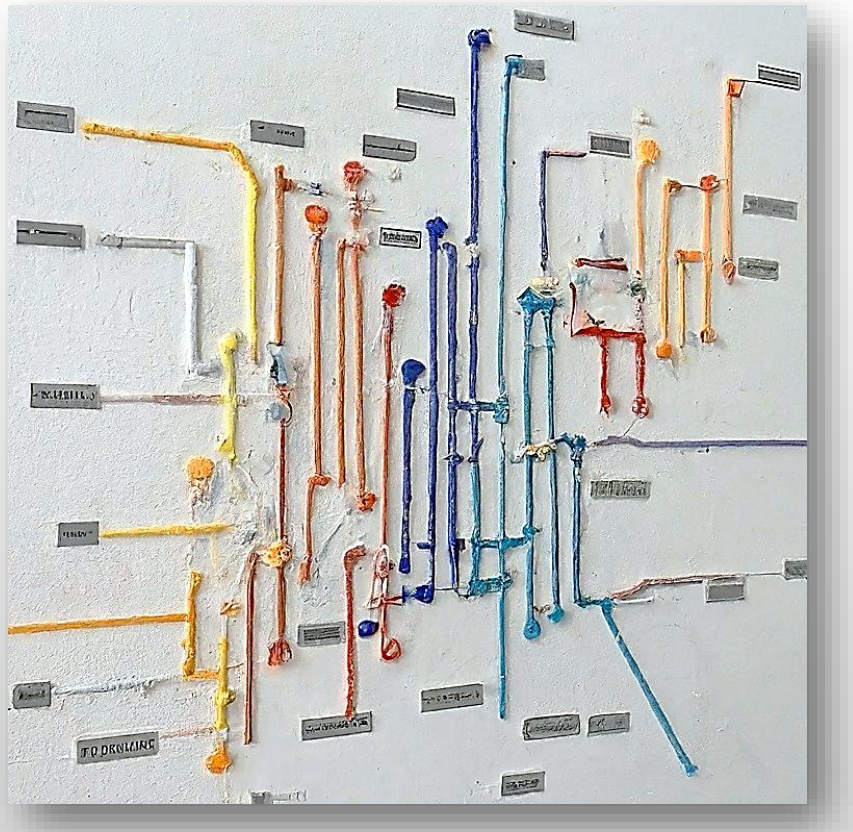
Bölüm 4: Çizge Algoritmaları

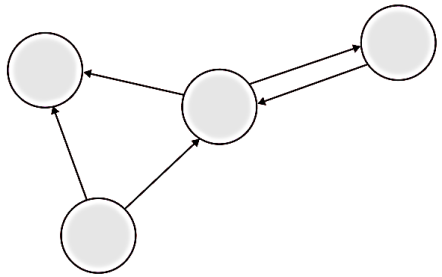
Algoritmalar

Çizge Algoritmaları

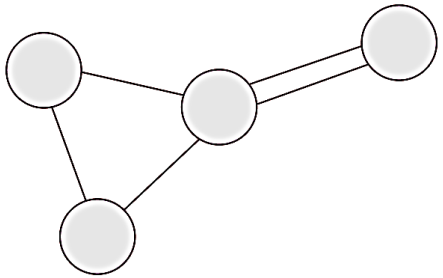
- Dünya bir ağ gibidir.
  - Şehirler yollarla,
  - İnsanlar ilişkilerle,
  - Bilgisayarlar kablolarla birbirine bağlıdır.
- Çizge algoritmaları bu ağları inceler ve anlamlandırır.



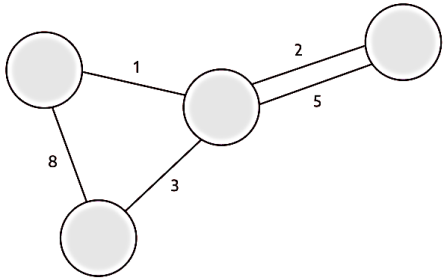
Çizge Türleri



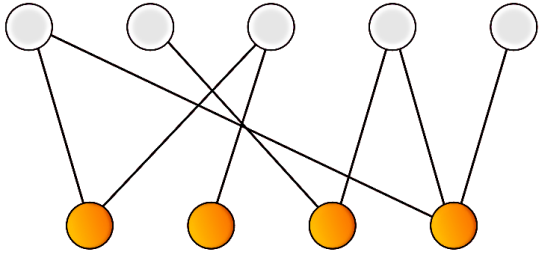
Directed graph



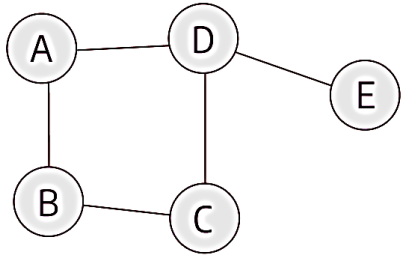
Undirected



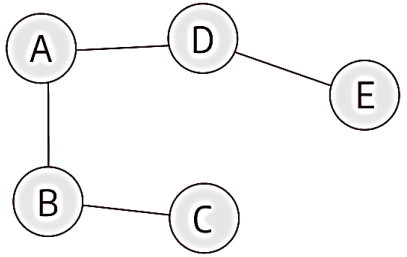
Weighted



Bipartite graph



Cyclic graph



Acyclic graph

Çizge Türleri

- **Basit Çizge**: Döngü veya çoklu kenar içermez.
- **Yönsüz Çizge**: Kenarları iki yönde de geçilebilir.
- **Yönlü Çizge**: Kenarları belirli bir yönde geçilebilir.
- **Ağırlıklı Çizge**: Her bir kenarına bir ağırlık değeri atanmıştır.
- **Tam Çizge**: Her bir düğüm çifti, bir kenarla birbirine bağlıdır.
- **İki Parçalı Çizge**: Kenarlar farklı kümedeki iki düğümü birbirine bağlar.
- **Ağaç**: Döngü içermez, iki köşesi arasında yalnızca bir yol bulunur.

Çizge Algoritmaları

- Birbirine bağlı noktalar (düğüm) ve bu noktaları birleştiren çizgiler (kenar) ile temsil edilen ağ yapılarını inceler.
- Ağlarda en kısa yolu hesaplama, gruplama gibi işlemleri gerçekleştirir.
- Sosyal ağlar, harita uygulamaları, navigasyon gibi birçok alanda kullanılır.

Çizge Algoritmalarının Çeşitleri

- Farklı çizge algoritmaları, farklı işlemler için kullanılır.
- Derinlik Öncelikli Arama (DFS):
  - Bir düğümden başlar, dallanarak tüm ağı gezer.
- Genişlik Öncelikli Arama (BFS):
  - Bir düğümden başlar, katman katman tüm ağı gezer.
- Dijkstra Algoritması:
  - Başlangıç düğümünden diğer düğümlere en kısa yolları bulur.
- Kruskal Algoritması:
  - Bir ağı minimum maliyetle birbirine bağlayan kenarları seçer.

Çizge Algoritmaları

- DFS bir labirentten çıkış yolu ararken kullanılabilir.
- BFS bir haberin tüm şehire yayılma sürecini modelleyebilir.
- Dijkstra en kısa sürede teslimat yapmak için kullanılabilir.

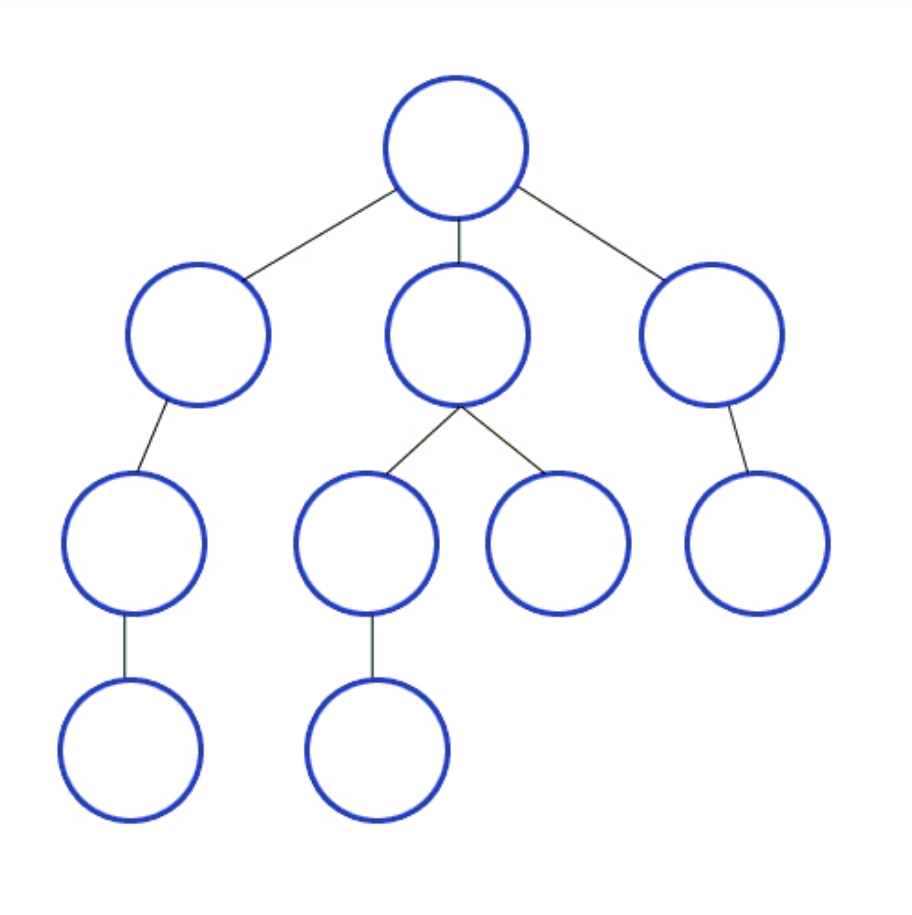
Çizge Algoritmaları

- Gezinme algoritmaları (*Graph traversal*)
- En kısa yol algoritmaları (*Shortest path*)
- Minimum yayılan ağaç algoritmaları (*Minimum spanning tree*)
- Ağ akış algoritmaları (*Network flow*)

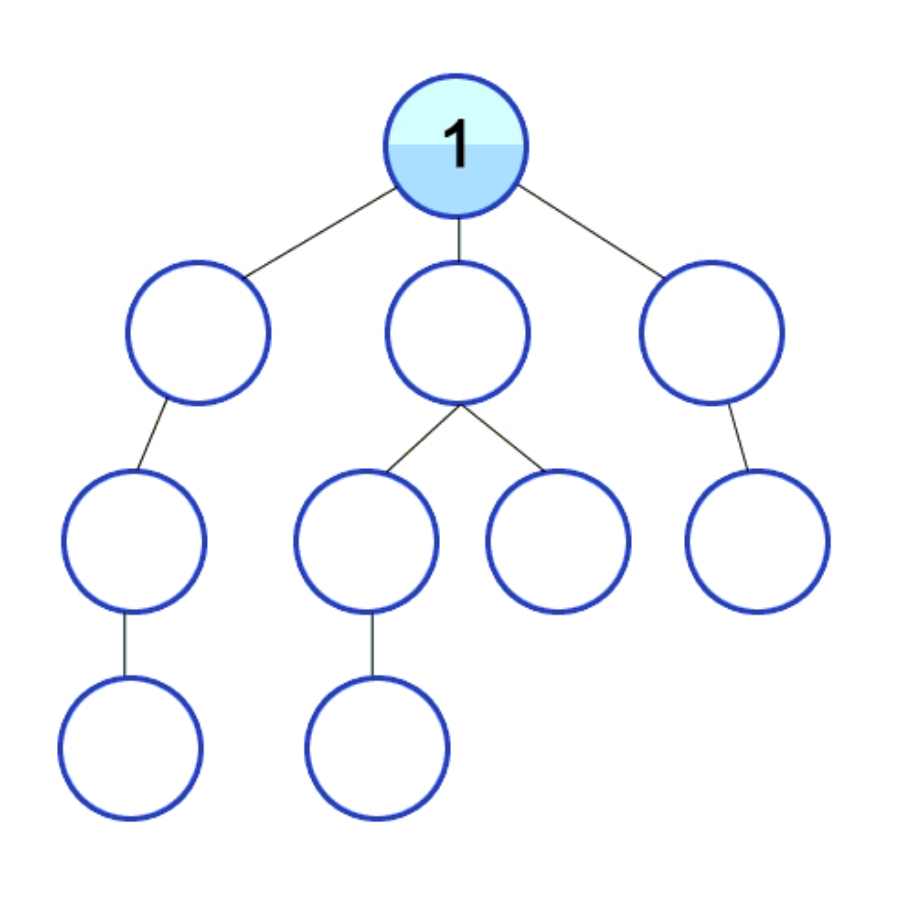
Çizge Gezinme Algoritmaları (Graph Traversal)

- Çizgenin yapısını sistematik bir şekilde keşfetmek için kullanılır.
- İki ana kategoriye ayrılır:
  - derinlik öncelikli arama (DFS) ve
  - genişlik öncelikli arama (BFS).
- *DFS*, yığıt veri yapısı kullanarak, bir düğümden başlar ve mümkün olduğunca derinlere iner, tüm komşularını ziyaret ettikten sonra geri döner.
- *BFS*, kuyruk veri yapısı kullanarak, seviye seviye tüm düğümleri ziyaret eder.

