Dynamic Programming

Modan

What is DP?

- Dynamic programming?
- Recursion?
- The hardest of the 4 common problem solving paradigms?
- The easiest of the 4 common problem solving paradigms?

Problem

DAG Paths

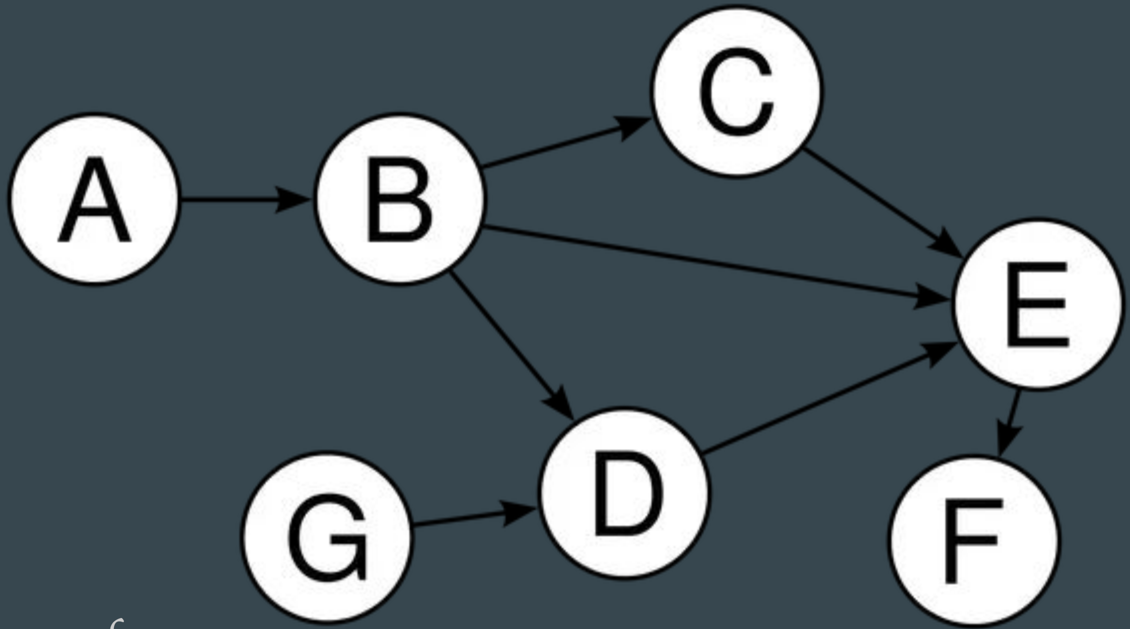
Given a directed acyclic graph, a start vertex and an end vertex, how many ways are there to go from start to end?

Definitions

A directed acyclic graph (DAG) is a directed graph with no cycles.

Definitions

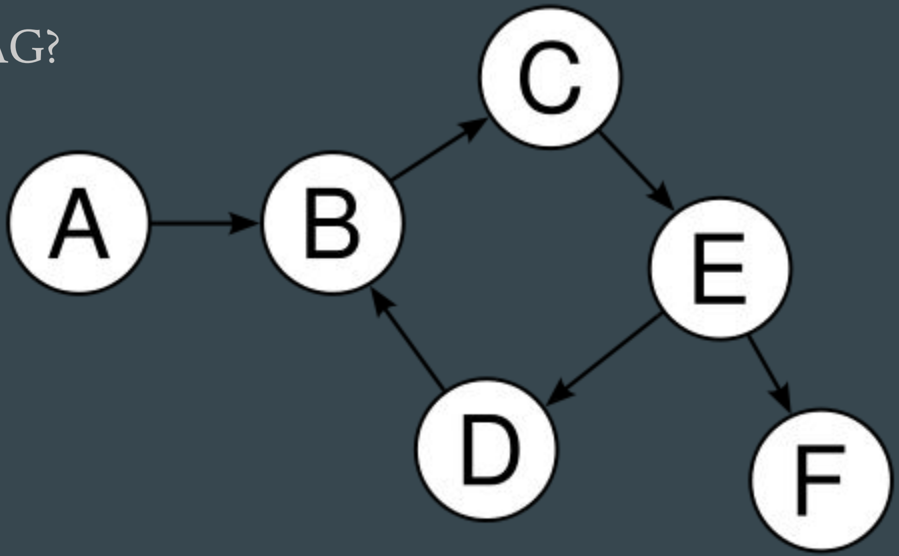
Example:



I pulled this image from https://www.quora.com/What-is-a-DAG-Directed-Acyclic-Graph Please do not sue me.

Definitions

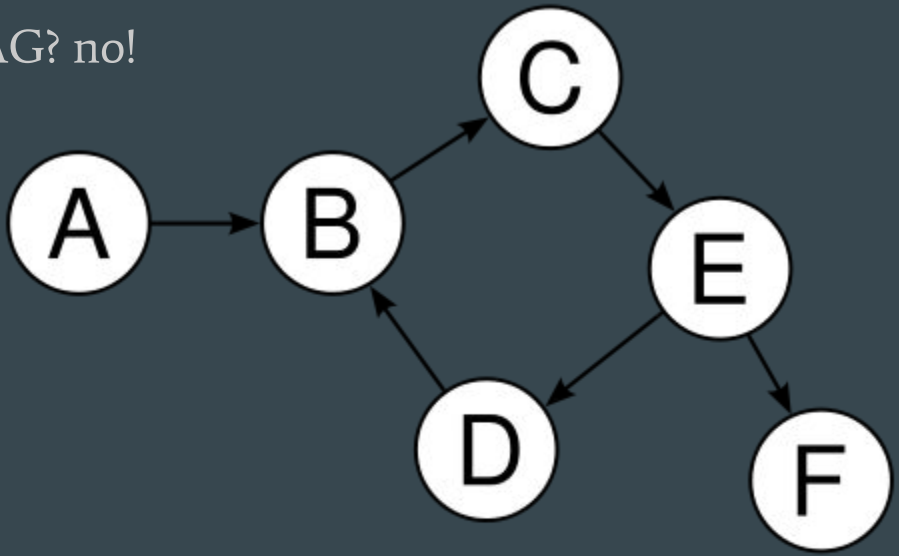
Example: Is this a DAG?



This is from https://commons.wikimedia.org/wiki/File:Directed_graph,_cyclic.svg
Please do not sue me.

Definitions

Example: Is this a DAG? no!



This is from https://commons.wikimedia.org/wiki/File:Directed_graph,_cyclic.svg
Please do not sue me.

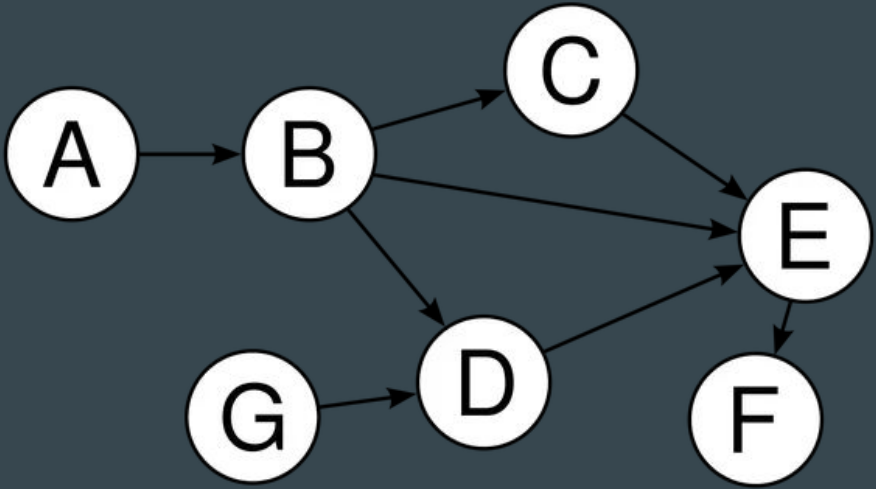
Definitions

A path in a graph is a list of vertices (v0,v1,...,vk) such that there is an edge from every vi to every vi+1.

Definitions

A path in a graph is a list of vertices (v0,v1,...,vk) such that there is an edge from every vi to every vi+1.

Is this a path? [A,B,E,F]

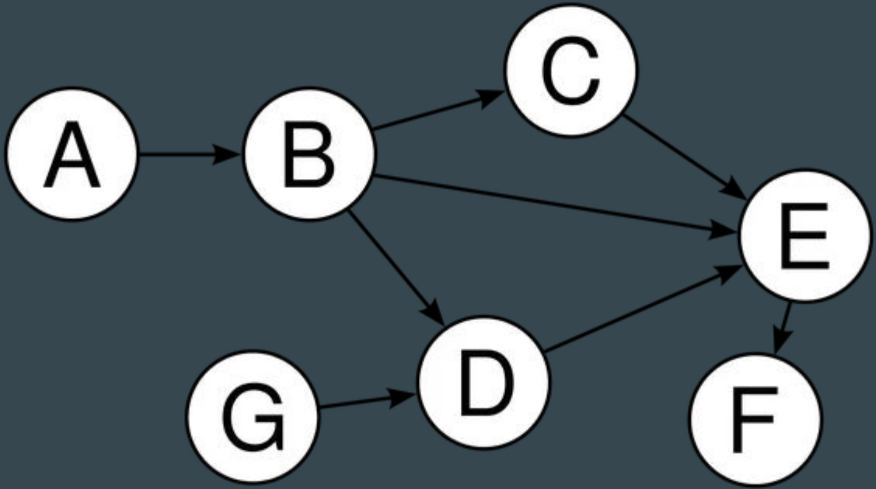


Definitions

A path in a graph is a list of vertices (v0,v1,...,vk) such that there is an edge from every vi to every vi+1.

Is this a path? [A,B,E,F] yes!

