SPOJ DAGCNT2

Counting in a DAG

Prof. Edson Alves

Faculdade UnB Gama

You are given a weighted DAG. For each vertex, calculate the sum of the weights of the vertices within its reach (including itself).

A você é dado um DAG ponderado. Para cada vértice, compute a soma dos pesos dos vértices que são atingíveis a partir deste vértice (inclusive o próprio vértice).

Input

The first line contains an integer T, denoting the number of test cases.

For each test case, the first line contains two positive integers n and m, denoting the number of vertices and the number of edges in the DAG.

The second line contains n positive integers $w_1, \ldots, w_n$, denoting the weights of vertices.

The next m lines contain two positive integers u, v, denoting an edge from u to v.

Output

For each test case, print a line consisting of n numbers, denoting the sum for each vertex.

Entrada

A primeira linha contém um inteiro T, o qual indica o número de casos de teste.

A primeira linha de um caso de teste contém dois inteiros positivos n e m, representando o número de vértices e o número de arestas no DAG.

A segunda linha contém n inteiros positivos $w_1, \ldots, w_n$, que representam os pesos dos vértices.

As próximas m linhas contém dois inteiros positivos u, v, indicando uma aresta de u a v.

Saída

Para cada caso de teste imprima uma linha contendo n números, que correspondem à soma para cada vértice.

Constraints

Input Set 1: $T \leq 40, n \leq 100, m \leq 10000$

Input Set 2: $T \leq 2, n \leq 1000, m \leq 500000$

Input Set 3: $T \leq 2, n \leq 20000, m \leq 500000$

The weights are no more than 1000.

