**Ág-korlátos backtrack – Cut and branch backtrack**

Gráfkereső algoritmusok közös sablonja:

1. adatbázis
   1. ez egy régi szóhasználat, olyan értelemben, hogy mi van a memóriában
   2. korszerűbb szóhasználat lenne az adatszerkezet, mert itt lényegében az adatszerkezetet kell meghatározni
   3. csak hagyománytiszteletből használjuk az adatbázis szót
2. inicializálás
   1. az adatszerkezetet alkotó változók kezdőértékét itt adjuk
   2. megmondjuk, hogy kezdetben mi van a memóriában
3. ciklus
   1. ez egy új szóhasználat, a régebben a vezérlő szót használtuk
   2. a jegyzetben a vezérlő szót használjuk
   3. ez egy végtelen ciklus, amelyben feltárjuk a gráfot
   4. legeneráljuk a gráfot
   5. mivel a gráf nagy, ezért általában kezdetben nem adott a gráf, hanem operátorok segítségével futás közben generálom a gráfot a cikluson belül
   6. ha mégis adott a gráf, akkor általában más jellegű gráfkereső algoritmusokat érdemes használni
4. pozitív kilépési feltétel
   1. mivel a ciklus az egy végtelen ciklus, ezért kilépési feltételekre van szükség
   2. a pozitív kilépési feltétel->megtaláltam a megoldást
   3. azt vizsgálja, hogy megtaláltuk-e a megoldást
   4. praktikusan azt nézzük meg, hogy az aktuális csúcs terminális csúcs-e
5. negatív kilépési feltétel
   1. azt vizsgálja, hogy felismertük-e, hogy nincs megoldás
   2. fel is lehet adni a megoldáskeresést, ilyen esetben nem győződök meg arról, hogy nincs megoldás, csak feladom
6. technikai művelet
   1. pl.: backtrack esetén a visszalépés
   2. mélységi keresés esetén a kiterjesztés
   3. olyan művelet, ami megváltoztatja az adatbázist, de nem operátor

És minden algoritmushoz felteszünk 3 kérdést

1. Ha van megoldás, megtalálja?
2. Ha nincs megoldás, felismeri?
3. Ha talál megoldást, az optimális?
4. Csak akkor tesszük fel, ha a 3. kérdésre igaz a válasz.
   1. Ha optimális, milyen értelemben optimális?

Def.: (Megoldás): A gráfkeresési probléma megoldása, 1 út, amely a start csúcsból valamely terminális csúcsba vezet.

Minden gráfkereső algoritmus erre épül és ezeket a kérdéseket tesszük fel.

**Adatbázis**: elavult szóhasználat, jobb lenne az adatszerkezet. Mi van a memóriában, merevlemezen? Milyen változóim vannak?

**Inicializálás**: Változóimnak mi a kezdőértéke?

**Ciklus**: A jegyzetben vezérlő van. A ciklus egy végtelen ciklus, while(true). A gráfkereső algoritmus, amit keresünk, nem feltételezik, hogy kész van a gráf. Úgy írjuk meg, hogy menet közben generálja a gráfot, a ciklusban. A gráf túl nagy, csak az érdekes részét tárom fel. Lusta megközelítés.

A ciklusban feltárom a gráfot. Mivel végtelen ciklus ezért kell a pozitív és a negatív kilépési feltétel.

**Pozitív kilépési feltétel:** Akkor teljesül, ha megtaláltam a megoldást, azaz terminális csúcsot találtam. A terminális csúcs nem a megoldás, csak a terminális csúcsba vezető út.

**Negatív kilépési feltétel:** Amikor azt mondom, hogy nem találtam meg a megoldást. Ez nem jelenti, hogy nincs megoldás, csak én nem találtam

**Technikai művelet:** Nem mindegyiknek van, a nem módosíthatóknál nincs. Arra szolgál, hogy kitudjak jönni a zsákutcából.

## Módosítható gráfkereső algoritmusok

* Visszalépéses keresés: Technikai művelet: visszalépés (backtrack).
  + Sima backtrack
  + Mélységi kvadrátos backtrack
  + Körfigyeléses backtrack
  + Ág-korlátos backtrack
* Keresőfával keresők: Technikai művelet: kiterjesztés (extension).

