Grafos

Decteção de ciclos em grafos de sucessores

Prof. Edson Alves

Faculdade UnB Gama

Identificação de ciclos

Identificação de ciclos

Seja G um grafo de sucessores que contém um caminho que termine em um ciclo.

Identificação de ciclos

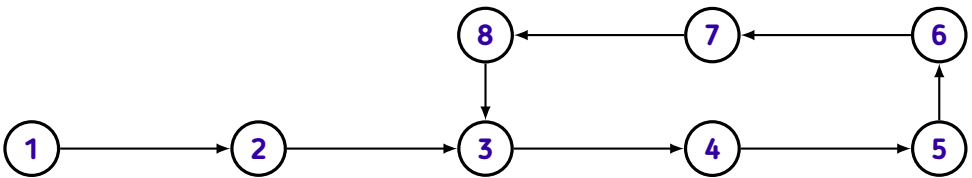
Seja G um grafo de sucessores que contém um caminho que termine em um ciclo.

1. **Qual é o nó de menor índice que pertence ao ciclo?** (μ)
2. **O ciclo contém quantos nós?** (λ)

Identificação de ciclos

Seja G um grafo de sucessores que contém um caminho que termine em um ciclo.

1. **Qual é o nó de menor índice que pertence ao ciclo?** (μ)
2. **O ciclo contém quantos nós?** (λ)



Algoritmo linear em execução e memória

Algoritmo linear em execução e memória

1. **Faça** $u = 1$ **e** $s = \emptyset$

Algoritmo linear em execução e memória

1. **Faça** $u = 1$ **e** $s = \emptyset$

2. **Enquanto** $u \notin s$**:**

Algoritmo linear em execução e memória

1. **Faça** $u = 1$ **e** $s = \emptyset$
2. **Enquanto** $u \notin s$**:**
   (a) **Insera** u **em** s

Algoritmo linear em execução e memória

1. **Faça** $u = 1$ **e** $s = \emptyset$
2. **Enquanto** $u \notin s$**:**
   (a) **Insera** u **em** s
   (b) **Faça** $u = \mathsf{succ}(u)$

Algoritmo linear em execução e memória

1. **Faça** $u = 1$ **e** $s = \emptyset$
2. **Enquanto** $u \notin s$**:**
   (a) **Insera** u **em** s
   (b) **Faça** $u = \mathsf{succ}(u)$
3. $\mu = u$

Algoritmo linear em execução e memória

1. **Faça** $u = 1$ **e** $s = \emptyset$
2. **Enquanto** $u \notin s$**:**
   (a) **Insera** u **em** s
   (b) **Faça** $u = \mathsf{succ}(u)$
3. $\mu = u$
4. $\lambda = N - \mu + 1$**, onde** $N = |V|$

```cpp
pair<int, int> detect_cycle(int N, const vector<int>& succ)
{
    int u = 1;
    unordered_set<int> s;

    while (not s.count(u))
    {
        s.insert(u);
        u = succ[u];
    }

    auto mu = u;
    auto lambda = N - u + 1;

    return { mu, lambda };
}
```

Algoritmo de Floyd

Algoritmo de Floyd

- Também conhecido como algoritmo da lebre e da tartaruga

Algoritmo de Floyd

- ⋆ **Também conhecido como algoritmo da lebre e da tartaruga**
- ⋆ **Identifica um ciclo com execução $O(N)$ e memória $O(1)$**

Algoritmo de Floyd

- ⋆ **Também conhecido como algoritmo da lebre e da tartaruga**
- ⋆ **Identifica um ciclo com execução $O(N)$ e memória $O(1)$**
- ⋆ **Utilizada dois ponteiros: a lebre e a tartaruga**

Algoritmo de Floyd

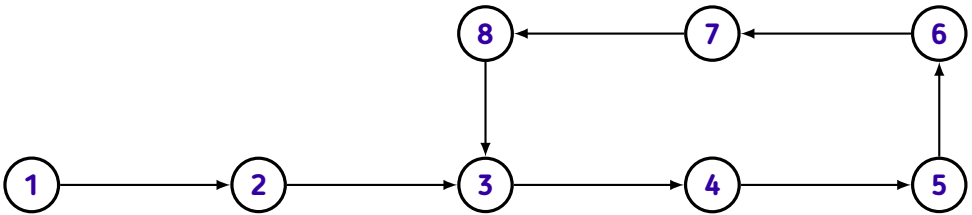
- ⋆ Também conhecido como algoritmo da lebre e da tartaruga
- ⋆ Identifica um ciclo com execução $O(N)$ e memória $O(1)$
- ⋆ Utilizada dois ponteiros: a lebre e a tartaruga
- ⋆ A cada passo da tartaruga, a lebre caminha dois passos

