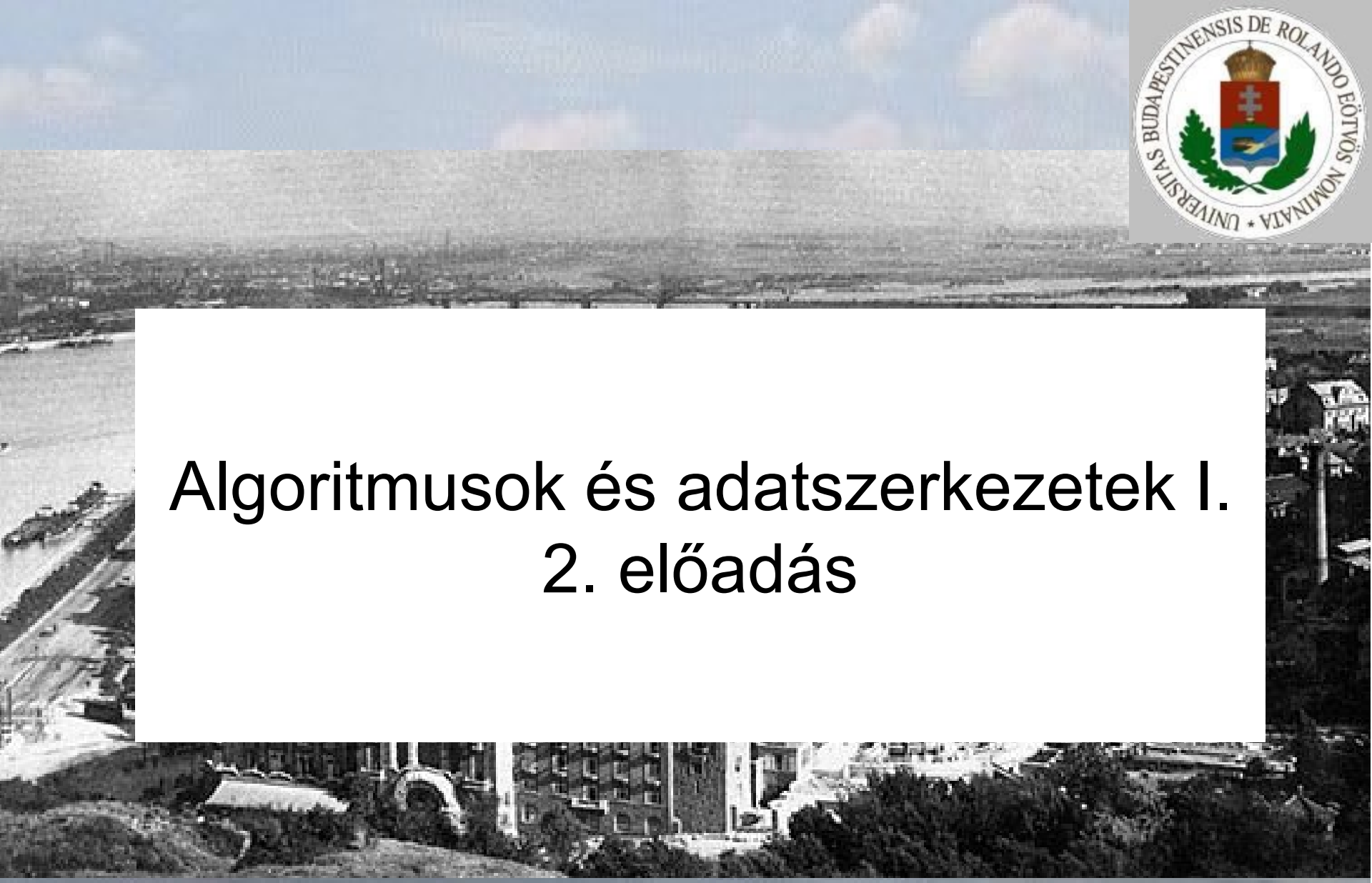


# Algoritmusok és adatszerkezetek I.
# 2. előadás

# Sorozattípusok



# Verem

**Verem= speciális sorozattípus**

**Műveletei:** Üres, üres?, Verembe, Veremből, tető

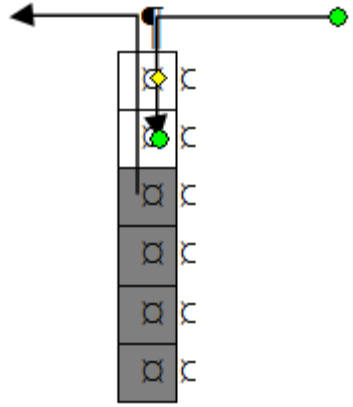
Üres: Verem

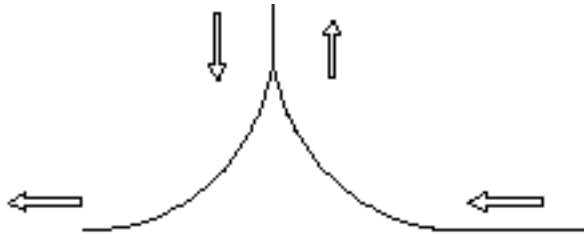
üres?(Verem): Logikai

tető(Verem): Elem ∪{NemDef}

Verembe(Verem,Elem): Verem ∪{NemDef}

Veremből(Verem,Elem): (Verem ×Elem) ∪{NemDef}





# Verem

## A műveletek tulajdonságai

- Üres veremre a verembe rakás és az üresség vizsgálat kivételével minden művelet értelmetlen.
- A veremből kivétel a verembe utoljára tett elemet veszi ki, azaz a verem alkalmas egy sorozat megfordítására.
- A tetőn mindig a verembe utoljára betett (legfelül levő) elem látszik.

# Verem

## Megvalósítás folytonos, szekvenciális ábrázolással

Verem(Típus TElem):

Konstans MaxMélység: Egész(???)

Típus VeremElem=TElem

Változó t: Tömb(1..MaxMélység: VeremElem)
teteje: 0..MaxMélység

Tárolni kell a verem elemszámát (teteje), valamint egy tömbben a verem elemeit (1-től indexelve).

Probléma: MaxMélység=????

Ha nem ismerjük → dinamikus tömb!

# Verem

```
Eljárás Üres(V):
  V.teteje:=0
Eljárás vége.
Függvény üres?(V):
  üres?:=V.teteje=0
Függvény vége.
Függvény tető(V):
  tető:=V.t(V.teteje)
Függvény vége.
```

# Verem

```
Eljárás Verembe(V,e):
  V.teteje:=V.teteje+1
