Codeforces Round #435 (Div. 2)

Problem B – Mahmoud and Ehab and the bipartiteness

Prof. Edson Alves

Faculdade UnB Gama

Mahmoud and Ehab continue their adventures! As everybody in the evil land knows, Dr. Evil likes bipartite graphs, especially trees.

A tree is a connected acyclic graph. A bipartite graph is a graph, whose vertices can be partitioned into 2 sets in such a way, that for each edge (u, v) that belongs to the graph, u and v belong to different sets. You can find more formal definitions of a tree and a bipartite graph in the notes section below.

Dr. Evil gave Mahmoud and Ehab a tree consisting of n nodes and asked them to add edges to it in such a way, that the graph is still bipartite. Besides, after adding these edges the graph should be simple (doesn't contain loops or multiple edges). What is the maximum number of edges they can add?

A loop is an edge, which connects a node with itself. Graph doesn't contain multiple edges when for each pair of nodes there is no more than one edge between them. ***A cycle and a loop aren't the same****.*

Mahmoud e Ehab continuam suas aventuras! Como todos na terra do mal sabem, Dr. Evil gosta de grafos bipartidos, especialmente árvores.

Uma árvore é um grafo conectado acíclico. Um grafo bipartido é um grafo cujos vértices podem ser particionados em 2 conjuntos de forma que, para cada aresta (u, v) que pertence ao grafo, u e v pertencem a conjuntos diferentes. Você pode encontrar definições mais formais de árvore e grafo bipartido na seção de notas abaixo.

Dr. Evil deu a Mahmoud e Ehab uma árvore composta por n vértices e pediu a eles para adicionarem arestas ao grafo de modo que ele permaneça ainda bipartido. Além disso, após adicionar as arestas o grafo ainda deve ser simples (não pode conter *loops* ou arestas múltiplas). Qual é o número máximo de arestas que eles podem adicionar?

Um *loop* é uma aresta que conecta o vértice a si mesmo. Grafos não contém múltiplas arestas quando para cada par de vértices existe no máximo uma aresta entre eles. Um ciclo e um *loop* não são a mesma coisa.

Input

The first line of input contains an integer n – the number of nodes in the tree $(1 \leq n \leq 10^5)$.

The next $n-1$ lines contain integers u and v $(1 \leq u, v \leq n, u \neq v)$ – the description of the edges of the tree.

It's guaranteed that the given graph is a tree.

Output

Output one integer – the maximum number of edges that Mahmoud and Ehab can add to the tree while fulfilling the conditions.

Entrada

A primeira linha da entrada contém um inteiro n – o número de vértices na árvore $(1 \leq n \leq 10^5)$.

As próximas $n-1$ linhas contém os inteiros u e v $(1 \leq u, v \leq n, u \neq v)$ – a descrição das arestas da árvore.

É garantido que o grafo dado é um árvore.

Saída

Imprima um número – o máximo de arestas que Mahmoud e Ehab podem adicionar à árvore respeitando as condições impostas.

