Kosaraju algoritmus

Innen: Algowiki

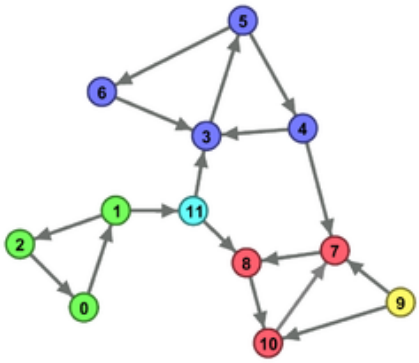
A Kosaraju, másnéven Kosaraju–Sharir algoritmust S. Rao Kosaraju és Micha Sharir egymástól függetlenül fedezték fel 1978-ban és 1981-ben. Maga az algoritmus egy hatékony (lineáris futásidejű) megoldást ad erősen összefüggő komponensek keresésére irányított gráfokban.

Tartalomjegyzék

Erősen összefüggő komponens

Az erősen összefüggő komponensek az **irányítatlan** gráfok esetén értelmezett összefüggő komponensek fogalmának **irányított** gráfokon értelmezett megfelelője. Egy irányítatlan gráf esetén azon csúcsok legbővebb halmazai alkotnak erősen összefüggő komponenseket, melyeken belül bármely két csúcs között létezik út mindkét irányban (oda-vissza).

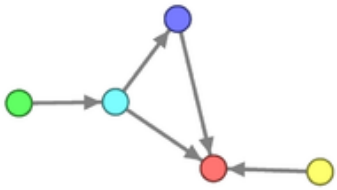
Könnyen belátható, hogy egy irányított gráfban ha minden élt megfordítunk (transzponáljuk a gráfot), az erősen összefüggő komponensek megegyeznek az eredeti és az így kapott gráfban: Az eredeti gráfban minden erősen összefüggő komponensen belül, minden A, B pontra igaz, hogy léteznek $A \rightsquigarrow B$ és $B \rightsquigarrow A$ utak. Ha a gráfot transzponáljuk, azzal minden élt megfordítunk az $A \rightsquigarrow B$ útban, így egy $B \rightsquigarrow A$ utat kapva. Hasonlóan az eredeti $B \rightsquigarrow A$ útból egy $A \rightsquigarrow B$ út keletkezik, így továbbra is teljesül az eredeti komponens minden A, B csúcsára, hogy létezik $A \rightsquigarrow B$ és $B \rightsquigarrow A$, tehát az erősen összefüggő komponensek változatlanok maradnak.



Az ábrán különböző színezéssel láthatóak az erősen összefüggő komponensek.

Erősen összefüggő komponensek gráfja (kondenzáció gráf)

Ha megállapítjuk egy irányított gráf erősen összefüggő komponenseit, képezhetjük belőle az ún. komponens gráfot. Ezt úgy kapjuk, hogy az egyes komponensek összes komponensen belüli csúcsát komponensenként egy darab csúccsal helyettesítjük, majd az így kapott csúcsokat úgy kötjük össze, hogy azzal az eredeti gráfban a komponensek közötti kapcsolatokat ábrázoljuk. Azaz a komponens gráfban akkor és csak akkor létezik $X \to Y$ él, ha az eredeti gráfban volt olyan X komponensbeli A csúcs és Y komponensbeli B csúcs, hogy köztük létezett $A \to B$ él.



A a fenti képen látható gráfból képzett komponens gráf.

Belátható, hogy a komponens gráf mindig körmentes. Hiszen ha létezik a komponens gráfban kör, akkor van két olyan $X \neq Y$ komponens, amelyek között létezik oda-vissza út. Ebből következik, hogy léteznek $A, B \in X$ és $C, D \in Y$ csúcsok úgy, hogy léteznek $A \to C$ és $D \to B$ élek. Ekkor viszont bármely X-beli csúcsból el lehet jutni bármely Y-beli csúcsba az $A \to C$ élen keresztül, valamint hasonlóan bármely Y-beli csúcsból el lehet jutni bármely X-beli csúcsba a $D \to B$ él felhasználásával. Tehát X és Y komponensek valójában egy erősen összefüggő komponenset alkotnak, azaz a komponens gráfban is csak egy csúcs reprezántálná őket.

