Dynamic Programming (II)

Fuzen Ng (yfng)

2022-02-26

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Why DP?

- DP is a very common technique in OI
- Some tasks may divide subtasks into different levels of DP
- Some subtasks could be done by DP even the full solution is not DP

How to DP?

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

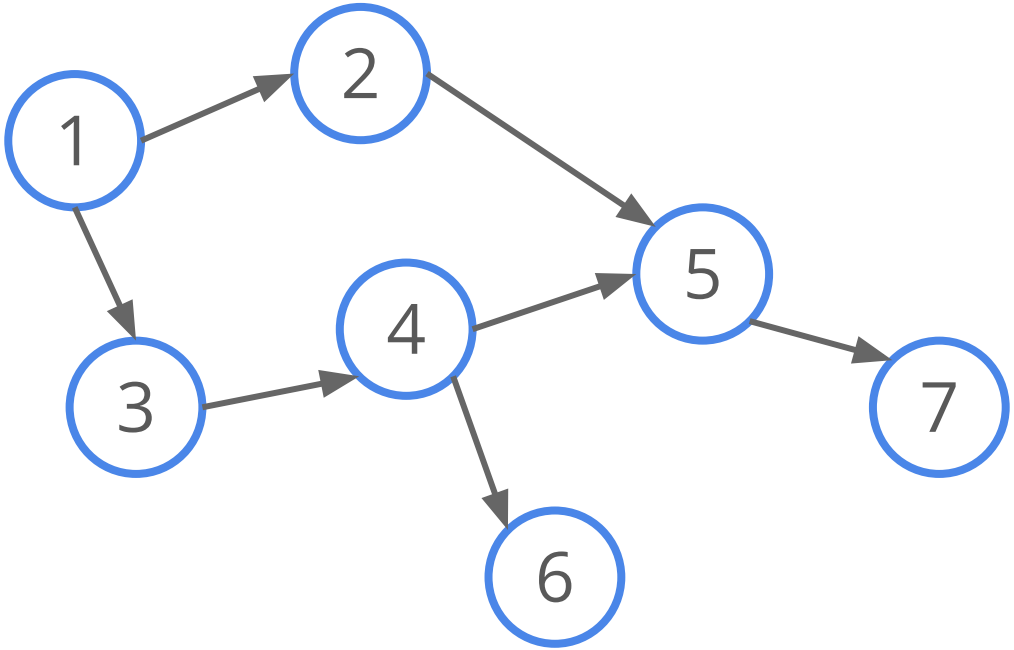
Things you should know

- Bases cases
  Subproblems that cannot be reduced
- States
  IDs of subproblems
- Transition formula
  Using results of subproblems to find the answer

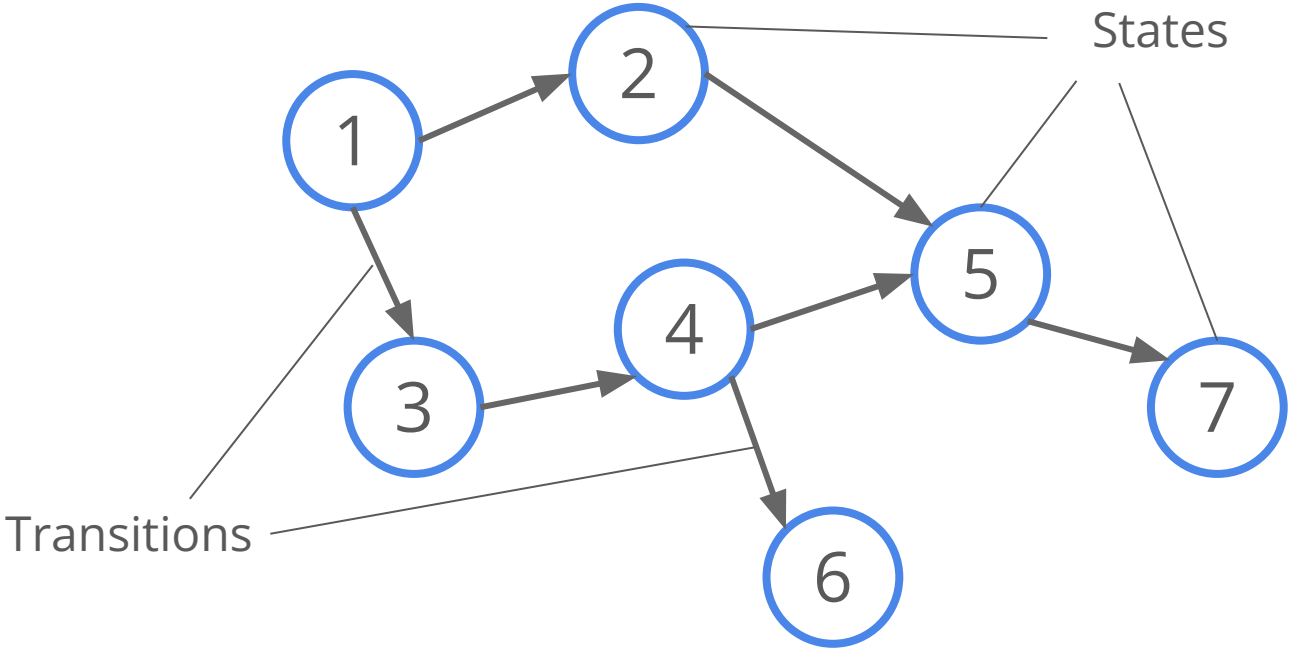
DAG

- **D**irected **A**cyclic **G**raph 有向無環圖
- Node = State
- Edge = Transition
- Can be used as a tool to visualize transitions

DAG

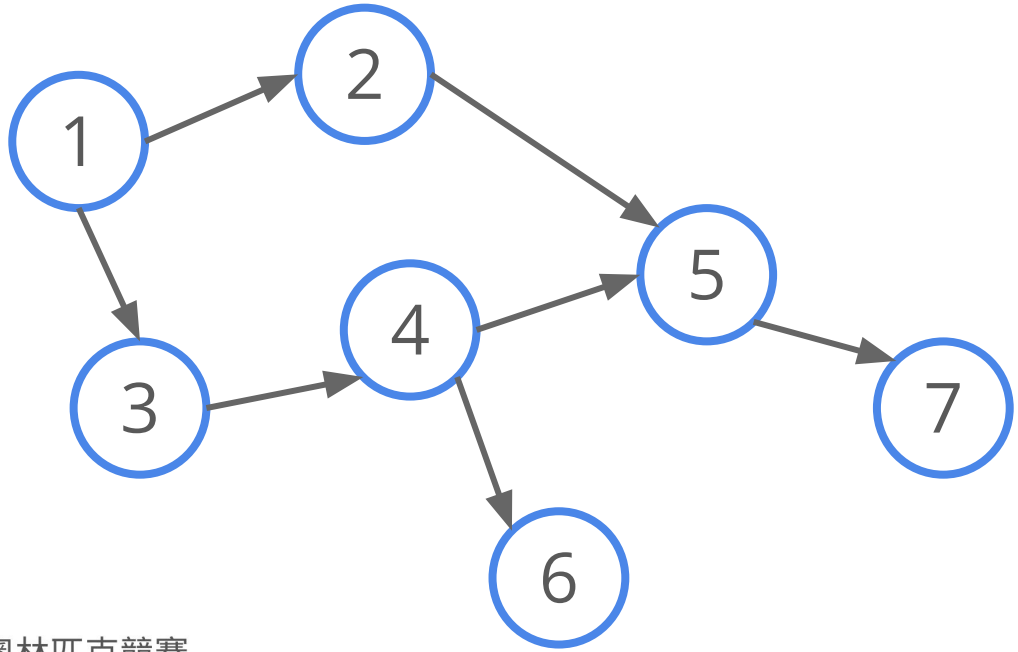


DAG



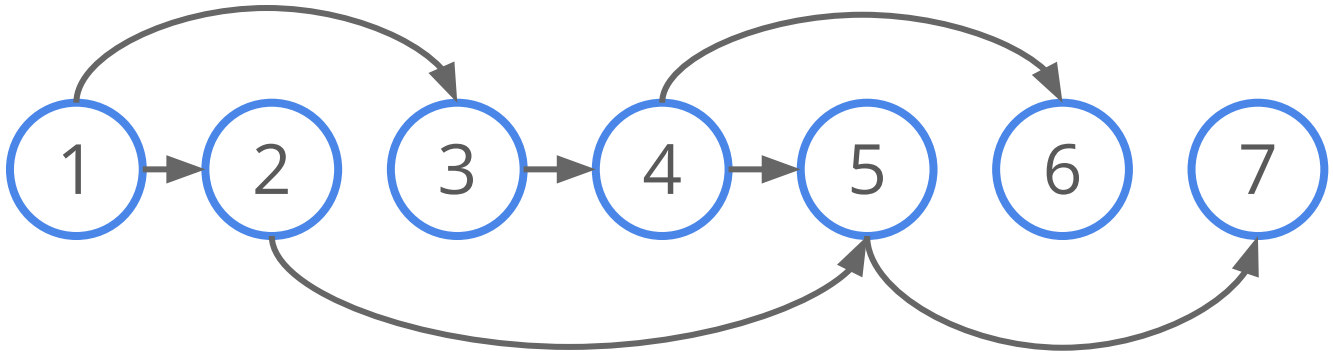
DAG - HKOJ M1862 Little Patterns, Big Canvas

