Korlátozott utak közül legjobb keresése gráfokban

SZAKDOLGOZAT

Készítette:

Csabai Balázs János

Matematika BSc

Témavezetők: Király Zoltán
Kolok Balázs Csegő

Eötvös Loránd Tudományegyetem

Természettudományi Kar

Budapest, 2025.

Tartalomjegyzék

Köszönetnyilvánítás

Szeretném köszönetemet kifejezni témavezetőmnek, Király Zoltánnak, a rengeteg visszajelzésért és tanácsért, illetve hogy figyelmembe ajánlotta ezt a témát.
Továbbá köszönöm Kolok Balázs Csegőnek a konzultációs alkalmakat, amelyeken hasznos tanácsokkal látott el.

1. fejezet

Bevezetés

A szakdolgozatomban bizonyos feltételeknek eleget tevő utak közül keresünk adott mérték szerint legjobbat. Röviden bemutatjuk az általános feladattal kapcsolatos eredményeket. Később speciális eseteket vizsgálunk meg, ahol a feladatunk az lesz, hogy hogyan kereshetünk két csúcs között a feltételeknek eleget tevő utak, illetve ezek közül legrövidebbet vagy leghosszabbat. A dolgozatban bemutatott módszerek gyakran a klasszikus útkereső algoritmusokra, például a Dijkstra- vagy a Bellman–Ford-algoritmusra [3][2] épülnek, amelyeket az adott feltételeknek megfelelően módosítunk. Az általunk elemzett utak esetében különösen fontos szerepet kap az út monotonitása, amely megkötést jelent az egyes élek bejárására. Az alábbi fejezetekben ezen problémakörrel foglalkozunk, és mutatjuk be azokat az algoritmusokat, amelyekkel megfelelő utakat kereshetünk egy gráfban.

1.1. Definíciók és alapfogalmak

1. Definíció. *Egy irányított gráf aciklikus, ha nincsen benne irányított kör.*

2. Definíció. *Egy c élsúlyozás nemnegatív, ha minden $e \in E$-re $c(e) \geq 0$.*

3. Definíció. *A G irányított gráf egy c élsúlyozása konzervatív, ha G minden irányított C körére $c(C) := \sum_{e \in C} c(e) \geq 0$.*

4. Definíció. *Egy $P = (v_0, e_1, v_1, ..., e_n, v_n)$ irányított út szigorúan monoton csökkenő egy c élsúlyozásra, ha $c(e_{k-1}) > c(e_k)$ minden $k \in \{2,3...n\}$-re.*

Ehhez hasonlóan definiálhatók a szigorúan monoton növekvő, a monoton csökkenő illetve a monoton növekvő út fogalmak.

1.2. Dijkstra-algoritmus

A Dijkstra-algoritmus az egyik, ha nem a legismertebb és legtöbbet használt gráfalgoritmus. Dijkstra eredeti cikkének [3] a Google Scholar szerint több, mint 37500 darab

idézése van, amely szám jóval nagyobb lehetne, de gyakran előfordul, hogy nem hivatkozzák le, amikor az algoritmust felhasználják. Mivel a szakdolgozatomban többször felhasználom, építek rá, ezért itt röviden bemutatom és leírom a működését.

Az algoritmus meghatározza a legrövidebb utat egy adott kezdőpontból a gráf többi csúcsához egy nemnegatív élsúlyozott, irányított vagy irányítatlan gráfban.
A megoldás azon az elven alapul, hogy egy adott s csúcsból kiinduló legrövidebb utak fát vagy irányított esetben fenyőt alkotnak a gráfban. Azaz, ha P egy s-ből kiinduló legrövidebb út, akkor az $s \in P'$, $P' \subseteq P$ utak is legrövidebb utak.
Az algoritmus fokozatosan építi fel a kezdőpontból elérhető csúcsokhoz vezető legrövidebb utakat, a távolságok növekvő sorrendjében.

Az algoritmus futása közben a gráf csúcsait három halmazba osztja:

- A: azon csúcsok halmaza, amelyekre már ismerjük a kezdőpontból vezető legrövidebb út hosszát
- B: azon csúcsok halmaza, amelyeket már elérünk egy A-beli csúcsból, de még nem biztos, hogy a legrövidebb úton
- C: többi csúcs, amelyeket még nem értünk el

Az s kezdőcsúcsot először a B halmazba helyezzük.
Ezután minden lépésben a B halmazból kiválasztjuk azt a csúcsot, amelyhez a legrövidebb út vezet, majd frissítjük ezen csúcs B és C-beli szomszédaihoz tartozó távolságokat és az újonnan elért C-beli csúcsokat áthelyezzük B-be. Végül áthelyezzük ezt a csúcsot az A halmazba.
Ezt az eljárást addig ismételjük, amíg B üres nem lesz.

A Dijkstra-algoritmus lépésszáma éllistás megadás esetén bináris kupaccal $O(m \log n)$.

1.3. Útkeresés megengedett élpárok mellett

Gyakorlatban előfordul, hogy olyan feladatot kapunk, melyben két csúcs között keresünk utat úgy, hogy az útban az egymást követő élekre feltétel van megszabva. Ezt úgy is megfogalmazhatjuk, hogy néhány élpártra megtiltjuk, hogy egy útban egymás után legyenek.

Garey és Johnson 1979-ben összegyűjtöttek és bebizonyítottak több, mint 100 NP-teljes problémákat [5]. Ebben a könyvben már szerepel az *útkeresés tiltott párokra* (path with forbidden pairs) probléma, amely a következő:
Adott $G(V, E)$ irányított gráf, $C = \{(a_1, b_1), ..., (a_n, b_n)\}$ csúcspárok és $s, t \in V$ csúcsok. Létezik-e s-ből t-be irányított út úgy, hogy a C-ben lévő csúcspárok közül legfeljebb egy szerepel az útban.

A szerzők megjegyzik, hogy ez abban az esetben is NP-teljes, ha a csúcsok helyett élpárokat tiltunk. Ez pedig pont a korábban felvezetett problémának egy általánosabb esete.

Stefan Szeider ennél részletesebben vizsgálta [13] a problémát, akinek eredményét itt röviden összefoglalom.

Adott egy G irányítatlan gráf. Legyen $T(v)$ a $v \in V(G)$ csúcs éleinek élgráfja, azaz $T(v)$ csúcsai a v csúcs élei. $T(v)$ két csúcsa között pontosan akkor fut él, ha megengedjük, hogy ez a két él egymás után szerepeljen egy útban a G gráfban. Továbbá legyen $T = \{T(v) : v \in V(G)\}$, ekkor egy $P = (v_0, e_1, v_1, ..., e_n, v_n)$ útra azt mondjuk, hogy T-megengedett út, ha e_k, e_{k+1} között $T(v_k)$-ban fut él minden $k \in \{1,2, ..., n-1\}$-re.

Legyen $\mathcal{A}$ egy gráfosztály, mely zárt az izomorfizmusra, illetve $\mathcal{A}^{\text{ind}}$ legyen az $\mathcal{A}$ gráfosztály feszítő részgráfokkal való lezártja.

$\mathcal{A}$-CP probléma: Létezik-e adott G gráfban és a hozzá tartozó $T \in \mathcal{A}$-ra $s, t \in V$ csúcsok között T-megengedett út?

1. Tétel. *Az $\mathcal{A}$-CP probléma NP-teljes, ha $\mathcal{A}^{ind}$ tartalmazza a következő gráfhalmazok valamelyikét: $\{P_3, K_2 + K_2\}, \{K_3, K_2 + K_2\}, \{P_4\}, \{L_4\}$.*
Minden más esetben ez a probléma lineáris időben megoldható.

A cikkben erre a 3-SAT [5] problémát vezeti vissza, amely közismerten NP-teljes, de a bizonyítást itt nem részletezzük.

2. fejezet

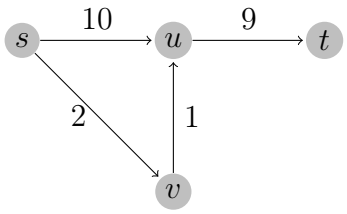
Legrövidebb monoton utak

Ebben a fejezetben két algoritmust ismertetek a legrövidebb szigorúan monoton csökkenő utak keresése problémára. Az első nemnegatív, a második konzervatív élsúlyozás mellett oldja meg a feladatot.
Ezen algoritmusok kiemelkedő szerepet töltenek be a szakdolgozatban, ugyanis több alkalommal a későbbiekben alapul fogjuk őket venni más problémák megoldásakor.

2.1. Nemnegatív élsúlyozás

Feladat: Adott s kezdőcsúcsból legrövidebb, szigorúan monoton csökkenő (ebben az alfejezetben röviden monoton) utat szeretnénk találni minden csúcsba egy nemnegatív élsúlyozott, irányított, egyszerű G = (V,E) gráfban.
Input: G irányított gráf, s kezdőpont, $c(uv)$ élsúlyok.

Kézenfekvő megoldásnak tűnik a Dijkstra-algoritmust alapul venni és csak annyit módosítani, hogy a feltételt ne sértsük. Azaz mindig a még nem átvizsgált legrövidebb megengedett útból javítunk úgy, hogy az adott legrövidebb út utolsó élének súlya nagyobb legyen, mint az utána következőé.
Ez a feltétel nélküli legrövidebb út problémára azért működik, mert s-ből egy csúcsba vezető legrövidebb útnak minden s-ből induló részútja is legrövidebb út. Azaz létezik olyan s gyökerű fenyő, melyben minden csúcsba legrövidebb úton lehet eljutni s-ből. A monoton esetben azonban ez nem teljesül. Az interneten több ilyen rossz megoldást is lehet erre a problémára találni, például a *geeksforgeeks.org* [6] avagy a *tutorialspoint.com* [7] programozási kurzusokat kínáló weboldalon.



