Alba Regia Kar

Természettudományi és Szoftvertechnológiai Intézet

PROJEKTMUNKA

Dijkstra Algoritmus

OE-AMK

2025

Hallgató(k) neve:

Puszta Tamás András, Simon Péter

Tartalomjegyzék

1. BEVEZETÉS

A gráf a matematika egyik alapvető fogalma. Csúcsokból, valamint ezeket összekötő élekből áll. Gráfok segítségével modellezhetünk útvonalhálózatokat, kommunikációs hálózatokat, vagy akár közlekedési rendszereket is.

A gráfelmélet egyik központi kérdése a legrövidebb út meghatározása. Különösen akkor, ha éleihez súlyok tartoznak. Többféle algoritmus létezik a legrövidebb út meghatározására, az egyik legelterjedtebb a Dijkstra-algoritmus [1]. Ezt a módszert Edsger W. Dijkstra dolgozta ki 1956-ban. A mohó algoritmus célja, hogy egy kiválasztott kezdőcsúcsból kiindulva meghatározza az összes többi csúcshoz vezető legkisebb összsúlyú utat. Fontos kikötés, hogy a gráf nem tartalmazhat negatív súlyú éleket. Az algoritmust gyakran alkalmazzák útvonaltervező rendszerek, hálózati protokollok.

A projektmunka célja, hogy bemutassuk a Dijkstra-algoritmus elméleti hátterét és működését, majd ennek alapján elkészítsük a programozott, működő implementációját.

2. TERVEZÉSI FOLYAMAT

A Dijkstra-algoritmus tervezéséhez, olyan eljárást kell meghatározni, amely egy nem negatív élhosszú gráfban, kezdőcsúcstól megvizsgálja a gráf minden csúcsához tartó legrövidebb élek összsúlyát.

2.1 Gráf megalkotása

A Dijkstra algoritmus megvalósításához első lépésként létre kell hoznunk egy olyan adattároló szerkezetet, amely alkalmas a gráf csúcsait és éleit hatékonyan reprezentálni. Pythonban több módon is lehetséges a gráf modellezése. A projekthez a szomszédsági listára esett a választás, ami könnyen bővíthető, gyorsan bejárható és jól illeszkedik a működéshez.

2.1.1 Gráf metódusai

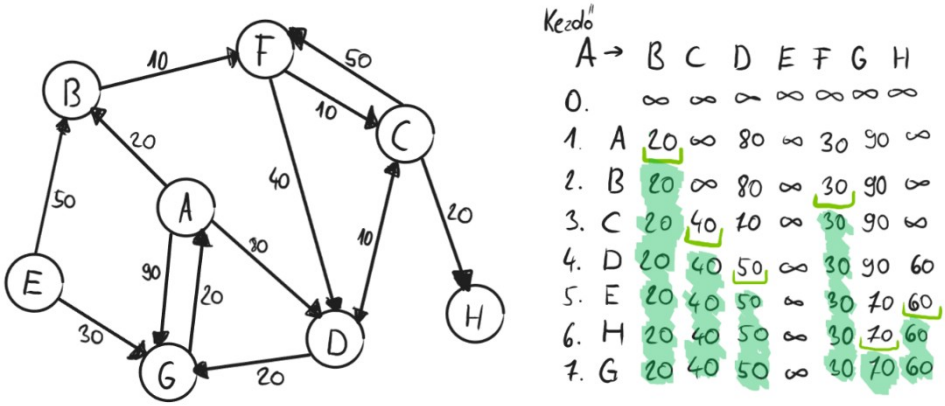
A gráf kezeléséhez több segédmetódusra is szükség van. Ezek megkönnyítik a csúcsok és élek lekérdezését, valamint a szomszédsági kapcsolatok vizsgálatát.

A gráf kezeléséhez tervezett metódusok:

- csúcsok lekérdezése, ami a gráf összes csúcsát adja vissza listaként,
- szomszédos csúcsok lekérdezése, ami visszaadja a csúcs közvetlen szomszédjait és az azokhoz tartozó súlyokat,
- élek lekérdezése, ami a gráf éleinek listáját adja vissza

2.2 Főalgoritmus

A Dijkstra-algoritmus megtervezése során olyan lépéssorozatra van szükség, amely a kezdőcsúcsból kiindulva meghatározza az összes többi csúcs felé vezető legrövidebb utak összes súlyát.