Algoritmo de Floyd

- Também conhecido como algoritmo da lebre e da tartaruga
- Identifica um ciclo com execução $O(N)$ e memória $O(1)$
- Utilizada dois ponteiros: a lebre e a tartaruga
- A cada passo da tartaruga, a lebre caminha dois passos
- É composto de 3 etapas

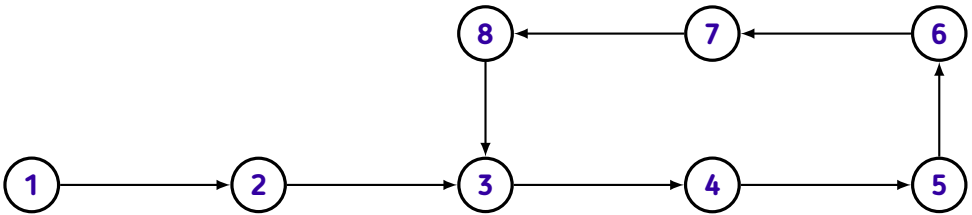
Etapa 1: Identificação do ciclo

Etapa 1: Identificação do ciclo



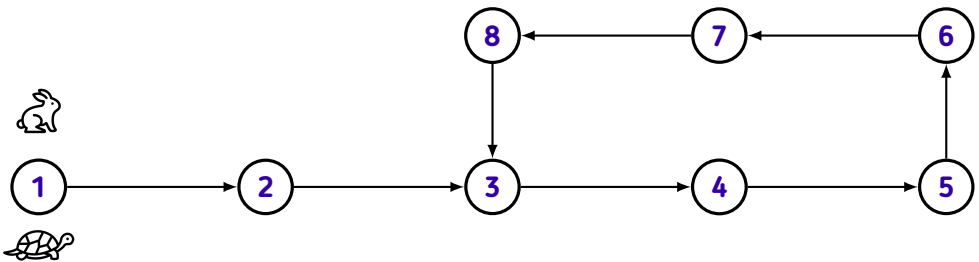
Etapa 1: Identificação do ciclo

(a) **A tartaruga e a lebre são posicionadas no primeiro vértice**



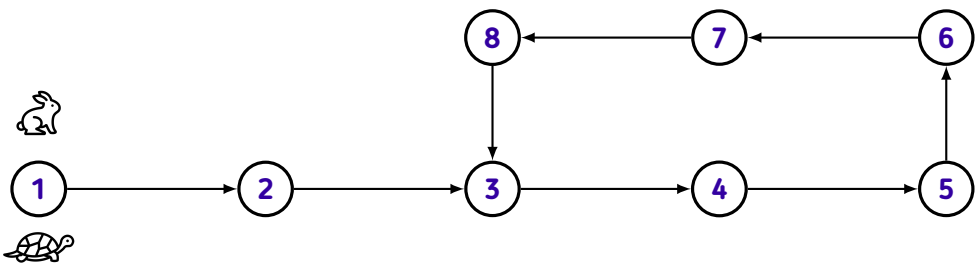
Etapa 1: Identificação do ciclo

(a) **A tartaruga e a lebre são posicionadas no primeiro vértice**



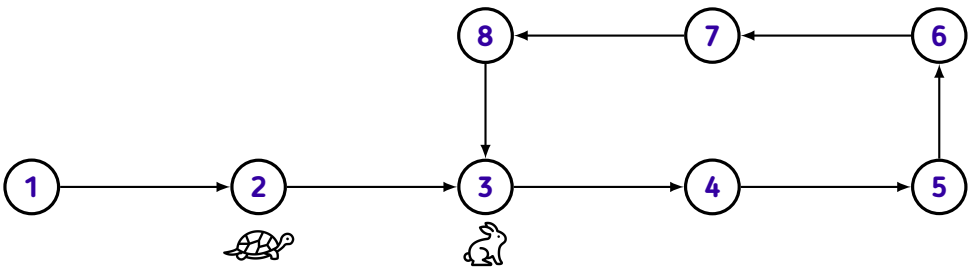
