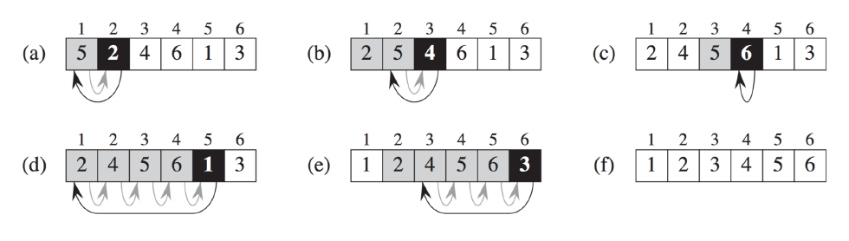
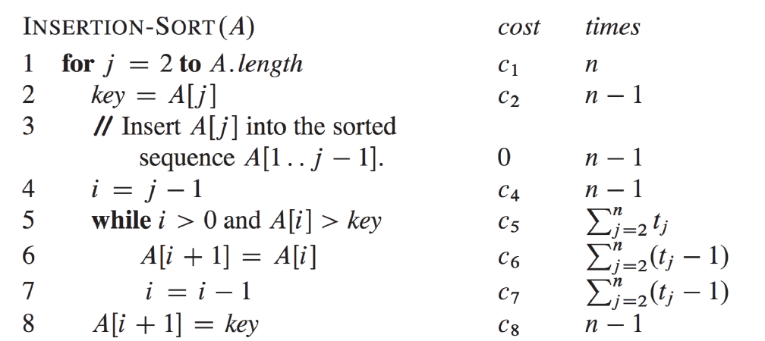
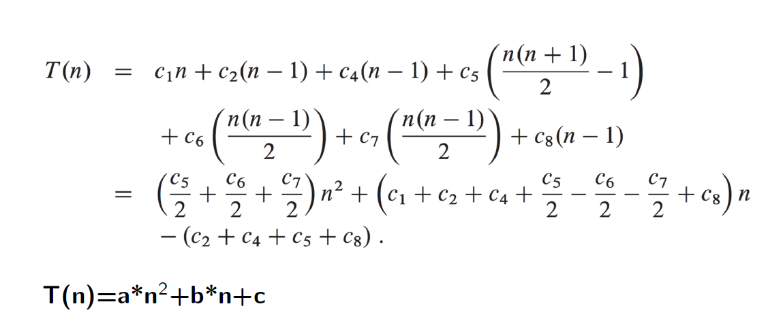
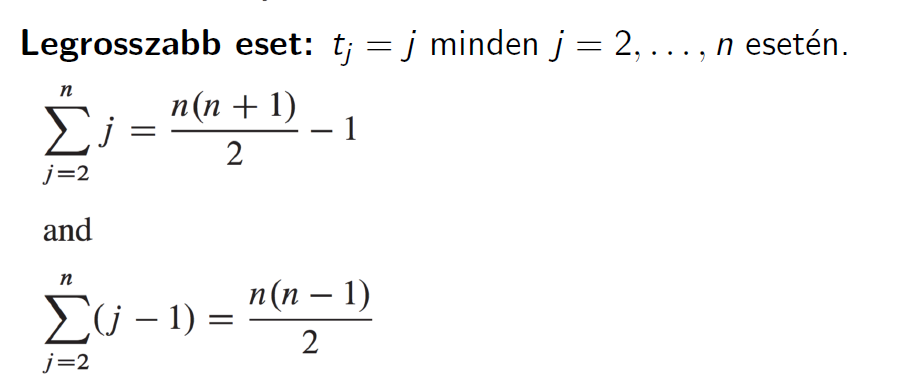
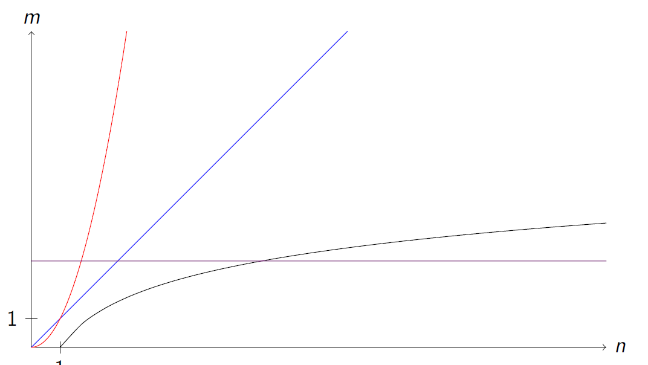
**Adatszerkezetek és Algoritmusok Vizsga Jegyzet**