- **DAG** is usually applied with topological sort in order to determine the DP order



DAG - HKOJ M1862 Little Patterns, Big Canvas

- **DAG** is usually applied with topological sort in order to determine the DP order



DAG - HKOJ M1862 Little Patterns, Big Canvas

- N patterns numbered from 1 to N
- Patterns are investigated one by one in an arbitrary order
- For each pattern under investigation, either assigned to it a partner pattern selected from previously investigated patterns, or assigned no partner pattern
- p_i = partner pattern of i (p_i = 0 if i has no partner pattern)
- s_i = number of strokes to draw pattern i
- One second is needed to draw a stroke
- Stokes of two patterns i and j can be drawn at the same time iff p_i = j or p_j = i
- Find the minimum number of seconds to finish N patterns

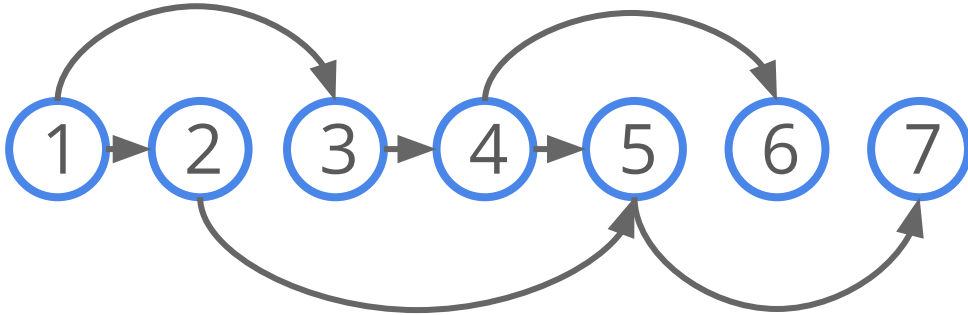
DAG - HKOJ M1862 Little Patterns, Big Canvas

- **DAG** is usually applied with topological sort in order to determine the DP order
- Then you can perform the DP easily with the order!

Like in M1862

If you sort the DAG with topological sort

It can be solved by greedy/dp

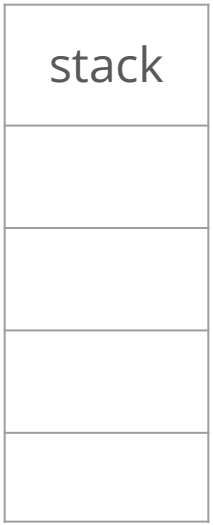


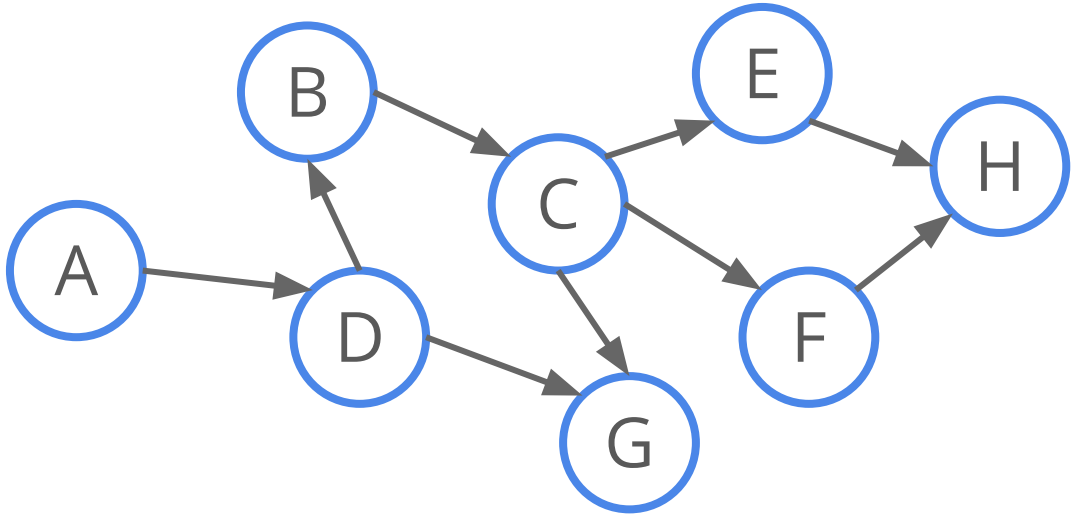
DAG - Topological Sort

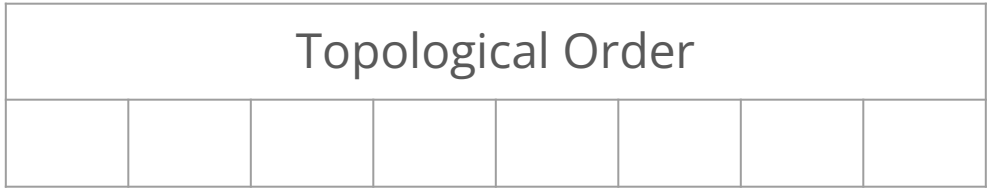
- Obtain an order of nodes so that if there exist a directed edge from node A to node B ⇒ node A appears before node B in the order

```
while some nodes are unvisited
    choose any unvisited node
    DFS from the node
        push the node into a stack
        recur to visit an unvisited node
        pop from the stack when there are no more unvisited nodes
            insert the node into the topological order in reverse order
```