Etapa 1: Identificação do ciclo

(b) **A lebre dá dois passos, a tartagura um, até se reencontrarem**



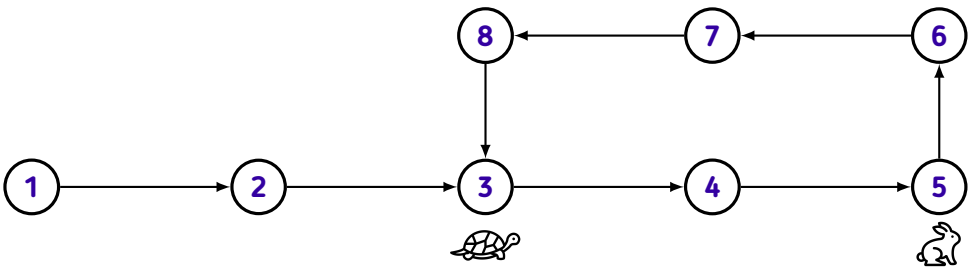
Etapa 1: Identificação do ciclo

(b) **A lebre dá dois passos, a tartagura um, até se reencontrarem**



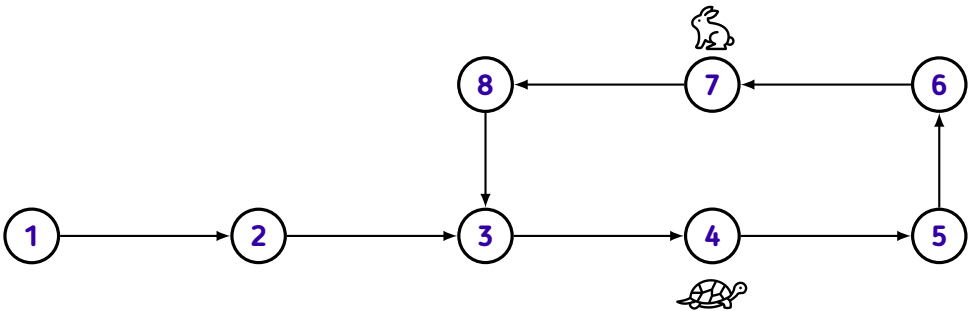
Etapa 1: Identificação do ciclo

(b) **A lebre dá dois passos, a tartagura um, até se reencontrarem**



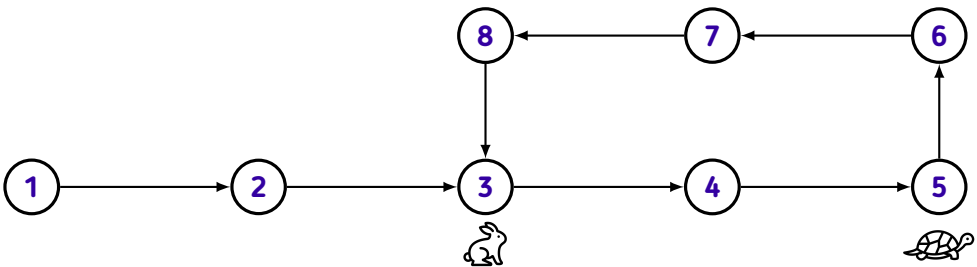
Etapa 1: Identificação do ciclo

(b) **A lebre dá dois passos, a tartagura um, até se reencontrarem**



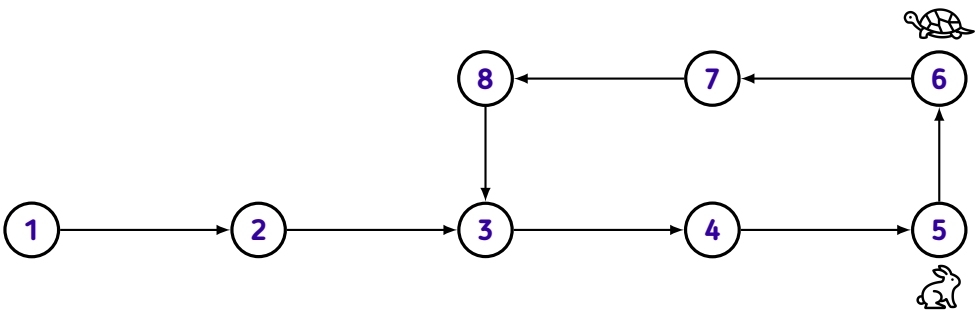
Etapa 1: Identificação do ciclo

(b) **A lebre dá dois passos, a tartagura um, até se reencontrarem**



