|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S.No**. | **Case**  **Studies** | **Line of codes**  **(LOC)** | **Description** | **Time**  **Complexity** | **Space**  **Complexity** |  |  |  |  |
| 1 | ACM  Team | 37 | Maximum  Number of  Topics each  Team can  know | O(n^2) | 1KB |  |  |  |  |
| 2 | BFS | 42 | Breadth First  Search Graph  Traversal | O(E+V) | 1KB |  |  |  |  |
| 3 | Calendar  Problem | 59 | Predict the  Day when  Date is given | O(m)  m-> month | 2KB |  |  |  |  |
| 4 | Cavity  Map | 41 | Identify the  Cavity in map  By observing  Adjacent cell | O(n^2) | 1KB |  |  |  |  |
| 5 | Cipher | 22 | Decode the  Message by  Message and  Key | O(n) | 1KB |  |  |  |  |
| 6 | Compute  Days | 58 | Computes the  Day between  Any two days | O(n) | 2KB |  |  |  |  |
| 7 | GCD | 20 | Greatest  Common  divisor of two  Number | O(n) | 1KB |  |  |  |  |
| 8 | Love Letter  Mystery | 29 | Minimum  efforts To  make a String  Palindrome | O(n) | 1KB |  |  |  |  |
| 9 | Sherlock  And  Array | 46 | Sum of all  Elements on its  Left is equal to  Sum of all elements on its right | O(n^2) | 1KB |  |  |  |  |
| 10 | Triangle  Problem | 29 | Identify the Triangle | O(1) | 1KB |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

**Result and working of algorithm**

**Problem Consideration:**

**CIPHER**

**Problem statement**:

Jack and Daniel are friends.   
They want to encrypt their conversation so that they can save themselves from interception by a detective agency. So they invent a new cipher.   
Every message is encoded to its binary representation of length.   
Then it is written down times, shifted by bits.   
If and it looks so:

1001010

1001010

1001010

1001010

Then calculate [XOR](https://www.hackerrank.com/external_redirect?to=http://en.wikipedia.org/wiki/XOR#Bitwise_operation) in every column and write it down. This number is called. For example, XOR-ing the numbers in the above example results in

1110100110

Then the encoded message and are sent to Daniel.

Jack is using this encoding algorithm and asks Daniel to implement a decoding algorithm.   
Can you help Daniel implement this?

**Input Format**   
The first line contains two integers and.   
the second line contains string of length consisting of ones and zeros.

**Output Format**   
Decoded message of length, consisting of ones and zeros.

**Constraints**   
 1 <= N <= 10^6  
 1 <= K <= 10^6  
 |S| = N+K-1  
   
It is guaranteed that is correct.

**Sample Input#00**

7 4

1110100110

**Sample Output#00**

1001010

**Sample Input#01**

6 2

1110001

**Sample Output#01**

101111

**Explanation**

**Input#00**

1001010

1001010

1001010

1001010

----------

1110100110

**Input#01**

101111

101111

-------

1110001

----------------------\*\*\*\*\*\*\*---------------------\*\*\*\*\*\*\*\*----------------------\*\*\*\*\*\*\*\*\*-------------------\*\*\*\*\*\*\*\*

**ALGORITHM: (The cuckoo algorithm)**

**Input**:

A graph (which can contain self-loops, bridge edges and nodes) representation of the modules with their interconnectivity, weights given to the edges between modules, and with the start node and end node designated/mentioned.

**Output**: Optimized test sequence(s).

**Procedure**:

**Step 1**: The graph that is given as input is first engineered to obtain sequences from it, ensuring completeness, in the sense that every edge and node is covered at least once, say we obtained n sequences (n nests). These n nests correspond to the n host nests.

**Step 2**: Define a function F(x) as min {f(x)} (here it is min because we are considering costs involved as static weights), where f(x) is the fitness function that shall depend on the sequences.

f(x) = Σ (dynamic wt. \* static wt.)

Where i ∈ E(x) (set of edges of the adjacency matrix)

Initially, set the value of F(x) to infinity (corresponding to the fitness of the cuckoo).

**Step 3:**  Start with any randomly chosen sequence say Si (corresponding to the nest the cuckoo has chosen randomly), then calculate its fitness f(x) by

**3a** Calculating the dynamic weights of the edge transitions in the sequence. Dynamic weight of the edge from Ni to Ni+1 = in-degree of Ni+2\* out-degree of Ni, ∀i.