Restrições

Conjunto de entradas 1: $T \leq 40, n \leq 100, m \leq 10000$

Conjunto de entradas 2: $T \leq 2, n \leq 1000, m \leq 500000$

Conjunto de entradas 3: $T \leq 2, n \leq 20000, m \leq 500000$

Os pesos não são maiores do que 1000**.**

Exemplo de entrada e saída

Exemplo de entrada e saída

```
4 3
```

Exemplo de entrada e saída

```
4 3
```

 # de vértices

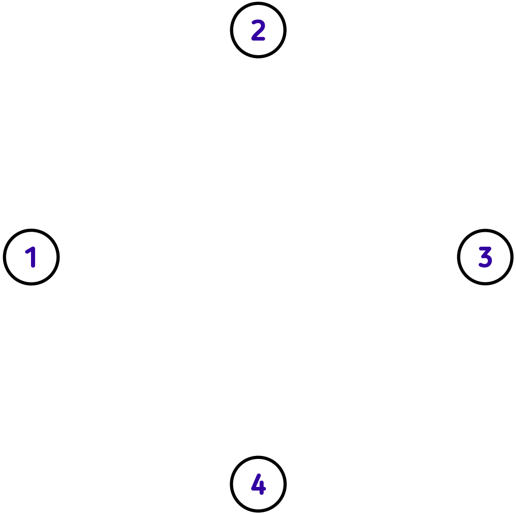
Exemplo de entrada e saída

```
4 3
```

 # de arestas

Exemplo de entrada e saída

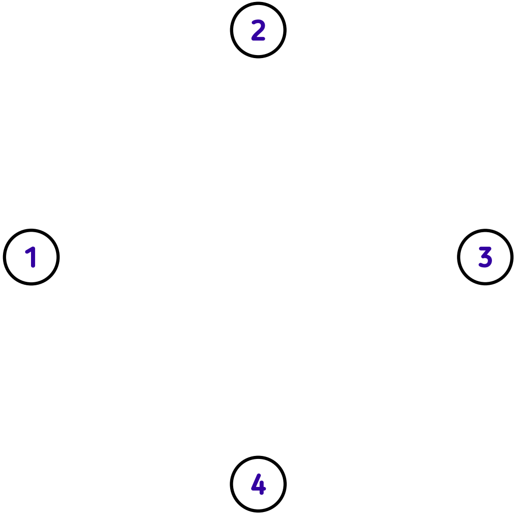
```
4 3
```



Exemplo de entrada e saída

```
4 3
510 713 383 990
```