DAG - Topological Sort

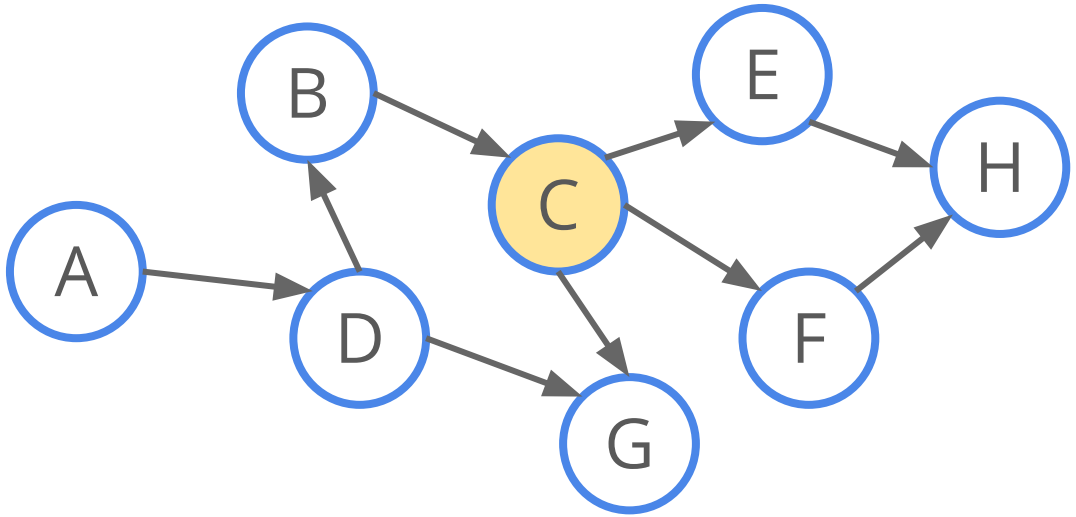






DAG - Topological Sort

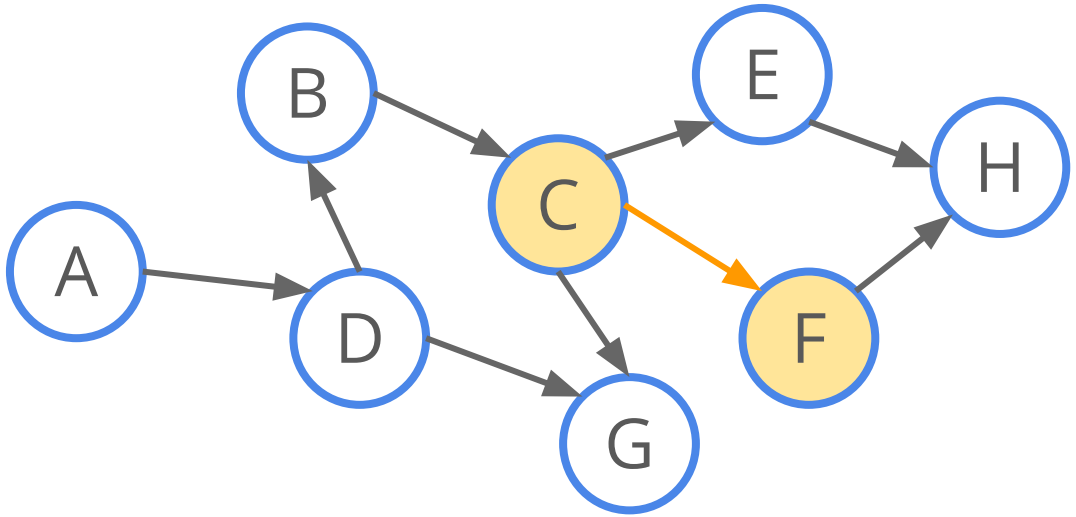
stack
C



Topological Order							

DAG - Topological Sort

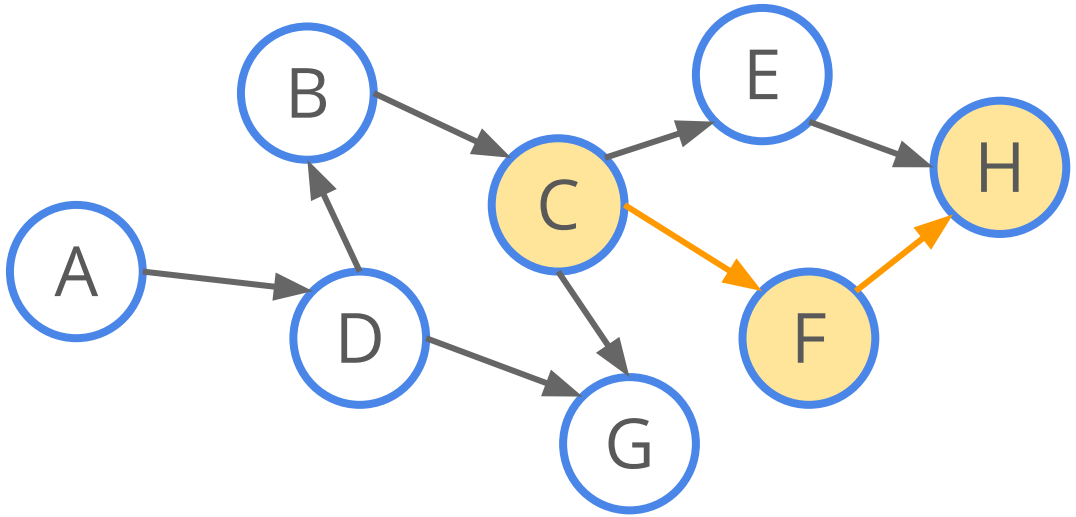
stack
C
F



Topological Order							

DAG - Topological Sort

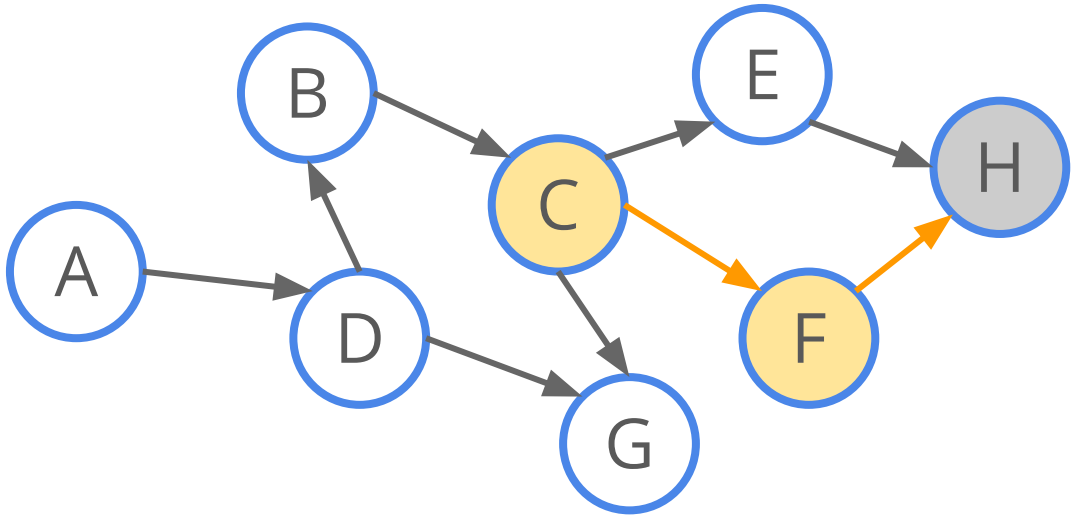
stack
C
F
H



Topological Order							

DAG - Topological Sort

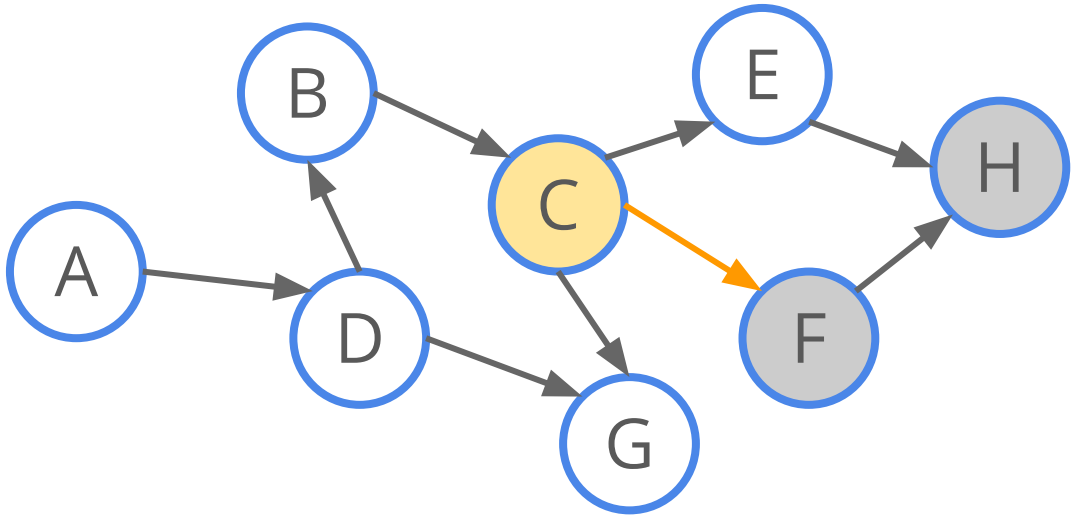
stack
C
F



Topological Order							
							H

DAG - Topological Sort

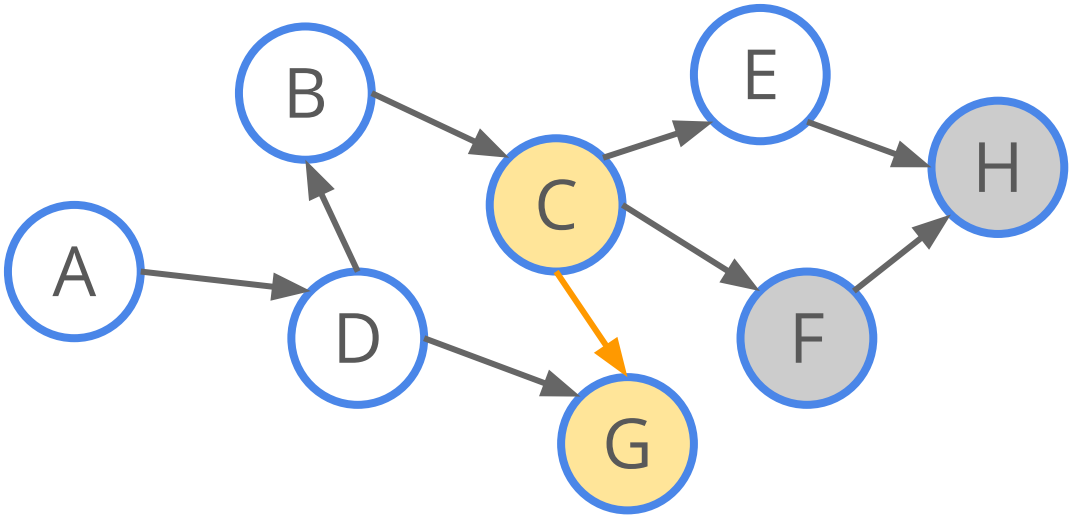
stack
C



Topological Order							
						F	H

DAG - Topological Sort

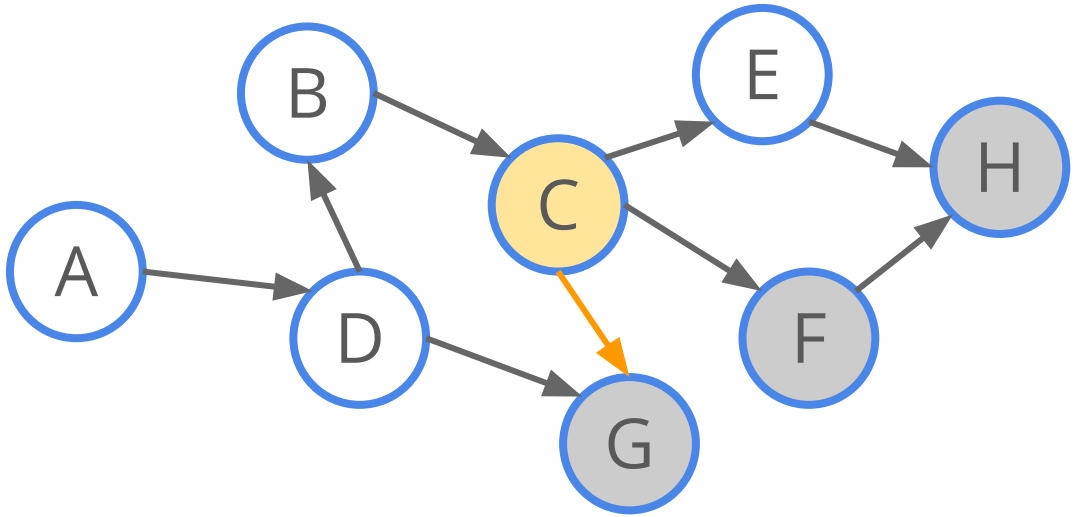
stack
C
G



Topological Order							
						F	H

DAG - Topological Sort

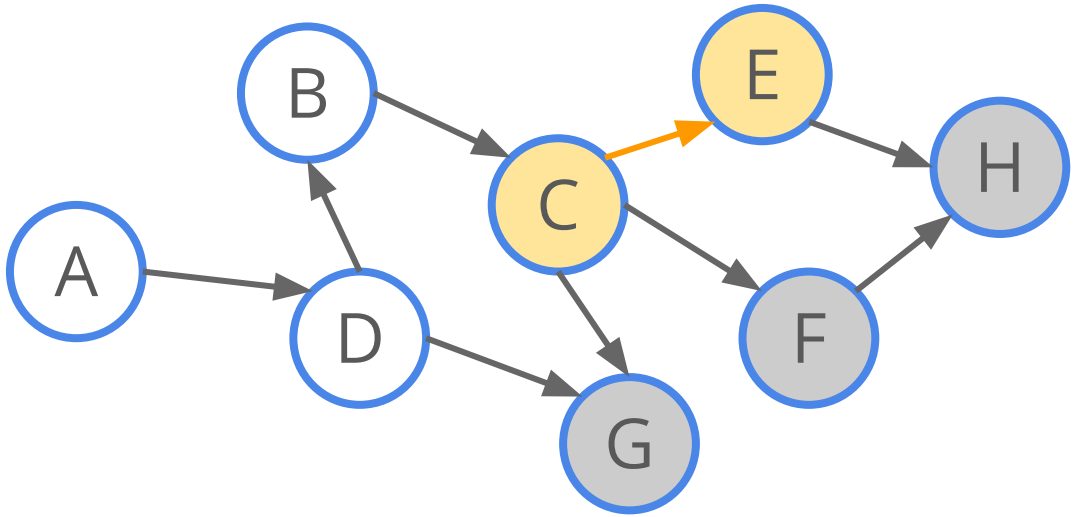
stack
C



Topological Order							
					G	F	H

DAG - Topological Sort

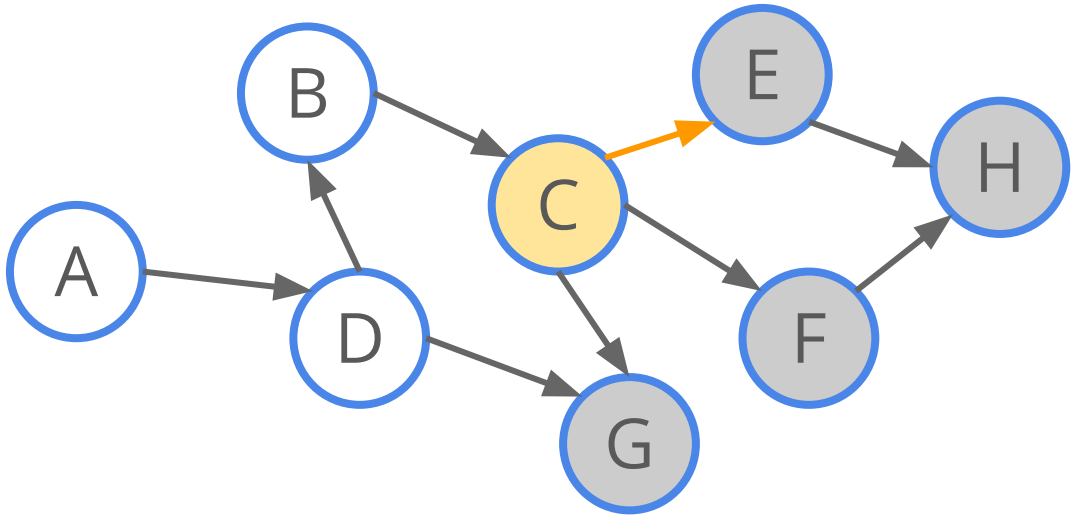
stack
C
E



Topological Order							