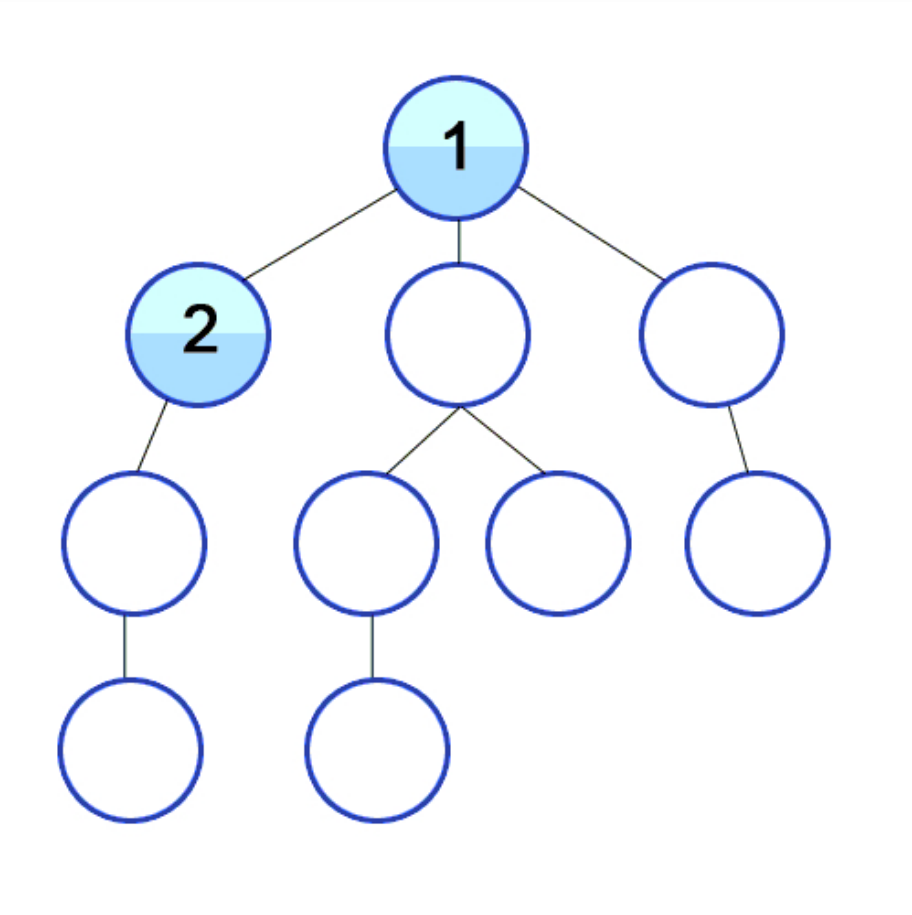
Derinlik Öncelikli Arama



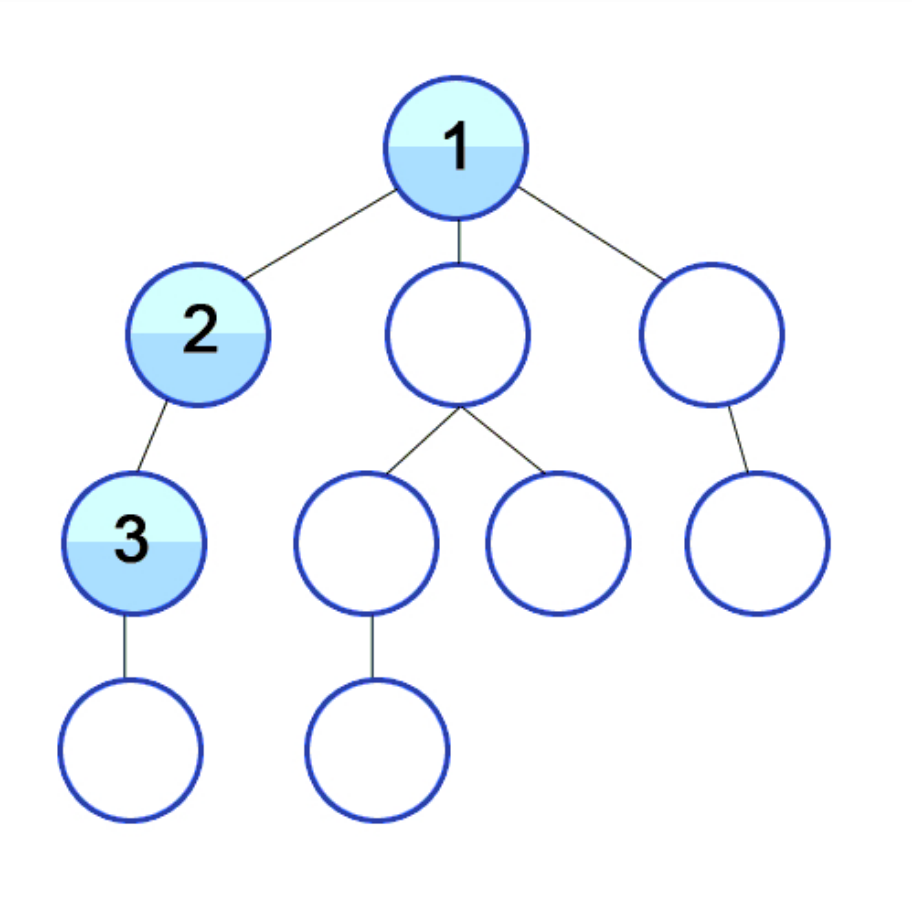
Derinlik Öncelikli Arama



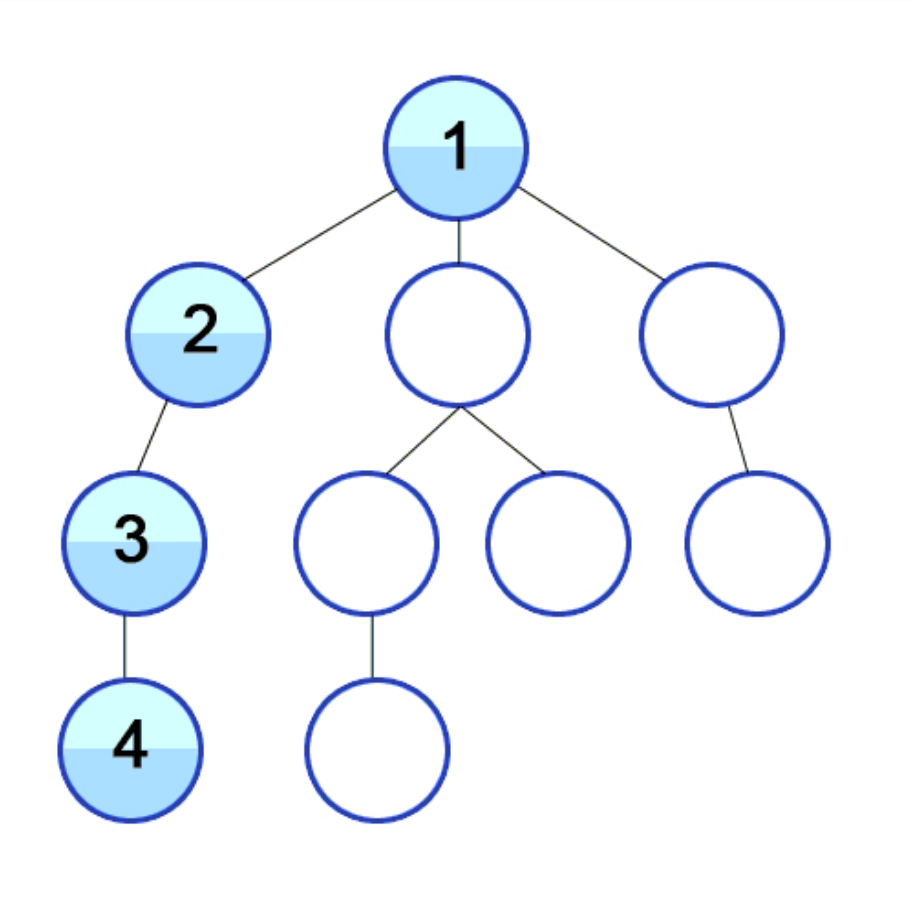
Derinlik Öncelikli Arama



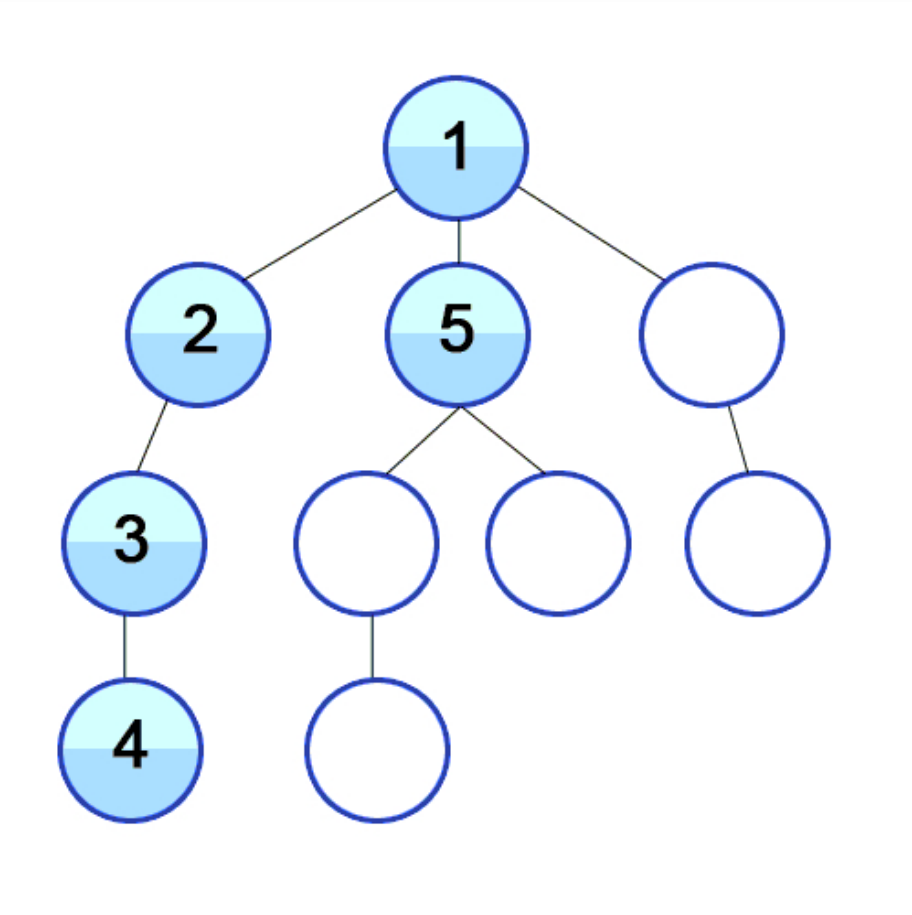
Derinlik Öncelikli Arama



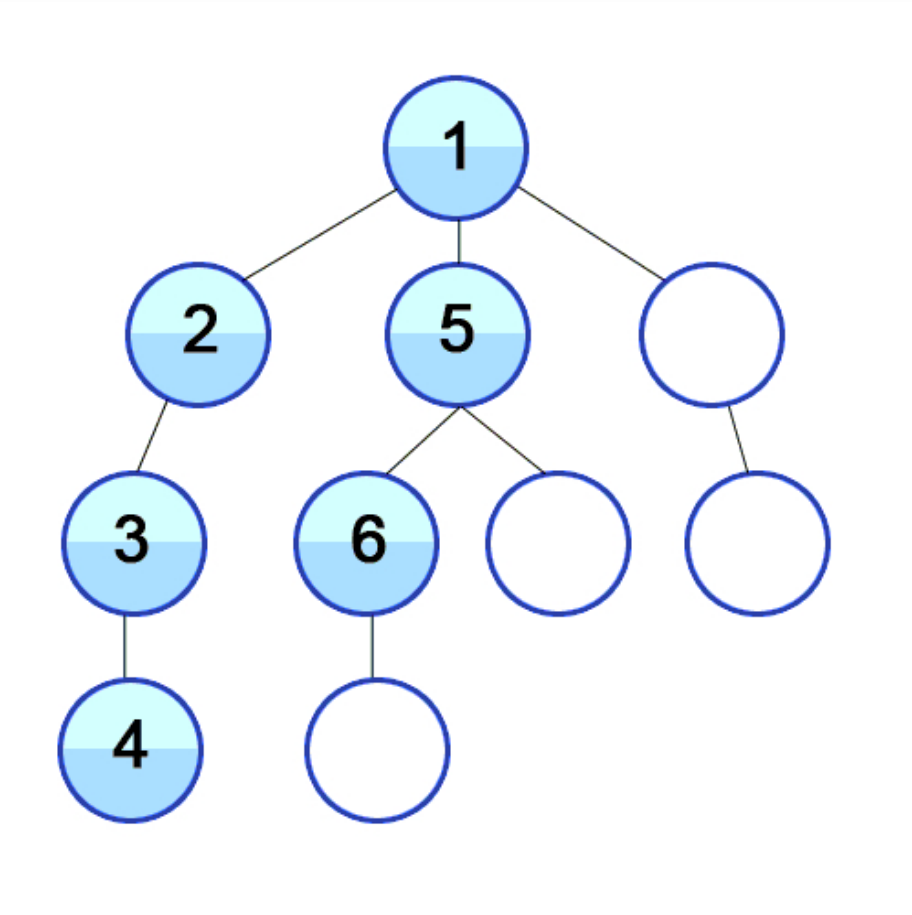
Derinlik Öncelikli Arama



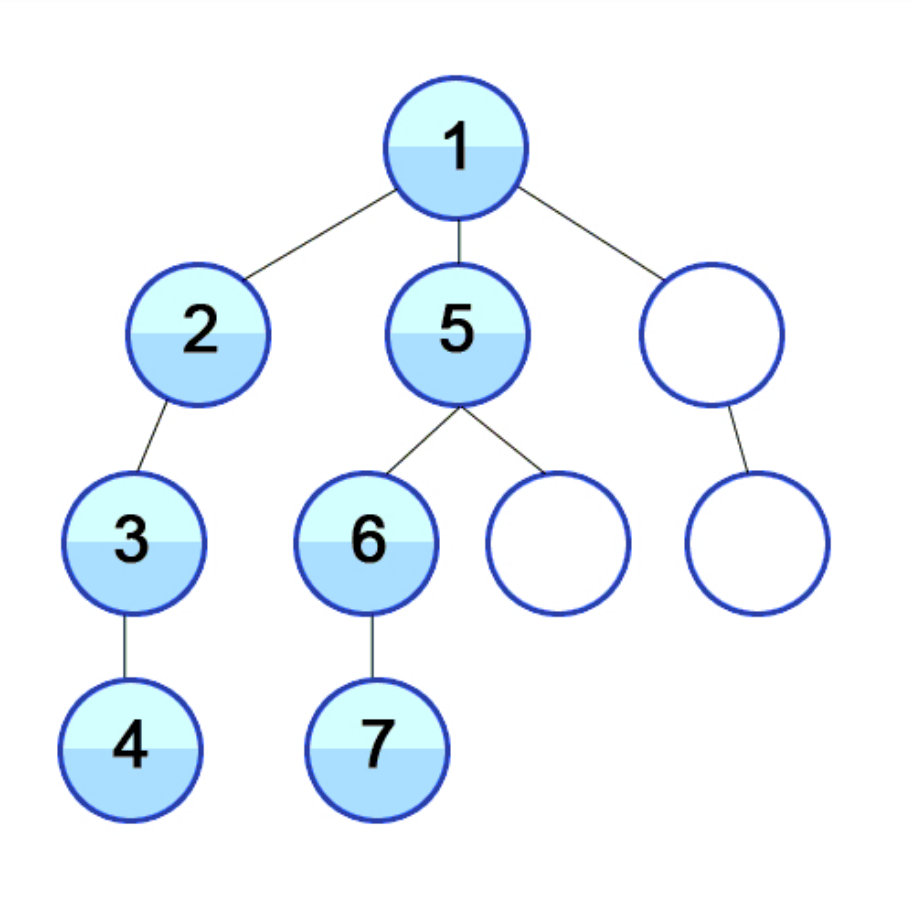
Derinlik Öncelikli Arama



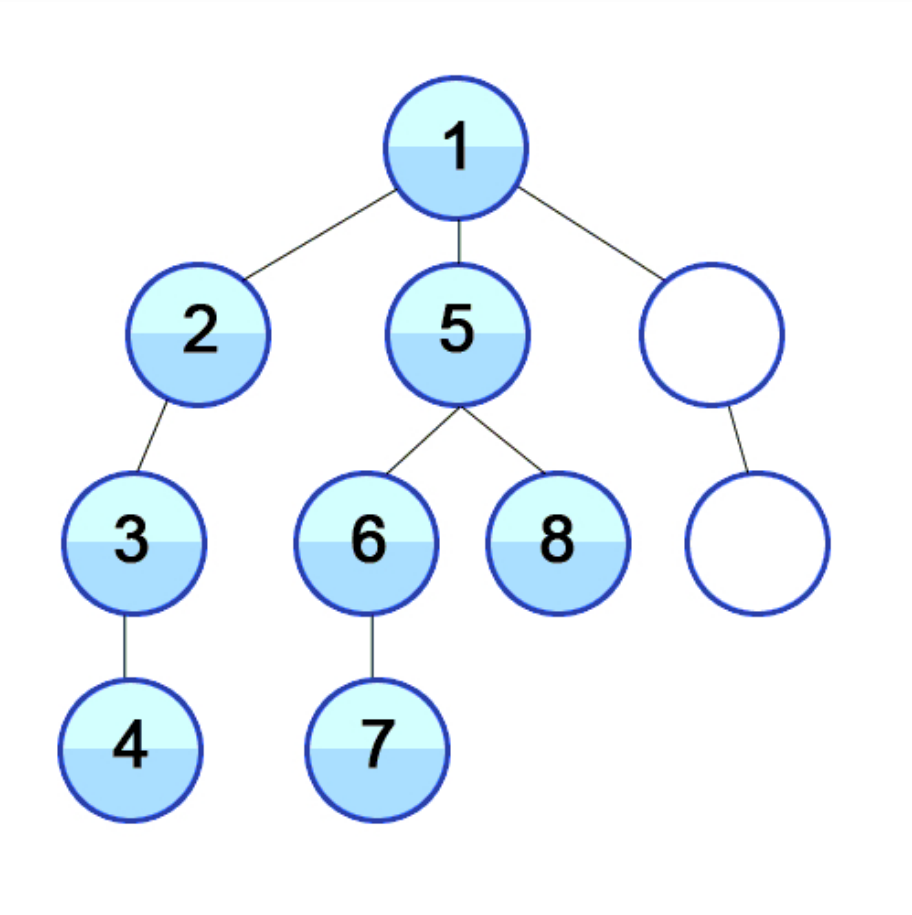
Derinlik Öncelikli Arama



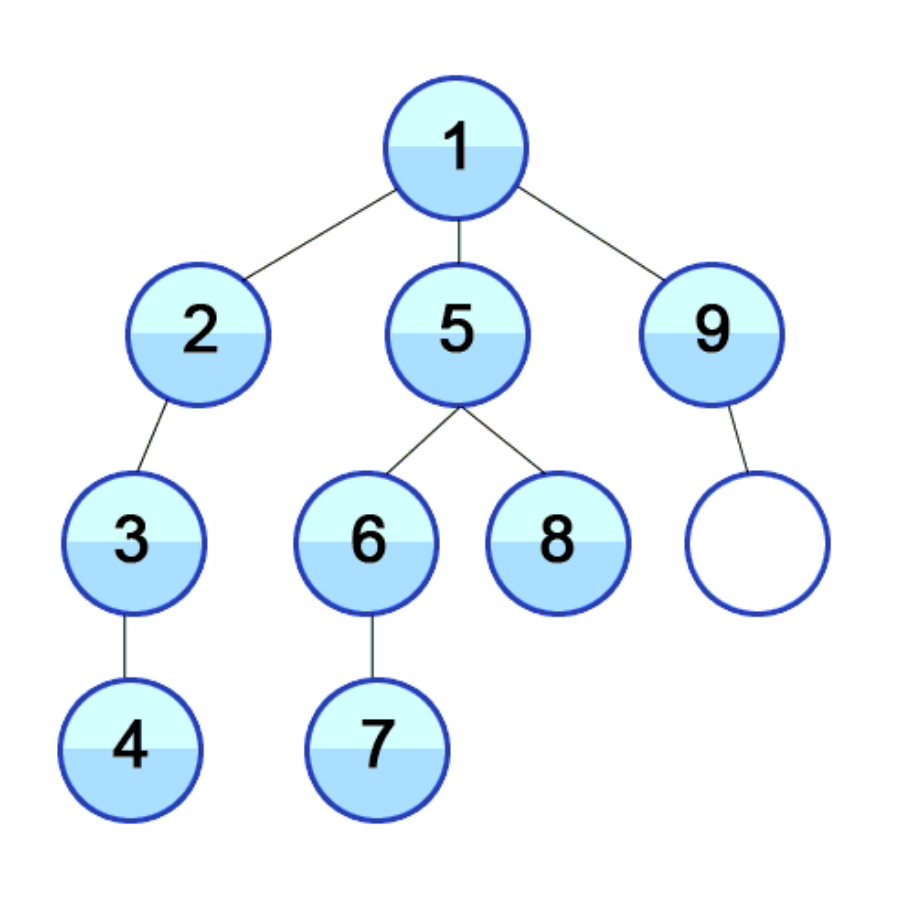
Derinlik Öncelikli Arama



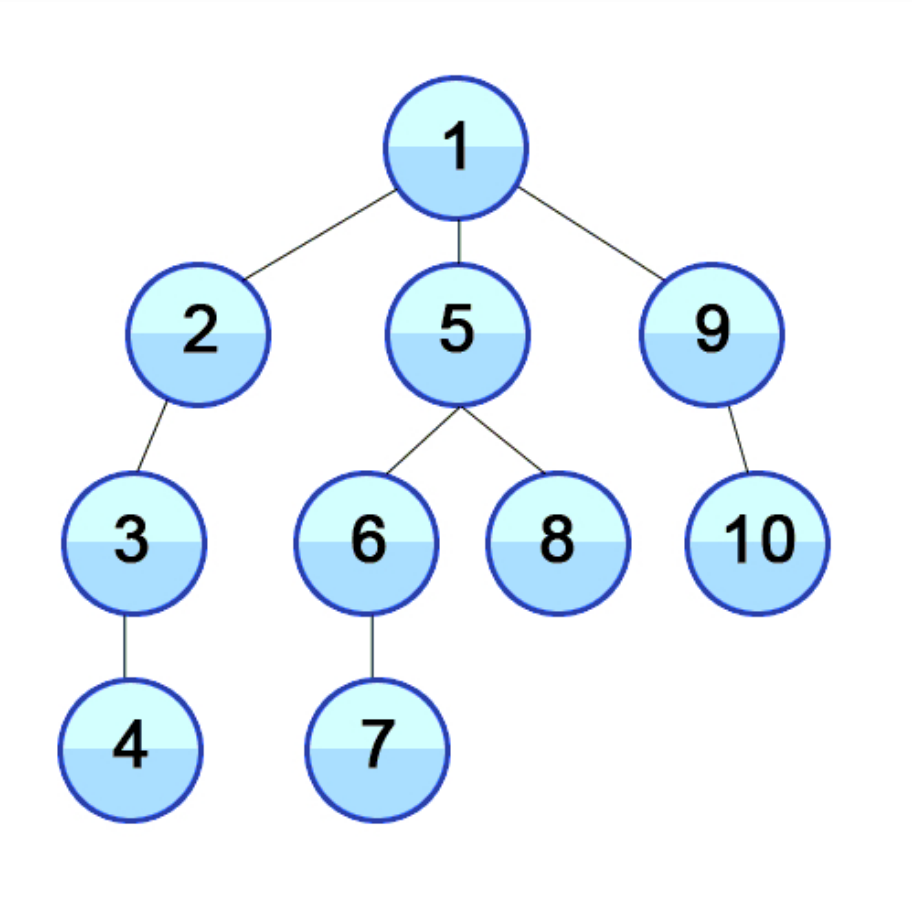
Derinlik Öncelikli Arama



Derinlik Öncelikli Arama



Derinlik Öncelikli Arama



Recursive DFS

```
procedure DFS (G, v) is
        label v as discovered
        for all directed edges from v to w that are in G.adjacentEdges(v) do
                if vertex w is not labeled as discovered then
                        call DFS (G, w)
```

Recursive DFS

DFS(düğüm, *ziyaretEdilenler*):

eğer düğüm *ziyaretEdilenler* içinde değilse:

düğüm'ü *ziyaretEdilenler*'e ekle

düğüm'ü işle (örneğin, yazdır)

her bir komşu düğüm için:

DFS(komşu, *ziyaretEdilenler*)

