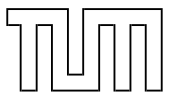
Grundlagen: Algorithmen und Datenstrukturen

Prof. Dr. Hanjo Täubig

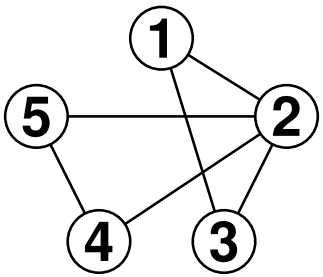
Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

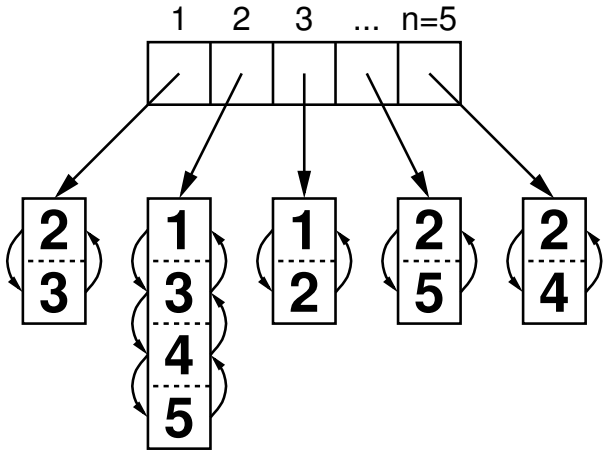
Sommersemester 2010



Übersicht

Adjazenzliste





Unterschiedliche Varianten:
einfach/doppelt verkettet, linear/zirkulär

Vorteil:

- Einfügen von Kanten in $\mathcal{O}(d)$ oder $\mathcal{O}(1)$
- Löschen von Kanten in $\mathcal{O}(d)$ (per Handle in $\mathcal{O}(1)$)
- mit unbounded arrays etwas cache-effizienter

Nachteil:

- Zeigerstrukturen verbrauchen relativ viel Platz und Zugriffszeit

Adjazenzliste + Hashtabelle

- speichere Adjazenzliste (Liste von adjazenten Nachbarn bzw. inzidenten Kanten zu jedem Knoten)
- speichere Hashtabelle, die zwei Knoten auf einen Zeiger abbildet, der dann auf die ggf. vorhandene Kante verweist

Zeitaufwand:

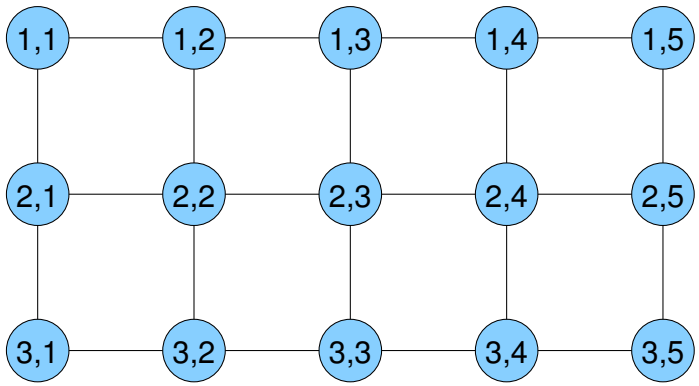
- *G*.find(Key *i*, Key *j*): $\mathcal{O}(1)$ (worst case)
- *G*.insert(Edge *e*): $\mathcal{O}(1)$ (im Mittel)
- *G*.remove(Key *i*, Key *j*): $\mathcal{O}(1)$ (im Mittel)
- Speicheraufwand: $\mathcal{O}(n+m)$

Implizite Repräsentation

Beispiel: Gitter-Graph (grid graph)

- definiert durch zwei Parameter k und ℓ

$$
\begin{aligned}
V &= [1,\ldots,k] \times [1,\ldots,\ell] \\
E &= \Big\{((i,j),(i,j')) \in V^2 : |j-j'| = 1\Big\} \cup \\
&\quad \Big\{((i,j),(i',j)) \in V^2 : |i-i'| = 1\Big\}
\end{aligned}
$$



- Kantengewichte könnten in 2 zweidimensionalen Arrays gespeichert werden:
  eins für waagerechte und eins für senkrechte Kanten

Graphtraversierung

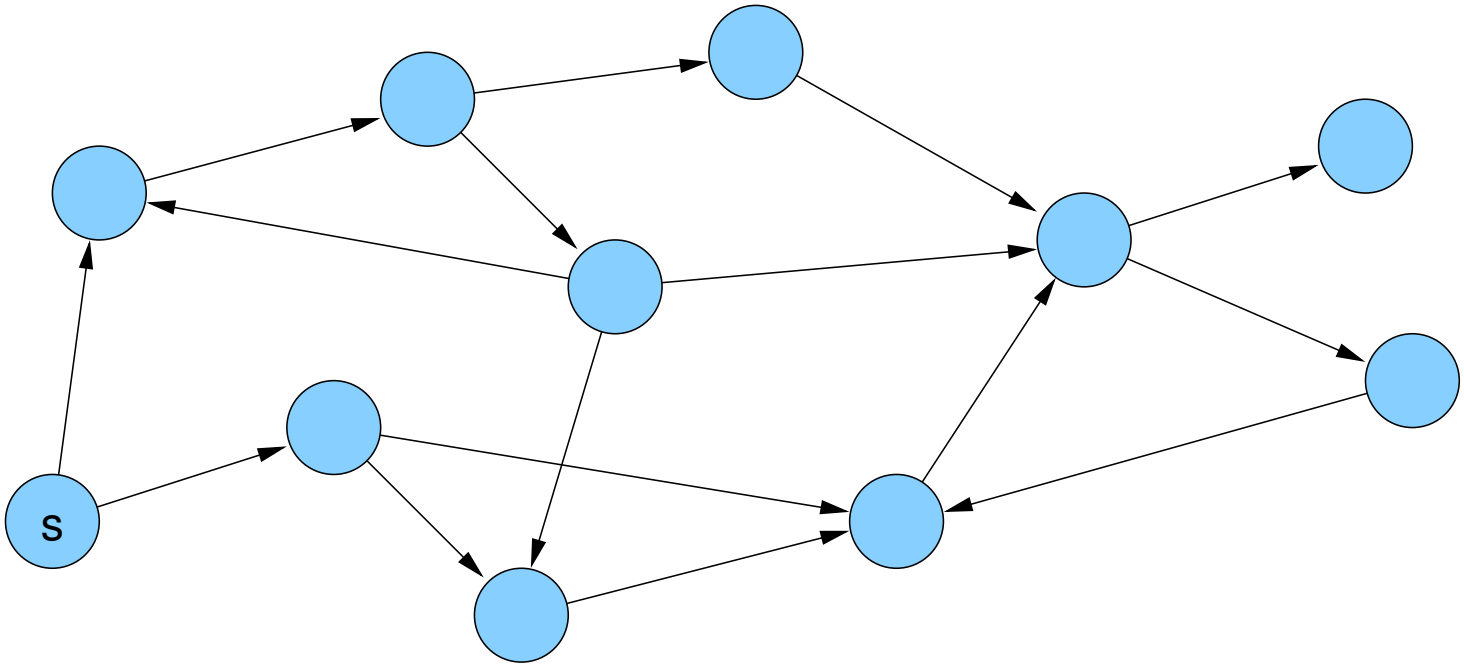
Problem:

Wie kann man die Knoten eines Graphen systematisch durchlaufen?

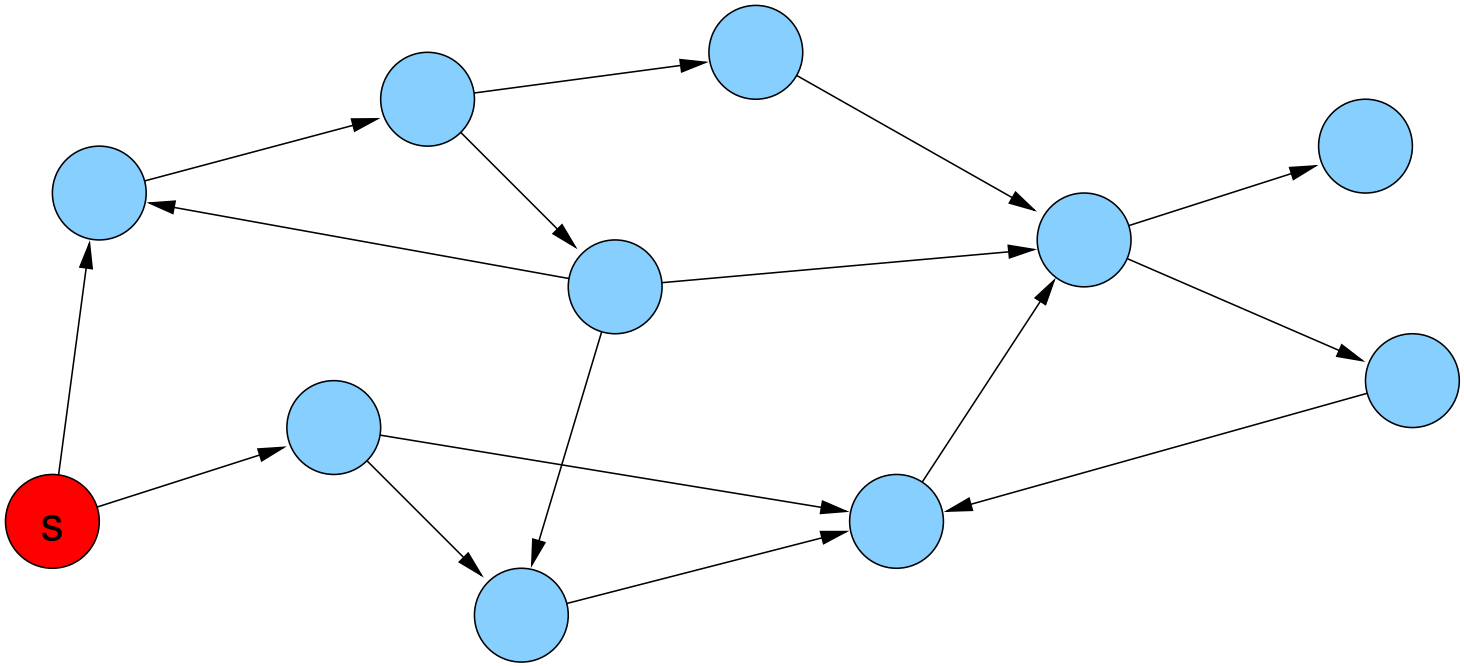
Grundlegende Strategien:

- Breitensuche (breadth-first search, BFS)
- Tiefensuche (depth-first search, DFS)