Is this a path? [A,B,D,E,F]



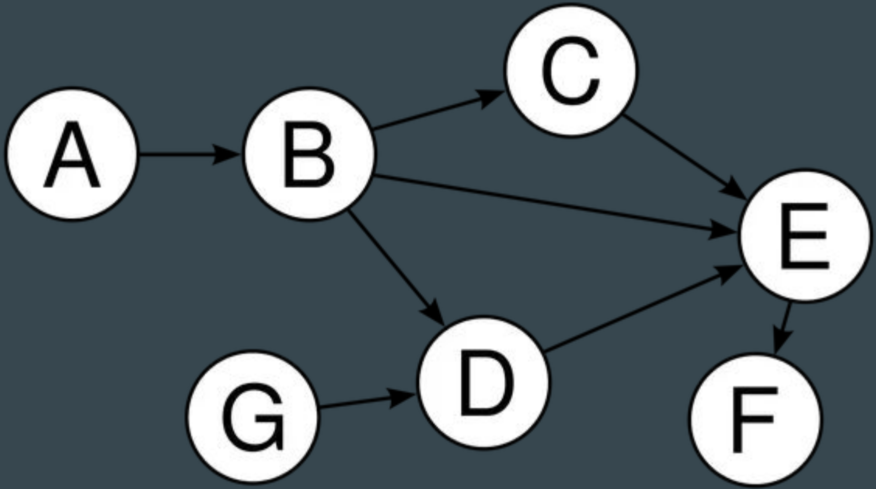
Definitions

A path in a graph is a list of vertices (v0,v1,...,vk) such that there is an edge from every vi to every vi+1.

Is this a path? [A,B,E,F] yes!

Is this a path? [A,B,D,E,F] yes!

Is this a path? [A,B,D,G]



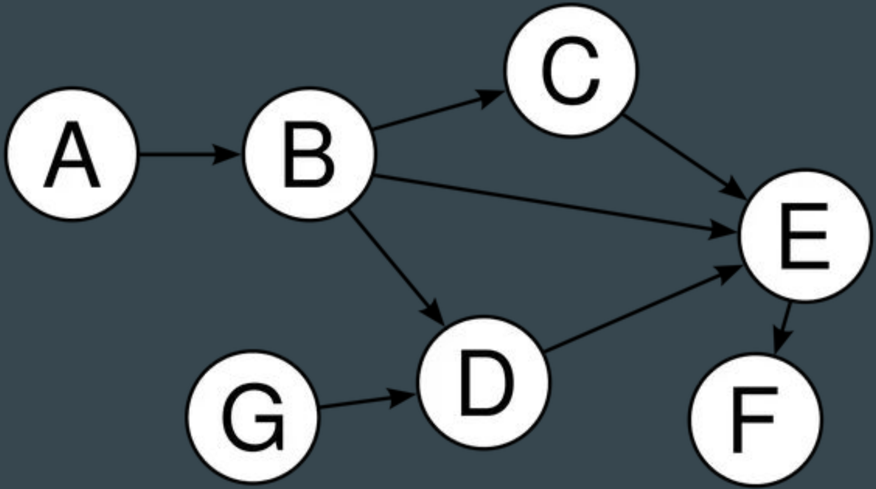
Definitions

A path in a graph is a list of vertices (v0,v1,...,vk) such that there is an edge from every vi to every vi+1.

Is this a path? [A,B,E,F] yes!

Is this a path? [A,B,D,E,F] yes!

Is this a path? [A,B,D,G] no!



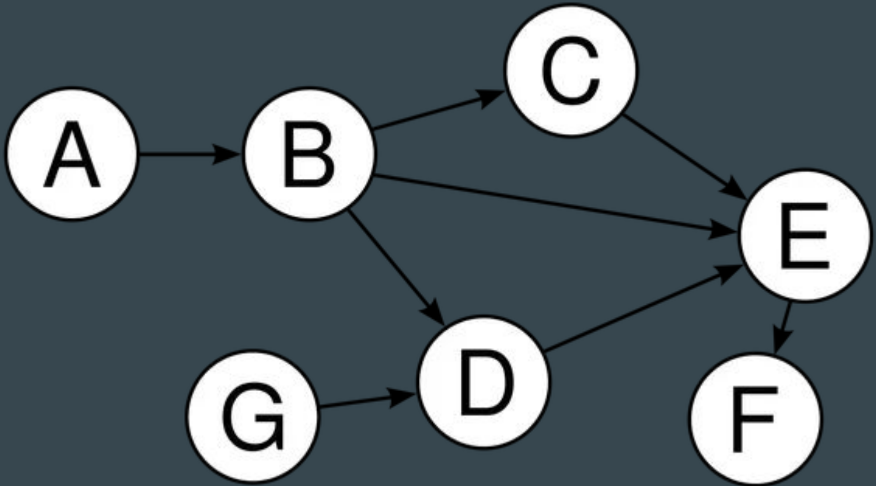
Definitions

Two paths are different if and only if the list of vertices are different.

Definitions

Two paths are different if and only if the list of vertices are different.

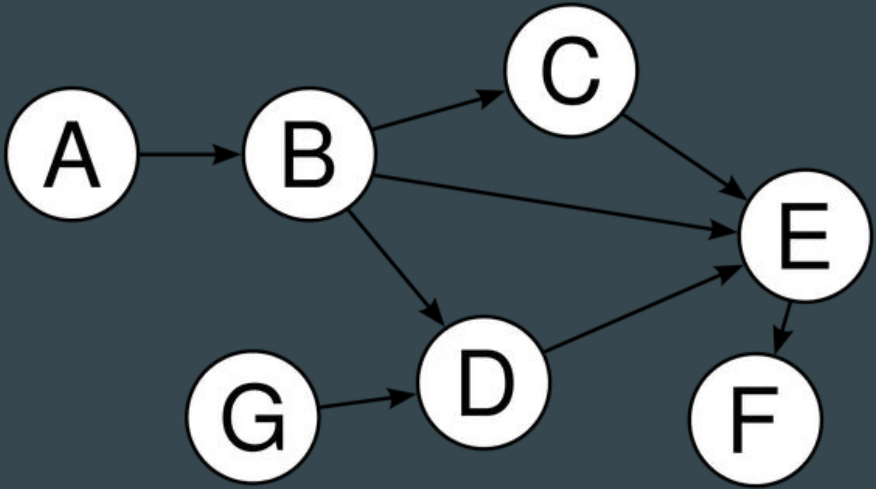
Are these path the same? [A,B,D,E] vs [A,B,C,E]



Definitions

Two paths are different if and only if the list of vertices are different.

Are these path the same? [A,B,D,E] vs [A,B,C,E] no!



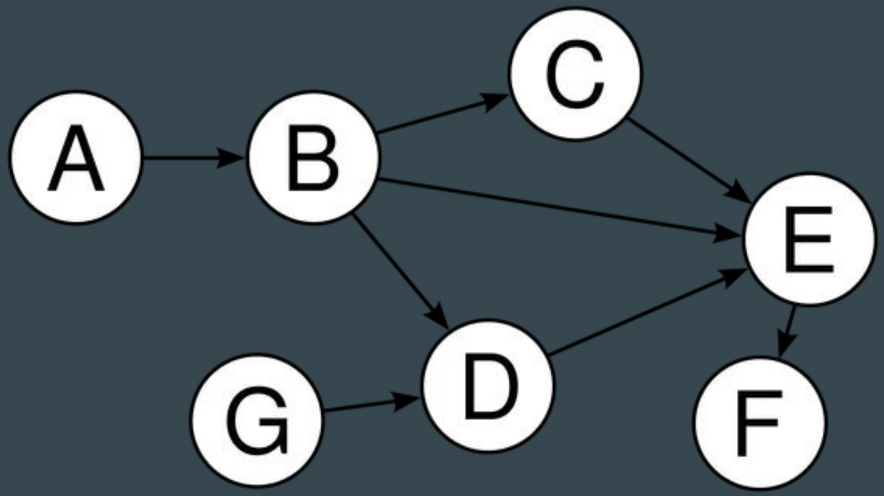
Problem Definition

Given a directed acyclic graph, a start vertex and an end vertex, how many unique paths are there from start to end?

Problem Definition

Given a directed acyclic graph, a start vertex and an end vertex, how many unique paths are there from start to end?

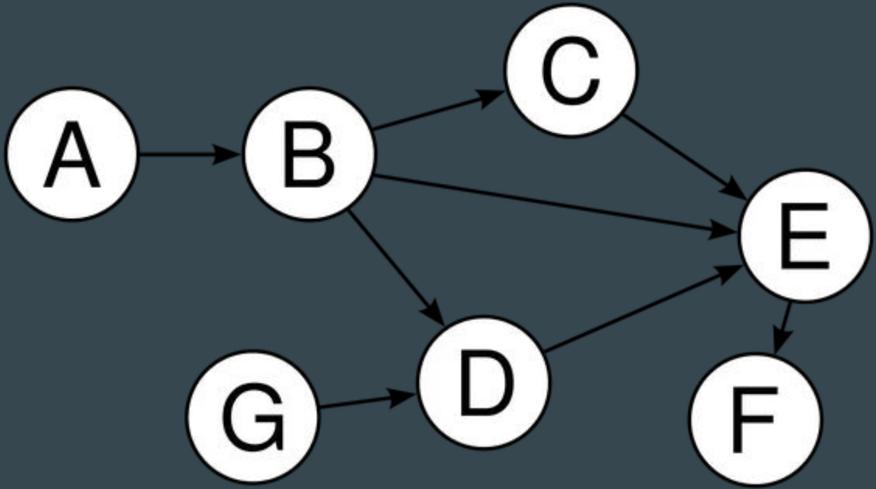
How many unique paths are there starting at A and ending at E?