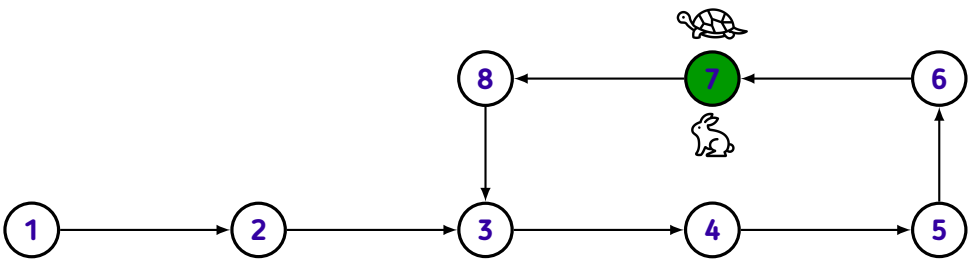
Etapa 1: Identificação do ciclo

(b) **A lebre dá dois passos, a tartagura um, até se reencontrarem**



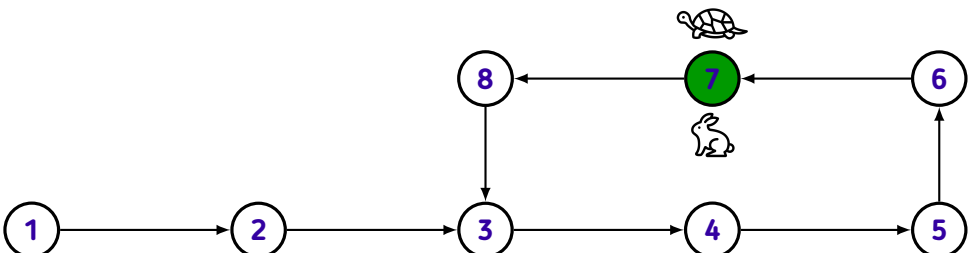
Etapa 1: Identificação do ciclo

(b) **A lebre dá dois passos, a tartagura um, até se reencontrarem**



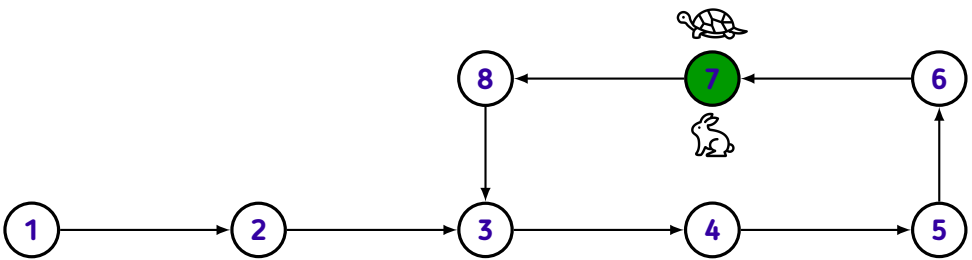
Etapa 1: Identificação do ciclo

A tartagura andou k passos e a lebre andou $2k$ passos



Etapa 1: Identificação do ciclo

Logo, λ divide k

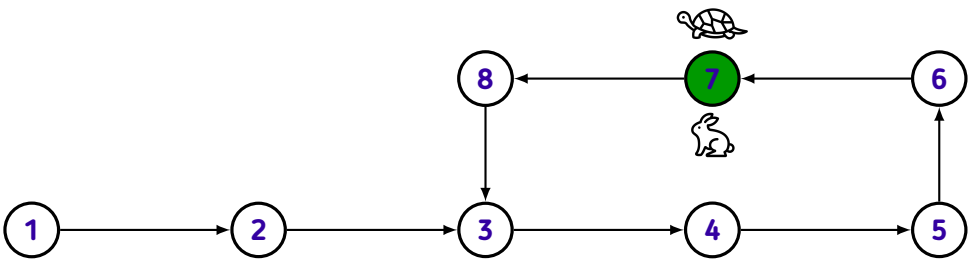


```cpp
pair<int, int> floyd(const vector<int>& succ)
{
    // Etapa 1: H é a lebre, T a tartaruga
    int L = 1, H = 1;

    do
    {
        L = succ[L];
        H = succ[succ[H]];
    } while (L != H);
```

