← Dynamic Programming IV

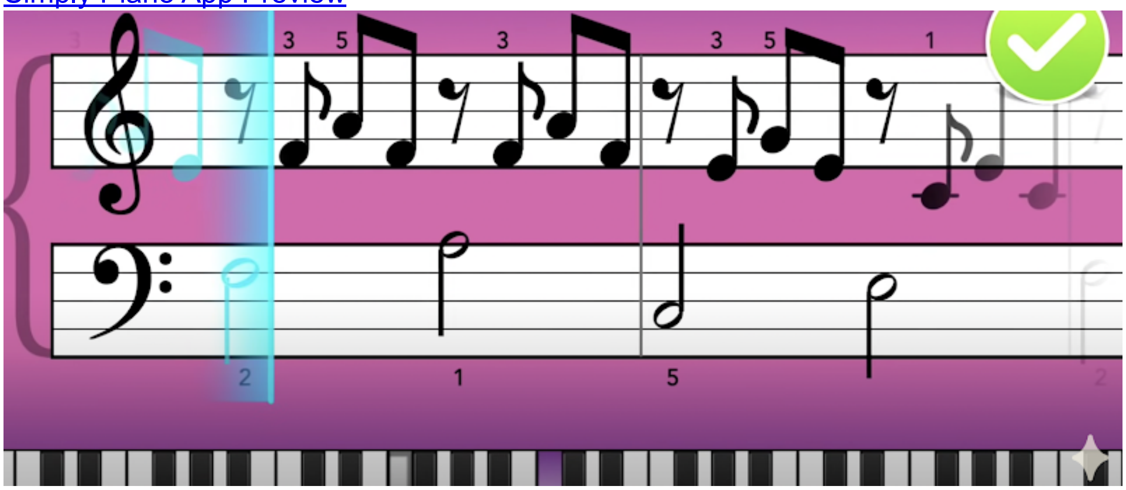
Dynamic Programming IV

Dynamic Programming IV

- Guitar Fingering, Tetris, Super Mario Bros

생각해 봅시다!

[Simply Piano App Preview]



피아노를 칠 때 손가락의 위치,
노트를 보고 어떤 손가락으로 치는지 어떻게 정하는 걸까?
노트들이 얼마나 가까운지, 얼마나 치기 어려운지를 기준으로?
어떤 방법으로 알고리즘을 짤 수 있을까?

5 Easy Steps to DP

1. define subproblems
2. guess (part of the solution)
3. recurrence (relate subproblem solutions)
4. recurse + memoize or Bottom-up
(check acyclic, check topological order)

← Dynamic Programming IV

Piano Fingering

1. Given a musical piece to play, sequence of n (single) notes with the right fingering
2. fingers 1, 2, . . . , F = 5 for humans, 1 손가락으로 한 음만!
3. metric d(f, p, g, q) of difficulty going from note p with finger f to note q with finger g

first note p/ first finger f / next note q / next finger g

=> p음에서 f손가락으로 노트를 치고 그 다음 g음을 q로 손가락으로 칠때 f -> q 로의 이동이 얼마나 어려운지 가중치를 두어서 함수를 만들 수 있다.

stretch rule ex):

- p q ⟹ uncomfortable
- legato (smooth) ⟹ ∞ if f = g
- weak-finger rule: prefer to avoid g ∈ {4, 5}
- 3 → 4 & 4 → 3 annoying ~ , etc.

ex) d(1,c,2,d) => 1, easy, d(1,c,2, a) => 20, difficult

First attempt : 이전까지 풀었던 방식으로,

1. subproblem = min. difficulty for suffix notes[i :]

   i 이후(i + 1)는 이미 최적화가 되었다고 가정하고, difficulty가 가장 작은 i번째 손가락을 찾는 것.
2. guessing = which finger f for first note[i]
3. recurrence: DP[i] = min(DP[i + 1] + d(note[i], f, note[i + 1], ?)