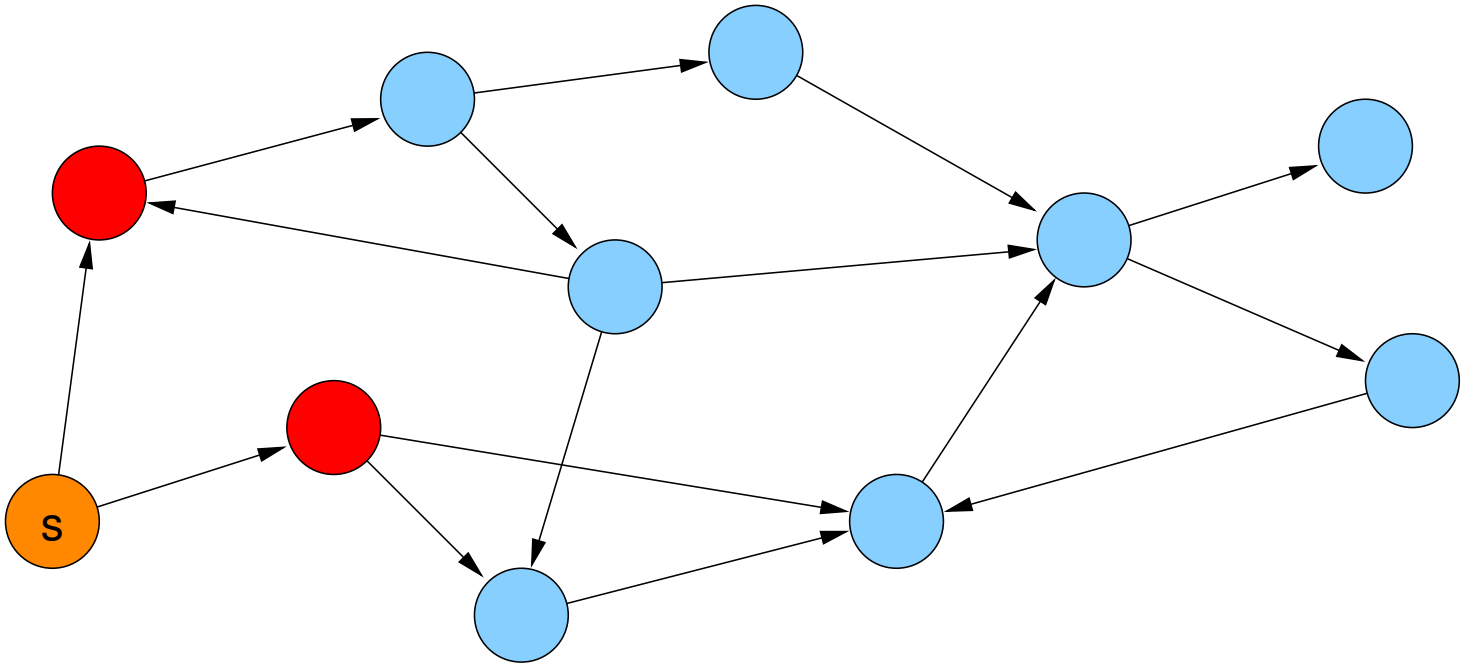
Breitensuche



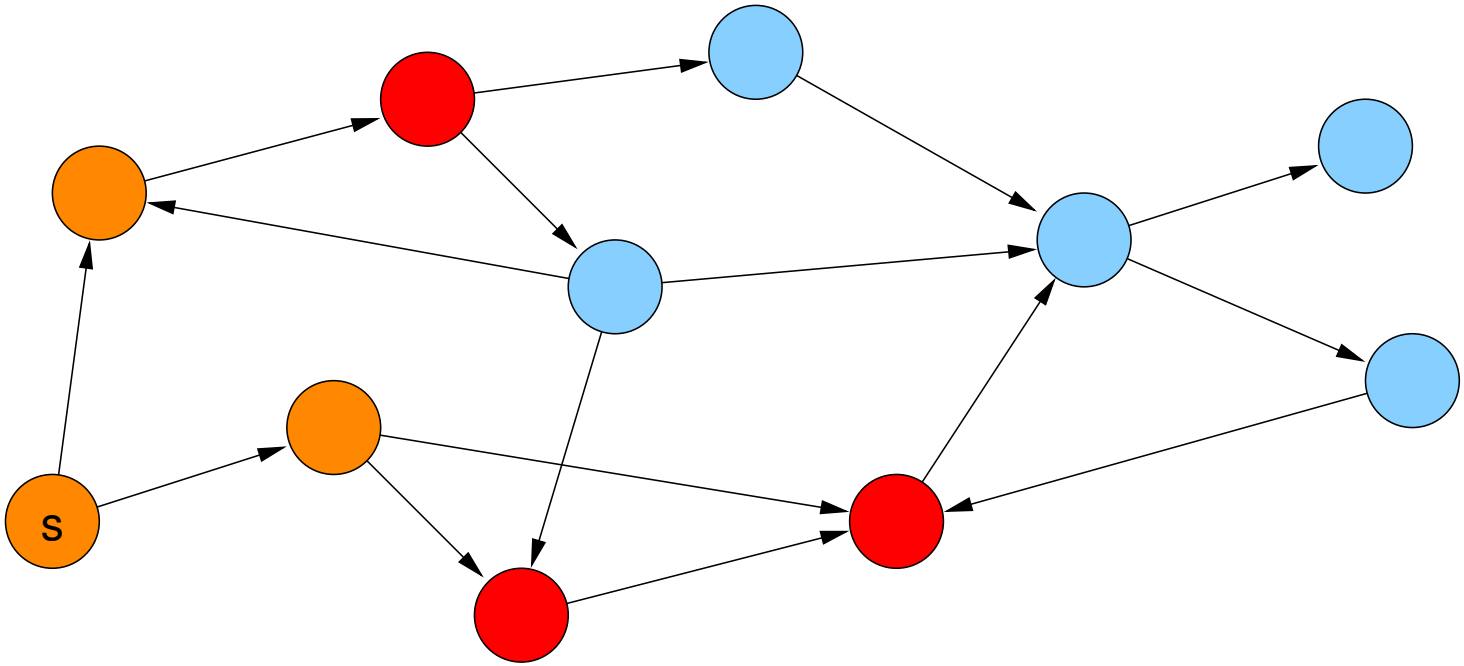
Breitensuche



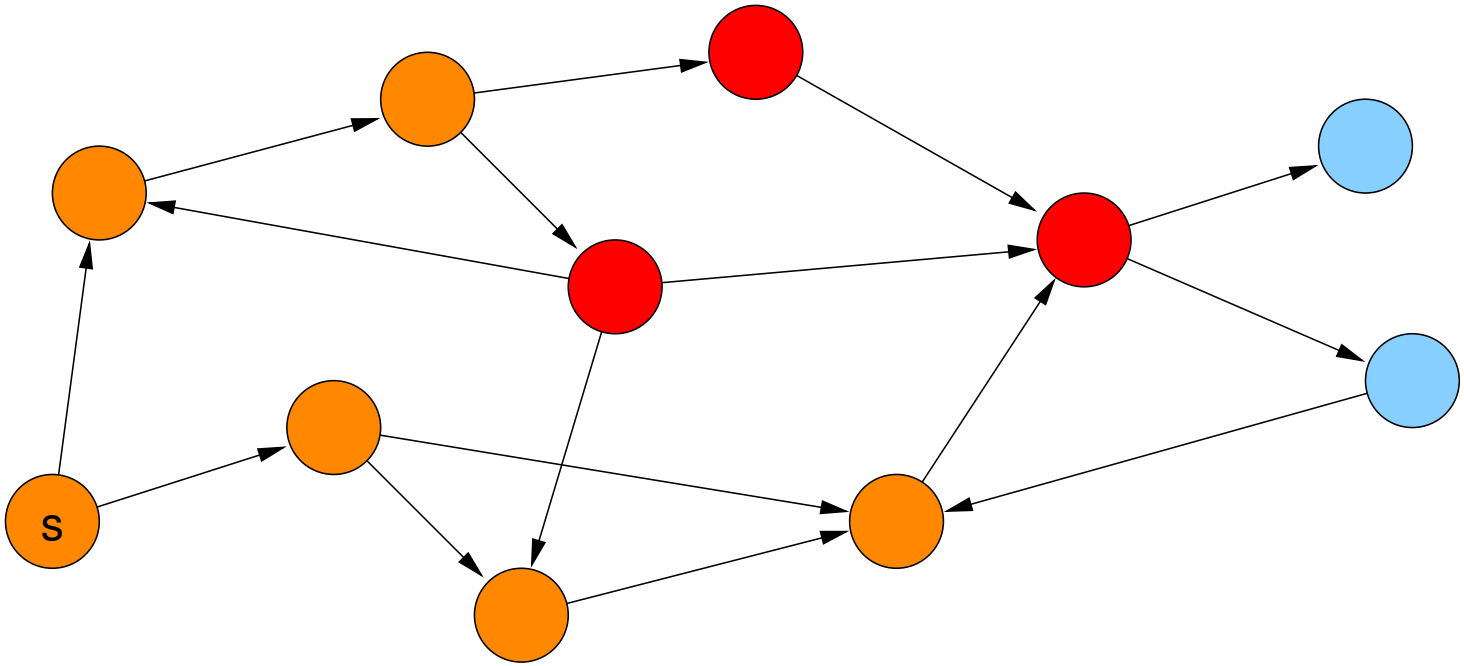
Breitensuche



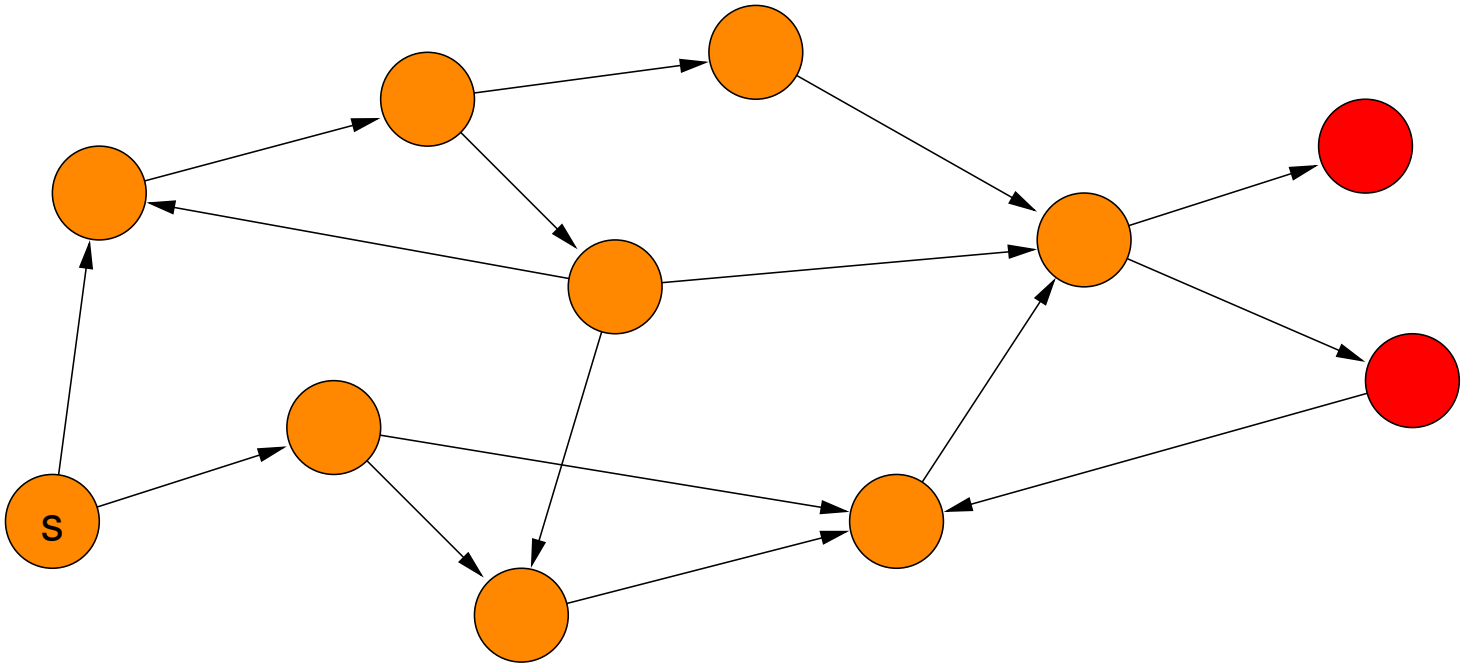
Breitensuche



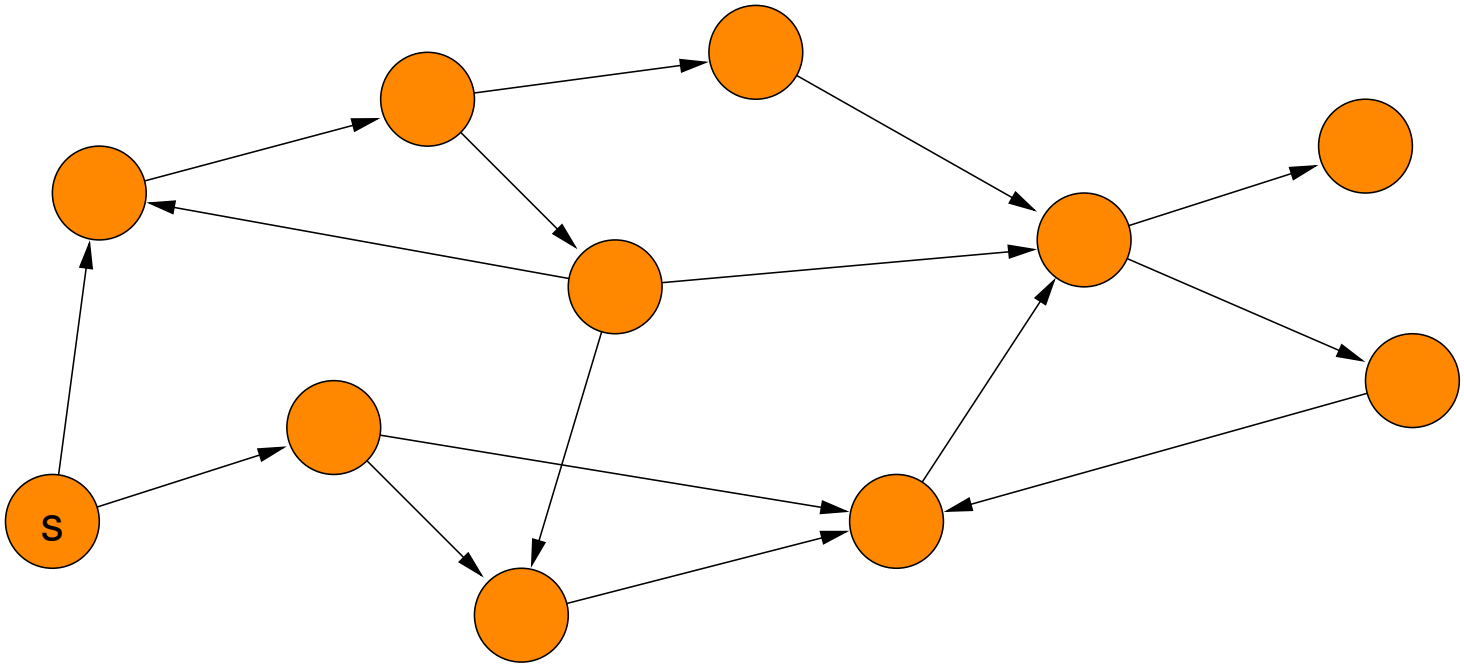
Breitensuche



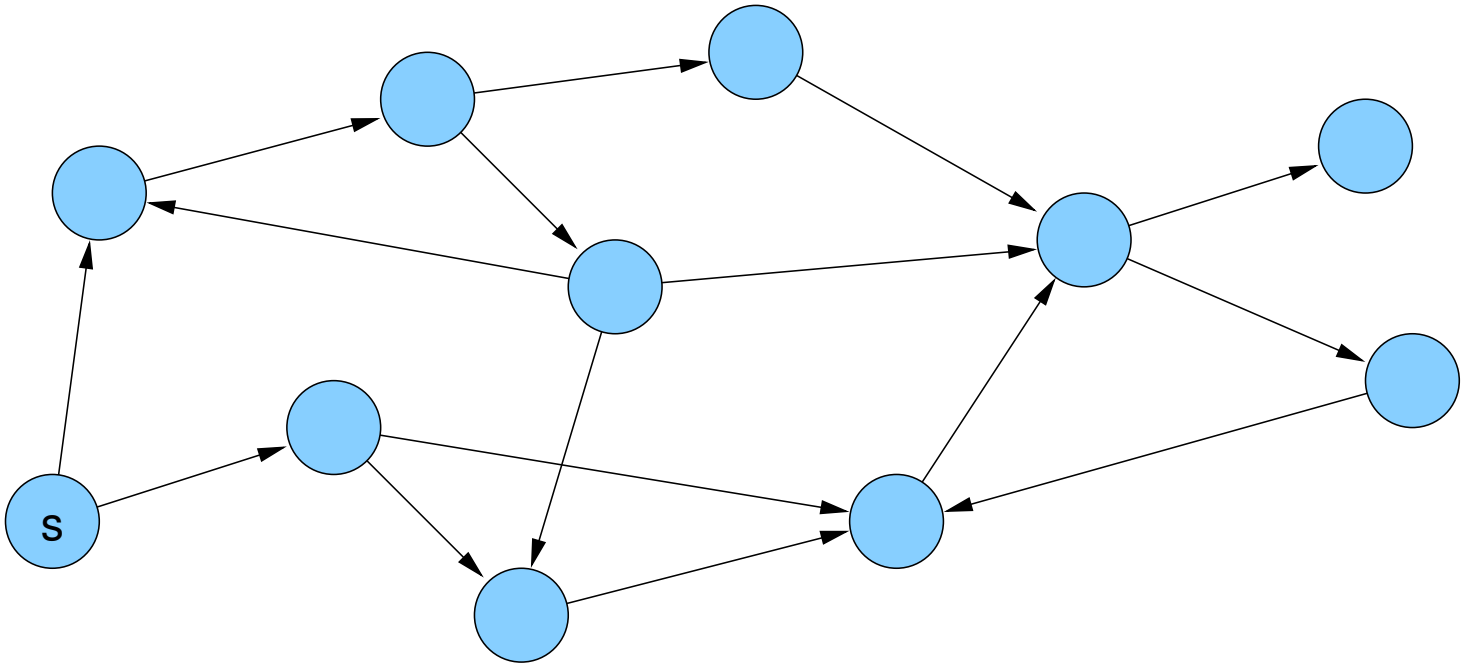
