Dynamic Programming (II)

Anson Ho

2 March 2019

Definition

- Solve subproblems
- Memorize and reuse the results of the subproblems

Definition

- Solve subproblems
- Memorize and reuse the results of the subproblems

Do you know the differences between *dynamic programming* and *divide and conquer*?

Things you should know

- Base cases
- States (ids of subproblems)
- Transition formula

DAG

- **D**irected **A**cyclic **G**raph

DAG

- **D**irected **A**cyclic **G**raph
- State $\rightarrow$ Vertex
  Transition $\rightarrow$ Edge

DAG

- **D**irected **A**cyclic **G**raph
- State $\rightarrow$ Vertex
  Transition $\rightarrow$ Edge
- Usually serves as a tool of visualizing transitions

DAG

- **D**irected **A**cyclic **G**raph
- State $\rightarrow$ Vertex
  Transition $\rightarrow$ Edge
- Usually serves as a tool of visualizing transitions

How to determine whether a directed graph is acyclic?

Visualization of transition

- Fibonacci number

Visualization of transition

- Fibonacci number
- Number of ways to pay $15

Visualization of transition

- Fibonacci number
- Number of ways to pay $15
  - $10 + $5 and $5 + $10 are the same (count once in total)

Visualization of transition

- Fibonacci number
- Number of ways to pay $15
  - $10 + $5 and $5 + $10 are the same (count once in total)
- Calculate $C_r^n = \frac{n!}{r!(n-r)!}$

Visualization of transition

- Fibonacci number
- Number of ways to pay $15
  - $10 + $5 and $5 + $10 are the same (count once in total)
- Calculate $C_r^n = \frac{n!}{r!(n-r)!}$
  - O(n^2) vs O(n)

Visualization of transition

- Fibonacci number
- Number of ways to pay $15
  - $10 + $5 and $5 + $10 are the same (count once in total)
- Calculate $C_r^n = \frac{n!}{r!(n-r)!}$
  - O(n^2) vs O(n)
- Number of ways to partition n students to k groups

Visualization of transition

- Fibonacci number
- Number of ways to pay $15
  - $10 + $5 and $5 + $10 are the same (count once in total)
- Calculate $C_r^n = \frac{n!}{r!(n-r)!}$
  - O(n^2) vs O(n)
- Number of ways to partition n students to k groups
- Longest path of a DAG

Visualization of transition

- Fibonacci number
- Number of ways to pay $15
  - $10 + $5 and $5 + $10 are the same (count once in total)
- Calculate $C_r^n = \frac{n!}{r!(n-r)!}$
  - O(n^2) vs O(n)
- Number of ways to partition n students to k groups
- Longest path of a DAG
  - Finding the longest path of a general graph is ”hard”

Topological order

- An order of vertices such that all edges go from vertex with smaller index to larger index

Topological order

- An order of vertices such that all edges go from vertex with smaller index to larger index
- Help you to determine the order of states (subproblems)

Topological order

- An order of vertices such that all edges go from vertex with smaller index to larger index
- Help you to determine the order of states (subproblems)
- In most of the cases, it is unnecessary since our transition formulae usually look like the following

$$dp[x][y][z] = f(dp[x_1][y_1][z_1], dp[x_2][y_2][z_2], \ldots, dp[x_k][y_k][z_k])$$

where $x \geq x_i, y \geq y_i, z \geq z_i$

Topological order

- An order of vertices such that all edges go from vertex with smaller index to larger index
- Help you to determine the order of states (subproblems)
- In most of the cases, it is unnecessary since our transition formulae usually look like the following

$$dp[x][y][z] = f(dp[x_1][y_1][z_1], dp[x_2][y_2][z_2], \ldots, dp[x_k][y_k][z_k])$$

where $x \geq x_i, y \geq y_i, z \geq z_i$

How to find **a** topological order of a graph?

Rolling DP

- Number of ways to pay $15

Rolling DP

- Number of ways to pay $15
- Avoid copying data

Rolling DP

- Number of ways to pay $15
- Avoid copying data
- Optimization on memory usage (tiny improvement on runtime)

Rolling DP

- Fibonacci number

Rolling DP

- Fibonacci number
- Find the number of **increasing** sequences of n ($n \leq 1000$) integers $(x_1, x_2, \ldots, x_n)$ such that
  - $1 \leq x_i \leq m$ ($m \leq 1000$)
  - x_i is not a multiple of i for $i > 1$

Rolling DP

- Fibonacci number
- Find the number of **increasing** sequences of n ($n \leq 1000$) integers $(x_1, x_2, \ldots, x_n)$ such that
  - $1 \leq x_i \leq m$ ($m \leq 1000$)
  - x_i is not a multiple of i for $i > 1$
  - Time complexity: O(nm)
  - Space complexity: O(m)

Rolling DP

- Weakness: cannot ”backtrack”
  - not to be confused with backtracking as an exhaustion method

