5. előadás

A lista

Lista, verem, sor

Adatszerkezetek és algoritmusok előadás
2011. március 9.

Kósa Márk és Pánovics János
Debreceni Egyetem
Informatikai Kar

A lista adatszerkezet

Dinamikus, homogén, szekvenciális adatszerkezet.

Jelölések:

lista:	$q = [x_1, x_2, \ldots, x_n]$
üres lista:	$q = [\,]$
a lista feje:	x_1
a lista farka:	$[x_2, \ldots, x_n]$
a lista vége:	x_n
a lista hossza, mérete:	n vagy $\lvert q \rvert$

Kósa Márk
Pánovics János

A lista adatszerkezet

Alapműveletek

- Hozzáférés, elérés: közvetlen.

$$q[i] = x_i$$

- Részlistaképzés, allistaképzés:

$$q[i \ldots j] = [x_i, x_{i+1}, \ldots, x_{j-1}, x_j]$$

- Konkatenáció, egyesítés, összefűzés:

$$r = [y_1, \ldots, y_m]$$
$$q \,\&\, r = [x_1, x_2, \ldots, x_n, y_1, \ldots, y_m]$$

Kósa Márk
Pánovics János

A lista adatszerkezet

Listával végezhető műveletek

- Létrehozás: explicit módon felsoroljuk az elemeit.
- Bővítés: bárhol bővíthető. Bővítéskor részlistákat képzünk, majd azokat konkatenáljuk az új elemből/elemekből álló részlistával. A k-adik elem mögé történő bővítés:

$$q[1 \ldots k] \,\&\, [\text{elem}] \,\&\, q[(k+1) \ldots n]$$

- Törlés: megvalósítható a fizikai törlés, melynek során részlistákat képzünk (melyekben már nem szerepel(nek) a törlendő elem(ek)), majd konkatenáljuk ezeket a részlistákat. A k-adik elem törlése:

$$q[1 \ldots (k-1)] \,\&\, q[(k+1) \ldots n]$$

- Csere: bármelyik elem cserélhető. Részlistákat képzünk, majd azokat konkatenáljuk az új értékből álló részlistával. A k-adik elem cseréje:

$$q[1 \ldots (k-1)] \,\&\, [\text{elem}] \,\&\, q[(k+1) \ldots n]$$

- Rendezés: értelmezhető, bármelyik rendezési algoritmus használható.
- Keresés: értelmezhető, bármelyik keresési algoritmus használható.
- Elérés: soros vagy közvetlen.
- Bejárás: értelmezhető.
- Feldolgozás: a lista alapműveletei segítségével.

Kósa Márk
Pánovics János

A lista adatszerkezet reprezentációja

- Folytonos reprezentáció: vektorral.
- Szétszórt reprezentáció: kétirányban láncolt listával.

Kósa Márk
Pánovics János

A lista adatszerkezet

A szélső elemekkel végezhető speciális műveletek

- ACCESS HEAD: az első elem elérése:

$$q[1] = x_1$$

- PUSH: bővítés az első elem előtt:

$$[\text{elem}] \,\&\, q$$

- POP: az első elem törlése:

$$q[2 \ldots]$$

- ACCESS END: az utolsó elem elérése:

$$q[n] = x_n$$

- INJECT: bővítés az utolsó elem után:

$$q \,\&\, [\text{elem}]$$

- EJECT: az utolsó elem törlése:

$$q[\ldots n-1]$$

Kósa Márk
Pánovics János

A verem adatszerkezet

Speciális lista adatszerkezet, melynek alapműveletei a speciális listaműveletek közül a következők:

- az első elemhez történő hozzáférés (TOP)
- bővítés az első elem elé (PUSH)
- az első elem törlése (POP)

Az első elemhez történő hozzáférés és az első elem törlésének műveletét egy műveletként is definiálhatjuk.