**I. Előadás**

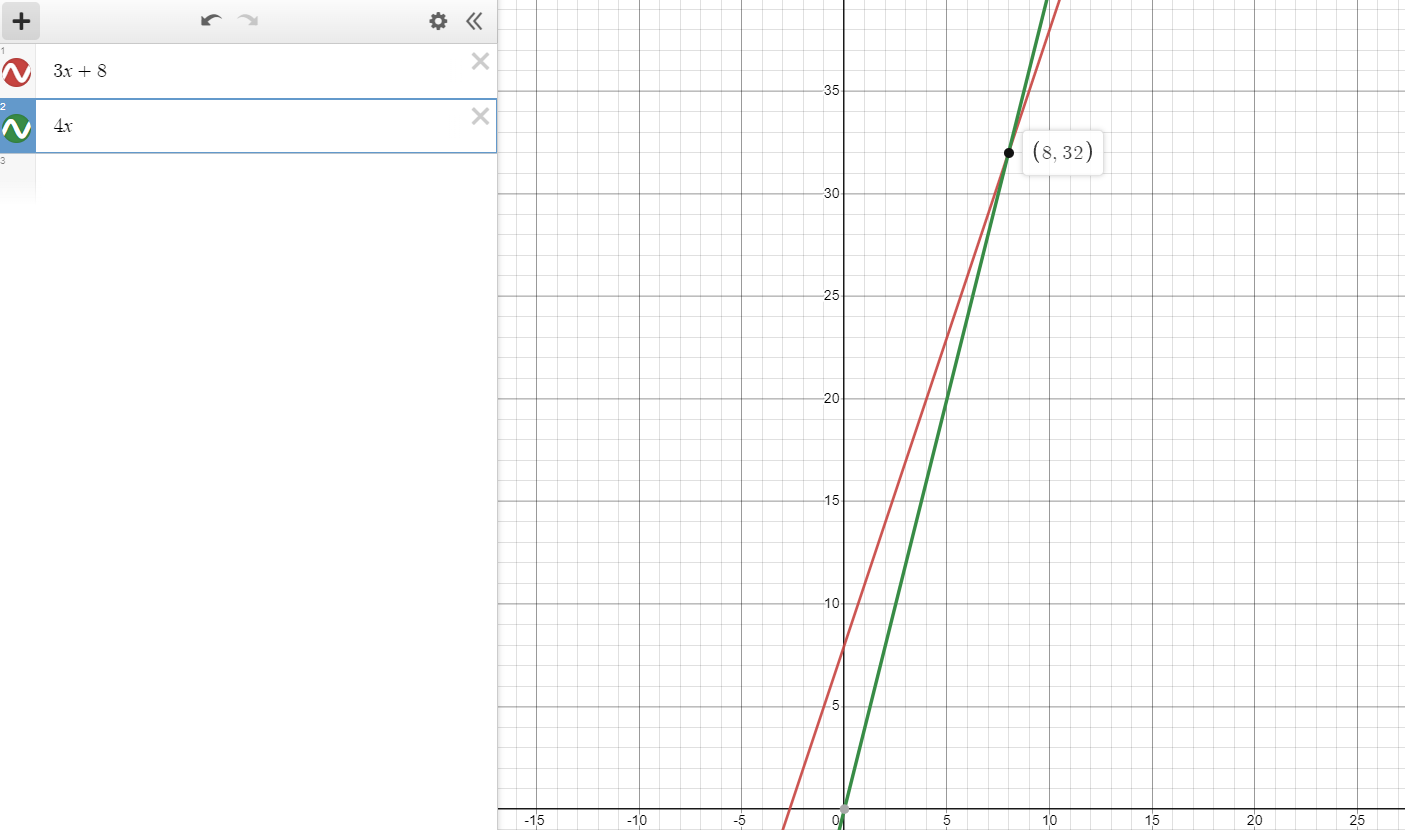
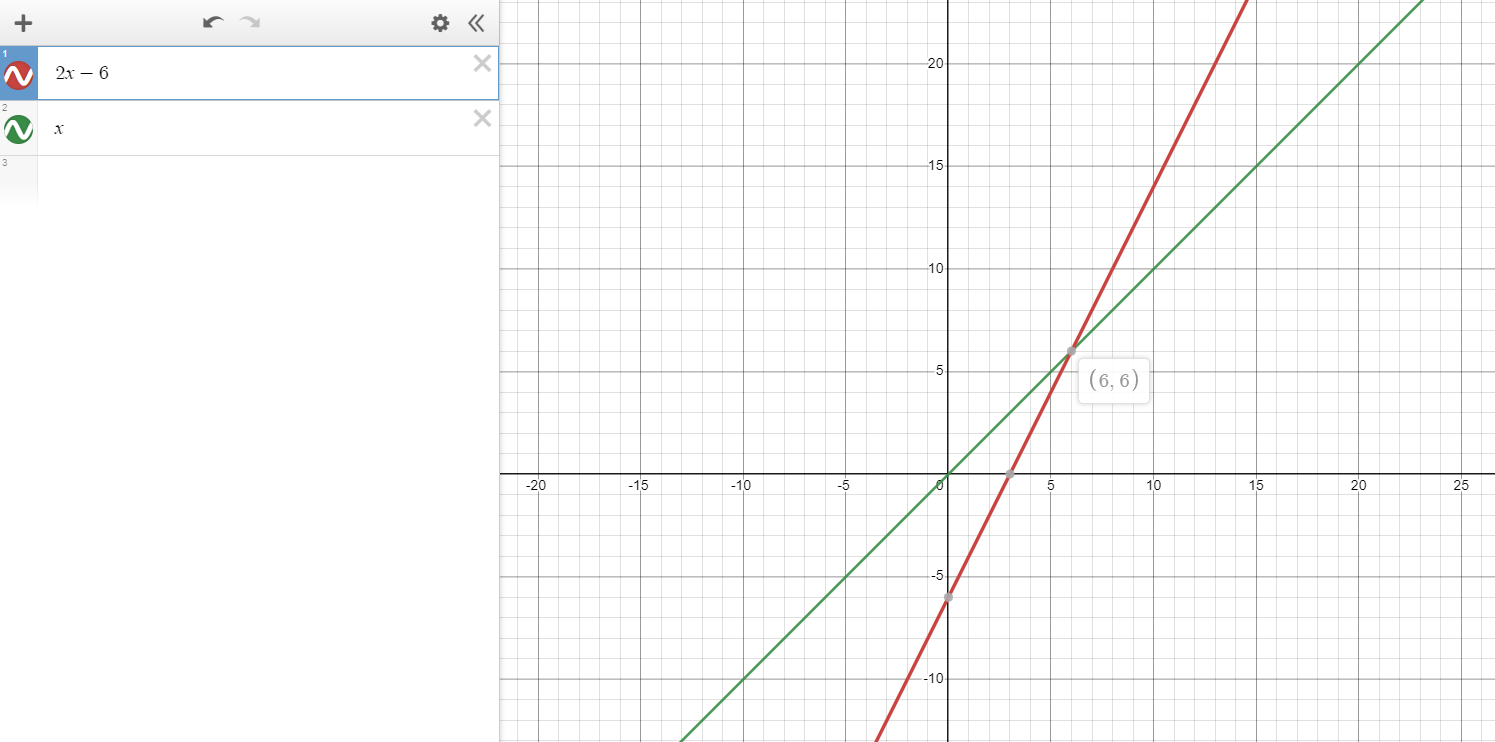
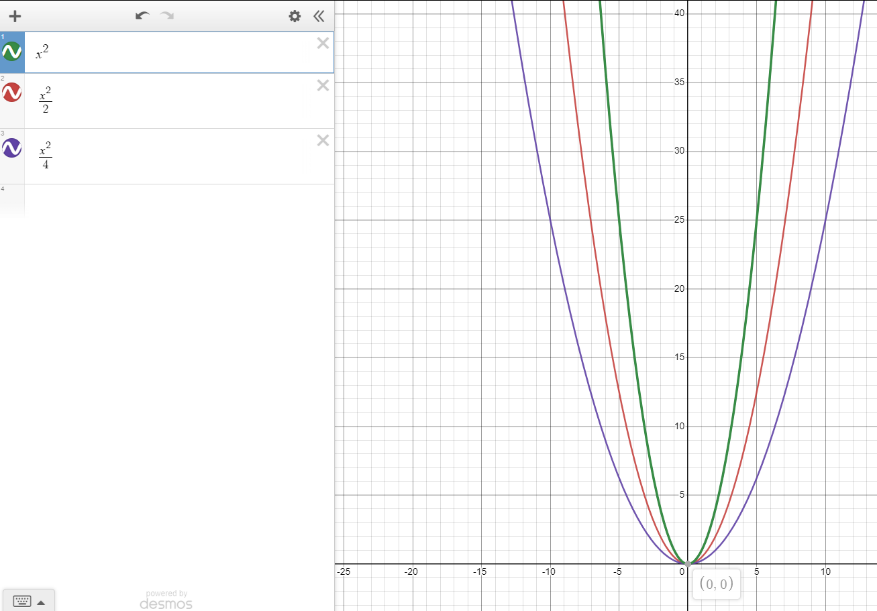
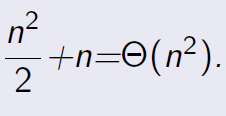
* Pszeudokód felépítés
  + Blokkokat (Scope) eltolással jelöljük, kinda mint pythonban
  + Ciklusok
    - For
    - While
    - Repeat Until
  + Elágaztatás
    - If
    - Else
  + Megjegyzés 🡪 //
* Mitől függ a futási idő?
  + Sokféle lehet, nekünk, illetve legtöbbször a bemenet mérete
  + Alapműveletek, lépések száma 🡪 Gépfüggetlen
  + Konvenció 🡪 Pszeudokód minden sorának végrehajtásához konstans idő kell 🡪C
  + Számok összeadása 1-100-ig
    - Naiv megközelítés 🡪 Futási idő lineáris
    - Gauss megoldása 🡪 Konstans 🡪1
  + Faktoriális Rekurzív, nem rekurzív
    - Lépészám🡪n
  + Lineáris Keresés futási ideje
    - Legjobb eset 🡪 Senkit nem érdekel lmao
    - Átlag 🡪 Ezzel már lehet valamit kezdeni, de nem igazán érdekel sok mindenkit
    - Legrosszabb eset🡪 Általában ez érdekel minket 🡪n
  + Bináris keresés futási ideje
    - Legrosszabb eset (ezután kb mindig ez kell)🡪log2(n)+1
    - Logaritmus alapú futási idő we likey

**II. Előadás**

Insertion Sort 🡪 Futási idő

* Első sor mindig lefut 🡪 n times
* A for-ban lévő dolgok csak n-1-szer, mert 2-től kezdődik.
* A While-ban lévő dolgok attól függ mennyiszer futnak le, hogy milyen a bemenetünk 🡪Szumma
* Legrosszabb eset
* Lényegében csak összeszorozgatjuk a sorokat, és hogy hányszor futnak le
* Futási idők sorrendben, legjobb🡪Legrosszabb
  + Konstans
  + Logaritmus
  + Lineáris
  + Polinomiális

Korlátok

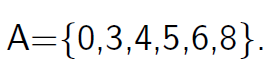
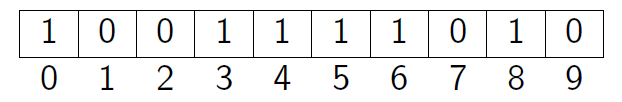
* Aszimptotikus felső korlát 🡪 Ordó 🡪 O
  + Az Ordó, egy nagyságrendet jelöl, ami egy adott pont után mindig felülről becsüli a függvényt, ha egy C konstanssal van megszorozva
    - Pl. 3n+8= O(n) , tehát C\*n egy adott pont után mindig nagyobb lesz mint 3n+8🡪 A 4\*n n0=8 pont után, mindig nagyobb lesz mint 3n+8
    - Hogy mi a C és n0 azt sokféleképpen ki lehet számolni, nekem a kedvencem a határértékkel való számolás
* Aszimptotikus alsó korlát 🡪 Ω
  + Ugyanaz, mint a felső korlát, csak alulról, van egy pont ami után mindig egy másik függvény alatt lesz az adott függvény
  + 2n-6= Ω(n) 🡪 Ez igaz, ha c=1 és n0=6
* Aszimptotikus éles korlát 🡪 Θ(theta)
  + Létezik egy pont, ami után van két függvény, az egyik adott pont után mindig felette, a másik mindig alatta lesz.
  + C1\*f mindig alulról becsüli, C2\*g mindig felülről
  + C1=0.25, C2=1
  + Tehát f=n^2/4, g=n^2
  + Ebből az alábbi jön ki, mely n0=0-tól igaz.
* Fontos az alsó, és felső korlátnál, hogy nem feltétlenül csak egy megoldás van, lehetséges több is.