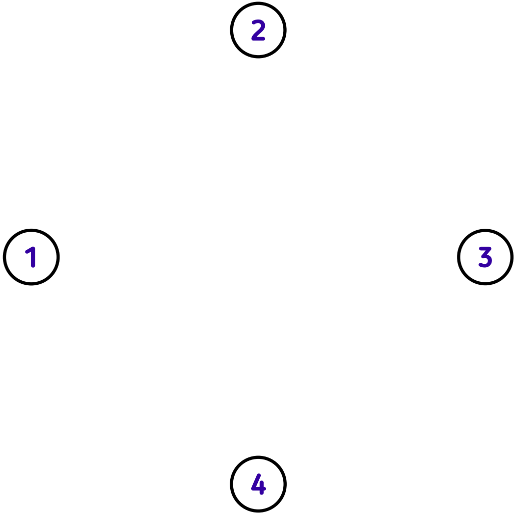
w_1



Exemplo de entrada e saída

```
4 3
510 713 383 990
```

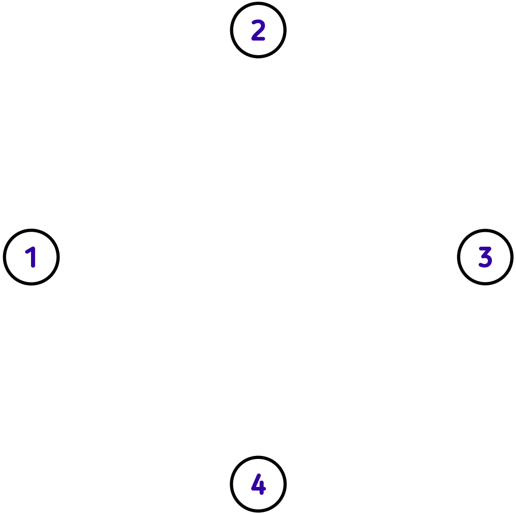
w_2



Exemplo de entrada e saída

```
4 3
510 713 383 990
```

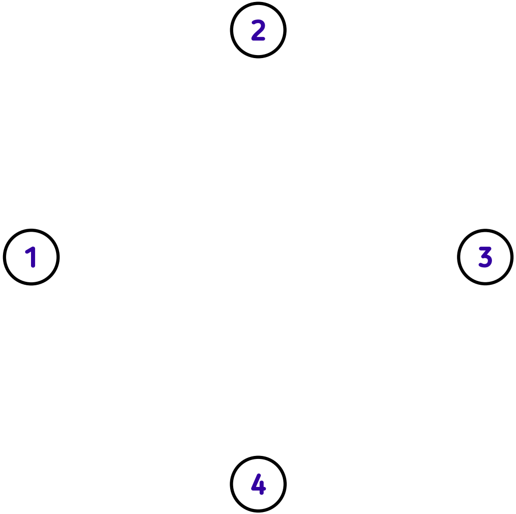
w_3



Exemplo de entrada e saída

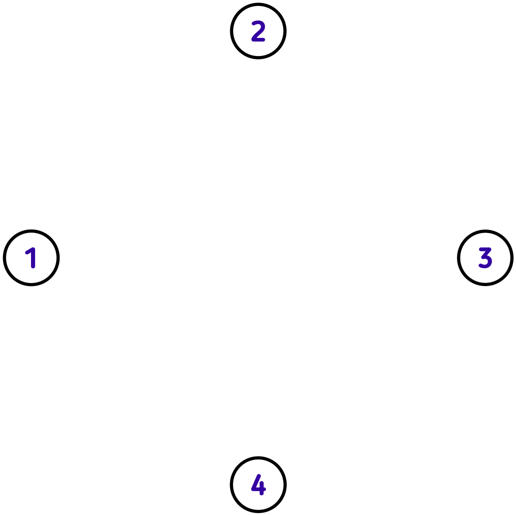
```
4 3
510 713 383 990
```

w_4



Exemplo de entrada e saída

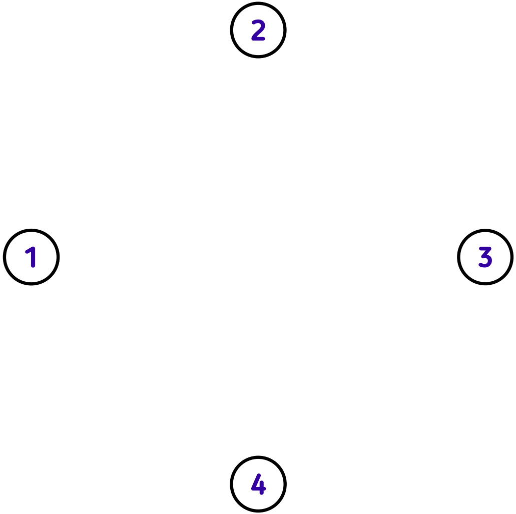
```
4 3
510 713 383 990
4 1
```



Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
```

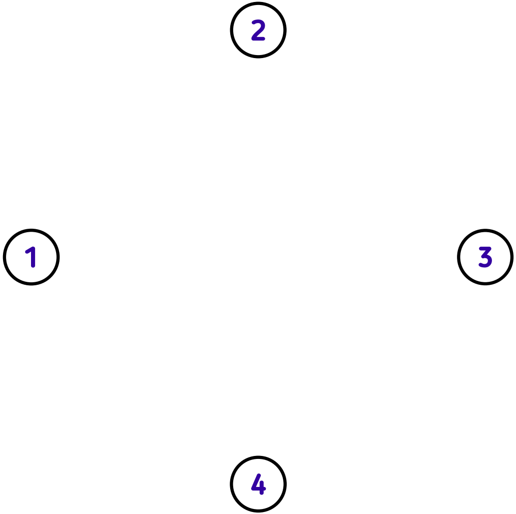
u



Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
```