Ebben az ellenpéldában kezdéskor átvizsgáljuk az s csúcsot, ezért az u és v csúcsokat

a látott kupacba beszúrjuk 10 és 2 értékekkel. Mintöröljük a kupacból a v csúcsot, hiszen rövidebb úton értük el, mint az u csúcsot, így most v-t vizsgáljuk át. Innen tudjuk javítani az u-ba vezető legrövidebb monoton utat. Ezután azonban u-ból nem fogjuk tudni elérni a t csúcsot, hiszen az sv, vu, ut élekből álló út nem monoton. Ez természetesen hibás, hiszen az su, ut élekből álló úton elérnénk s-ből a t csúcsot.

Azt is láthatjuk, hogy minden él benne van az s-ből kiinduló legrövidebb szigorúan monoton csökkenő utak uniójában és látszik, hogy ez nem egy fenyő.

A feladat megoldása abban rejlik, hogy nem a csúcsokba vezető legrövidebb monoton utakból javítunk, hanem az élekben végződő legrövidebb monoton utak szerint tesszük ezt.

ALGORITMUS:

$P := \{\}$; $S := \{s\}$; $R(s) := 0$; $p(s) := s$
for $u \in V$ **do**
    $h(u) := 1$
    $degree(u) :=$ u-ból kilépő élek száma
    $F(u) :=$ u-ból kilépő élek c szerint növekvő sorba rendezve
end for
for $s\underline{u} \in E$ **do**
    $K(su) := c(su)$; $p(su) := s$
    $P \leftarrow su$
end for
while $P \neq \emptyset$ **do**
    $uv := argmin\{K(e) \mid e \in P\}$
    $uv \leftarrow P$
    if $h(v) > degree(v)$ **then continue**
    end if
    for $i = h(v)..degree(v)$ **do**
        $vt := F(v)[i]$
        if $c(vt) \geq c(uv)$ **then break**
        end if
        $K(vt) := K(uv) + c(vt)$; $p(vt) := uv$; $P \leftarrow vt$
        if $t \in S$ && $R(t) > K(vt)$ **then**
            $R(t) := K(vt)$; $p(t) := vt$
        end if
        if $t \notin S$ **then**
            $R(t) := K(vt)$; $p(t) := vt$; $S \leftarrow t$
        end if
        $h(v)++$
    end for
end while

Ezen pszeudokódot Evgeny Kluev leírása és C++ programkódja [10] alapján írtam.

Jelölések:
P: kupac, melybe beszúrjuk azokat az éleket, melyeket elérünk monoton úton
S: kupac, melyben a már látott csúcsokat tároljuk
$R(v)$: s-ből v-be a legrövidebb monoton út hossza
$K(uv)$: uv élben végződő legrövidebb monoton út hossza
$p(v)$: v-be vezető legrövidebb monoton út utolsó éle
$p(uv)$: uv élben végződő legrövidebb monoton út utolsó előtti éle
$h(v)$: pointer, mely a legkisebb indexű, még P-be be nem szúrt, v-ből kiinduló él indexe

2. Tétel. *Az algoritmus lépésszáma $O(m \log n)$ éllistás megadás esetén és bináris kupacok használatával.*

Bizonyítás: Inicializálni a szükséges változókat és minden csúcsnál a kilépő éleket növekvő sorrendben rendezni összesen $O(m \log m)$ lépés. A while ciklusban minden éllel legfeljebb kétszer végzünk műveleteket, egyszer amikor beszúrjuk P-be, és egyszer amikor mintöröljük P-ből. S-ben is legfeljebb $2m$ darab kupacműveletet végzünk, mert miután egy uv élt beszúrunk P-be, megvizsgáljuk, hogy v az S-ben benne van-e és ha nem, akkor beszúrjuk, illetve az $R(v)$ értéket frissítjük, ha szükséges. A kupac műveleteket (P-ben és S-ben is) $O(\log n)$ időben tudjuk végrehajtani. Mivel $O(\log m)$ egyszerű gráfokban megegyezik $O(\log n)$-nel, így $O(m \log n)$ a futásideje az algoritmusnak.

Rendezésről és kupacműveletekről részletesen Király Zoltán jegyzetében [8] olvashatunk.

3. Tétel. *Az algoritmus s-ből minden csúcsba, melybe vezet monoton út, megad egy legrövidebb monoton utat.*

Bizonyítás: Először azt bizonyítjuk teljes indukcióval, hogy P-ben csak olyan uv él van, melyre az algoritmus talált $K(uv)$ hosszú, uv élben végződő monoton utat s-ből. Kezdetben az s-ből kilépő éleket szúrjuk be P-be. Ezen élekre teljesül, hogy monoton utak s-ből. Az algoritmus futása alatt, ha egy uv élre a $K(uv)$ értéket definiálja, akkor később már nem fog rajta változtatni. Tegyük fel, hogy egy új élt szeretnénk beszúrni P-be és eddig P-re igaz, hogy csak olyan uv él van benne, melyre teljesül, hogy az algoritmus talált $K(uv)$ hosszú, uv élben végződő monoton utat. Ez a feltétel sértetlen marad a beszúrás után, hiszen az algoritmus P-be pontosan akkor szúr be egy uv élt, ha előtte mintörölt P-ben olyan élt, mely u-ba vezet és a súlya nagyobb, mint az uv élnek, illetve $K(uv)$-t is eszerint frissíti. Így teljesül, hogy azon uv élekre, melyekre definiált a $K(uv)$ érték, az algoritmus talál $K(uv)$ hosszú monoton utat, amely uv-ben végződik.
Kell még, hogy minden élben, melyekben végződik monoton út, az algoritmus is találjon utat és ezek legrövidebb monoton utak. Minden $uv \in$ E-re, ha létezik uv-ben végződő monoton út, akkor legyen $K^*(uv)$ a legrövidebb monoton út költsége, mely az uv élben végződik, különben nem definiáljuk.
Indirekt tegyük fel, hogy létezik olyan uv él, melyre a $K(uv)$ érték nem definiált, de $K^*(uv)$ igen (nem talált monoton utat az algoritmus) vagy $K(uv) < K^*(uv)$ (nem legrövidebb monoton utat talált az algoritmus). Legyen T egy uv-ben végződő legrövidebb monoton út, ekkor T minden élére, ha T-ből elhagyjuk az adott él utáni éleket,

akkor egy legrövidebb monoton utat kapunk, amely az adott élben végződik. Továbbá tudjuk, hogy T első éle egy s-ből kilépő él, melyre a K és K^* értékek biztosan megegyeznek. Ha $K(uv)$ nem definiált, akkor legyen a bc él az első él T-ben, melyre $K(bc)$ nem definiált, és legyen ab él a bc előtti él T-ben. A bc él választása miatt $K(ab)$ érték definiált, vagyis P-be valamikor beszúrta az algoritmus ab-t, ezért mintörölte is azt. Amikor mintörölte, akkor bc élt megvizsgálta az algoritmus és mivel $c(ab) > c(bc)$, ezért be is szúrta volna P-be bc élt, hiszen még korábban nem volt beszúrva, így ellentmondásra jutunk. Ha $K(uv) > K^*(uv)$, akkor hasonlóan az előbbi jelöléshez, legyen bc él az első él T-ben, melyre $K(bc) > K^*(bc)$, és legyen ab a bc előtti él. Ekkor létezik egy xb ($\neq ab$) él, melyet az algoritmus mintörölt P-ből, és xb-ból elérte bc-t, és beszúrta P-be. Az algoritmus biztosan hamarabb mintörölte xb-t, mint ab-t, különben ab-ből értük volna el bc-t, ezért $K^*(xb) \leq K(xb) \leq K(ab)$, hiszen minden mintörölt élre igaz, hogy a nála később mintörölt éleknél kisebb egyenlő a súlya. A bc él megválasztása miatt $K(ab) = K^*(ab)$, így $K(bc) = c(bc) + K(xb) \leq c(bc) + K^*(ab) = K^*(bc)$, azaz ismét ellentmondásra jutottunk. Tehát T-ben minden monoton úton elérhető élre K definiált és értéke megegyezik K^*-gal.

Az algoritmus minden csúcsra megadja, hogy mekkora a legrövidebb monoton út hossza s-ből, hiszen $R(v)$ értéke a legkisebb $K(uv)$ értékkel egyenlő, ahol uv egy v-be belépő él.
A legrövidebb monoton utakat pedig úgy kaphatjuk meg, hogy minden u ($\neq s$) csúcsra $p(u)$ megadja, hogy melyik élből értük el $R(u)$ hosszú monoton úton u-t. Az s-ből kilépő élekre $p(su)$ s-et adja vissza, illetve minden nem s-ből kilépő uv élre $p(uv)$ megadja, hogy melyik élből értük el $K(uv)$ hosszú monoton úton uv-t. Így vissza lehet fejteni a legrövidebb monoton utat egy tetszőleges csúcsból egészen s-ig.

Megjegyzés: Ez az algoritmus nem egyszerű gráfokon is működik, azonban ekkor a futásidő $O(m \log m)$ lesz.

