Algoritmos

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

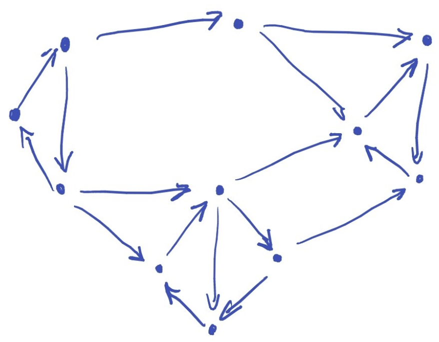
Qualquer erro é de minha responsabilidade.

Componentes fortemente conexas - Strongly Connected Components

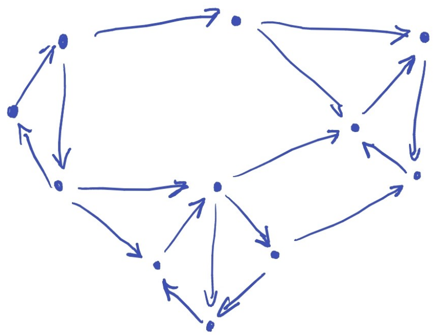
Componentes fortemente conexas - SCC

Definição

Componentes Fortemente Conexas de um Grafo Direcionado são classes de equivalência da relação $u \sim v \iff$ existe um $u - v$ caminho, e um $v - u$ caminho em G. ($\sim$ é uma relação de equivalência)



DFS para encontrar as SCCs?



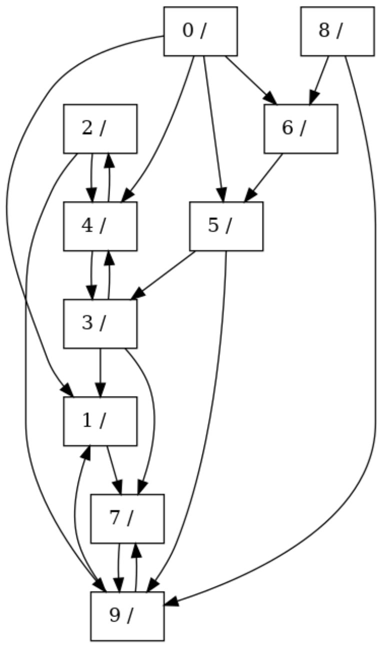
Algoritmo de Kosaraju

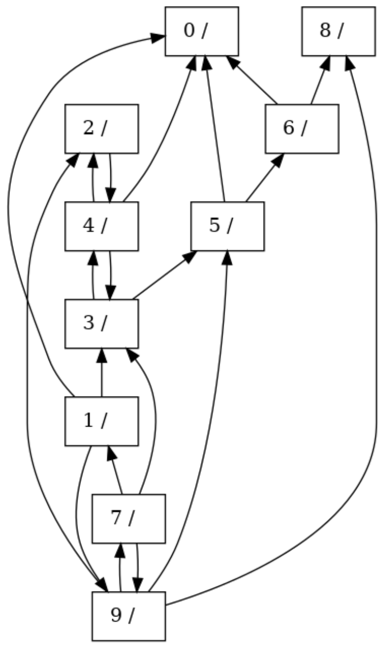
O algoritmo de Kosaraju faz duas passadas no DFS (com pequenas modificações), tem complexidade $O(|V| + |E|)$.
Ideia do algoritmo:

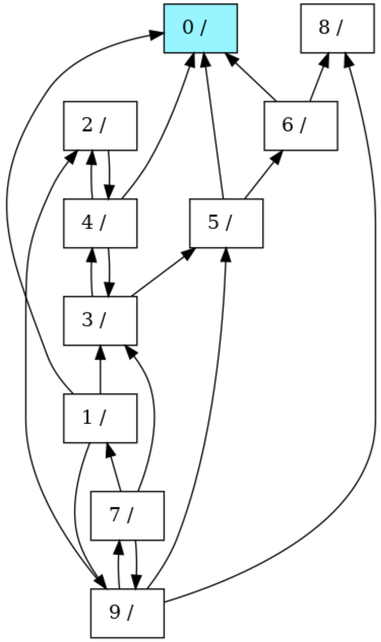
- Reverter todos os arcos de G e chamar de G^{rev}
- DFS-Loop em G^{rev} (salvando os tempos de termino)
- DFS-Loop em G (na ordem decrescente de tempos de termino da primeira passada)