Exemplo de entrada e saída

Exemplo de entrada e saída

```
3
```

Exemplo de entrada e saída

Exemplo de entrada e saída

3

 # de vértices

Exemplo de entrada e saída

3

1 2

Exemplo de entrada e saída

```
3

1 2
```

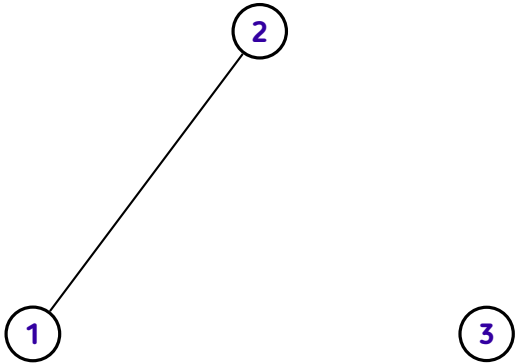
primeira aresta

Exemplo de entrada e saída

3

1 2

primeira aresta

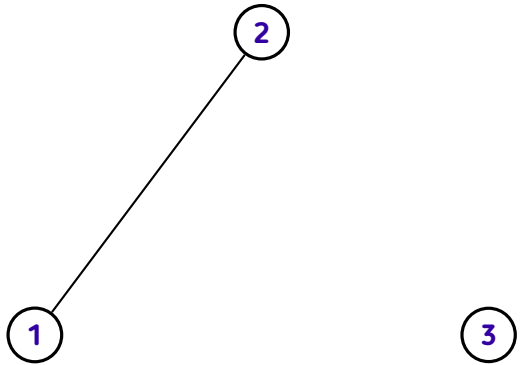


Exemplo de entrada e saída

3

1 2

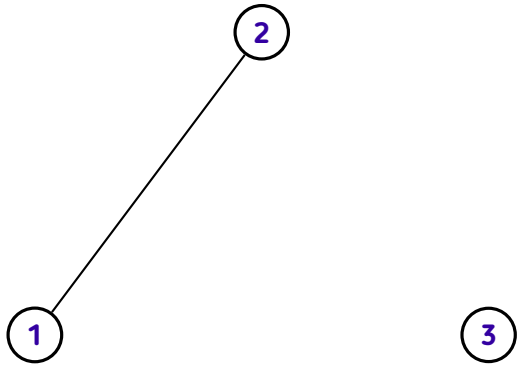
1 3



Exemplo de entrada e saída

```
3
1 2
1 3
```

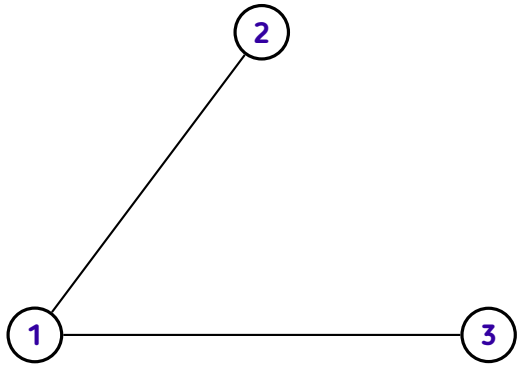
segunda aresta



Exemplo de entrada e saída