Breitensuche



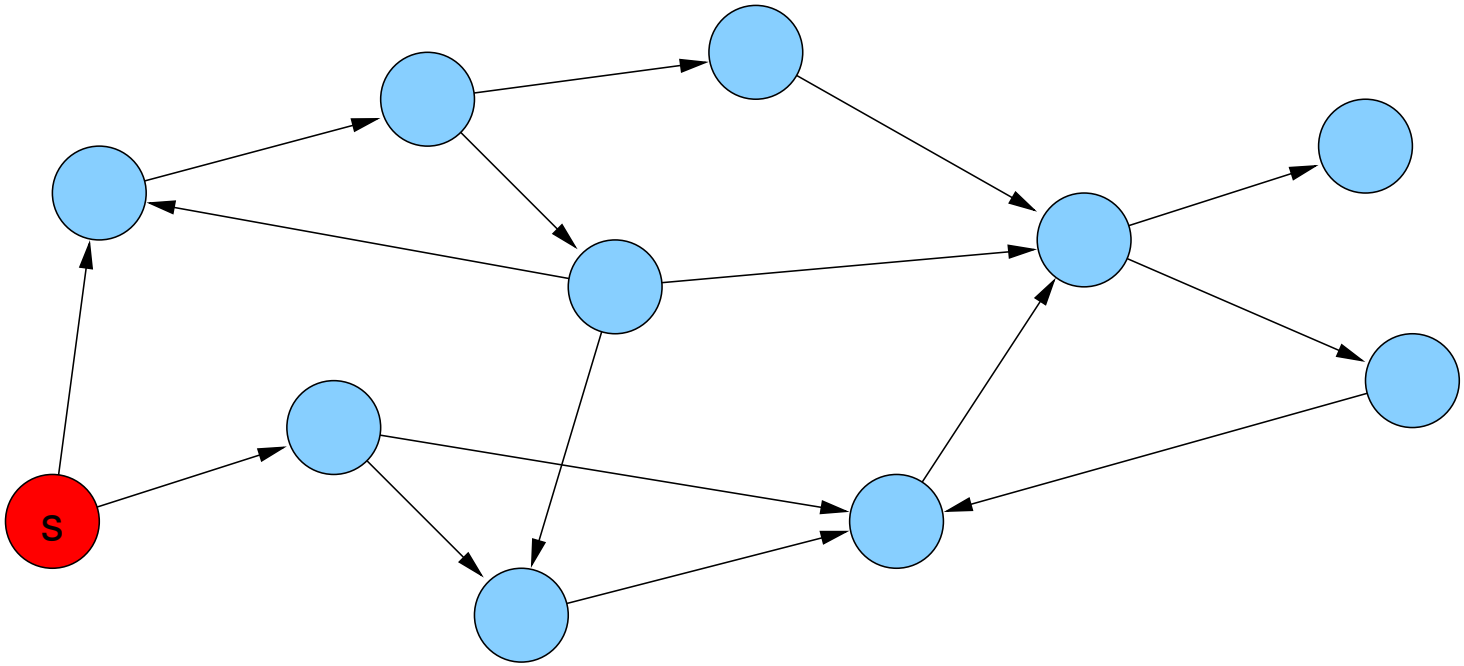
Breitensuche



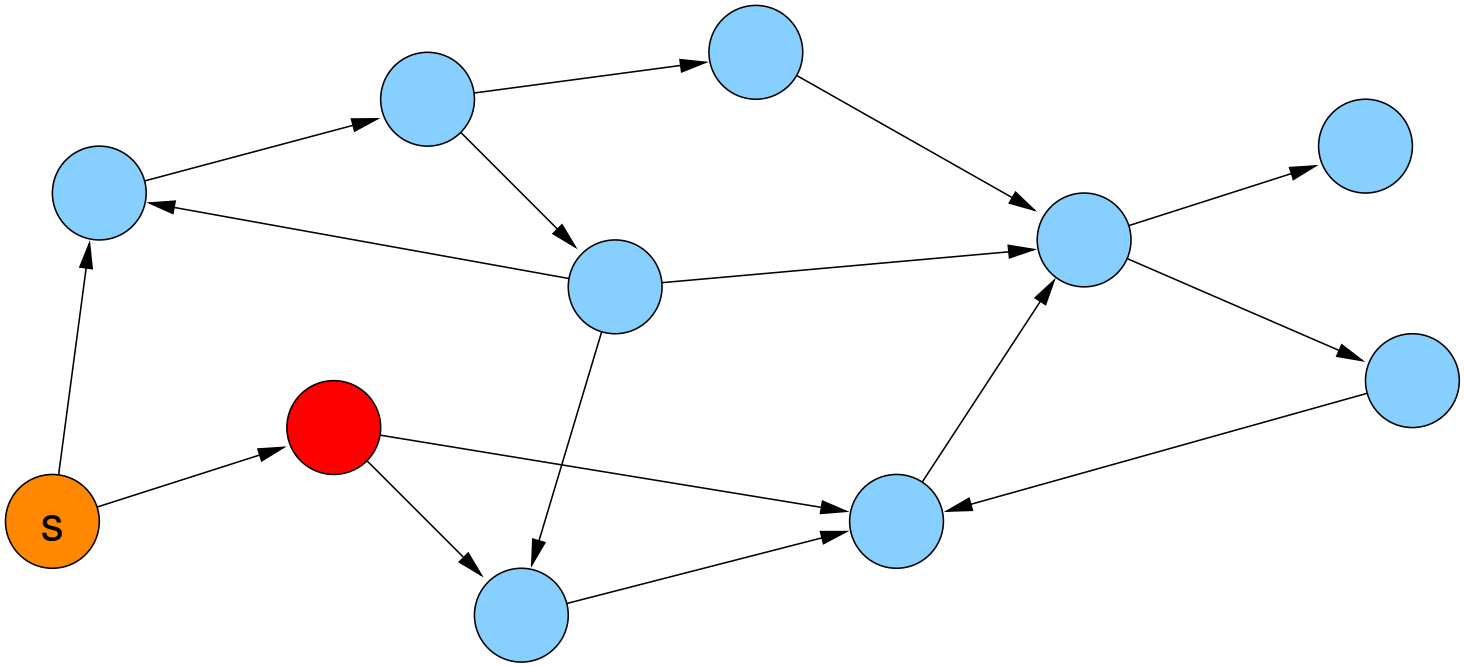
Tiefensuche



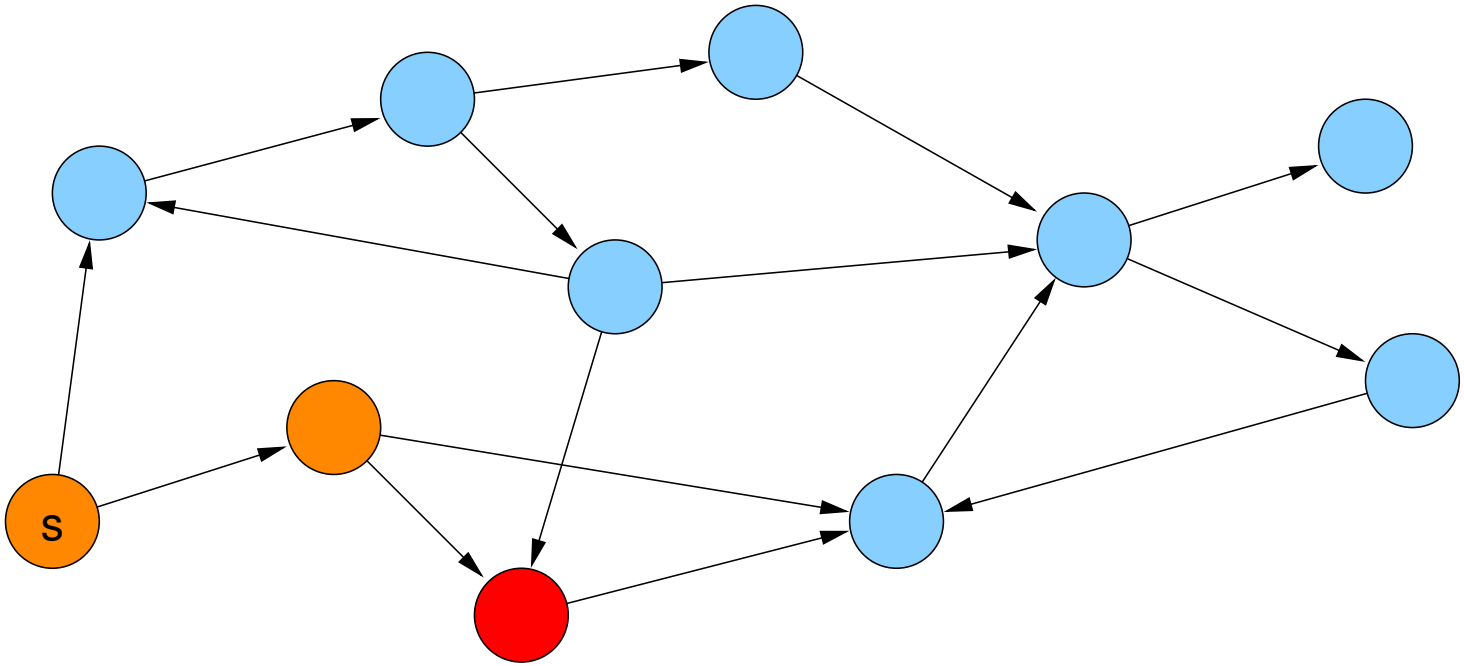
Tiefensuche



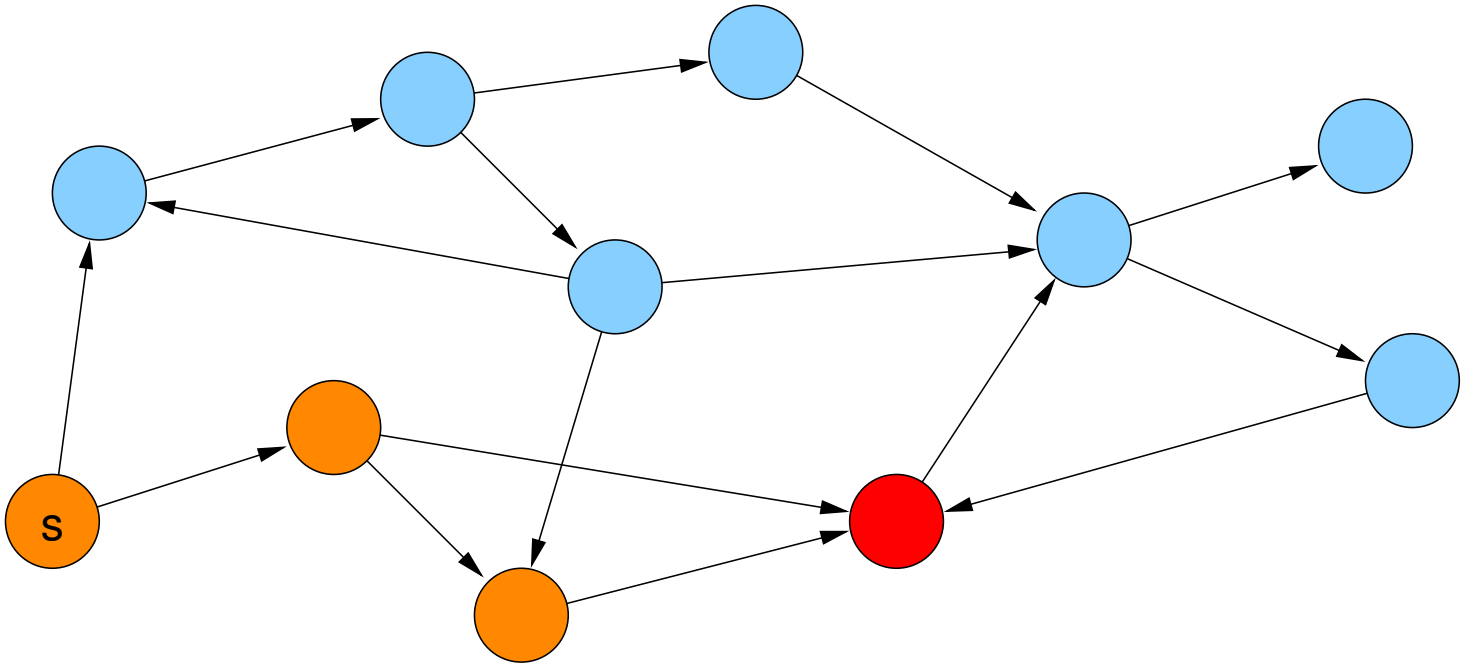
Tiefensuche



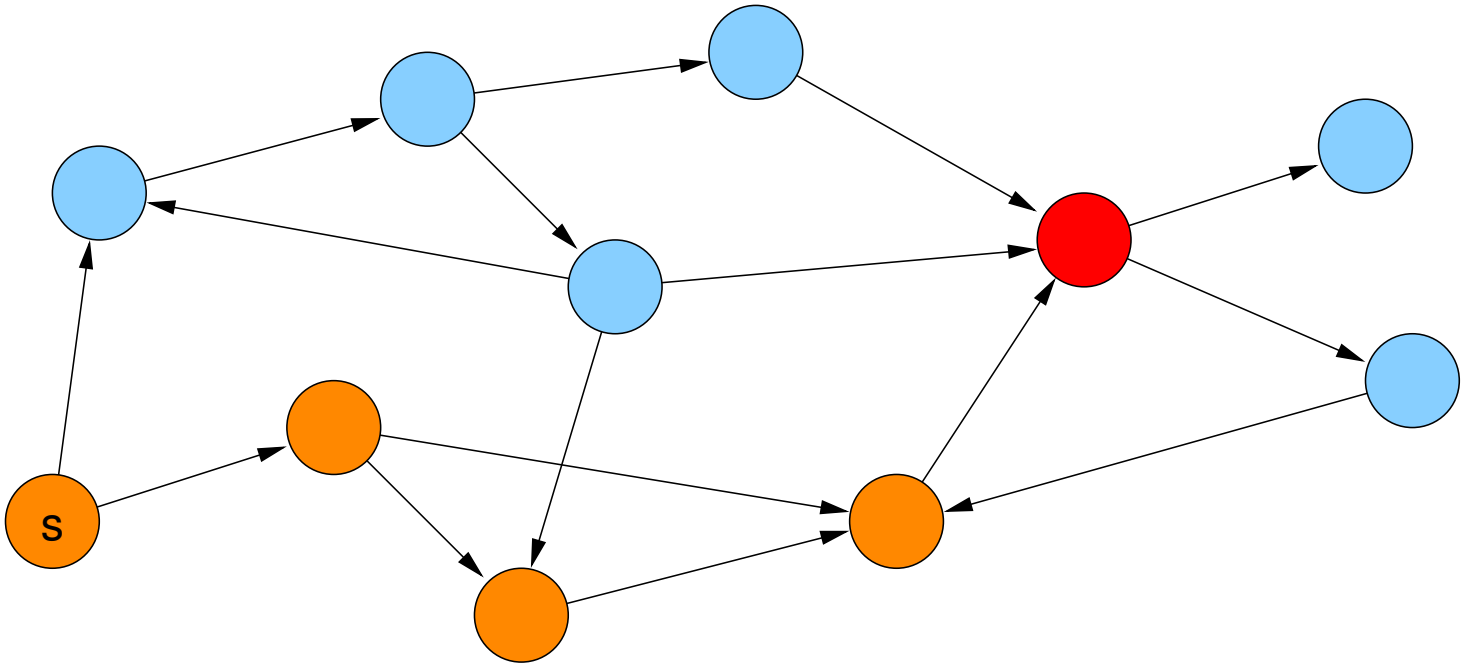
Tiefensuche



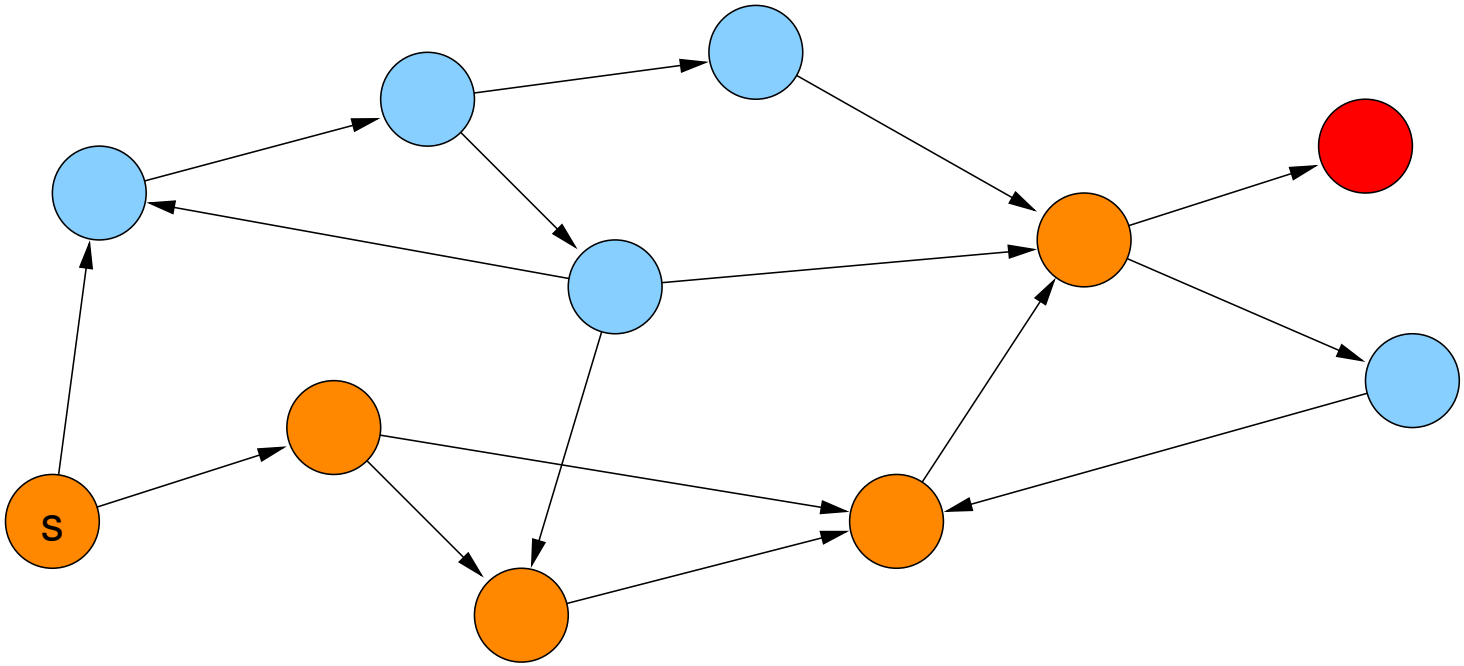