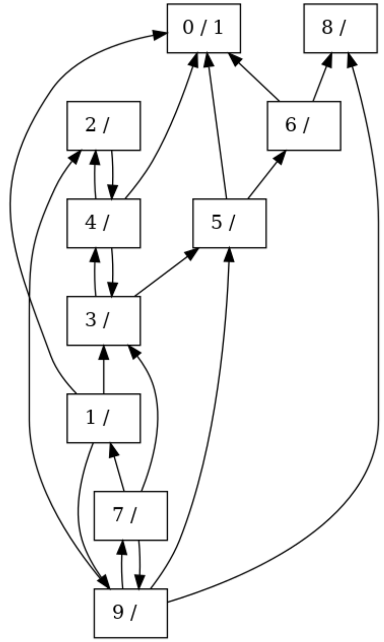
Observações:

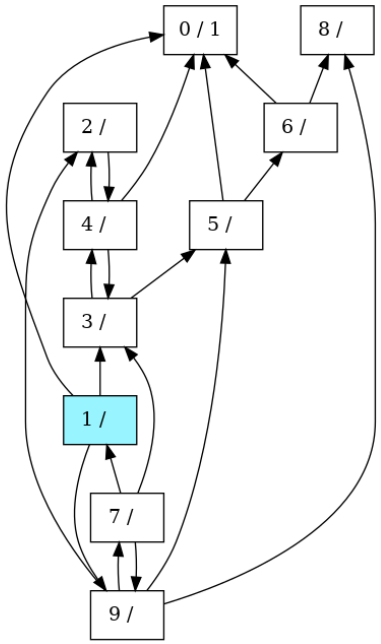
- Computaremos lideres para cada vértice, vértices com o mesmo líder estarão na mesma SCC.