**III. Előadás**

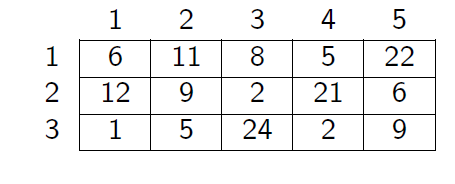
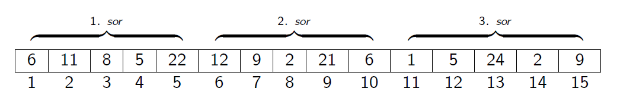
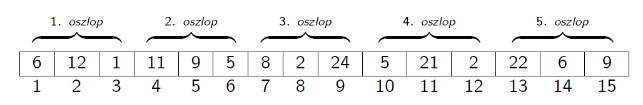
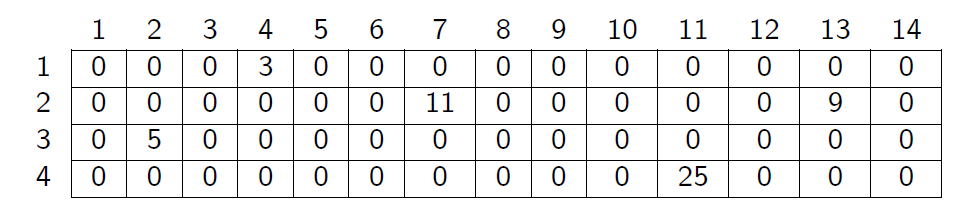
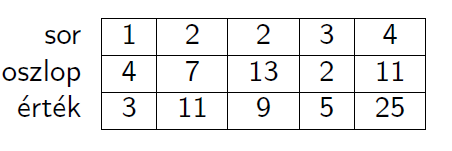
Adatszerkezetek

* Miből áll?
  + Adatelemek
  + Kapcsolatok
  + Műveletek
  + Reprezentáció
* Absztrakt adattípus🡪 Adatelemek közötti kapcsolat logikai modellje, független attól, hogyan is tároljuk az adatokat fizikailag
* Adatszerkezet ezzel szemben tartalmazza azt is, hogyan tároljuk az adott elemeket a memóriában/háttértáron.
* Adatszerkezetműveletek
  + Létrehozás
  + Módosítás
    - Elem hozzáadása
    - Elem cseréje
    - Elem törlése
  + Elem elérése
  + Esetleg 🡪 Rendezés, keresés, bejárás
* Adatszerkezetek osztályozása
  + Elemeinek típusa alapján
    - Homogén 🡪Azonos típusú elemek
    - Heterogén 🡪Különböző típusú elemek 🡪 Ilyennel nem nagyon foglalkoztunk
  + Adatszerkezet elemeinek száma
    - Dinamikus 🡪 Elemszám változhat
    - Statikus 🡪 Elemszám adott, fix
  + Homogén adatszerkezet esetén elemek kapcsolata
    - Struktúra nélküli
    - Asszociatív
    - Szekvenciális
    - Hierarchikus
    - Hálós
  + Reprezentáció
    - Folytonos 🡪 Vektorszerű
    - Szétszórt🡪Láncolt
* Néhány dolog megvalósításhoz egy adatszerkezetnek kellenek bizonyos tulajdonságok   
  Pl.: 🡪 Bináris Keresés 🡪 Homogén adatszerkezet, rendezett adatok, folytonos reprezentáció

Halmaz

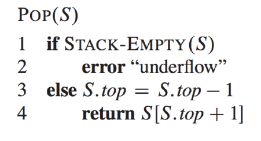
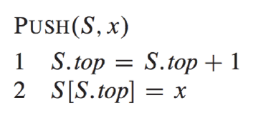
* Megkülönböztetett elemek, minden max egyszer
* Tulajdonságok
  + Homogén
  + Dinamikus
  + Struktúra nélküli
  + Folytonos reprezentáció
* Halmaz reprezentációja binárisan
  + „A” egy halmaz, melyben egész számok vannak 0-9-ig
  + Halmazműveletek ezzel a bináris reprezentációval
    - Unió🡪 A VAGY B
    - Metszet 🡪 A ÉS B
    - Különbség A/B 🡪 A ÉS NEM(B)
* Adatszerkezeti műveletek
  + Létrehozás 🡪Folytonos reprezentáció
  + Módosítás
    - Elem hozzáadása 🡪 Unió
    - Elem törlése 🡪 Különbség
    - Elem cseréje🡪 Különbség majd Unió
  + Elem Elérés 🡪**∈**
* Multihalmaz 🡪 Olyan halmaz, melyben egy adott elem többször is előfordulhat
  + Majdnem ugyanaz minden benne, mint egy halmazban, csak nem binárisan adjuk meg, hanem azt adjuk meg hányszor fordul elő.

Tömb

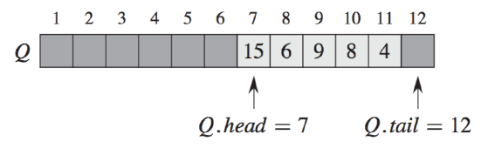
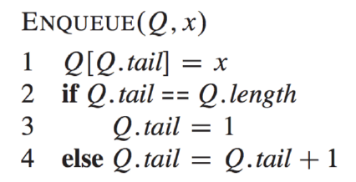
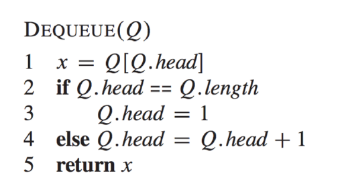
* Tulajdonságok
  + Homogén
  + Statikus
  + Asszociatív
  + Folytonos reprezentáció
* Lehet több dimenziós, ahol minden eleme a tömbnek is egy másik tömb
  + Egydimenziós tömb 🡪 Vektor
  + Kétdimenziós tömb 🡪 Mátrix
* Vektor reprezentációja 🡪 Lefoglalunk a memóriában annyi egymást követő pozíciót a memóriában egész számok tárolására.
* Mátrix reprezentáció 🡪 Megmarad a közvetlen hozzáférés
  + Sorfolytonos reprezentáció + A mátrix mérete
  + Oszlopfolytonos reprezentáció + A mátrix mérete
* Ritka mátrix
  + 3 soros reprezentáció+Mátrix mérete
  + Nincs közvetlen hozzférés
  + Csak akkor hatékony, ha az elemek száma legfeljebb harmada a max elemszámnak