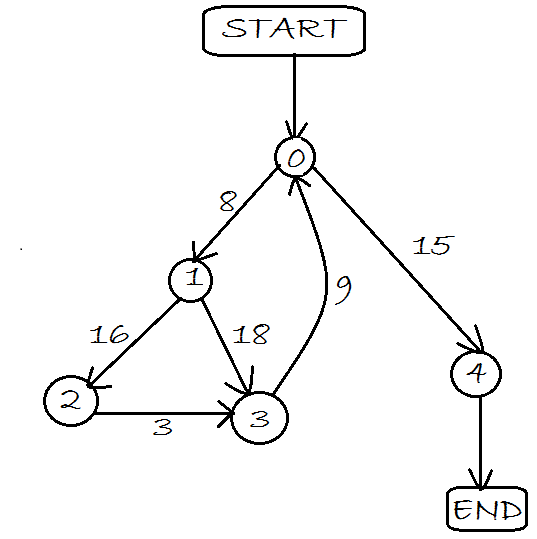
**3b** Calculate the product of the static weight (the weight assigned in the graph explicitly for the edges) and the dynamic weights calculated in Step 3a. This step is done over all the edges.

**3c** Calculate the sum of the products (calculated in Step 3b) over all the edges in the sequence.

**Step** **4**: Update the function value based on the fitness function, and pass those sequence(s) that were used in updating the function F(x) value, to the next iteration and discard the remaining sequences (the sequences that were used in comparison), which is analogous to discarding some fraction of the worst nests and retaining the best nests. In case of tie, in fitness values, all the corresponding sequences participating in the tie shall be passed to the further iteration.

**Step 5:** If all the sequences have been checked and compared, then end the procedure. Else, go to Step 3 with another sequence.

**DD (Decision To Decision) Graph**

****

**Graph data (Input)**

**I j weight**

**0 1 8**

**0 4 15**

**1 2 16**

**1 3 18**

**2 3 3**

**3 0 9**

**Fitness Calculation**

Path Optimized

[ 0 -> 1 -> 2 -> 3 -> 0 -> 4 ]

Fitness value = Dynamic weight \* Static weight

Dynamic weight = out degree \* in degree

Therefore,

Fitness value = out degree \* in degree \* static weight

2\*1\*8 + 2\*2\*16 + 1\*1\*3 + 1\*1\*9 + 2\*1\*15

16 + 64 + 3 + 9 + 30 = 122

**Process of Execution (Case study: Cipher problem)**

Step 1:

The program’s decision-to-decision graph data is considered. Graph data in form

Of node connectivity and their corresponding weight is given as input.

Step2:

Main input to the program:

Enter number of nodes (n):

5

Enter number of edges (e):

6

(Now next six lines will be input as the edge data)

I j weight

0 1 8

0 4 15

1 2 16

1 3 18

2 3 3

3 0 9

Based on that some considered paths are:

0 -> 4

0 -> 1 -> 3 -> 0 -> 4

0 -> 1 -> 2 -> 3 -> 0 -> 4 (We can see maximum node coverage)

Now let’s calculate its fitness

Fitness is the will be based on fitness function value i.e. objective value.

And Fitness function is:

F(x) = ∑ (static weight \* dynamic weight)

Summation will be taken of whole path

Static weight of a particular edge will be given as input

We have to calculate dynamic weight of the edge will be calculated as:

Dynamic weight = (out-degree of Ni node) \* (in-degree of Ni+2 node)

Ni = this is the current node

Ni+2 = this is the next to next node in the current path

Out-degree = number of outgoing edges from the a node

In-degree = number of ingoing edges to a node.

Calculation of dynamic weight of all edges of path (0 -> 1 -> 2 -> 3 -> 0 -> 4)

1. For 0 -> 1:

Out-degree (0) = 2

In-degree (2) = 1

Dynamic weight = out-degree\*in-degree

= 2\*1

= 2

1. For 1 -> 2:

Out-degree (1) = 2

In-degree (3) = 2

Dynamic weight = out-degree\*in-degree

= 2\*2

= 4

1. For 2 -> 3:

Out-degree (2) = 1

In-degree (0) = 1

Dynamic weight = out-degree\*in-degree

= 1\*1

= 1

1. For 3 -> 0:

Out-degree (3) = 1

In-degree (4) = 1

Dynamic weight = out-degree\*in-degree

= 1\*1

= 1

1. For 0 -> 4:

Out-degree (0) = 2

In-degree = 1 (As in-degree will be 1 because no next node is present)

Dynamic weight = out-degree\*in-degree

= 2\*1

= 2

We have successfully calculate the dynamic weights of all edges

Now we have to calculate the summation of the product of dynamic weights

And static weights of edges of the whole path.

Therefore,

Fitness function will be:

F(x) = ∑ (static weight \* dynamic weight)

= 8\*2 + 16\*4 + 3\*1 + 9\*1 + 15\*2

= 16 + 64 + 3 + 9 + 30

= 122