   → not enough information! g값을 몰라 difficulty를 찾을 수 없다. 기존의 방식으로는 풀 수 없는 문제.

5 Easy Steps to DP

1. **define subproblems**
2. **guess (part of the solution)**
3. recurrence (relate subproblem solutions: 식을 도출)
4. recurse + memoize: or Bottom-up: (check acyclic, check topological order)
5. solve original problem

2 kinds of guessing

1. **In steps 2** & 3: Guess **which subproblem** to use in order to solve the bigger subproblem (All DP have used this kind of guessing except for Fibonacci)
   - shortest path: indegree(v)
   - text justification: where to start second line

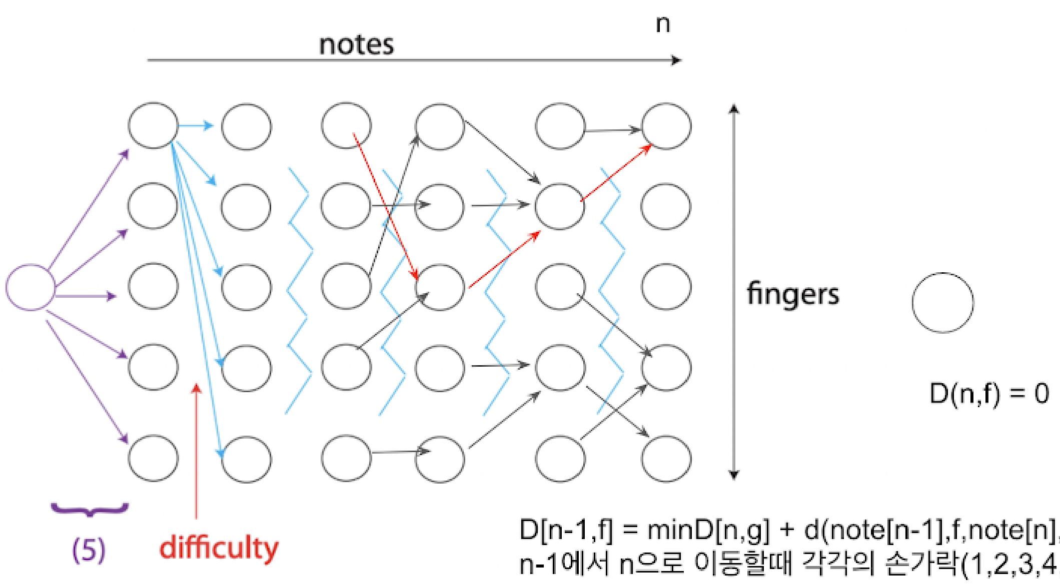
2. **in 1** create more subproblems to guess/remember more solution structure
   - Used by knapsack DP: how much capacity we've used up

Dynamic Programming IV

DP가 두가지 이상의 가정을 가진, 1차원이 아닌 2, 3, … n차원까지 확장과 풀이가 가능!

Correct DP

1. subproblem = min difficulty for suffix notes[i :] **given finger f on first note[i]**
   ==⇒ n · F subproblems
2. guessing = finger g for next note[i + 1]
   ==⇒ F choices
3. recurrence:
   DP[i, f] = min(DP[i + 1, g] + d(note[i], f, note[i + 1], g)
   for g in range(F))
   DP[n, f] = 0 = ⇒ Θ(F) time/subproblem



D[n-1,f] = minD[n,g] + d(note[n-1],f,note[n],g)
n-1에서 n으로 이동할때 각각의 손가락(1,2,3,4,5)에서
다음 손가락으로 가는 최적값을 찾아 둠.