Megoldás élgráffal

Legyen $T(E, A)$ G-nek az irányított élgráfja, ahol a csúcsok a G élei és egy $e \in E$ csúcsból, akkor fut él $f \in E$ csúcsba, ha $c(e) > c(f)$, tehát a monotonitást nem sérti. Ekkor könnyen látszik, hogy G-ben az s-ből kiinduló legrövidebb szigorúan monoton csökkenő út probléma megoldható ebben a T gráfban, ha a c csúcssúlyozás mellett keresnünk legrövidebb utat az s-ből kilépő élekből (amik T-ben csúcsok). Ez a probléma egyszerűbb, a 3.5 alfejezetben ennek a probléma megoldásával foglalkozunk.

Fontos azonban megjegyezni, hogy egy ilyen T élgráf gyakran nagyon nagy, hiszen csúcsszáma G-hez képest négyzetes is lehet. Ezért ilyet előállítani ebben az esetben nem érdemes.

2.2. Konzervatív élsúlyozás

Feladat: Konzervatív élsúlyozott gráfban szigorúan monoton csökkenő (ebben az alfejezetben röviden monoton) legrövidebb út keresés adott s csúcsból.
Input: G irányított gráf, s kezdőpont, $c(uv)$ élsúlyok.

ALGORITMUS:

for $u \in V$ **do**
  $degree(u) :=$ u-ból kilépő élek száma
  $F(u) :=$ u-ból kilépő élek súlyuk szerint növekvő sorba rendezve
  $p(u) := 0$; $R(u) := \infty$
end for
for $uv \in E$ **do**
  $p(uv) := 0$; $K(uv) := \infty$
end for
$p(s) := 0$; $R(s) := 0$
for $sv \in E$ **do**
  $p(sv) := s$; $K(sv) := c(sv)$
end for
for $i = 1..(n-1)$ **do**
  for $u \in V$ **do**
    $h(u) := 1$
  end for
  $Q := K$ érték szerint növekvő sorrendbe rendezve az élek
  for $j = 1..m$ **do**
    $uv := Q[j]$
    if $h(v) > degree(v)$ **then continue**
    end if
    for $i = h(v)..degree(v)$ **do**
      $vt := F(v)[i]$
      if $c(vt) \geq c(uv)$ **then break**
      end if
      $K(vt) := K(uv) + c(vt)$; $p(vt) := uv$
      if $R(t) > K(vt)$ **then**
        $R(t) := K(vt)$; $p(t) := vt$
      end if
      $h(v)++$
    end for
  end for
end for

4. Tétel. *Az algoritmus lépésszáma $O(m \cdot n \log n)$ éllistás megadás esetén és bináris kupacok használatával.*

Bizonyítás: Egy cikluson belül $O(m \log n)$ lépést teszünk, hiszen lépésszám tekintetében azt csináljuk, mint az első algoritmusban. Azaz az éleket sorba rendezzük, majd minden élen legfeljebb egyszer frissítünk. Ezt $n-1$-szer ismételjük, így kapjuk az $O(m \cdot n \log n)$ futásidőt.

5. Tétel. *Ha c konzervatív élsúlyozás, akkor ez az algoritmus s csúcsból minden csúcsra kiszámítja az oda vezető legrövidebb szigorúan monoton csökkenő út hosszát.*

Bizonyítás: Teljes indukcióval belátjuk, hogy minden uv élre, melyre $K(uv)$ véges, akkor $K(uv)$ egy létező s-ből induló uv élben végződő monoton séta összsúlya, és

legfeljebb akkora, mint a legkisebb összsúlyú, legfeljebb i élből álló s-ből induló uv élben végződő monoton sétának az összsúlya. Az első ciklus előtt ez természetesen teljesül, hiszen ekkor még csak az s-ből kilépő éleknek véges a $K(su)$ értéke.
Legyen a $K^*(uv)$ a $K(uv)$ érték a $k-1$. iterációban.
Ha a k. iterációban frissítünk egy $K(uv)$ értéket egy tu élből, akkor létezik $K(tu)+c(uv)$ hosszú uv élben végződő monoton séta, hiszen az indukciós feltétel miatt létezik tu-ban végződő monoton séta, mely hossza $K(tu)$, és mivel tu-ból frissítettünk, ezért $c(tu) > c(uv)$. Illetve egy uv élre, melyre $K(uv)$ véges, a legrövidebb legfeljebb k élből áll uv-ben végződő monoton séta hossza legalább $K(uv)$. Hiszen vegyünk egy legrövidebb ilyen sétát és legyen tu az utolsó előtti éle. Ekkor az indukciós feltétel miatt $K^*(tu)$ legfeljebb akkora, mint a legfeljebb $k-1$ élből álló legrövidebb megengedett tu-ban végződő séta hossza. Így $K^*(tu) \leq K(uv)-c(uv)$ és az algoritmus egy legfeljebb $K^*(tu)$ értékű élből frissíti a k. iterációban a $K(uv)$-t.
Ha c élsúlyozás konzervatív, akkor a legrövidebb s-ből induló uv élben végződő monoton út hossza megegyezik a legrövidebb megengedett séta hosszával, amely $K(uv)$. Mivel minden élre kiszámolja az élben végződő legrövidebb megengedett út hosszát, ezért minden csúcsra is a legrövidebb megengedett út hosszát adja az algoritmus.

3. fejezet

Rokon problémák

3.1. A monoton út feladat változatai

A 2.1-es alfejezetben legrövidebb szigorúan monoton csökkenő utakat kerestünk nemnegatív élsúlyozásra. Most megvizsgáljuk a problémának a különböző változatait.

16 alternatív feladatot kapunk, ha

- *irányított* vagy *irányítatlan* gráfban
- *növekvő* vagy *csökkenő*
- *szigorúan monoton* vagy *monoton*
- *legrövidebb* vagy *leghosszabb* utakat keresünk.

Először vizsgáljuk azt a 8 esetet, amikor legrövidebb utat keressük egy adott pontból a többibe. Nem nehéz látni, hogy ezek az előző algoritmust felhasználva megoldhatók. Az irányítatlan eset könnyen visszavezethető az irányított problémára, hiszen csak meg kell irányítanunk az éleket mindkét irányba.
A monoton eset is hasonló lesz a szigorúan monoton esethez, annyi különbséggel, hogy a while ciklusban $c(vt) \geq c(uv)$ helyett $c(vt) > c(uv)$ esetben lesz break, azaz szakítja meg a ciklust az algoritmus, hiszen azonosan hosszú éleket is megengedünk egymás után.
A növekvő esetben két módosítás is szükséges az algoritmuson. Először is megfordul a while ciklusban a $c(vt) \geq c(uv)$ egyenlőtlenség (vagy monoton esetben a $c(vt) > c(uv)$), hiszen most azokat az éleket szeretnénk P-be beszúrni, melyek nagyobbak, mint az az él, melyből elértük. Másrészt pedig az elején az éleket nem növekvő, hanem csökkenő sorrendbe kell rendezni, mert ha már egy nagyobb súlyú elért élt se tudjuk bevenni P-be, akkor a nála kisebbeket biztosan nem fogjuk tudni.
Ezen három feltételt egymástól függetlenül változtathatjuk, így ez a 8 esetet vissza tudjuk vetetni a leírt algoritmusra és a futásidő változatlanul $O(m \log n)$ marad.

Megjegyzés: Ezen 8 probléma konzervatív súlyozású esetben hasonlóan visszavezethető a 2.2-es alfejezetben leírt algoritmusra, mint itt a nemnegatív súlyozású esetek.

A másik 8 eset, amikor leghosszabb utat keresünk, már nem általánosítható ilyen könnyen, mert szigorúan monoton és monoton esetben is NP-nehéz feladatot kapunk.

6. Tétel. *Az a probléma, hogy létezik-e adott s csúcsból kiinduló legalább k hosszú monoton növekvő út a gráfban, NP-teljes.*

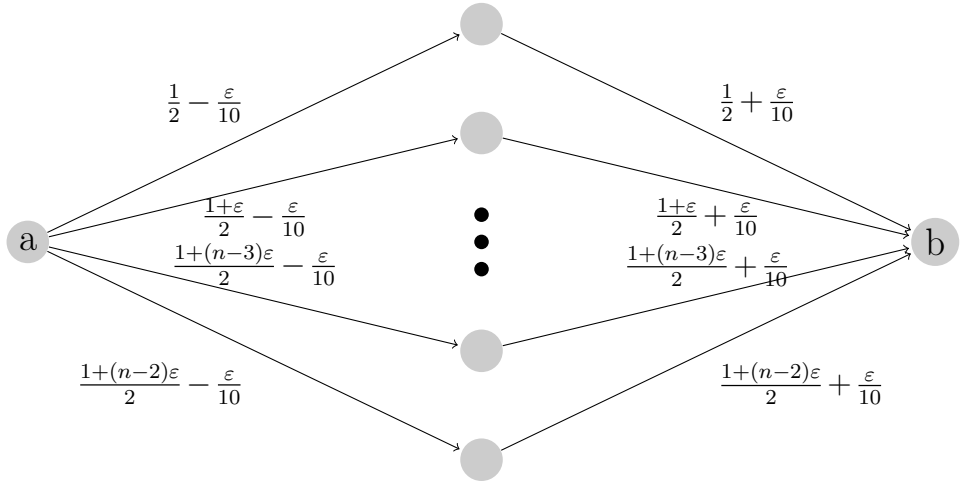
Bizonyítás: Világos, hogy NP-ben van annak eldöntése, hogy létezik-e legalább k hosszú monoton növekvő út egy gráfban, hiszen létezik tanú, az út, melyet polinom időben le tudunk ellenőrizni, hogy valóban legalább k hosszú és monoton növekvő út-e. Illetve erre polinomiálisan visszavezethető a Hamilton-út keresés feladat, mely NP-teljes [5], a következőképpen. Adott gráfról el szeretnénk dönteni, hogy van-e benne Hamilton-út. Válasszuk a gráf minden élének költségét 1-nek, és ezzel az élsúlyozással szeretnénk eldönteni minden csúcsból, hogy van-e legalább $n-1$ hosszú monoton növekvő út. Pontosan, akkor találunk ilyet, ha az eredeti gráfban volt Hamilton-út, így beláttuk a tételt.