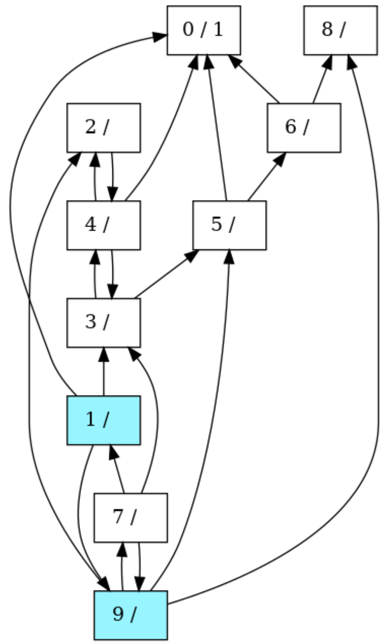


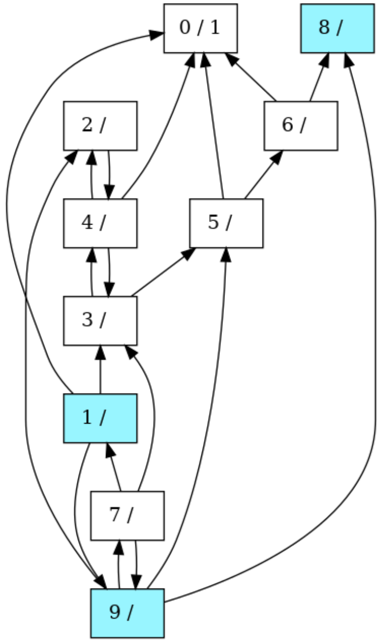


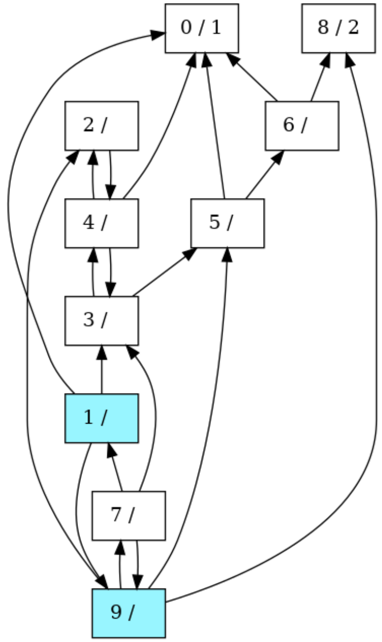

0 /
8 /
2 /
6 /
4 /
5 /
3 /
1 /
7 /
9 /

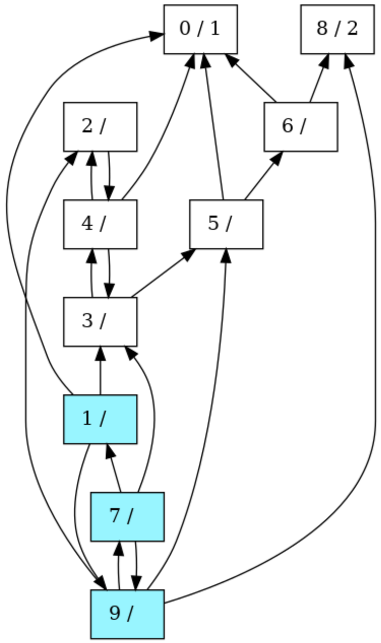

0 / 1
8 /
2 /
6 /
4 /
5 /
3 /
1 /
7 /
9 /

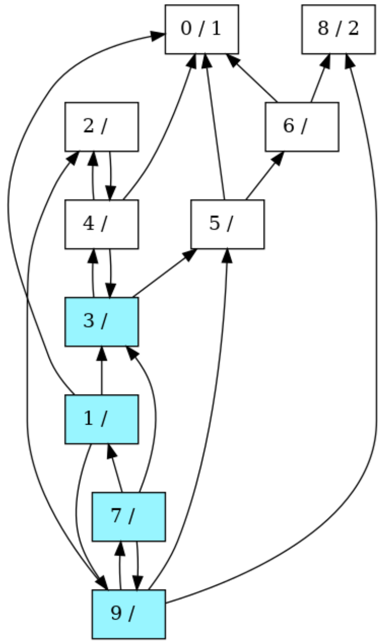

0 / 1
8 /
2 /
6 /
4 /
5 /
3 /
1 /
7 /
9 /

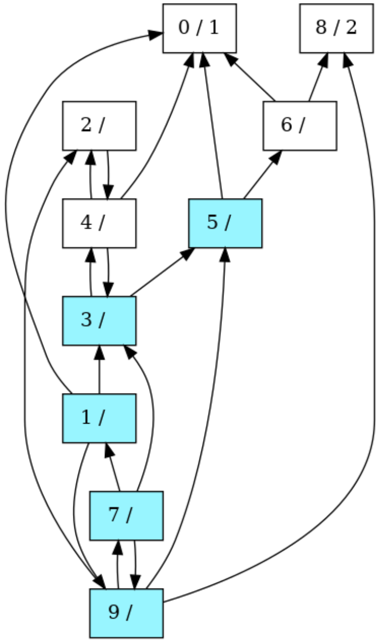

0 / 1
8 /
2 /
6 /
4 /
5 /
3 /
1 /
7 /
9 /

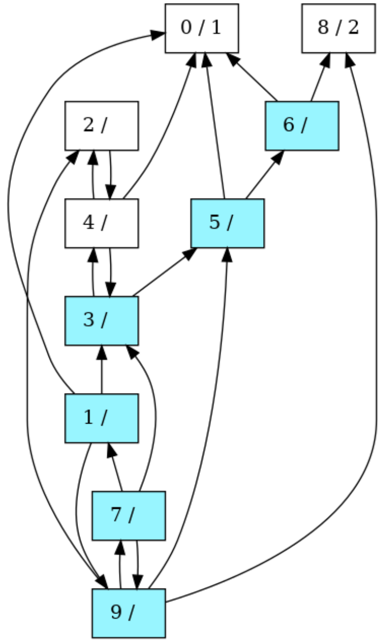

0 / 1
8 /
2 /
6 /
4 /
5 /
3 /
1 /
7 /
9 /

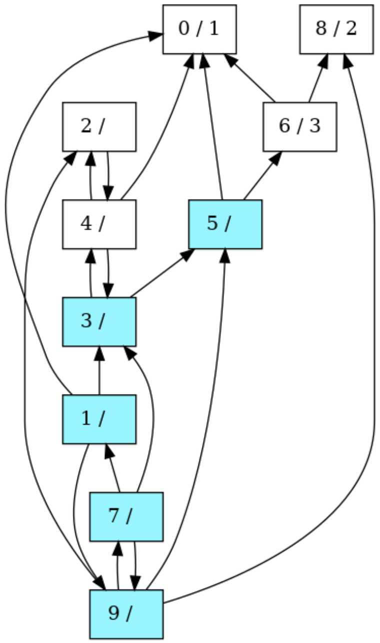

0 / 1
8 / 2
2 /
6 /
4 /
5 /
3 /
1 /
7 /
9 /

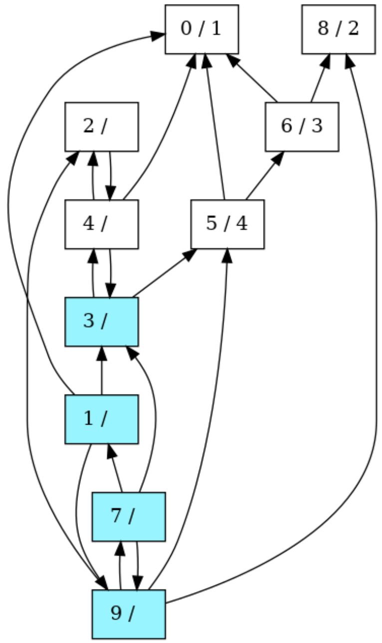

0 / 1
8 / 2
2 /
6 /
4 /
5 /
3 /
1 /
7 /
9 /

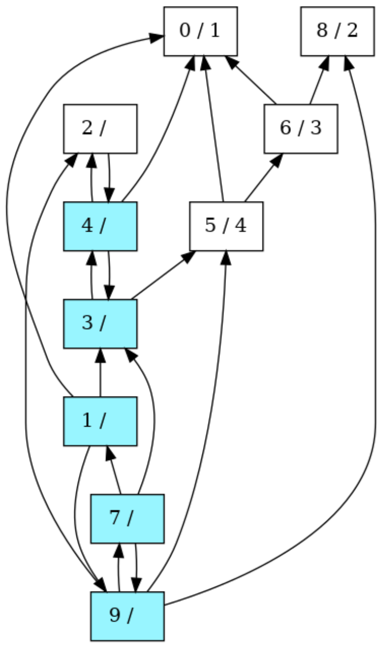

0 / 1
8 / 2
2 /
6 /
4 /
5 /
3 /
1 /
7 /
9 /

