Verem

Statikus

Áll egy méretből, egy méret méretű tömbből, és egy integerrel, ami azt jelzi, hogy épp hol a head

- Konstriktor
  - head = 0
- Destruktor
  - üres
- Isempty
  - ha a head 0vége a bulinak
- push
  - ha a head nagyobb mint a méret akkor baj van
  - amúgy az array[head] legyen a beadott érték, és a headot növeljük eggyel
- top
  - ha nincs benne elem, nyilván exception
  - amúgy visszatér az array[head-1]-gyel
- pop
  - ha nincs benne elem, nyilván exception
  - létrehoz egy temp = array[head-1]-gyet, hhogy majd visszaadja a végén
  - amúgy csökkenti a headet eggyel
- print
  - 0-tól headig ciklus aztán kiírsz mindent

Dinamikus

Niylván van egy belső osztály, ami elvan magában, van egy értéke, meg egy nextje, illetve az egész veremnek van egy speciális node pointere a phead.

- Konstruktor
  - pHead legyen nullptr
- Destruktor
  - egyszerűen popolsz amíg üres nem lesz
- isEmpty()
  - ha pHead nullptr
- push()
  - Létrehozol egy új pointert ezzel az értékkel
  - ez nyilván arra fog mutatni ami eddig a head volt
  - majd a head pedig ő lesz
- top()
  - ha üres nyilván exception
  - aműgy egyszerűen return pHead->value
- pop
  - ha üres nyilván exception

    - int tmp = pHead->value, ez majd azért kell hogy törlés után visszatudjunk adni
    - majd létrehozunk egy phead_tempet a törléshet
    - a phead legyen egyenlő a nextjével
    - majd deleteljük az eddigi pheadot
- print
    - a fejtől egészen amíg nem érek egy olyantaghoz akinek a nextje nullptr kiirogatom az értékeket

Sor

Statikus

nyilván van benne megint egy méret, egy méret méretű tömb, és most kell egy head és egy tél integer, az egyik a sor elejét a másik a végét jelzi, illetve muszáj lesz egy empty bool változó még plusszba

- Konstruktor
    - empty = true
    - head = tail = 0
- Destruktor
    - üres
- isEmpty
    - csak simán visszadobja az empty értékét
- isFull
    - he nem isempty, és a head == tail
    - akkor tele
- in
    - Ha tele van, nyilván exception
    - ha üres, akkor változzon az empty falsera
    - az array[tail] legyen az amit beraktunk
    - majd ha a tail = max, akkor menjen vissza 0-ra, amúgy növelje a tailt eggyel
- out
    - ha üres nyilván exception
    - amúgy létrehozol egy tempet, ami a head értékét viszi (nyilvánvaló okokból)
    - majd fogod és ha a head az max akarna lenni 0 lesz, amúgy ++head
    - ha a head és az empty ezután egyenlő akkor legyen empty = true
    - a végén vissza adod a tempet
- first
    - ha üres nyilván exception
    - amúgy visszaadod az arrray[head]-et

Dinamikus

Hasonlóan mint a statikus vs nem statikus tömb

- Konstruktor

    - phead és ptail kezdetben nullptr
- Destruktor
    - amíg nem üres a lista addig kiveszem az elemeket
- isempty
    - ha phead == nullptr
- in
    - létrehozol egx nodeot ezzel az értékkel
    - ha eddig üres volt a lista akkor a phead és a ptail és erre a p nodera mutasson
    - amúgy a ptail-nak a nextje legyen p
    - majd a ptail legyen p
- out
    - ha tele van nyilván exception
    - létrehozol egy tempet a head értékével mert majd visszakell adni
    - illetve egy pointert ami a jelenlegi pheadre mutat
    - aztán a phead legyen a phead nextrje
    - majd töröld a p-t
    - és add vissza tempet
- first
    - simán visszaadod a a phead értékét

Kétszeresen láncolt lista

áll egy belső osztályból, aminek van értéke, nextje és prevje is. a lista tartalmaz egy pheadet, egy p tailt és egy p actot, ami az aktuális elemére mutató pointer

- konstruktor
    - mindhárom paraméter legyen nULL
- destruktor
    - amíg nem üres kiveszem az utolsót
- stepnext
    - if(act) ha az akt értelmes elem
    - akkor az akt mutasson az akt nextre
- stepprev
    - hasonló
- getvalue
    - ha üres a lista akkor exception
    - ha if(act), akkor retrun act->data
- setvalue
    - hasonló
- isempty
    - ha head egyenlő NULL
- islast
    - act == tail
- isfiers
    - hasonló