					G	F	H

DAG - Topological Sort

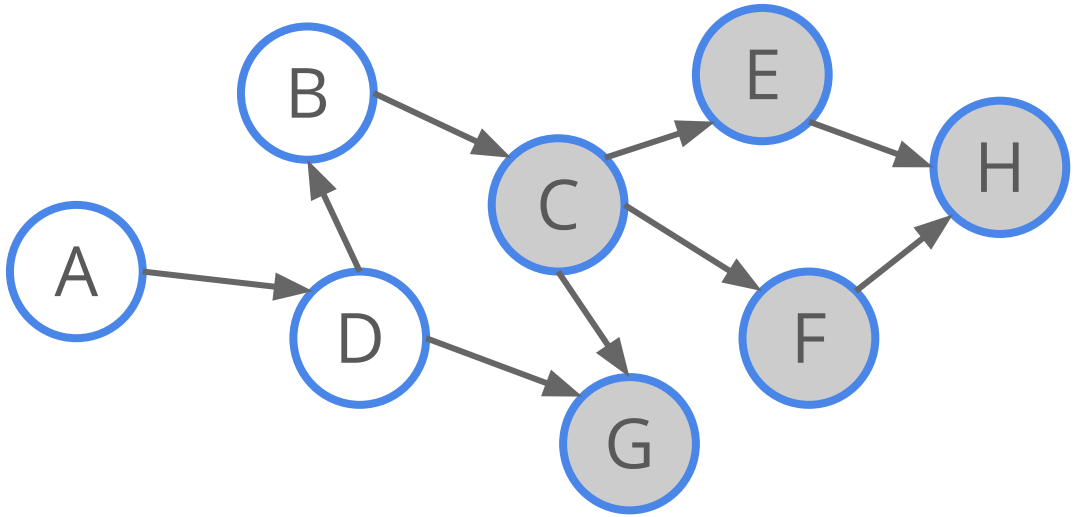
stack
C



Topological Order							
				E	G	F	H

DAG - Topological Sort

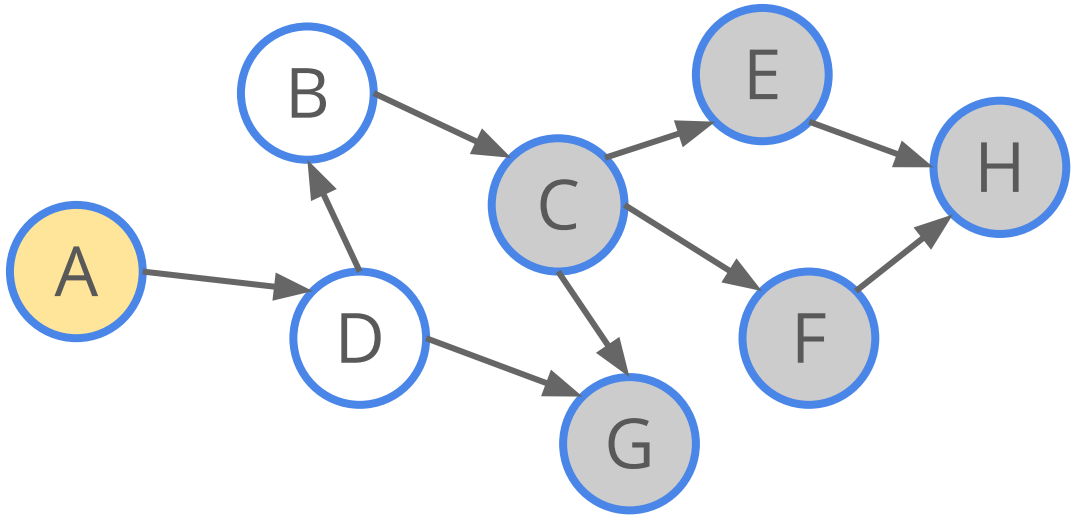
stack



Topological Order							
			C	E	G	F	H

DAG - Topological Sort

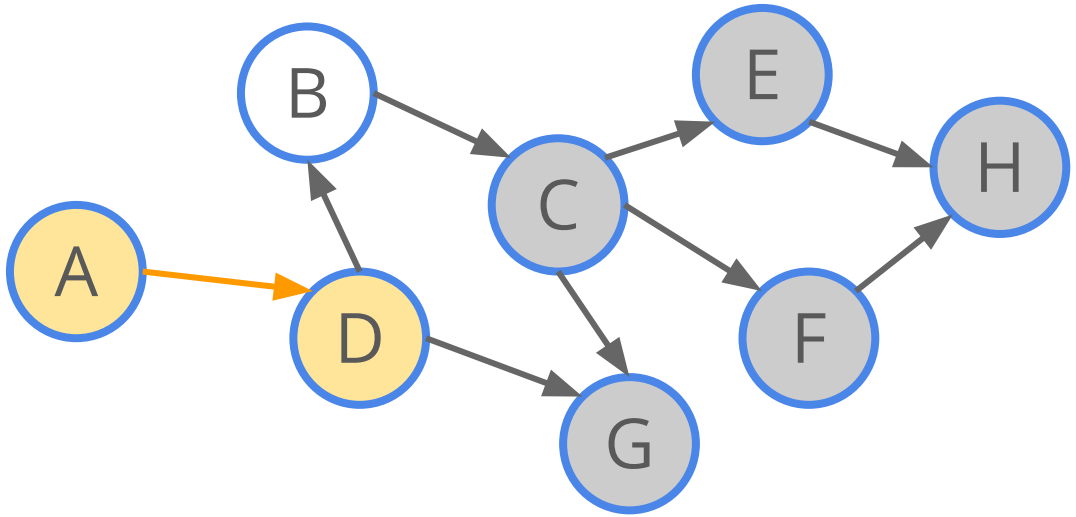
stack
A



Topological Order							
			C	E	G	F	H

DAG - Topological Sort

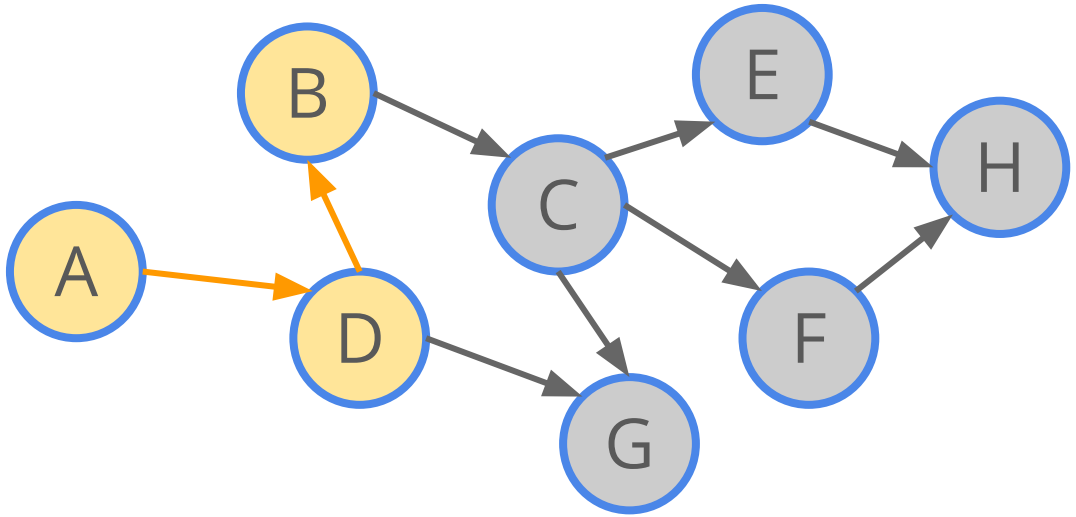
stack
A
D



Topological Order							
			C	E	G	F	H

DAG - Topological Sort

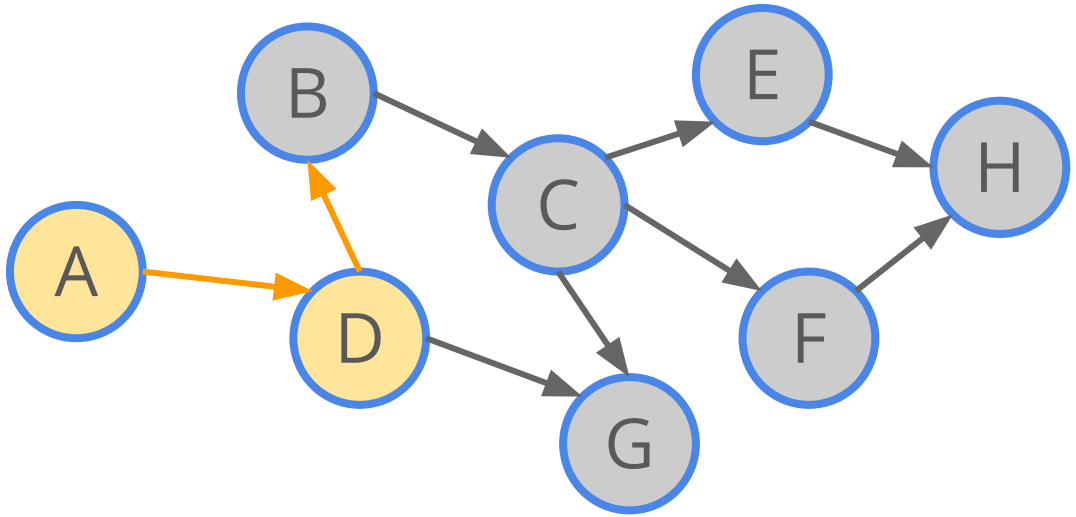
stack
A
D
B



Topological Order							
			C	E	G	F	H

DAG - Topological Sort

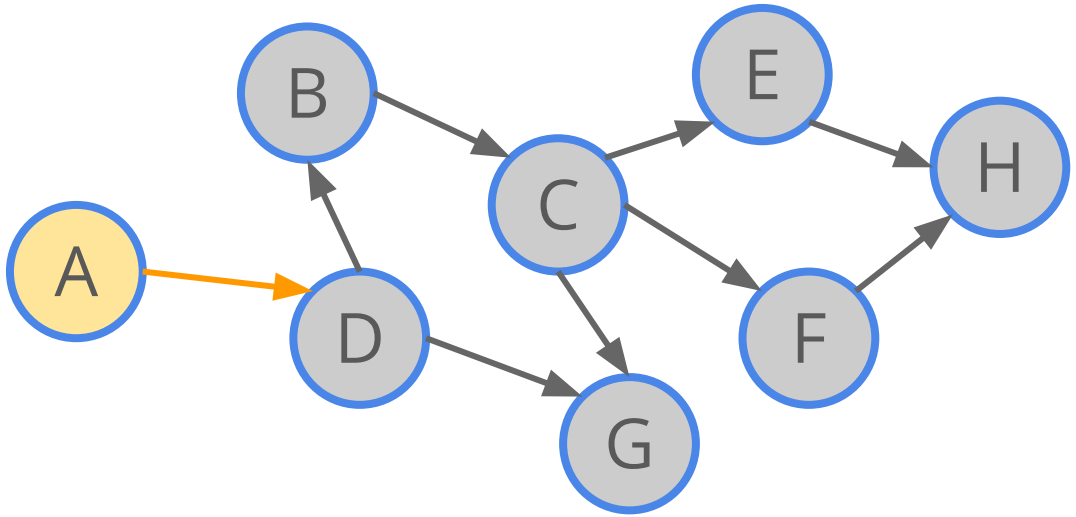
stack
A
D



Topological Order							
		B	C	E	G	F	H

DAG - Topological Sort

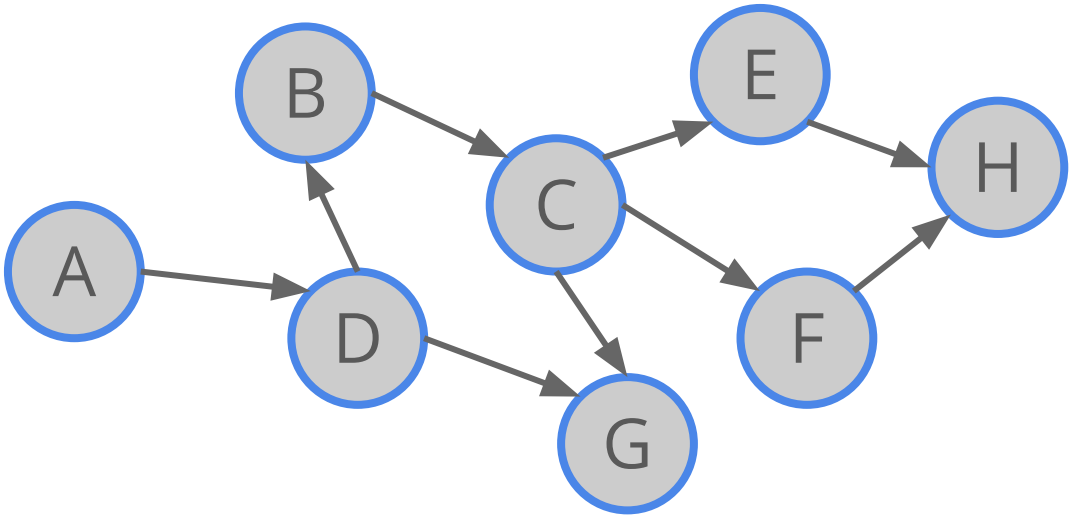
stack
A



Topological Order							
	D	B	C	E	G	F	H

DAG - Topological Sort

stack



Topological Order							
A	D	B	C	E	G	F	H