Problem Definition

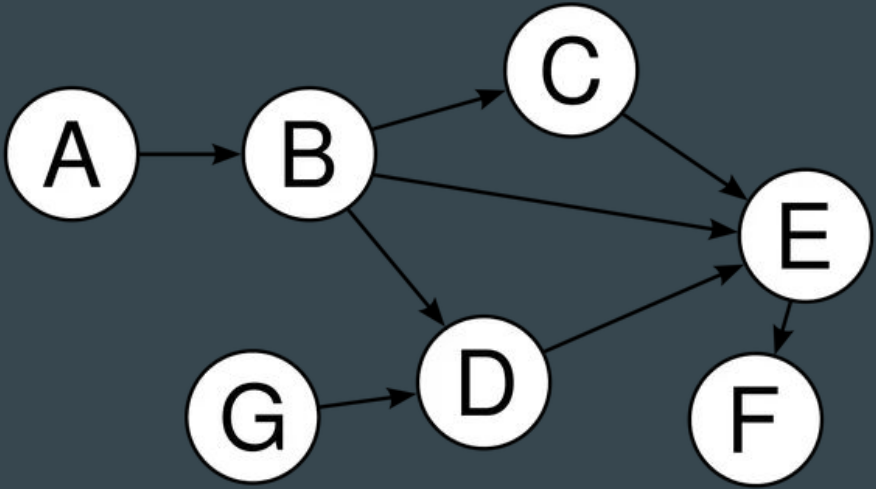
Given a directed acyclic graph, a start vertex and an end vertex, how many unique paths are there from start to end?

How many unique paths are there starting at A and ending at E? 3!



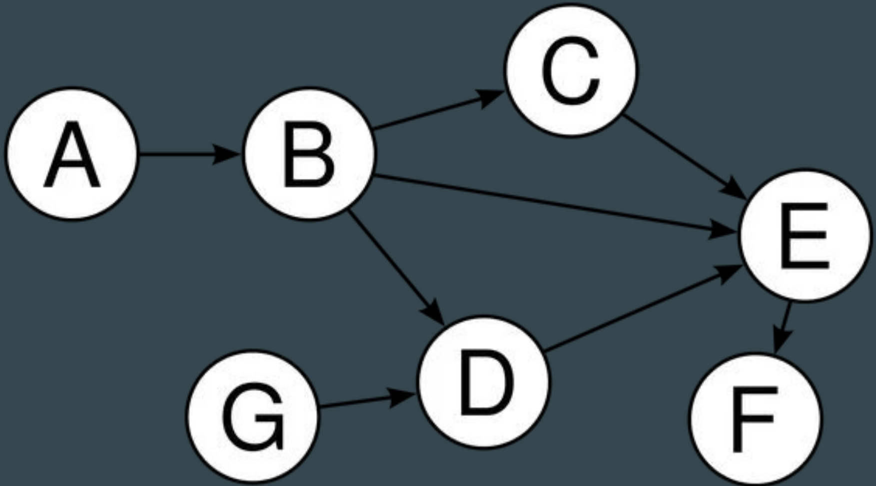
Problem Definition

How many unique paths are there starting at G and ending at F?



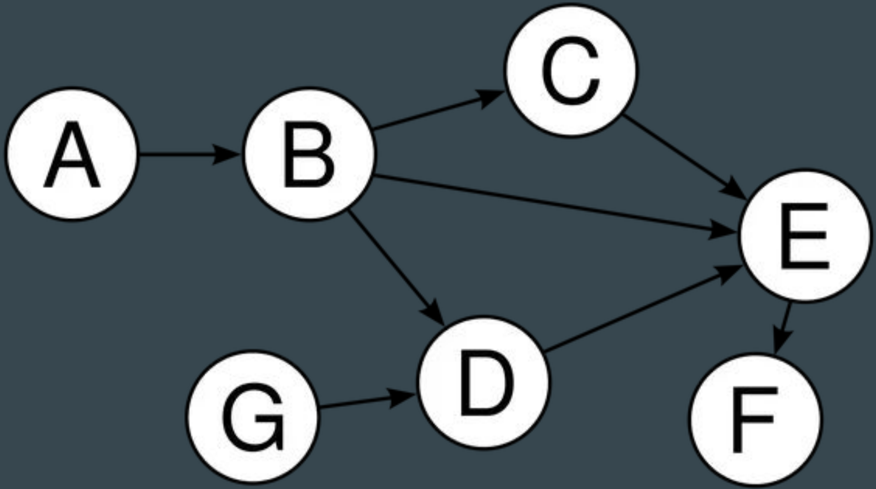
Problem Definition

How many unique paths are there starting at G and ending at F? 1!



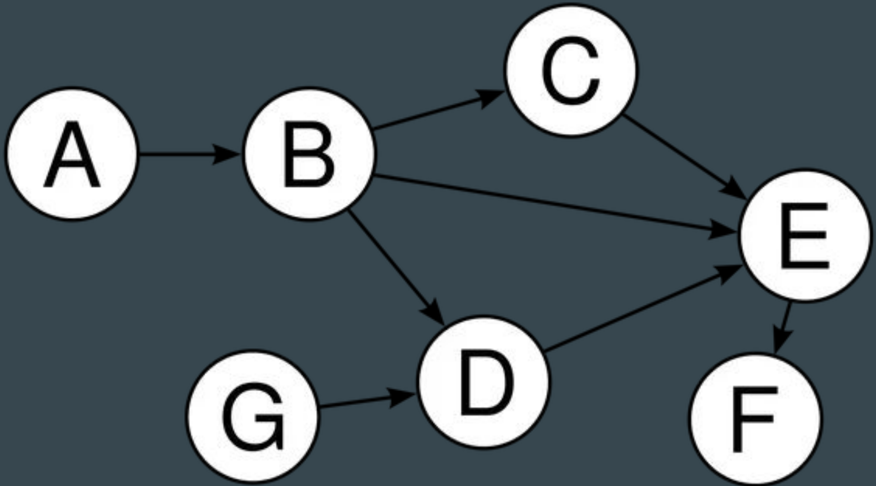
Problem Definition

How many unique paths are there starting at G and ending at B?



Problem Definition

How many unique paths are there starting at G and ending at B? 0!



Solution?

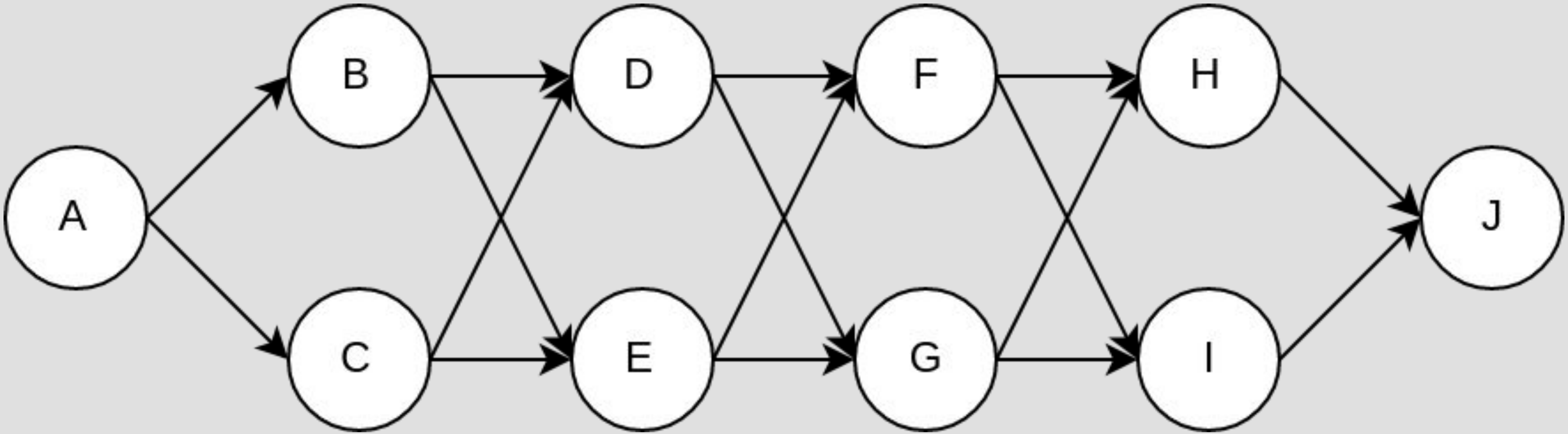
- Brute force?
- Try every path?

Complexity?

- How many paths are there in total?

Complexity?

- Can easily construct a case where there are O(2^V) paths!

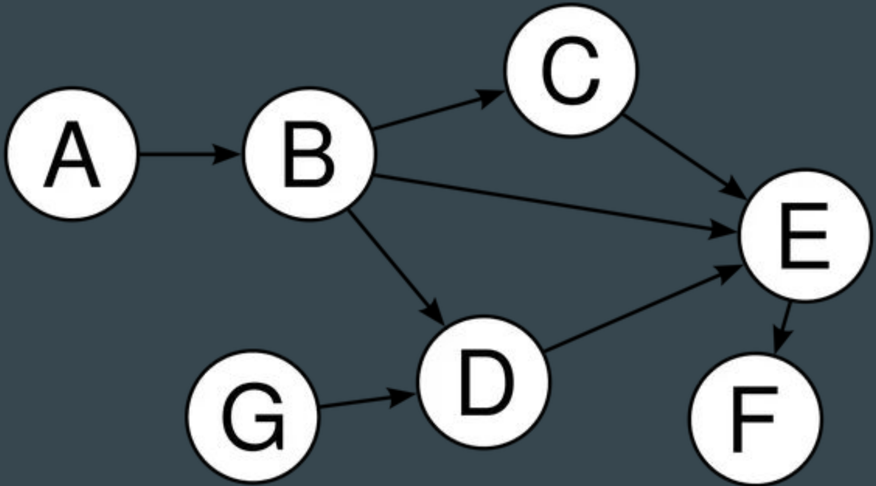

A
B
C
D
E
F
G
H
I
J

O(2^V) in the worst case?