Tiefensuche



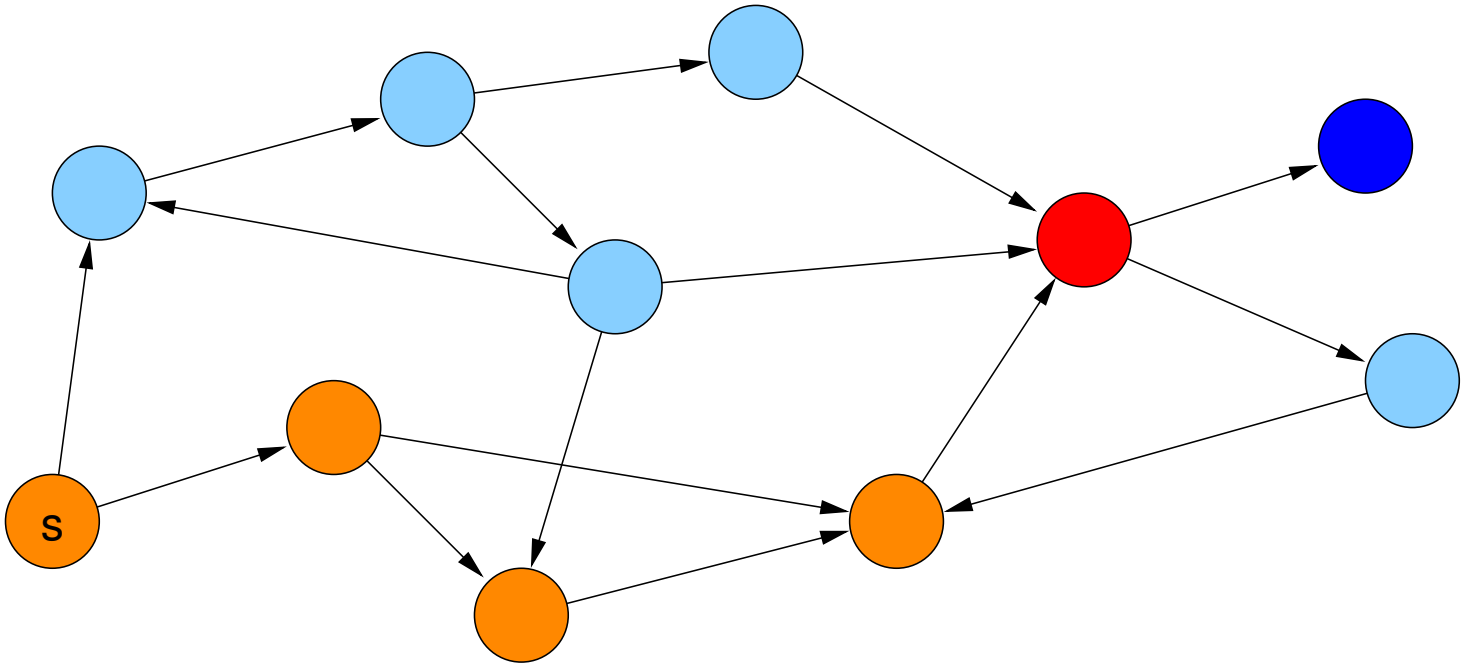
Tiefensuche



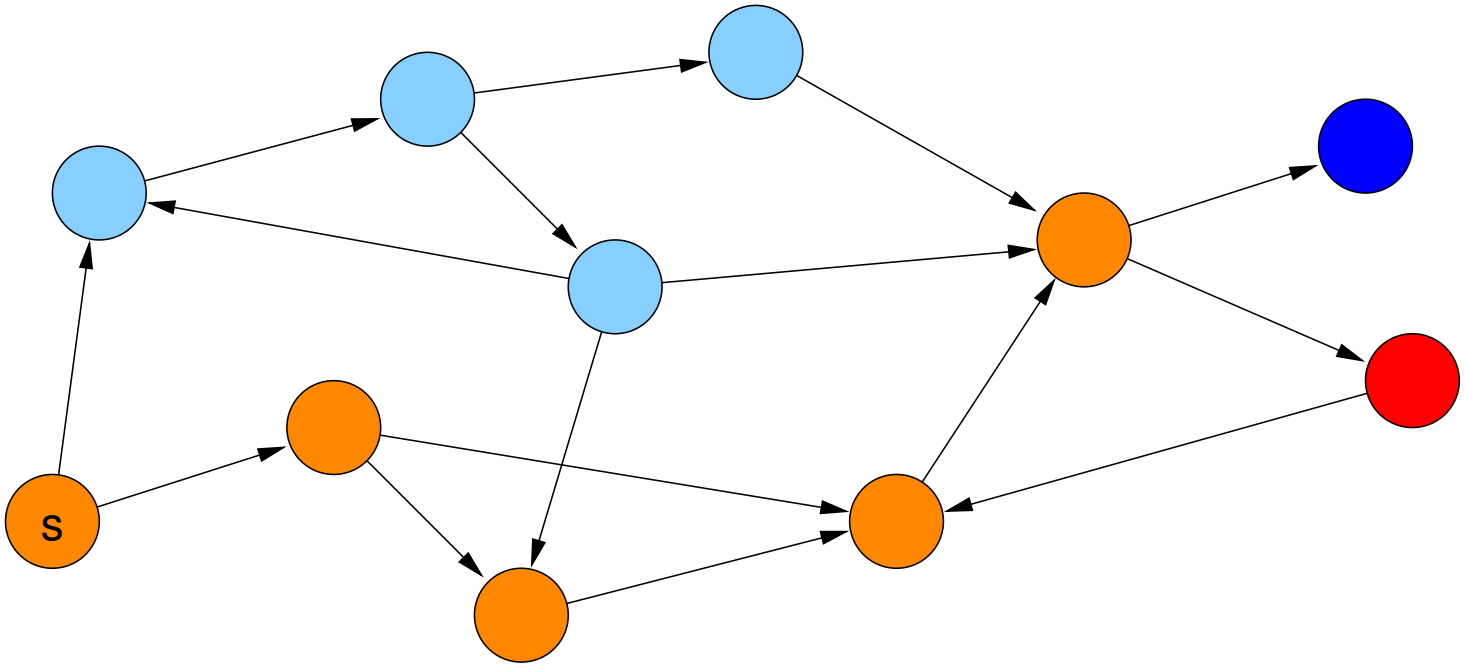
Tiefensuche



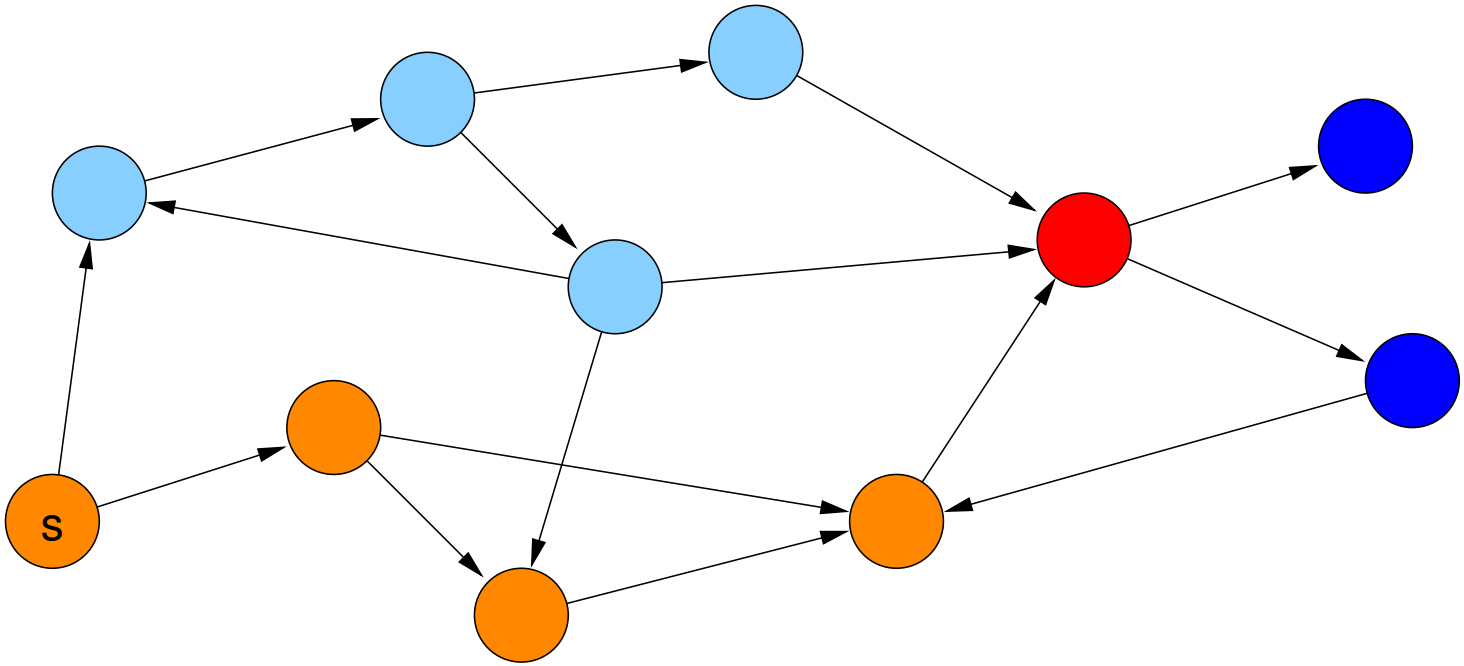
Tiefensuche



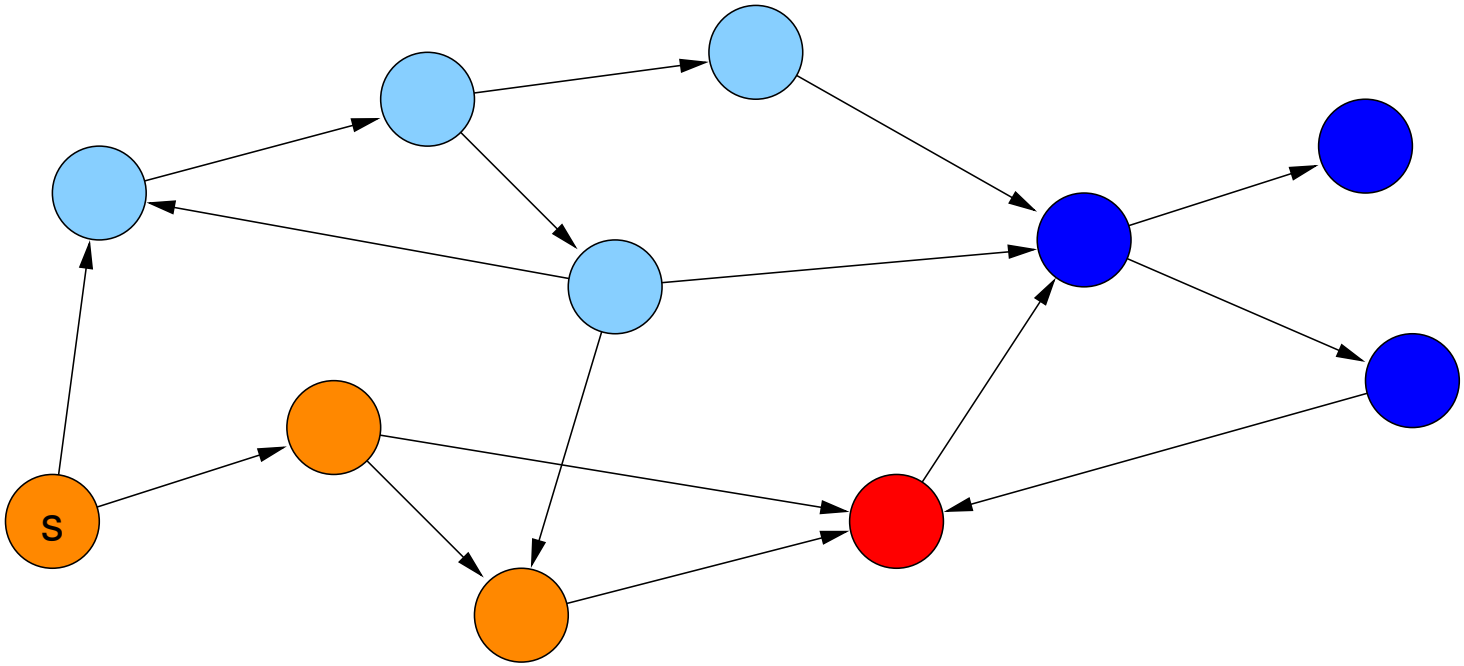
Tiefensuche



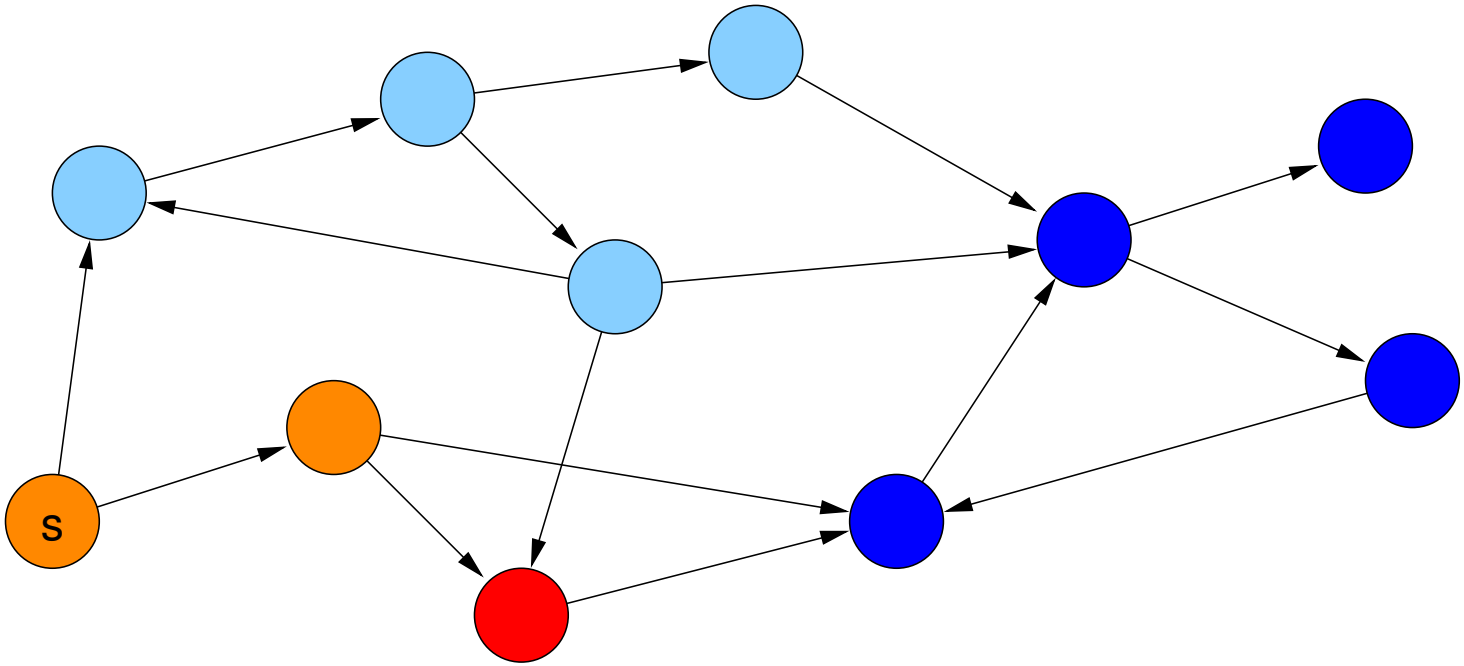
Tiefensuche



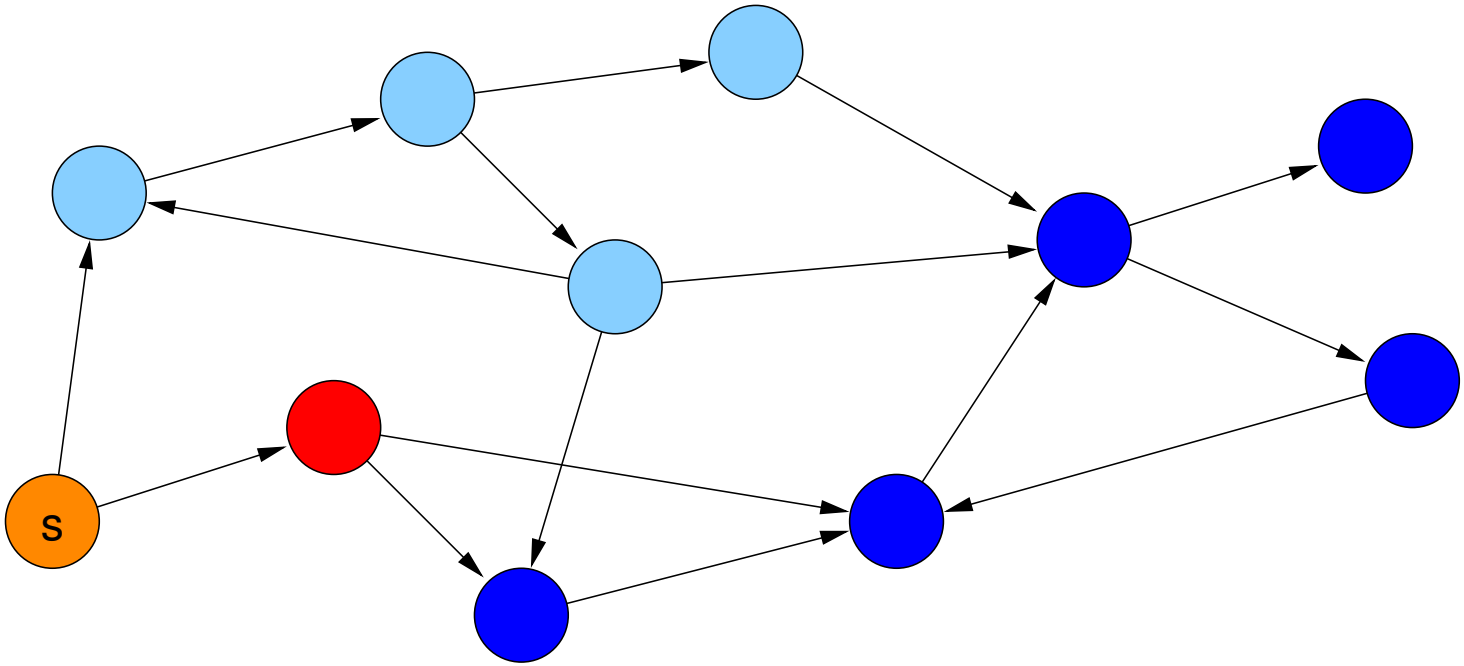
