

Problem 2497: Maximum Star Sum of a Graph

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

There is an undirected graph consisting of

n

nodes numbered from

0

to

$n - 1$

. You are given a

0-indexed

integer array

vals

of length

n

where

`vals[i]`

denotes the value of the

`i`

th

node.

You are also given a 2D integer array

`edges`

where

`edges[i] = [a`

`i`

, b

`i`

]

denotes that there exists an

undirected

edge connecting nodes

`a`

`i`

and

`b`

i.

A

star graph

is a subgraph of the given graph having a center node containing

0

or more neighbors. In other words, it is a subset of edges of the given graph such that there exists a common node for all edges.

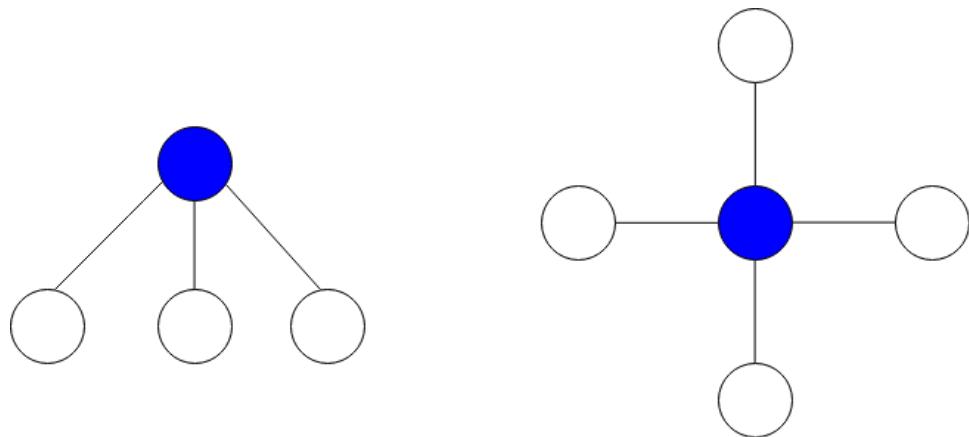
The image below shows star graphs with

3

and

4

neighbors respectively, centered at the blue node.



The

star sum

is the sum of the values of all the nodes present in the star graph.

Given an integer

k

, return

the

maximum star sum

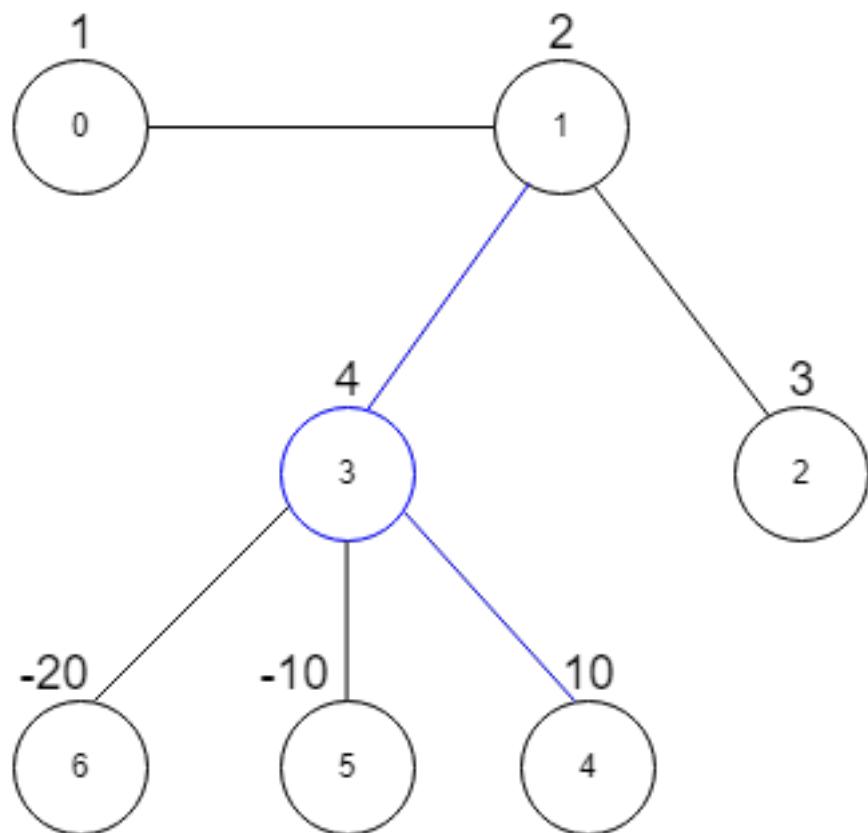
of a star graph containing

at most

k

edges.