Iterative DFS

```
procedure DFS_iterative(G, v) is
    let S be a stack
    S.push(v)
    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

Iterative DFS

DFS(*başlangıç_düğüm*):

boş bir yığın oluştur

başlangıç_düğüm'ü yığına ekle

döngü yığın boş değilken:

mevcut_düğüm = yığından çıkar

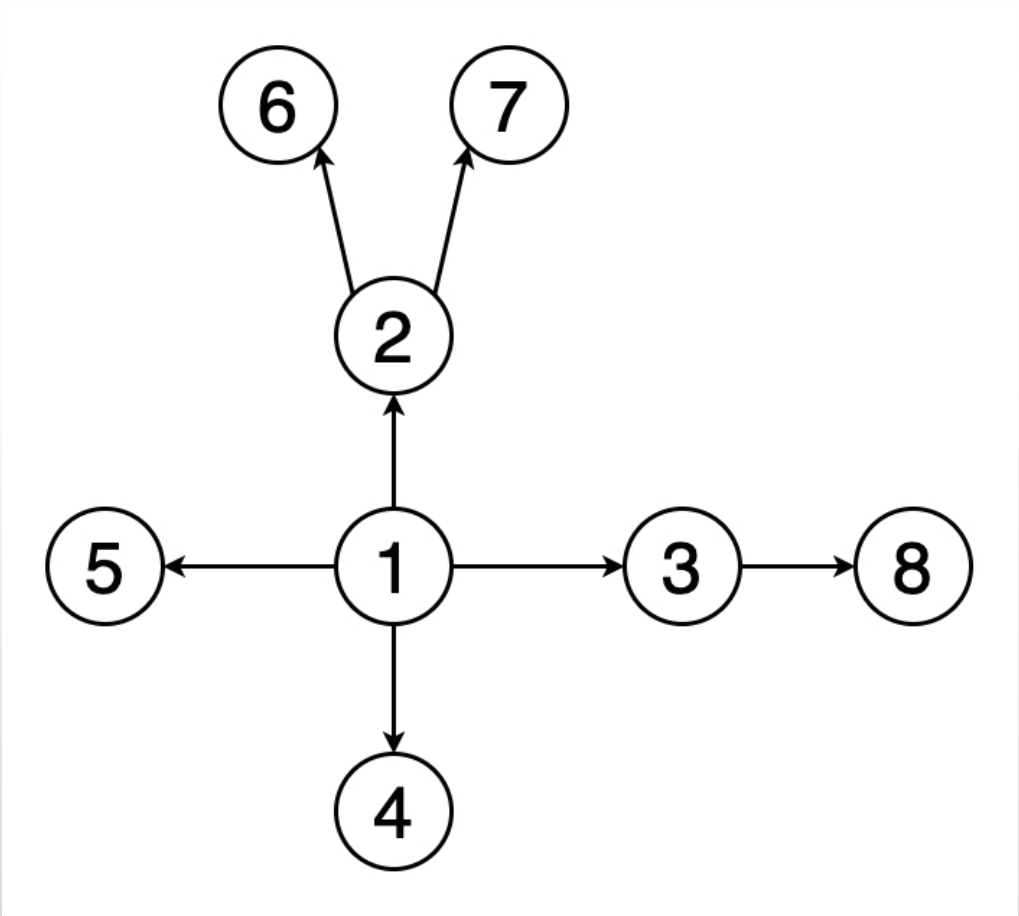
eğer *mevcut_düğüm ziyaretEdilenler* içinde değilse:

mevcut_düğüm'ü *ziyaretEdilenler*'e ekle

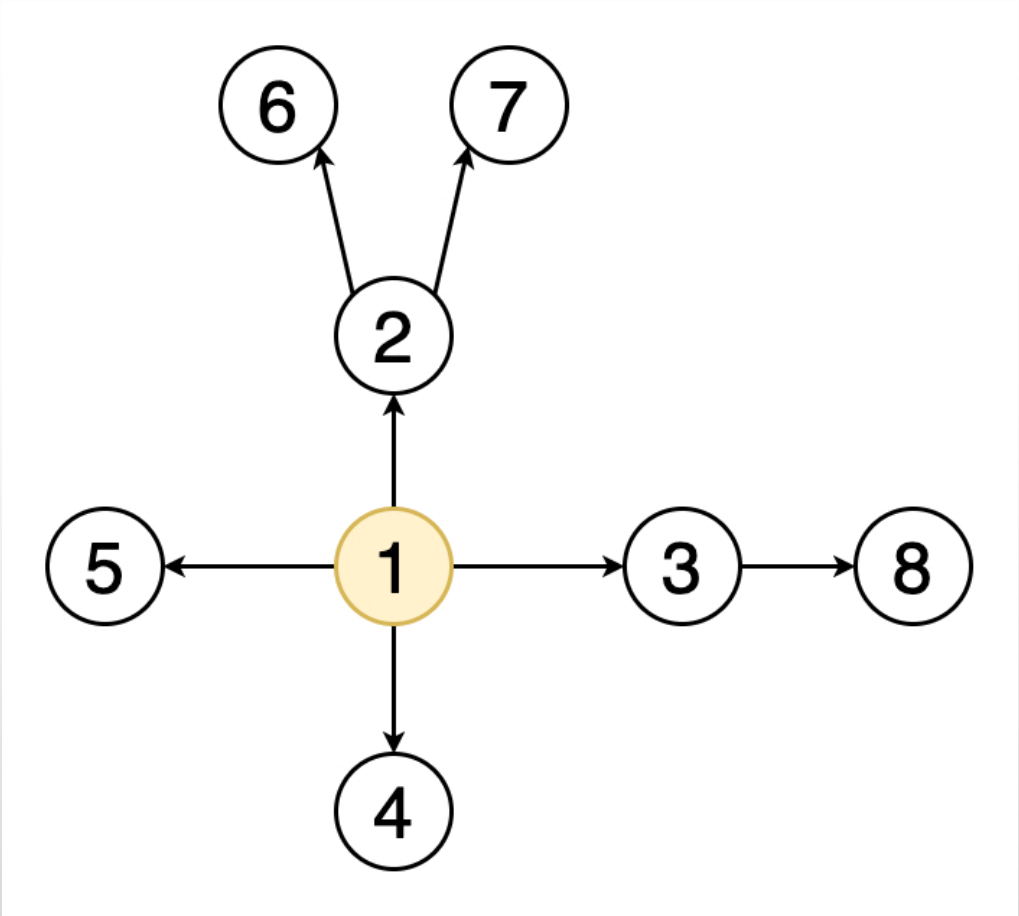
her bir *komşu_düğüm* için:

komşu_düğüm'ü yığın'a ekle

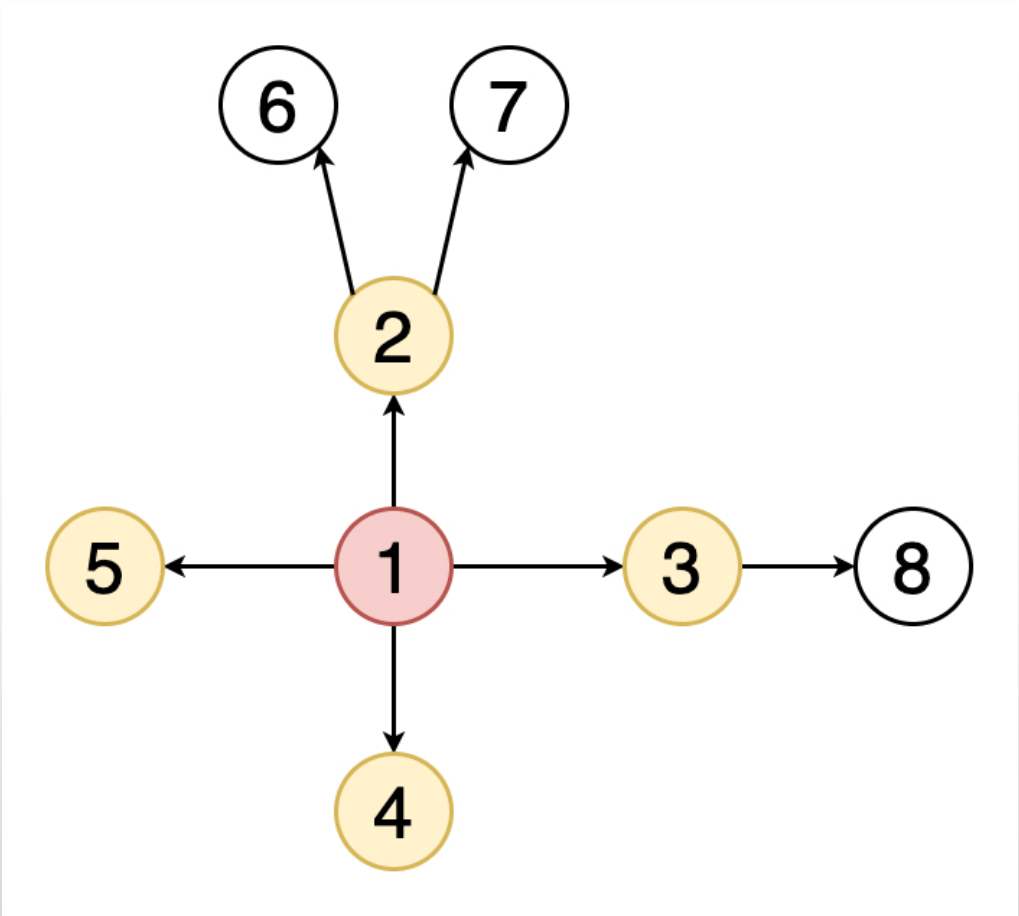
Genişlik Öncelikli Arama