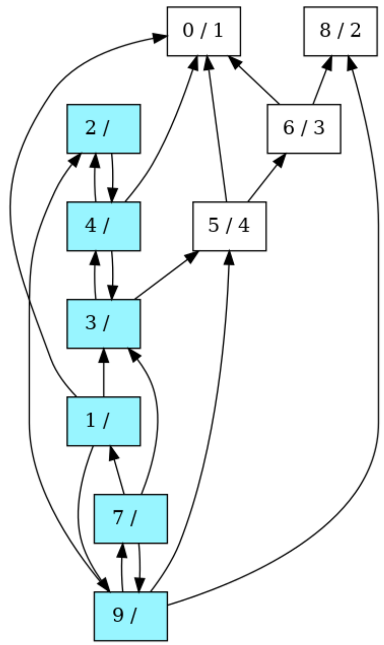

0 / 1
8 / 2
2 /
6 /
4 /
5 /
3 /
1 /
7 /
9 /

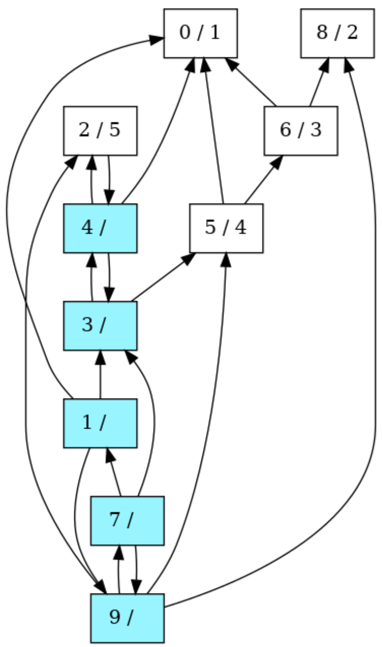

0 / 1
8 / 2
2 /
6 /
4 /
5 /
3 /
1 /
7 /
9 /

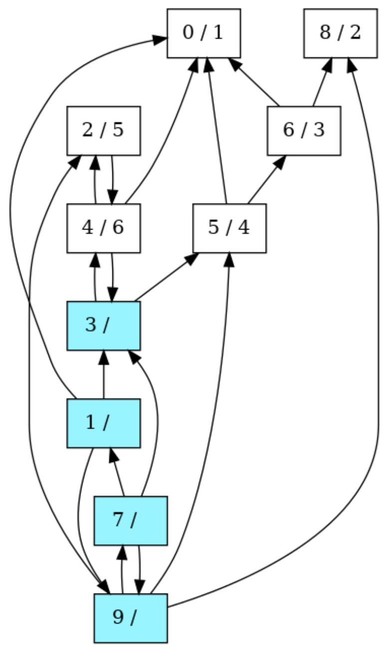

0 / 1
8 / 2
2 /
6 / 3
4 /
5 /
3 /
1 /
7 /
9 /

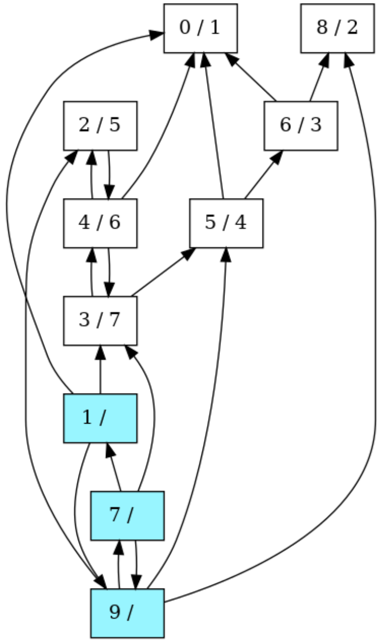

0 / 1
8 / 2
2 /
6 / 3
4 /
5 / 4
3 /
1 /
7 /
9 /

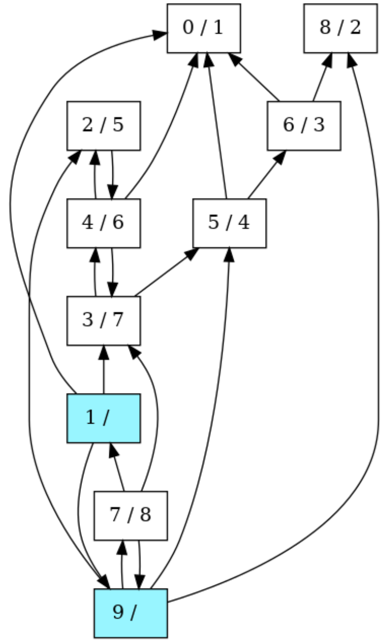

0 / 1
8 / 2
2 /
6 / 3
4 /
5 / 4
3 /
1 /
7 /
9 /

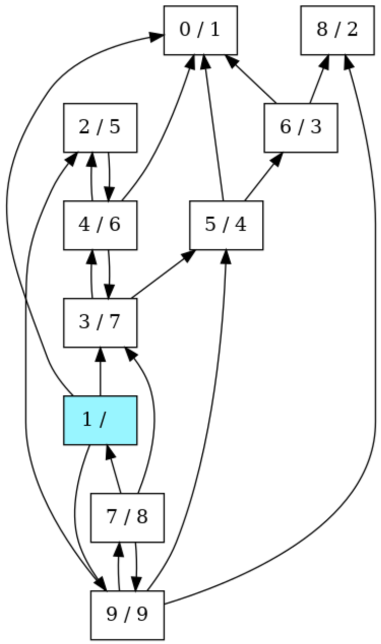

0 / 1
8 / 2
2 /
6 / 3
4 /
5 / 4
3 /
1 /
7 /
9 /

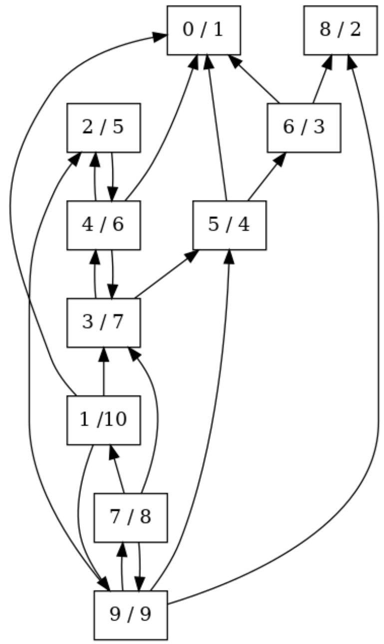

0 / 1
8 / 2
2 / 5
6 / 3
4 /
5 / 4
3 /
1 /
7 /
9 /

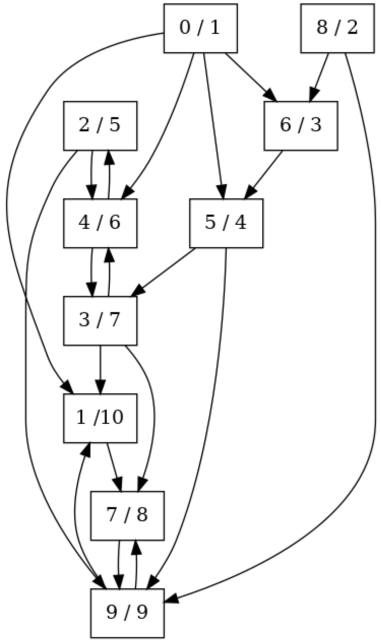

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 /
1 /
7 /
9 /

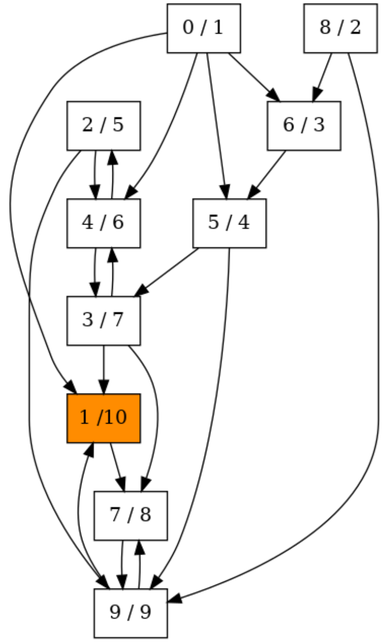