Következmény: Adott s csúcsból kiinduló leghosszabb monoton növekvő út keresése NP-nehéz. Hiszen ha a leghosszabb monoton növekvő út keresés megoldható lenne polinom időben, akkor a legalább k hosszú monoton növekvő út probléma is megoldható polinom időben, mely problémáról beláttuk, hogy NP-teljes.

7. Tétel. *NP-teljes az a probléma, hogy létezik-e adott s csúcsból kiinduló legalább $2k$ hosszú szigorúan monoton növekvő út, ahol k egész szám.*

Bizonyítás: NP-ben van annak eldöntése, hogy létezik-e legalább $2k$ hosszú szigorúan monoton növekvő út egy adott s csúcsból, hiszen létezik tanú, az út, melyet polinom időben le tudunk ellenőrizni, hogy valóban legalább $2k$ hosszú és szigorúan monoton növekvő út-e.
Illetve polinomiálisan visszavezethető erre a legalább k élből álló út keresés adott s csúcsból feladat [5] a következőképpen.

Egy adott gráf minden éléhez hozzárendelünk n-1 darab élt, melyek súlyai 1, $1+\varepsilon$, ..., $1+(n-2)\varepsilon$, ahol $\varepsilon := \frac{1}{(n-2)*10n}$. Továbbá minden ilyen uv élt kettéosztunk egy t csúccsal (tehát t befoka és kifoka is 1) és legyen a $c(ut) := \frac{c(uv)}{2} - \frac{\varepsilon}{10}$ és $c(tv) := \frac{c(uv)}{2} + \frac{\varepsilon}{10}$.



Ezt rendeljük egy ab élhez.

Az így előállított gráfban pontosan akkor van legalább $2k$ hosszú szigorúan monoton növekvő út s-ből, ha az eredeti gráfban van legalább k hosszú út s-ből.
$\Leftarrow$: Ha létezik az eredeti gráfban P s-ből induló legalább k hosszú út, ekkor vegyük minden $i \leq k$-ra a P i. élénél a $\frac{1+(i-1)\varepsilon}{2}-\frac{\varepsilon}{10}$ és $\frac{1+(i-1)\varepsilon}{2}+\frac{\varepsilon}{10}$ súlyú éleket. Az így kapott élhalmaz egy s-ből kiinduló legalább $2k$ hosszú szigorúan monoton növekvő út lesz.
$\Rightarrow$: Legyen P x élből álló, legalább $2k$ hosszú szigorúan monoton növekvő út.
Ekkor $2k \leq c(P) \leq x(1+(n-2)\varepsilon) = x+\frac{x}{10n} \leq x+\frac{n}{10n} = x+\frac{1}{10}$.
Mivel x és k is egész szám, ezért $2k \leq x$. Így ha a P-ben lévő élek őseit vesszük az eredeti gráfban, akkor az egy legalább k hosszú s-ből kiinduló utat fog meghatározni.

Következmény: Adott s csúcsból kiinduló leghosszabb szigorúan monoton út keresése egyszerű gráfban NP-nehéz.

3.2. Leghosszabb monoton utak aciklikus gráfban

Előző alfejezetben beláttuk, hogy a leghosszabb monoton út probléma NP-nehéz, azonban aciklikus gráfokon ez a probléma P-ben van.

Hasonlóan a legrövidebb monoton út kereséshez, az élekre kiszámoljuk a beléjük vezető leghosszabb monoton út hosszát.

Az aciklikusság következménye, hogy a topologikus sorrend szerinti kisebb sorszámú csúcsból vezethet csak a nagyobb sorszámú csúcsba él. Így ha a topologikus sorrend szerinti x. csúcs előtti csúcsokból kilépő élekre tudjuk a beléjük vezető leghosszabb monoton utat, akkor az x. csúcsból kilépő élekre is ki tudjuk számolni ezen értékeket. Ezen ötlet mentén, ha a a következők szerint módosítjuk a 2.1-es alfejezetben leírt algoritmust, akkor irányított, aciklikus gráfban a leghosszabb szigorúan monoton csökkenő utakat fogja az algoritmus megtalálni, ráadásul a lépésszám sem változik, tehát a futásidő $O(m \log n)$ marad.

Először meghatározzuk a gráf csúcsainak topologikus sorrendjét. Ezután a csúcsokon a topologikus sorrend szerint haladunk végig, és minden csúcsban a már elért, belépő

élekből frissítjük a kilépő éleket. Ehhez minden $u \in V$ csúcshoz egy $P(u)$ max-kupacot használunk, melybe az elért, u csúcsba belépő éleket szúrjuk be.
Egy iterációban, egy adott v csúcsra a $P(v)$ max-kupacból egy maxtörölt uv élből próbáljuk elérni a v-ből kilépő éleket (monotonitás feltételét kielégítve) és beszúrni őket a megfelelő max-kupacba. Ezt a while ciklus addig csinálja míg $P(v)$ üres nem lesz. Így minden élt a leghosszabb útból frissítünk.

A 4 darab leghosszabb út keresése eset (szigorúan monoton vagy csak monoton és növekvő vagy csökkenő) pedig ugyanúgy visszavezethető az irányított, aciklikus gráfban leghosszabb szigorúan monoton csökkenő út keresése esetre, mint a legrövidebb esetekben.

3.3. Pontosan k élű monoton utak, color coding

Ez az alfejezet Király Zoltán [8][9], Dániel Marx [11] és Huacheng Yu [12] jegyzetei alapján készült.

Feladat: Adott egyszerű, irányított $G = (V, E)$ gráfban keressünk s csúcsból kiinduló pontosan k élből álló szigorúan monoton növekvő utat minden csúcsba $w(uv)$ élsúlyok mellett.

A k élből álló monoton vagy szigorúan monoton út keresése NP-nehéz. Azonban FPT (fixed parameter tractable) algoritmus létezik a problémára.
Itt egy randomizált algoritmust ismertetünk. A módszer neve *color coding*.

A gráf minden csúcsát véletlenszerűen és függetlenül kiszínezzük az $\{1,2,...,k\}$ színek egyikével. Eszerint minden uv élt kiszínezünk a v csúcs színére. Legyen $c(uv)$ az uv él színe. Ekkor egy utat szín-teljes megengedettnek nevezünk, ha az egy monoton pontosan k élből álló út, és az út páronként különböző színű élekből áll, azaz minden szín pontosan egyszer szerepel az útban. Szín-teljes megengedett utat könnyen kereshetünk dinamikus programozás segítségével.

Legyen egy $H \subseteq \{1,2,...,k\}$ színhalmazra és $uv \in E$ élre $f(H, uv)$ pontosan akkor *Igaz*, ha létezik olyan s-ből kiinduló út, melynek utolsó éle uv és H színei pontosan egyszer szerepelnek benne. Ekkor $f(c(sv), sv) = Igaz$. És így általánosan felírva

$$f(H, uv) = \begin{cases} \bigvee_{tu \in E : w(tu) < w(uv)} f(H \setminus \{c(uv)\}, tu) & \text{ha } c(uv) \in H \text{ és } v \neq s \\ \text{Hamis} & \text{különben} \end{cases}$$

Ha a színek részhalmazain az elemszámok szerint növekvő sorrendben haladunk, akkor minden színhalmaz és él párosra kiszámolható az f függvény értéke.

ALGORITMUS:

$P :=$ A színek részhalmazai elemszám szerint növekvő sorrendben
for $i := 2, ..., k+1$ **do**
  for $uv \in E$ **do**
    $f(P(i), uv) := Hamis$
  end for
  for $sv \in E$ **do**
    if $P(i) = c(sv)$ **then** $f(P(i), sv) := Igaz$; $p(P(i), sv) := s$
    end if
  end for
end for
for $i := k+2, ..., 2^k$ **do**
  for $uv \in E$ **do**
    $f(P(i), uv) := Hamis$
    if $c(uv) \in P(i)$ **then**
      for $tu \in E$ **do**
        if $f(P(i) \setminus c(uv), tu) = Igaz$ && $w(tu) < w(uv)$ && $v \neq s$ **then**
          $f(P(i), uv) := Igaz$; $p(P(i), uv) := tu$
        end if
      end for
    end if
  end for
end for
$H := \{1,2, ..., k\}$
for $uv \in E$ **do**
  if $f(H, uv) =$ Igaz **then** $f(v) := Igaz$; $p(v) := uv$
  end if
end for

8. Tétel. *Ha $200 \cdot e^k$ független színezés egyikére sincsen $s \rightsquigarrow t$ szín-teljes megengedett út, akkor legfeljebb e^{-200} eséllyel van a gráfban pontosan k élből álló $s \rightsquigarrow t$ monoton út.*

Bizonyítás: Ha létezik a gráfban P k élből álló $s \rightsquigarrow t$ monoton út, akkor $(k!/k^k)$ eséllyel lesz P szín-teljes megengedett út egy véletlen színezéskor, hiszen az összes k^k színezésből pontosan $k!$ darab színezésre lesz szín-teljes megengedett út P. Mivel $k! \geq (k/e)^k$, így legalább e^{-k} eséllyel olyan színezést kapunk, melyben P szín-teljes megengedett út lesz. Ebből következik, hogy annak az esélye, hogy az összes színezés rossz $\leq (1-(e^{-k}))^{200 \cdot e^k} < e^{-200}$.