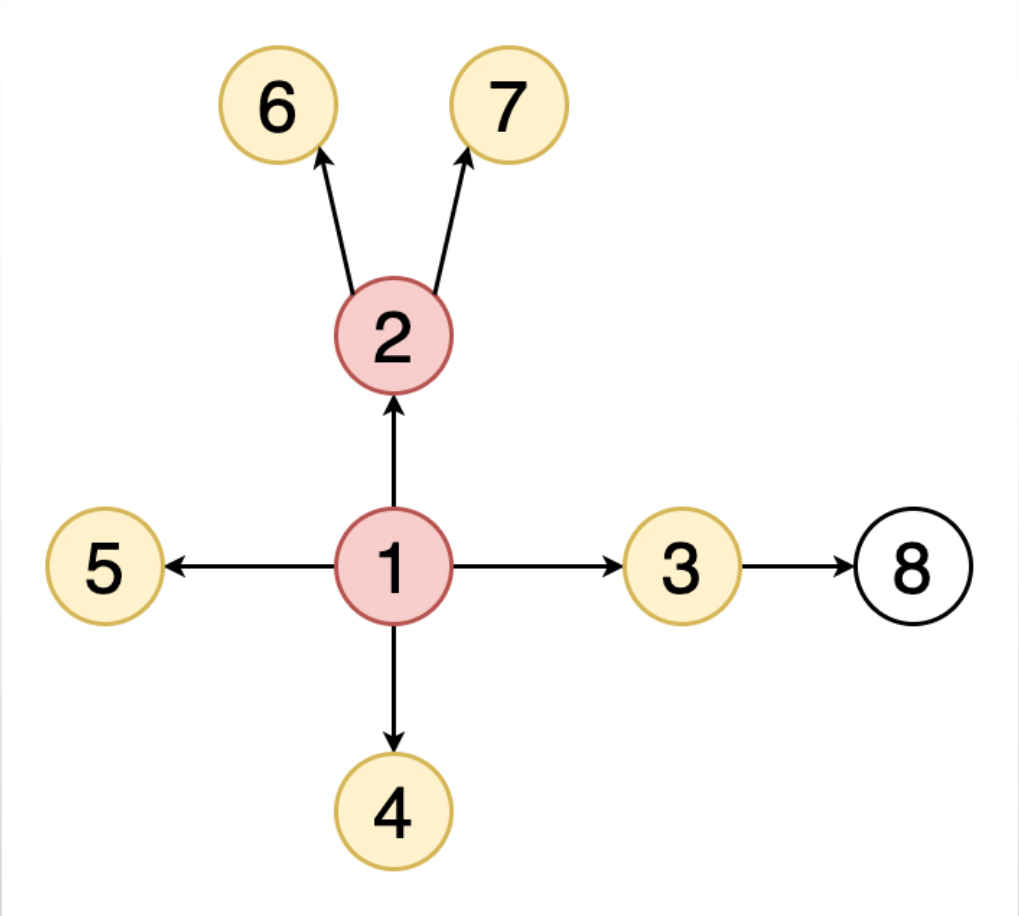
Genişlik Öncelikli Arama



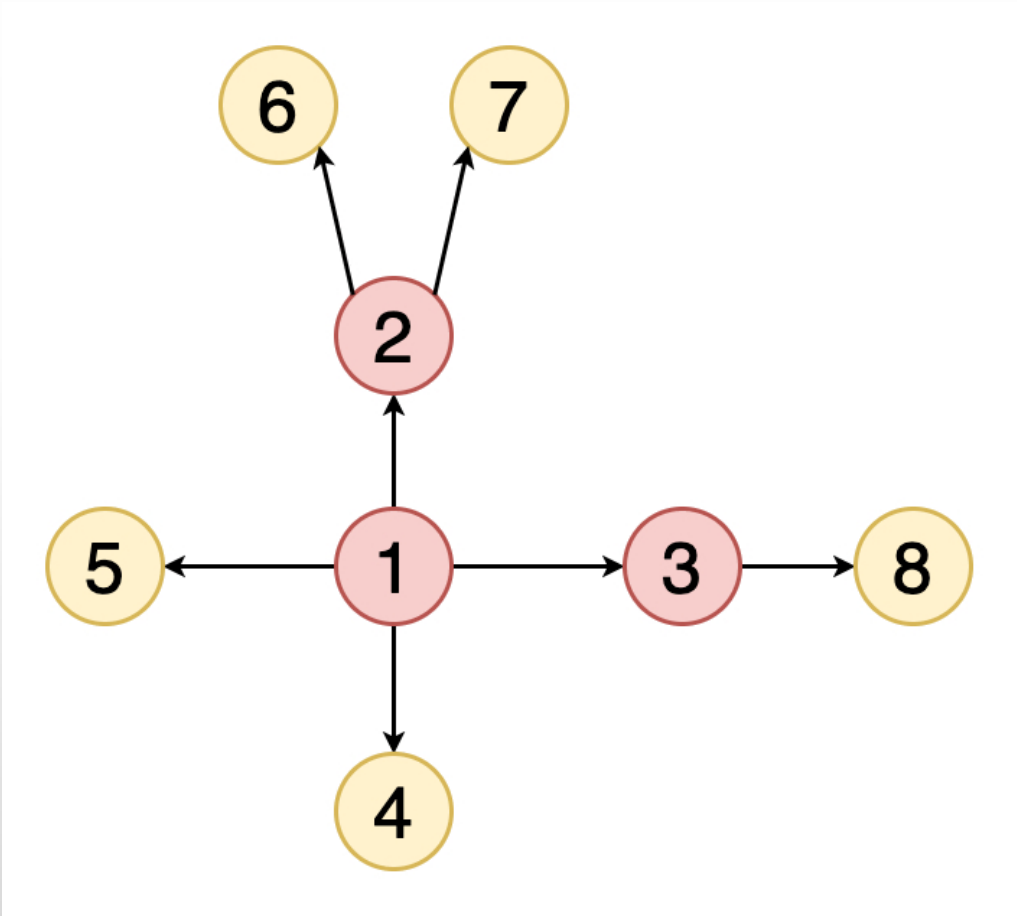
Genişlik Öncelikli Arama



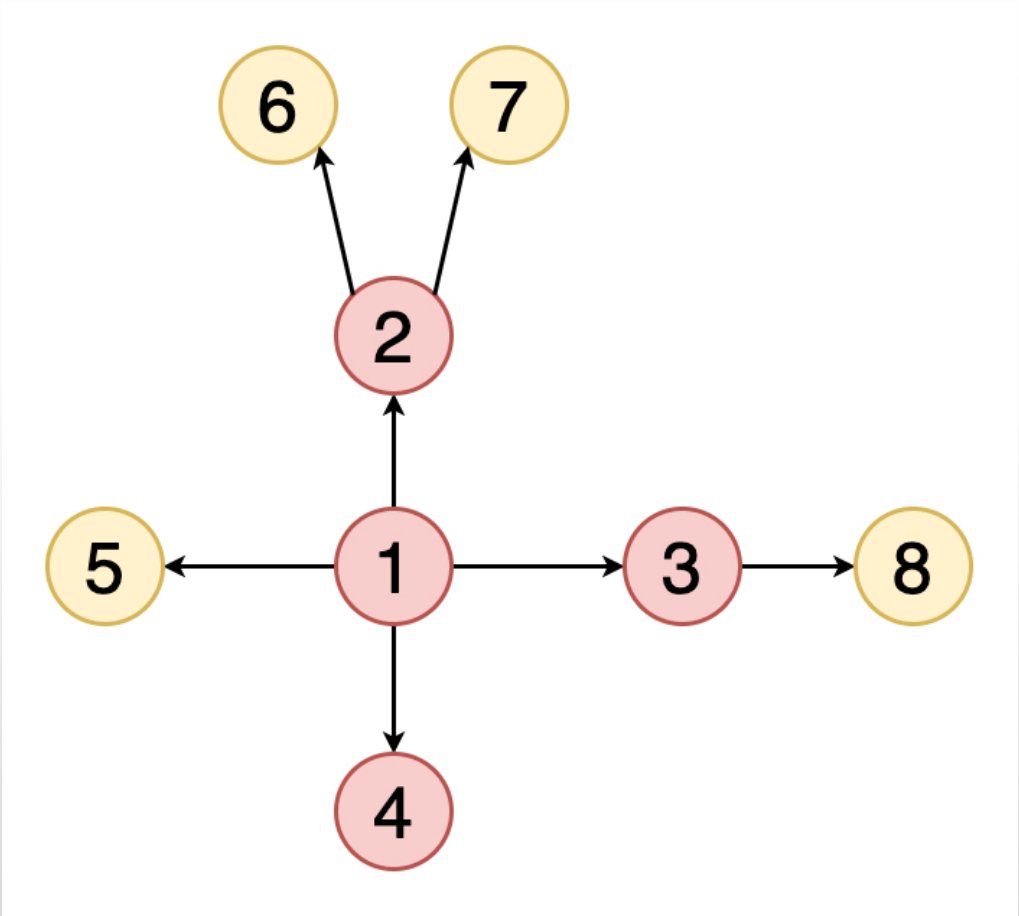
Genişlik Öncelikli Arama



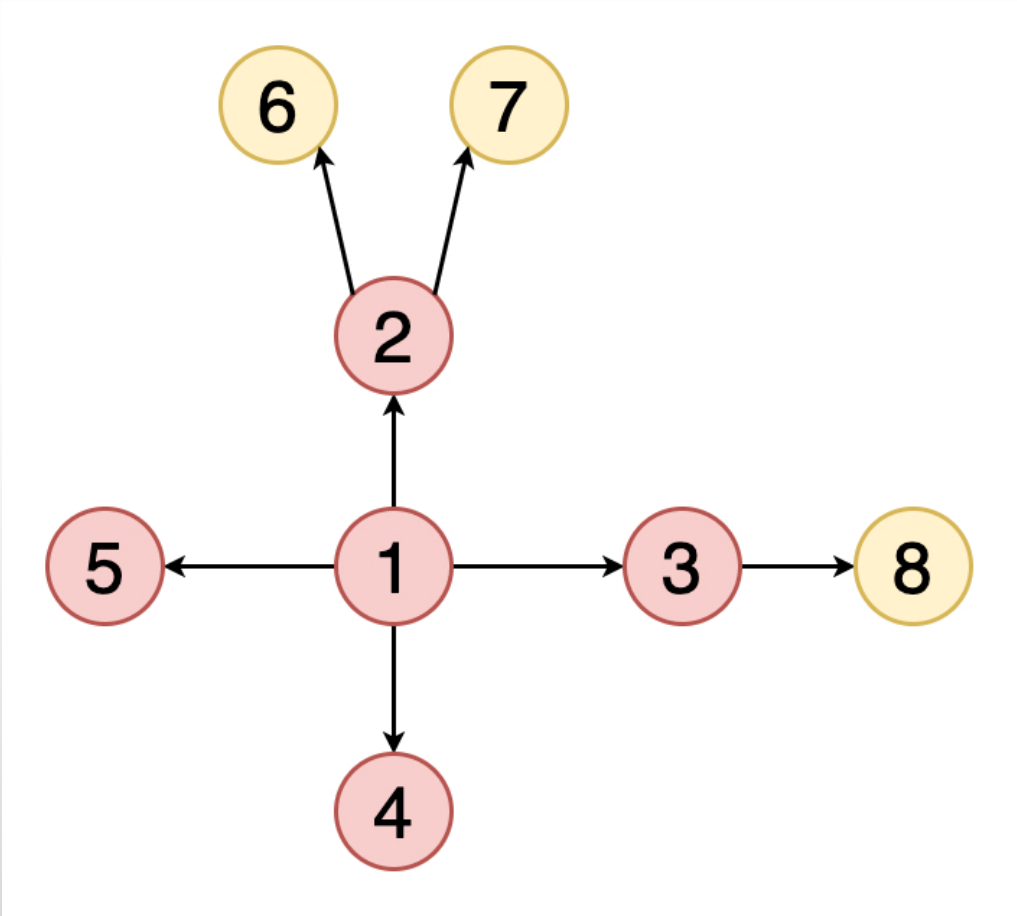
Genişlik Öncelikli Arama



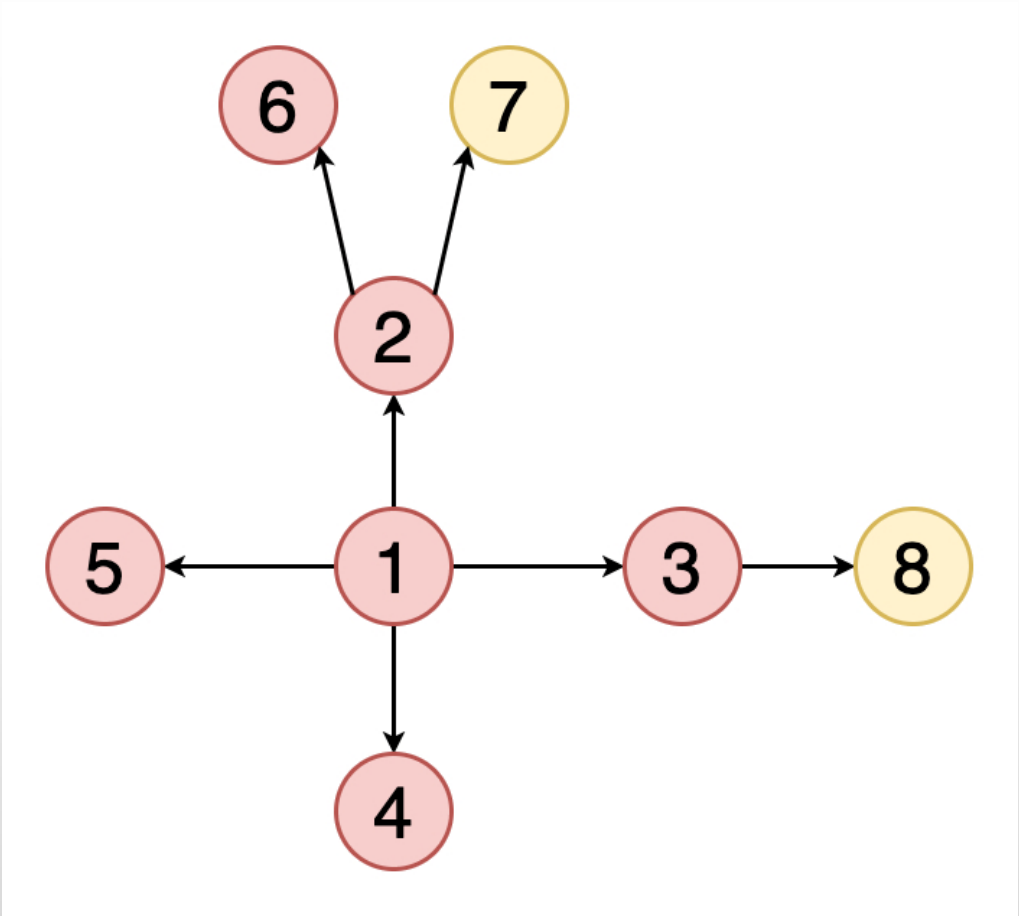
Genişlik Öncelikli Arama



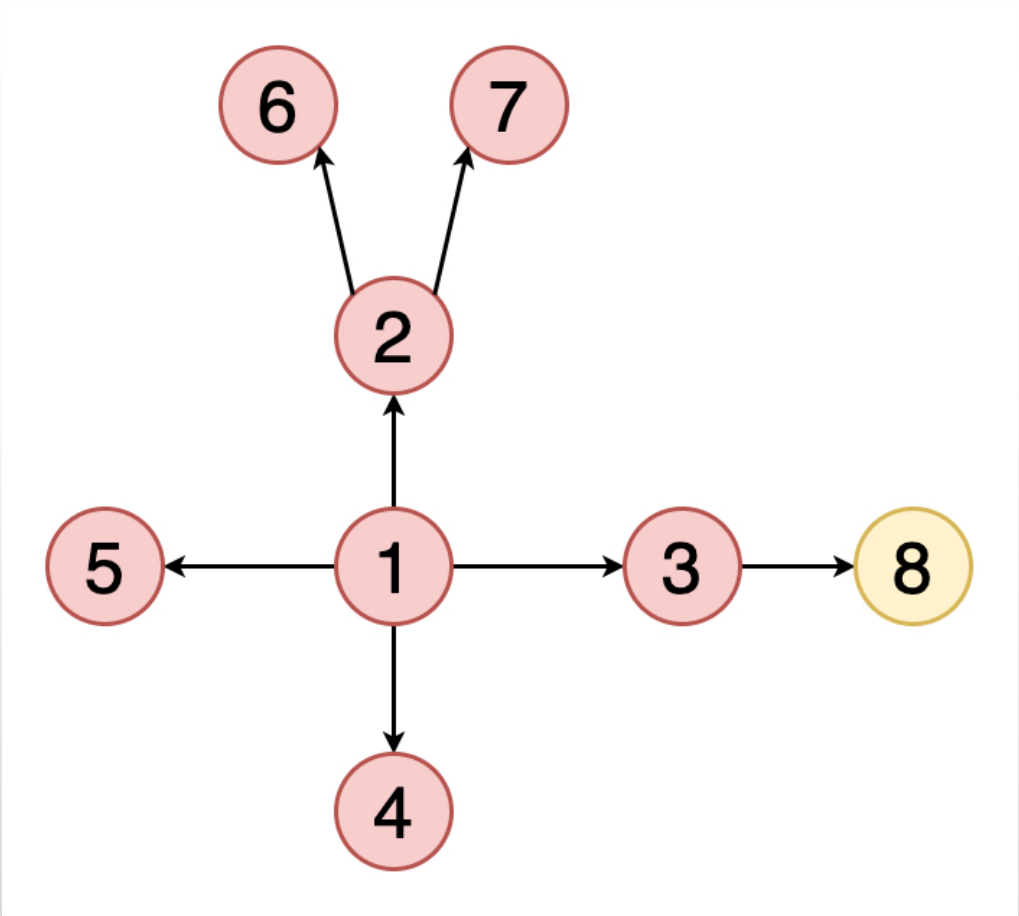
Genişlik Öncelikli Arama



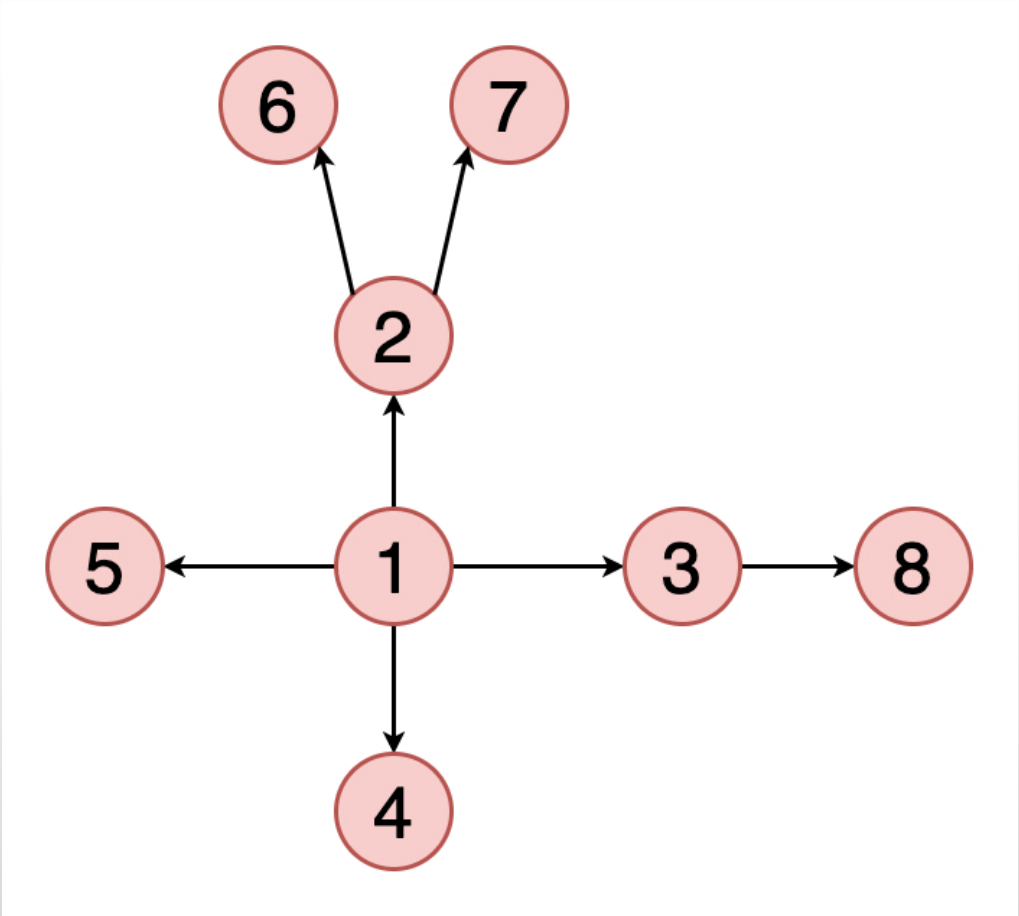
Genişlik Öncelikli Arama



Genişlik Öncelikli Arama



Genişlik Öncelikli Arama



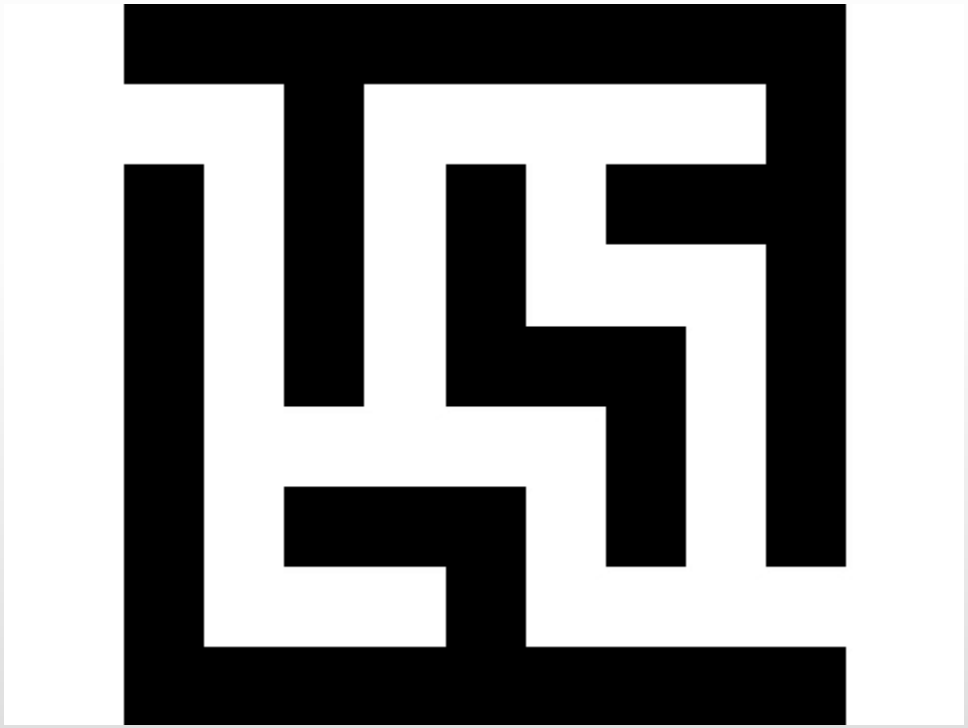
Iterative BFS