**IV. Előadás**

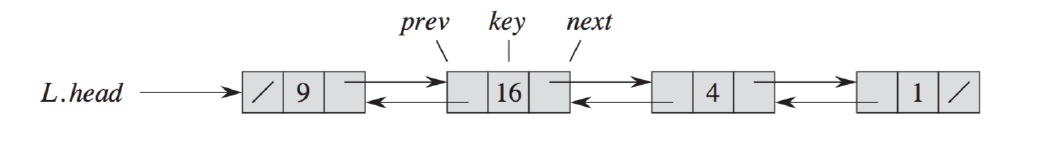
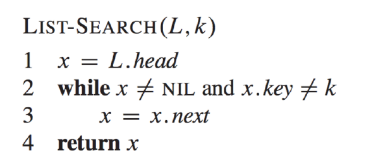
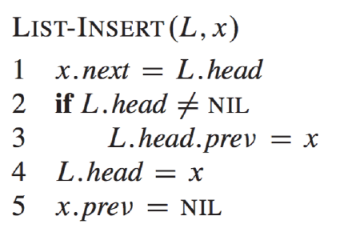
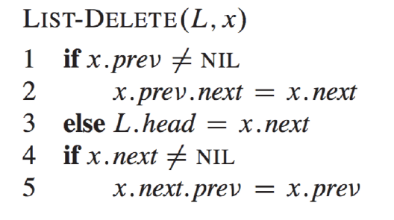
Verem/Stack 🡪 S

* Tulajdonságok
  + Homogén
  + Dinamikus
  + Szekvenciális
  + Folytonos reprezentáció
* LIFO 🡪 Last in, first out
* Van egy hozzá tartozó S.Top érték, ez azt mondja meg hány elem van a veremben
* Műveletek
  + Pop 🡪 Kiveszi a legutóbb betett elemet, hiszen LIFO
    - Vigyázni kell, lehet alulcsordulás, ha S.Top=0
    - Ezzel az elemtörlő műveletünk, és a hozzáférési is
  + Push🡪 Hozzáad egy elemet a veremhez
    - Mivel egy N elemű vektorban tároljuk a vermet, ha S.Top>N, akkor túlcsordulás
    - Ez az elemhozzáadó műveletünk.

Sor/Queue 🡪Q

* Tulajdonságok
  + Homogén
  + Dinamikus
  + Szekvenciális
  + Folytonos reprezentáció
* FIFO 🡪 First in, first out
* Kell két változó
  + Head🡪A sor legelején lévő elem
  + Tail🡪 A legutoljára hozzáadott, azaz utolsó elem
* Műveletek
  + Dequeue🡪Kiveszi a sor elején lévő elemet, mivel first in, first out
    - Ez a hozzáférési, és törlési műveletünk
  + Enqueue 🡪 Beszúr a sor végére egy elemet
    - Ez a hozzáadási műveletünk
* Reprezentáció
  + Naiv🡪Dunno what that is but it aint good
  + Setáló sor🡪 Jobb, de still nem beszélünk nagyon róla
  + Ciklikus sor 🡪 That’s ma boi
* Ciklikus Sor
  + Van egy N elemű vektorunk
  + Megvan a Q.Head, és Q.Tail változónk
  + Sor létrehozásakor Q.Head=Q.Tail=1
  + Ahogy hozzáadunk, és törlünk, úgy ciklikusan megy végig az n elemű vektoron a sor, Körbe-körbe.

Láncolt lista 🡪 L

* Tulajdonságok
  + Homogén
  + Dinamikus
  + Szekvenciális
  + Szétszórt/Láncolt reprezentáció (Benne van a nevében dumbass)
* Láncolt lista minden elemének van
  + Érték
  + És egy mutató, kettős láncolt lista esetén kettő
    - Prev, és Next melyek az előtte és utána lévő elemre mutatnak
* Változók amik még kellenek
  + Head🡪 Pointer ami a lista első elemére mutat a memóriában, ha Head=NULL, akkor üres
* Lista típusok 🡪 Csak kétirányban láncolt listával foglalkozunk lmao
  + Egyirányban láncolt lista
  + Kétirányban láncolt lista
  + Ciklikus lista
  + Mulitilista
* Kétirányban láncolt lista - Műveletek
  + Search 🡪 Végigmegy szekvenciálisan a listán, ha nem talál 🡪 Null
  + Insert 🡪 Beszúrja az adott elemet, a lista legelső pozíciójába
  + Delete 🡪 Eltávolítja az adott elemet a listából, megváltozatja a prev.next-et, és a next.prev-et, hogy megmaradjon a láncolás
* Adatszerkezet műveletek
  + Létrehozás 🡪 Szétszórt reprezentáció
  + Módosítás
    - Hozzáadás 🡪Insert
    - Törlés 🡪Delete
    - Csere nincs
  + Elérés 🡪 Search

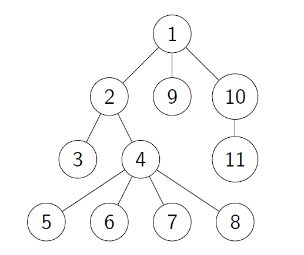
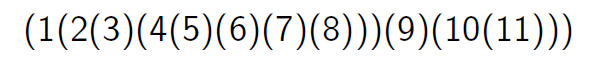
**V. Előadás**

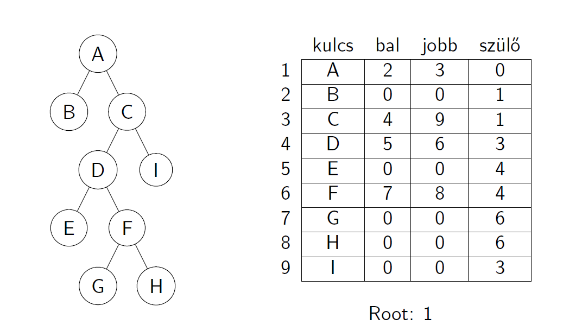
Táblázat

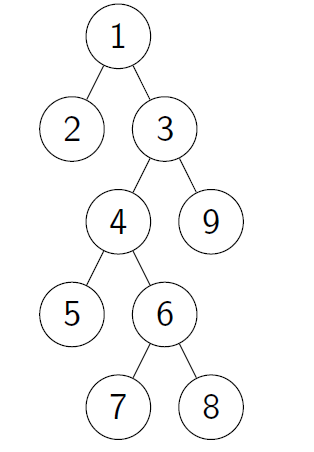
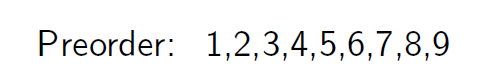
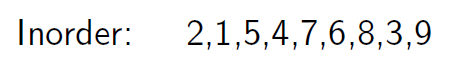
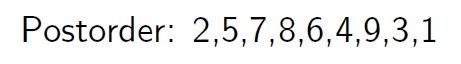
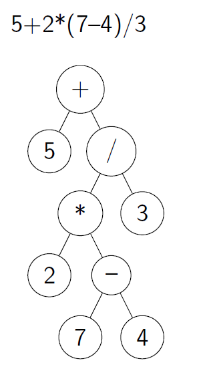
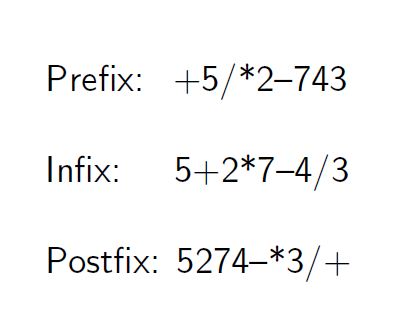
* Tulajdonságok
  + Homogén
  + Dinamikus
  + Asszociatív
  + Néha folytonos, néha láncolt
* Típusok
  + Soros táblázat
  + Önátrendező
    - A Search művelet a legfelülre hozza a keresett elemet, tehát a sokszor keresett elemeket gyorsabban érjük el 🡪 Gyorsan reagál mit használunk sokszor
  + Közvetlen címzésű
    - Van egy kulcsterünk, amiből minden elem kap egy kulcsot
    - Emiatt csak akkor hatékony, ha a kulcstér nem túl nagy
  + Hash tábla
    - Van egy függvényünk, ami meghatározza egy adott kulcstérben hova kerül egy elem
    - Ha ütközés történik, akkor több megoldásunk van
      * Láncolás, ahol a kulcstér minden elem egy mutató, ami egy elemre mutat, ami mutathat a következő elemre és így tovább
      * Lineáris Próba
        + Egy függvény alapján rendelünk kulcsot egy elemhez
      * Kvadratikus próba
        + Egy négyzetes függvény alapján rendelünk kulcsot elemhez
* Műveletek
  + Létrehozás 🡪 Folytonos vagy láncolt
  + Insert 🡪 Elem hozzáadás
  + Delete🡪 Elem törlés
  + Search (mily meglepő) 🡪Elem hozzáférés