9. Tétel. *Az algoritmus egyszeri futásának ideje $O(2^k m \cdot n)$.*

Bizonyítás: Összesen $2^k m$ darab különböző él és színhalmaz páros van. Minden él és színhalmaz párosra az f értéket $O(n)$ időben ki tudjuk számolni a korábban számoltak alapján, hiszen egy csúcsból legfeljebb n él lép ki. Tehát összesen a futásidő $O(2^k m \cdot n)$.

Következmény: Ha az algoritmust $200 \cdot e^k$ független színezésre futtatjuk, akkor legfeljebb e^{-200} hibával $O((2e)^k m \cdot n)$ időben el tudjuk dönteni minden t csúcsra, hogy

létezik-e pontosan k élből álló $s \rightsquigarrow t$ monoton út a gráfban.

Megjegyzés: Amennyiben az algoritmus egy t csúcsra talált szín-teljes megengedett utat és az utat meg szeretnénk kapni, akkor $O(k)$ lépésben $p()$ szerint visszakereshetjük azt.

A szigorúan monoton vagy monoton, illetve a csökkenő és növekvő eseteket könnyen megoldhatjuk ugyanígy, hiszen csak a relációs jeleket kell módosítani. Az irányítatlan esetet pedig természetesen visszavezethetjük az irányított esetre, ha megirányítjuk az éleket.

3.4. Minimális költségű szigorúan monoton csökkenő utak

Feladat: Egy adott egy nemnegatív élsúlyozott, irányított G = (V,E) gráfban, s kezdőcsúcsból minimális, szigorúan monoton csökkenő (ebben az alfejezetben röviden monoton) utat szeretnénk találni minden csúcsba, ahol egy út „hossza" az út utolsó élének költsége.
Input: G irányított gráf, s kezdőpont, $c(uv)$ élsúlyok.

Ez a feladat a 2.1-es alfejezetben szereplő algoritmus módosításával megoldható.
Ugyanis nem nehéz látni, hogy ha végződik megengedett út egy élben, akkor minden ebben az élben végződő megengedett út hossza pontosan az él súlya lesz. Tehát, ha minden élről el tudjuk dönteni, hogy végződik-e benne megengedett út, akkor minden élre tudjuk, hogy mekkora ezen megengedett utaknak a hossza. Ezáltal könnyen meghatározható, hogy milyen hosszú megengedett utak vezetnek a csúcsokba. Emiatt a feladat leegyszerűsödik annak eldöntésére, hogy az adott s csúcsból, mely élekbe vezet megengedett út.
Az első alfejezetben megadott algoritmus meghatározza, hogy mely éleket lehet s-ből szigorúan monoton csökkenő úton elérni, hiszen pontosan azon uv élben végződik szigorúan monoton csökkenő út, melyekre a $K(uv)$ érték definiált. Tehát elegendő, az utolsó for ciklusban módosítani $K(vt) := K(uv) + c(vt)$-t $K(vt) := c(vt)$-re, és így az algoritmus pontosan a minimális, szigorúan monoton csökkenő út egy adott s csúcsból feladatot fogja megoldani.

16 változata a feladatnak
Az 2.1-es alfejezetben beláttuk, hogy az a 8 eset, ahol irányított vagy irányítatlan gráfban, szigorúan monoton vagy monoton, csökkenő vagy növekvő, legrövidebb utakat keresünk, megoldhatóak az első alfejezetben szereplő algoritmus módosításával.

Az ott leírt módosítások és indoklások ezen esetekre is teljesülnek. Azaz irányítatlan esetben meg kell irányítani az éleket mindkét irányba, monoton esetben a $\geq$ helyett $>$, illetve növekvő esetben az elején az éleket csökkenő sorrendbe kell rendezni és $\geq$ helyett $\leq$.

A maximális út keresés azonban, az előző alfejezettől eltérően, megoldható polinom időben. Ez esetben is csak egy kisebb módosítás szükséges a minimális, megengedett út kereséséhez képest. Az $R(t)$ értéket, akkor frissítsük, ha $R(t) < K(vt)$, hiszen egy csúcsba vezető maximális megengedett út hossza megegyezik a legnagyobb súlyú, megengedett úton elérhető, belépő él súlyával.
Így mind a 16 különböző eset megoldható $O(m \log n)$ időben.

3.5. Súlyozott csúcsokon minimális út

Feladat: Adott egy G irányított gráf és a csúcsokon $w(v)$ egy nemnegatív költség. Keressünk adott s pontból minden csúcsba minimális költségű utat úgy, hogy egy út költsége a csúcsai költségeinek összege.

ALGORITMUS:

$P := \{s\}$; $S := \{\}$; $M := V \setminus \{s\}$; $K(s) := w(s)$; $p(s) := s$
while $P \neq \emptyset$ **do**
  $u := argmin\{K(v) \mid v \in P\}$
  $S \leftarrow u \leftarrow P$
  for $u\underline{v} \in E$ **do**
    if $v \in M$ **then**
      $K(v) := K(u) + w(v)$; $p(v) := p(u)$; $P \leftarrow v \leftarrow M$
    end if
  end for
end while

10. Tétel. *Az algoritmus lépésszáma* $O(m + n \log n)$.

Bizonyítás: Legfeljebb n darab beszúrást és n darab mintörlést végez legfeljebb n elemből álló bináris kupacban az algoritmus. Továbbá minden élen keresztül legfeljebb egyszer próbál meg javítani. Így összesen $O(m + n \log n)$ a futásidő.

11. Tétel. *Az algoritmus s-ből minden csúcsba megad egy minimális utat.*

Bizonyítás: Teljes indukció arra, hogy minden v csúcs, amely bekerül S-be, arra $K(v)$ egy v-be vezető minimális út költsége. Kezdéskor ez teljesül.
Indirekt tegyük fel, hogy v csúcsra, $K(v)$ nem egy v-be vezető minimális út költsége és ezen csúcsok közül v-re a legkisebb a K érték, illetve ha több ilyen csúcs van, akkor amelyiket hamarabb szúrtunk be S-be.
Legyen u az a csúcs, melyből elértük a v csúcsot. Tekintsük azt a pillanatot, amikor P-be beszúrtuk v-t. Ekkor u-t már beszúrtuk S-be, illetve $K(u) \leq K(v)$, tehát u-ra teljesül, hogy $K(u)$ érték egy u-ba vezető minimális út költsége.
Legyen T egy minimális költségű út v-be és t ennek az útnak az utolsó előtti csúcsa. Ekkor $w(T) = K(t) + w(v) < K(u) + w(v)$, azaz $K(t) < K(u)$. Tehát $K(t)$-t hamarabb mintöröltük, mint $K(u)$-t. De ekkor t-ből kellett volna elérnünk v-t. Ez ellentmondás.

Alkalmazás: Úgy keresünk legolcsóbb vagy legrövidebb utat, hogy nem a folyamat során, hanem a csomópontokban (csúcsokban) vannak a költségek, ez logisztikában,

gazdaságban és számos más gyakorlati példában előfordul.
Például árut szállítunk két pont kötött. Minden közigazgatási régióban egy adott költségért használhatjuk a térség úthálózatát. Keressünk legolcsóbb útvonalat.

Monoton minimális út

Monoton minimális utat keresünk az adott w csúcssúlyozás szerint.
Töröljük azon éleket, melyekre nem teljesül a monotonitás, hiszen ezeket nem használhatjuk fel. Azaz növekvő esetben azon uv éleket, ahol $w(u) > w(v)$, illetve csökkenő esetben azokat, melyekre $w(u) < w(v)$. Természetesen szigorúan monoton esetben az egyenlőséget sem engedjük meg.
Ezután már csak megengedett utakat tudunk meghatározni a maradék élek felhasználásával, így az előző algoritmust használhatjuk. A futásidő $O(m+n\log n)$ lesz.

3.6. Legfeljebb k élű minimális költségű utak

Azt mondjuk, hogy egy út k-megengedett, ha legfeljebb k élet tartalmaz.

Feladat: Adott G, $c(uv)$ nemnegatív élköltség, s csúcs és egy k szám. Keressünk s-ből minden csúcsba minimális költségű k-megengedett utat.

ALGORITMUS:

for $u \in V$ **do**
  $K(0,u) := \infty$
end for
$K(0,s) := 0$; $p(0,s) := s$
for $j = 1..k$ **do**
  for $u \in V$ **do**
    $K(j,u) := \infty$; $p(j,u) := p(j-1,u)$
  end for
  for $uv \in E$ **do**
    if $K(j,v) > K(j-1,u)+c(uv)$ **then**
      $K(j,v) := K(j-1,u)+c(uv)$; $p(j,v) := u$
    end if
  end for
end for

12. Tétel. *Az algoritmus lépésszáma $O((m+n)\cdot k)$.*

Bizonyítás: A cikluson belül minden élen keresztül próbálunk javítani. Így összesen $O((m+n)\cdot k)$ a futásidő.