Tiefensuche



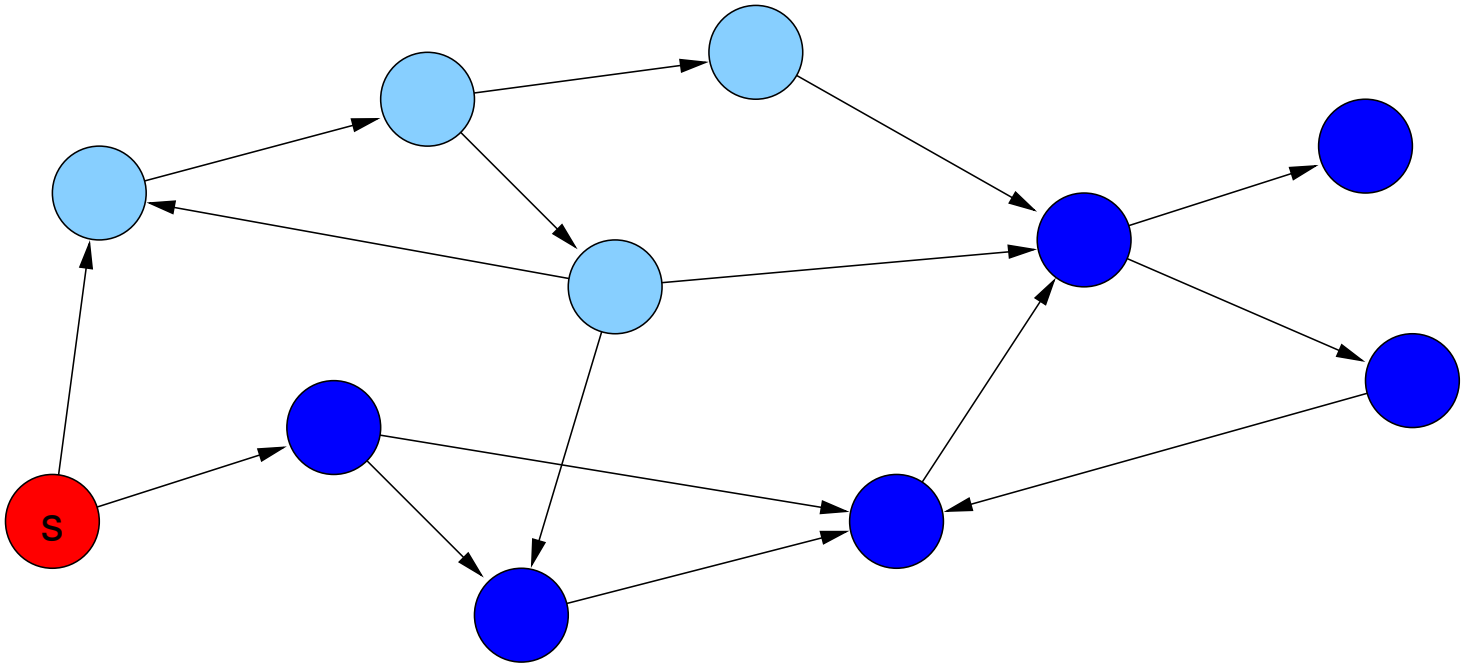
Tiefensuche



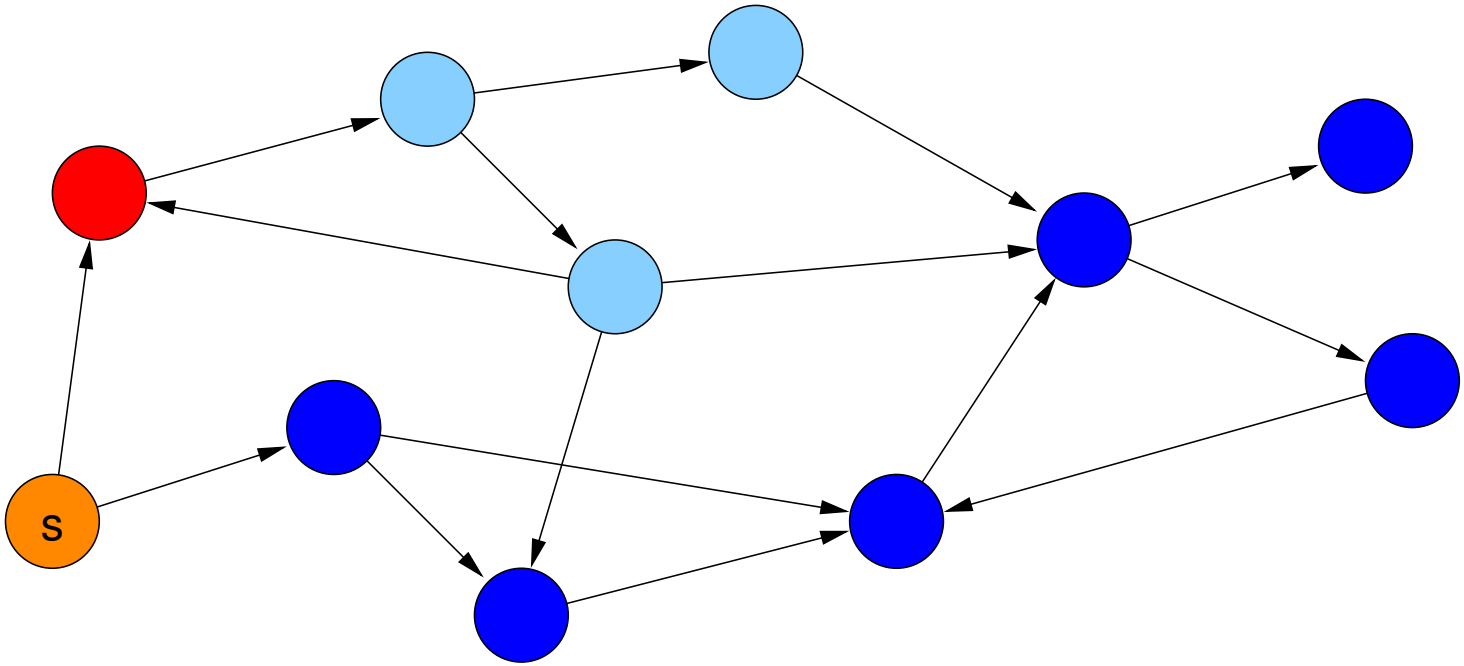
Tiefensuche



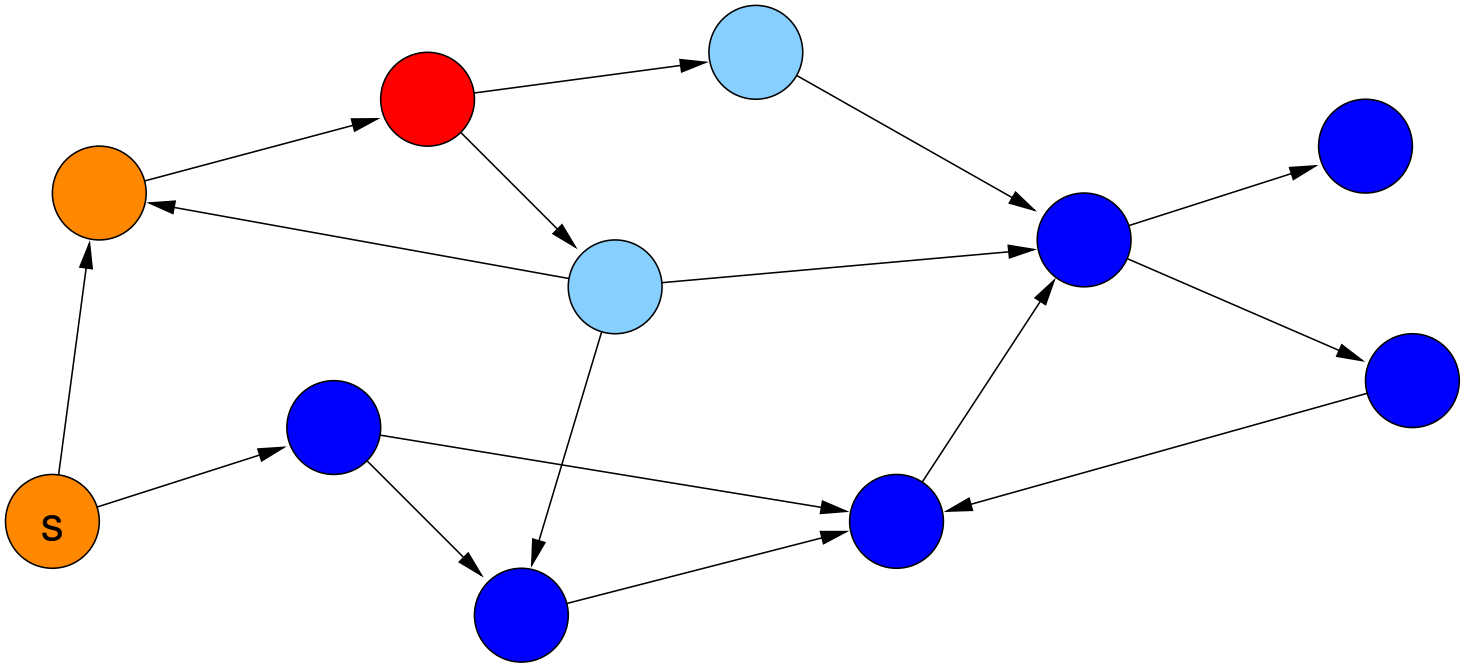
Tiefensuche



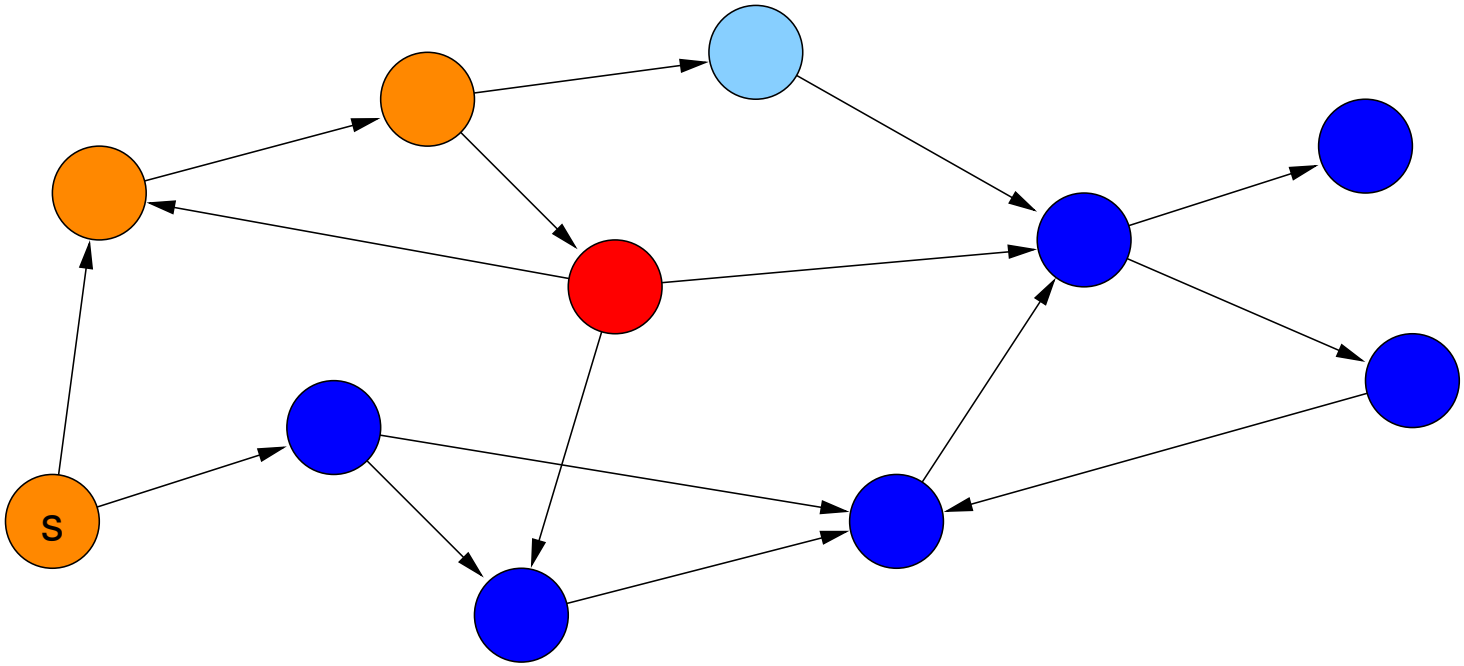
Tiefensuche



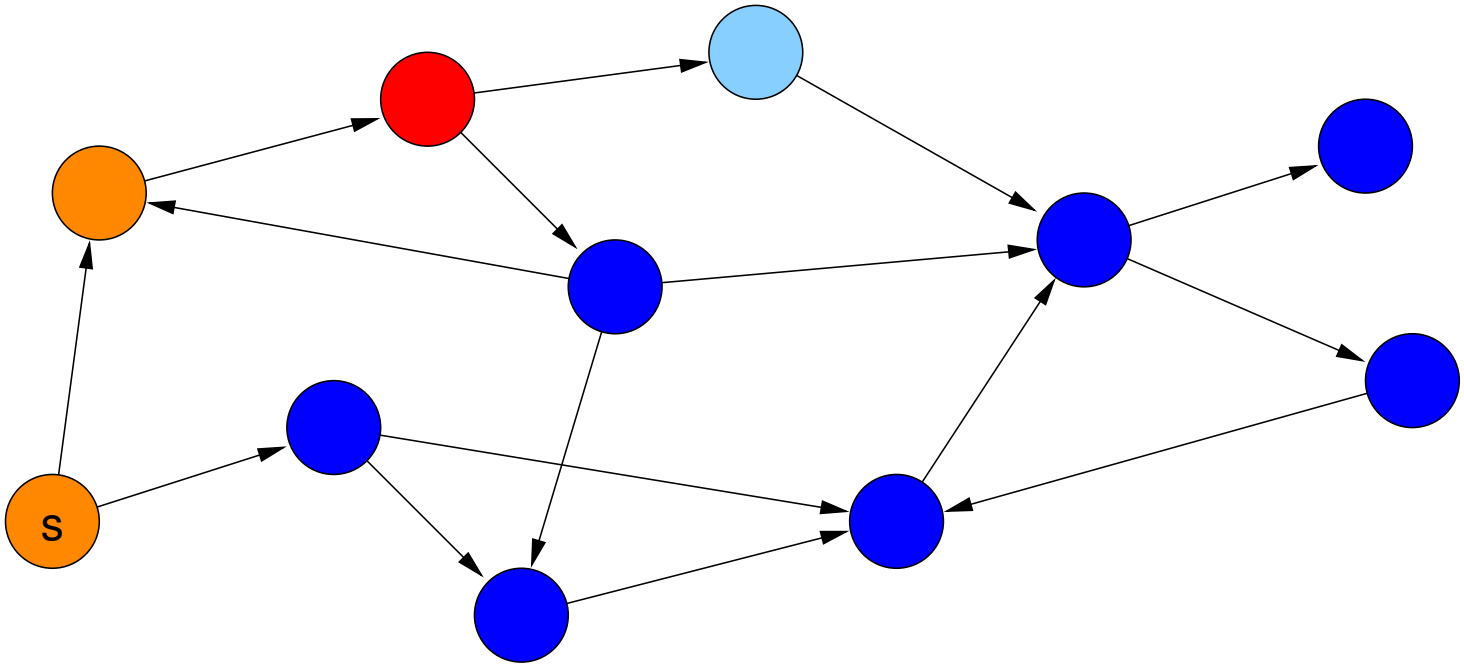
Tiefensuche



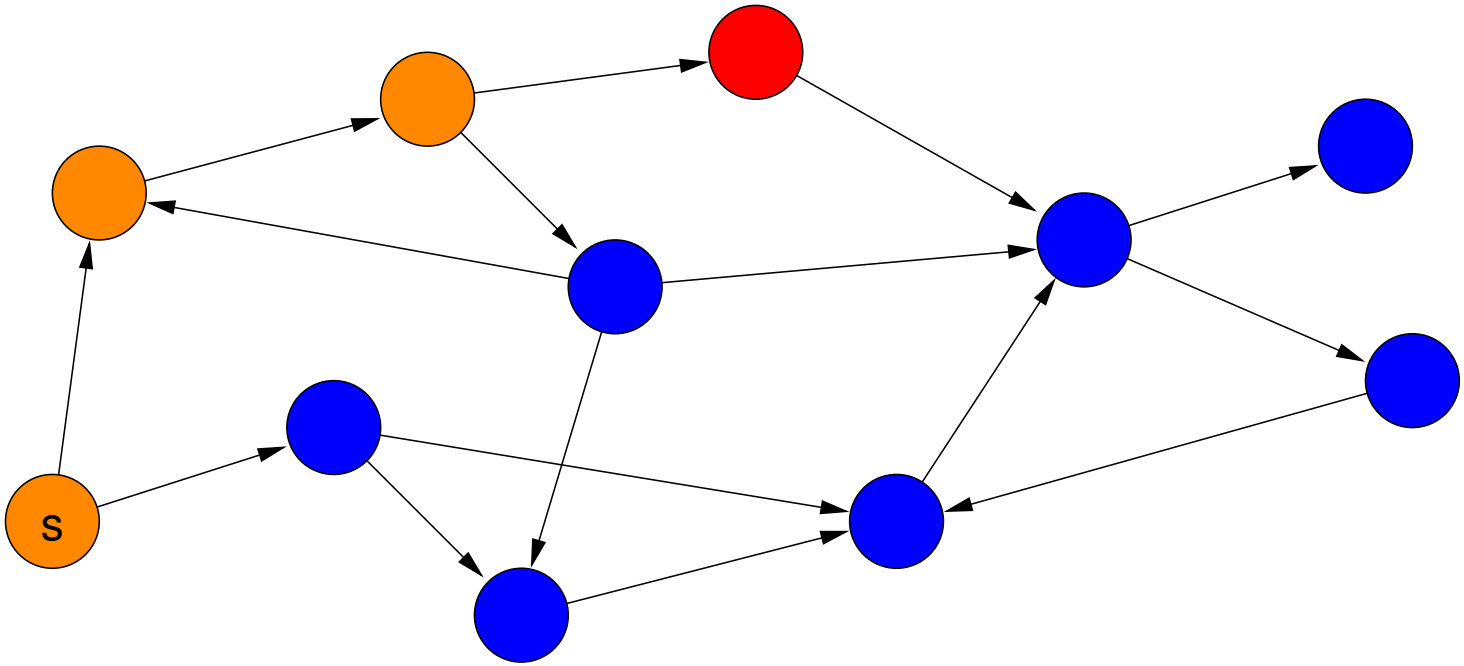