  V.t(V.teteje):=e
Eljárás vége.
Eljárás Veremből(V,e):
  e:=V.t(V.teteje)
  V.teteje:=V.teteje-1
Eljárás vége.
```

Meggondolandó: hibakezelés hol és mikor kell?

# Verem alkalmazás

Egy kifejezés (,) és [,] zárójelpárokat tartalmaz.

Ellenőrizzük a helyességét!

Hibás esetek:

- `x([y)]` – hibás párosítás
- `x(a` – nyitó zárójelnek nincs csukó párja
- `x)a(` – csukó zárójelnek nincs nyitó párja

# Verem alkalmazás

```
Függvény helyes(K):
 i:=1; h:=igaz; Üres(V)
 Ciklus amíg i≤hossz(K) és h
  Ha K(i)='(' vagy K(i)='[' akkor Verembe(V,K(i))
  különben ha K(i)=')' vagy K(i)=']' akkor
            Veremből(V,s)
            h:= K(i)=')' és s='(' vagy
                K(i)=']' és s='['
  i:=i+1
 Ciklus vége
 helyes:=h és nem V.hiba és üres?(V)
Függvény vége.
```

# Verem alkalmazás

Egy számsorozat minden eleméhez add meg az őt megelőzők közül a hozzá legközelebb álló, nála kisebb elem indexét! Ha nincs ilyen, akkor -1-et!

# Verem alkalmazás

```
Legközelebbiek(X,Y):
 Y(1):=-1
 Ciklus i=2-től N-ig
  j:=i-1
  Ciklus amíg j>0 és X(j)≥X(i)
   j:=j-1
  Ciklus vége
  Ha j=0 akkor Y(i):=-1 különben Y(i):=j
Ciklus vége
Eljárás vége.
```

A két ciklus miatt a futási idő az elemszám négyzetével arányos.

# Verem alkalmazás

Egy számsorozat minden eleméhez add meg az őt megelőzők közül a hozzá legközelebb álló, nála kisebb elem indexét! Ha nincs ilyen, akkor -1-et!