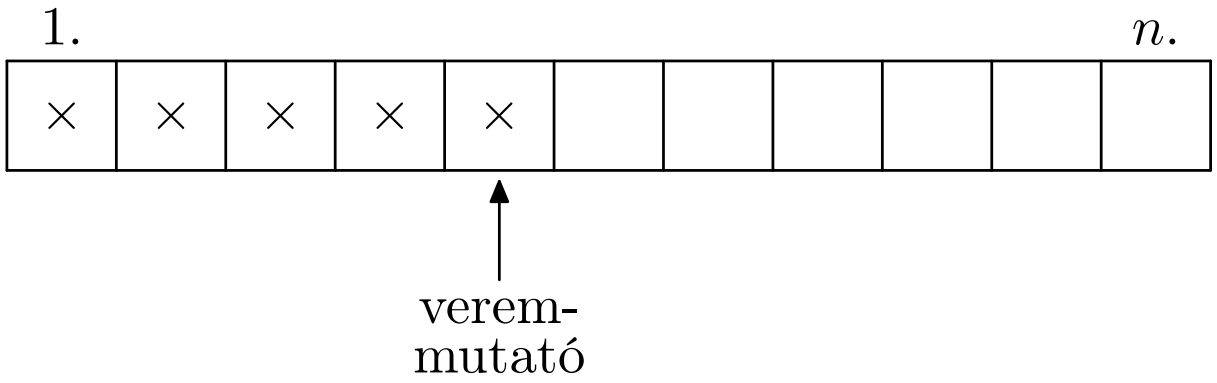
Veremmel végezhető műveletek

- Létrehozás: üres verem.
- Bővítés: az első elem elé.
- Csere: nincs.
- Törlés: fizikai, az első elemet.
- Rendezés, keresés és bejárás: nem értelmezett.
- Elérés: az első elemet közvetlenül, a többit sehogyan sem.
- Feldolgozás: Last In First Out (LIFO) adatszerkezet, az utolsóként érkező elemet dolgozzuk fel először.

Kósa Márk
Pánovics János

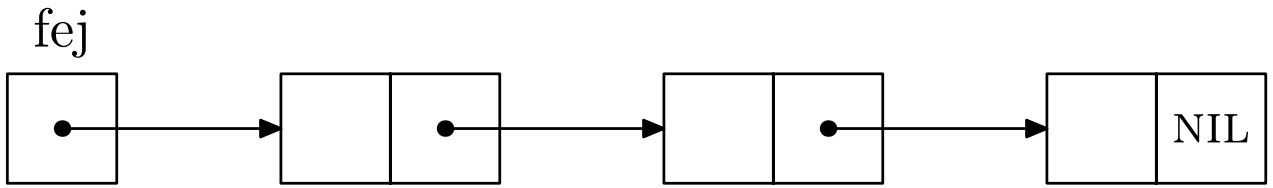
A verem adatszerkezet reprezentációi

Folytonos reprezentáció:



A veremmutató mindig a verem tetején lévő elemet indexeli. Ha a veremmutató értéke 0, a verem üres. Ha a veremmutató értéke *n*, a verem tele van.

Szétszórt reprezentáció: egyirányban láncolt listával.



A fej mutató mindig a verem tetején lévő elemre mutat. Ha a fejnek NIL az értéke, a verem üres.

Kósa Márk
Pánovics János

A sor adatszerkezet

Speciális lista adatszerkezet, melynek alapműveletei a speciális listaműveletek közül a következők:

- az első elemhez történő hozzáférés (GET)
- az első elem törlése (GET)
- bővítés az utolsó elem mögé (PUT)

Az első elemhez történő hozzáférés és az első elem törlésének műveletét egy műveletként is definiálhatjuk.