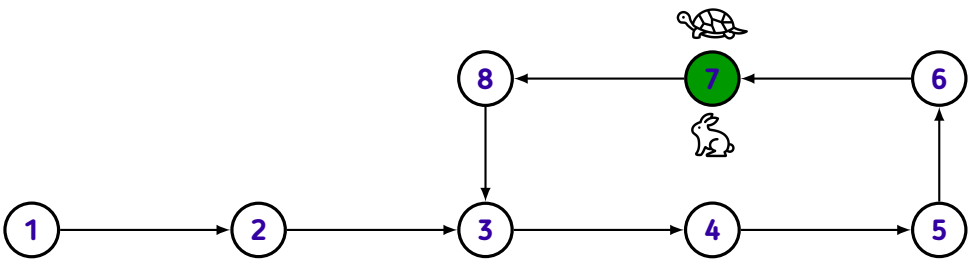
Etapa 2: Encontrando μ

Etapa 2: Encontrando μ



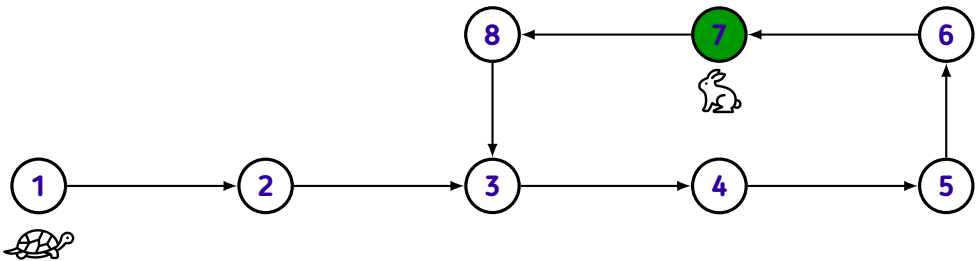
Etapa 2: Encontrando μ

(a) A tartaruga volta para o ponto de partida



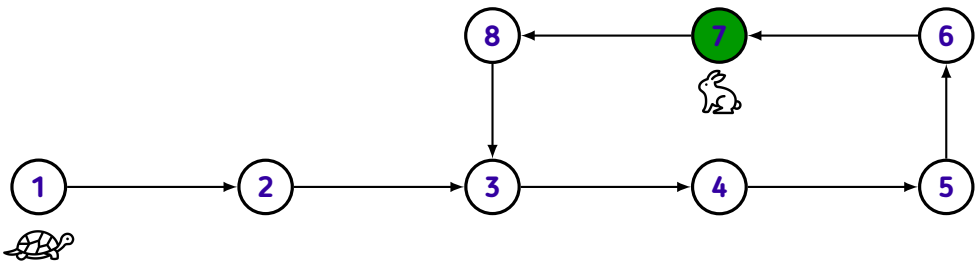
Etapa 2: Encontrando μ

(a) A tartaruga volta para o ponto de partida



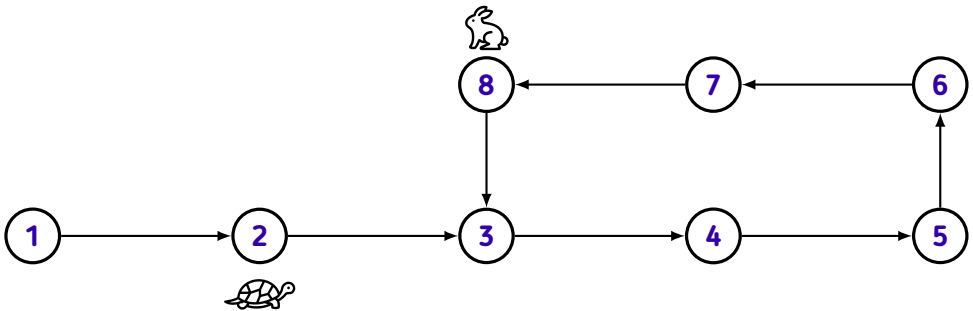
Etapa 2: Encontrando μ

(b) **Agora ambos andam um passo por vez até se encontrarem**



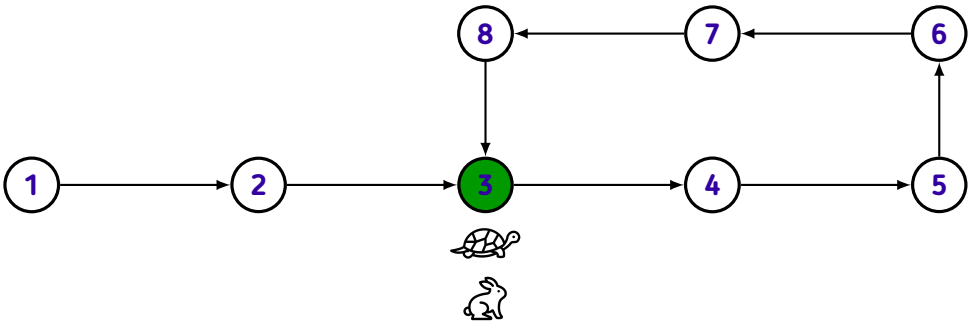
Etapa 2: Encontrando μ