Tervezési folyamat.2. ábra Dijkstra algoritmus [Hajnal Éva: Gráf adatszerkezet II.]

2.2.1 Kezdeti állapot

Az első lépés a szükséges adatszerkezetek megvalósítása:

- távolságtömb, minden csúcshoz a legrövidebb távolság rendelése,
- feldolgozásra váró, még be nem járt csúcsok halmaza, ami tartalmazza az útkeresésre váró csúcsokat,
- látogatott csúcsok halmaza, véglegesített távolságú csúcsokat tartalmazza.

2.2.2 Mohó kiválasztás

A Dijkstra algoritmus alapja, hogy minden iterációban kiválasztjuk a csúcsot, amelynek a kezdőcsúcstól való távolsága a legkisebb a nem látogatott csúcsok közül.

2.2.3 Szomszédos távolságok frissítése

A kiválasztott csúcs szomszédjainak éleit vizsgáljuk. Ha a kezdőcsúcson át vezető út rövidebb, mint a jelenleg tárolt út, akkor frissítjük a szomszéd távolságát.

3. MEGVALÓSÍTÁSI FÁZIS

A Dijkstra algoritmus implementálása során Python nyelvet használtunk, mivel ez volt a feladat specifikációjának követelménye.

3.1 Gráf osztály megvalósítása

A gráf modellezéséhez egy osztályt hoztunk létre, amely a gráfot szomszédsági lista formájában tárolja. A konstruktor egy szótár paramétert fogad, amely a gráf szerkezetét definiálja. Ez a szomszédsági lista betöltését teszi lehetővé. Ha üres a paraméter, akkor az osztály üres szótárral inicializál.

3.2 Segédmetódusok implementációja

A gráf kezeléséhez létrehoztuk a tervezési fázisban említett három metódust. A getEdge metódus visszaadja a gráf összes élét egy lista adatszerkezetben. A getVertices visszaadja a gráf csúcsait listaként. Ezeket a főalgoritmus nem használja, csak hibakeresési célokra alkalmaztuk.

A Neighbor metódus pedig egy adott csúcs szomszédjait és a hozzájuk tartozó élsúlyokat adja vissza.

3.3 Dijkstra-algoritmus implementációja

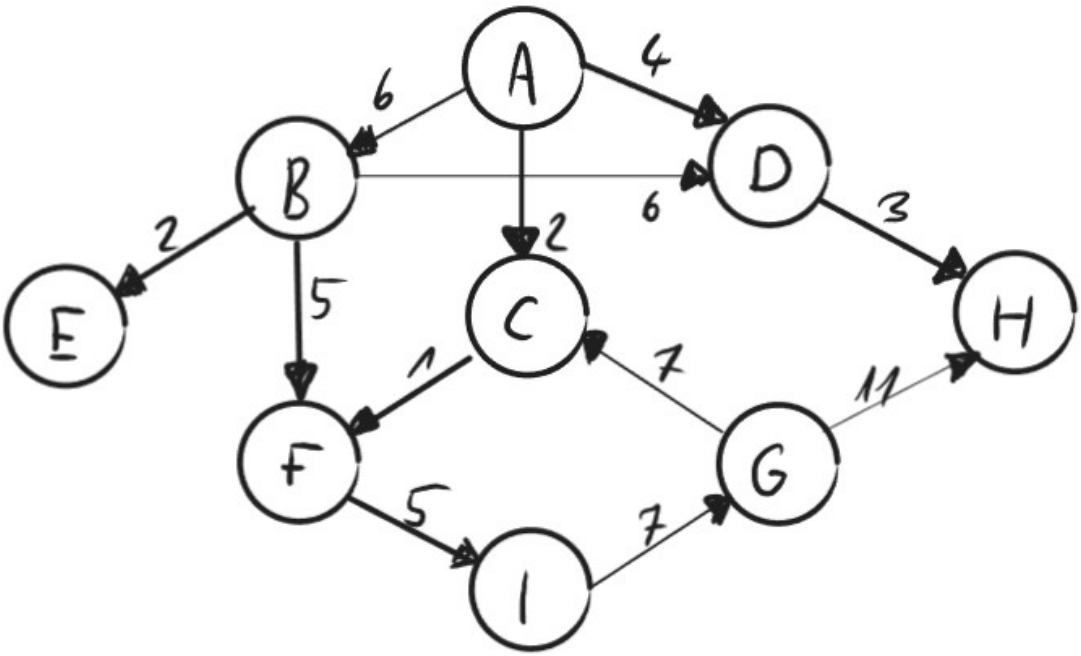
A fő algoritmus a dijkstra metódusban került megvalósításra. Az inicializálás során létrehozzuk a még nem feldolgozott csúcsok halmazát, minden csúcshoz végtelen távolságot rendelünk. A kezdőcsúcshoz nullát rendelünk. Létrehozunk egy szótárat az útvonalhoz.