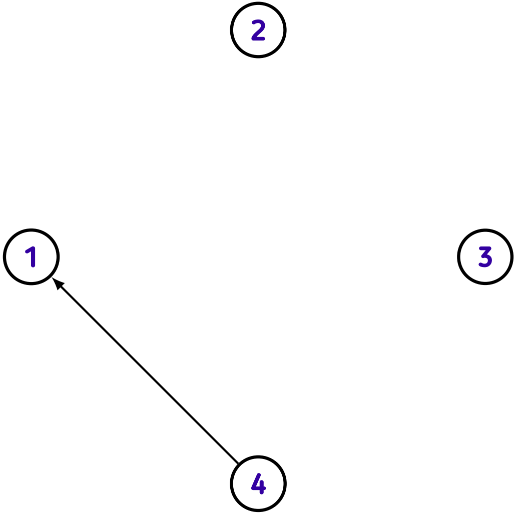
v



Exemplo de entrada e saída

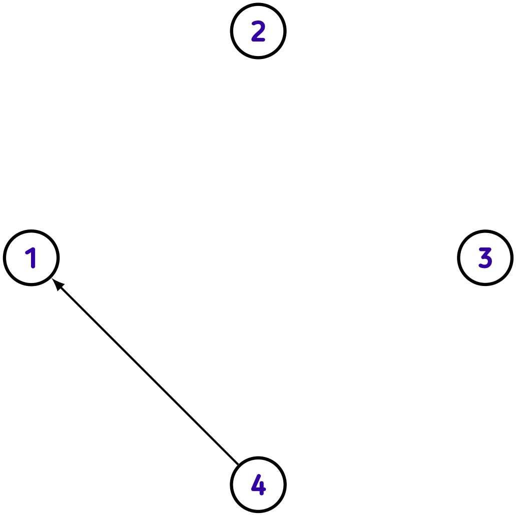
```
4 3
510 713 383 990
4 1
```

v



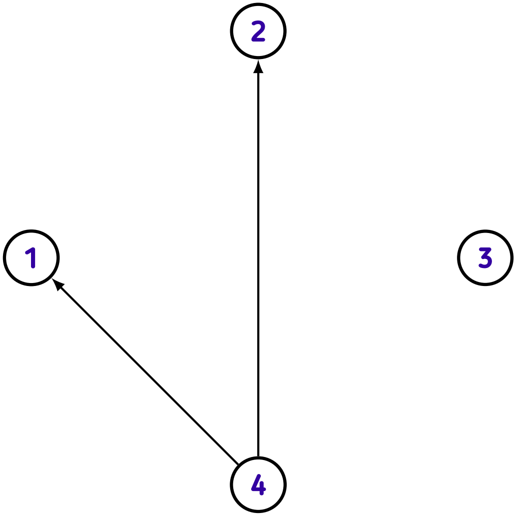
Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
```



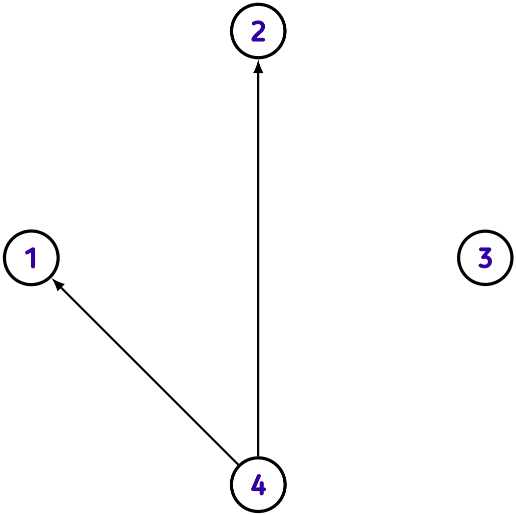
Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
```



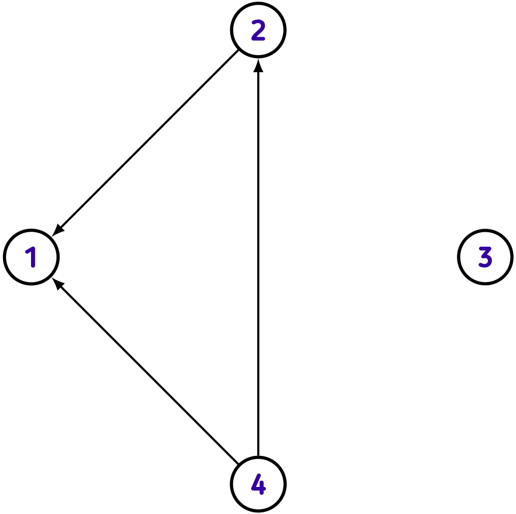
Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
2 1
```



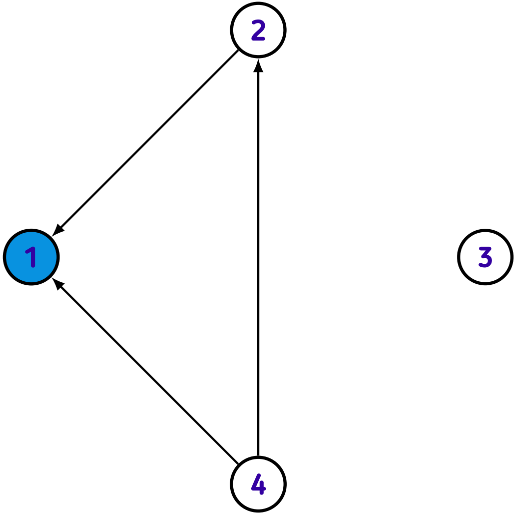
Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
2 1
```