(b) **Agora ambos andam um passo por vez até se encontrarem**



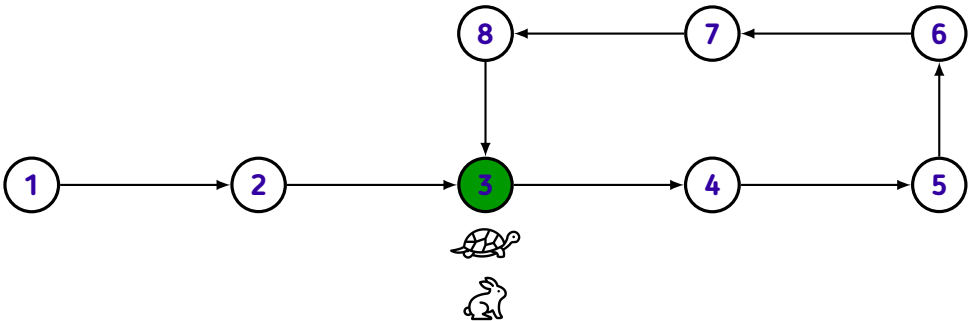
Etapa 2: Encontrando μ

(b) **Agora ambos andam um passo por vez até se encontrarem**



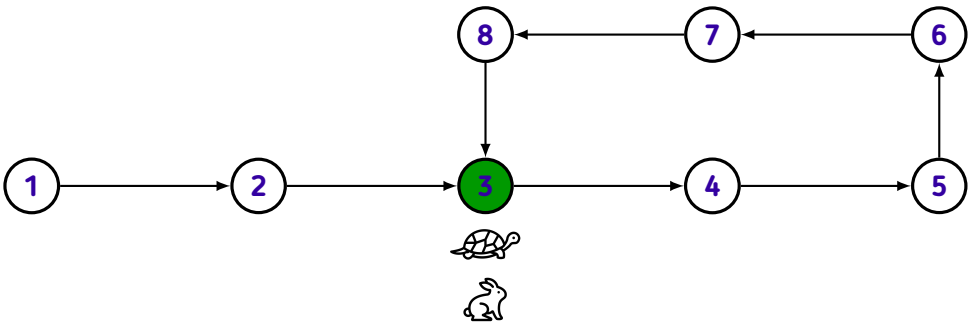
Etapa 2: Encontrando μ

O ponto de encontro é o primeiro nó do ciclo



Etapa 2: Encontrando μ

Logo, $\mu = 3$



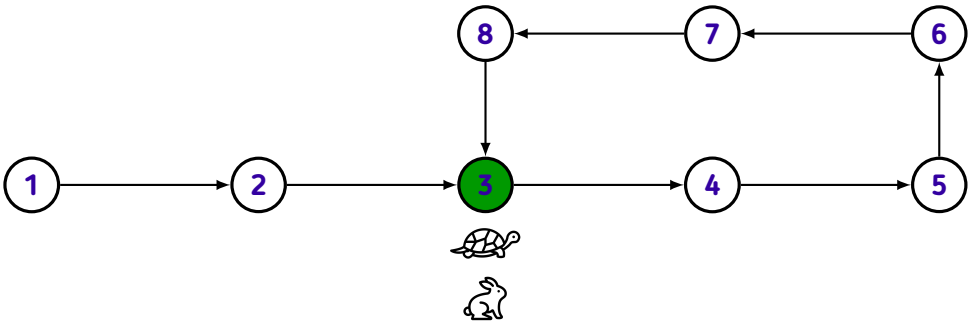
```
    L = 1;

    while (L != H)
    {
        L = succ[L];
        H = succ[H];
    }

    auto mu = L;
```

