Problem 3530: Maximum Profit from Valid Topological Order in DAG

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a

Directed Acyclic Graph (DAG)

with

n

nodes labeled from

0

to

n - 1

, represented by a 2D array

edges

, where

edges[i] = [u

i

, v

i

]

indicates a directed edge from node

u

i

to

v

i

. Each node has an associated

score

given in an array

score

, where

score[i]

represents the score of node

i

.

You must process the nodes in a

valid topological order

. Each node is assigned a

1-based position

in the processing order.

The

profit

is calculated by summing up the product of each node's score and its position in the ordering.

Return the

maximum

possible profit achievable with an optimal topological order.

A

topological order

of a DAG is a linear ordering of its nodes such that for every directed edge

$u \rightarrow v$

, node

u

comes before

v

in the ordering.

Example 1:

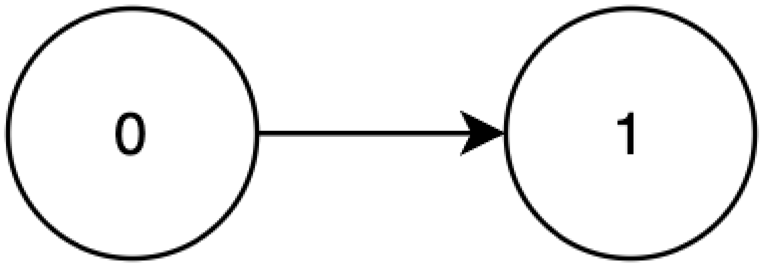
Input:

n = 2, edges = [[0,1]], score = [2,3]

Output:

8

Explanation:



Node 1 depends on node 0, so a valid order is

[0, 1]

.

Node

Processing Order

Score

Multiplier

Profit Calculation

0

1st

2

1

2 × 1 = 2

1

2nd

3

2

3 × 2 = 6

The maximum total profit achievable over all valid topological orders is

2 + 6 = 8

.

Example 2:

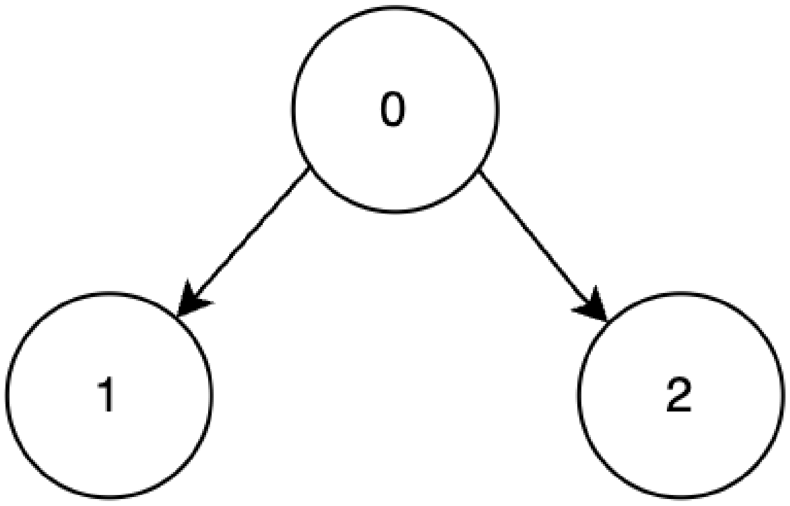
Input:

n = 3, edges = [[0,1],[0,2]], score = [1,6,3]

Output:

25

Explanation:



Nodes 1 and 2 depend on node 0, so the most optimal valid order is

[0, 2, 1]

.

Node

Processing Order

Score

Multiplier

Profit Calculation

0

1st

1

1

1 × 1 = 1

2

2nd

3

2

3 × 2 = 6

1

3rd

6

3

6 × 3 = 18

The maximum total profit achievable over all valid topological orders is

1 + 6 + 18 = 25

.

Constraints:

1 <= n == score.length <= 22

1 <= score[i] <= 10

5

0 <= edges.length <= n * (n - 1) / 2

edges[i] == [u

i

, v

i

]

denotes a directed edge from

u

i

to

v

i

.

0 <= u

i

, v

i

< n

u

i

!= v

i

The input graph is

guaranteed

to be a

DAG

.

There are no duplicate edges.