Megoldási ötlet: tároljuk veremben azokat az elemeket, amelyek még bekerülhetnek az eredménybe!

Invariáns tulajdonság: a veremben az elemek szigorúan monoton növekvően vannak, a tetején az i-edik elemmel.

Példa: 3, 6, 4, 9, 8, 5

Verem állapotok: (), (3), (3,6), (3,4), (3,4,9), (3,4,8), (3,4,5)

# Verem alkalmazás

```
Legközelebbiek(X,Y):
 Y(1):=-1; Üres(V); Verembe(V,1)
 Ciklus i=2-től N-ig
  Ciklus amíg nem üres?(V) és X(tető(V))≥X(i)
   Veremből(V,t)
  Ciklus vége
  Ha üres?(V) akkor Y(i):=-1
              különben Y(i):=tető(V)
  Verembe(V,i)
 Ciklus vége
Eljárás vége.
```

Példa: 3, 6, 4, 9, 8, 5

Verem állapotok:

(), (3), (3,6), (3,4), (3,4,9), (3,4,8), (3,4,5)

# Sor

**Sor= speciális sorozattípus**

**Műveletei:** Üres, üres?, Sorba, Sorból, első

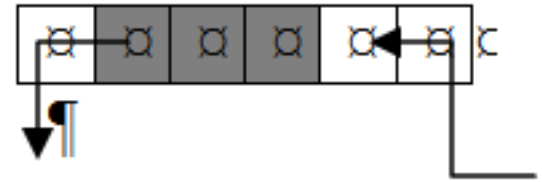
Üres: Sor

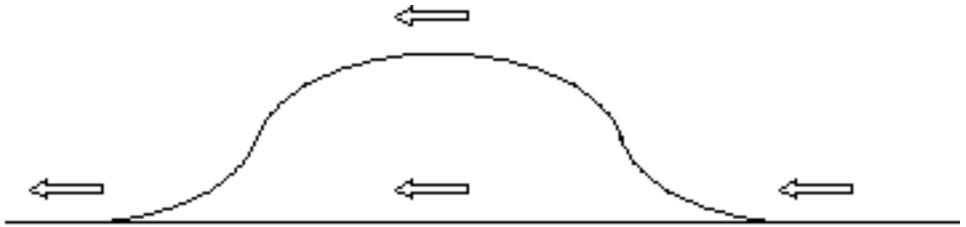
üres?(Sor): Logikai

első(Sor): Elem ∪{NemDef}

Sorba(Sor,Elem): Sor ∪{NemDef}

Sorból(Sor,Elem): (Sor ×Elem) ∪{NemDef}





# Sor

## A műveletek tulajdonságai

- Üres sorra a sorba rakás és az üresség vizsgálat kivételével minden művelet értelmetlen.
- A sorból kivétel a sorba legrégebben tett elemet veszi ki, azaz a sor alkalmas egy sorozat feldolgozására, ha a keletkezés és a feldolgozás üteme eltér egymástól.
- A sor elején mindig a sorba legrégebben betett (legelöl levő) elem látszik.

# Sor

## Megvalósítás folytonos, szekvenciális ábrázolással

Sor(Típus TElem):

Konstans MaxHossz: Egész(???)

Típus SorElem=TElem

Változó t: Tömb(1..MaxHossz: SorElem)
első,utolsó,db: 0..MaxHossz

A tömböt ciklikusan kezeljük: ha a végére értünk és az elején már szabadult fel hely, ott folytatjuk a kitöltést. A sorban az `első` és az `utolsó` tömbindexű hely között `db` darab elem van.

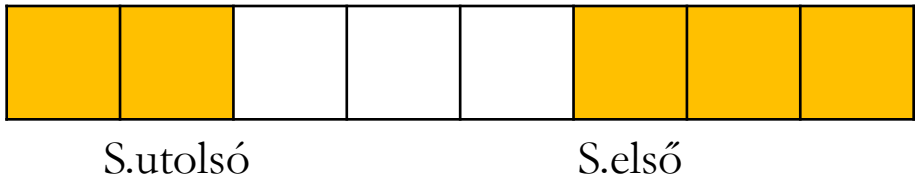
Probléma: MaxHossz=????