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /
7 /
9 /

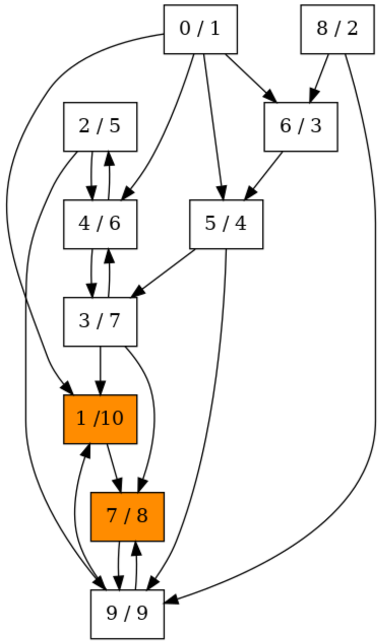

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /
7 / 8
9 /

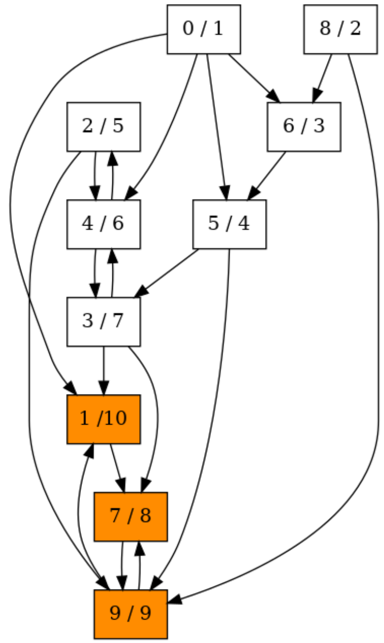

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /
7 / 8
9 / 9

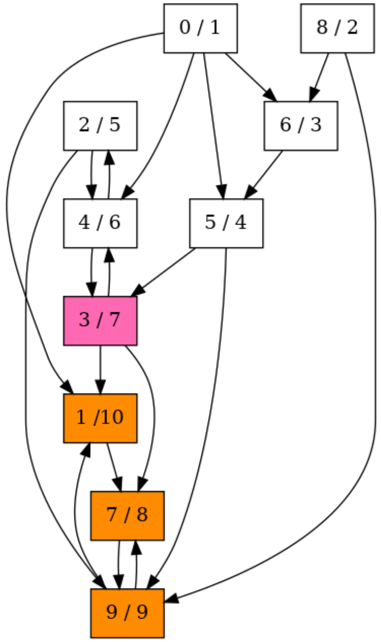

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9

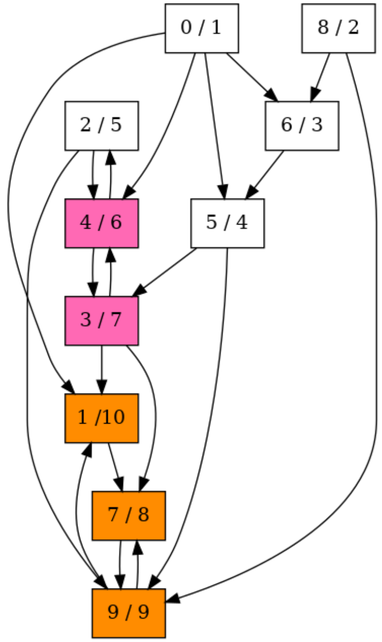

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9

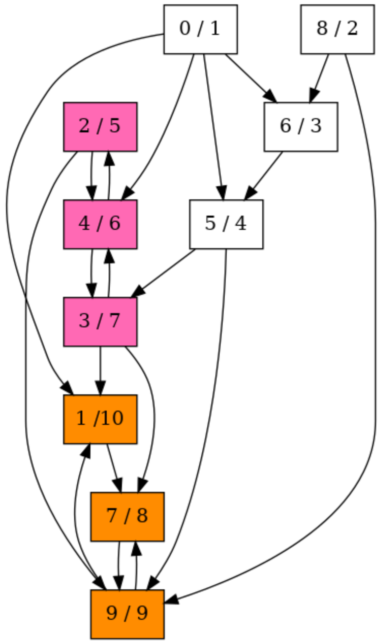

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9

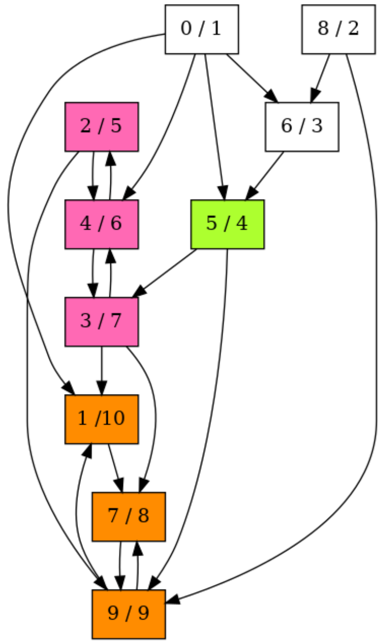

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9

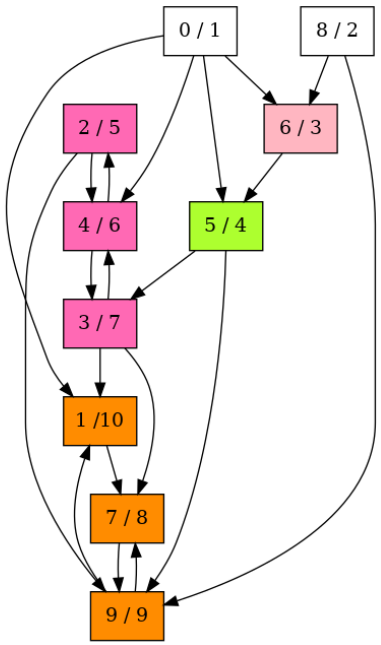

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9