Tiefensuche



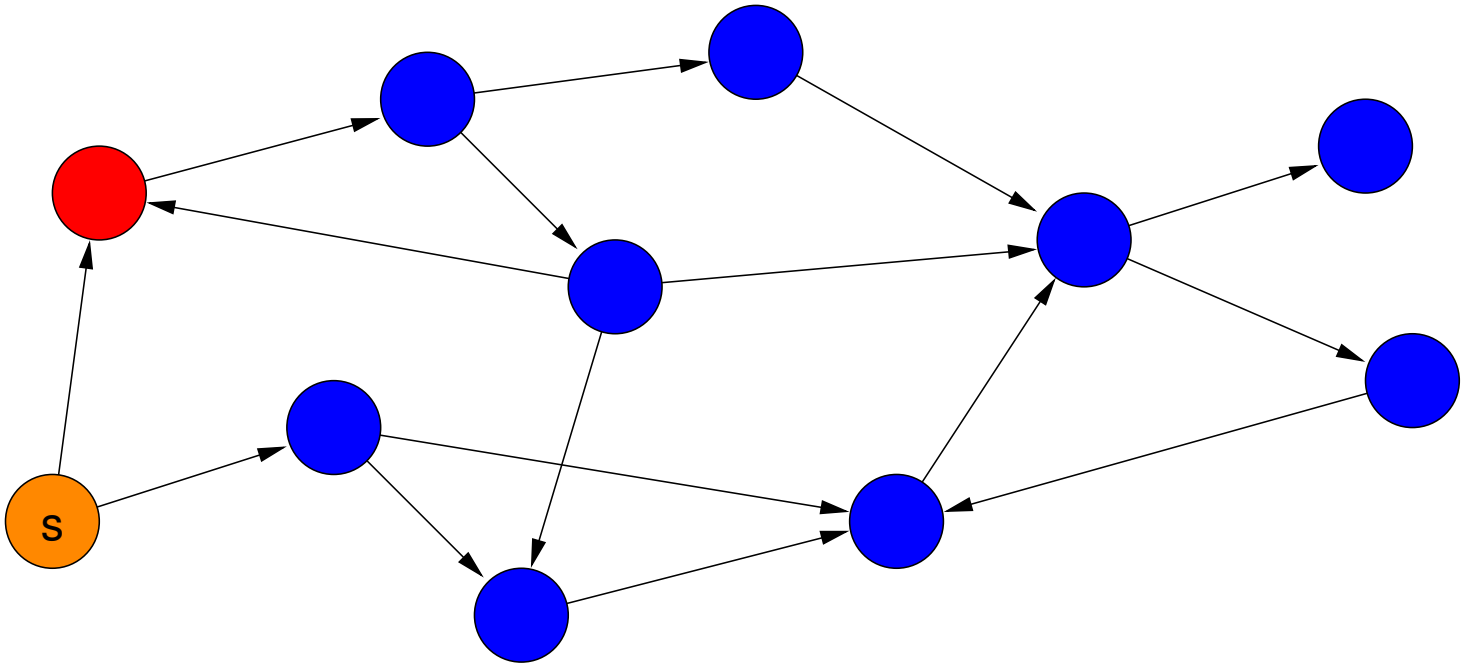
Tiefensuche



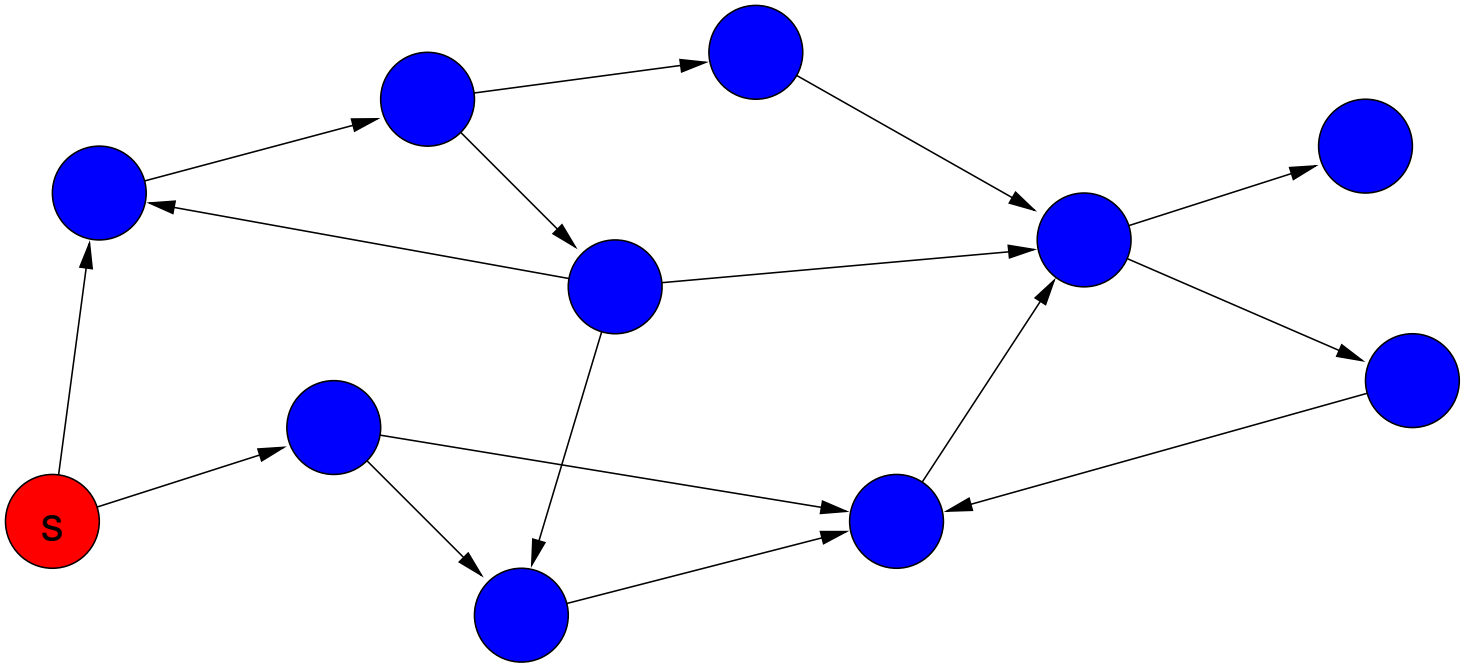
Tiefensuche



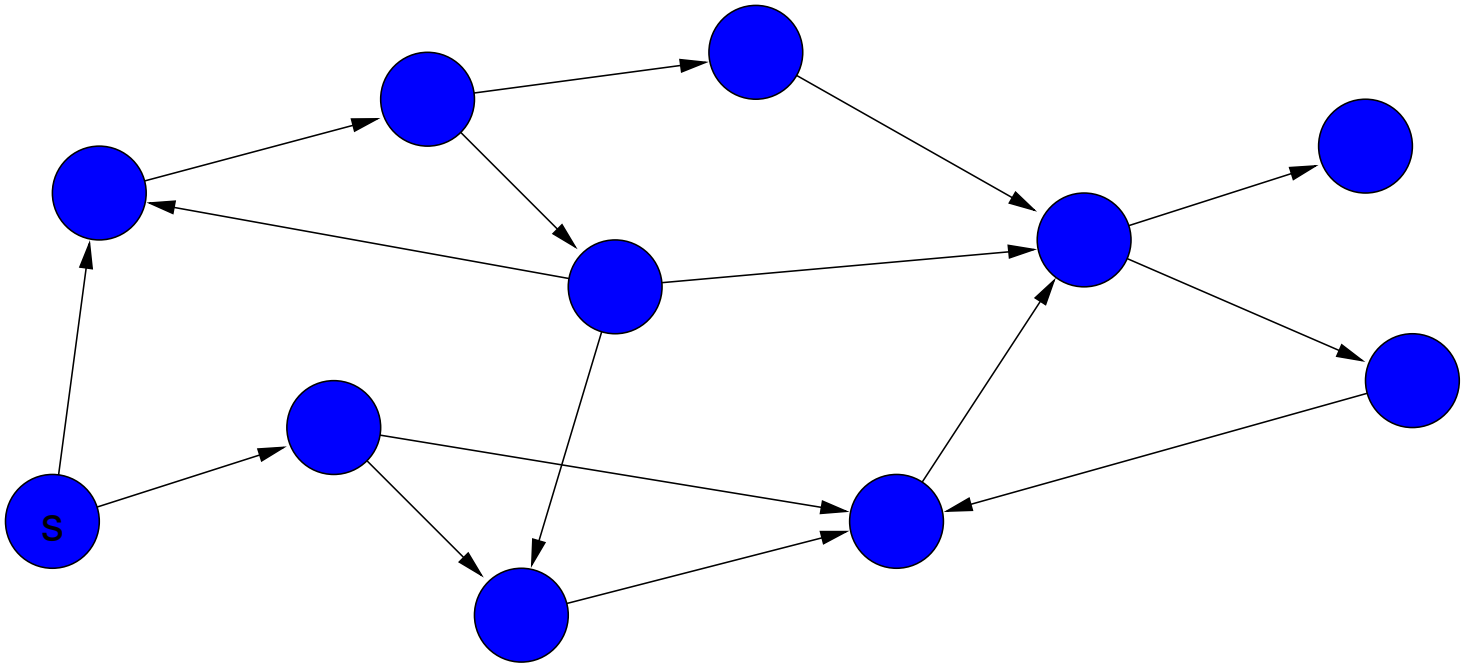
Tiefensuche



Tiefensuche

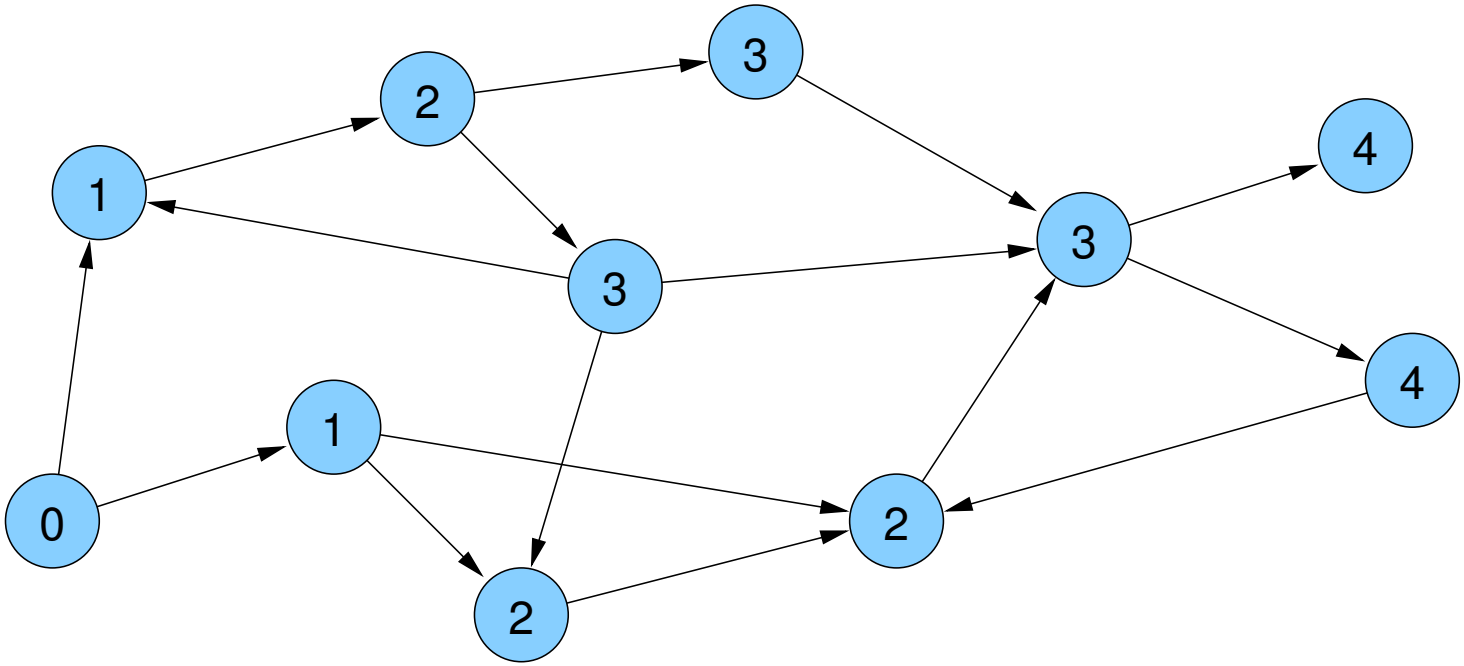


Tiefensuche



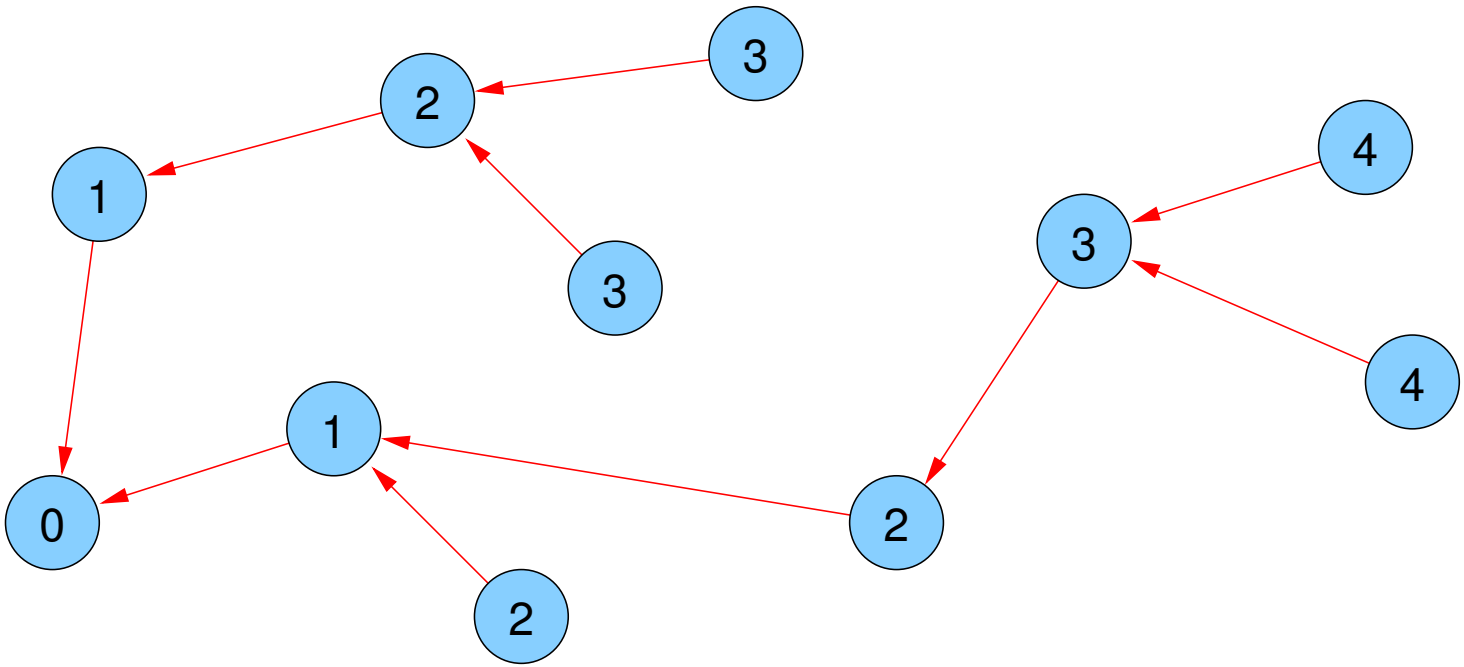
Breitensuche