Can we do better?

Recurrence Relation

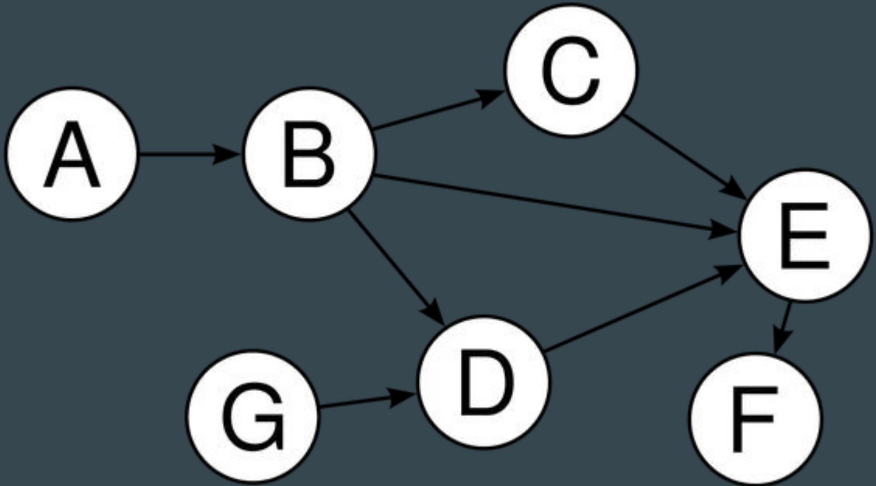
How many unique paths are there starting at A and ending at F?



Recurrence Relation

How many unique paths are there starting at A and ending at F?

Suppose Modan, the ultimate Recurrence Master, tells you, he's 100% certain there are 3 unique paths from A to E. Does this help you solve the problem?

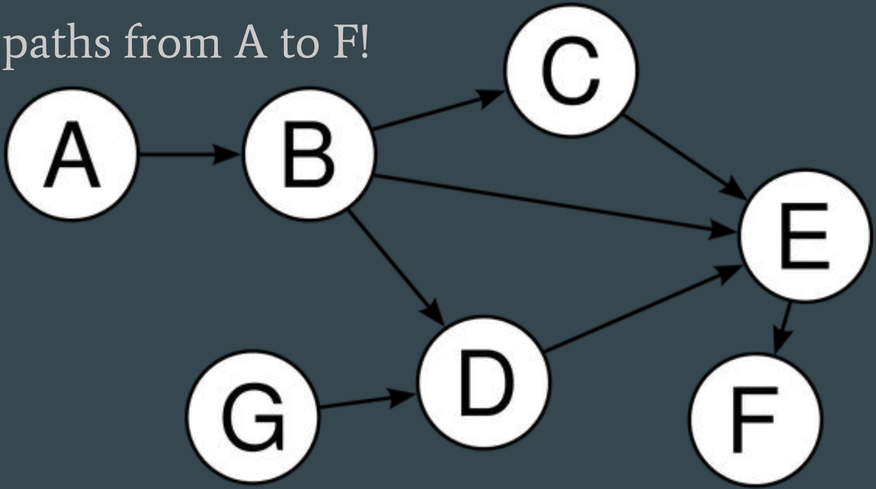


Recurrence Relation

How many unique paths are there starting at A and ending at F?

Suppose Modan, the ultimate Recurrence Master, tells you, he’s 100% certain there are 3 unique paths from A to E. Does this help you solve the problem?

It is now easy to conclude there are 3 unique paths from A to F!



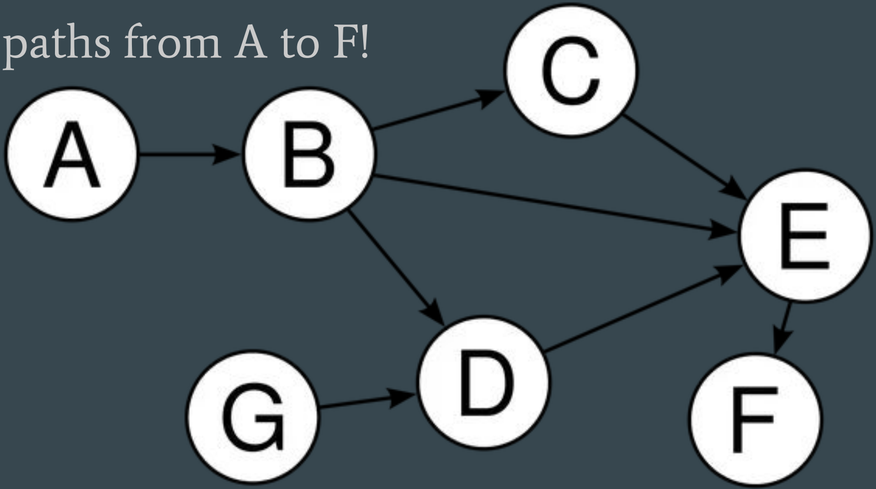
Recurrence Relation

How many unique paths are there starting at A and ending at F?

Suppose Modan, the ultimate Recurrence Master, tells you, he's 100% certain there are 3 unique paths from A to E. Does this help you solve the problem?

It is now easy to conclude there are 3 unique paths from A to F!

Why?



Recurrence Relation

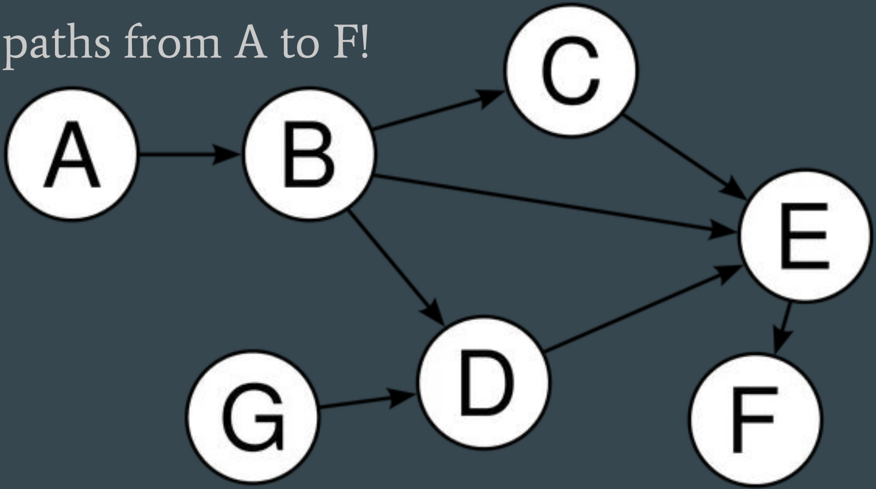
How many unique paths are there starting at A and ending at F?

Suppose Modan, the ultimate Recurrence Master, tells you, he's 100% certain there are 3 unique paths from A to E. Does this help you solve the problem?

It is now easy to conclude there are 3 unique paths from A to F!

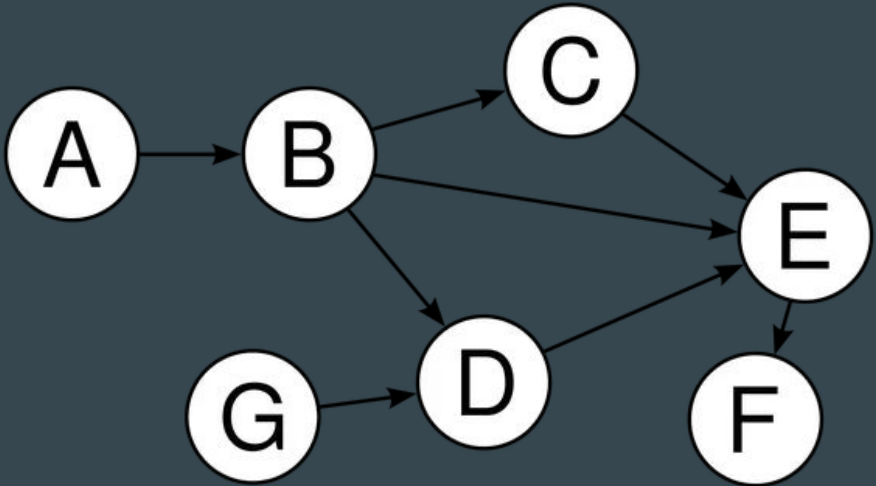
Why?

The only way to get to F is from E.



Recurrence Relation

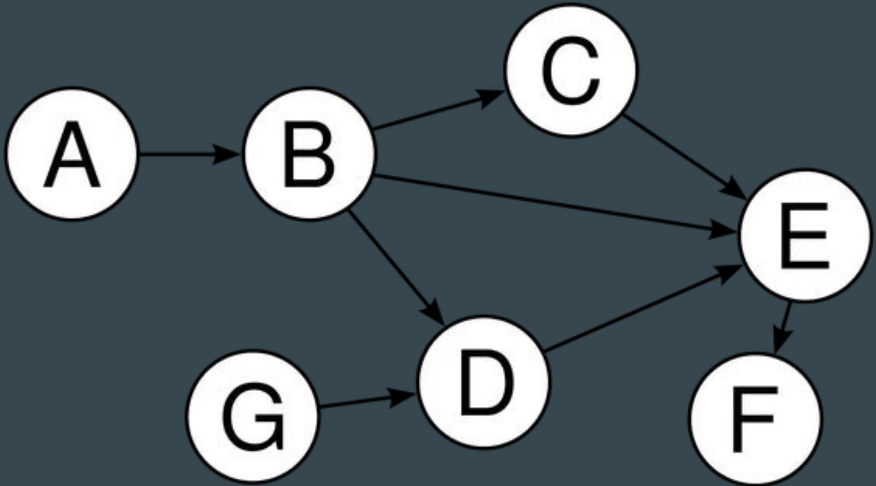
Let f(X) be a function where X is a vertex. f(X) = Y if and only if there are Y unique paths from S to X. Compute f(T) to get the answer!