```
procedure BFS(G, root) is
        let Q be a queue
        label root as explored
        Q.enqueue(root)
        while Q is not empty do
                v := Q.dequeue()
                for all edges from v to w in G.adjacentEdges(v) do
                        if w is not labeled as explored then
                                label w as explored
                                w.parent := v
                                Q.enqueue(w)
```

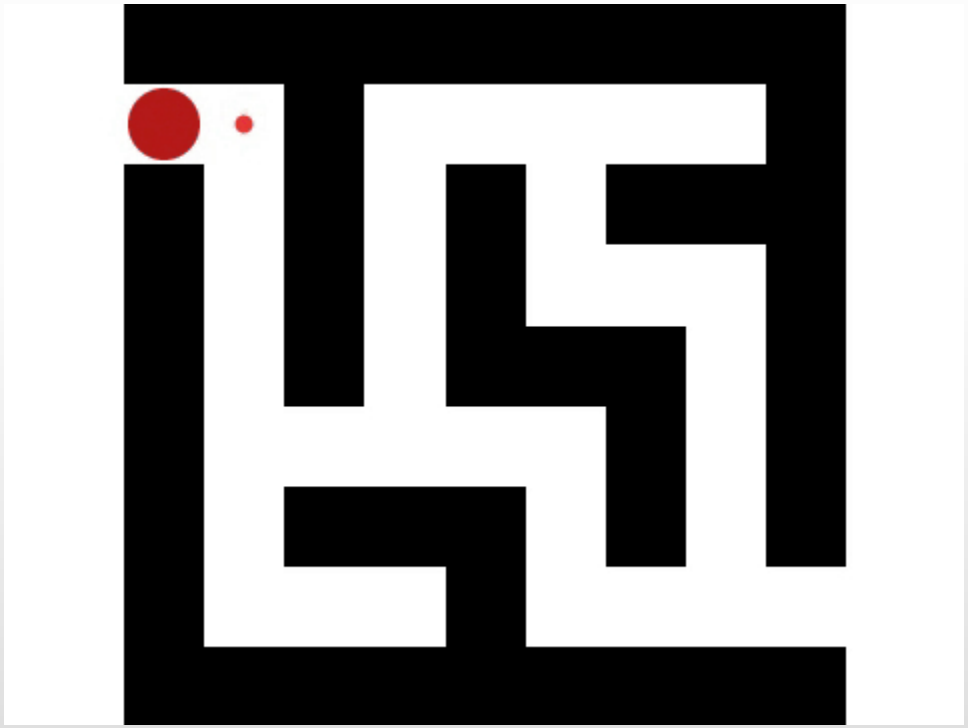
Iterative BFS

BFS (*başlangıç_düğüm*):
    boş bir kuyruk oluştur
    başlangıç_düğüm'ü keşfedildi işaretle
    başlangıç_düğüm'ü kuyruğa koy
    döngü kuyruk boş değilken:
        mevcut_düğüm = kuyruktan al
        her bir *komşu_düğüm* için
            eğer *komşu_düğüm* keşfedilmemiş ise
                komşu_düğüm'ü keşfedildi işaretle
                *komşu_düğüm'*ün atası = *mevcut_düğüm*
                komşu_düğüm'ü kuyruğa koy

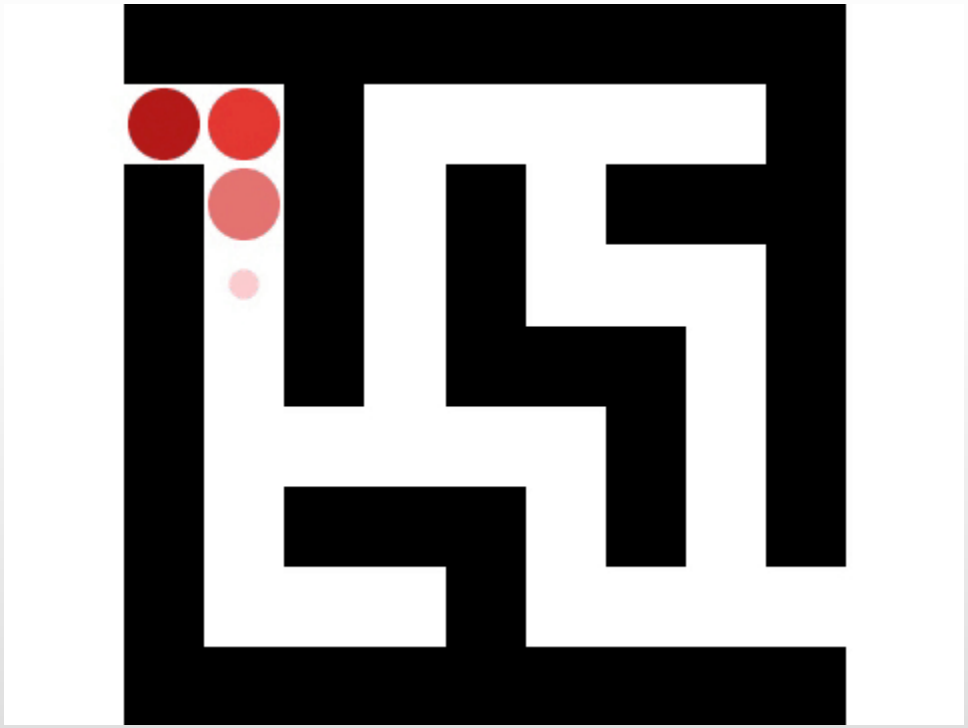
BFS Search Way



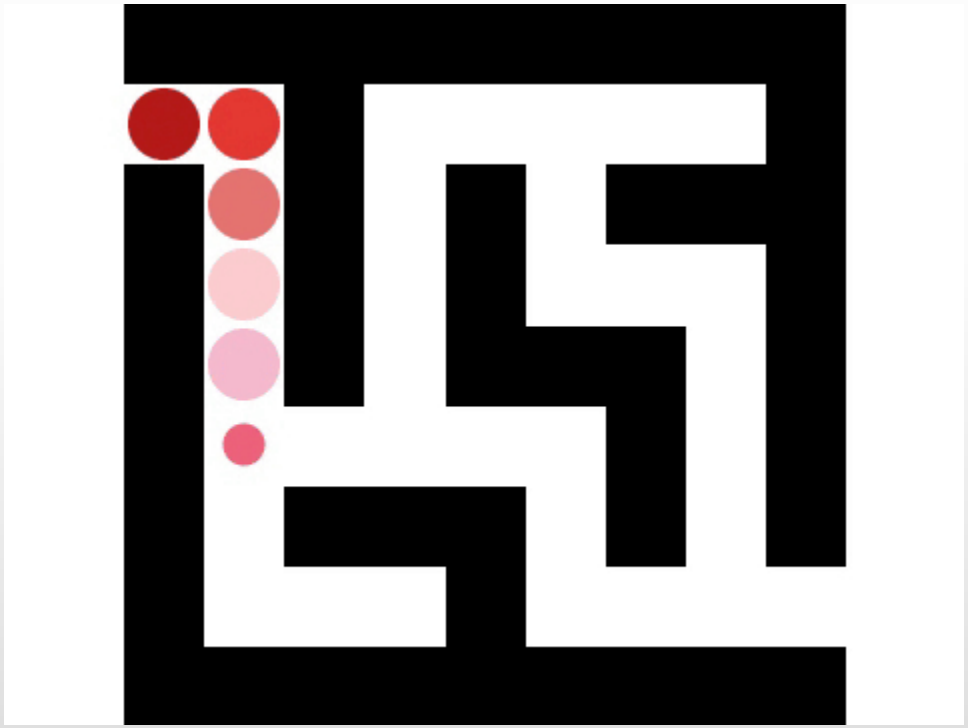
BFS Search Way



BFS Search Way

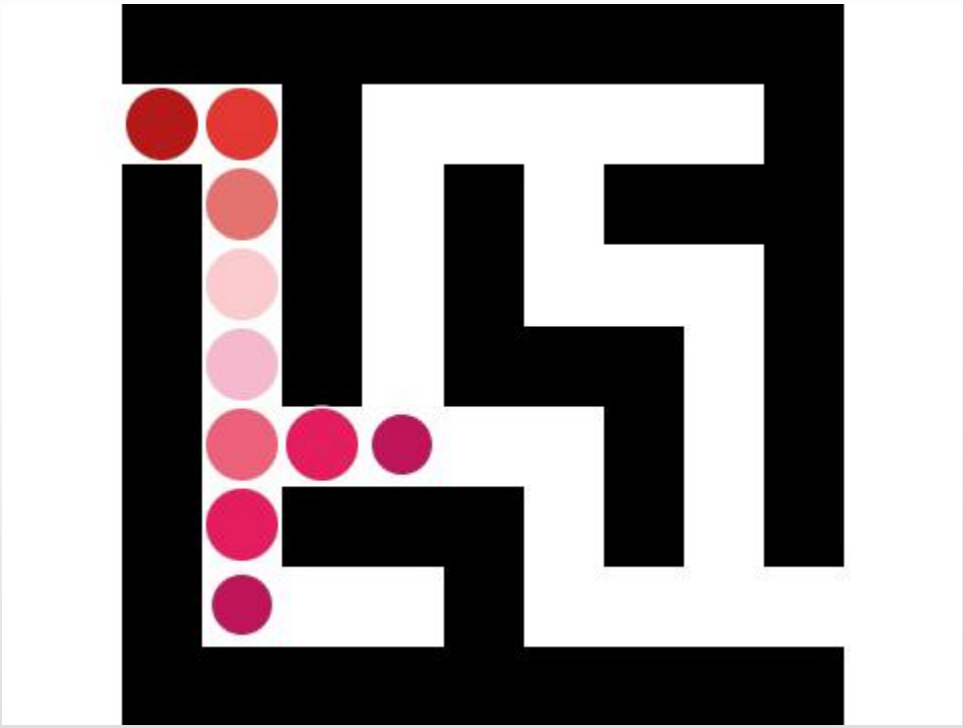


BFS Search Way



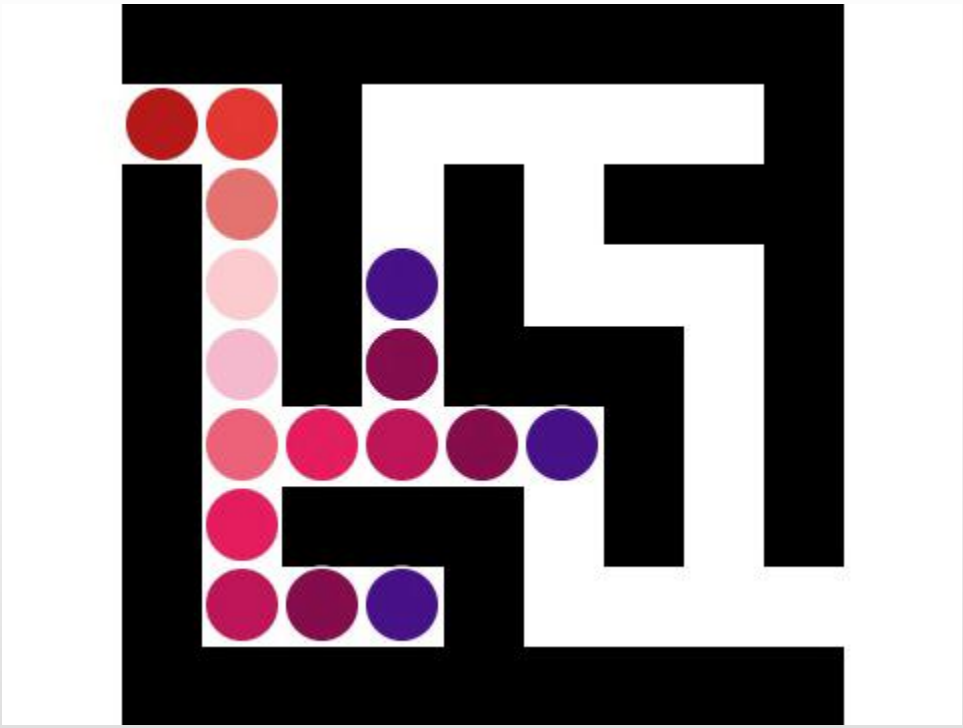
BFS Search Way

- Var olan tüm yolları eşzamanlı dene.

