Elmélet

1. Rendezések
   - **Stabil rendezés:** amely az elemeknek, egyenlő kulcsok esetén, az egymáshoz viszonyított sorrendjét nem változtatja meg (pl Counting_sort ilyen).
2. Hashelés
   - **Jelölések:**
     - m: a hasító tábla mérete
     - T [0..(m − 1)]:R : a hasító tábla (R a rések típusa, ld. alább)
     - T[0],T[1],...,T[m−1] : a hasító tábla rései (slot-jai)
     - n : a hasító táblában tárolt adatok száma
     - α = n/m : a hasító tábla kitöltöttségi aránya
     - U: a kulcsok univerzuma, k,k,ki ∈ U
     - h : U → 0..(m − 1) : hasító függvény
   - Feltesszük, hogy h(k), Θ(1) időben számolhatók
   - **Direkt címzés**:
     - Feltesszük, hogy U = 0..(m − 1), ahol m ≥ n, de m nem túl nagy. A T [0..(m − 1)] hasító tábla rései ponterek, p a beszúrandó/törlendő rekordra mutat. A hasító táblát pointerekkel inicializáljuk.
   - **Nyílt címzés:**
     - Feltesszük, hogy az adatrekordok közvetlenül a résekben vannak.
     - Jelölések a nyílt címzéshez:
       - s:R→{ü,f,t,}: státuszfüggvény,
       - ahol ü : a rés üres, f : a rés foglalt, t : a résből töröltük a bejegyzést. h:U×0..(m−1)→0..(m−1): hasítópróba
       - ⟨h(k, 0), h(k, 1), . . . , h(k, m − 1)⟩ : potenciális próbasorozat
     - Feltesszük, hogy
       - kezdetben a hasító tábla minden rése üres;
       - az init(T [j]) művelet hatására a T [j] rés üres lesz;
       - tetszőleges üres T [j] résre, s(T [j]) = ü ;
       - ha r : R, akkor a T[j] := r művelet hatására a T[j] rés foglalt lesz;
       - tetszőleges foglalt T [j] résre, s(T [j]) = f ;
       - a töröl(T [j]) művelet hatására a T [j] rés törölt lesz;
       - tetszőleges törölt T [j] résre, s(T [j]) = t ;
       - a hasító táblában nincsenek duplikált kulcsok
     - Az üres és a törölt réseket együtt szabad réseknek nevezzük. A többi rés foglalt. Egyetlen hasító függvény helyett most m darab hasító függvényünk van:
       - h(·,i):U →0..(m−1) (i∈0..(m−1))
     - sikertelen keresés/sikeres beszúrás hossza: 1/1- α
     - sikeres keresés/sikertelen beszúrás: (1/ α)ln*(1/1- α)
   - **Egyszerű, egyenletes hasítás:** Minden slotra ugyanakkora a h leképezási valószínűsége
   - **Elsődleges csomósodás:** Ha van egy másik kulcs, amely szintén egy hosszú, összefüggő részt alakít ki, ezek összekapcsolódhatnak egymással.
   - **Másodlagos csomósodás:** h(ki) = (h′(k) + $c1i2$ + $c2i$) mod m, és a h′($k1$) = h′($k2$) ≫ itt a teljes próbasorozatunk ugyanaz
3. Gráfok
   - A gráf egy kulcsokból és élekből álló halmaz, ahol a csúcsok halmaza véges, az élhalmaz pedig részhalmaza az él*él halmaznak. Képletekkel leírva:

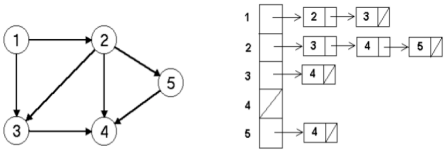
1. $G = (V, E)$
   $0 < |V| \leq +\infty$ $E \subseteq V * V$ $W: E \rightarrow \mathbb{R}$

- Szomszédsági mátrix(C):