```
3
1 2
1 3
```

segunda aresta

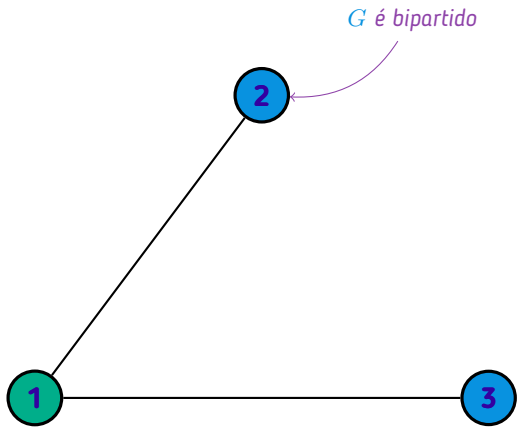


Exemplo de entrada e saída

3

1 2

1 3

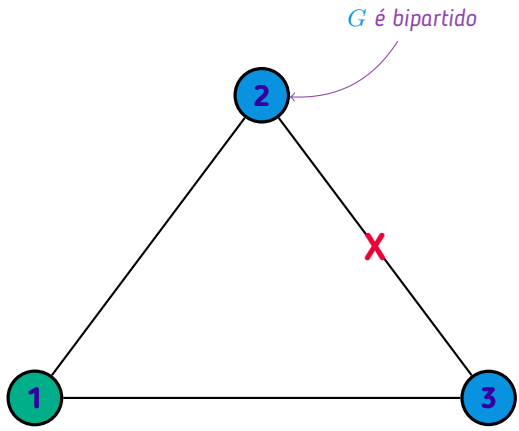


Exemplo de entrada e saída

3

1 2

1 3



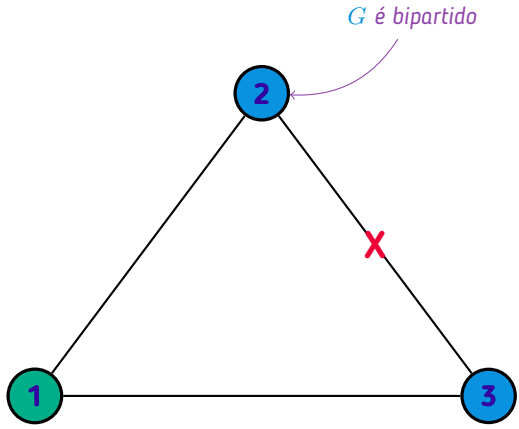
Exemplo de entrada e saída

3

1 2

1 3

0



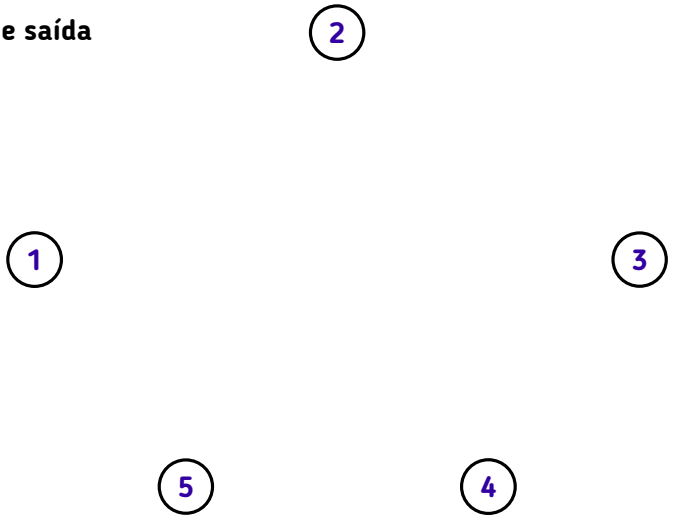
Exemplo de entrada e saída

Exemplo de entrada e saída

```
5
```

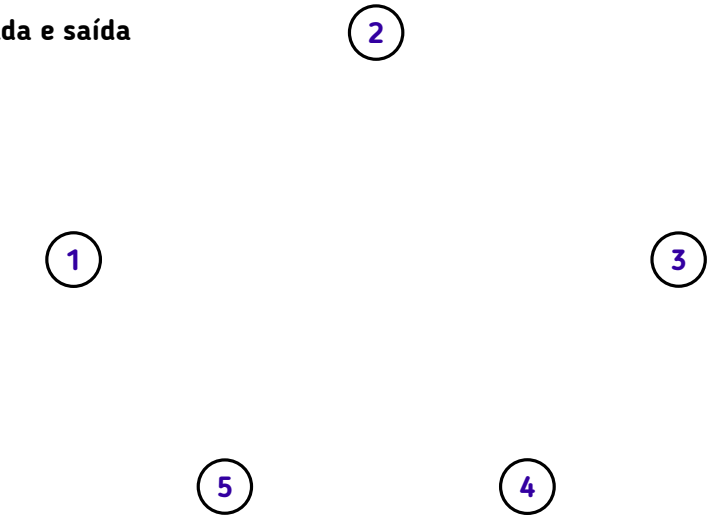
Exemplo de entrada e saída

```
5
```



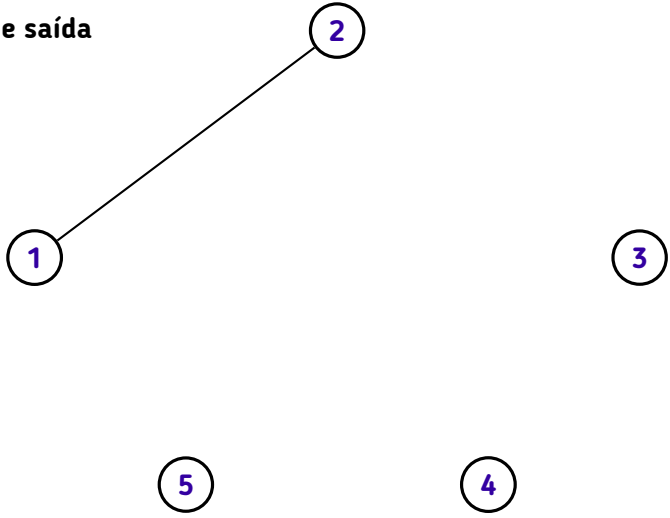
Exemplo de entrada e saída

```
5
1 2
```