# Sor

```
Eljárás Üres(S):
  S.első:=1; S.utolsó:=0; S.db:=0
Eljárás vége.
Függvény üres?(S):
  Üres?:=S.db=0
Függvény vége.
Függvény első(S):
  első:=S.t(S.első)
Függvény vége.
```

# Sor

```
Eljárás Sorba(S,e):
  Ha S.utolsó=Maxhossz akkor S.utolsó:=1
     különben S.utolsó:=S.utolsó+1
  S.db:=S.db+1; S.t(S.utolsó):=e
Eljárás vége.
Eljárás Sorból(S,e):
  e:=S.t(S.első); S.db:=S.db-1
  Ha S.első=Maxhossz akkor S.első:=1
     különben S.első:=S.első+1
Eljárás vége.
```




# Sor alkalmazás

**Példa:**

Egy vasútvonal két állomásán egy nap feljegyeztük, az összes egyik irányba áthaladó vonat indulási, illetve érkezési idejét (idő szerinti sorrendben). Adjuk meg az áthaladt vonatok menetidőit a két állomás között! (Nincs előzés.)

Ötlet: A korábban érkező adatot egy sorban várakoztatjuk addig, amíg a párja meg nem érkezik.

# Sor alkalmazás

```
Forgalom:
  Üres(S)
  Ciklus amíg jön vonat
    Olvas(adat)
    Ha adat.állomás=1
         akkor  Sorba(S,adat.idő)
       különben Sorból(S,előző)
                Ír(adat.idő-előző)
  Ciklus vége
Eljárás vége.
```

# Sor alkalmazás

## Példa:

Egy egyirányú utcában két jelzőlámpánál figyelik az áthaladó autókat: *mikor* haladt át a *lámpá*nál. Az adatok megfigyelési *idő szerinti* sorrendben érkeznek. N db autót figyeltek meg. Csoportosítsuk az adatokat, hogy az *első* lámpánál történt megfigyelések – időbeli sorrendjüket megtartva – előzzék meg a *második* lámpánál megfigyelt adatokat! (Az utcában előzés nincs.)

Ötlet: Az 1-es lámpa adatai azonnal írhatók az eredménybe, a 2-es lámpa adatait egy átmeneti sorban tároljuk, s a végén írjuk az eredménybe. A sor – ha a mérete indokolja – persze lehet háttértáron is.

# Sor alkalmazás

Ötlet: Az 1-es lámpa adatai azonnal írhatók az eredménybe, a 2-es lámpa adatait egy átmeneti sorban tároljuk, s a végén írjuk az eredménybe. A sor – ha a mérete indokolja – persze lehet háttértáron is.

A bemenet és a kimenet is forgalom típusú adatok fájlja, és a sorban is ilyeneket tárolunk:

```
Forgalom=Rekord(lámpa: {1,2}, idő: Egész)
Változó S: Sor(Forgalom)
```

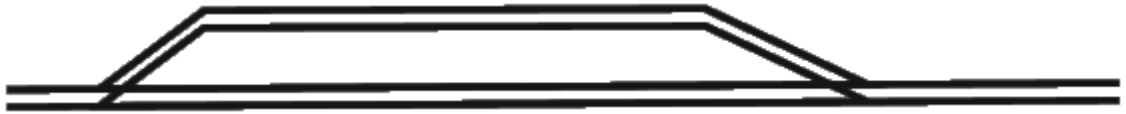
# Sor alkalmazás

```
Forgalom:
  Üres(S)
  Ciklus amíg jön autó
    Olvas(adat)
    Ha adat.lámpa=1 akkor Ír(adat)
                    különben Sorba(S,adat)
  Ciklus vége
  Ciklus amíg nem üres?(S)
    Sorból(S,adat); Ír(adat)
  Ciklus vége
Eljárás vége.
```

# Sor alkalmazás

**Példa:**

Egy vasútállomáson a fő vágányról kitérő vágányt építettek, mindegyiken csak jobbról balra lehet haladni. A kitérőn akárhány kocsit elhelyezhetünk, a fő vágányon azonban nem állhat meg kocsi. Egy szerelvény érkezik jobbról, sorszámozott, összekevert kocsikkal. A kocsikat 1 és K közötti különböző sorszámokkal azonosítjuk.



Készíts programot, amely megadja, hogy maximum hány kocsit tudunk sorba rendezni a kitérő segítségével és elküldeni balra úgy, hogy a kitérőn végül nem marad kocsi!