$G = (V,E)$ $E \subseteq V \times V$
$V = 1..n$
$C[1..n,1..n]$
$C[i,j] = \begin{cases} 1 \Leftrightarrow (i,j) \in E \\ 0 \Leftrightarrow (i,j) \notin E \end{cases}$
ha számít az él költsége $w: E \rightarrow \mathbb{R}$
$C[i,j] = \begin{cases} w(i,j) \Leftrightarrow (i,j) \in E \quad (i \neq j) \\ 0 \Leftrightarrow i = j \\ +\infty \text{ különben} \end{cases}$

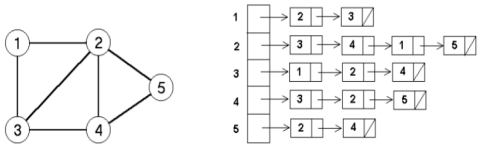
- Éllistás ábrázolás:

Irányított gráf esetén:



Helyfoglalás: $n+e$

Irányítatlan gráf esetén:



Helyfoglalás: $n+2e$

- **Szélleségi bejárás/keresés:** Meghatározzuk a start csúcsból a további csúcsokba a legkevesebb élet tartalmazó utat.
  1. d – hány élen keresztül jutunk a csúcsba
  2. π – melyik csúcsból jutunk a csúcsba
- **Mélységi bejárás/keresés:**
  1. Élek osztályozása:
     - a ⟶ b faél: ez kerül a mélységi fába, e mentén járjuk be a gráfot –fehér
     - f ⟵ g visszaél (f a g őse egy mélységi fában.) –szürke
     - b ⟶ f előreél, az előreél két leszármazottat köt össze, ahol b-ből kettő vagy több faélből álló út vezet f-be. -fekete
     - g ⟶ h keresztél, olyan két csúcs, amelyek más ágon vannak vagy másik mélységi fába mutat át -fekete
- **Mélységi bejárás alkalmazásai:**
  1. Topológikus rendezés:
     - **Def:** A csúcsok olyan sorrendje, amelyben minden él egy-egy később jövő csúcsba (szemléletesen: balról jobbra) mutat.
     - **Mélységi bejárással:** Lefuttatjuk a mélységi bejárást, közben figyeljük a visszaéleket. A csúcsokat befejezéskor egy verembe tesszük.
     - **Tétel:** Pontosan akkor $\exists$ topologikus rendezés, ha nincs irányított kör a gráfban.
       - Ha van irányított kör a gráfban jelölje: $<u_1,u_{2...}u_k,u_1>$. Ekkor tetszőleges topológikus rendezésben u_1 után jön valahol u_2, aztán a többi, végül u_1 után jön u_k és utána megint u_1 ami ellentmondás, tehát ekkor nincs topológikus rendezés
       - Ha nincs irányított kör a gráfban, akkor nyilván van olyan csúcs, aminek nincs megelőzője.
       - Ha veszünk egy megelőzővel nem rendelkező csúcsot és töröljük a gráfból, akkor a maradék gráfban nem keletkezik irányított kör, lesz megint legalább egy olyan, amelyiknek nincs megelőzője.
         Sorban a törölt csúcsok adják a topologikus rendezést.

- **Szélleségi bejárás alkalmazásai:**
  1. Minimális költségű feszítőfák:

     TÉTEL. $G = (V, E)$, u: $E \rightarrow$ R. Tegyük fel, hogy $A \subseteq$ valamely MST élhalmaznak, illetve $(S, V\backslash S)$ vágás elkerüli A-t, és $(u, v) \in E\backslash A$ könnyű él $(S, V\backslash S)$ vágásban. Ekkor (u, v) biztonságos A-ra néz- ve. (