13. Tétel. *Az algoritmus az i. ciklusban s-ből minden csúcsba megad egy minimális költségű i-megengedett utat.*

Bizonyítás: Teljes indukcióval belátjuk, hogy az i. ciklusban a legfeljebb i élből álló minimális költségű út hossza a v csúcsba pontosan a $K(i,v)$ érték lesz.

Az első ciklus előtt ez teljesül. Legyen P az i. ciklusban egy v csúcsba vezető legfeljebb i élből álló minimális út s-ből, illetve P utolsó éle az uv él. Ekkor az indukciós feltétel miatt a $K(i-1,u)$ egy s-ből u-ba vezető legfeljebb $i-1$ élből álló minimális út hossza, így $K(i,v) = K(i-1,u)+c(uv) \leq c(P)$.

Megjegyzés: Ez egy k cikluson keresztül futtatott Bellman-Ford-algoritmus [2], ahol figyelünk arra, hogy az i. ciklusban csak a legfeljebb i élből álló utak közül adjuk meg a legrövidebbet.

Monoton k élű minimális út

Monoton k-megengedett minimális utat keresünk.
A legrövidebb monoton út keresése konzervatív élsúlyozás esetén problémához leírt algoritmust (2.2-as alfejezet) kell csak módosítani a következőképpen.
A külső for ciklust elég $n-1$ helyett k-ig futtatni. Legyen a $K : \mathbb{N} \times E \to \mathbb{R}$ függvény (melyik él, hányadik ciklusban). Minden i. ciklusban az előző ciklusban kiszámolt $K(i-1,uv)$ érték alapján javítunk. Tehát, ha az i. ciklusban a vw élt javítani lehet az uv élből, akkor a $K(i,vw)$ értéket kell frissíteni a $K(i-1,vw)+c(uv)$ értékre.
Korábban beláttuk az algoritmusról, hogy az i. ciklusban minden csúcsra és élre kapunk egy legfeljebb olyan hosszú utat, mint a legfeljebb i élből álló legrövidebb út. Azáltal, hogy mindig a korábbi ciklusban kiszámolt értékek szerint javítunk, garantálni lehet, hogy az i. ciklusban a legfeljebb i élből álló utakat vizsgáljuk. Így tényleg monoton k-megengedett minimális utakat kapunk.
Az algoritmus lépésszáma $O(k \cdot m \log n)$, hiszen nem kell a ciklust csak k-szor futtatni.

3.7. Monotonitást sértő legrövidebb utak

A monotonitás megsértését több módon is lehet vizsgálni. A legegyszerűbb esetben azt mondjuk, hogy egy út k-szor sértő, ha az úton maximum k alkalommal fordul elő, hogy egymás követő élek nem tesznek eleget a monotonitás feltételének.
Egy másik esetben a monotonitást megsértő éleknél azt vesszük figyelembe, hogy mennyivel sértik a feltételt.

3.7.1. Legrövidebb k-szor sértő utak

Feladat: Adott s kezdőcsúcsból legrövidebb, a szigorúan monoton csökkenő feltételt legfeljebb k-szor megsértő utat szeretnénk találni minden csúcsba egy nemnegatív élsúlyozott, irányított, egyszerű gráfban.

Ismét dinamikus programozással oldjuk meg a feladatot. Legyen $K(x,uv)$ egy legrövidebb s-ből kiinduló uv élben végződő pontosan x-szer a monotonitást megsértő út költsége. Természetesen, hogyha tudjuk ezen K értéket minden $0 \leq x \leq k$ és $uv \in E$ párosra, akkor könnyedén meg tudjuk adni a legrövidebb k-szor sértő utakat.

Kezdetben minden $K(x,uv)$ értéket végtelenre állítunk. Tudjuk, hogy minden su élre

teljesül $K(0,su)=c(su)$, így ezen $(0,su)$ párokat beszúrjuk egy P kupacba a K értékük szerint. Ezután addig mintörlünk a P kupacból, amíg üres nem lesz. Az éppen mintörölt (x,uv) páros szerint próbálunk javítani a v csúcsból kilépő éleken a következőképpen. Ha $c(uv)>c(vt)$ $(vt\in E)$ és $K(x,vt)\geq K(x,uv)+c(vt)$, akkor $K(x,vt):=K(x,uv)+c(vt)$. Illetve, ha $c(uv)\leq c(vt)$, $x<k$ és $K(x+1,vt)\geq K(x,uv)+c(vt)$, akkor $K(x+1,vt):=$ $=K(x,uv)+c(vt)$. Ha olyan (y,vt) párosra javítjuk a K értéket, mely nincsen P-ben, akkor beszúrjuk azt.

Ezen algoritmus kiszámolja az összes szükséges K értéket, melyekből minden csúcsra meg tudjuk adni a legrövidebb k-szor sértő út hosszát $O(k\cdot n)$ időben.

Minden (x,uv) párost legfeljebb egyszer mintörlünk. Továbbá minden mintörléskor legfeljebb n élen próbálunk javítani. Mivel ebben az esetben előfordulhat, hogy egy páros K értékén többször is javítani tudunk, ezért a kulcs-csökkentés kupacművelet nem elkerülhető.

Ezért összességében az algoritmus lépésszáma $O(k\cdot m\cdot n\log n)$.

3.7.2. Minimális sértésű legrövidebb utak

5. Definíció. *Egy* $P=(e_1,...,e_k)$ *útra a sértés mennyisége* $s(P)=\sum_{i=1}^{k-1}(c(e_{i+1})-c(e_i))^+$,

ahol $a^+=\begin{cases}a & ha\ a\geq 0\\ 0 & különben\end{cases}$

Feladat: Adott r kezdőcsúcsból keressünk minden csúcsba olyan P utat, mely $s(P)$-re minimális és ezek közül legrövidebb egy nemnegatív élsúlyozott, irányított, egyszerű gráfban.

Legyen $T(uv)=(s(P),c(P))$, ahol P a feladatban leírt optimális út, mely uv élben végződik. Ekkor az algoritmus a következőképpen működik. Minden r-ből kilépő élre a $T(ru)$ változókat a $(0,c(ru))$ értékekre állítunk. Ezen éleket beszúrjuk a Q kupacba $T(ru)$ első értéke szerint. Ha több élnek megegyeznek ezen értékeik, akkor a második érték szerint rendezünk őket (ez minden kupacműveletet lépésszámát legfeljebb megkétszerez). Azaz ha ugyanakkora sértési értéket kaptunk, akkor a költségük szerint rendezzük őket. Ezután addig mintörlünk Q-ból, amíg üres nem lesz. Minden mintörölt uv élből megpróbáljuk javítani a v-ből kilépő éleket. Legyen

$$T(vt):=(T(uv)[1]+(c(uv)-c(vt))^+,T(uv)[2]+c(vt)),\text{ ha valamelyik teljesül:}$$

(a) $T(vt)$ nem értelmezett

(b) $T(vt)[1]>T(uv)[1]+(c(uv)-c(vt))^+$

(c) $T(vt)[1]=T(uv)[1]+(c(uv)-c(vt))^+$ és $T(vt)[2]>T(uv)[2]+c(vt)$

Továbbá ha nem volt értelmezve a $T(vt)$ érték, akkor beszúrjuk vt élt Q-ba.
Így ha minden élre kiszámoltuk az optimális értékeket, akkor a csúcsokra is meg tudjuk

ezt tenni úgy, hogy megnézzük a belépő élekre a T értékeket.

Ha az utakat is meg szeretnénk kapni, nem csak az utak sértését és költségét, akkor minden élhez egy pointert kell használnunk, mely rámutat azon élre, melyből utoljára javítottuk. Így ezen pointerek alapján visszafejthetjük az utakat.

Az algoritmus helyessége és lépésszáma

Miután az algoritmus lefutott pontosan akkor definiált $T(uv)$, ha az uv él elérhető az r csúcsból az irányított gráfban. Kezdetben ugyanis csak az r-ből kilépő élekre definiáltuk a T értékeket. Továbbá, ha egy élre T két valós számpár, akkor az uv élből közvetlenül elérhető élekre is beállítjuk a T értékét.

Amikor mintörlünk a kupacból egy uv élt, akkor $T(uv)$ értékek optimálisak lesznek, hiszen a korábban mintörölt élekből már megpróbáltunk javítani, a maradék élből pedig biztosan nem fogunk tudni javítani a kupac tulajdonsága miatt.

Az algoritmus lépésszáma $O(m \cdot n \log n)$, mert minden élt legfeljebb egyszer mintörlünk a Q kupacból. Egy mintörlés után a szomszédos éleket megvizsgálunk, hogy tudjuk-e javítani. Legfeljebb n szomszédos élt nézünk meg, illetve ha javítunk kell, de az az él már a kupacban volt, akkor kulcs-csökkentés műveletet kell alkalmaznunk, melyet $O(\log n)$ időben tudunk elvégezni.

3.7.3. Költség és sértés arányában optimális utak

Feladat: Egy nemnegatív élsúlyozott, irányított, egyszerű gráfban adott $\alpha \geq 0$ valós számra r kezdőcsúcsból keressünk minden csúcsba olyan P utat, melyre $\alpha \cdot s(P) + c(P)$ minimális.