**VI. Előadás**

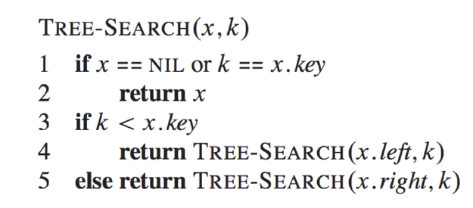
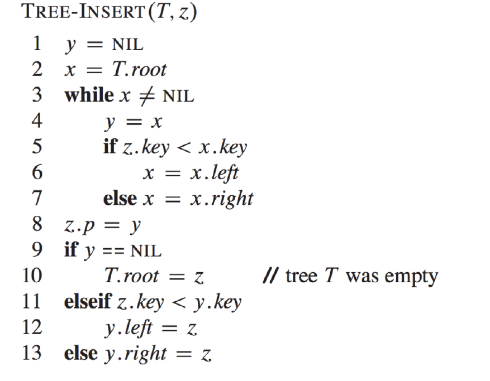
Fa

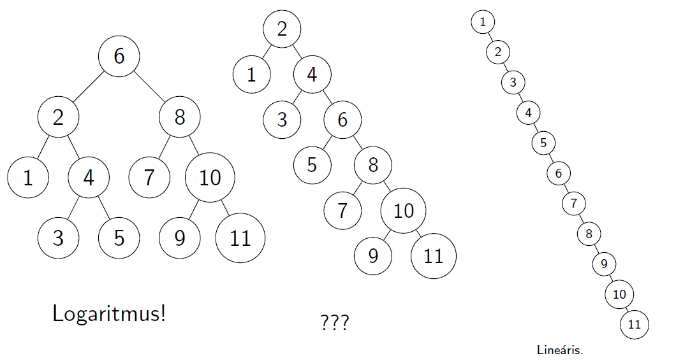
* Csúcsok, és azokat összekötő élek halmaza
* Gráf, mely összefüggő, körmentes, hurokélmentes, párhuzamos él mentes.
* Fa részei
  + Gyökér 🡪 A hiearchiában a legfelső csúcs, nincs szülője
  + Levél 🡪 A hierarchiában egy alsó csúcs, nincs leszármazottja/gyerek
  + Közbenső elem 🡪 Nem levél és gyökér
* Tulajdonságok
  + Homogén
  + Dinamikus
  + Hiearchikus
  + Folytonos/Láncolt
* Folytonos reprezentáció 🡪 Hiearchikus lista
* Többféle bejárás 🡪 Majd később

Bináris Fa

* Fa, amelyben minden csúcsnak max 2 gyerek van, ami vagy jobboldali, vagy baloldali (we got political really sudden )
* Bináris Fa folytonos reprezentációja 🡪 Négysoros reprezentáció
* Bejárás
  + Eval🡪 Egy adott dolog elvégzése, ezen esetben kiiratás
  + Left és right azt jelenti melyik irányba menjünk
  + Rekurzívan működik, miután balra vagy jobbra mentünk, újra meghívjuk a bejáró függvényt, azaz mondjuk az inordernél addig megyünk balra, amíg csak tudunk
  + Preorder
    - Eval
    - Left
    - Right
  + Inorder
    - Left
    - Eval
    - Right
  + Postorder
    - Left
    - Right
    - Eval
* Kifejezésfa
  + Minden csúcs egy műveleti jel, vagy szám
  + Infix bejárás nem egyértelmű
  + Prefix 🡪Jobbról megyünk, amíg nem találunk két  
    számot és egy műveleti jelet egymás mellett, utána   
    elvégezzük
  + Postfix ugyanez csak balról

Bináris Keresőfa

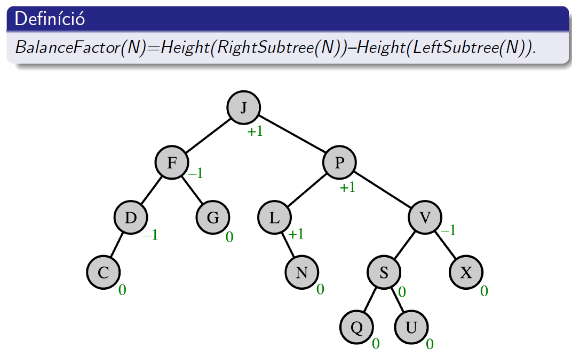
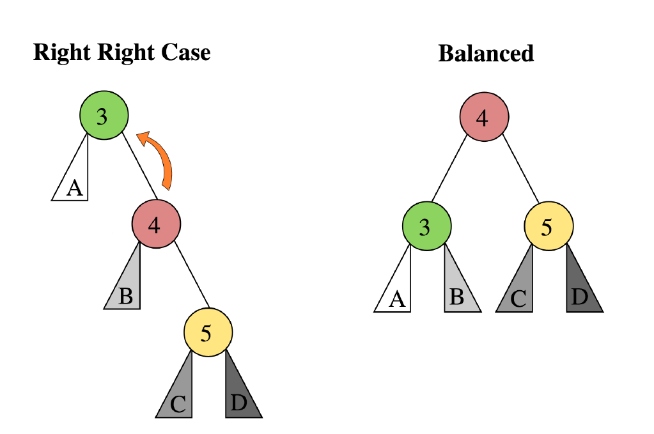
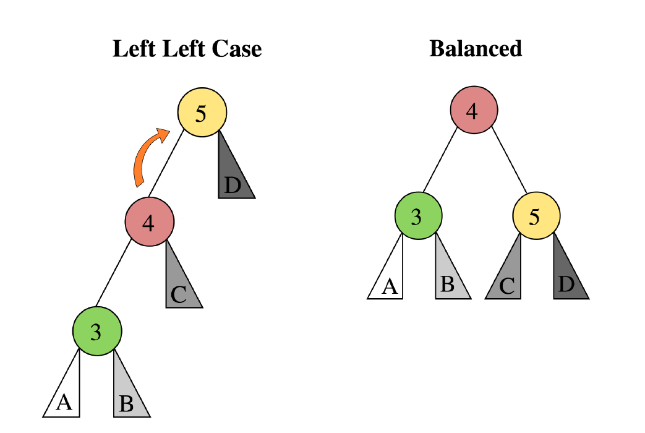
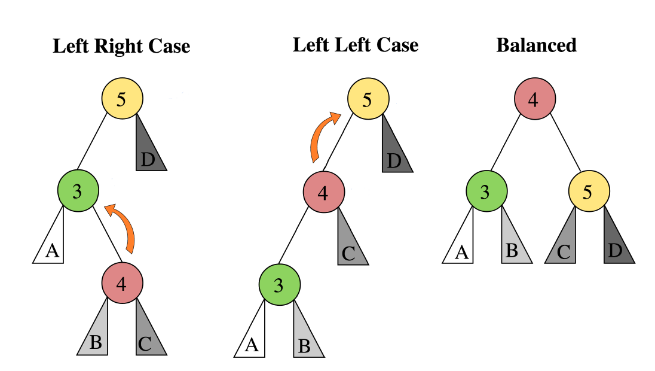
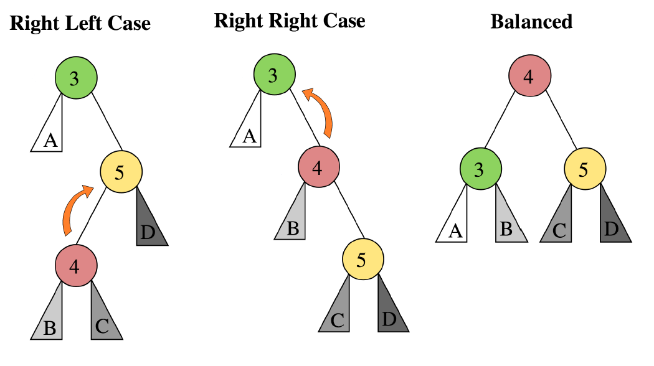
* Olyan Bináris Fa, melyben számok vannak, és minden csúcsra igaz, hogy tőle balra csak tőle kisebb, tőle jobbra csak tőle nagyobb számok vannak
* Műveletek
  + Keresés/Search 🡪 Ha a keresett szám kisebb mint a csúcs, balra megyünk, ha nagyobb jobbra 🡪repeat until done, ha NULL-t kapunk, nincs benne a keresett szám
  + Beszúrás/Insert 🡪 Ha a csúcsnál kisebb a szám, mint amit be akarunk szúrni, balra megyünk, ha nagyobb, jobbra, amíg nem érünk egy számhoz aminek nincs gyerek
  + Törlés 🡪 3 eset
    - Ha a számnak nincs gyerek, amit törölni akarunk, akkor csak egyszerűen töröljük, x szülőjét változtatjuk
    - Ha egy gyereke van, a gyerek x helyére kerül
    - Ha 2 gyereke van, akkor a bal részfa legnagyobb eleme kerül a helyére