Code Snippets

C++:

```
class Solution {
public:
int maxProfit(int n, vector<vector<int>>& edges, vector<int>& score) {

}
};
```

Java:

```
class Solution {
public int maxProfit(int n, int[][] edges, int[] score) {

}
}
```

Python3:

```
class Solution:
def maxProfit(self, n: int, edges: List[List[int]], score: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxProfit(self, n, edges, score):
```

```
"""
:type n: int
:type edges: List[List[int]]
:type score: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number[]} score
* @return {number}
*/
var maxProfit = function(n, edges, score) {

};
```

TypeScript:

```
function maxProfit(n: number, edges: number[][], score: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxProfit(int n, int[][] edges, int[] score) {

}
}
```

C:

```
int maxProfit(int n, int** edges, int edgesSize, int* edgesColSize, int*
score, int scoreSize) {

}
```

Go:

```
func maxProfit(n int, edges [][]int, score []int) int {

}
```

Kotlin:

```
class Solution {
fun maxProfit(n: Int, edges: Array<IntArray>, score: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxProfit(_ n: Int, _ edges: [[Int]], _ score: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_profit(n: i32, edges: Vec<Vec<i32>>, score: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer[]} score
# @return {Integer}
def max_profit(n, edges, score)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer[] $score
* @return Integer
*/
function maxProfit($n, $edges, $score) {

}
}
```

Dart:

```
class Solution {
int maxProfit(int n, List<List<int>> edges, List<int> score) {

}
}
```

Scala:

```
object Solution {
def maxProfit(n: Int, edges: Array[Array[Int]], score: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_profit(n :: integer, edges :: [[integer]], score :: [integer]) ::
integer
def max_profit(n, edges, score) do

end
end
```

Erlang:

```
-spec max_profit(N :: integer(), Edges :: [[integer()]], Score ::
[integer()]) -> integer().
max_profit(N, Edges, Score) ->
.
```

Racket:

```
(define/contract (max-profit n edges score)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Profit from Valid Topological Order in DAG
* Difficulty: Hard
* Tags: array, graph, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxProfit(int n, vector<vector<int>>& edges, vector<int>& score) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Profit from Valid Topological Order in DAG
* Difficulty: Hard
* Tags: array, graph, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
```

```
public int maxProfit(int n, int[][] edges, int[] score) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Profit from Valid Topological Order in DAG
Difficulty: Hard
Tags: array, graph, dp, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxProfit(self, n: int, edges: List[List[int]], score: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxProfit(self, n, edges, score):
"""
:type n: int
:type edges: List[List[int]]
:type score: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Profit from Valid Topological Order in DAG
* Difficulty: Hard
* Tags: array, graph, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @param {number[][]} edges
* @param {number[]} score
* @return {number}
*/
var maxProfit = function(n, edges, score) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Profit from Valid Topological Order in DAG
* Difficulty: Hard
* Tags: array, graph, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxProfit(n: number, edges: number[][], score: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Profit from Valid Topological Order in DAG
* Difficulty: Hard
* Tags: array, graph, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
public class Solution {
public int MaxProfit(int n, int[][] edges, int[] score) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Profit from Valid Topological Order in DAG
* Difficulty: Hard
* Tags: array, graph, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxProfit(int n, int** edges, int edgesSize, int* edgesColSize, int*
score, int scoreSize) {

}
```

Go Solution:

```
// Problem: Maximum Profit from Valid Topological Order in DAG
// Difficulty: Hard
// Tags: array, graph, dp, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxProfit(n int, edges [][]int, score []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxProfit(n: Int, edges: Array<IntArray>, score: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxProfit(_ n: Int, _ edges: [[Int]], _ score: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Profit from Valid Topological Order in DAG
// Difficulty: Hard
// Tags: array, graph, dp, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_profit(n: i32, edges: Vec<Vec<i32>>, score: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer[]} score
# @return {Integer}
def max_profit(n, edges, score)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer[] $score
* @return Integer
*/
function maxProfit($n, $edges, $score) {

}
}
```

Dart Solution:

```
class Solution {
int maxProfit(int n, List<List<int>> edges, List<int> score) {

}
}
```

Scala Solution:

```
object Solution {
def maxProfit(n: Int, edges: Array[Array[Int]], score: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_profit(n :: integer, edges :: [[integer]], score :: [integer]) ::
integer
def max_profit(n, edges, score) do

end
end
```

Erlang Solution:

```
-spec max_profit(N :: integer(), Edges :: [[integer()]], Score ::
[integer()]) -> integer().
max_profit(N, Edges, Score) ->
.
```

Racket Solution:

```
(define/contract (max-profit n edges score)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```