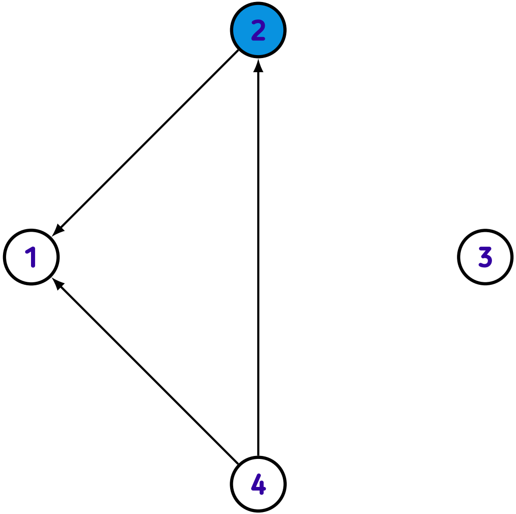
Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
2 1
```



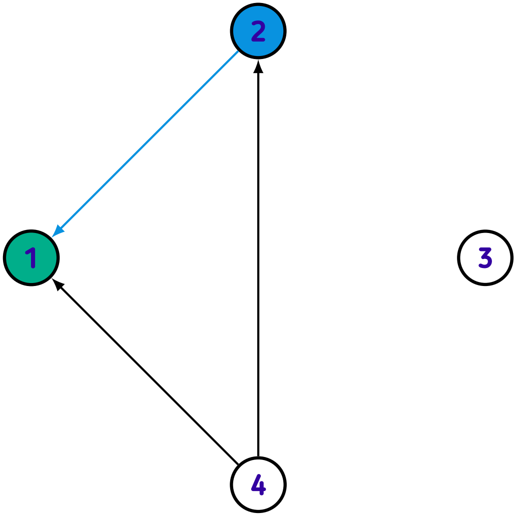
Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
2 1
```



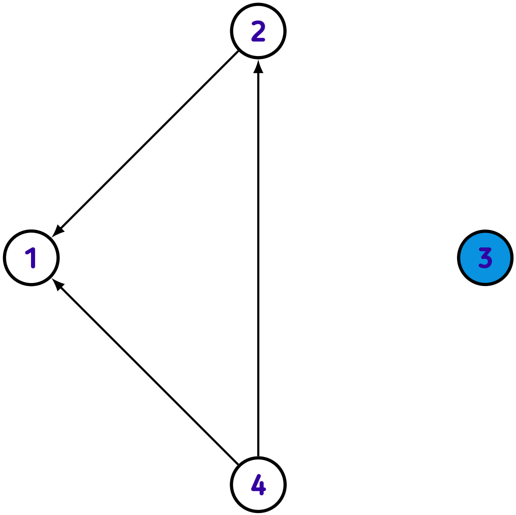
Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
2 1
```



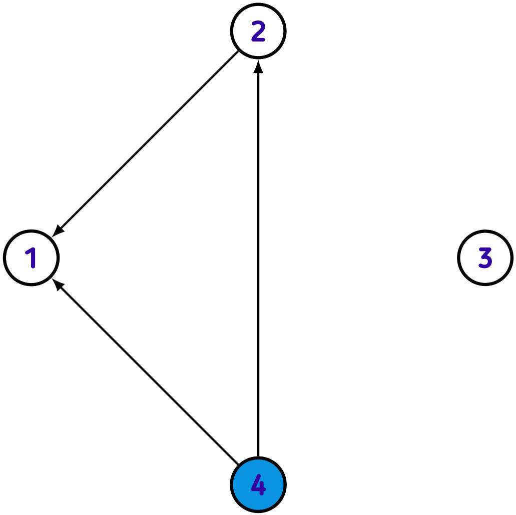
Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
2 1
```