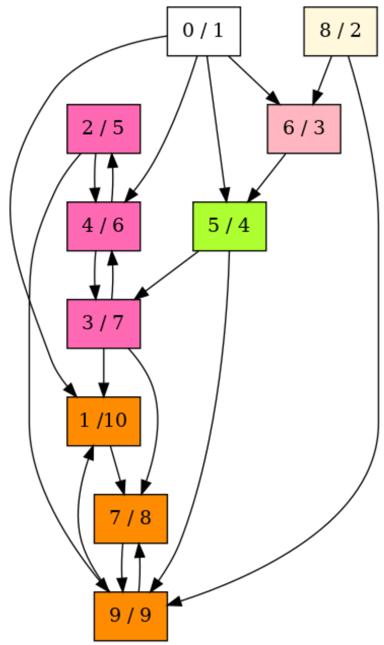

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9

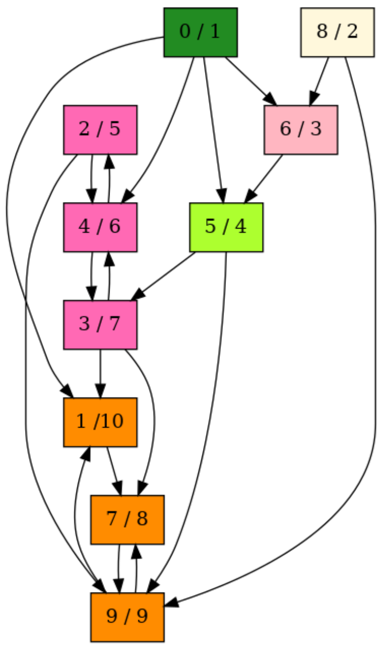

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9

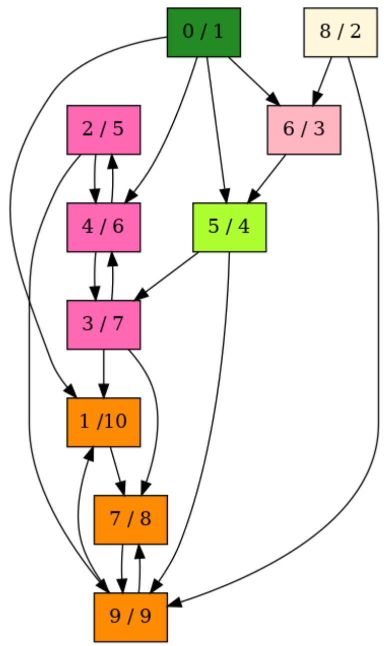

0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9


0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9


0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9


0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9


0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9


0 / 1
8 / 2
2 / 5
6 / 3
4 / 6
5 / 4
3 / 7
1 /10
7 / 8
9 / 9

Algoritmo 1: DFS-Loop

Entrada: Um Grafo G

Marcar $v \in V$ como não-visitado;
$T = 0$;
$L = NULL$;
Assuma que os vértices estão rotulados de 1 até n;
para $v = n$ *até* 1 **faça**