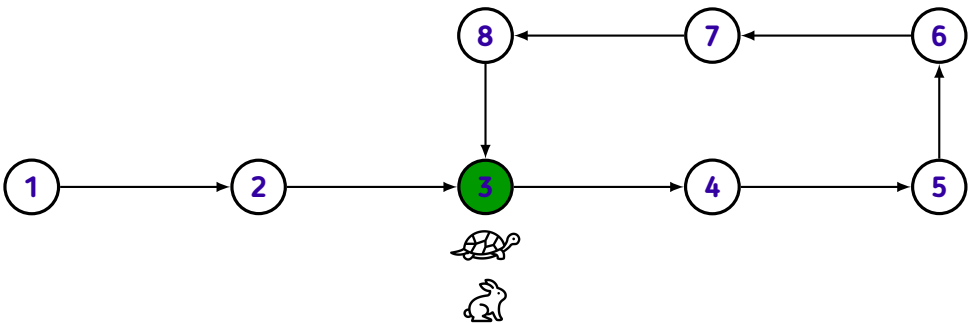
Etapa 3: Encontrando λ

Etapa 3: Encontrando λ



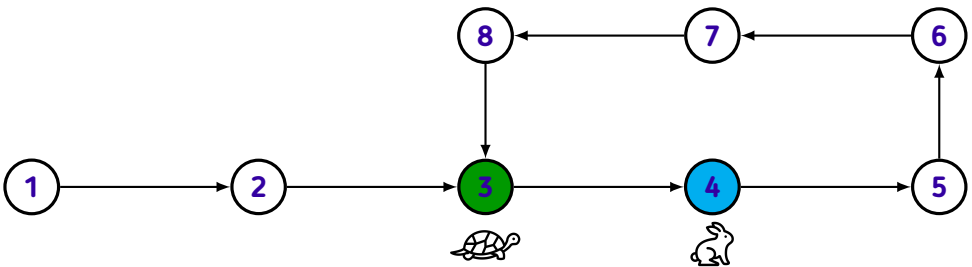
Etapa 3: Encontrando λ

(a) A tartaruga espera enquanto a lebre percorre o ciclo



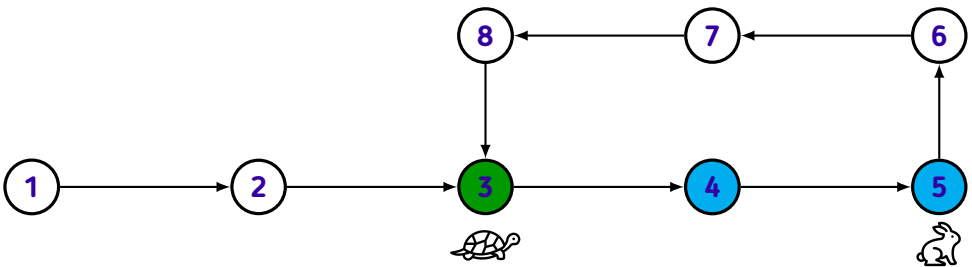
Etapa 3: Encontrando λ

(a) A tartaruga espera enquanto a lebre percorre o ciclo



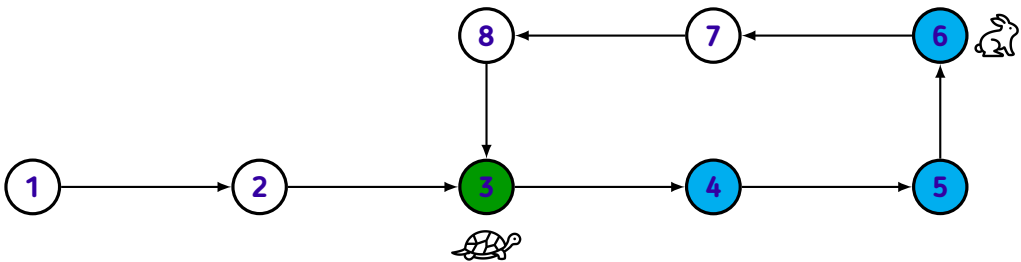
Etapa 3: Encontrando λ

(a) **A tartaruga espera enquanto a lebre percorre o ciclo**



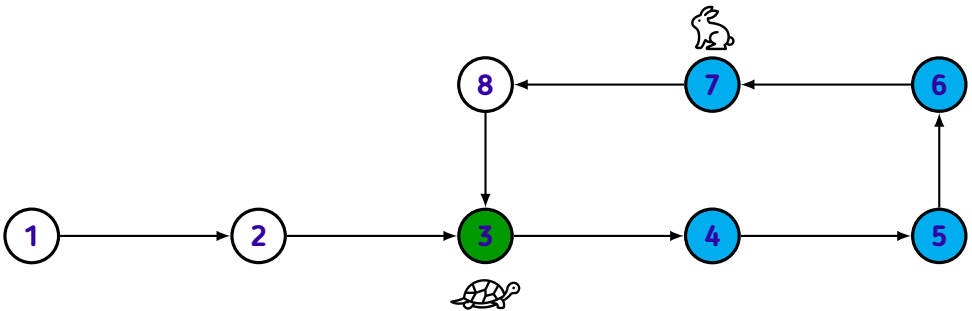
Etapa 3: Encontrando λ

(a) A tartaruga espera enquanto a lebre percorre o ciclo



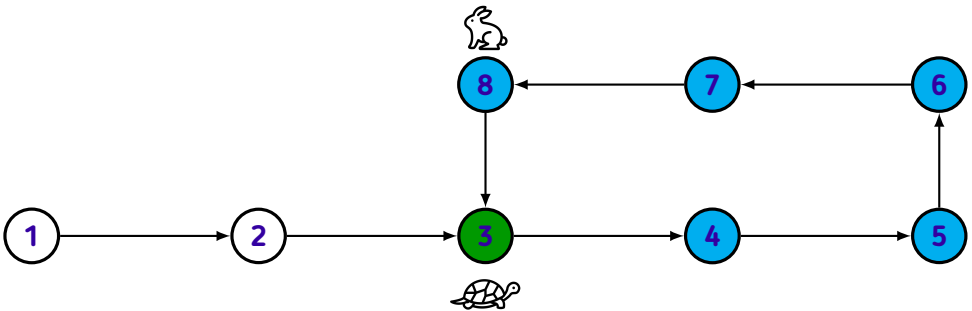
Etapa 3: Encontrando λ

(a) **A tartaruga espera enquanto a lebre percorre o ciclo**