Example 1:



Input:

vals = [1,2,3,4,10,-10,-20], edges = [[0,1],[1,2],[1,3],[3,4],[3,5],[3,6]], k = 2

Output:

16

Explanation:

The above diagram represents the input graph. The star graph with the maximum star sum is denoted by blue. It is centered at 3 and includes its neighbors 1 and 4. It can be shown it is not possible to get a star graph with a sum greater than 16.

Example 2:

Input:

vals = [-5], edges = [], k = 0

Output:

-5

Explanation:

There is only one possible star graph, which is node 0 itself. Hence, we return -5.

Constraints:

$n == vals.length$

$1 \leq n \leq 10$

5

-10

4

$\leq \text{vals}[i] \leq 10$

4

$0 \leq \text{edges.length} \leq \min(n * (n - 1) / 2$

, 10

5

)

$\text{edges}[i].length == 2$

$0 \leq a$

i

, b

i

$\leq n - 1$

a

i

$\neq b$

i

$0 \leq k \leq n - 1$

Code Snippets

C++:

```
class Solution {  
public:  
    int maxStarSum(vector<int>& vals, vector<vector<int>>& edges, int k) {  
  
    }  
};
```

Java:

```
class Solution {  
public int maxStarSum(int[] vals, int[][] edges, int k) {  
  
}  
}
```

Python3:

```
class Solution:  
    def maxStarSum(self, vals: List[int], edges: List[List[int]], k: int) -> int:
```

Python:

```
class Solution(object):  
    def maxStarSum(self, vals, edges, k):  
        """  
        :type vals: List[int]  
        :type edges: List[List[int]]  
        :type k: int  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number[]} vals  
 * @param {number[][]} edges  
 * @param {number} k  
 * @return {number}  
 */  
var maxStarSum = function(vals, edges, k) {
```

```
};
```

TypeScript:

```
function maxStarSum(vals: number[], edges: number[][][], k: number): number {  
}  
};
```

C#:

```
public class Solution {  
    public int MaxStarSum(int[] vals, int[][][] edges, int k) {  
        }  
    }  
}
```

C:

```
int maxStarSum(int* vals, int valsSize, int** edges, int edgesSize, int*  
edgesColSize, int k) {  
}  
}
```

Go:

```
func maxStarSum(vals []int, edges [][]int, k int) int {  
}  
}
```

Kotlin:

```
class Solution {  
    fun maxStarSum(vals: IntArray, edges: Array<IntArray>, k: Int): Int {  
        }  
    }  
}
```

Swift:

```
class Solution {  
    func maxStarSum(_ vals: [Int], _ edges: [[Int]], _ k: Int) -> Int {  
}
```

```
}
```

```
}
```

Rust:

```
impl Solution {
    pub fn max_star_sum(vals: Vec<i32>, edges: Vec<Vec<i32>>, k: i32) -> i32 {
        }
    }
}
```

Ruby:

```
# @param {Integer[]} vals
# @param {Integer[][]} edges
# @param {Integer} k
# @return {Integer}
def max_star_sum(vals, edges, k)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $vals
     * @param Integer[][] $edges
     * @param Integer $k
     * @return Integer
     */
    function maxStarSum($vals, $edges, $k) {

    }
}
```

Dart:

```
class Solution {
    int maxStarSum(List<int> vals, List<List<int>> edges, int k) {
    }
}
```

```
}
```

Scala:

```
object Solution {  
    def maxStarSum(vals: Array[Int], edges: Array[Array[Int]], k: Int): Int = {  
          
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec max_star_sum(vals :: [integer], edges :: [[integer]], k :: integer) ::  
    integer  
    def max_star_sum(vals, edges, k) do  
  
    end  
end
```

Erlang:

```
-spec max_star_sum(Vals :: [integer()], Edges :: [[integer()]], K ::  
    integer()) -> integer().  
max_star_sum(Vals, Edges, K) ->  
.
```

Racket:

```
(define/contract (max-star-sum vals edges k)  
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?  
exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Maximum Star Sum of a Graph
```

```

* Difficulty: Medium
* Tags: array, graph, greedy, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