Az algoritmus fő ciklusa addig fut, amíg léteznek meg nem látogatott csúcsok. Minden iterációban kiválasztjuk azt a csúcsot, amelyhez a legkisebb távolság tartozik a még nem feldolgozottak közül. Ha ennek a csúcsnak a távolsága végtelen, megszakítjuk a ciklust, mivel ekkor zsákutcába futunk. Ezután eltávolítjuk a kiválasztott csúcsot a feldolgozandók halmazából, és minden szomszédjára ellenőrizzük, hogy a jelenlegi úton

keresztül rövidebb-e az út. Ha igen, frissítjük a szomszéd távolságát és bejegyezzük az előző csúcsot.

3.4 Tesztelés

A megvalósított algoritmust egy kilenc csúcsból álló példagráffal teszteltük, amely jól illusztrálja az algoritmus működését. A gráf irányított és minden élhez pozitív súly tartozik. A tesztesetben meghatároztuk az A csúcsból a G csúcsba vezető legrövidebb utat, valamint az összes többi csúcs távolságát is:



3.4. ábra Tesztgráf [saját]

Távolságok: {'A': 0, 'B': 6, 'C': 2, 'D': 4, 'E': 8, 'F': 3, 'G': 15, 'H': 7, 'I': 8}

Előző csúcsok: {'B': 'A', 'C': 'A', 'D': 'A', 'F': 'C', 'I': 'F', 'H': 'D', 'E': 'B', 'G': 'I'}

4. ÖSSZEFOGLALÓ

A projekt során sikeresen létrehoztuk a Dijkstra útkereső algoritmust Python programnyelvben. Az algoritmus egy gráfban keresi a legrövidebb utakat egy kezdőcsúcsból indulva. Fontos kikötés, hogy a gráf élei pozitív súlyúak legyenek. Implementáltuk a gráf osztályt és a hozzá tartozó segédmetódusokat. A fő algoritmust mohó stratégia alapján készült. Az algoritmus inicializálásakor minden csúcshoz végtelen távolságot rendeltünk, kivéve a kezdőcsúcsot. Iteratívan frissítjük a távolságokat. A megoldást kilenc csúcsból álló gráffal ellenőriztük, amely igazolja a helyes működést.

A feladat nem volt könnyű. A megvalósítási fázisban gyakran megakadtunk, de hosszas hibakeresés után létrehoztuk a kész algoritmust. Idő szűkében kezdtük a megvalósítást, de összességében jól dolgoztunk közösen. A tervezési és algoritmizálási fázisban közösen dolgoztunk. A bemutató videót Péter vette fel, feldolgozta Tamás. A dokumentációt Tamás készítette.

Segítséget kértünk AI-tól (ChatGPT, Deepseek):

- kód ellenőrzésében
- dokumentációs módszertan ellenőrzésében:
    - hivatkozások
    - kódrészlet beillesztése

IRODALOMJEGYZÉK

[1] K. Géza, „Útvonaltervező algoritmusok,” *Közlekedésszervezés*, pp. 35-45, 2016.

5. ÁBRAJEGYZÉK