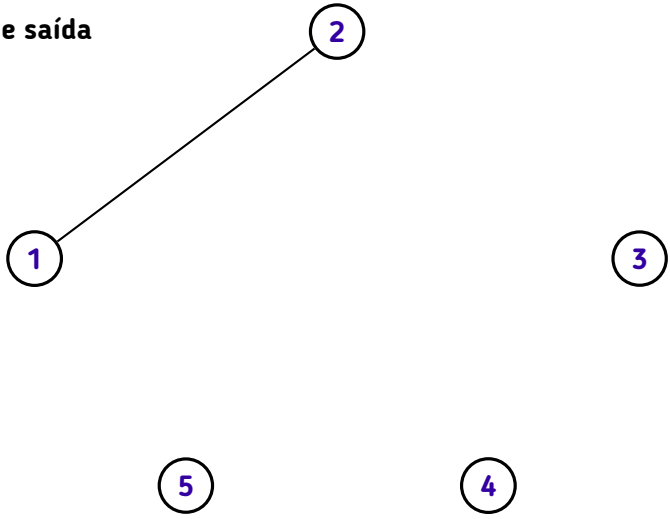
Exemplo de entrada e saída

5
1 2



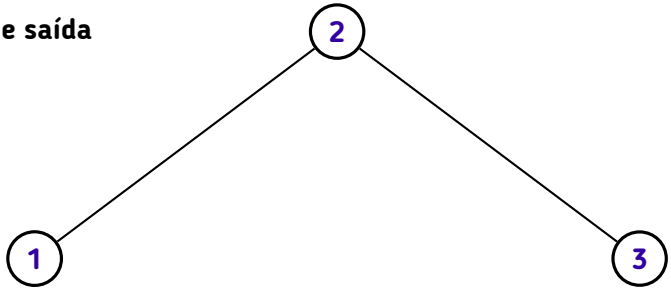
Exemplo de entrada e saída

5
1 2
2 3



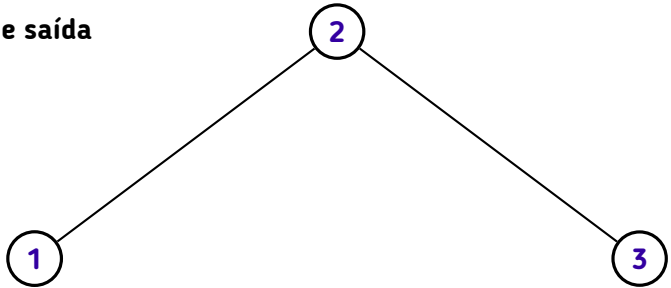
Exemplo de entrada e saída

```
5
1 2
2 3
```



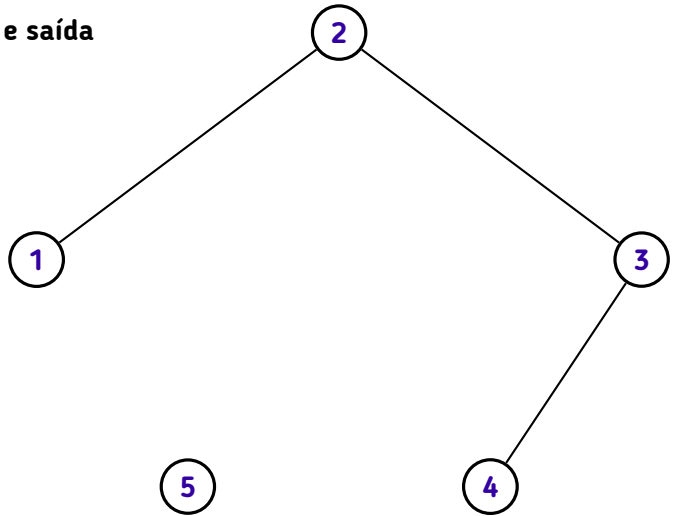
Exemplo de entrada e saída

```
5
1 2
2 3
3 4
```



