서 나쁨의 합을 최소화한다

1. 하위 문제 = suffix 단어$[i :]$의 최소 나쁨

⇒ 하위 문제 개수 = $\Theta(n)$, n = 단어 개수

2. 추측 = 첫 줄을 어디서 끊을지 $(i : j)$
⇒ 선택지 개수 = $n-i$ = O(n)

3. 재귀:
   - DP[i] = i + 1부터 n + 1까지의 j에 대해 나쁨 (i,j) + DP[j]의 최솟값
   - DP[n] = 0

   ⇒ 하위 문제당 시간 = Θ(n)

4. 순서: $i = n,n-1,...,1,0$ 에 대해

   총 시간 = Θ(n^2)

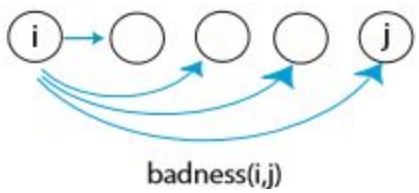

그림 2: DAG.

5. 답 = DP[0]

정보를 모두 알고 있는 블랙잭

- $c_0, c_{1,\cdots} c_{n-1}$ 가 모든 카드의 순서일때
- 17이상에서 카드를 더 받지 않는 딜러와 한 명의 참가자
- 받을지 말지 추측
- 목표: $1 내기에서 최대한 많이 따기
- 한 번 져서 나중에 더 좋은 패를 얻을 수 있음

1. subproblems: BJ(i) = best play of $\underbrace{c_i, \ldots c_{n-1}}_{\text{remaining cards}}$ where i is # cards "already played"

   ⇒ 하위 문제 개수 = n

2. 추측: 참가자가 몇 장의 카드를 더 받는지
   ⇒ 선택지 개수 ≤ n

3. 재귀: BJ(i) = 최댓값(
   결과 ∈{+1,0,−1} + BJ(i + 사용된 카드) O(n)

21을 넘지 않을 동안 0,1,... 인 받은 카드 개수 $O(n)$

)

⇒ 하위 문제당 시간 = $\Theta(n^2)$

4. 순서: n부터 0까지 i에 대해
   총 시간 = $\Theta(n^3)$

time is really $\sum_{i=0}^{n-1} \sum_{\#h=0}^{n-i-O(1)} \Theta(n-i-\#h) = \Theta(n^3)$ still

5. 답: BJ(0)
   자세한 재귀: 메모이제이션 전(나누기와 두 배 걸기 제외할 때)

$\Theta(n^2)$ { $\Theta(n)$ { BJ(i):
if $n-i<4$: return 0 (not enough cards)
for p in range$(2, n-i-1)$: (# cards taken)
player = sum$(c_i, c_{i+2}, c_{i+4:i+p+2})$
if player >21: (bust)
options.append$(-1(bust) + BJ(i+p+2))$
break
$\Theta(n)$ with care { for d in range$(2, n-i-p)$
dealer = sum$(c_{i+1}, c_{i+3}, c_{i+p+2:i+p+d})$
if dealer ≥ 17: break
if dealer >21: dealer $= 0$ (bust)
options.append(cmp(player, dealer) + BJ$(i+p+d)$)
return max(options)

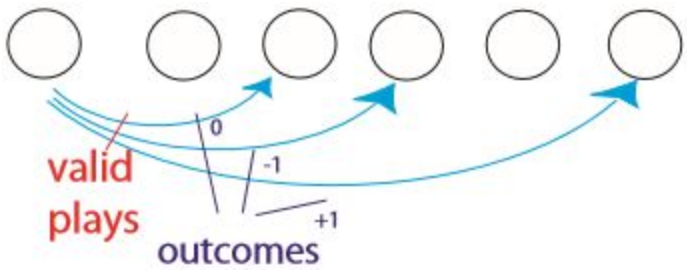

그림 3: DAG view

부모 포인터

총 비용과 실제 해를 찾기 위해 부모 포인터 (각 하위 문제에서 어떤 추측이 쓰였는지)

를 저장하고 거꾸로 푼다

- 보통 최솟/최댓값과 최소/최대 인수를 기억함
- 예제: 글 정렬

(3)' DP[i] = 최솟값(나쁨(i,j) + DP[i][0],j)
for j 가 (i+1,n+1) 사이일때
DP[n] = (0, None)
(5)' i = 0
while i is not None:
i 번째 단어 전에 새 줄을 시작한다
i = DP[i][1]

- 메모이제이션과 상향식 방법과 같이 이 변환은 <u>자동이다</u>
생각할 필요 없음