Rolling DP

- Weakness: cannot "backtrack"
  - not to be confused with backtracking as an exhaustion method
- Find the k^{th} lexicographically smallest **increasing** sequences of n ($n \leq 1000$) integers $(x_1, x_2, \ldots, x_n)$ such that
  - $1 \leq x_i \leq m$ ($m \leq 1000$)
  - x_i is not a multiple of i for $i > 1$

Rolling DP

- Weakness: cannot "backtrack"
  - not to be confused with backtracking as an exhaustion method

- Find the k^{th} lexicographically smallest **increasing** sequences of n ($n \leq 1000$) integers $(x_1, x_2, \ldots, x_n)$ such that
  - $1 \leq x_i \leq m$ ($m \leq 1000$)
  - x_i is not a multiple of i for $i > 1$
  - Time complexity: $O(nm)$
  - Space complexity: $O(\sqrt{n}m)$

Undo

- Some DP transitions can be reversed

Undo

- Some DP transitions can be reversed
- Fibonacci number

Undo

- Some DP transitions can be reversed
- Fibonacci number
- Online handle the following types of queries (1000 queries in total)
  - Add a \$$x$ coin ($x \leq 1000$)
  - Remove a \$$y$ coin (which is added before)
  - Output whether \$$z$ can be paid exactly

Undo

- Some DP transitions can be reversed
- Fibonacci number
- Online handle the following types of queries (1000 queries in total)
  - Add a \$$x$ coin ($x \leq 1000$)
  - Remove a \$$y$ coin (which is added before)
  - Output whether \$$z$ can be paid exactly
  - Solution 1: count number of ways to pay \$$w$ module p (not 100% correct)

Undo

- Some DP transitions can be reversed
- Fibonacci number
- Online handle the following types of queries (1000 queries in total)
  - Add a \$$x$ coin ($x \leq 1000$)
  - Remove a \$$y$ coin (which is added before)
  - Output whether \$$z$ can be paid exactly
  - Solution 1: count number of ways to pay \$$w$ module p (not 100% correct)
  - Solution 2: bitset

Tree

- What is tree?

Tree

- What is tree?
- Tree is a DAG
  - Root a unrooted tree for DP
  - Direct all edges from the root

  (DAG is not necessarily a tree)
  Then we can do DP on a tree

Things you should know

- Root, leaf

Things you should know

- Root, leaf
- Parent, child

Things you should know

- Root, leaf
- Parent, child
- Ancestor, descendant

Things you should know

- Root, leaf
- Parent, child
- Ancestor, descendant
- Height, depth

Things you should know

- Root, leaf
- Parent, child
- Ancestor, descendant
- Height, depth
- Subtree

Tree DP

- MinMax in game tree

Tree DP

- MinMax in game tree
- Number of paths passing through each node

Tree DP

- MinMax in game tree
- Number of paths passing through each node
- Sum of path weights
  - Path weight = sum of edge weights
  - Path weight = product of edge weights

Tree DP

- MinMax in game tree
- Number of paths passing through each node
- Sum of path weights
  - Path weight = sum of edge weights
  - Path weight = product of edge weights
- Number of connected subgraphs of size k

Suggested reading

- Looking for a Challenge - Barricades (inside free preview) (`http://www.lookingforachallengethebook.com/`)

Bit manipulation

- Bitwise and (`&`)
- Bitwise or (`|`)
- Bitwise xor (`^`)
- Bitwise shift (<<, >>)

Bitwise trick (C++)

- test j^{th} bit on i

```
if (i && (1 << j))
```

Bitwise trick (C++)

- test j^{th} bit on i
  `if (i && (1 << j))`
- get i ones from the least significant bit
  `(1 << i) - 1`

Bitwise trick (C++)

- test j^{th} bit on i
  `if (i && (1 << j))`
- get i ones from the least significant bit
  `(1 << i) - 1`
- is i a submask of j
  `(i & j) == i`

Bitwise trick (C++)

- test j^{th} bit on i
  `if (i && (1 << j))`
- get i ones from the least significant bit
  `(1 << i) - 1`
- is i a submask of j
  `(i & j) == i`
- enumerate non-empty submasks of j
  `for(int i = j; i > 0; i = (i - 1) & j)`

Bitwise DP

- some part of the state is binary number
- e.g. dp[2][0111011_2]

Bitwise DP

- N light bulbs ($N \leq 15$)
- M buttons, each toggles a set of light bulbs when pressed ($M \leq 30$)
- Find minimum number of times of pressing the buttons to achieve a given state
  Or output ”impossible”

Bitwise DP

- M1830 Lazy Tutor

Bitwise DP

- M1830 Lazy Tutor
- See last year mini comp editorial (or Codeforces blog - SOS DP)
  - `https://assets.hkoi.org/training2018/2018-mc3.pdf`
  - `https://codeforces.com/blog/entry/45223`

Suggested problems

- AtCoder Educational DP Contest (`https://atcoder.jp/contests/dp`)