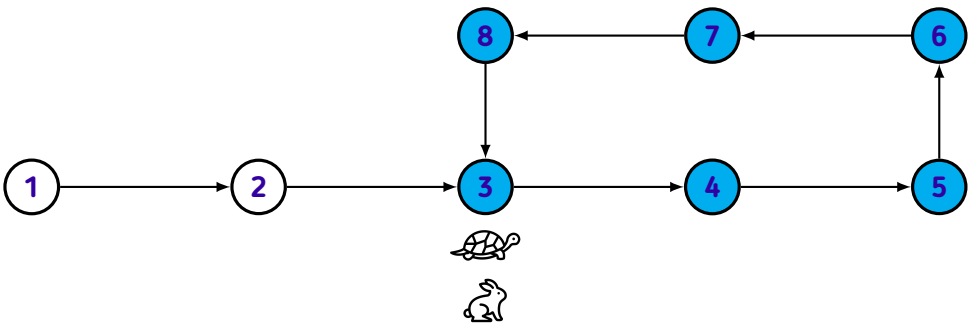
Etapa 3: Encontrando λ

(a) A tartaruga espera enquanto a lebre percorre o ciclo



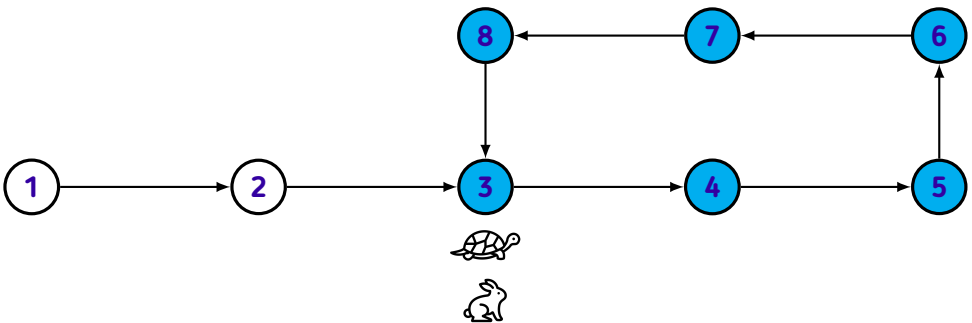
Etapa 3: Encontrando λ

(a) A tartaruga espera enquanto a lebre percorre o ciclo



Etapa 3: Encontrando λ

(b) λ será igual ao número de passos da lebre



```
    auto lambda = 0;

    do {
        H = succ[H];
        ++lambda;
    } while (L != H);

    return { mu, lambda };
}
```

Referências

1. **HALIM**, **Felix**; **HALIM**, **Steve**. ***Competitive Programming 3*, 2010.**

2. **LAAKSONEN**, **Antti**. ***Competitive Programmer's Handbook*, 2018.**

Créditos

- Tortoise by Zohaib Bajwa from Noun Project (CC BY 3.0)
- Rabbit by ARIS ARISA from Noun Project (CC BY 3.0)