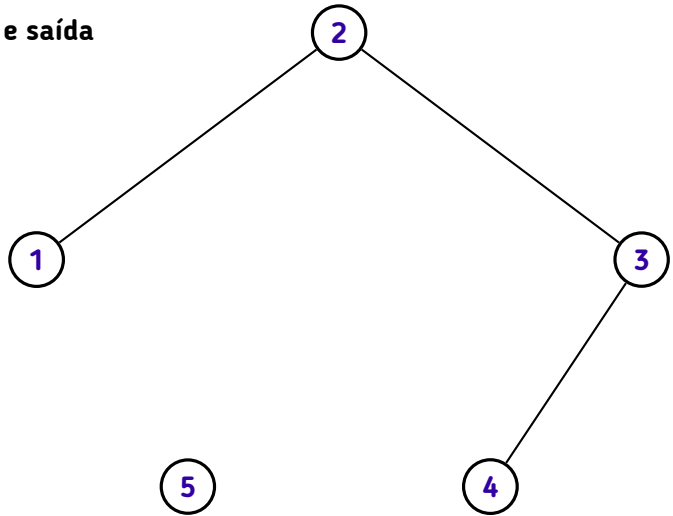
Exemplo de entrada e saída

```
5
1 2
2 3
3 4
```



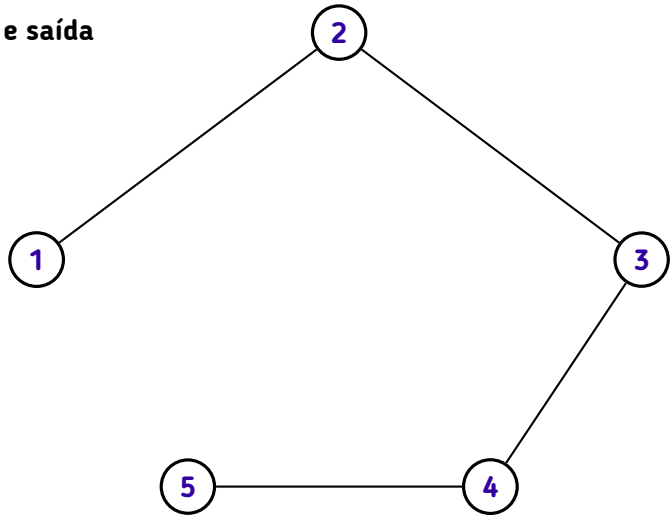
Exemplo de entrada e saída

```
5
1 2
2 3
3 4
4 5
```



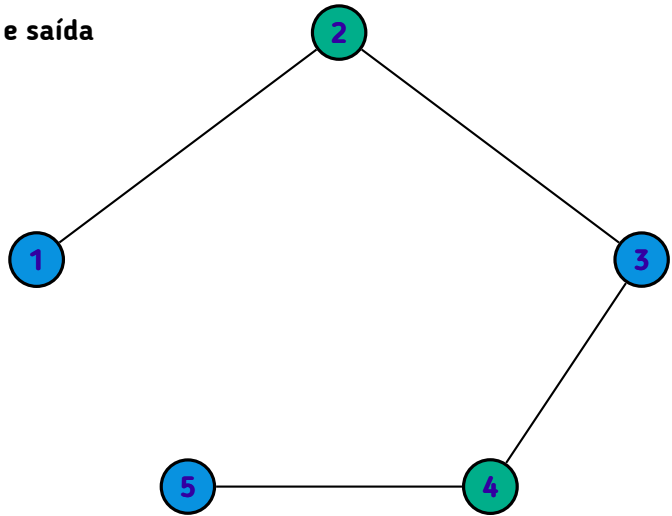
Exemplo de entrada e saída

```
5
1 2
2 3
3 4
4 5
```