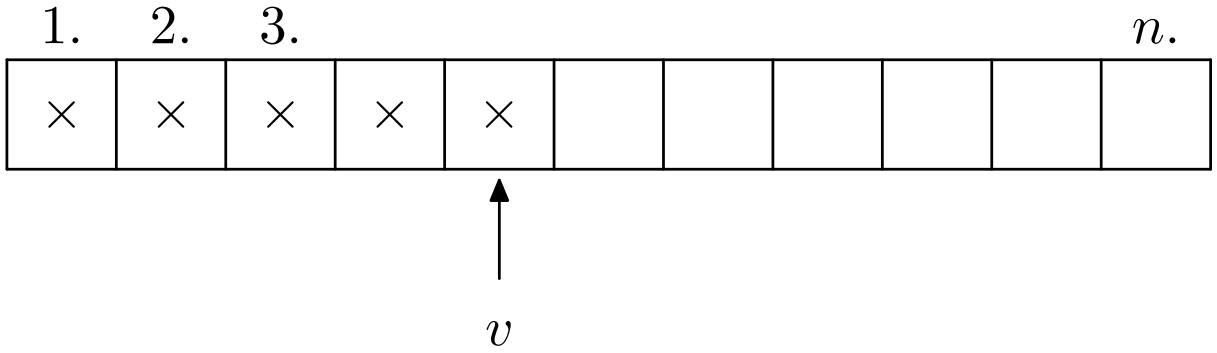
Sorral végezhető műveletek

- Létrehozás: üres sor.
- Bővítés: az utolsó elem mögé.
- Csere: nincs.
- Törlés: fizikai, az első elemet.
- Rendezés, keresés és bejárás: nem értelmezett.
- Elérés: az első elemet közvetlenül, a többit sehogyan sem.
- Feldolgozás: First In First Out (FIFO) adatszerkezet, az elsőként érkező elemet dolgozzuk fel először.

Kósa Márk
Pánovics János

A sor adatszerkezet folytonos reprezentációi

Fix kezdetű sor:



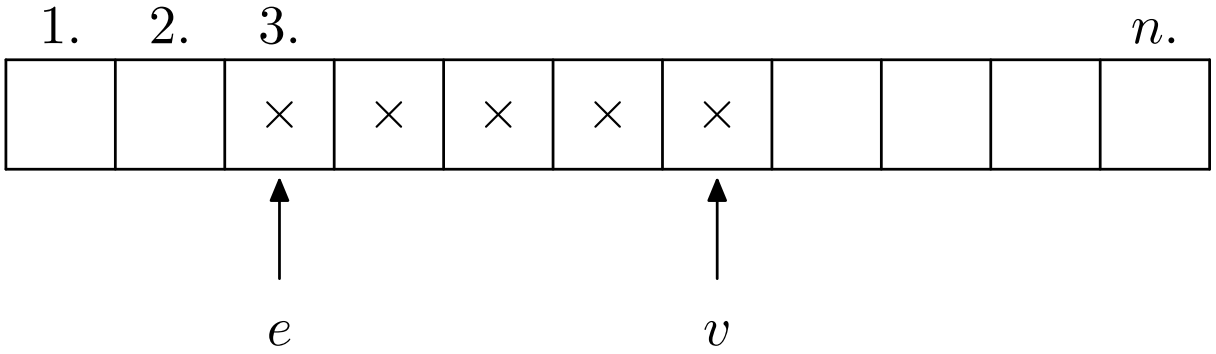
A sor első elemének a helye rögzített, mindig az 1. indexű tárhely a vektorban. A v (vége) mutató a sor utolsó elemét indexeli. Üres a sor, ha $v = 0$. Tele van a sor, ha $v = n$.

Az új elemet a $(v + 1)$-edik pozícióra helyezzük el, ha a sor nincs tele. Törölni az 1. pozíción lévő elemet tudjuk, ha a sor nem üres. Törléskor a sor megmaradó elemeit egy tárhellyel előrébb csúsztatjuk.

Kósa Márk
Pánovics János

A sor adatszerkezet folytonos reprezentációi

Vándorló sor:



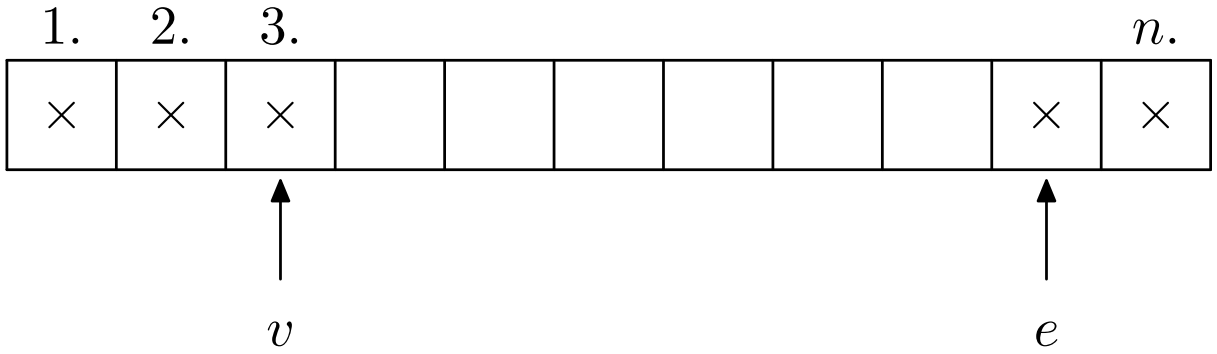
A sor első elemét az e (eleje) mutató, az utolsót a v (vége) mutató indexeli. Üres a sor, ha $e = 0$ és $v = 0$. Tele van a sor, ha $e = 1$ és $v = n$.