**VII. Előadás**

Keresőfák

* Mire jó? 🡪 Egy adott dolog gyors megtalálására
* Nem minden keresőfára igaz, hogy gyorsan tudunk benne keresni
* Ki kell egyensúlyozni a fákat, hogy úgy nézzen ki nagyjából mint a baloldali, mivel úgy a keresési idő logaritmus alapú

AVL Fa – Önkiegenysúlyozó Fa

* Műveletek
  + Létrehozás🡪 Folytonos/láncolt
  + Módosítás
    - Beszúrás: Sima Fa insert + Rebalance
    - Törlés: Siima Fa delete + Rebalance
  + Elérés
    - Tree Search
* Mi is az a Rebalance? Mi az az egyensúlyfaktor?
* Az AVL fa, minden csúcshoz rendel egy számot, ami abból jön ki, hogy a bal részfájában max hány lépés elérni leveleket, majd ebből ki kell vonni, hogy a jobb részfában max hány lépésből lehet elérni leveleket Amíg ez a szám a   
  [-1,0,1] intervallumban van nincs probléma
* AVL Fa Forgatás🡪Akkor kell ha az egyensúly faktor nem es ik az adott intervallumba 🡪4 eset

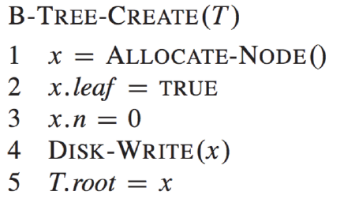
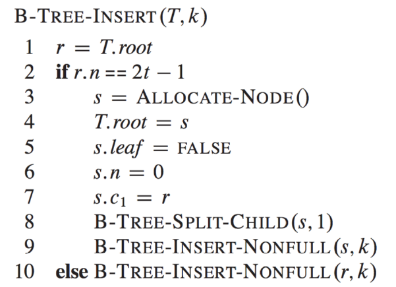
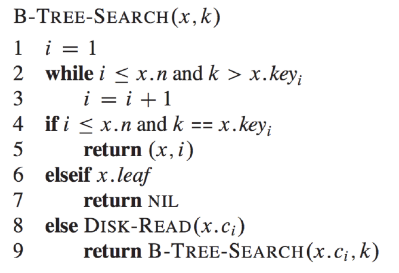
**VIII. Előadás**

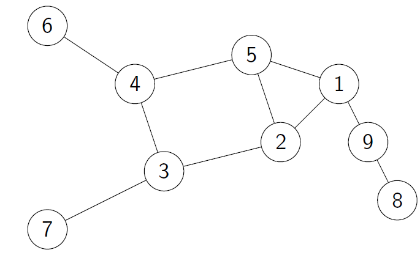
Piros – Fekete Fa

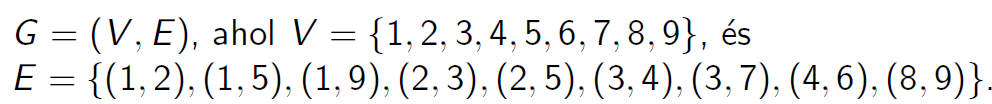
* Szabályok
  + Minden csúcs piros vagy fekete
  + Gyökér mindig fekete
  + Minden levél NULL, és fekete
  + Szülő és gyerek egyszerre nem lehet fekete/Minden piros csúcs gyerekei feketék
  + Minden csúcs esetén az abból induló összes levélig futó úton a fekete csúcsok száma megegyezik
* Kiegyensúlyozottak? Eh, néha
* Műveletek
  + Keresés 🡪 BST keresés
  + Beszúrás 🡪 BST beszúrás +Színezés, forgatás ha kell
  + Törlés 🡪 Nem tanuljuk
* Beszúrás 🡪 a beszúrt csúcs mindig piros
  + Ha a beszúrt csúcs szülője fekete 🡪 Nincs dolgunk
  + Ha a beszúrt csúcs szülője piros 🡪 2 eset
    - Ha a beszúrt csúcs szülője piros, és a nagybácsija(uncle) is piros
      * Beszúrom a csúcsot, majd PUG színezés, azaz az ellentettjére színezzük a beszúrt csúcs szülőjét(P), nagyszülőjét(G), nagybácsiját(U)
    - Ha a beszúrt csúcs nagybácsija fekete 🡪 Ha NULL, akkor is fekete
      * AVL Forgatésok kellenek 🡪 G=Nagyszülő, P=Szülő, X=Beszúrt csúcs
        + LL és RR esetén megcsináljuk a forgatást majd PG Színezés
        + LR és RL esetén megcsináljuk a forgatést, majd XG Színezés

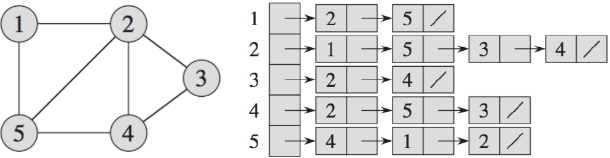
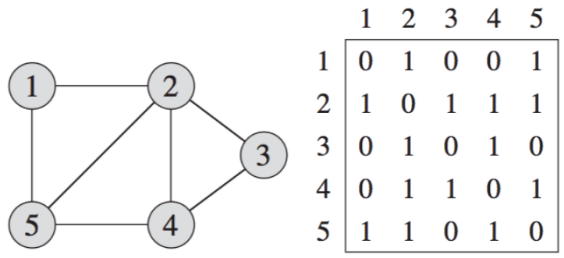
**XI. Előadás**

B-Fa

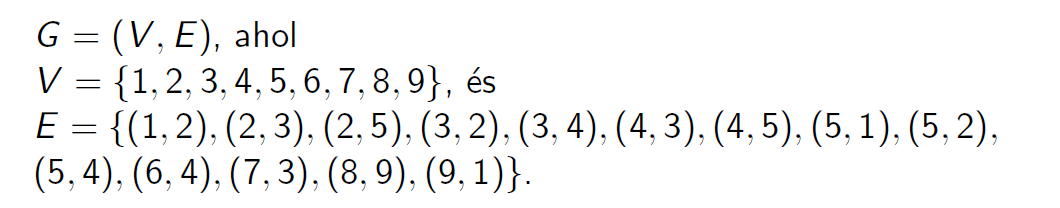
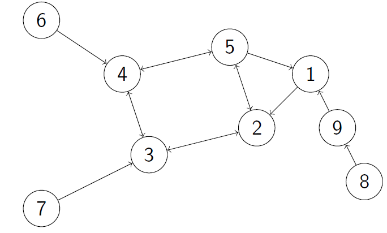
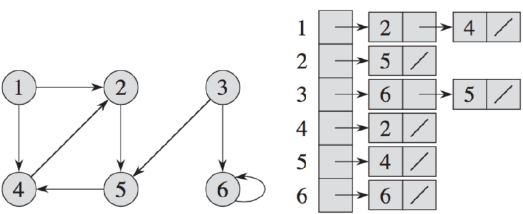
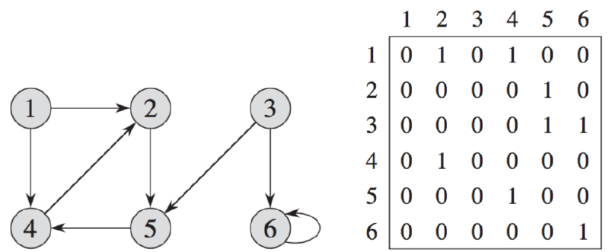
* Egy olyan fa, mely arra van optimalizálva, hogy háttértárakon legyen tárolva, mivel akkora a mérete, hogy memóriában nem lehetne 🡪 Archívumok készítéséhez használják
* Nem bináris fák 🡪 Céljuk a háttértárbeli lapozások számának minimalizálása
* Szabályok
  + Minden B-Fának van egy foka, melyet mi adunk meg Fok=M
  + Minden csúcsnak a gyökér kivételével minimum M-1 értéke van
  + Minden csúcsnak maximum 2M-1 értéke van.
  + Minden csúcsnak a levelek kivételével pontosan Kulcs+1 gyereke van, ahol a kulcs a csúcsban lévő elemek száma
  + Minden levél ugyanazon a szinten van
* Műveletek
  + Létrehozás 🡪 B – Tree Create
  + Modify
    - Beszúrás 🡪 B – Tree Insert 🡪 Lehet reaktív és proaktív
    - Törlés 🡪 B – Tree Delete
  + Elérés 🡪 B – Tree Search