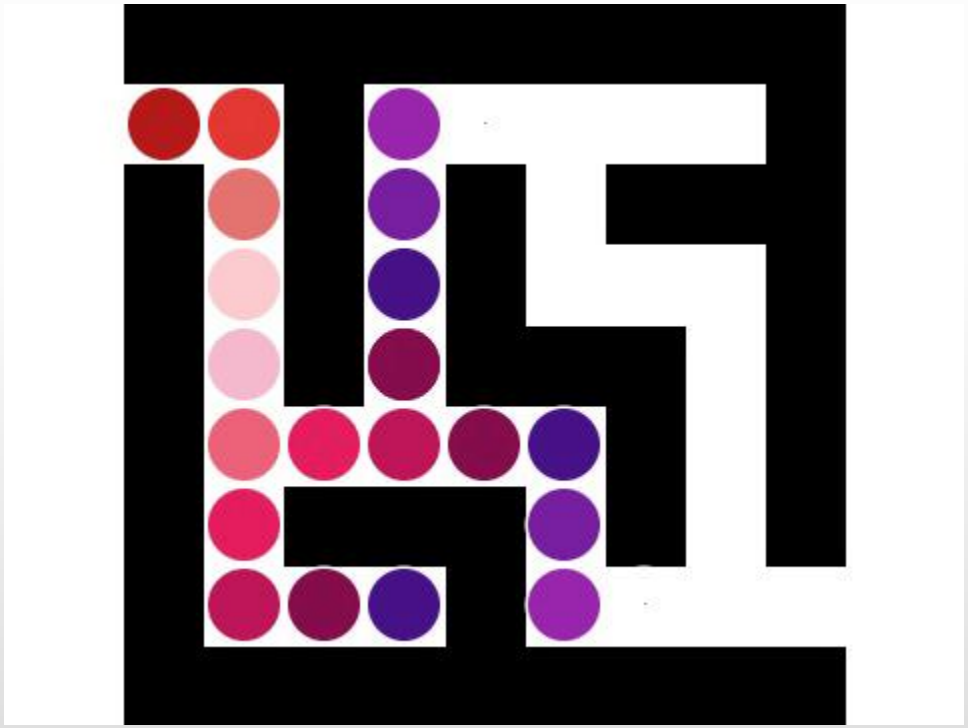


BFS Search Way

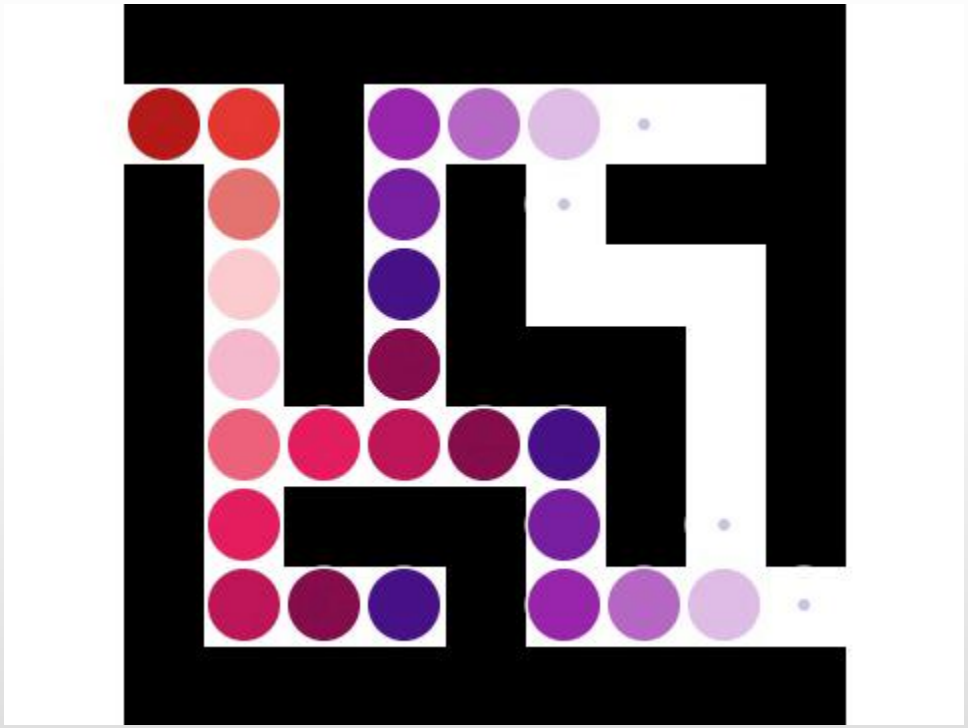
- Var olan tüm yolları eşzamanlı dene.



BFS Search Way



BFS Search Way



Tarjan'ın Güçlü Bağlantılı Bileşenler Algoritması

- Bir yönlü çizgede güçlü bağlantılı bileşenleri bulur.
- Bir SCC, çizgede her düğümün birbirine ulaşabildiği bir düğüm kümesidir.
- Güçlü bağlantılı bileşenler, (*strongly connected components*)
  - iki düğüm arasında hem ileri hem de geri yönlü yollar bulunur.
  - bir düğüm ve ona erişilebilen tüm düğümleri içeren alt çizge.

Algoritma İlkeleri

- DFS (*Depth-First Search*) tabanlı bir algoritmadır.
- Çizgenin tüm düğümlerini sıra ile ziyaret eder.
- Ziyaret edilen her bir düğüm için,
  - o düğümden ulaşılabilen en düşük düğüm numarası hesaplanır.

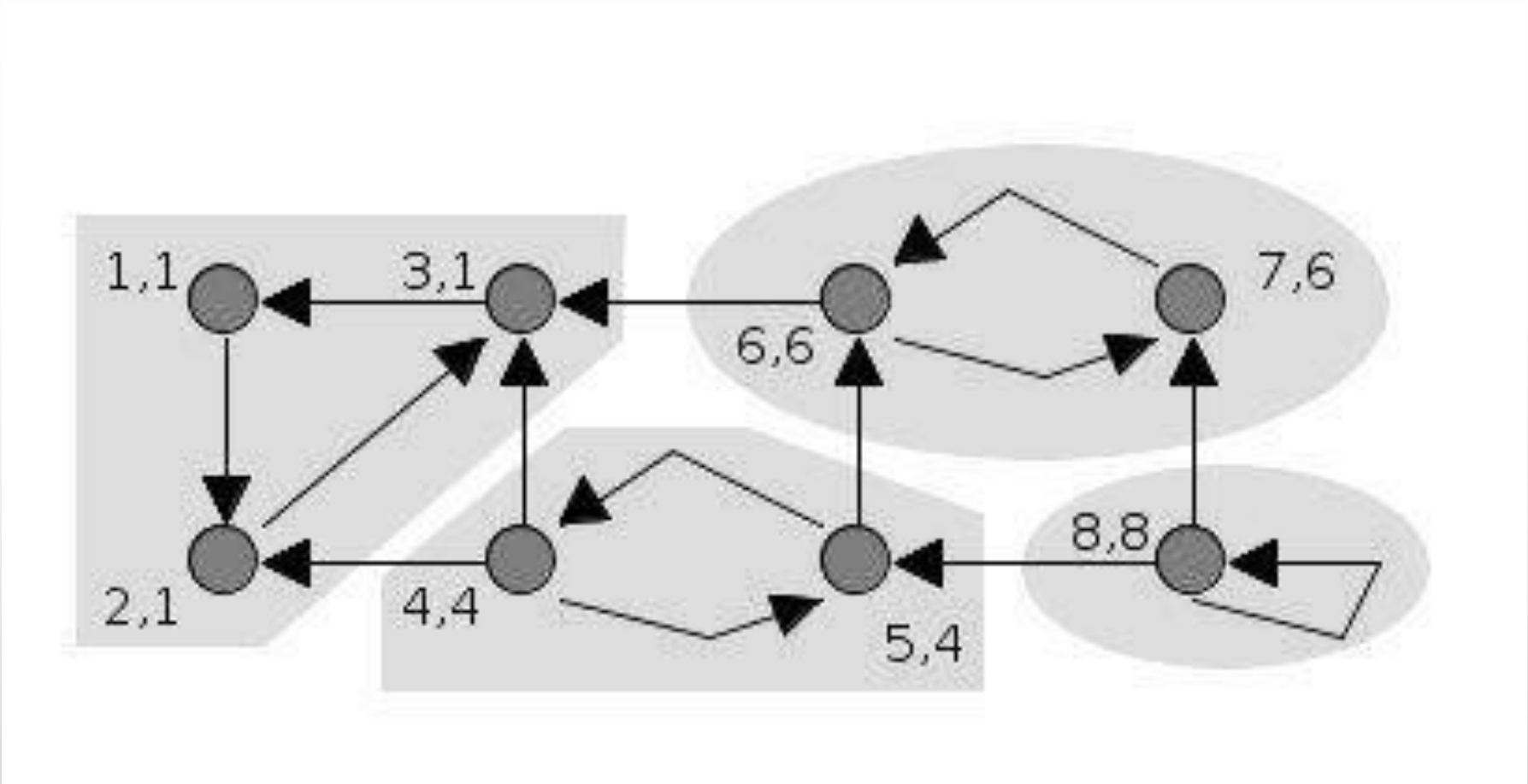
Algoritma Adımları

- Adım 1: Her bir düğüm DFS algoritması ile ziyaret edilir.
- Adım 2: DFS sırasında, ziyaret edilen her düğüme bir indis değeri atanır.
  - İndis, düğümün çizge içindeki konumunu temsil eder.
- Adım 3: Her bir düğüm için, o düğümden ulaşılabilen en düşük düğüm indis değeri hesaplanır.
  - Bu, çizgedeki ileri ve geri yönlü yolları kontrol eder.
- Adım 4: Düğüm indisleri (*index*) ve en düşük düğüm indis (*low index*) değerleri kullanılarak güçlü bağlantılı bileşenler bulunur.