Ha bővítéskor $v = n$, de a sor nincs tele, akkor először a sor minden elemét $e - 1$ pozícióval előrébb csúsztatjuk, majd végrehajtjuk a bővítést az új elemmel. Az új elemet a $(v + 1)$-edik pozícióra helyezzük el, ha a sor nincs tele. Törölni az e-edik pozíción lévő elemet tudjuk, ha a sor nem üres.

Kósa Márk
Pánovics János

A sor adatszerkezet folytonos reprezentációi

Ciklikus sor:



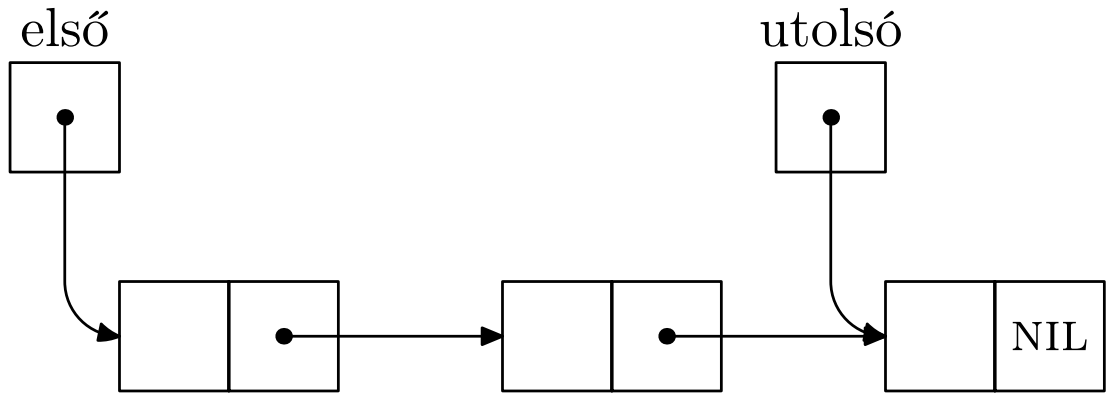
A sor első elemét az e (eleje) mutató, az utolsót a v (vége) mutató indexeli. Üres a sor, ha $e = 0$ és $v = 0$. Tele van a sor, ha $e = v \bmod n + 1$.

Az új elemet a $(v \bmod n + 1)$-edik pozícióra helyezzük el, ha a sor nincs tele. Törölni az e-edik pozíción lévő elemet tudjuk, ha a sor nem üres.

Kósa Márk
Pánovics János

A sor adatszerkezet szétszórt reprezentációja

Szétszórt reprezentáció egyirányban láncolt listával, két segédmutatóval:

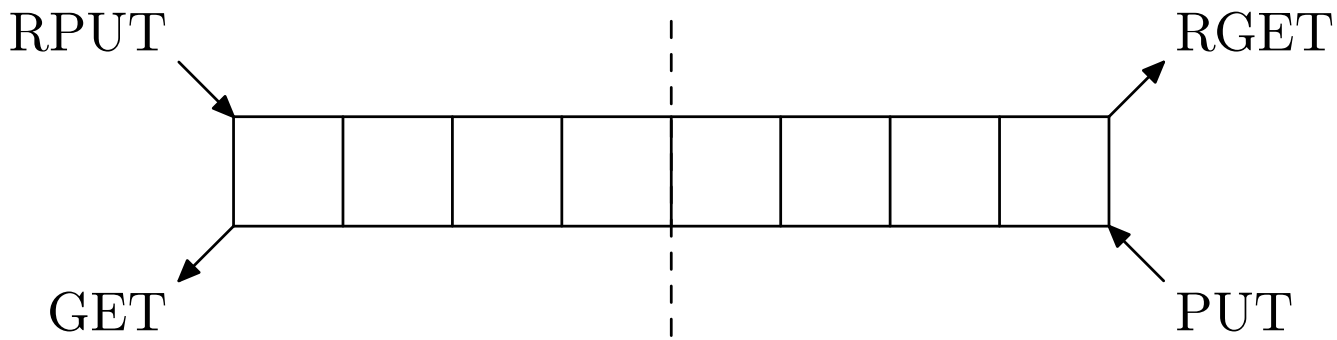


Elérni és feldolgozni az „első” mutató által hivatkozott elemet tudjuk, bővíteni pedig az „utolsó” mutató által hivatkozott elem mögé tudunk. A sor üres, ha mindkét segédmutató értéke NIL.

Kósa Márk
Pánovics János

Kétvégű sorok

- Kétvégű sor: a sor műveletei mellett további két művelet jelenik meg: RGET és RPUT (reverse GET és PUT).



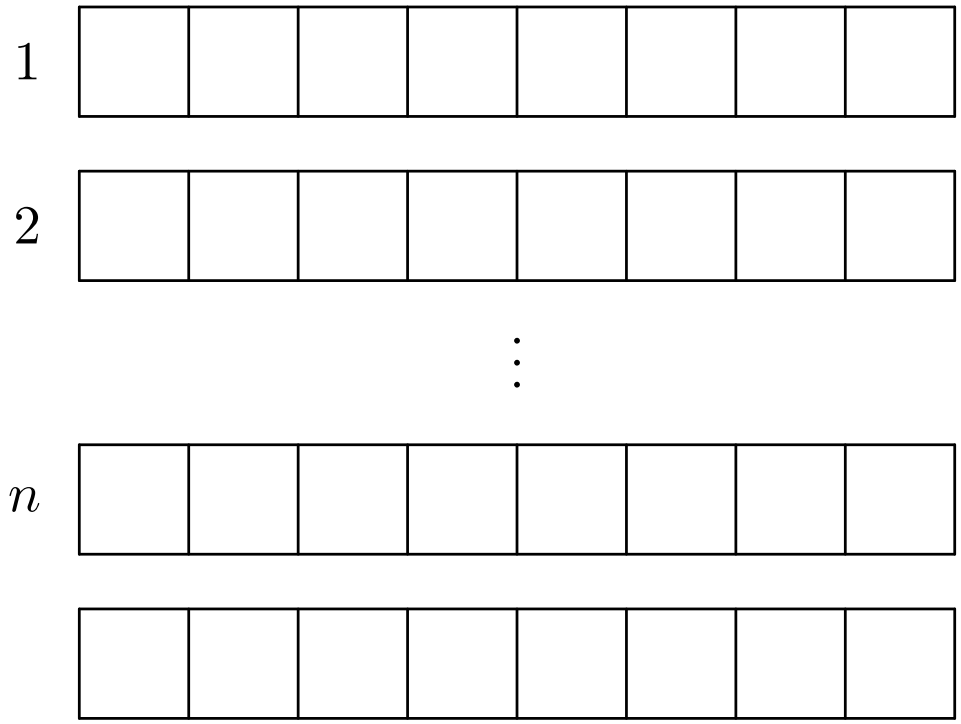
  Tekinthető két, az aljuknál összeragasztott veremnek.

- Inputkorlátozott kétvégű sor: nincs RPUT művelet.
- Outputkorlátozott kétvégű sor: nincs RGET művelet.

Kósa Márk
Pánovics János

Prioritásos sor

Olyan sor, amelyben az adatelemekhez prioritásértéket rendelünk, majd ezen értékek sorrendjében (azonos prioritású elemek esetén pedig továbbra is a bekerülés sorrendjében) dolgozzuk fel őket. Megvalósítása n különböző prioritásérték esetén $n+1$ (hagyományos) sorral történhet: minden prioritásértékhez tartozik egy-egy sor, a prioritás nélküli elemeket pedig külön sorban tároljuk:



Kósa Márk
Pánovics János