D[n-2,f] = minD[n-1,g] + d(note[n-2],f,note[n-1],g)
n-2 에서 n-1로 이동하는 최적값을 확인 가장 작은 값의
메모

4. topo. order:
   for i in reversed(range(n)):

3. Orig. prob.: min(DP [0, f] for f in 1, . . . , F)
(guessing very first finger)

Guitar

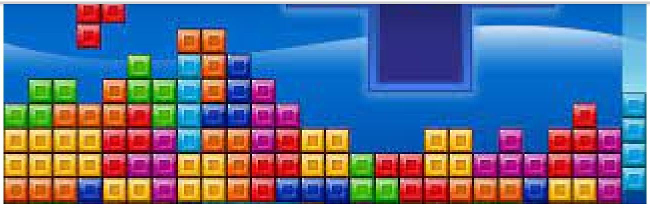
Up to S ways to play a same note!
(where S is # strings)
• redefine “finger” = finger playing note + string playing note
기타는 피아노와 치는 방법이 다르다. 어떤 손가락을 사용해서 어떤 줄을 쓸지도 함께 고려 해야 한다.
=⇒ F → F · S

Multiple notes

Multiple notes at once e.g. chords
• input: notes[i] = list of ≤ F notes (can’t play > 1 note with a finger)
모든 손가락의 위치와 어떤 노트를 치고 있는지 기억해야 한다. 아무것도 치지 않는 경우까지 총 6개의 선택이 가능하다.
• state we need to know about “past” now assignment of F fingers to F ≤ F +1 notes/null
=⇒ (F + 1) such mappings
(1) n · (F + 1)F subproblems where (F + 1)F is how notes[i] is played
(2) (F + 1)F choices (how notes[i + 1] played)
(3) n · (F + 1)2F total time • works for 2 hands F = 10 • just need to define appropriate d

Figure 2: Tetris

Dynamic Programming IV



Tetris Training:

condition

- given sequence of n Tetris pieces
- an empty board of small width w
- must drop from top
- full rows do not clear

Goal: can you survive?

1. subproblem

survive? in suffix [i:]

- How high each column is?
- Remember the height of each column

보드의 현재 상태를 기억해야 한다. 높이 기준!

given initial column occupancies h0, h1, · · ·, hw−1, call it h

=> $n(h+1)^w$ subproblem

2. Guess: how to drop piece i

- Rotate and choose the column to drop

=> $4 * w$ choices

회전 3가지, 아무것도 안하는 것 1가지

$O(nw(h+1)^w)$

3. recurrence:

$DP(i,h)$ = max($DP(i,m)$ for valid moves m of the piece i in h)

=⇒ time per subproblem = $O(w)$

4. topo. order: for i in reversed(range(n)): for h · · ·

total time = $O(nwh^w)$ (DAG as above)

5. solution = $DP(o,o)$

(& use parent pointers to recover moves)

Figure 3: Super Mario Bros

Goal: maximize score or minimum times use for level clear

condition

- if anything moves out screen it disappears
- Given entire level (objects, enemies, . . .): n

Dynamic Programming IV

– screen shift (← n, 스크린이 level 의 어느 위치에 있는지)

– player position & velocity (O(1)) (w← mario's action)

– imfomation everything on screen: object states, monster positions, etc. (← $c^{w \cdot h}$)

– anything outside screen gets reset (← $c^{w \cdot h}$)

– score (← S)

– time (← T : 0이 되면 종료)

subproblems: w x S x T x c^{whx}

- transition function δ: (config, action) → config'

nothing, ↑, ↓, ←, →, B, A press/release

(1) subproblem: best score (or time) from config. C
$\implies n \cdot c^{w \cdot h} \cdot S \cdot T$ subproblems

(2) guess: next action to take from C
$\implies O(1)$ choices

(3) recurrence:

$$DP(C) = \begin{cases} C.\text{score} & \text{if on flag} \\ \infty & \text{if } C.\text{dead or } C.\text{time} = 0 \\ \max(DP(\delta(C, A))) & \text{for } A \text{ in actions} \end{cases}$$

$\implies O(1)$ time/subproblem

(4) topo. order: increasing time

(5) orig. prob.: DP(start config.)

- pseudopolynomial in S & T
- polynomial in n
- exponential in $w \cdot h$