A Kosaraju algoritmus

Felhasználjuk a fent említett két észrevételt:

- Az élek megfordításával az erősen összefüggő komponensek nem változnak
- Az erősen összefüggő komponensekből képzett komponens gráf körmentes

Az algoritmus két lépésből áll:

- Először mélységi bejárással bejárjuk a teljes gráfot és feljegyezzük az egyes csúcsok elhagyási idejét. Ez megvalósítható például úgy, ha a mélységi bejárás rekurzív változatában visszatérés előtt közvetlenül berakjuk az aktuális csúcsot egy globális verembe. Ha a bejárás során nem érintettünk minden csúcsot, akkor új bejárást indítunk egy még nem feldolgozott csúcsból.
- Ezt követően megfordítjuk a gráfban az éleket (vagy már a beolvasáskor két gráfot hozunk létre), majd elhagyási idő szerint csökkenő sorrendben bejárást indítunk a csúcsokból. (Itt tetszőleges bejárás alkalmazható) Az élek megfordítása, valamint a fenti két észrevétel garantálja, hogy a csúcsokat ilyen sorrendben bejárva, az egyes bájárások által érintett még feldolgozatlan csúcsok egy-egy erősen összefüggő komponenst alkotnak.

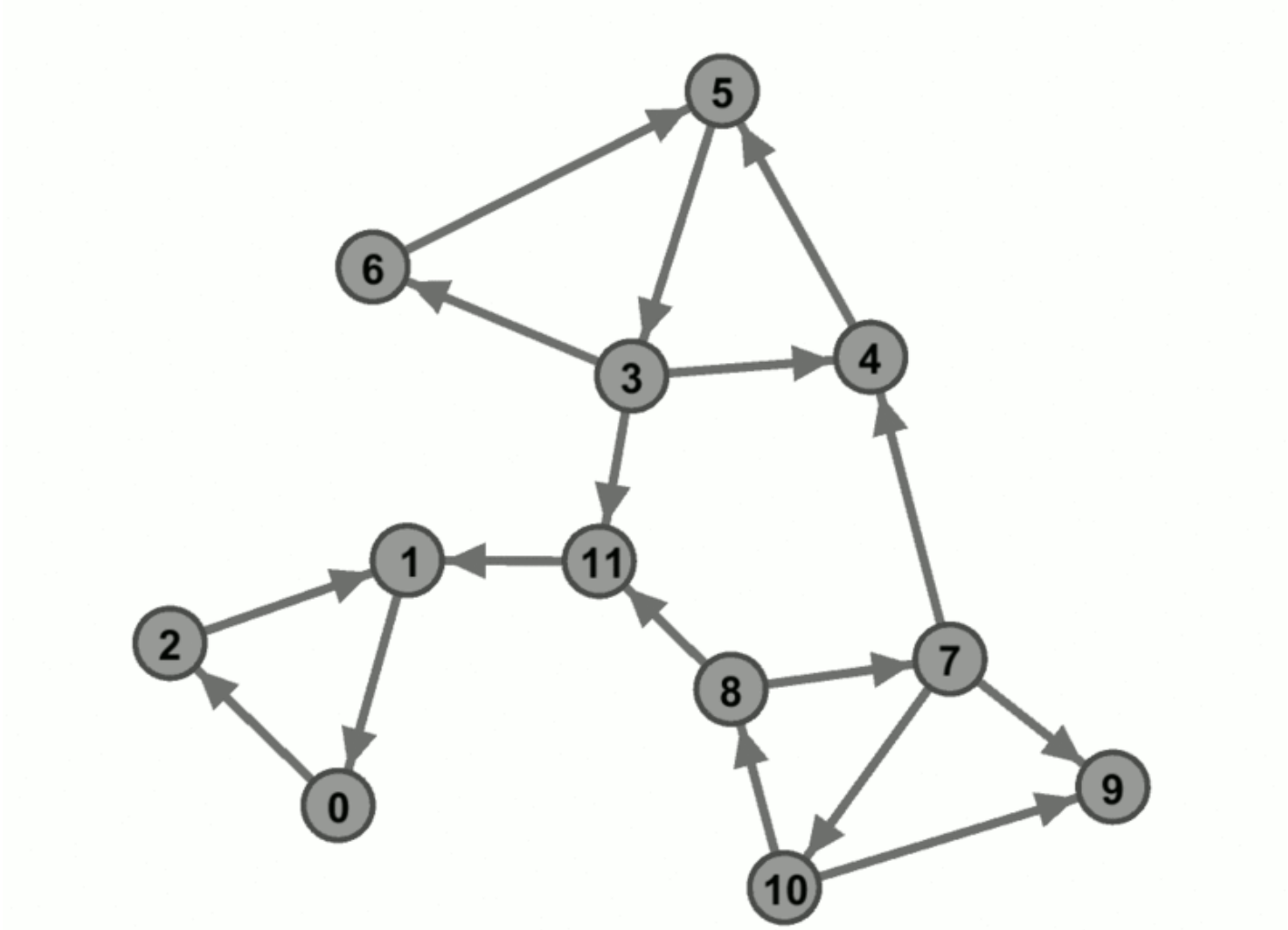
Példa

Futtassuk az algoritmust az első képen látható gráfra:

Indítsuk az első bejárást a **0** sorszámú csúcsból. Ezt követően a csúcsok egy lehetséges elhagyási idő szerint csökkenő sorrendje a következő: **9, 0, 1, 11, 3, 5, 6, 4, 7, 8, 10, 2**

Ennek a sorozatnak fontos tulajdonsága, hogy bármely A csúcsra teljesül, hogy az összes olyan B csúcs, amelyre teljesül, hogy létezik $A \rightsquigarrow B$ út (B-be el lehet jutni A-ból), A után van a sorban. (Másképpen: Minden csúcs megelőzi a sorban azokat, amelyekbe el lehet jutni belőle, azaz a csúcsok topologikus sora. Megjegyzés: nyilvánvalóan mélységi bejárás helyett egyéb topologikus sorrendet megállapító algoritmust is használhatunk)

Készítsük el a gráf transzponáltját, majd kezdjük el feldolgozni a csúcsokat a fenti sorrendben. A feldolgozás folyamatát az alábbi ábra szemlélteti:



A Kosaraju algoritmus második fázisának szemléltetése. A csúcsok sötétítése azt jelenti, hogy bár az éppen elsötétített csúcs a következő a sorban, mégis kihagyjuk, mert korábban már fel lett dolgozva. (Ha nem játssza le egyből, kattints rá!)

Futásidő

Az algoritmus mindkét fázisa egy-egy gráfbejárást alkalmaz, ezért a teljes futásidő $O(V + E)$, ahol V jelöli a csúcsok számát, E pedig az élekét.

Implementáció [becsuk]

Forrás: cp-algorithms (https://cp-algorithms.com/graph/strongly-connected-components.html#toc-tgt-2)

```
vector < vector<int> > g, gr; // g az eredeti gráf, gr a transzponáltja
vector<bool> used; // bejártuk-e már az adott csúcsot
vector<int> order, component; // order: az első fázis eredményeként kapott sorrend (itt növekvő sorrendben)
                              // component: ide gyűjtjük az aktuálisan bejárt komponens csúcsait (2. fázisban)

void dfs1 (int v) { // első fázis, topologikus sor felállítása
    used[v] = true;
    for (size_t i=0; i<g[v].size(); ++i)
        if (!used[ g[v][i] ])
            dfs1 (g[v][i]);
    order.push_back (v);
}

void dfs2 (int v) { // második fázis, 'v' komponensének bejárása
    used[v] = true;
    component.push_back (v);
    for (size_t i=0; i<gr[v].size(); ++i)
        if (!used[ gr[v][i] ])
            dfs2 (gr[v][i]);
}

int main() {
```

```
        int n;
        ... n beolvasása ...
        for (;;) {
            int a, b;
            ... következő él (a,b) beolvasása ...
            g[a].push_back (b);
            gr[b].push_back (a);
        }

        used.assign (n, false);
        for (int i=0; i<n; ++i)
            if (!used[i])
                dfs1 (i);
        used.assign (n, false);
        for (int i=0; i<n; ++i) {
            int v = order[n-1-i];
            if (!used[v]) {
                dfs2 (v);
                ... kiírjuk az aktuális komponenst: 'component' tartalmát ...
                component.clear();
            }
        }
    }
```

A lap eredeti címe: „https://algowiki.miraheze.org/w/index.php?title=Kosaraju_algoritmus&oldid=1255”