Vegyük észre, hogy ez a probléma nagyon hasonlít az előző problémához. A különbség csak annyi, hogy itt két mennyiség aránya szerint szeretnénk minimalizáljuk az utakat, illetve a minimális utak közül nem kell a legrövidebbet megtalálni. A korábbiakhoz hasonlóan itt is minden élre kiszámoljuk a beléjük vezető minimális utak értékét. Ezen $\alpha \cdot s(P) + c(P)$ értékek lesznek a kulcsok a bináris kupacban. Ez azért fog jó megoldást adni, mert $\alpha \cdot s(P) + c(P)$ mindig nemnegatív a feladatban leírtak miatt, így a még nem átvizsgált legrövidebb útból javíthatunk tovább.

A futásidő változatlanul $O(m \cdot n \log n)$ marad, hiszen minden élből legfeljebb egyszer próbálunk meg javítani az élszomszédokon és kulcs-csökkentést is lehet, hogy használnunk kell.

4. fejezet

Implementációk

A korábbi fejezetekben pszeudokóddal leírt algoritmusokat Python nyelven implementáltam, a forráskódok a *githubon* érhetők el.

4.1. Input és output

Input: a gráf csúcsainak száma, az éllista (a csúcssúlyozás kivételével) az élköltségekkel, a kezdő csúcs, illetve ha szükséges, egy paramétert vagy a csúcssúlyozást.

Output: a csúcsokra vett optimális értékek és pointerek, melyekkel megkaphatjuk a keresett utakat. Természetesen ez is a feladattól eltérően változik.

4.2. Tesztelés

Az algoritmusat teszteléséhez a következő véletlen gráf generáló modelleket implementáltam a *networkx* programcsomag segítségével.

I. Erdős–Rényi-modell[4]: Egy n csúcsú gráf, melyben bármely két csúcs között p valószínűséggel fut él.

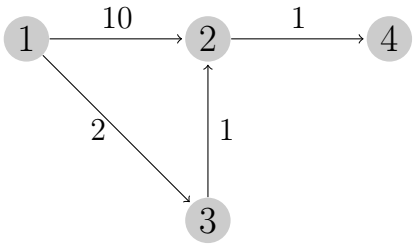
II. Barabási–Albert-modell[1]: Egy gráfot növesztünk n csúcsúvá úgy, hogy minden új csúcs m éllel kapcsolódik a már meglévő gráfhoz és nagyobb valószínűséggel kapcsolódik olyan csúcshoz, amelynek nagyobb a fokszáma. Ez skálafüggetlen fokeloszlást eredményez.

III. Watts–Strogatz-modell[14]: Egy n csúcsból álló körgráfból indulunk és minden csúcsot k szomszéddal összekötünk, majd minden élt egy p valószínűséggel áthúzunk véletlenszerűen egy másik csúcsba.

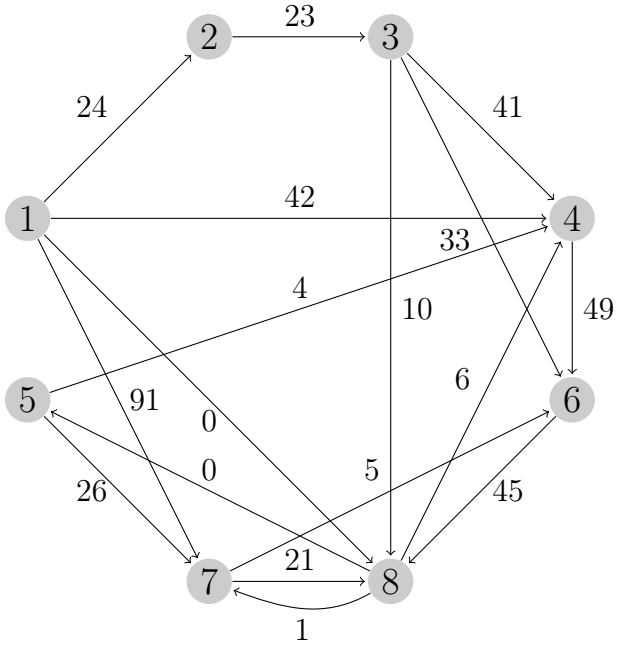
Az élsúlyokat úgy generáljuk, hogy az élekhez véletlenszerűen rendelünk az $[a, b]$ intervallumról egész számokat. Alapértelmezett paraméterek az $a = 0, b = 1000$.

4.2.1. Kisebb gráfokon

A *legrövidebb szigorúan monoton csökkenő utak keresése nemnegatív élsúlyozásra* program helyességét először egy általam gyártott gráfon, utána pedig kicsi, véletlen generált gráfokon tesztelem, majd ellenőrzöm a kimenetet.

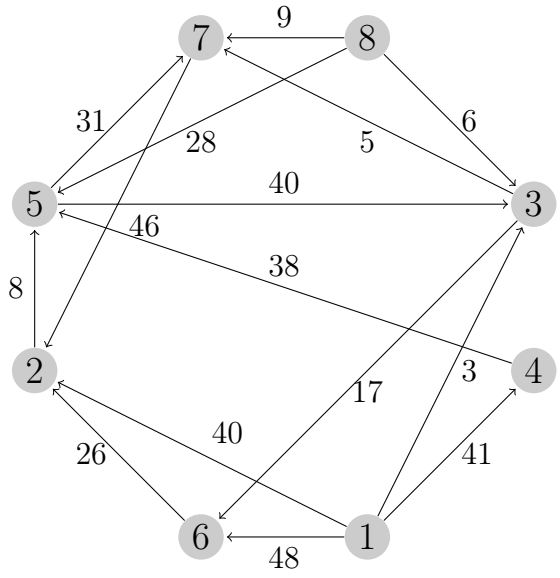


A program outputja:
Optimális úthosszak: 1: 0, 2: 3, 3: 2, 4: 11
A csúcsokhoz tartozó pointerek: 1: 1, 2: (3, 2), 3: (1, 3), 4: (2, 4)
Az élekhez tartozó pointerek: (1, 2): 1, (1, 3): 1, (3, 2): (1, 3), (2, 4): (1, 2)



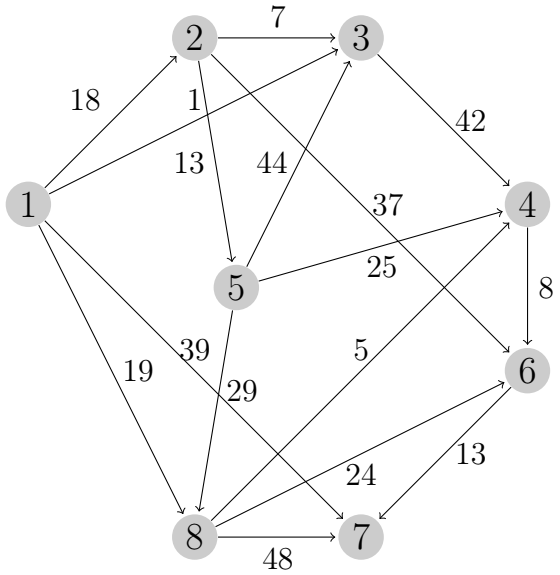
Erdős-Rényi($n = 8, p = 0.35$) gráf.

A program outputja:
Optimális úthosszak: 1: 0, 2: 24, 4: 42, 7: 58, 8: 0, 3: 47, 5: 57, 6: 96
A csúcsokhoz tartozó pointerek: 1: 1, 2: (1, 2), 4: (1, 4), 7: (8, 7), 8: (1, 8), 3: (2, 3), 5: (8, 5), 6: (7, 6)
Az élekhez tartozó pointerek: (1, 2): 1, (1, 4): 1, (1, 7): 1, (1, 8): 1, (2, 3): (1, 2), (3, 8): (2, 3), (8, 5): (3, 8), (8, 7): (3, 8), (8, 4): (3, 8), (7, 6): (1, 7), (7, 8): (1, 7)



Barabási–Albert($n = 8, m = 3$) gráf, kezdőcsúcs: 1

A program outputja:
Optimális úthosszak: 1: 0, 2: 40, 3: 3, 4: 41, 6: 48, 5: 48, 7: 110
A csúcsokhoz tartozó pointerek: 1: 1, 2: (1, 2), 3: (1, 3), 4: (1, 4), 6: (1, 6), 5: (2, 5), 7: (5, 7)
Az élekhez tartozó pointerek: (1, 2): 1, (1, 3): 1, (1, 4): 1, (1, 6): 1, (2, 5): (1, 2), (4, 5): (1, 4), (6, 2): (1, 6), (5, 7): (4, 5)



Watts–Strogatz($n = 8, k = 4, p = 0.3$) irányított gráf

A program outputja:
Optimális úthosszak: 1: 0, 2: 18, 3: 1, 7: 39, 8: 19, 5: 31, 4: 24
A csúcsokhoz tartozó pointerek: 1: 1, 2: (1, 2), 3: (1, 3), 7: (1, 7), 8: (1, 8), 5: (2, 5), 4: (8, 4)
Az élekhez tartozó pointerek: (1, 2): 1, (1, 3): 1, (1, 7): 1, (1, 8): 1, (2, 3): (1, 2), (2, 5): (1, 2), (8, 4): (1, 8)

4.2.2. Nagyobb gráfokon

A tesztelt algoritmusok:
a) Legrövidebb szigorúan monoton utak nemnegatív élsúlyozásra
b) Legrövidebb szigorúan monoton utak konzervatív élsúlyozásra
c) Legrövidebb nemnegatív csúcssúlyozásra
d) Legfeljebb k élű legrövidebb utak konzervatív élsúlyozásra
e) Pontosan k élű szigorúan monoton növekvő utak, colorcoding

Annak érdekében, hogy nagyobb (legalább 10000 csúcsból álló) véletlen gráfokon tesztelni tudjam az algoritmusok helyességét több tesztprogramot implementáltam, amelyekkel a következőket vizsgáltam.