**X. Előadás**

Irányítatlan Gráf

* Csúcsok és élek véges rendezetlen halmaza
* Reprezentáció
  + Szomszédsági lista
  + Szomszédsági Mátrix

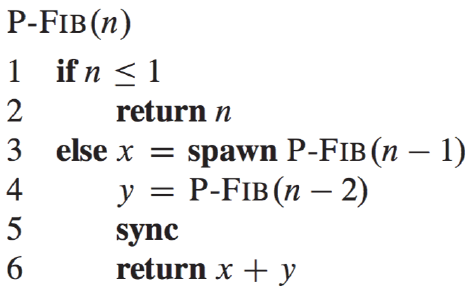
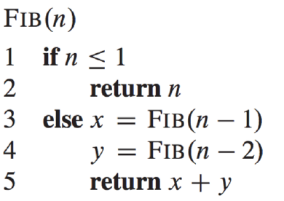
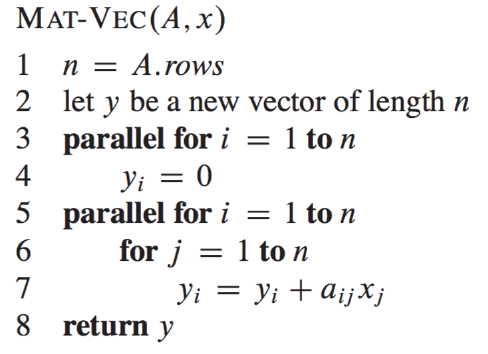
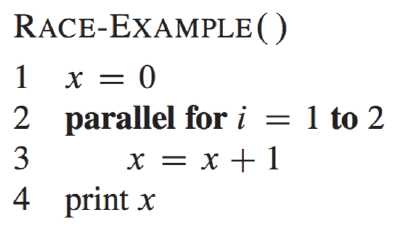
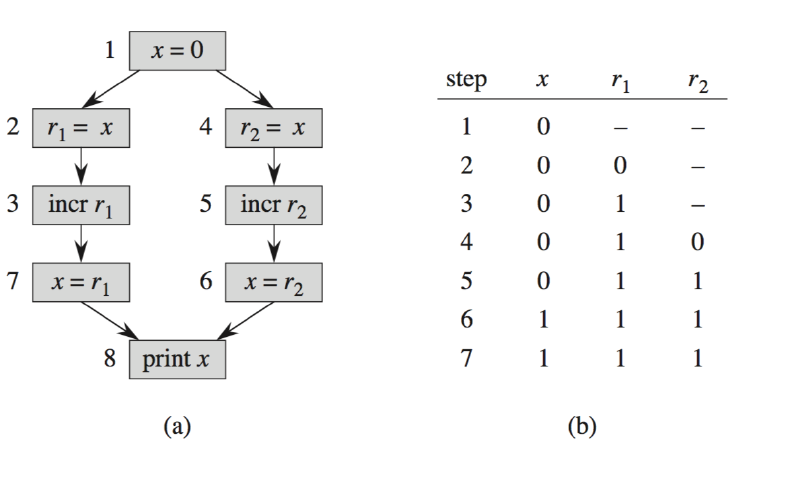
Irányított gráf

* Csúcsok és élek véges, rendezett halmaza
* Reprezentáció
  + Szomszédsági lista Szomszédsági mátrix
  + Melyik a jobb?
    - Szomszédsági lista 🡪 Az élek számától függ
    - Szomszédsági Mátrix
    - Ritka gráfnál a lista a jobb
    - Sűrű gráfnál a mátrix
    - Keresés
      * Mátrixban könnyű ellenőrizni van-e két csúcs között él
      * Listában könnyú végigmenni egy adott csúcshoz tartozó éleken
* Bejárás
* Szélességi bejárás 🡪 BFS Q=Sor, V=vizsgált csúcs
  + 1. Készítünk egy üres sort
  + 2. Enqueue(Q,Start csúcs)
  + 3. V=Dequeue(Q) if V=NULL🡪 Vége
  + 4. Enqueue(Q,a V csúcs összes nem megjelölt, és nem lezárt szomszédja)
  + 5. Megjelölöm a 4-ben talált szomszédokat, melyeket betettem a Q-ba, 4-5 párhuzamosan
  + 6. Lezárom a V-t 🡪 Vissza a 3.-ra
* Mélységi bejárás🡪 DFS S=Verem, V=vizsgált csúcs
  + 1. Készítünk egy üres vermet
  + 2. Push(Q,Start csúcs)
  + 3. V=Pop(S) if V=NULL🡪 Vége
  + 4. Push(S,a V csúcs összes nem megjelölt, és nem lezárt szomszédja)
  + 5. Megjelölöm a 4-ben talált szomszédokat, melyeket betettem a S-be, 4-5 párhuzamosan
  + 6. Lezárom a V-t 🡪 Vissza a 3.-ra

**XI. Előadás**

**Innentől már nem fix semmi, ezeken az előadásokon nem voltam, és nem vettük gyakorlaton szóval megpróbálok összehozni valamit, lehet nem jó**

Párhuzamos algoritmusok

* Minden algoritmus amit eddig vettünk szekvenciális volt, azaz 1 magos processzorokra készült, azonban manapság már minden gépben legalább 2-4, akár 16 mag is lehet
* Párhuzamos algoritmusok 🡪 Dinamikus többszálúság
  + Beágyazott párhuzamosság
    - Alprogramok,függvények, eljárások meghívására, futtatására van lehetőség a főprogram mellett, így egy időben hajtódik össze több programrész
  + Párhuzamos Ciklusok
    - Ciklus minden egyes lépése egyszerre hajtódik végre, egyidőben számolódnak ki az értékek a ciklusváltozó minden egyes értékére.
* Új műveletek pszeudokódban
  + Parralel
  + Spawn
  + Sync
* Fibonacci számok
  + Szekvenciálisan
  + Beágyazott párhuzamos algoritmussal
  + A spawn utáni utasítás egy külön szálon hajtódik végre, míg az y utáni rész az eredeti, szülőprogram részeként fut le, tehát a   
    3. És 4. Sor párhuzamos
  + A főprogram addig nem használhatja az x értékét, azaz ami párhuzamosan ment, amíg nem jut el a sync utasításhoz
* NxN Mátrix szorzása 🡪 Párhuzamos ciklusok
* Hozzáférési verseny
  + Akkor alakul ki, amikor több párhuzamosan futó szálon ugyanahhoz a memóriarészhez próbálunk meg hozzáférni
  + Ebben a példában az X-et 2-re akarnánk növelni, de a kimenet 1 lesz, mivel két külön szálon növeltük 1-el, ami nem azt jelenti hogy 2-vel növeltük.
* Van még valami összefésülés, idk what that is