DAG - HKOJ M1862 Little Patterns, Big Canvas

- Find an order of the patterns so that for any $p_i = j$, i is earlier than j
- Build a graph with directed edges $i \rightarrow j$ for $p_i = j$

- Topological sort

- Greedily draw the strokes according to the topological order

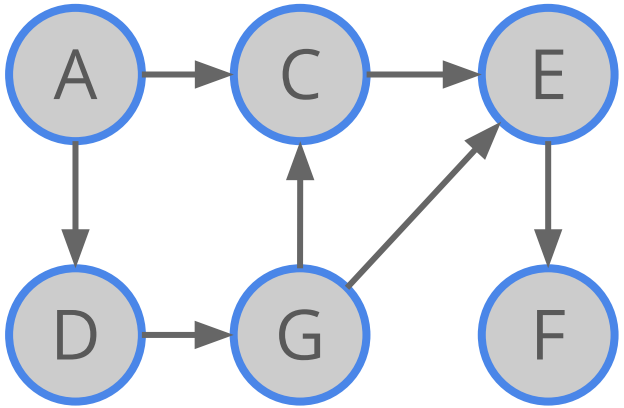
DAG - Counting number of paths

How can we use dp to count number of paths of a DAG?

Eg: count number of paths from A to F

1. A → C → E → F
2. A → D → G → E → F
3. A → D → G → C → E → F

There are three paths in total



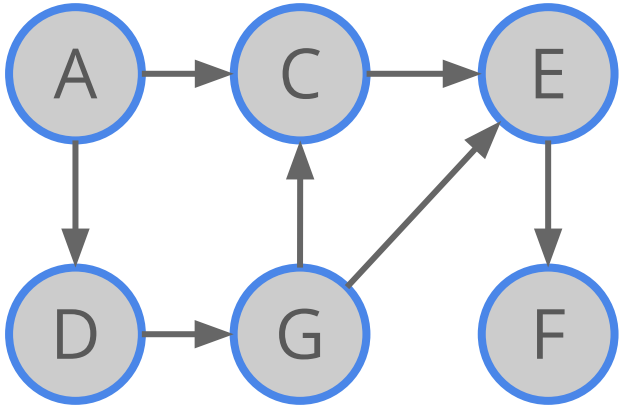
DAG - Counting number of paths

Let paths(x) denote the number of paths from node A to node x

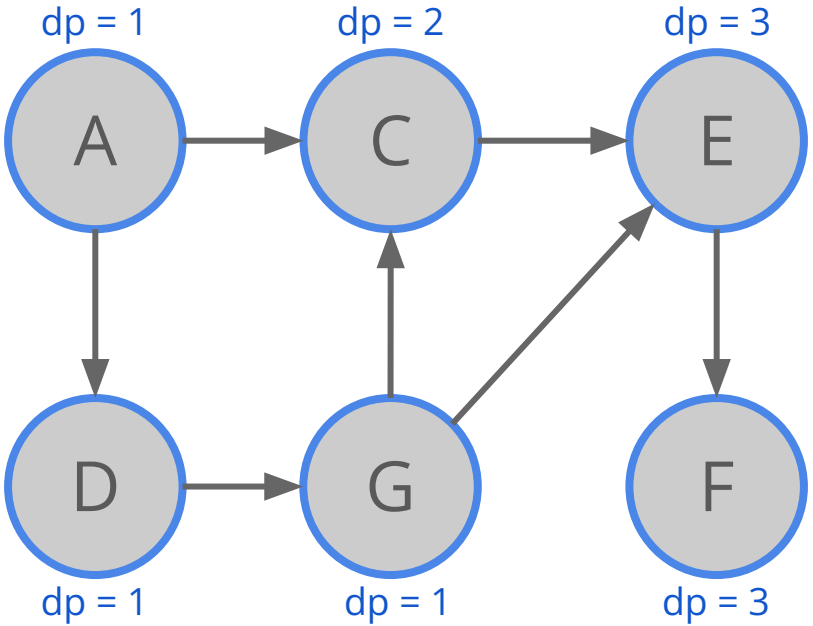
Since the graph is acyclic, it can be counted as follows:

$$\text{paths}(x) = \text{paths}(a_1) + \text{paths}(a_2) + \ldots + \text{paths}(a_k)$$

Where a_i are the nodes from which there is an edge to x.