```

```

class Solution {
public:
    int maxStarSum(vector<int>& vals, vector<vector<int>>& edges, int k) {

    }
};

```

Java Solution:

```

/**
 * Problem: Maximum Star Sum of a Graph
 * Difficulty: Medium
 * Tags: array, graph, greedy, sort, queue, heap
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
*/

```

```

class Solution {
public int maxStarSum(int[] vals, int[][] edges, int k) {

}
}

```

Python3 Solution:

```

"""
Problem: Maximum Star Sum of a Graph
Difficulty: Medium
Tags: array, graph, greedy, sort, queue, heap

Approach: Use two pointers or sliding window technique

```

```

Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:

def maxStarSum(self, vals: List[int], edges: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def maxStarSum(self, vals, edges, k):
"""
:type vals: List[int]
:type edges: List[List[int]]
:type k: int
:rtype: int
"""

```

JavaScript Solution:

```

/**
 * Problem: Maximum Star Sum of a Graph
 * Difficulty: Medium
 * Tags: array, graph, greedy, sort, queue, heap
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * @param {number[]} vals
 * @param {number[][]} edges
 * @param {number} k
 * @return {number}
 */
var maxStarSum = function(vals, edges, k) {

};

```

TypeScript Solution:

```
/**  
 * Problem: Maximum Star Sum of a Graph  
 * Difficulty: Medium  
 * Tags: array, graph, greedy, sort, queue, heap  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(1) to O(n) depending on approach  
 */  
  
function maxStarSum(vals: number[], edges: number[][][], k: number): number {  
  
};
```

C# Solution:

```
/*  
 * Problem: Maximum Star Sum of a Graph  
 * Difficulty: Medium  
 * Tags: array, graph, greedy, sort, queue, heap  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(1) to O(n) depending on approach  
 */  
  
public class Solution {  
    public int MaxStarSum(int[] vals, int[][][] edges, int k) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Maximum Star Sum of a Graph  
 * Difficulty: Medium  
 * Tags: array, graph, greedy, sort, queue, heap  
 *  
 * Approach: Use two pointers or sliding window technique
```

```

* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
int maxStarSum(int* vals, int valsSize, int** edges, int edgesSize, int*
edgesColSize, int k) {
}

```

Go Solution:

```

// Problem: Maximum Star Sum of a Graph
// Difficulty: Medium
// Tags: array, graph, greedy, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maxStarSum(vals []int, edges [][]int, k int) int {
}

```

Kotlin Solution:

```

class Solution {
    fun maxStarSum(vals: IntArray, edges: Array<IntArray>, k: Int): Int {
    }
}

```

Swift Solution:

```

class Solution {
    func maxStarSum(_ vals: [Int], _ edges: [[Int]], _ k: Int) -> Int {
    }
}

```

Rust Solution:

```

// Problem: Maximum Star Sum of a Graph
// Difficulty: Medium
// Tags: array, graph, greedy, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
    pub fn max_star_sum(vals: Vec<i32>, edges: Vec<Vec<i32>>, k: i32) -> i32 {
        }

    }
}

```

Ruby Solution:

```

# @param {Integer[]} vals
# @param {Integer[][]} edges
# @param {Integer} k
# @return {Integer}
def max_star_sum(vals, edges, k)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[] $vals
     * @param Integer[][] $edges
     * @param Integer $k
     * @return Integer
     */
    function maxStarSum($vals, $edges, $k) {

    }
}

```

Dart Solution:

```
class Solution {  
    int maxStarSum(List<int> vals, List<List<int>> edges, int k) {  
        }  
    }  
}
```

Scala Solution:

```
object Solution {  
    def maxStarSum(vals: Array[Int], edges: Array[Array[Int]], k: Int): Int = {  
        }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec max_star_sum(vals :: [integer], edges :: [[integer]], k :: integer) ::  
        integer  
    def max_star_sum(vals, edges, k) do  
  
    end  
    end
```

Erlang Solution:

```
-spec max_star_sum(Vals :: [integer()], Edges :: [[integer()]], K ::  
    integer()) -> integer().  
max_star_sum(Vals, Edges, K) ->  
.
```

Racket Solution:

```
(define/contract (max-star-sum vals edges k)  
  (-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?  
       exact-integer?)  
)
```