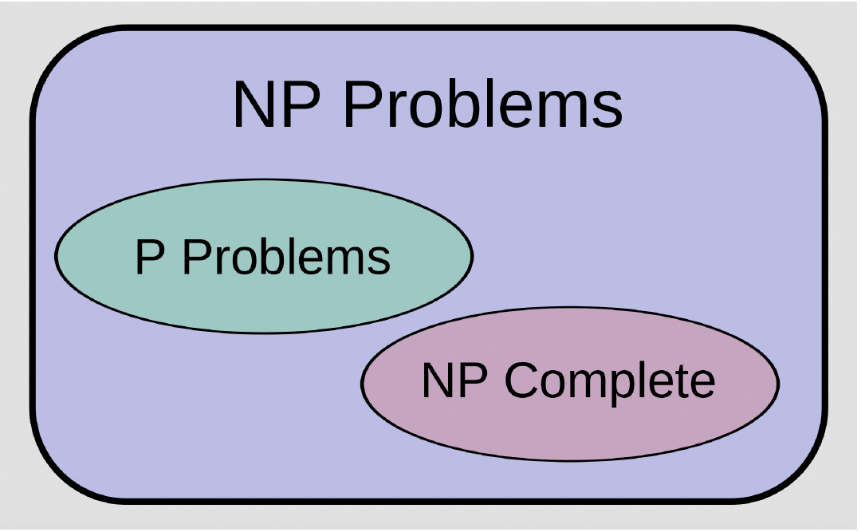
**XII. Előadás**

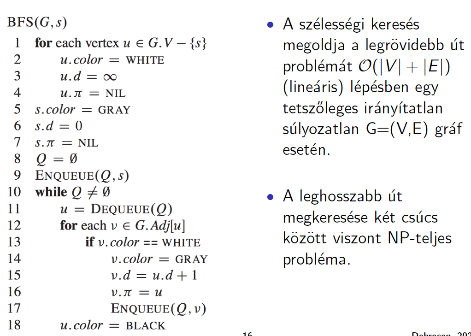
Eldönthetőség, eldönthetetlen, könnyű, nehéz

* Eddig lényegében minden algoritmus polinomiális idejű volt, azaz max O(nk) futási idejű algoritmusokról volt szó.
* Minden probléma megoldható polinomiális időben? 🡪 Nope
* Vannak problémák, melyek nem oldhatóak meg számítógéppel, attól függetlenül mennyi időnk van.
* A polinomiális idejű algoritmussal megoldható problémákat könnyűnek tekintjük, a szuperpolinomiális idejűeket pedig nehéznek
* Miről beszélünk most? NP-Teljes problémák osztálya a fő téma
  + NP Teljes problémára még senki nem adott polinomiális idejű algoritmust, de még szuperpolinomiális alsó becslést sem. Azaz P!=NP,

P vs NP

* Feladat 🡪 Van 400 diákunk akik koliba jelentkeznek, és egy listánk kik nem lehetnek egy szobában, viszont csak 100 helyünk van
  + Könnyű leellenőrizni, hogy egy lista amit kaptunk helyes-e, de egy ilyen listát összeállítani számítógép által, lényegében lehetetlen. Többféleképpen választhatjuk ki azt a 100 diákot, mint amennyi atom van a megfigyelhető univerzomban  
    That’s quite biiiiig
* A P osztály megoldható problémákból áll, azaz bármely bemenet esetén legfeljebb O(nk) időben megoldható, és eldönthető/ellenőrizhető
* Az NP osztály olyan problémákból áll, amelyek polinom időben ellenőrizhetőek, azaz ha valaki valamilyen bizonyítékot adna nekünk egy állításról, annak helyességét le tudnánk ellenőrizni.

NPC - Lmao

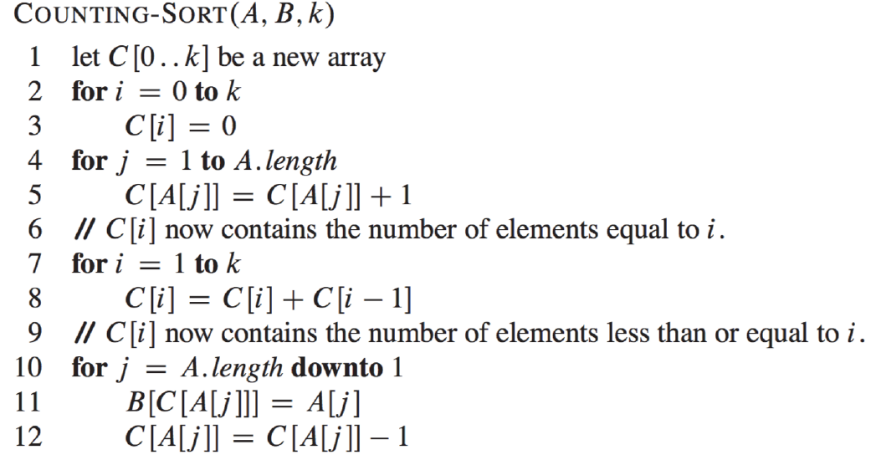
* NP Complete, azaz NP Teljes problémák
* Azok a problémák, melyek legalább olyan nehezek mint bármely NP probléma
* Ha egy NP teljes probléma is megoldható lenne polinomiális időben, akkor az összes megoldható lenne
* P!=NP az állítás, melyet máig nem tudtak sem bebizonyítani, sem cáfolni
* Ami minket általában érdekel, azok az optimalizálási problémák 🡪 Legrövidebb út megtalálása
* Az NP Teljesség azonban nem optimalizálási problémákra alkalmazzuk, hanem eldönthetőségi problémákra 🡪 Egy adott természetes szám prímszám-e
* Lehet keresztezni az optimaliálási és eldönhetőségi problémákat   
  Van egy irányítatlan gráf, és 2 csúcs, továbbá k egész szám, döntsük el létezik-e u és v között olyan út, mely legfeljebb k élből áll
* Másik példa erre az optimalizáció/eldönthetőség példa
* Euler-kör
  + Egy irányított összefüggő gráf(E)-nak O(|E|) lépésben el lehet dönteni van-e Euler köre
* Hamilton-kör
  + Annak a megválaszolása hogy egy adott gráfban van-e Hamilton-Kor NP Teljes probléma
  + Ahhoz, hogy ellenőrizzük, egy adott gráfban van-e Hamilton kör, kell egy bizonyíték, és egy polinomiális idejű algoritmus, hogy el tudjuk dönteni, valóban van-e Hamilton kör egy Gráfban

Karp-Redukció

* Legyen egy A döntési probléma, amit szeretnék polinom idő alatt megoldani
* Tegyük fel, hogy van egy B döntési probléma, amit már meg tudunk oldani polinom idő alatt
* Legyen egy olyan eljárásunk, ami az A probléma minden Alfa esetét átalakítja a B probléma Béta esetévé, úgy hogy
  + Az átalakítás polinom idő alatt történik
  + A válasz minden esetben azonos lesz, azaz Béta pontosan akkor igaz, amikor Alfa is igaz, és Béta pontosan akkor hamis, amikor Alfa is hamis.
* Ez a polinom időre visszevezető algoritmus, vagy Karp Redukció

NP Teljesség

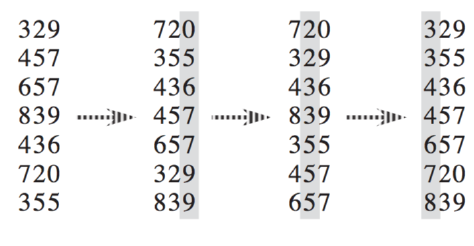
* A B probléma NP-nehéz, ha minden NP-beli A probléma esetén létezik az A problémának Karp Redukciója a B problémára.
* Egy probléma NP teljes, ha
  + NP-beli
    - Igazoljuk polinom idejú ellenőrző algoritmussal, és bizonyítékkal
  + NP-Nehéz
    - Igazoljuk Karp Redukció, és egy előző igazolatlan NP Teljes probléma felhasználásával

**XIII. Előadás**

Leszámláló rendezés

* Don’t really know
* Neten néztem magyarázatot hogyan működik, nem olyan vészes ha egyszer megérti az ember, de leírni nem tudom lmao

Radix Sort

* Háromszoros Counting Sort
* Először a 3. Majd 2. Majd 1. számjegyekre