DAG - Counting number of paths



DAG - DP problems

Try to solve the following DAG DP problems:

- the minimum number of edges in a path of a DAG

DAG - DP problems

Try to solve the following DAG DP problems:

- the minimum number of edges in a path of a DAG
- the path which contains the maximum number of edges of a DAG

DAG - DP problems

Try to solve the following DAG DP problems:

- the minimum number of edges in a path of a DAG
- the path which contains the maximum number of edges of a DAG
- the number of shortest paths from a source in an undirected graph

DAG - DP problems

The number of shortest paths from a source in an undirected graph:

1. Perform Dijkstra to find shortest path from source to each node (or any shortest path algorithm you like)

2. If (dist[A] + edge_cost == dist[B]) then we add a directed edge from A to B

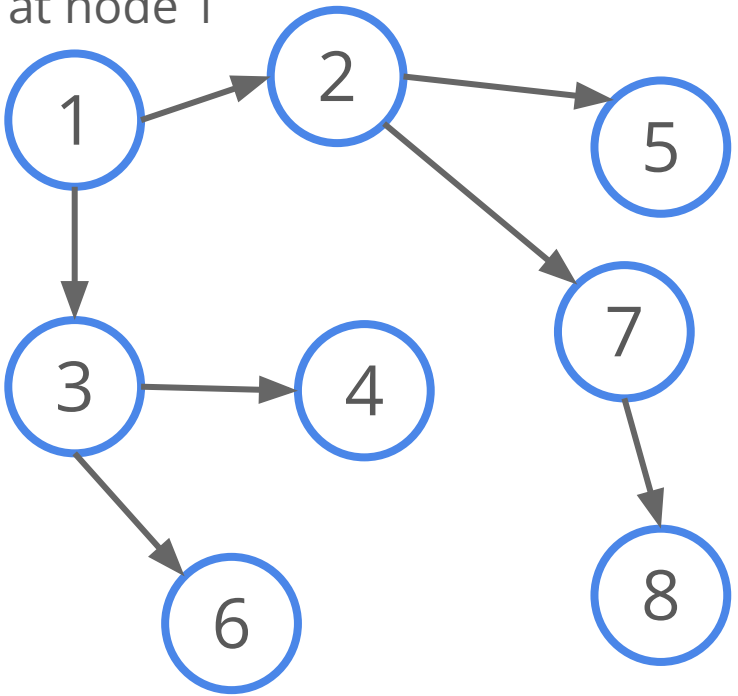
3. Transformed into counting number of paths in DAG!!!