Recurrence Relation

Let f(X) be a function where X is a vertex. f(X) = Y if and only if there are Y unique paths from S to X. Compute f(T) to get the answer!

We just concluded f(F) = f(E).

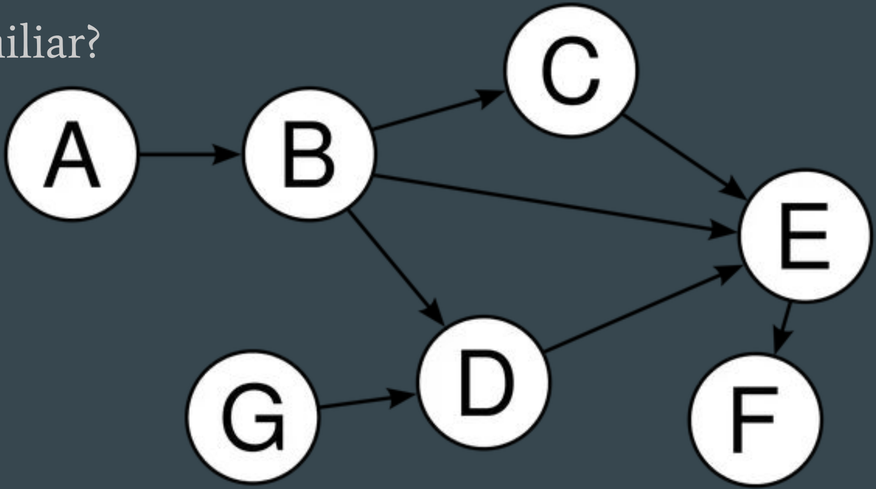


Recurrence Relation

Let f(X) be a function where X is a vertex. f(X) = Y if and only if there are Y unique paths from S to X. Compute f(T) to get the answer!

We just concluded f(F) = f(E).

Does this sort of function definition look familiar?



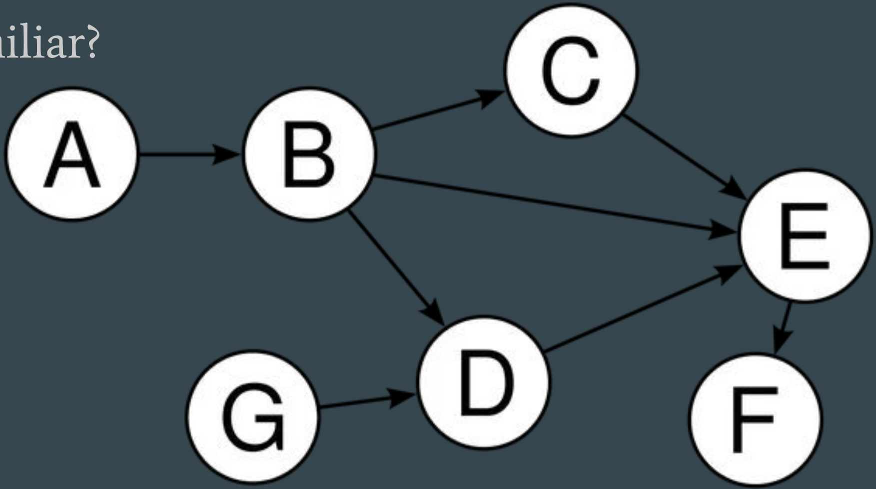
Recurrence Relation

Let f(X) be a function where X is a vertex. f(X) = Y if and only if there are Y unique paths from S to X. Compute f(T) to get the answer!

We just concluded f(F) = f(E).

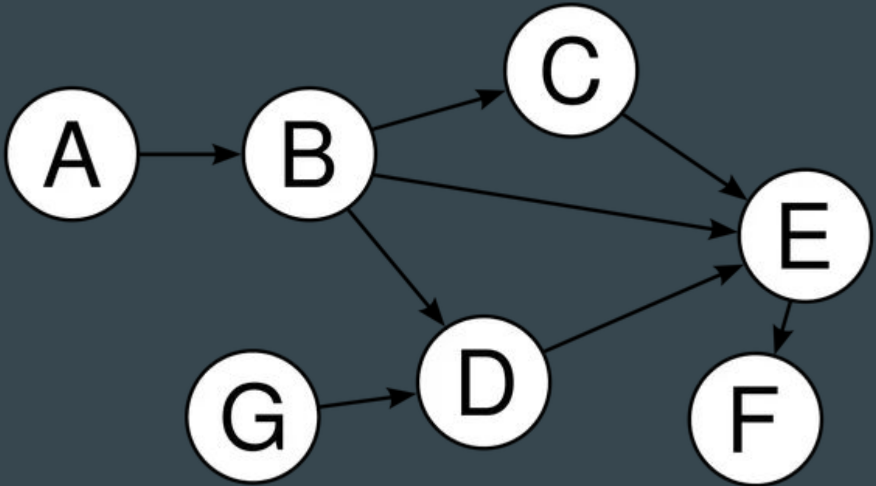
Does this sort of function definition look familiar?

RECURSION!



Recurrence Relation

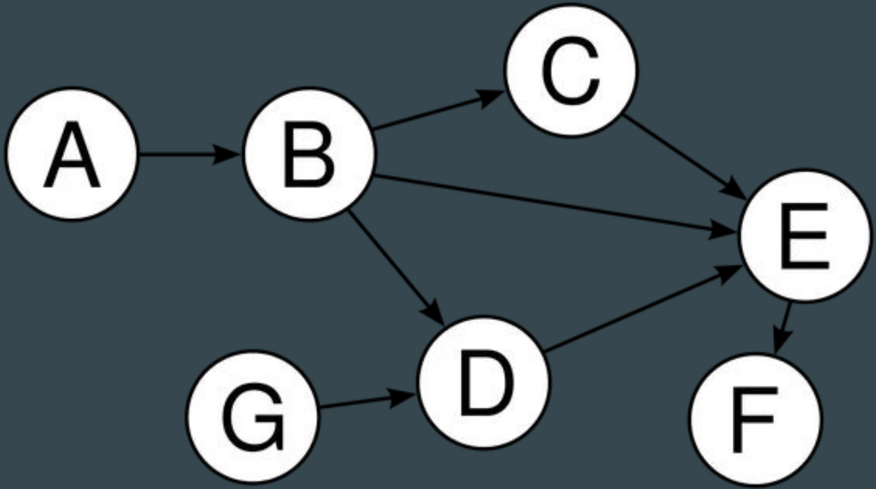
Now all we have to do if find out what f(E) is.



Recurrence Relation

Now all we have to do if find out what f(E) is.

What is f(E)?

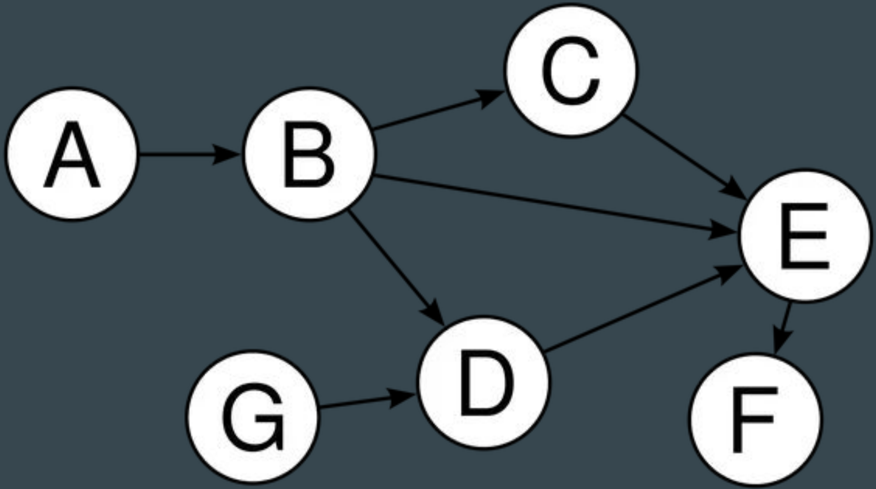


Recurrence Relation

Now all we have to do if find out what f(E) is.

What is f(E)?

f(E) = f(B) + f(C) + f(D)



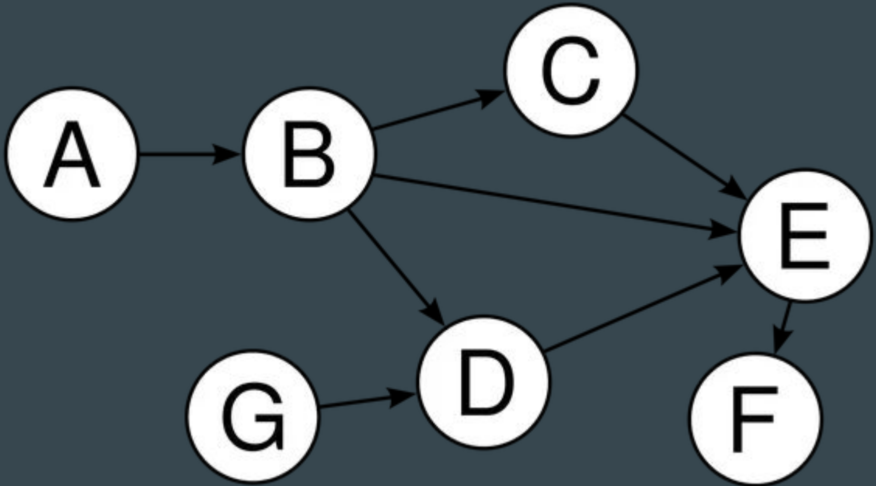
Recurrence Relation

Now all we have to do if find out what f(E) is.