- $d(v)$: Distanz von Knoten v zu s ($d(s) = 0$)



Breitensuche

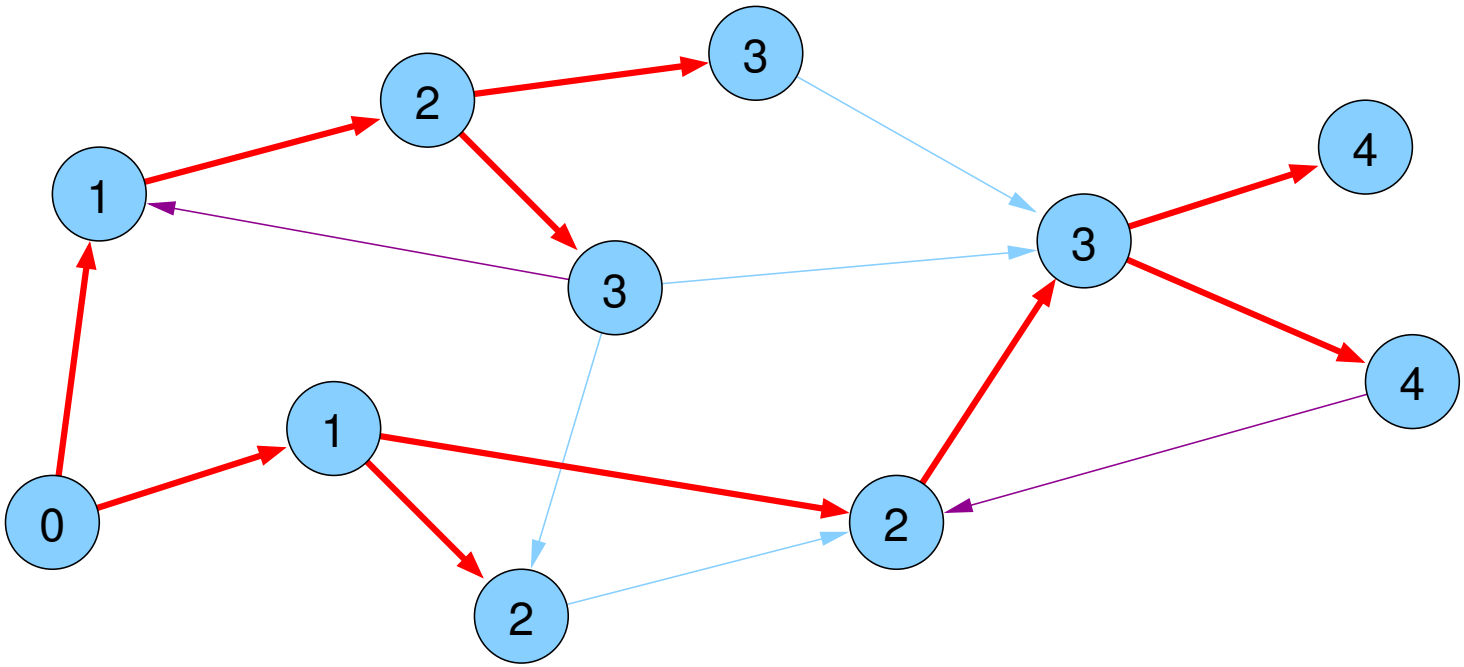
- parent(v): Knoten, von dem v entdeckt wurde
- parent wird beim ersten Besuch von §v gesetzt ($\Rightarrow$ eindeutig)



Breitensuche

Kantentypen:

- Baumkanten: zum Kind
- Rückwärtskanten: zu einem Vorfahren
- Kreuzkanten: sonstige



Breitensuche

```
void BFS(Node s) {
    d[s] = 0;
    parent[s] = s;
    List<Node> q = ⟨s⟩;
    while (!q.empty()) {
        u =q.popFront();
        foreach ((u, v) ∈ E) {
            if (parent[v] == null) {
                q.pushBack(v);
                d[v] = d[u]+1;
                parent[v] = u;
            }
        }
    }
}
```