     - Kruskál algoritmus
       - Az oszlopok mentén haladva, vettük az éleket, és ha a berajzolásával még nem keletkezett kör, akkor berajzoltuk a gráfba. Ha kihagytuk, mindig mentünk tovább. A lényeg, hogy addig menjünk, míg annyi élt rajzoltunk be, míg már többet nem tudunk berajzolni kör létrehozása nélkül.
       - Ha van két komponens, amelyek csak egy éllel vannak összekötve, akkor vegyünk egy vágást, amely kettészedi ezeket. Ez az egy él tehát könnyű él lesz, mert ennek a legalacsonyabb költségű, ha lenne ennél olcsóbb, az a rendezésben is előrébb lett volna.
       - Holvan-Unió adatszerkezet:
         i. A HOL_VAN() megállapítja, a csúcs melyik ilyen köztes fában van, ennek megfelelően az unió később a fákat köti össze. (Mindegyik fát a gyökércsúcsa azonosítja.)
         ii. Az új élt úgy vesszük a köztes fához, hogy a fák magassága ne változzon, de a komponenshez tartozás jelezve maradjon. (Az alacsonyabb fa gyökerét a nagyobb fa gyökere alá kötjük be.)
     - Prím algoritmus:
       - A Prim algoritmus minden lépésben a kék szabályt alkalmazza egy s kezdőcsúcsból kiindulva. Az algoritmus működése során egyetlen kék fát tartunk nyilván, amely folyamatosan növekszik, míg végül minimális költségű feszítőfa nem lesz.
       - Kezdetben a kék fa egyetlen csúcsból áll, a kezdőcsúcsból, majd minden lépés során, a kék fát tekintve a kék szabályban szereplő X halmaznak, megkeressük az egyik legkisebb súlyú élt (mohó stratégia), amelynek egyik vége eleme a kék fának (X-ben van), a másik vége viszont nem (nem eleme X-nek). Az említett élt hozzá vesszük a kék fához, azaz az élt kékre színezzük, és az él X-en kívüli csúcsát hozzávesszük az X-hez.
         - Piros-kék algoritmus
  2. Legkisebb költségű út:
     - Dijktsra: Adott S kezdőpontból minimális költségű út
       - Elindultunk a startcsúcsból, majd vettük a szomszédait, és leírtuk az elérési költségeit. A soron lévő csúcsot kivettük a Q sorból, és utána leírtuk az adott csúcs szülőjét, hogy onnan tudjuk elérni. Itt viszont figyeljük az elérési költségeket, ezért 3 a c elérési költsége d- ből,viszont még mindig kevesebb, mintha a-ból közvetlen érnénk el. Az épp soron következő kulcsoknál figyelni kell az költségeket, mert mindig az aktuálisan legalacsonyabbat kell figyelembe venni. Ha egy már járt csúcsot el tudnánk érni máshonnan is, de magasabb költséggel, akkor azt nem írjuk bele a táblázatba, figyelmen kívül hagyjuk.
     - Bellman Ford: Lehet negatív élköltség is, negatív összköltségű kör viszont nem.
       - Sorban haladtunk a csúcsok mentén ábécé sorrendben. Megnéztük, hogy melyik csúcsokat érhetjük el belőle, és beírtuk a táblázatba először is, hogy mekkora az elérés költsége. Majd ezt beírjuk a sorba, kivesszük a sorból azt a csúcsot, amelyiknek a gyermekeit keressük, és leírjuk az adott csúcs szülőjét. Ha sokadik elérésre jutunk el egy csúcsra, akkor az élek költségeit összeadjuk. Ha olyan csúcsba jutunk el, ahol már voltunk, akkor összehasonlítjuk a költségeiket, és az alacsonyabbat vesszük, ugyanezt vesszük a értékeinél is, azaz hogy az adott csúcsnak melyik a szülője. Az egész algoritmust addig ismételgetjük, míg üres nem lesz a Q sor.

4. Mintaillesztés
   - Brute-Force: Mindig eggyel toljuk arrébb
   - Quick Search
   - KMP
5. Tömörítés
   - Huffman, LZW