Példa: 4, 5, 2, 6, 3, 1

Sor: (4), (4,5), 2 át, (4,5,6), 3 át, sorból mind át

# Sor alkalmazás

```
Vasútállomás:
 ut:=0; jó:=igaz; i:=1; Üres(S)
 Ciklus amíg i≤k és v[i]≥ut
   Ha üres?(S) akkor Sorba(S,v[i])
   különben ha v[i]>utolsó(S) akkor Sorba(S,v[i])
   különben Ciklus amíg v[i]>első(S)
               Sorból(S,x)
            Ciklus vége
            ut:=v[i]
   Elágazások vége
   i:=i+1
  Ciklus vége
  ered:=i-1
Eljárás vége.
```

Ötlet: Sorba tesszük a kitérőre kerülő kocsikat, tároljuk a balra kilépő utolsó kocsi azonosítóját.

Új művelet a sor utolsójának vizsgálata.

Példa: 4, 5, 2, 6, 3, 1

Sor: (4), (4,5), 2 át, (4,5,6), 3 át, sorból mind át

# Kétvégű sor

**Kétvégű sor= speciális sorozattípus**

**Műveletei:** Üres, üres?, Sorelejére, Sorvégére, első, Sorelejéről, Sorvégéről

Üres: Sor

Sorelejére(Sor,Elem): Sor ∪{NemDef}

Sorvégére(Sor,Elem): Sor ∪{NemDef}

Sorelejéről(Sor,Elem): (Sor ×Elem) ∪{NemDef}

Sorvégéről(Sor,Elem): (Sor ×Elem) ∪{NemDef}

üres?(Sor): Logikai

első(Sor): Elem ∪{NemDef}

utolsó(Sor): Elem ∪{NemDef}

# Kétvégű sor alkalmazás

Egy számsorozat minden eleméhez add meg a vele végződő M hosszú részsorozat minimumát! Az első M elemnél minden i-re a vele végződő i hosszú részsorozat minimumát kell megadni!

Megoldási ötlet: sok minimumkiválasztás

Példa: 5, 3, 6, 4, 7, 9, 2, 6, 4  A 3 hosszú részek minimumát keressük
Sor állapotok: (5), (3), (3,6), (3,4), (4,7), (4,7,9), (2), (2,6), (2,4)

# Kétvégű sor alkalmazás

```
Minimumok(X,Y):
 Y(1):=1
 Ciklus i=2-től N-ig
  min:=i
  Ciklus j=max(1,i-M+1)-től i-1-ig
   Ha X(j)<X(min) akkor min:=j
  Ciklus vége
  Y(i):=min
 Ciklus vége
Eljárás vége.
```

Példa: 5, 3, 6, 4, 7, 9, 2, 6, 4

A 3 hosszú részek minimumát keressük

# Kétvégű sor alkalmazás

Egy számsorozat minden eleméhez add meg a vele végződő M hosszú részsorozat minimumát! Az első M elemnél minden i-re a vele végződő i hosszú részsorozat minimumát kell megadni!

Megoldási ötlet: tároljuk kétvégű sorban azokat az elemeket, amelyek még bekerülhetnek az eredménybe!

Invariáns tulajdonság: a sorban az elemek szigorúan monoton növekvően vannak. Az elejéről kikerülnek azok, amelyek M-nél régebbiek. A végéről kerülnek ki azok, akik a monotonitási feltételt nem teljesítik.

Példa: 5, 3, 6, 4, 7, 9, 2, 6, 4  A 3 hosszú részek minimumát keressük
Sor állapotok: (5), (3), (3,6), (3,4), (4,7), (4,7,9), (2), (2,6), (2,4)

# Kétvégű sor alkalmazás

```
Minimumok(X,Y):
 Y(1):=1; Üres(S); Sorvégére(S,1)
 Ciklus i=2-től N-ig
  Ha i-első(S)=M akkor Sorelejéről(S,t)
  Ciklus amíg nem üres?(S) és X(i)≤X(utolsó(S))
   Sorvégéről(S,t)
  Ciklus vége
  Sorvégére(S,i); Y(i):=első(S)
Ciklus vége
Eljárás vége.
```

Példa: 5, 3, 6, 4, 7, 9, 2, 6, 4

A 3 hosszú részek minimumát keressük

Sor állapotok:

(5), (3), (3,6), (3,4), (4,7), (4,7,9), (2), (2,6), (2,4)

# Sorozattípusok