  - SCC, kök düğümün en düşük indisi ile kendi indisi eşleştiğinde bulunur.
  - Mevcut DFS yolundaki düğümler yığında tutulur.

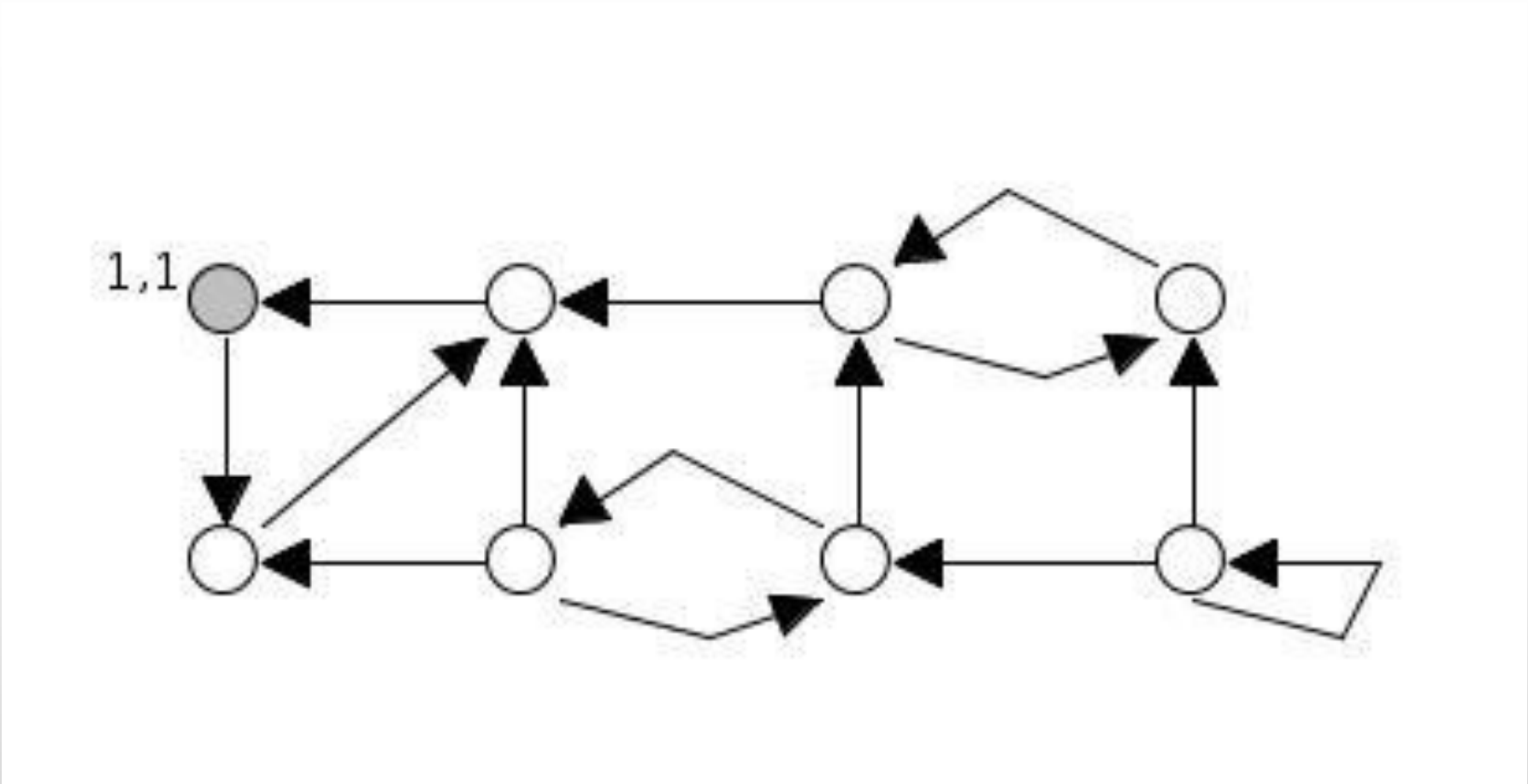
Karmaşıklık Analizi

- Tarjan'ın Algoritması'nın karmaşıklığı,
  - *O(V+E)*
    - V düğüm sayısı
    - E kenar sayısı

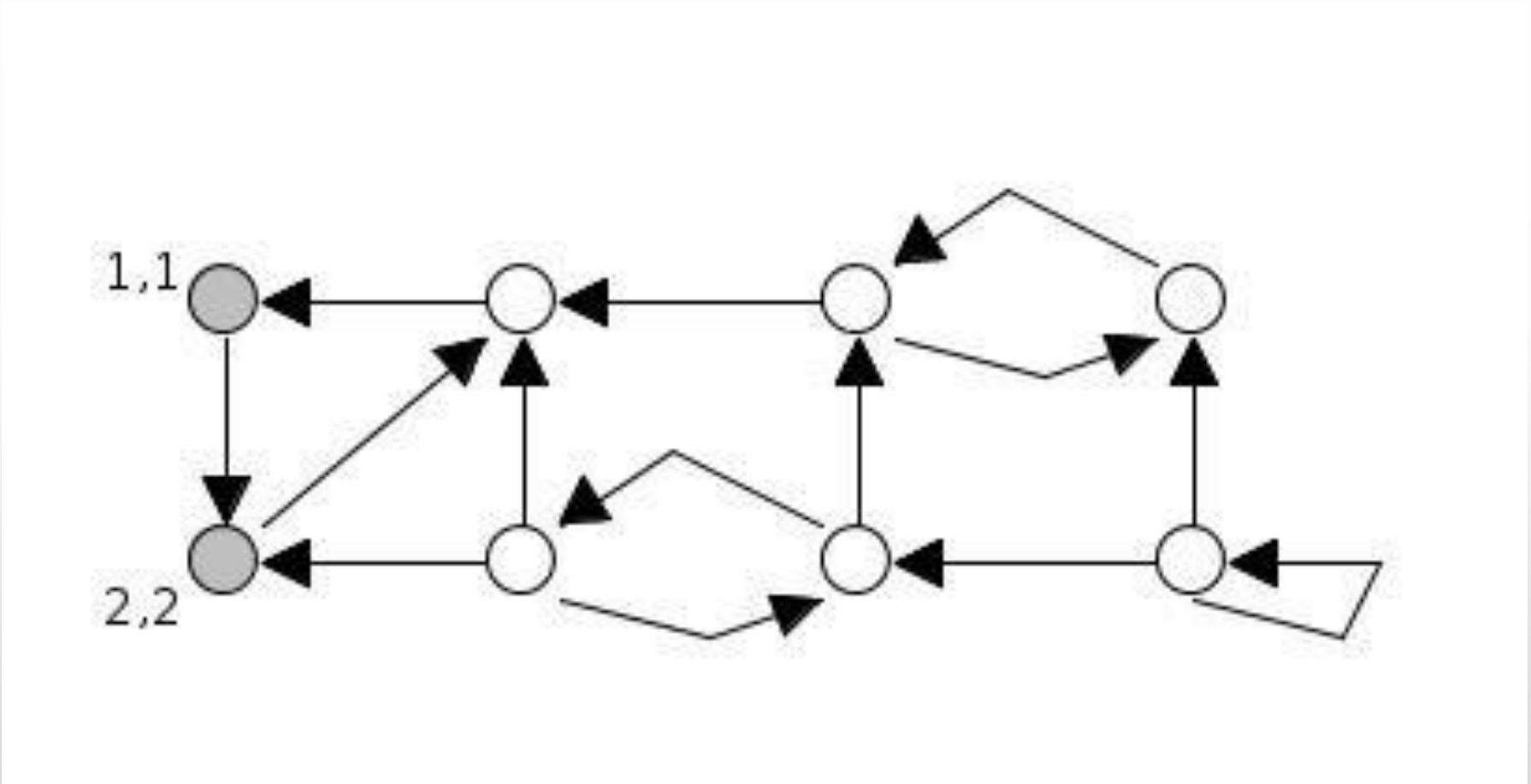
Tarjan's Strongly Connected Components



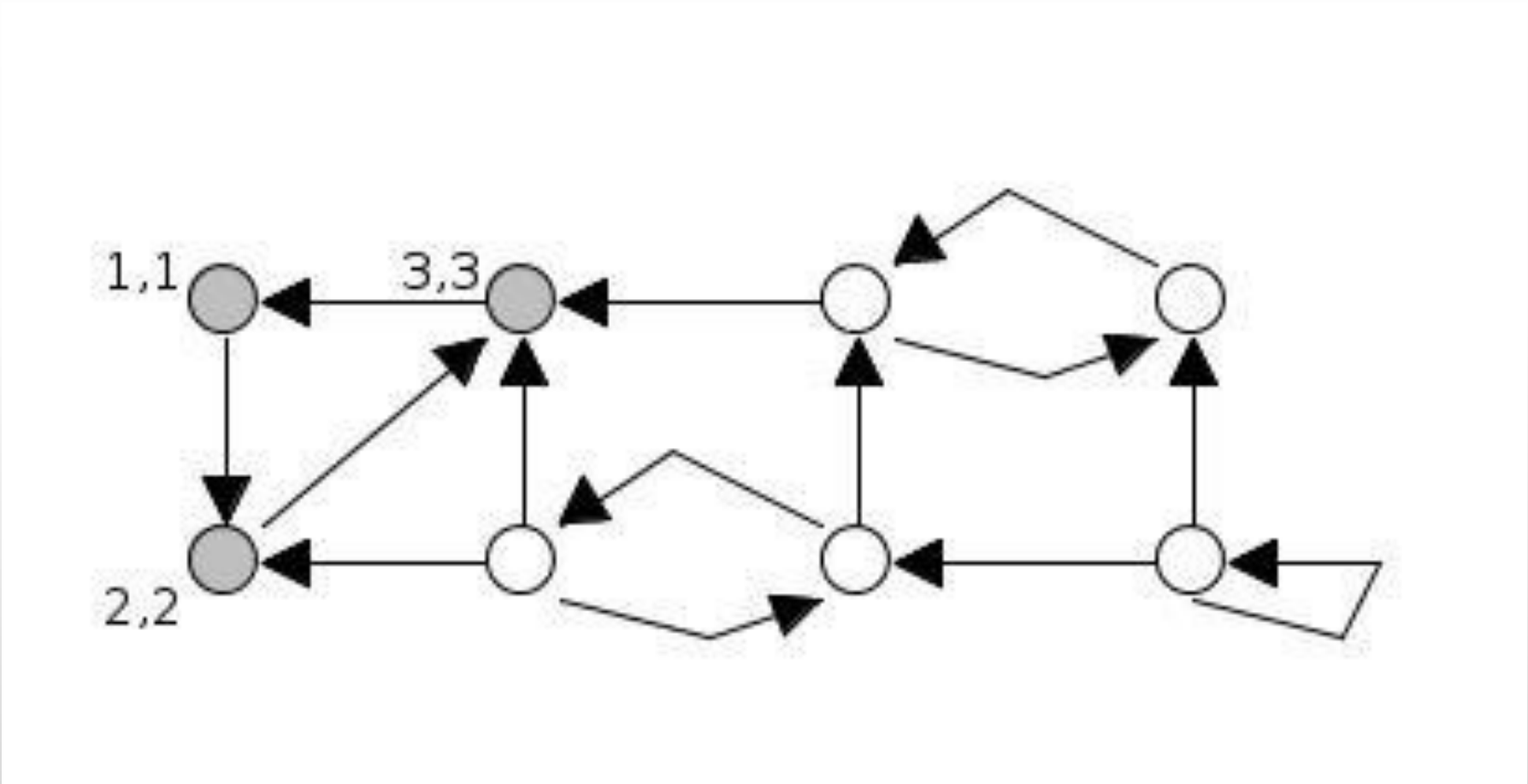
Tarjan's Strongly Connected Components



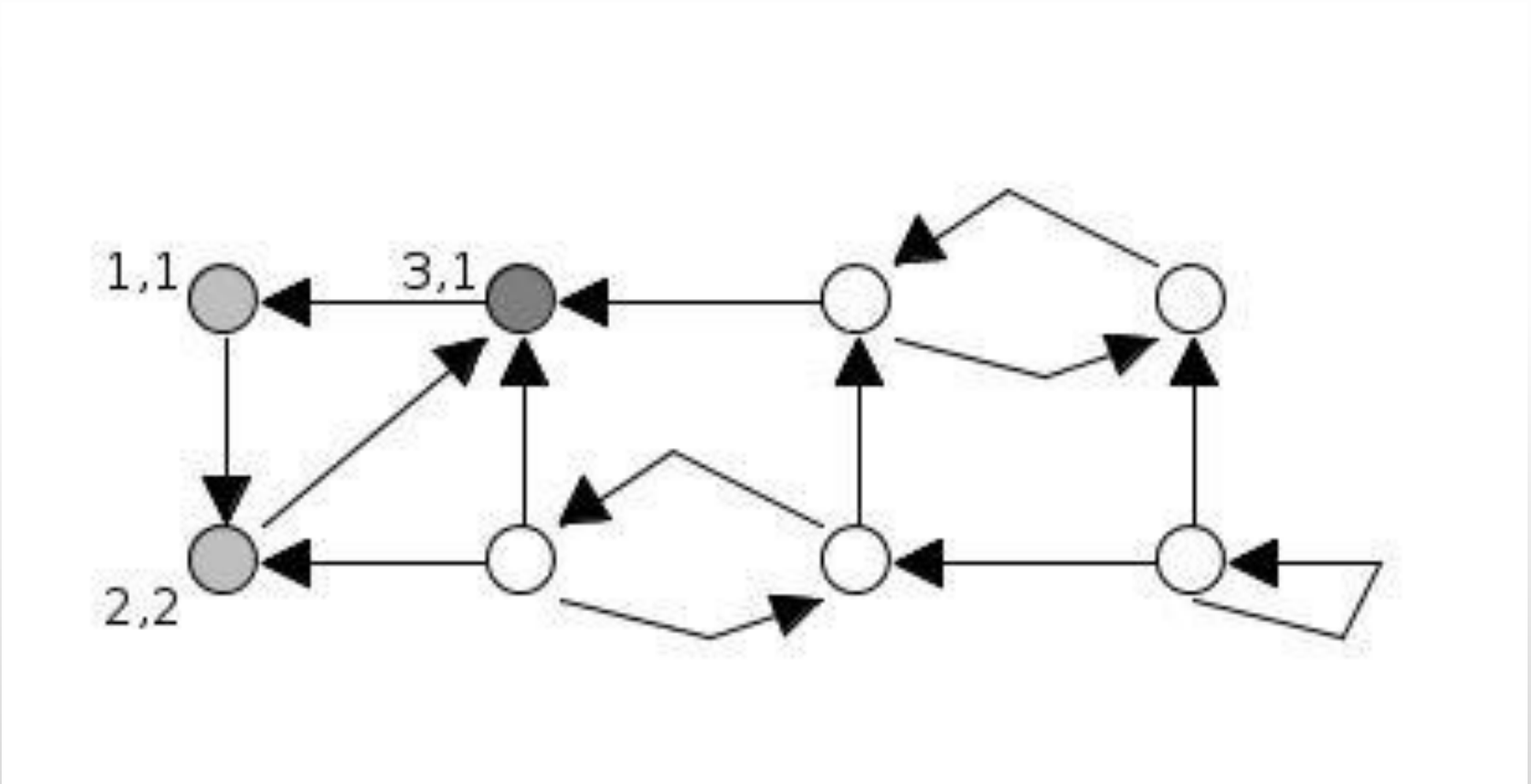
Tarjan's Strongly Connected Components



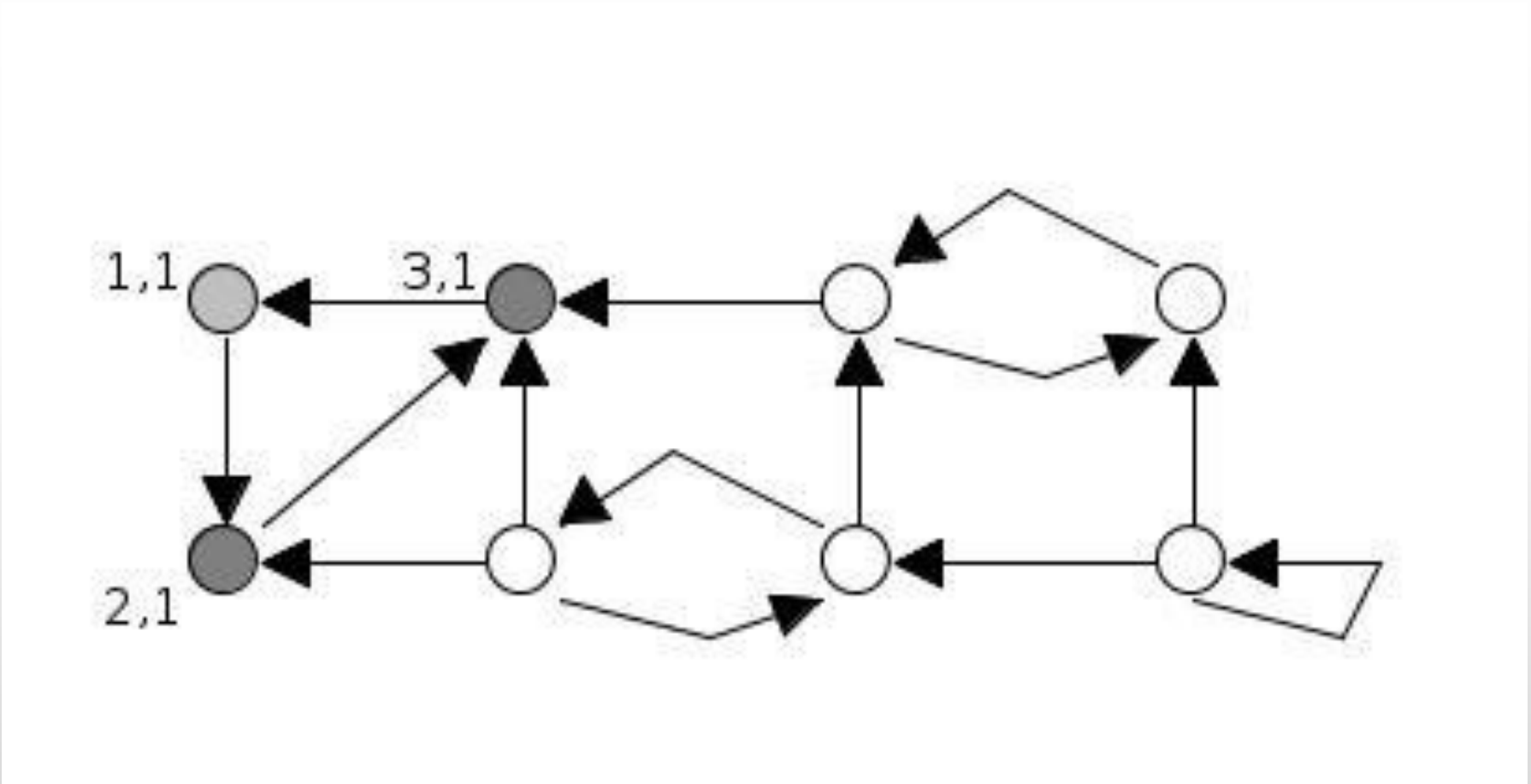
Tarjan's Strongly Connected Components



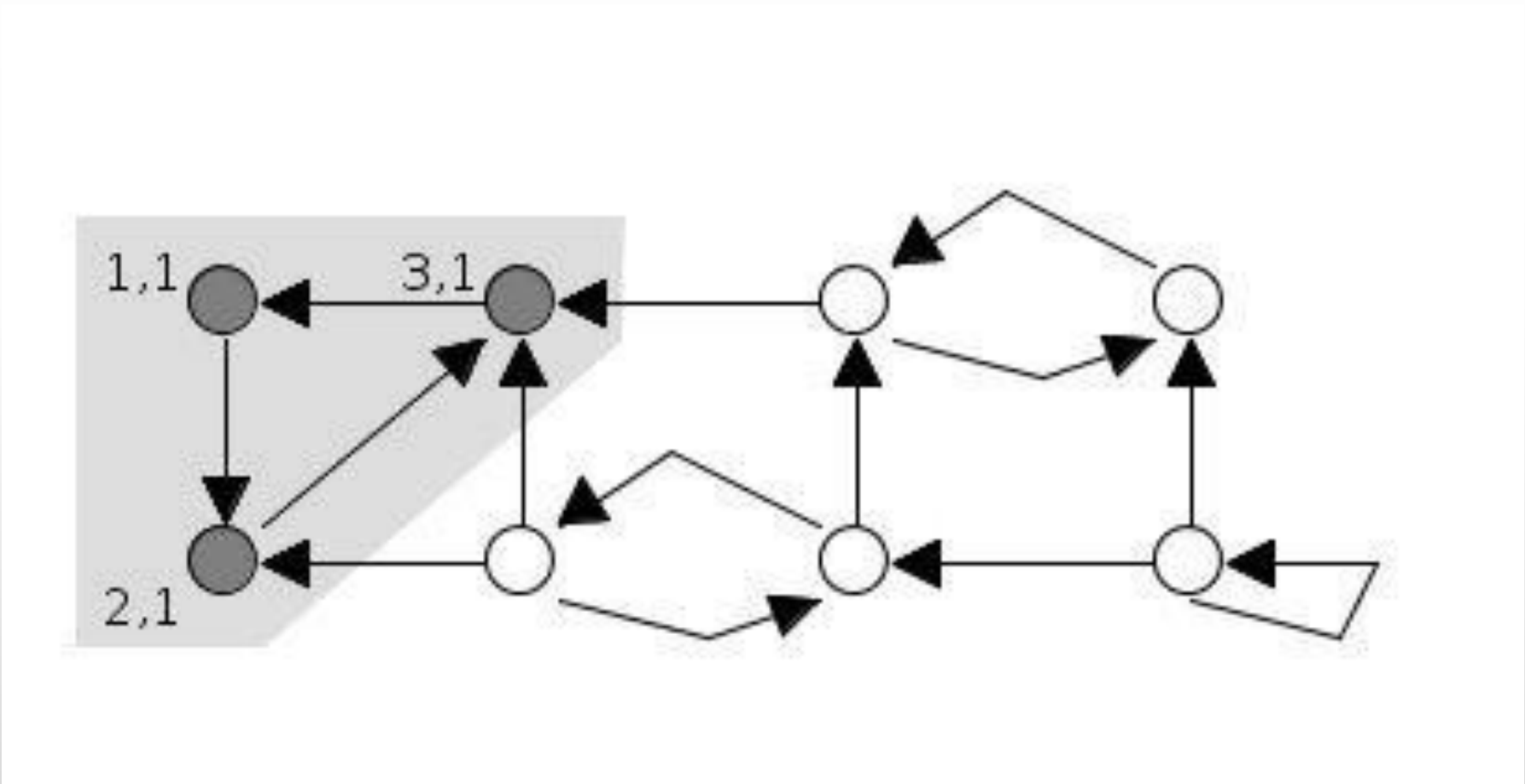
Tarjan's Strongly Connected Components



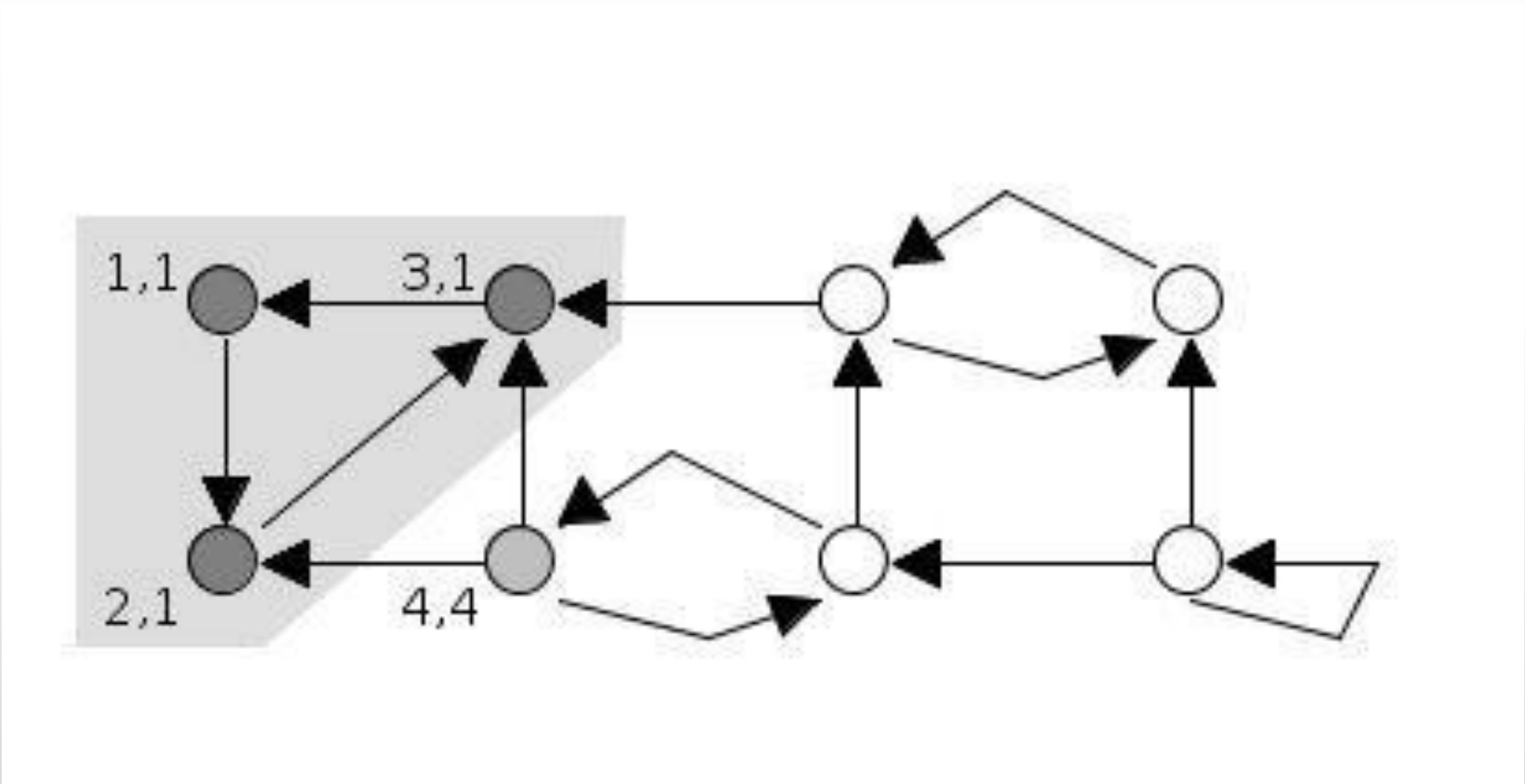
Tarjan's Strongly Connected Components



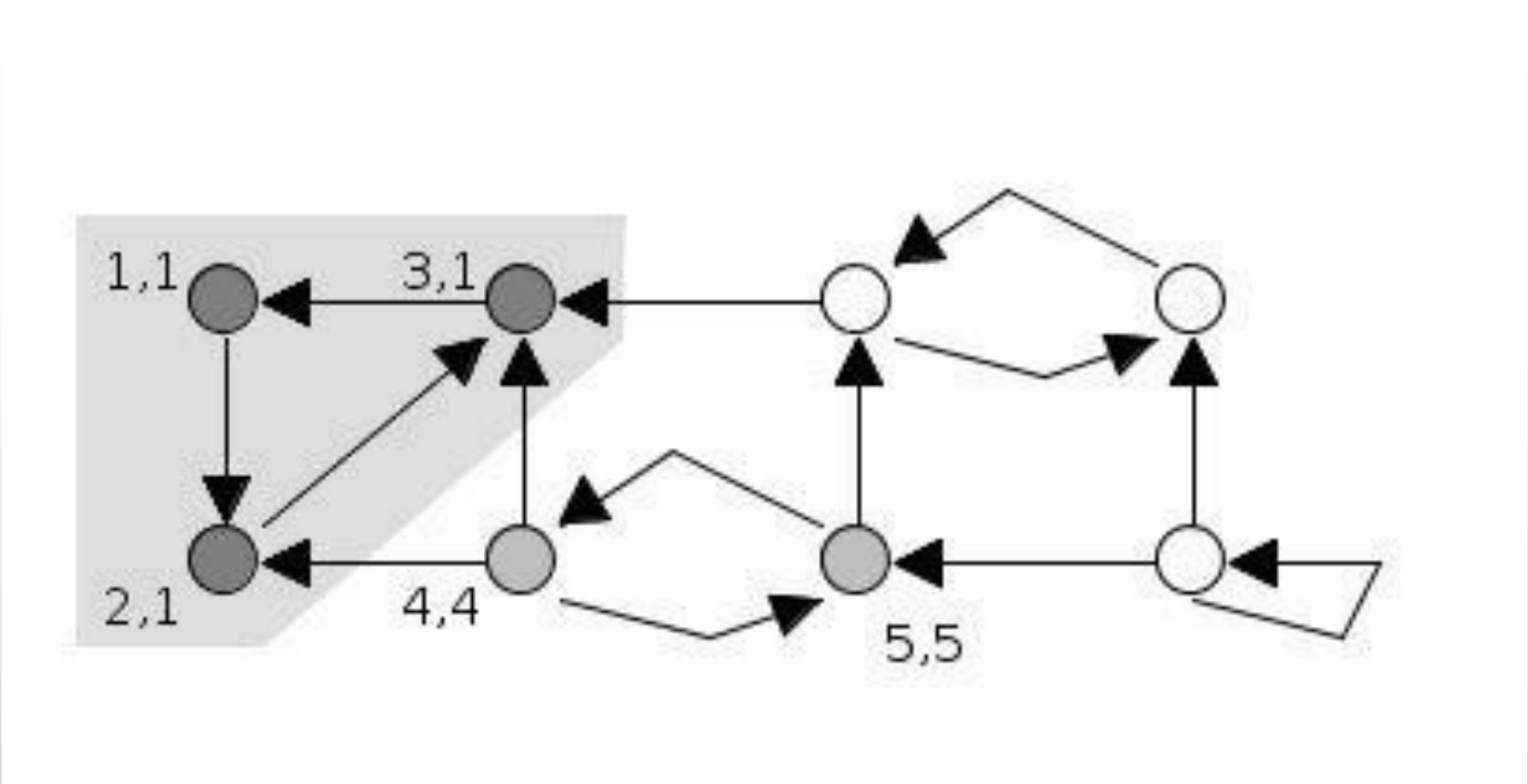
Tarjan's Strongly Connected Components



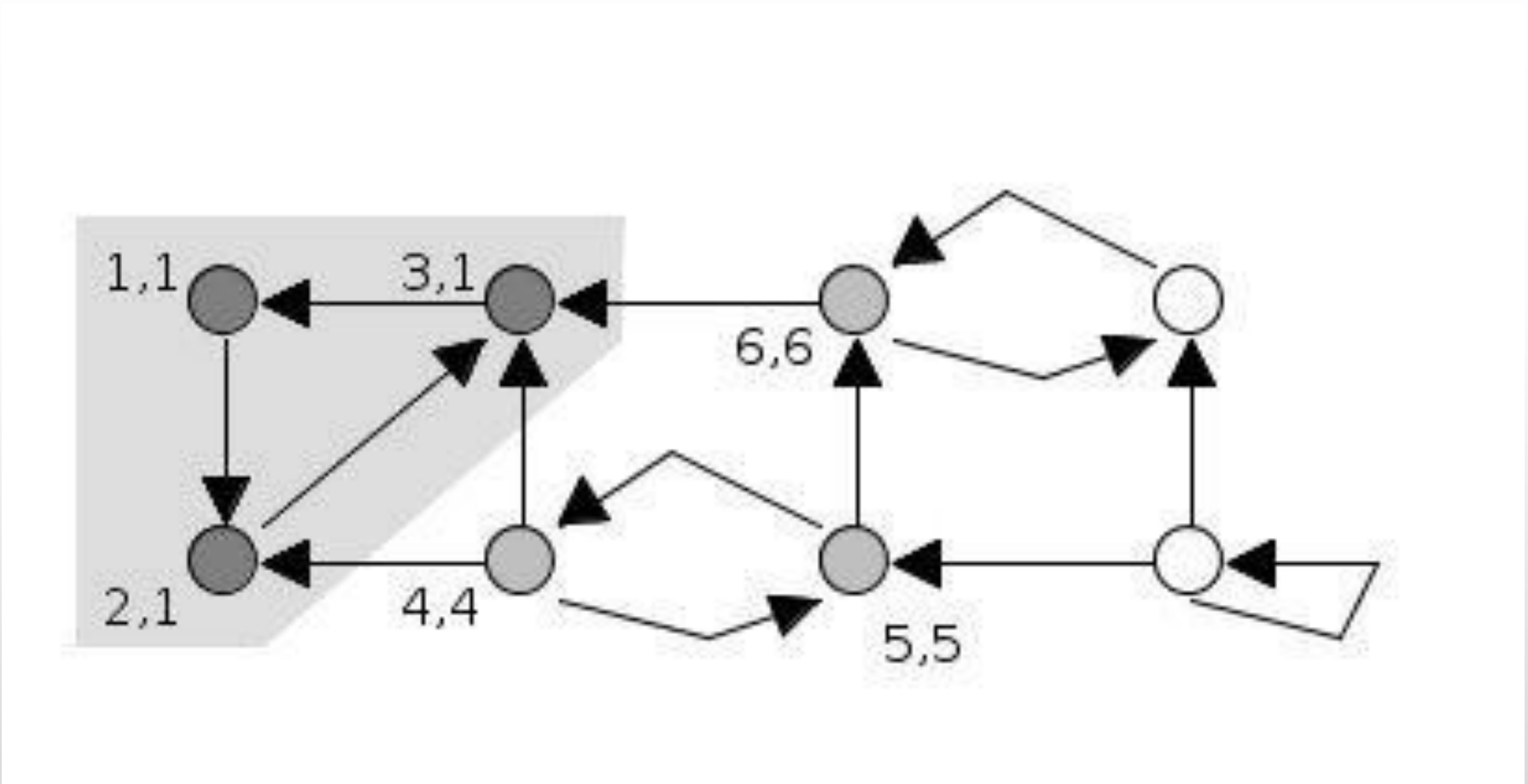
Tarjan's Strongly Connected Components



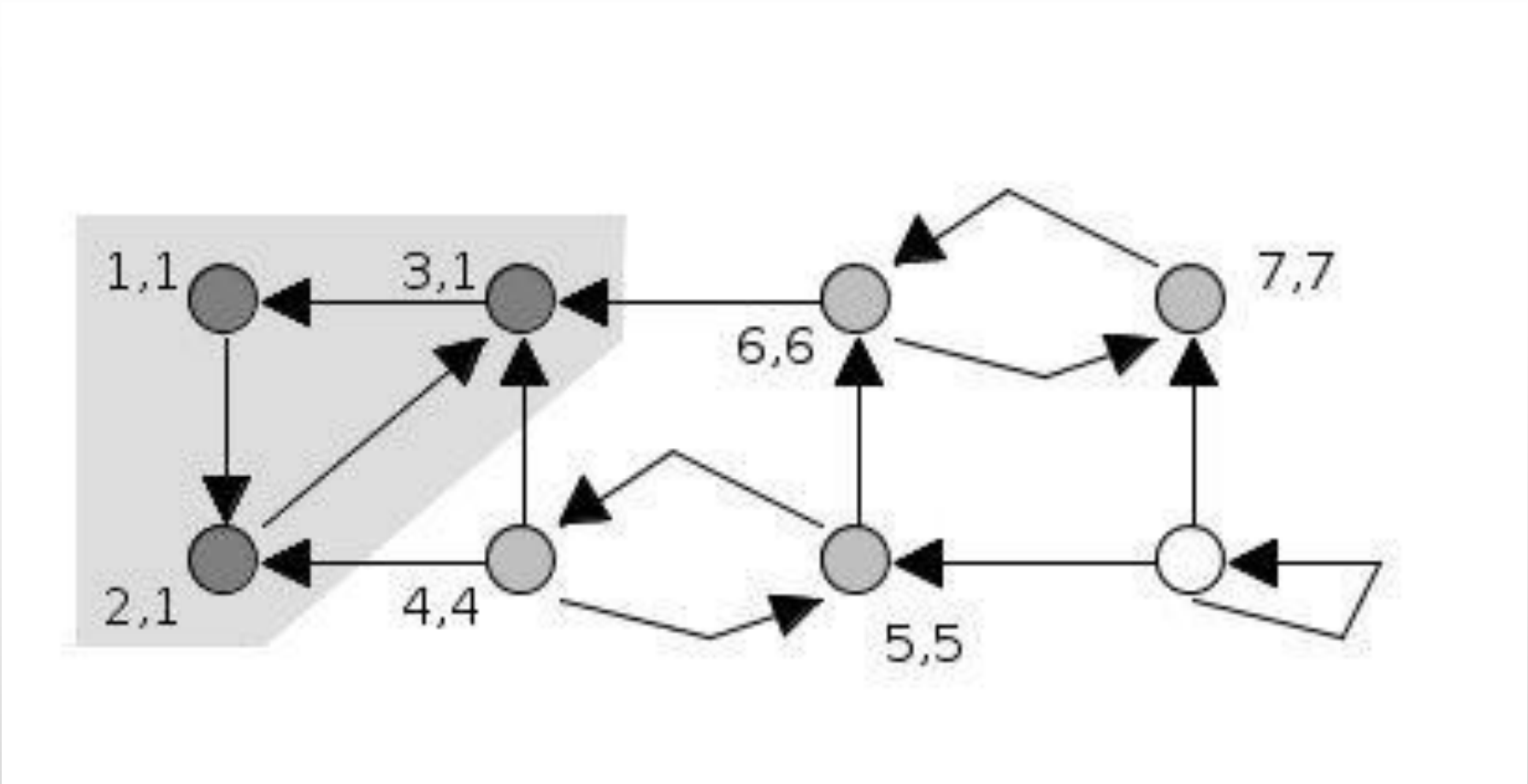
Tarjan's Strongly Connected Components



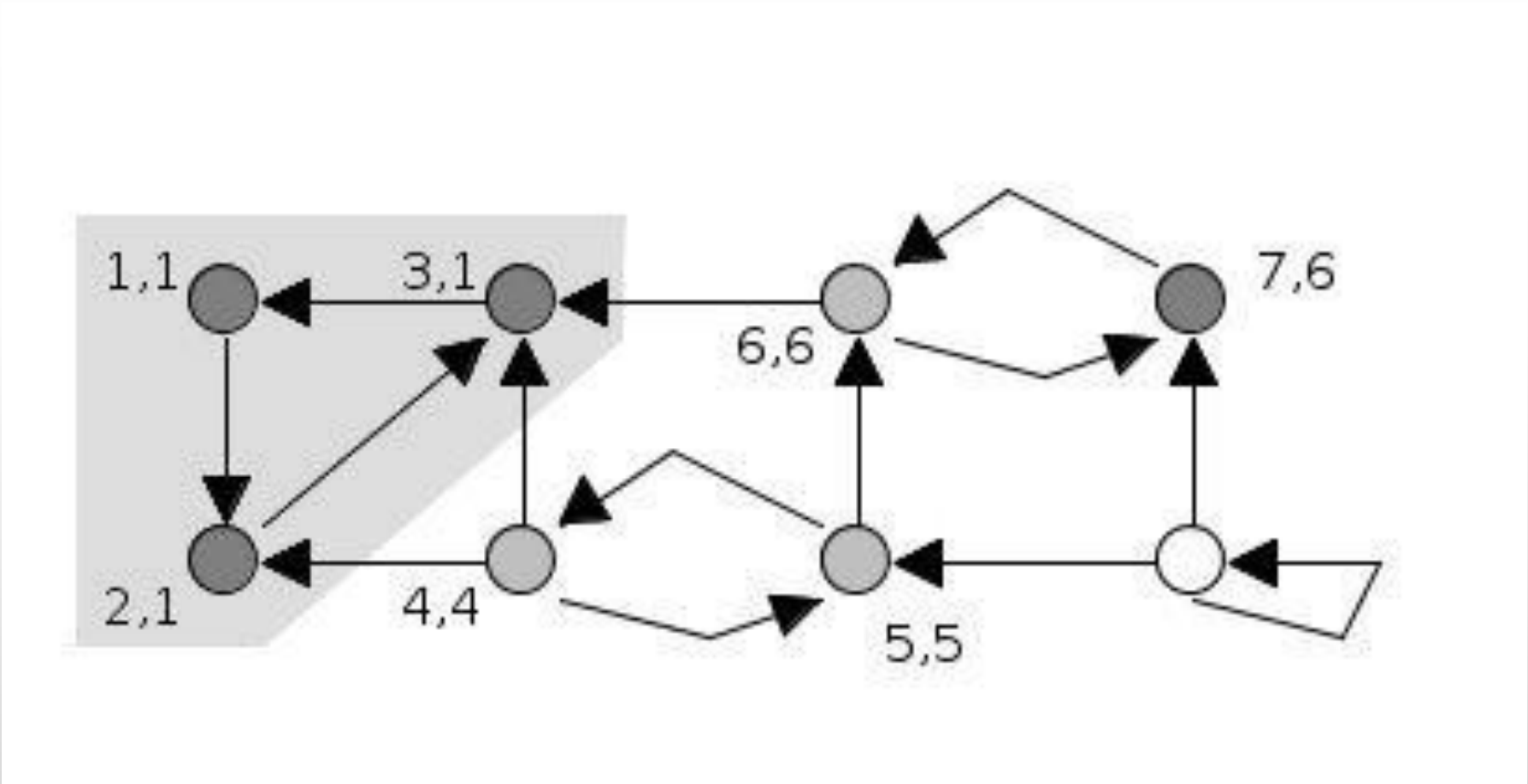
Tarjan's Strongly Connected Components



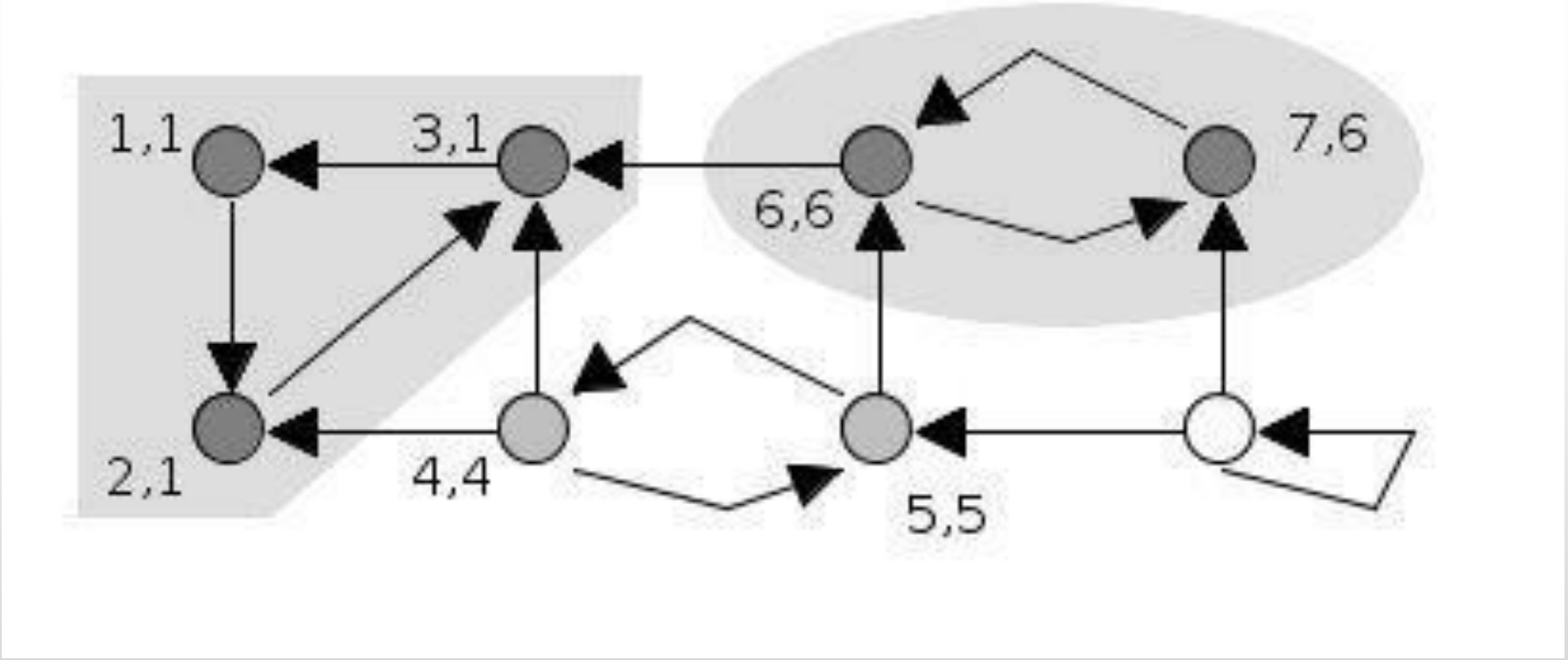
Tarjan's Strongly Connected Components