break;

Tree

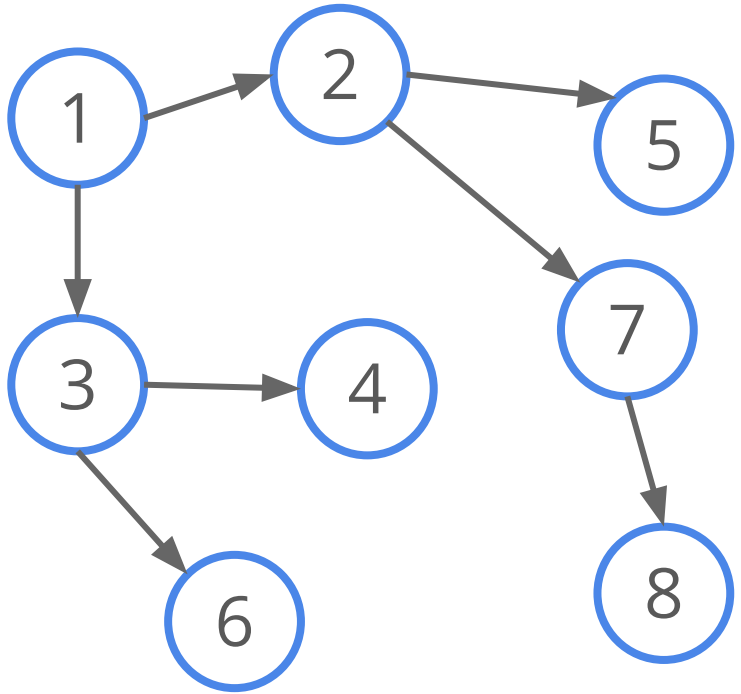
- A special case of DAG (if it is rooted)
- N nodes and N-1 edges
- Exist an unique path from a node to any other nodes

Rooting the tree at node 1



Tree - Glossary

- Root, Leaf
- Parent, Child
- Ancestor, Descendant
- Height, Depth
- Subtree



Tree DP

- Given a rooted tree - easier

- Given an unrooted tree with bidirectional edges
- You may need to root it yourself
- e.g. by assigning a random node as the root

Tree DP

- Assume a tree is rooted
- Use nodes as DP states
- Use nodes' children as transition formula reference

Tree DP - Example 0

- Given a rooted tree of size **N**
- Calculate the size of each subtree

- For each node, recursively count number of nodes in the subtree

- Time complexity = $O(N^2)$

Tree DP - Example 0

- Given a rooted tree of size **N**
- Calculate the size of each subtree

- `dp[i] = size of subtree i`
- `dp[i] = 1 + sum(dp[j]) where j is i's children`

- Each node is visited once only
- Time complexity = `O(N)`

Tree DP - Example 1

- Given a rooted tree of size **N**
- Each node is labeled by a value
  Node `i` has the value `v[i]`
- **Q** queries
  Find the greatest value in a subtree

Tree DP - Example 1

- Given a rooted tree of size **N**
- Each node is labeled by a value
  Node `i` has the value `v[i]`
- **Q** queries
  Find the greatest value in a subtree

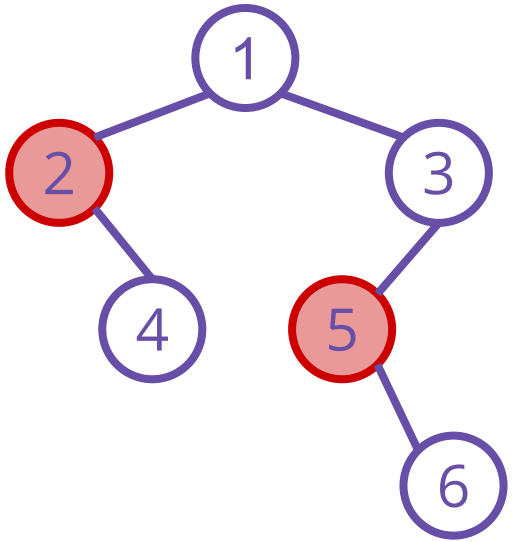
- `dp[i] = answer for subtree i`
- `dp[i] = max(v[i], dp[j]) where j is i's children`

- Each node is visited once only
- Time complexity = `O(N)`

Tree DP - Example 2

- Given a rooted binary tree with size **N**
- You have to paint all nodes by assigning painter to nodes
- A painter at a node can paint the node itself, its parent and its immediate children
- Find the minimum number of painters required

Time complexity required : `O(N)`



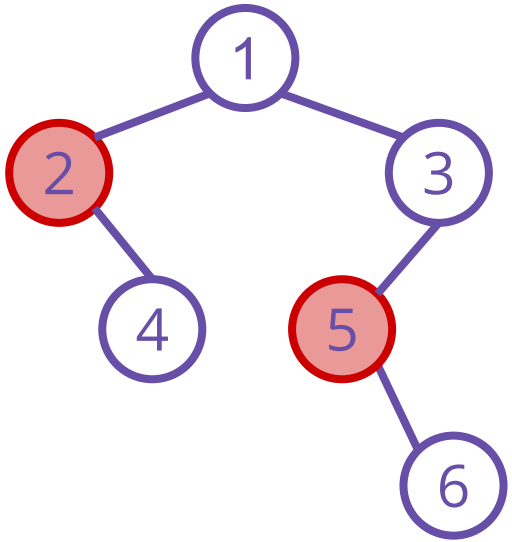
Tree DP - Example 2

- For each node define 3 dp state:

```
dp[i][0] = {assign a painter to this node}
dp[i][1] = {this node is covered by its children node's painter}
dp[i][2] = {this node is not covered by any other node}
```

  Take min value in the transition

Time complexity achieved : O(N)



Tree DP - Example 2

- Base case(leaf): dp[i][0] = 1, dp[i][1] = 1, dp[i][2] = 0
- Observation: dp[i][2] <= dp[i][1] <= dp[i][0]
- Not necessary

- consider node i with children j and k:
- dp[i][0] = {assign a painter to this node}
  - 1 + dp[j][2] + dp[k][2]
- dp[i][1] = {this node is covered by its children node's painter}
  - min(dp[j][0] + dp[k][1], dp[j][1] + dp[k][0])
- dp[i][2] = {this node is not covered by any other node}
  - dp[j][1] + dp[k][1]

Tree DP - Example 3

- Given a rooted tree of size **N**
- Calculate number of simple paths passing through each node

Tree DP - Example 3

- Given a rooted tree of size **N**
- Calculate number of simple paths passing through each node

- `sz[i] = size of subtree i` (Example 0)
- Answer for node x can be calculated with `sz[j]` where `j` is x's children
- Treat x as the root
  `(N - sz[x])` as one of the subtrees
- For each subtree of x,
  `ans += sz[this subtree] * sum(sz[all other subtrees])`
  Final `ans = ans / 2`
- Time complexity = `O(N)`

Tree DP - HKOJ I1022 Traffic Congestion

- Given an unrooted tree of size **N**, representing **N** cities
- `P[i]` people live in city `i`

- A city will be chosen so that all people travel between their home and the chosen city
- Congestion of a road is defined as the number of people that travel along the road
- Choose the city so that the greatest congestion is as small as possible

- Time complexity required : `O(N)`

Tree DP - HKOJ I1022 Traffic Congestion

- Root the tree at node 0

- `Sum[i] = sum of people in subtree i`
- `Max[i] = maximum congestion in subtree i if node i is chosen`
- `Max[i] = max(Sum[j]) where j is i's children`

Tree DP - HKOJ I1022 Traffic Congestion

- Root the tree at node 0

- `Sum[i] = sum of people in subtree i`
- `Max[i] = maximum congestion in subtree i if node i is chosen`
- `Max[i] = max(Sum[j]) where j is i's children`