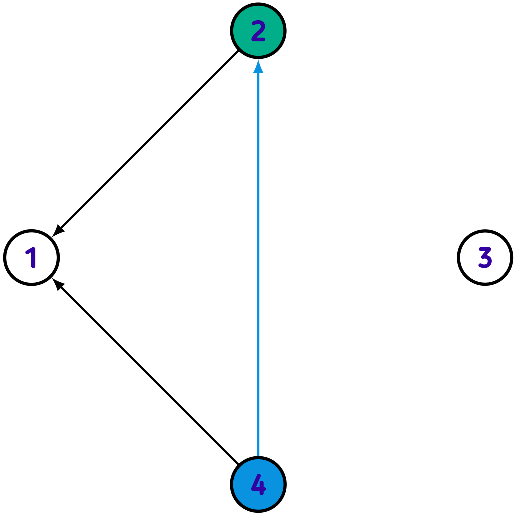
Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
2 1
```



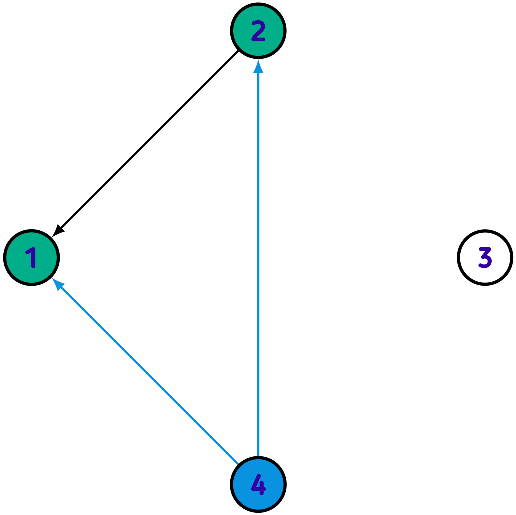
Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
2 1
```



Exemplo de entrada e saída

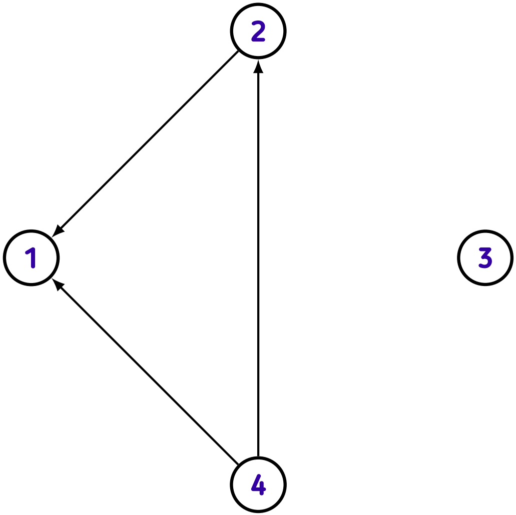
```
4 3
510 713 383 990
4 1
4 2
2 1
```



Exemplo de entrada e saída

```
4 3
510 713 383 990
4 1
4 2
2 1
```

```
510 1223 383 2213
```



Solução

Solução

$\star$ **Uma travessia pode determinar o alcance do vértice u e determinar a soma desejada**

Solução

- $\star$ **Uma travessia pode determinar o alcance do vértice u e determinar a soma desejada**

- $\star$ **Seriam N travessias, cada uma realizada em $O(N+M)$**

Solução

- $\star$ **Uma travessia pode determinar o alcance do vértice u e determinar a soma desejada**

- $\star$ **Seriam N travessias, cada uma realizada em $O(N+M)$**

- $\star$ **A complexidade desta solução é $O(N^2+NM)$**

Solução

- $\star$ Uma travessia pode determinar o alcance do vértice u e determinar a soma desejada
- $\star$ Seriam N travessias, cada uma realizada em $O(N + M)$
- $\star$ A complexidade desta solução é $O(N^2 + NM)$
- $\star$ **Veredito:** TLE!

Solução

- $\star$ **Contudo, com as devidas otimizações, é possível obter um veredito AC com uma solução com complexidade** $O(N^2 + NM)$

Solução

- $\star$ **Contudo, com as devidas otimizações, é possível obter um veredito AC com uma solução com complexidade** $O(N^2 + NM)$

- $\star$ **A primeira providência é utilizar I/O eficiente (`printf()`/`scanf()`)**

Solução

- $\star$ Contudo, com as devidas otimizações, é possível obter um veredito AC com uma solução com complexidade $O(N^2 + NM)$
- $\star$ A primeira providência é utilizar I/O eficiente (`printf()`/`scanf()`)
- $\star$ Devido a natureza do problema, não é possível utilizar uma DP que use as somas como estado do problema