What is f(E)?

f(E) = f(B) + f(C) + f(D)

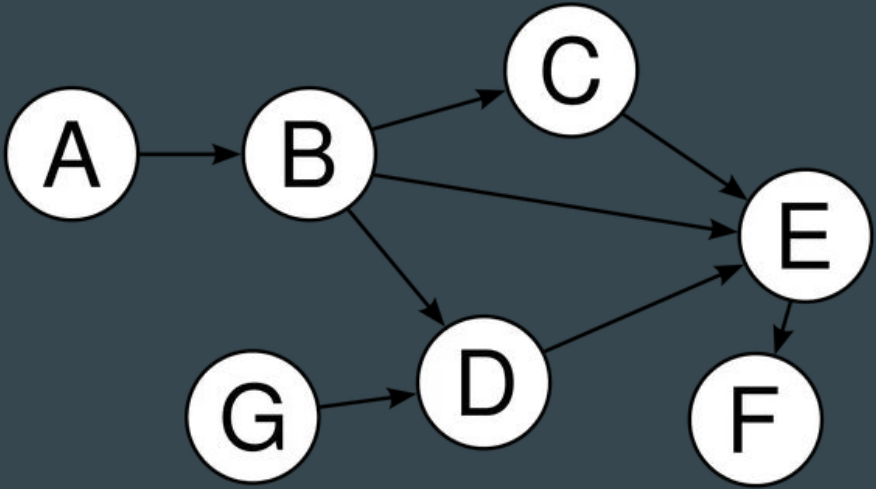
Wow! Not complicated at all!



Recurrence Relation

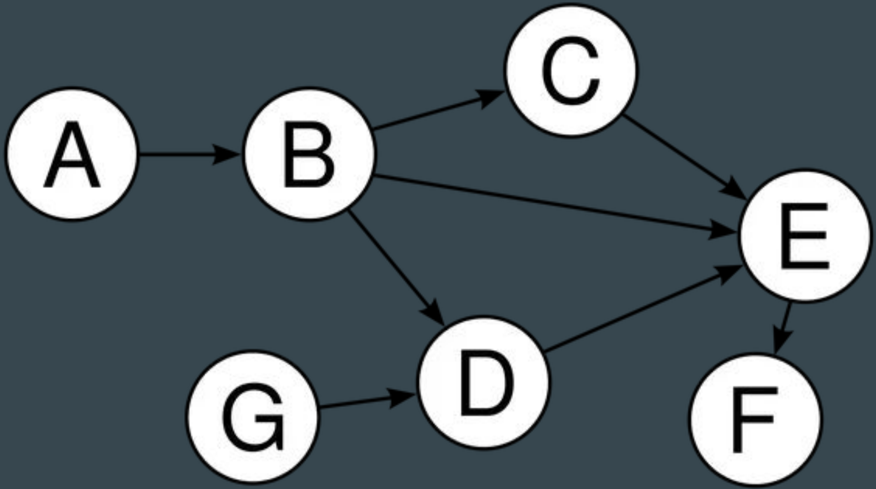
Now all we have to do if find out what f(B), f(C), f(D) are...

No more gimmicks! Time to work out a solution for a general case!



Recurrence Relation

Given a vertex X, what is f(X)?



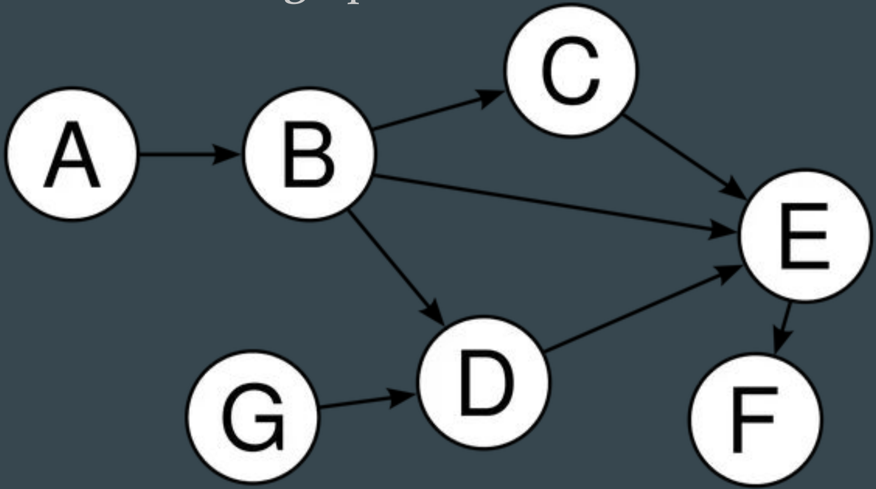
Recurrence Relation

Given a vertex X, what is f(X)?

f(X) = f(Y0) + f(Y1) + ... + f(Yk),

Where there edges (Y0, X), (Y1, X), ..., (Yk, X) exists in the graph.

i.e. X is Y0, Y1, ...,Yk's neighbour.



Recurrence Relation

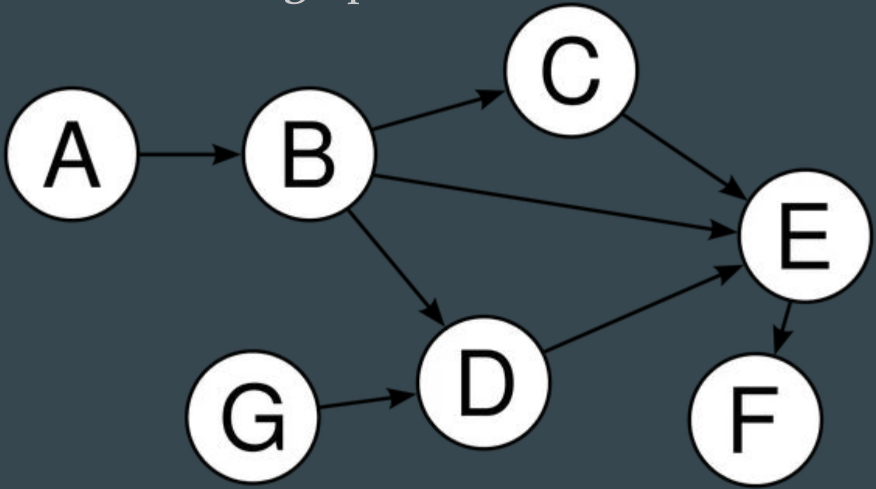
Given a vertex X, what is f(X)?

f(X) = f(Y0) + f(Y1) + ... + f(Yk),

Where there edges (Y0, X), (Y1, X), ..., (Yk, X) exists in the graph.

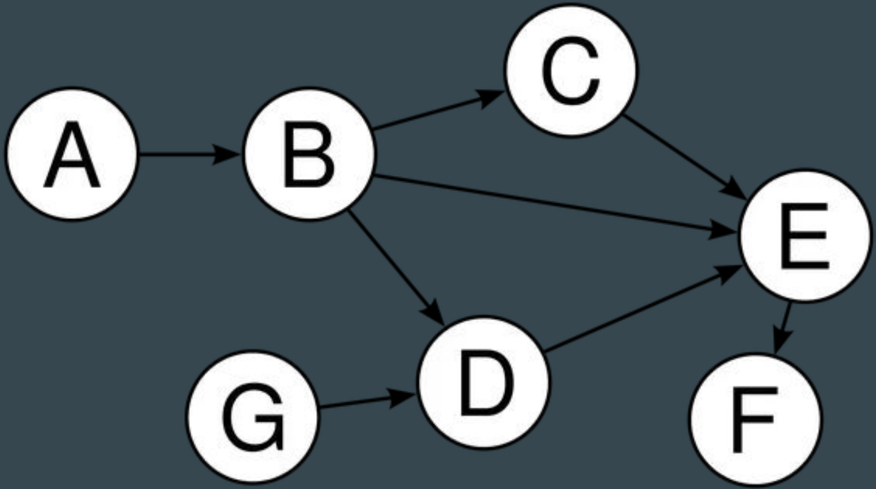
i.e. X is Y0, Y1, ...,Yk's neighbour.

Wow! So easy!



Recurrence Relation

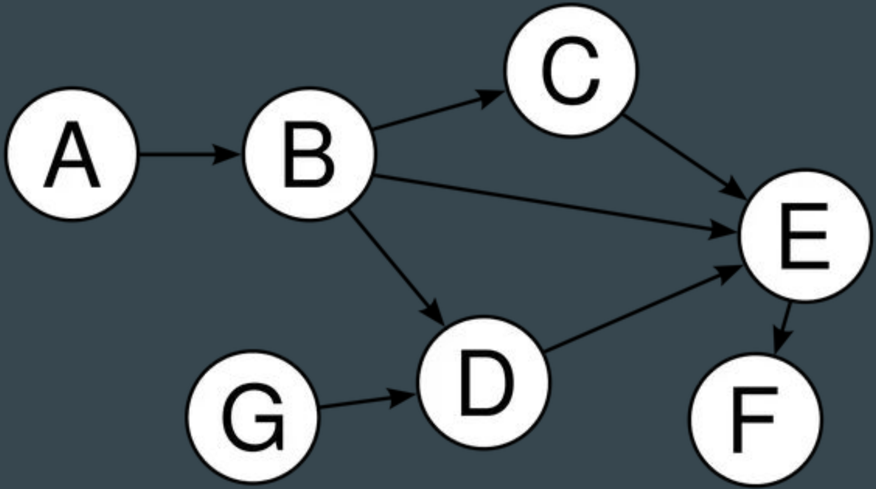
We defined A to be our starting vertex S. What if we want to compute f(A) = f(S)?