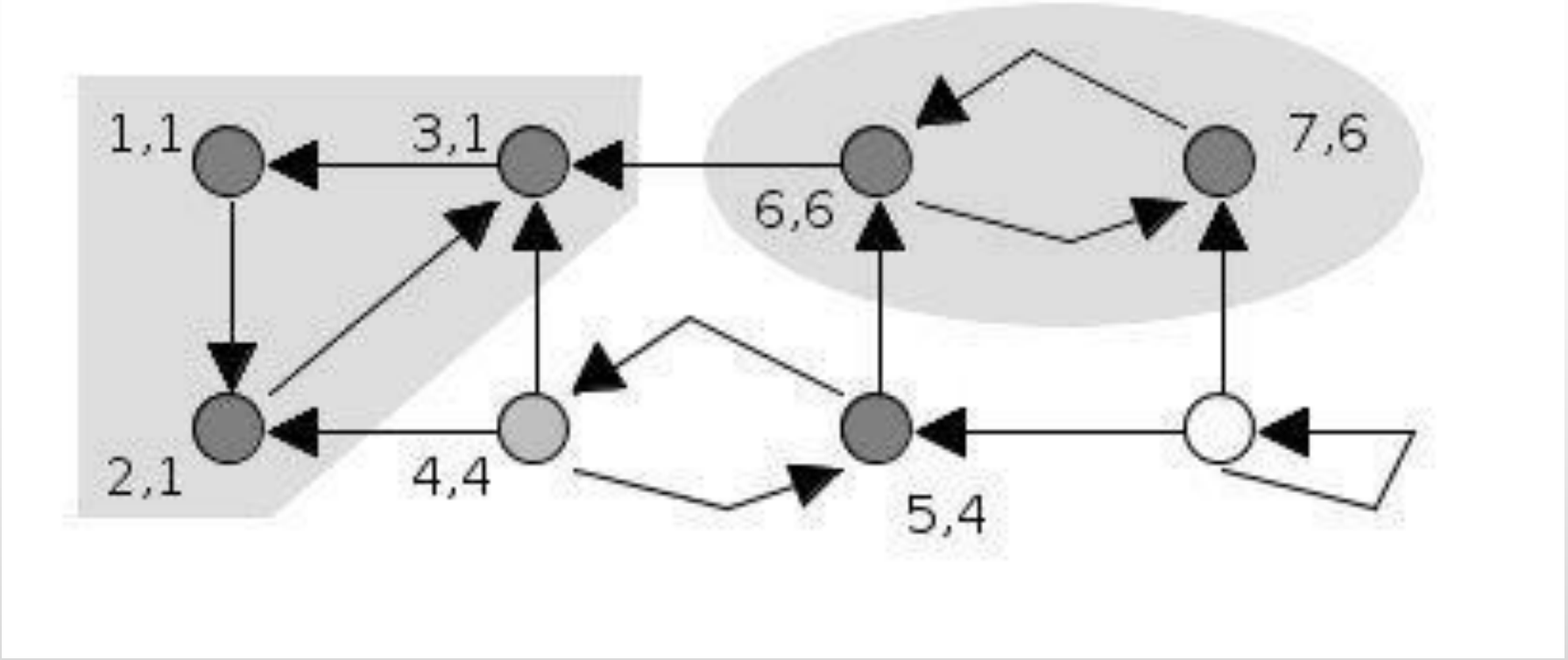
Tarjan's Strongly Connected Components



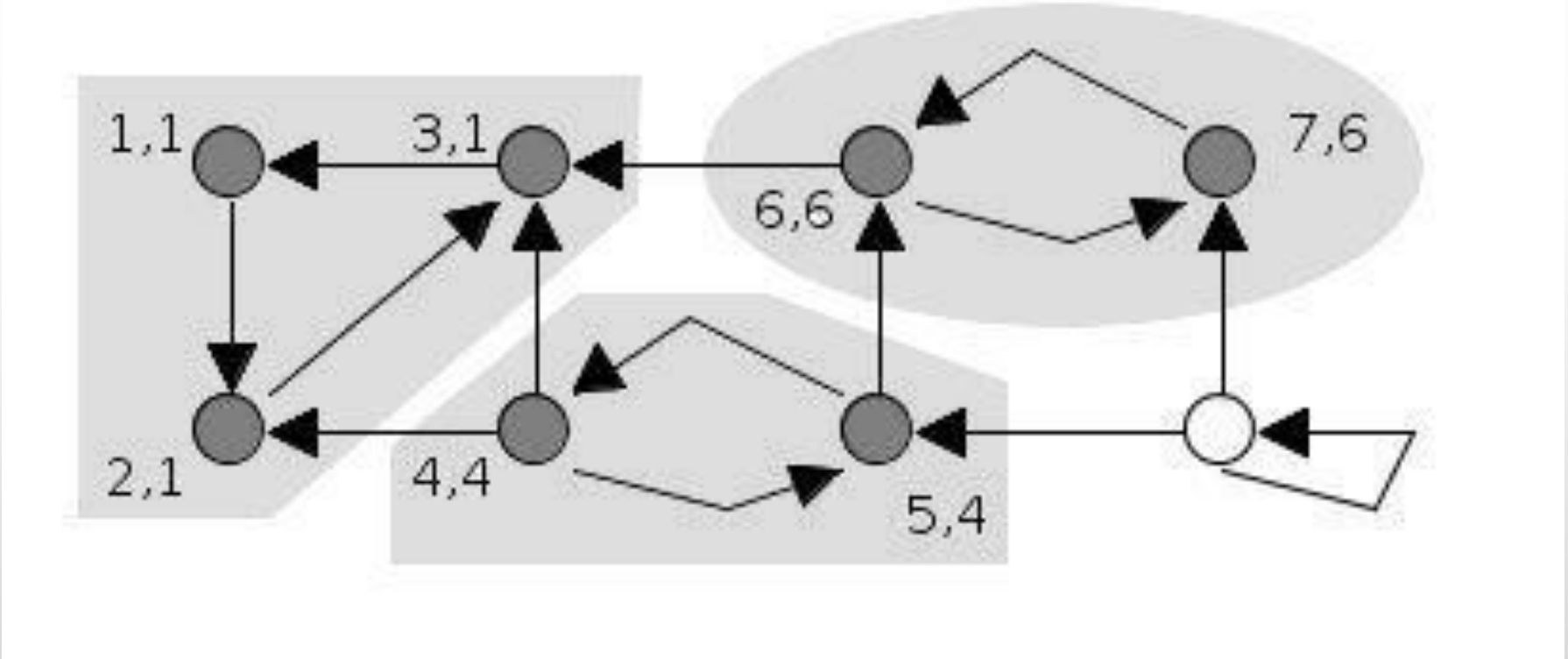
Tarjan's Strongly Connected Components



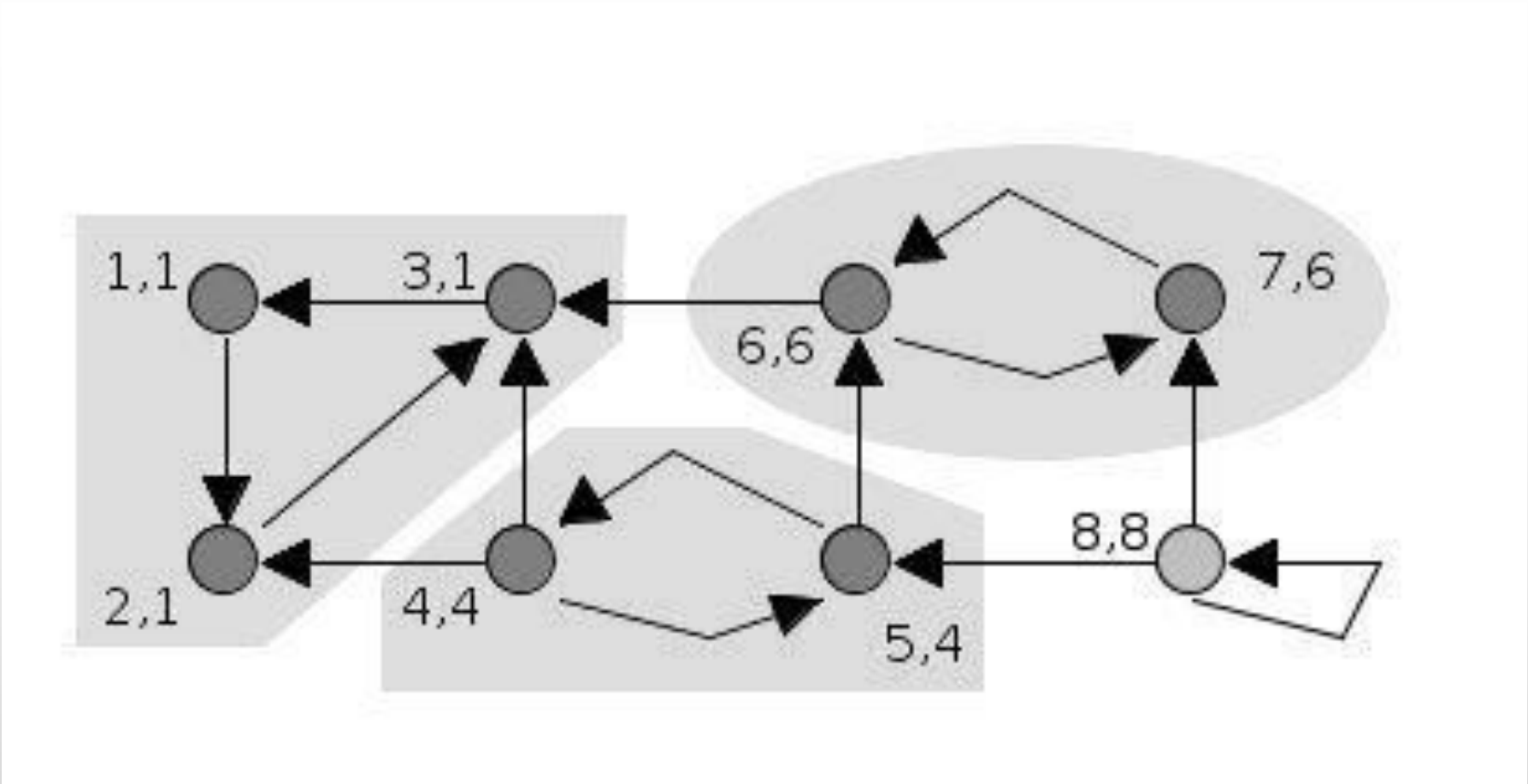
Tarjan's Strongly Connected Components



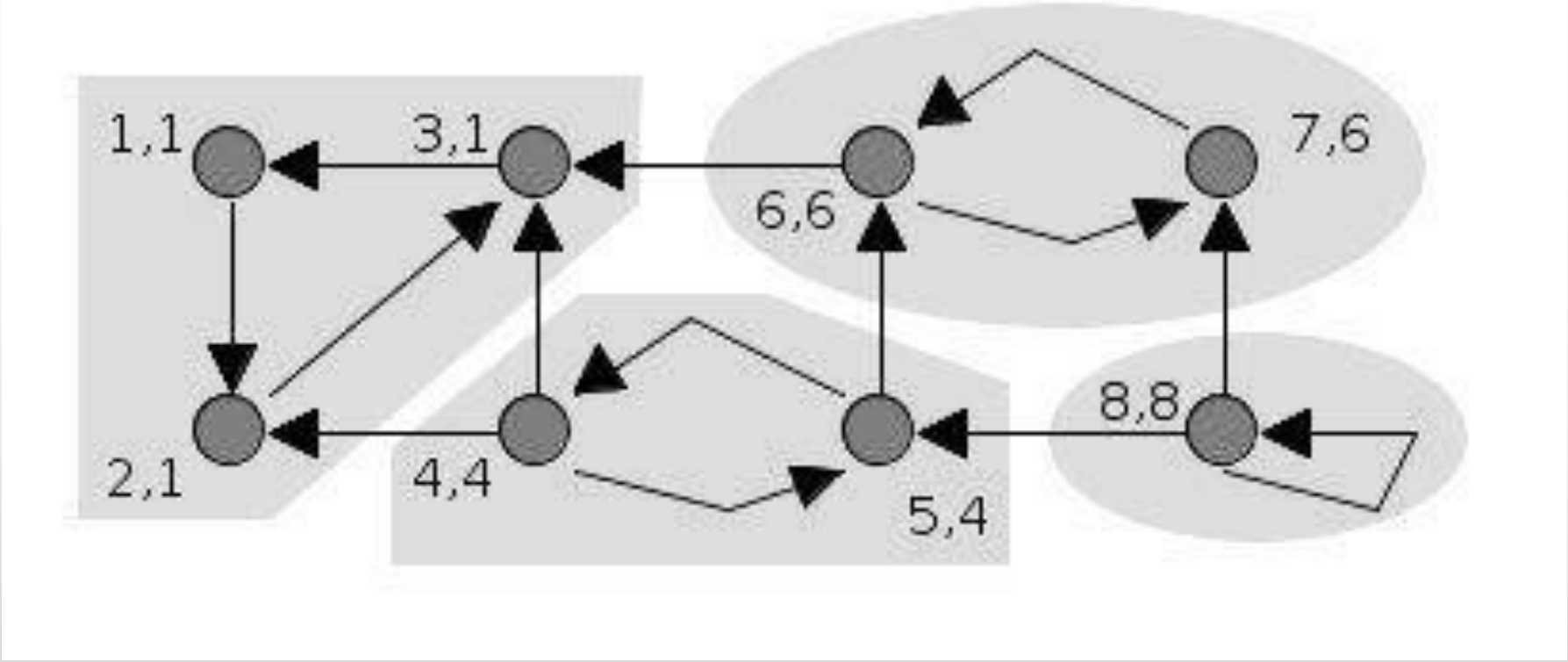
Tarjan's Strongly Connected Components



Tarjan's Strongly Connected Components



Tarjan's Strongly Connected Components



Sözde Kod

```
TARJAN_SCC (G):
        indeks = 0
        yığın = [ ]
        SCCler = [ ]

        her bir v için V içinde:
                eğer v.indeks tanımlı değilse:
                        SCC (v)

        döndür SCCler
```

Sözde Kod (2)

```
SCC (v):
        v.indeks = v.enDüşükBağ = indeks++
        yığın.ekle(v)

        her bir (v, w) için E içinde:
                eğer w.indeks tanımlı değilse:
                        SCC (w)
                        v.enDüşükBağ = min(v.enDüşükBağ, w.enDüşükBağ)
                eğer w yığında ise:
                        v.enDüşükBağ = min(v.enDüşükBağ, w.indeks)
```

Sözde Kod (3)

```
eğer v.enDüşükBağ == v.indeks:
        scc = [ ]
        döngü w ≠ v olana kadar:
                w = yığın.çıkar()
                scc.ekle(w)
        SCCler.ekle(scc)
```

SON