- Maximum congestion for node `i = max(Max[i], Sum[0] - Sum[i])`

- Time complexity achieved : `O(N)`

Tree DP - Summary

- Rooted tree
- DFS from root
- Recursively calculate answers of the children
- Calculate answer for this node
- May require traveling several times to precompute values used to find answers
- Subtree size / sum of subtree / height

break;

Bitwise DP

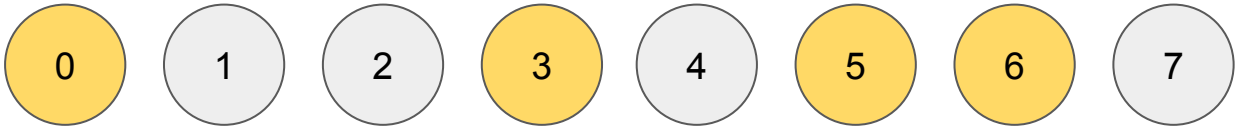
- Using bitmask as some states of the dp

- E.g. `dp[3][01101001₂]` (`dp[3][105]`)

- Bitmask: a sequence of bits, usually an integer written in binary notation
- Each bit can take on the value of 0 or 1, usually used to represented state of on / off or being chosen / not being chosen

Bitwise DP - State

- Example 1 (assume the followings are light bulbs):



- Treating the lit bulbs as 1, unlit bulbs as 0, this state can be represented by bitmask $\mathbf{01101001_2}$ (105_{10})

- We corresponds the i-th bit (counting from right to left) with the i-th light bulb. In this order, the bitmask can be calculated by $\mathbf{\Sigma 2^i}$ for i-th bulb being lit

Bitwise DP - State

- Example 2 (0-1 Knapsack Problem):
- Given **N** items with weight $\mathbf{w_i}$ and value $\mathbf{v_i}$, you may pick a subset of items such that their total weight ≤ **K**. Find the maximum total value of items picked
- As with the previous example, each subset can be represented by a bitmask (i-th item ↔ i-th bit), and can be fitted into a dp state

- Although there exist better solution, coming up with state of bitmask dp is usually easy and can earn you some basic marks

Bitwise DP - Bitwise Tricks

- Bit manipulation tricks are usually used in bitwise dp

- Bitwise AND (`&`)
- Bitwise OR (`|`)
- Bitwise XOR (`^`)
- Bitwise SHIFT (`<<`, `>>`)
  - `x << y`: Shift x left by y bits
  - `5 << 4 = 80` ($5_{10} = 101_2$, $80_{10} = 1010000_2$)

Bitwise DP - Bitwise Tricks

- 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` (from large to small)

  ```
  for (int i = j; i > 0; i = (i - 1) & j)
  ```

Bitwise DP - Transition

- Given **N** light bulbs ($N \leq 15$), **M** buttons, each toggles (on → off, off → on) a set of light bulbs ($\mathbf{B}_i$ - in bitmask form) when pressed ($M \leq 30$)
- Find minimum number of times of pressing the buttons to achieve a given state (**K**) Or output ”impossible”

Bitwise DP - Transition

- Given **N** light bulbs (N ≤ 15), **M** buttons, each toggles (on → off, off → on) a set of light bulbs ($\mathbf{B}_i$ - in bitmask form) when pressed (M ≤ 30)
- Find minimum number of times of pressing the buttons to achieve a given state (**K**) Or output ”impossible”

- `dp[i][bitmask]`: Considering only button 1 to `i`, the minimum number of presses needed to achieve the light bulb state in bitmask
- Base case: `dp[0][0] = 0`
- Answer: `dp[M][K]`
- Transition: `dp[i][bitmask] = min(____)`

Bitwise DP - Transition

- For each button, either choose to press it, or not press it
- `dp[i][bitmask]`: Considering only button 1 to `i`, the minimum number of presses needed to achieve the light bulb state in bitmask

- Transition (assume unachievable states are handled):

```
dp[i][bitmask]
= min(dp[i - 1][bitmask ^ B[i]] + 1, dp[i - 1][bitmask])
```

- Time complexity: $O(M * 2^N)$

Bitwise DP - HKOJ M0712 Maximum Sum II

- Given N×N positive integers
- Find the maximum sum of N numbers
- No two numbers are on the same row or the same column

- $1 \le N \le 16$

SAMPLE TESTS

	Input	Output
1	3 1 1 10 2 5 10 1 10 3	22

Bitwise DP - HKOJ M0712 Maximum Sum II

- `dp[i][bitmask]` = the maximum sum of i numbers from the first i rows, by choosing columns represented by the bitmask

- Transition:
  for each column j

```
if (bitmask & (1 << j) == 0)
dp[i][bitmask + (1 << j)] = max(dp[i][bitmask + (1 << j)],
dp[i - 1][bitmask] + a[i][j])
```

- Answer: $dp[N][2^N - 1]$
- Time complexity: $O(N^2 * 2^N)$

Bitwise DP - HKOJ M0712 Maximum Sum II

- Transition:
  for each column j

```
if (bitmask & (1 << j) == 0)
dp[i][bitmask + (1 << j)] = max(dp[i][bitmask + (1 << j)],
dp[i - 1][bitmask] + a[i][j])
```

- precompute number of 1s in all bitmasks
- For each i, only consider bitmasks that number of 1s equals i - 1
- Time complexity: $O(N * 2^N)$

Bitwise DP - M2136 Guardian

- N witches, each with strength S_i and after effect E_i
- Need to fight all witches one by one
- Choosing witch x as the first one to fight against costs S_X % M energy.
- For $2 \le j \le N$, choosing witch x as the j^{th} one to fight against and witch y as the $j-1^{th}$ one to fight against costs $(j \times S_X \times E_y)$ % M energy.
- Find minimum sum of energy to fight N witches with optimal order

Bitwise DP - M2136 Guardian

The bit (1 << i) in the bitmask represent whether witch i is already chosen

- dp[i][j] = the minimum cost to choose the witches represented by the bitmask j while the most recently chosen one is witch i
- Base case: dp[i][1 << i] = s_i % M, other states = inf
- Try all k that (j & (1 << k) == 0) If (bitmask & (1 << i) != 0)
  Update dp[k][j ^ (1 << k)] with dp[i][j] + cnt * S_k * E_i % M, where cnt is (number of 1s in j) + 1

Bitwise DP - M2136 Guardian

- Answer: minimum of dp[i][(1 << N) - 1]
- Time complexity: $O(N^2 * 2^N)$

Memory optimization - Rolling Array

- Optimization on memory usage
- Avoid saving data that is no longer useful
- Tiny improvement on runtime

Memory optimization - Rolling Array

- Solve the following problem with DP
- N * M grid
- Calculate the number of ways to move from (1, 1) to (N, M) with right and down movement only

Memory optimization - Rolling Array

- Solve the following problem with DP
- N * M grid
- Calculate the number of ways to move from (1, 1) to (N, M) with right and down movement only
- `dp[i][j] = number of ways to move from (1,1) to (i, j)`
- `dp[1][1] = 1`
- `dp[i][j] = dp[i - 1][j] + dp[i][j - 1]`

- Memory complexity = `O(NM)`

Memory optimization - Rolling Array

- `dp[i][j] = dp[i - 1][j] + dp[i][j - 1]`
- Only `dp[i - 1][1..M]` is needed
- `dp[i][1..M]` can be calculated without referring to `dp[1..i-2][1..M]`

- Keeping two rows of dp states is enough
- Alternatively use `dp[0][1..M]` and `dp[1][1..M]` for 1 to N

- Memory complexity = `O(M)`

- Swapping `N` and `M` if `M > N` ⇒ `O(min(N, M))`

Practice Problem Summary

- DAG DP

  M1862 Little Patterns, Big Canvas
- Tree DP

  I1022 Traffic Congestion
- Bitwise DP

  M0712 Maximum Sum II

  M2136 Guardian

More Practice Problems

- Bitwise DP

  M1830 Lazy Tutor
  NP1722 寶藏
  CF1285D Dr. Evil Underscores
  CF1391D 505

- Tree DP

  CF839C Journey
  T153 Congressman Lee Sin
  M0422 Christmas Tree
  2021 ICPC Asia Nanjing Regional H Crystalfly

- Rolling Array

  I0011 Palindrome
  T003 Scheduling Lectures
  I0721 Miners

Additional Readings

- SOS Dynamic Programming
  https://codeforces.com/blog/entry/45223
- Non-trivial DP Tricks and Techniques
  https://codeforces.com/blog/entry/47764

Q&A