Recurrence Relation

We defined A to be our starting vertex S. What if we want to compute f(A) = f(S)?

Why do we want to compute this???

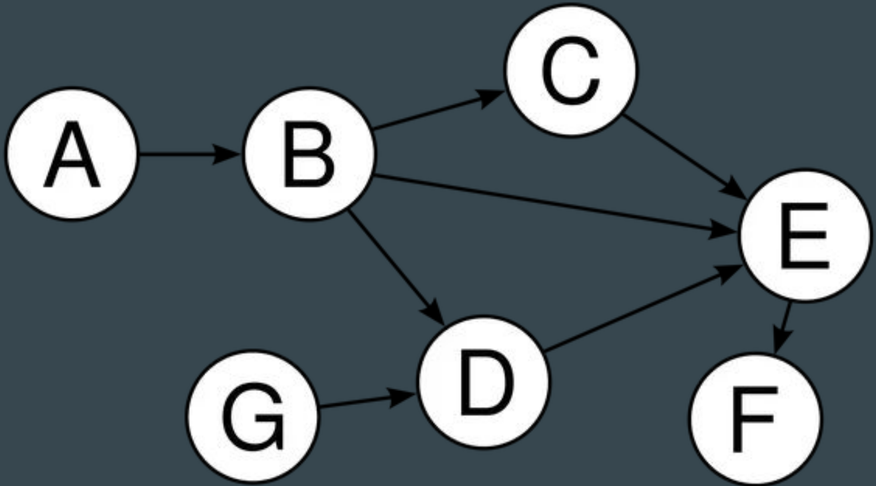


Recurrence Relation

We defined A to be our starting vertex S. What if we want to compute f(A) = f(S)?

Why do we want to compute this???

Because f(B) = f(A), we will need it at some point.



Recurrence Relation

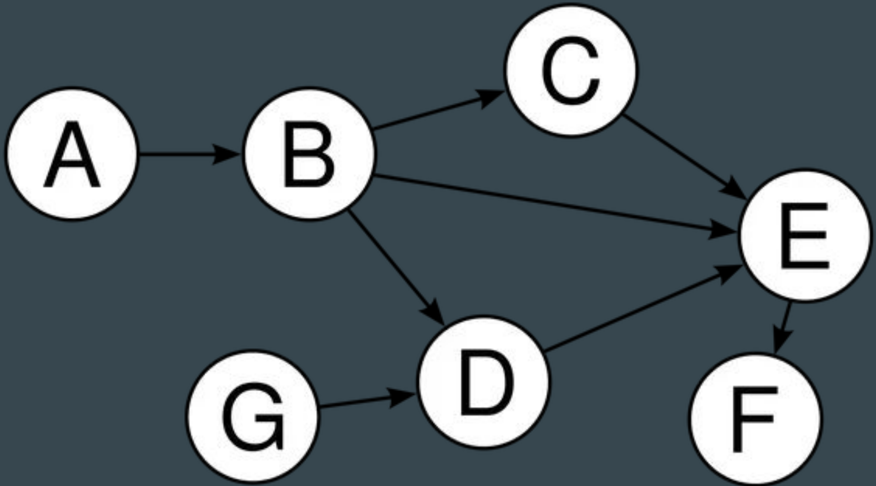
We defined A to be our starting vertex S. What if we want to compute f(A) = f(S)?

Why do we want to compute this???

Because f(B) = f(A), we will need it at some point.

f(A) = f(S) = 1.

There is one path from A to A, namely, [A]!



Recurrence Relation

We defined A to be our starting vertex S. What if we want to compute f(A) = f(S)?

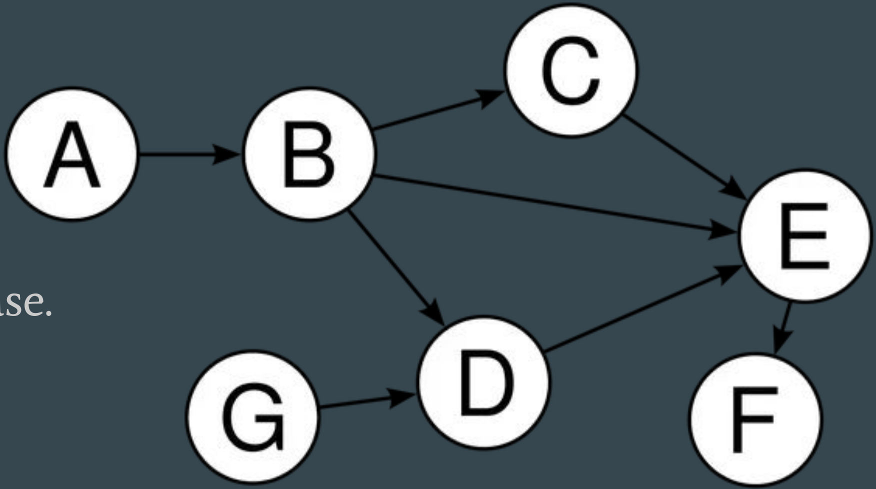
Why do we want to compute this???

Because f(B) = f(A), we will need it at some point.

f(A) = f(S) = 1.

There is one path from A to A, namely, [A]!

No recursion for this one! We call it a base case.



Problem solved!

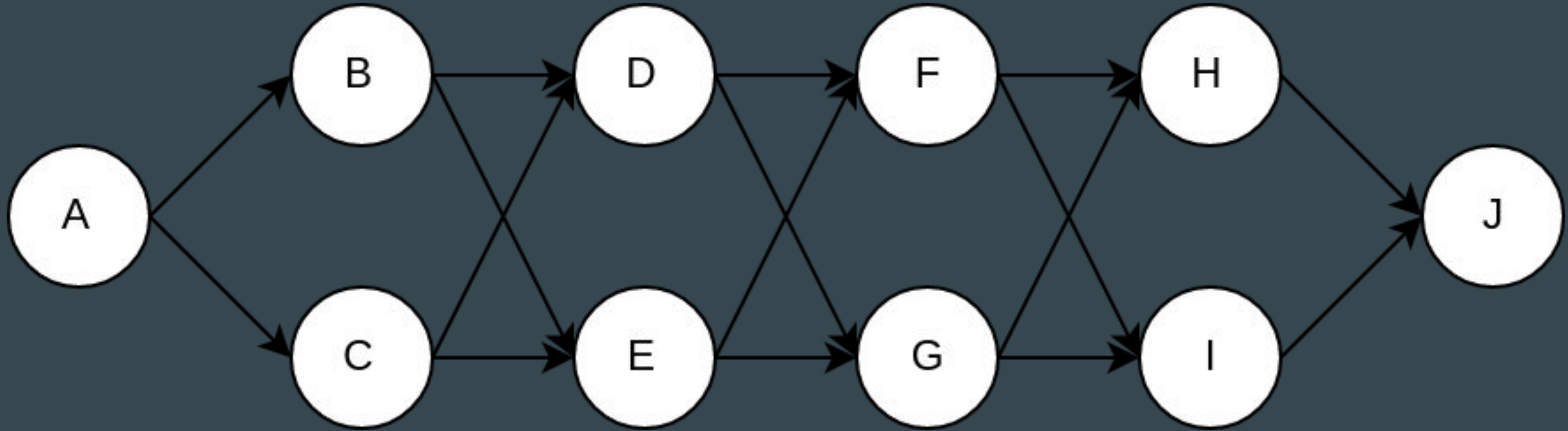
We did it!

Problem solved???

We did it???

Call stack

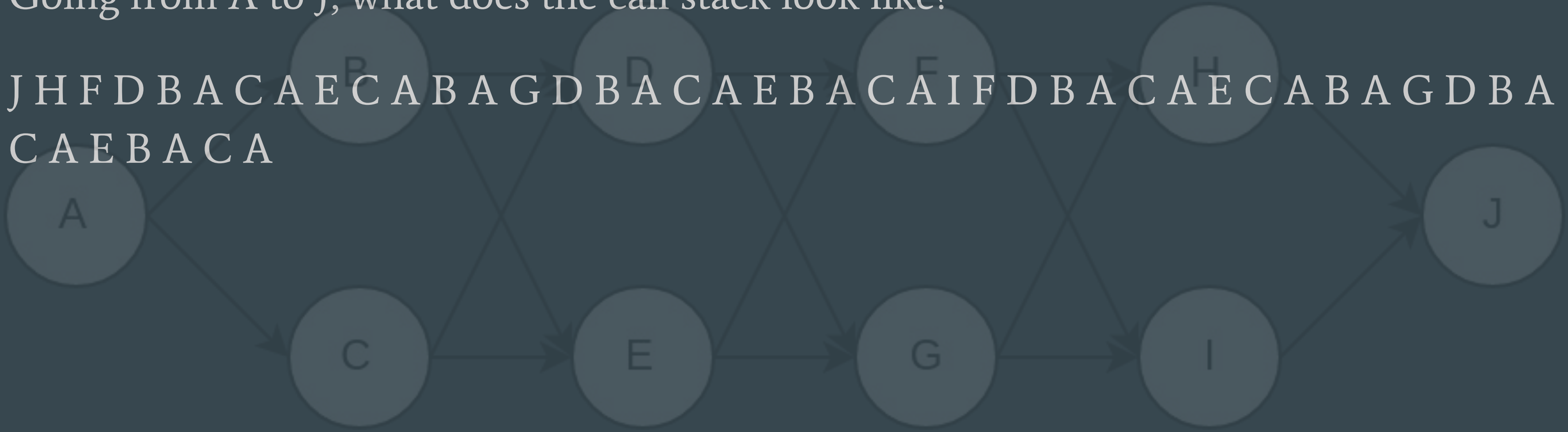
Going from A to J, what does the call stack look like?



Call stack

Going from A to J, what does the call stack look like?