## Visszalépéses keresők

A módosítható megoldáskereső algoritmusok egy fajtája a visszalépéses (vagy idegen szóval: backtrack) kereső, melynek több változata is létezik. A backtrack keresők alapötlete: ne csak az aktuális csúcsot tároljuk az adatbázisban, hanem azokat a csúcsokat is, melyeken keresztül az aktuális csúcsba eljutottunk. Ez azt jelenti tulajdonképpen, hogy az adatbázisban az állapottérgráfnak most már nagyobb részét fogjuk tárolni: a startcsúcsból az aktuális csúcsba vezető utat.

A backtrack keresők nagy előnye, hogy a keresés nem kerülhet zsákutcába. Ha az aktuális csúcsból nincs továbblépés a gráfban, akkor visszalépünk az aktuális csúcs szülőjébe, és abból próbálunk ezúttal más irányba lépni. Ez a speciális lépés – amit visszalépésnek neveznek – adta a nevét ennek a fajta keresőnek.

Logikus, hogy minden egyes az adatbázisban tárolt csúcsban az állapot mellett azt is el kell tárolni, hogy a csúcsból eddig merrefelé próbáltunk továbblépni. Vagyis: minden csúcsban regisztrálni kell azokat az operátorokat, melyeket a csúcsban tárolt állapotra nem próbáltunk még alkalmazni. Abban a pillanatban, mikor egy operátort alkalmaztam az állapotra, az operátort törlöm a csúcsban tárolt regisztrációból.

## Ág-korlátos backtrack – Cut and branch backtrack

**Ág-korlátos backtrack**

* adatbázis
  + aktuális út
    - a ’start’ csúcsból vezet az aktuális csúcsba
    - le kell tárolnom a mélységi korlátot is
  + az utoljára megtalált a terminális a csúcs
* inicializálás
  + S = (k, 0, null, {o, | k-ra alkalmazható operátorok})
  + Akt = S
  + MélységiKorlát = végtelen (c#-ban: int.MaxValue, Java-ban: Integer.MAX\_VALUE)
  + UtoljáraMegtaláltTerminálisCsúcs = 0;
* ciklus
  + az aktuális csúcsra alkalmazok a csúcsban tárolt alkalmazható operátorok közül 1-t, ezt törlöm a csúcsból
  + az így létrejövő ’új’ állapotban létrehozok egy ’Új’ csúcsot
  + Új = (új, Akt.melyseg + 1, Akt, {o | az új állapotra alkalmazható operátor})
* pozitív kilépési feltétel
  + a ’start’ csúcsból kellene visszalépni, ami lehetetlen, és az utoljára megtalált terminális csúcs **nem null**
  + ebben az esetben az a megoldás adódik az utoljára megtalált terminális csúcsból visszafelé haladva a szülő referenciákon keresztül
* negatív kilépési feltétel
  + ’start’ csúcsból kellene visszalépni, ami lehetetlen, és az utoljára megtalált terminális csúcs értéke **null**, ebben az esetben nincs megoldás
* technikai művelet
  + backtrack
    - backtrack feltétele:
      * ha az aktuális csúcs zsákutca, vagy az aktuális csúcs mélysége >= mélységi korlát
      * **vagy** az aktuális csúcs állapota, már szerepel valamely előző csúcsban
      * **vagy** az aktuális csúcs terminális csúcs
      * **ha** az aktuális csúcs terminális, akkor mélységi korlát = aktuális csúcs mélységével és UtoljáraMegtaláltTerminálisCsúcs = akt
      * illetve minden fenti esetben visszalépés, azaz
        + akt = akt.szülő

3 kérdés:

* Ha van megoldás, megtalálja?
  + igen
* Ha nincs megoldás, felismeri?
  + igen
* Ha talál megoldást, az optimális?
  + igen
* Milyen értelemben optimális?
  + a legkisebb mélységű terminális csúcsot találja meg, azaz a legrövidebb megoldást találja meg

Ez egy jó algoritmus, de viszonylag drága, mert ha találok egy terminális csúcsot, akkor abból visszalépek. Egészen addig megyek, amíg a ’start’ csúcsból kellene visszalépni, azaz szisztematikusan, de mindig csökkenő mélységi korlát mellett feltárom az egész gráfot.

Nagyon gyakran használjuk, ha optimális megoldást kell keresni. (ha nem kell optimális megoldás, akkor nem, mert drága)

**Alapötlet**: Ha találok egy terminális csúcsot, ott nem állok meg, hanem folytatom a keresést, azaz visszalépek. De visszalépés előtt beállítom a mélységi korlátot a megtalált terminális csúcs mélységére. Ez azért jó mert, ha találok még egy terminális csúcsot az még jobb megoldás lesz, és ezt addig folytatom, amíg el nem fogynak az ágak, azaz visszakell lépni a start csúcsra.