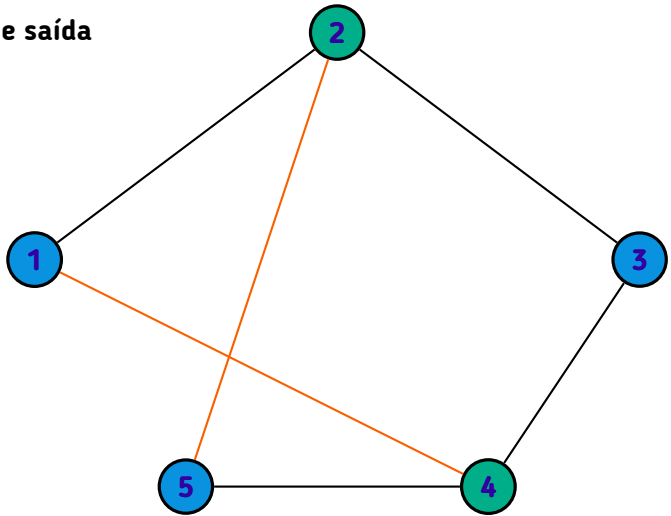
Exemplo de entrada e saída

```
5
1 2
2 3
3 4
4 5
```



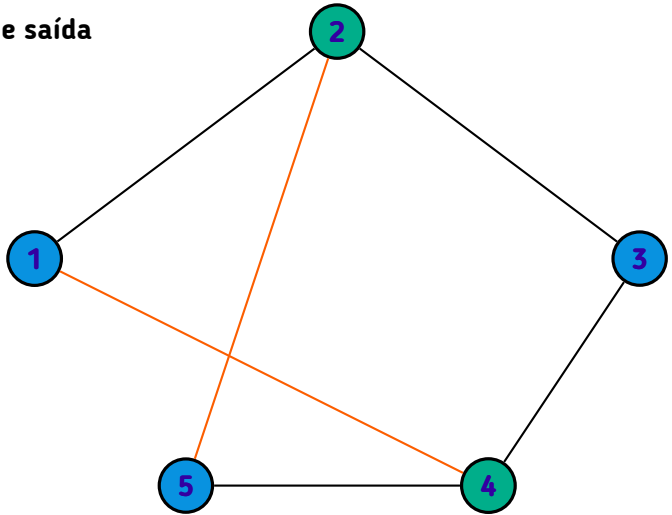
Exemplo de entrada e saída

```
5
1 2
2 3
3 4
4 5
```