    se v *está não-visitado* **então**
        $L = v$;
        $DFS(G, v)$;

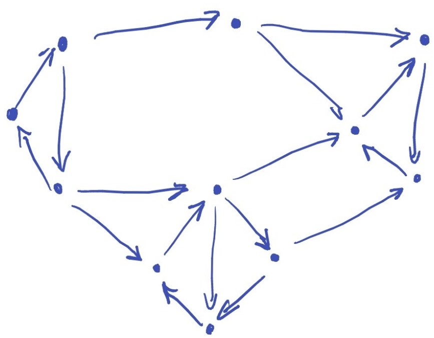
Algoritmo 2: DFS

Entrada: Um Grafo G, e um vértice fonte s

Marcar s como visitado ;
$lider(s) = L$;
para *todo arco* (s, v) **faça**
    se v *é não visitado* **então**
        $DFS(G, v)$;
$T{+}{+}$;
$f(s) = T$;

Corretude do Kosaraju

- As SCCs de um grafo direcionado G correspondem a um "Meta-Grafo"acíclico em que:
- Os meta-vértices são as SCCs $C_1, \ldots, C_k$ de G.
- Existe um arco de (C, C') $\iff$ Existe $(i, j) \in A$ com $i \in C$



e $j \in C'$.
- O grafo é acíclico, senão cada C não seria uma SCC.

Lema 1

Considere duas SCC "adjacentes"em G, sendo que existe uma aresta de C_1 para C_2. Seja f(v) = tempo de termino do DFS-Loop no G^{rev}. Então:

$$\max_{v \in C_1} f(v) < \max_{v \in C_2} f(v)$$

Prova: Seja v o primeiro vértice visitado de $C_1 \cup C_2$, na primeira passagem do algoritmo.
Note que em G^{rev} a aresta considerada está invertida.

1. Caso 1: $v \in C_1$, então todos os vértices de C_1 serão visitados antes de qualquer vértice de C_2
2. Caso 2: $v \in C_2$, eventualmente algum vértice de C_1 é alcançado e todo C_1 é explorado antes de C_2 terminar.

Corolário 1

O maior valor de f deve ficar em um SCC sorvedouro.

Corretude do Kosaraju

- Como o algoritmo começa a executar em uma SCC sorvedouro, ele encontra toda a componente sem sair dela.
- A partir dai os vértices dessa SCC não é mais considerado e a próxima SCC sorvedouro é executada.
- Dessa forma cada SCC é encontrada separadamente $\square$.