**Adatbázis**: Az aktuális út + mélységi korlát + T, ami az utoljára megtalált terminális csúcsra mutat

**Inicializálás**: Létrehozom a start csúcsot, az ismert módon. AKT = S. Mélység korlát = végtelen. T = null.

**Ciklus**: Ugyan az, mint a simánál

Csak akkor lesz vége az algoritmusnak, ha a start csúcsból szeretnék visszalépni.

**Pozitív kilépési feltétel:** Ha a start csúcsból kellene visszalépnem és T nem null, ekkor a megoldás adódik T.szülő referencián keresztül visszafelé haladva.

**Negatív kilépési feltétel**: Ha a start csúcsból kellene visszalépnem és T null, ekkor kiírom, hogy nincs megoldás.

**Technikai művelet:** Ha az aktuális csúcs terminális, akkor mélységi korlát = AKT.méylség, T = AKT és visszalépek, azaz AKT = AKT.szülő . Illetve abban az esetben is visszalépek, ha az aktuális csúcs zsákutca, vagy az aktuális csúcs mélysége nagyobb vagy egyenlő, mint a mélységi korlát, vagy az aktuális csúcs állapota szerepel valami előző csúcs állapotában, ekkor visszalépek, azaz AKT = AKT.szülő.

Három kérdés:

IGEN- IGEN-IGEN

Ha a 3. kérdésre igen a válasz akkor a 4. kérdés:

Ha optimális megoldást talált, milyen értelemben optimális? A legkisebb mélységű megoldást találja.

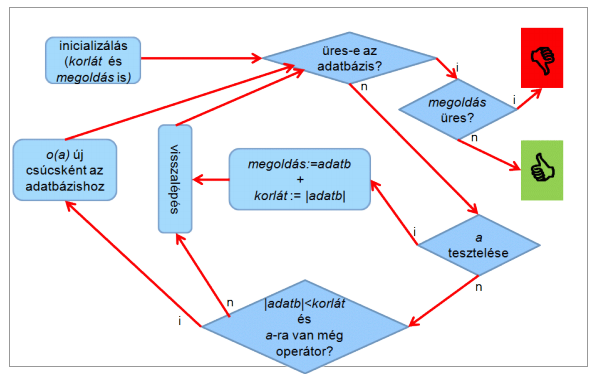
Jegyzet:

Az előző fejezetben bemutatott backtrack keresőt a következő célból próbálhatjuk meg még továbbfejleszteni: a kereső garantálja az optimális megoldás előállítását! Egy ilyen továbbfejlesztés elég logikusan adódik: a feladat megoldásainak univerzumában végezzen a kereső egy sima minimumkiválasztást.

A kereső tehát egy megoldás megtalálásakor nem fog terminálni, hanem az adatbázisról készít egy másolatot (erre fogja használni a megoldás nevű vektort), majd visszalép, és folytatja a keresést. Ebből adódik, hogy a keresés csak akkor ér véget, ha kiürül az adatbázis. Hogy ne kelljen a teljes állapottér-gráfot bejárnunk ehhez, használjuk az úthosszkorlátos backtrack keresőnek egy olyan változatát, melyben a korlát értéke dinamikusan változik. Egy megoldás megtalálásakor (és a megoldás vektorban való letárolásakor) a korlát új értéke legyen a megtalált megoldás hossza! Azaz a megtalált megoldás alatti mélységben már nem járjuk be az állapottér-gráfot.

Triviális, hogy az így kapott backtrack kereső – melyet ág és korlát algoritmusnak neveznek – az optimális megoldást állítja elő (már ha létezik az adott feladatnak megoldása).

A keresés elején a felhasznált megoldás és korlát változókat inicializálni kell. A megoldás kezdetben legyen üres vektor, a korlát pedig egy olyan nagy szám, mely az optimális megoldás hosszánál mindenképpen nagyobb. A programozók a korlát kezdeti értékeként a legnagyobb ábrázolható egész számot szokták választani. Az ág és korlát algoritmust általában körfigyeléssel is kombinálják



Az így kapott ág és korlát algoritmus tulajdonságai:

**Teljesség**:

* Ha van megoldás, akkor tetszőleges állapottér-gráfban talál megoldást.
* Ha nincs megoldás, akkor ezt tetszőleges állapottér-gráf esetén felismeri.

**Optimalitás**: garantált az optimális megoldás előállítása.