Exemplo de entrada e saída

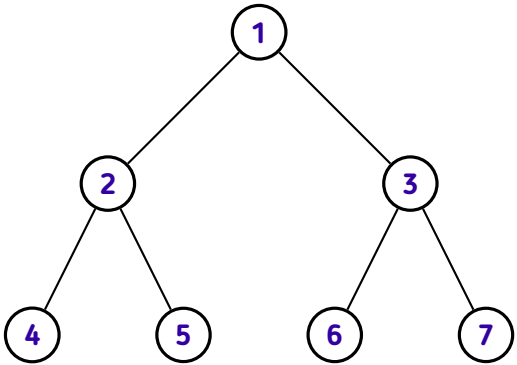
```
5
1 2
2 3
3 4
4 5
```



2

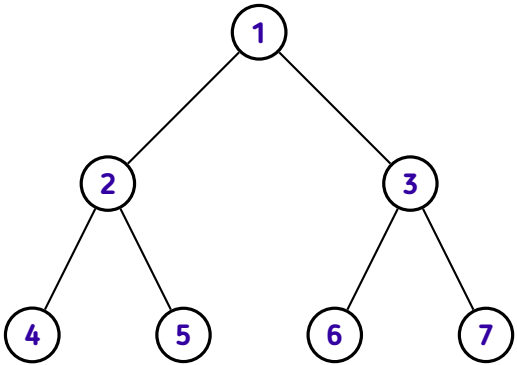
Solução

Solução



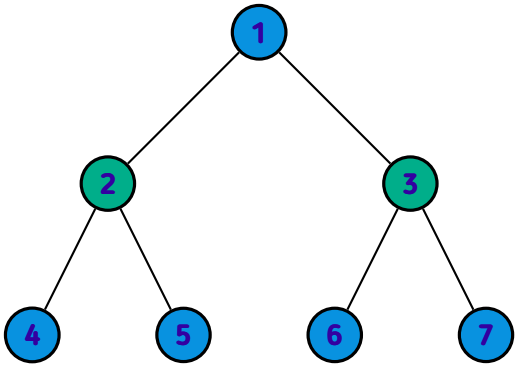
Solução

Toda árvore é bipartida



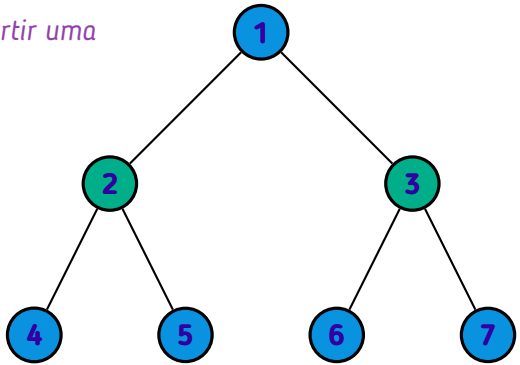
Solução

Toda árvore é bipartida



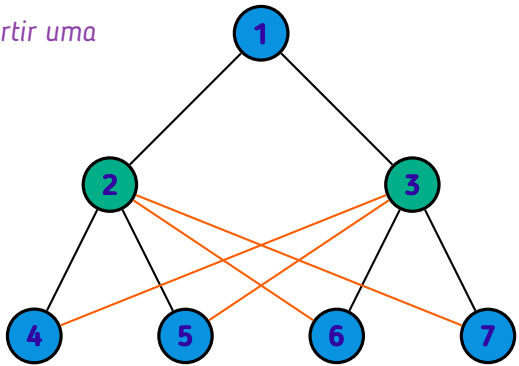
Solução

De cada nó verde deve partir uma aresta para um nó azul



Solução

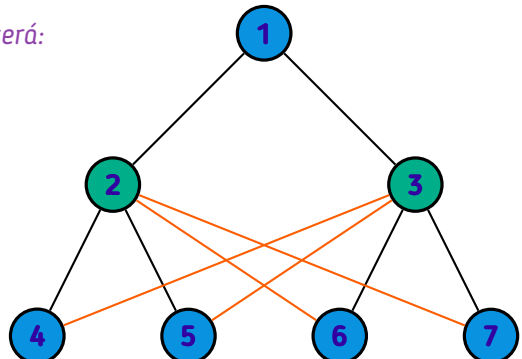
De cada nó verde deve partir uma aresta para um nó azul



Solução

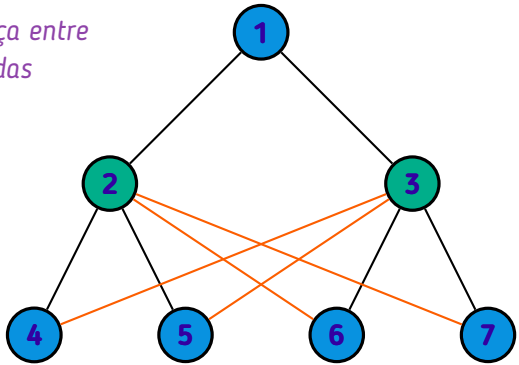
Portanto o total de arestas será:

(# verde) $\times$ *(# azul)*



Solução

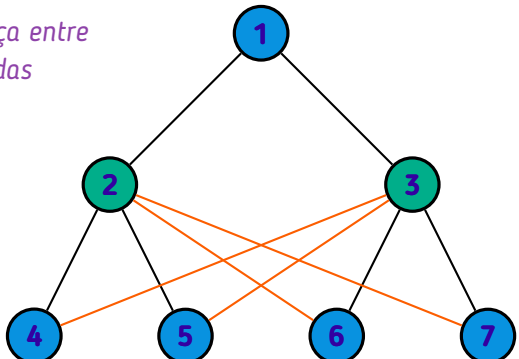
A resposta será a diferença entre o total e as arestas já definidas



Solução

A resposta será a diferença entre o total e as arestas já definidas

$2 \times 5 - 6 = \mathbf{4}$



```
ll solve(int N)
{
    queue<int> q; q.push(1);
    color[1] = 1;

    while (not q.empty()) {
        auto u = q.front(); q.pop();

        for (auto v : adj[u])
            if (color[v] == -1) {
                color[v] = 1 - color[u];
                q.push(v);
            }
    }

    auto blue = accumulate(color.begin() + 1, color.begin() + N + 1, 0LL);
    auto green = N - blue, ans = (blue * green) - (N - 1);

    return ans;
}
```