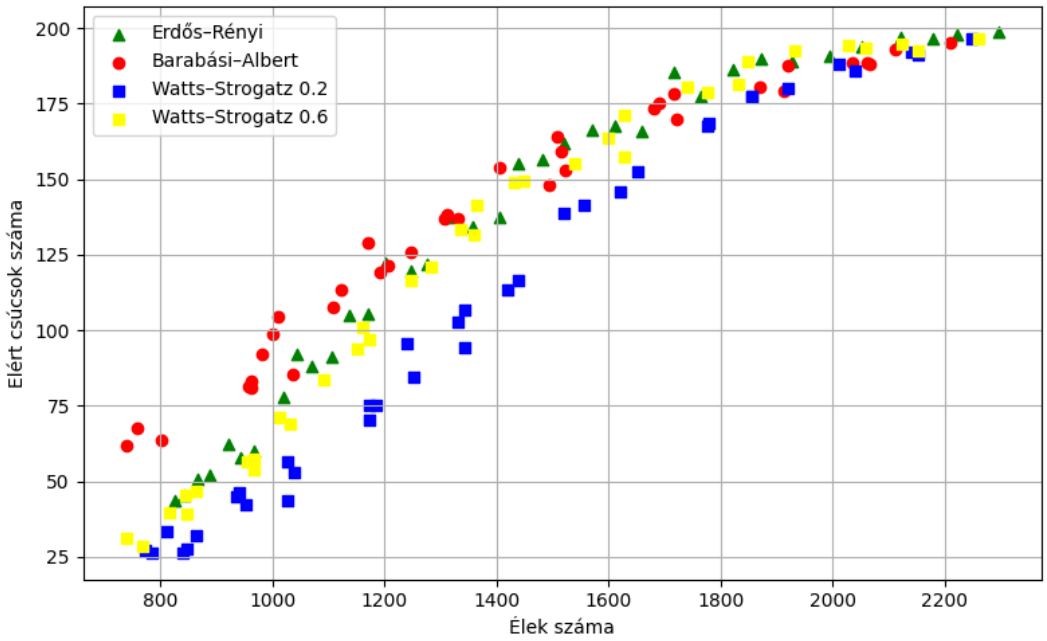
Minden algoritmusra ellenőriztem, hogy a talált utak valóban utak és a feltételeknek eleget tesznek. Illetve azt, hogy amely csúcsokat nem értünk el, azokba valóban nem vezet megengedett út.
Továbbá az a), b) és d) programokra teszteltem azt, hogy teljesül-e a potenciál-feltétel. A potenciál-feltétel az, hogy minden uv élre az u-ba vezető utak közül, amelyekhez uv megengedett folytatás, a legrövidebb $+c(uv)$ hosszúságú utat találtuk meg. Illetve az is ellenőriztem, hogy minden csúcsot a legrövidebb úton elért belépő élen érjük el.
A c) és d) programokon megvizsgáltam azt, hogy a talált legrövidebb utak részútjai, melyek a kezdő csúcsból indulnak ki is legrövidebb utak.

Mivel a tesztek során a programok hibamentesen lefutottak, és az alkalmazott feltételek szükségesek és elégségesek a legrövidebb megengedett utakra, így kijelenthető, hogy az implementált a), b), c) és d) algoritmusok helyesek. Emellett az e) feladatra írt program tesztjei is alátámasztják, hogy a kód valóban jó megoldást ad.
Tehát mind az öt algoritmus optimális, megengedett megoldást szolgáltat.

4.3. Monoton úton elérhető csúcsok

Ebben az alfejezetben azt vizsgálom, hogy közel azonos csúcs és élszámú gráfokban adott kezdőcsúcsból hány csúcs érhető el szigorúan monoton csökkenő úton. Ehhez három, korábban már említett gráf generáló modellet használok (Erdős–Rényi-modell, Barabási–Albert-modell, Watts–Strogatz-modell). Mivel ezen modellek véletlen gráfokat generálnak, ezért az élszámoknak csak a várható értéke egyezik meg. Azért, hogy ez ne okozzon problémát a méréseket átlagolom.

Előzetesen azt az eredményt várnánk egy megfelelően nagy hálózatban, hogy azon gráfokban lesz a legtöbb megengedett úton elérhető csúcs, ahol két véletlenül választott csúcs között futó utak kevés élből állnak. Ezt azért gondolhatjuk, mert egy kevesebb élből álló út nagyobb eséllyel lesz megengedett, mint egy több élből áll. Természetesen, ha több (lehetőleg különböző élekből álló) út is vezet egy csúcsból a másikba, akkor is nagyobb egy megengedett út létezésének valószínűsége.



$n = 200$, minden pont 50 mérés átlaga

Láthatjuk, hogy a Barabási-Albert-modell teljesít a legjobban, azonban minél sűrűbb a gráf, annál kiegyensúlyozottabb a modellek közötti különbség. A diagram megerősíti az előzetes sejtést, hiszen a Barabási-Albert-modell által generált gráfokban néhány csúcs fokszáma magas, ezen csomópontokon keresztül rövid utakon eljuthatunk egyik pontból a másikba.

Továbbá azt is észrevehetjük, hogy a $p = 0.2$ paraméterű Watts–Strogatz-modell a többihez képest elmarad, ezt a csúcsok magas klaszterezettsége magyarázhatja. Azonban, ahogyan növeljük a p paramétert, úgy hasonlítani kezd az Erdős-Rényi-modellhez, ezt a grafikon is alátámasztja.

Irodalomjegyzék

[1] Barabási, A. L.; Albert R. *Emergence of scaling in random networks*, Science 286, 509-512 (1999), https://doi.org/10.1126/science.286.5439.509

[2] Bellman, R. *On a routing problem*, Q. Appl. Math. 16. köt., 87–90 ISSN: 0033-569X (1958), https://doi.org/10.1090/qam/102435

[3] Dijkstra, E. W. *A note on two problems in connexion with graphs*, Numer. Math. 1. köt., 269–271 (1959), https://doi.org/10.1007/BF01386390

[4] Erdős, P.; Rényi, A. *On Random Graphs I.*, Publicationes Mathematicae Debrecen 6. köt., 290-297 (1959), https://snap.stanford.edu/class/cs224w--readings/erdos59random.pdf

[5] Garey, Michael R.; Johnson, David S. *Computers and Intractability*, W.H. Freeman and Company, New York (1979), https://perso.limos.fr/~palafour/PAPERS/PDF/Garey-Johnson79.pdf

[6] *Monotonic shortest path from source to destination in Directed Weighted Graph* (2023), https://www.geeksforgeeks.org/monotonic-shortest-path-from--source-to-destination-in-directed-weighted-graph/

[7] Jati, S. *Monotonic Shortest Path from Source to Destination In Directed Weighted Graph*, Monotonic Shortest Path in Graphs (2023.10.09), https://www.tutorialspoint.com/monotonic-shortest-path-from-source--to-destination-in-directed-weighted-graph

[8] Király, Z. *Algoritmuselmélet*, Typotex, 12-28. és 111. oldal (2014), https://zkiraly.web.elte.hu/Algoritmusok.pdf

[9] Király, Z. *Recent techniques in algorithm design* (2016), https://zkiraly.web.elte.hu/LN_FPT.pdf

[10] Kluev, E. *Find a monotonic shortest path in a graph in O(E logV)* (2014), https://stackoverflow.com/questions/22876105/find-a-monotonic--shortest-path-in-a-graph-in-oe-logv

[11] Marx, D. *Fixed Parameter Algorithms*, Open lectures for PhD students in computer science (2009), https://www.cs.bme.hu/~dmarx/papers/marx-warsaw--fpt1

[12] Yu, H. *Parameterized Algorithms*, Princeton University Course Notes (2023), https://www.cs.princeton.edu/~hy2/teaching/fall23--cos521/notes/parameterized_algorithms.pdf

[13] Szeider, S. *Finding paths in graphs avoiding forbidden transitions.*, Satadru Discrete Appl. Math. 126 köt., No. 2-3, 261-273 (2003), https://doi.org/10.1016/S0166-218X(02)00251-2

[14] Watts, D. J.; Strogatz, S. H. *Collective dynamics of small-world networks*, Nature 393, 440–442 (1998), https://doi.org/10.1038/30918

Alulírott Csabai Balázs János nyilatkozom, hogy szakdolgozatom elkészítése során az alább felsorolt feladatok elvégzésére a megadott MI alapú eszközöket alkalmaztam:

Feladat	**Felhasznált eszköz**	**Felhasználás helye**	**Megjegyzés**
Ábra készítés	GPT-4	4.2 fejezet	Gráfok kirajzolása
Programkód generálása	GPT-4	4.2, 4.3 fejezetek	Pythonban gráf generáló modellek implementálása
Nyelvhelyesség ellenőrzése	GPT-4	Teljes dolgozat	

A felsoroltakon túl más MI alapú eszközt nem használtam.

Csabai Balázs János