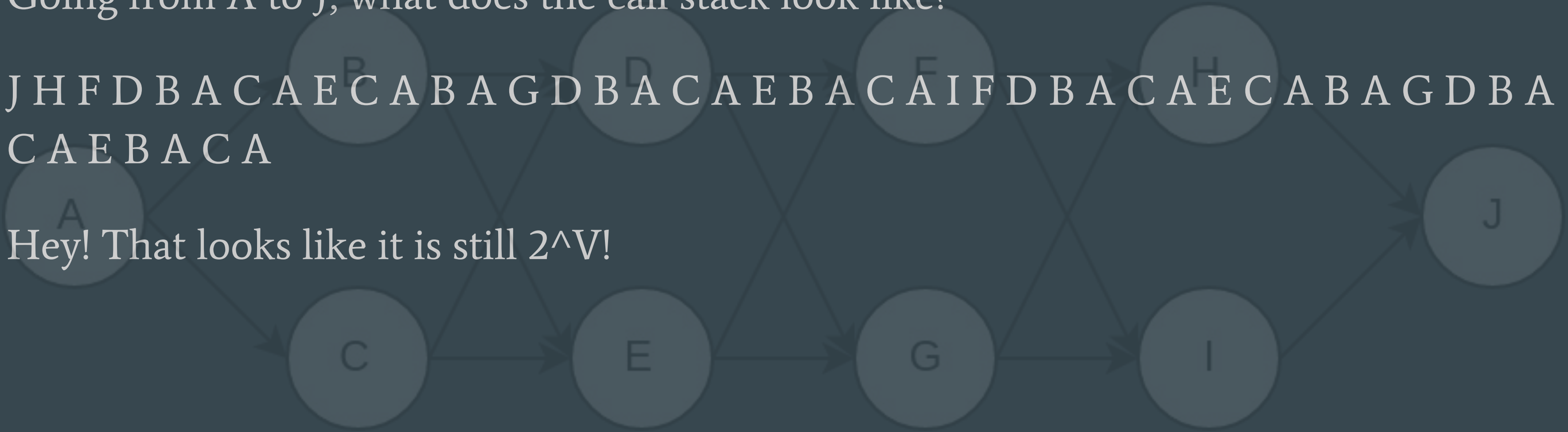
J H F D B A C A E C A B A G D B A C A E B A C A I F D B A C A E C A B A G D B A
C A E B A C A



Call stack

Going from A to J, what does the call stack look like?

J H F D B A C A E C A B A G D B A C A E B A C A I F D B A C A E C A B A G D B A C A E B A C A



Hey! That looks like it is still 2^V!

Overlapping Subproblem

What happened? Why are there still so many calls?

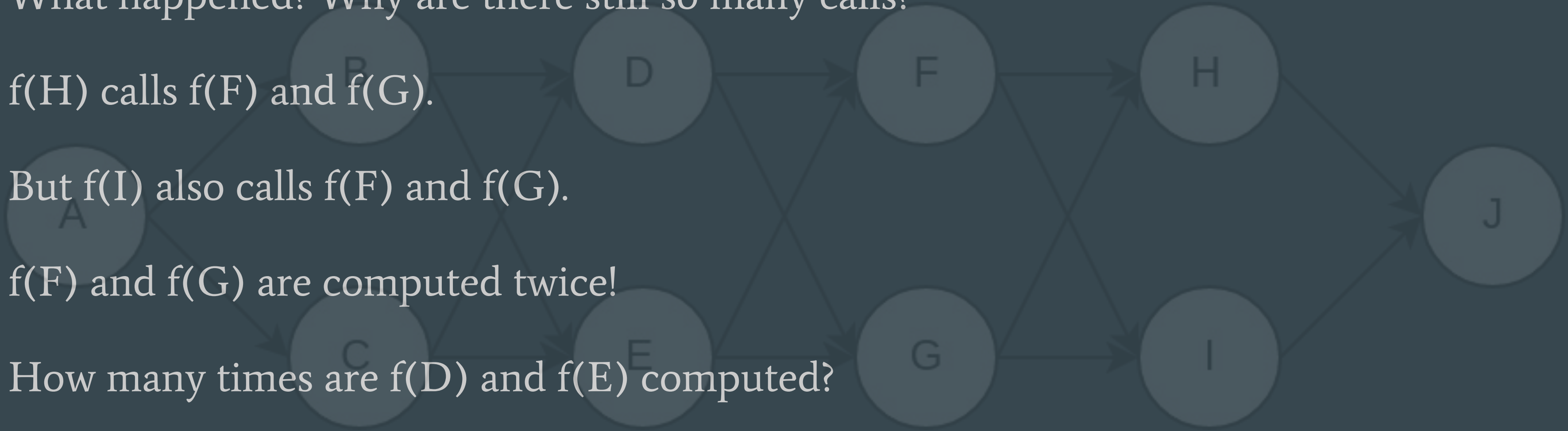
f(H) calls f(F) and f(G).

But f(I) also calls f(F) and f(G).

f(F) and f(G) are computed twice!

How many times are f(D) and f(E) computed?

4 times!



Overlapping Subproblem

How many times are f(B) and f(C) computed?

8 times!

How many times is f(A) called?

16 times!

Oh no!

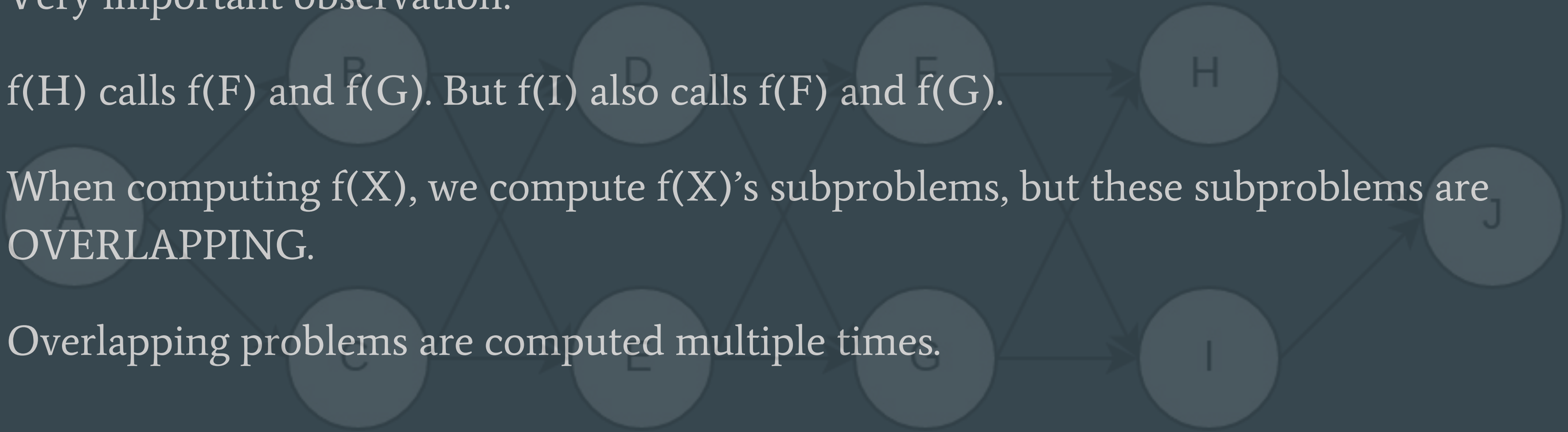
Overlapping Subproblem

Very important observation:

f(H) calls f(F) and f(G). But f(I) also calls f(F) and f(G).

When computing f(X), we compute f(X)'s subproblems, but these subproblems are OVERLAPPING.

Overlapping problems are computed multiple times.



Overlapping Subproblem

Solution?

Don’t compute the overlapping subproblems multiple times.

But how??

Memoization!

Memoization

Create a map M which maps from vertices to numbers, and it has the property

M[X] = f(X).

How does this help?

Suppose we want to compute f(X). Instead of computing the subproblems, check whether we've already solved the problem. If we have already solved it, use M[X]. Otherwise, recursively compute the subproblems, then write the solution to M!

Memoization

What is the complexity now?

O(E)! Wow!

Why did memoization turn an $O(2^V)$ algorithm into an $O(V^2)=O(E)$ algorithm?

Because each recursive call takes O(V) time while a lookup in M takes O(1) time.

Dynamic Programming!

Dynamic Programming

No formal definition...

As formal as it gets: has recurrence relation, repeated subproblems and an optimal substructure (whatever that means).

Dynamic Programming

Sounds very difficult at first. Becomes easy once recurrence relation is found.

i.e. solving this problem is hard, but suppose the value of some subproblems are given for free, it becomes easy to solve the problem.

i.e. solving this problem recursively is kind of easy, but the recursion may end up making the exact same called multiple times.

i.e. the solution to the problem depends on the solution to the same problem except with smaller data (subproblem).

...... Many ways to understand DP!

Dynamic Programming

DAG Paths is one of the most classic DP problems, in fact, every DP problem can be reduced to DAG Paths on an implicit graph!

Other classic problems: Longest Increasing Subsequence, Longest Common Subsequence, Knapsack, Subset Sum, Minimum String Distance, Coin Change……

City Destruction on Kattis is also a classic (not really, it's written by Modan, he'll be very happy if you give it a try)!

Dynamic Programming

Lots of advanced DP techniques such as

Multidimensional DP,

DP with trees,

DP with bitmask (very fun topic) (solves the famous Travelling Salesman Problem)!

Dynamic Programming

Recurrence is hard to come up with and the concept is hard to grasp at first.

Once recurrence is found the code usually turns out to be very short and clean!

(DP with bitmask tends to have even shorter codes!)

Dynamic Programming

There are 2 approaches to DP.

Top down uses recursion and memoization.

Bottom up (CP3 book described this as the “true form” of DP) starts with base cases and builds up.

That’s it! Thanks!