Solução

- $\star$ Contudo, com as devidas otimizações, é possível obter um veredito AC com uma solução com complexidade $O(N^2 + NM)$

- $\star$ A primeira providência é utilizar I/O eficiente (`printf()`/`scanf()`)

- $\star$ Devido a natureza do problema, não é possível utilizar uma DP que use as somas como estado do problema

- $\star$ Isto porque, se um vértice u atingir v por mais de um caminho, o peso de v seria totalizado mais de uma vez

Solução

- Seja $R[u]$ o conjunto do vértices alcancáveis a partir de u

Solução

- $\star$ **Seja $R[u]$ o conjunto do vértices alcancáveis a partir de u**
- $\star$ **Se o grau de saída de u é igual a zero, então $R[u] = \{ u \}$**

Solução

$\star$ **Seja** $R[u]$ **o conjunto do vértices alcancáveis a partir de** u

$\star$ **Se o grau de saída de** u **é igual a zero, então** $R[u] = \{ \ u \ \}$

$\star$ **Caso contrário, se os estados forem computados na ordem inversa da ordenação topológica, então**

$$R[u] = \{ \ u \ \} \cup \{ \ R[v_1] \cup R[v_2] \cup \ldots, R[v_k] \ \}, \quad \forall (u, v_i) \in E$$

Solução

- $\star$ **Seja $R[u]$ o conjunto do vértices alcancáveis a partir de u**

- $\star$ **Se o grau de saída de u é igual a zero, então $R[u] = \{\ u\ \}$**

- $\star$ **Caso contrário, se os estados forem computados na ordem inversa da ordenação topológica, então**

$$R[u] = \{\ u\ \} \cup \{\ R[v_1] \cup R[v_2] \cup \ldots, R[v_k]\ \}, \quad \forall (u, v_i) \in E$$

- $\star$ **Dadas as restrições de tempo, $R[u]$ deve ser representado por um *bitset***

Solução

- **Além disso, a implementação deste *bitset* deve ser customizada**

Solução

- $\star$ Além disso, a implementação deste *bitset* deve ser customizada

- $\star$ Se os elementos são representados em $R[u]$ por seus índices na ordenação topológica, a união pode ser feita de forma mais eficiente

Solução

- $\star$ **Além disso, a implementação deste *bitset* deve ser customizada**

- $\star$ **Se os elementos são representados em $R[u]$ por seus índices na ordenação topológica, a união pode ser feita de forma mais eficiente**

- $\star$ **Isto porque todo vértice v alcancável a partir do vértice u o sucederá na ordenação topológica**

Solução

- $\star$ Além disso, a implementação deste *bitset* deve ser customizada
- $\star$ Se os elementos são representados em $R[u]$ por seus índices na ordenação topológica, a união pode ser feita de forma mais eficiente
- $\star$ Isto porque todo vértice v alcancável a partir do vértice u o sucederá na ordenação topológica
- $\star$ Assim, ao contrário do *bitset* do C++, é possível unir $R[u]$ e $R[v_i]$ somente até o índice de u na ordenação topológica

```cpp
void solve(int N)
{
    auto o = reverse_topological_sort(N);

    for (int i = 0; i < N; ++i) {
        auto u = o[i];
        setbit(R, u, i);

        for (auto v : adj[u])
            sets_union(R, u, v, i);
    }

    for (int u = 1; u <= N; ++u) {
        ans[u] = 0;

        for (int i = 0; i < N; ++i)
            if (getbit(R, u, i))
                ans[u] += cost[o[i]];
    }
}
```

```
#define setbit(R, u, pos) R[u][pos >> 5] |= (1U << (pos & 0x1f))
#define getbit(R, u, pos) R[u][pos >> 5] & (1U << (pos & 0x1f))
#define sets_union(R, u, v, pos) for (int k = 0; k <= (pos >> 5); ++k) \
                               { R[u][k] |= R[v][k]; }
```