- isactnull
    - na ez is elég trú
- insert first
    - létrehozunk egy nodeot az értékkel
    - ha üres a lista, akkor
        - act head tail mind legyen p
    - amúgy pedig
        - ennek a nextje legyen a head
        - a head prevje legyen p
        - és a head legyen p
        - és az akt is legyen p
- insert last
    - ugyenaz logikusan csak a taillel játszol
- insert before
    - ezt is végig lehet gondolni
- insert after
    - same here
- remove first
    - ha üres akkor nincs értelme
    - ha a head ==tail akkor egy elem van, tehát törlöm a headet
    - majd az összes akt head tail mutató legyen null
    - minden más esetben a head nextje lesz head
    - majd a head prevet törlöm és legyen null ptr
    - az act legyen head
- többi is hasnan működik

Bináris keresőfa

van egy belső osztály, ami tartalmaz egy szülőt, balgyereket és jobb gyereket, és egy kulcsot
magának a fának az egyetlen adattagja a root

- konstruktor
    - legyen a root nullptr
- destruktor
    - destroy(root)
- destroy(x)
    - a destroy rekurzívan kitröli az alatta lévő csúcsokat
    - ha x nem nullptr, akkor meghívom a destroyt a lefjére meg a rightjára
    - majd delete x
- másoló konstruktor
    - copyof
        - ha amásolandó node nem üres node, akkor
        - akkor meghívjuk a x->left copyt a left-re és a x->right rightra
        - és visszadobom az xt a legvégén
    - ebből valahogy ki lehet hozni

- min - legkiesebb értékű csúcs az x gyökerű részfában
    - amíg az x->left nem nulla addig az x legyen x->left, aztán visszatérek az x-szel
- max - hasonlóan csak rightra
- next - viszaadja a rákövetkező elemet
    - hogyha az a jobb részfája nem nullptr akkor egyszerűáen abból a legkisebb elem lesz
    - ha nullptr, akkor létrehozunk egy pontert x papájának (y)
    - ha az y nullptr, akkor kész
    - ha az x balgyerek, akkor is kész, y a rákövetkező
    - amíg nem ez áll fen, megnézzük az y szülőjére ugyanezt, tehát x = y y = y->parent
    - a végén mindeképp visszdobjuk ypszit
- prev - hasonló elven
    - ha a balgyeeke nem nullptr, akkor aegyszerűen a legnagyobb elem legy a balrészfában
    - minden más esetben csinálunk az apukájának egy pontiert (y)
    - ha az y nullpinter kész
    - ha az x jobb gyerek nyilván y a rákövetkező
    - amíg ez nem áll fel, megnézzük y szülőjére, hogy mizu, ugyanúgy mint az előbb
    - végén mindenképp dobjuk yt
- size
    - amíg az x nem nullptr, visszadobjuk a a balrészfa és a jobbrészfa méretét + 1-et
- find(k)
    - létrehozunk egy segéd változót ami legyen a rootra mutató pinter, ajmd
    - amíg az x nem nullptr, vagy nem egyenlő k-val
        - megnézem, hogy a kulcsa x-nek nagyobb e mint k, ha k nagyobbb
            - megy jobbra
            - amúgy balra
    - végén visszaadod az x!=nullptr-t
- insert //Mindig leaf lesz
    - létrehozunk egy segéd változót ami legyen a rootra mutató pinter, ajmd
    - amíg az x nem nullptr
        - megnézem, hogy a kulcsa x-nek nagyobb e mint k, ha k nagyobbb
            - megy jobbra
            - amúgy balra
    - aztán az x legyen a beszúrandó node
    - x parentjének pedig megdelleően ő legyen jobb vagy balgyerek
- delete
    - az eddigiek alapján kikeresed a helyét
    - ha meg van, megnézed, hogy van e gyereke
    - ha nincs, csak simán kitörlöd
    - ha egy gyereke van, egyszerűen a szülőnek a pointere, mutasson a törlendő gyerekére

    - ha két gyereke van keresd ki a legkisebb elemet a jobb részfából, majd cseréld meg a kettőt, és töröld a levelet

AVL fa

nagyon hasonlít a bináris keresőfához, csak ugye van benne ez a kiegyensúlyozós cuccozás is, ami 95%ban abban nyilvánul meg hogy forgatni, kell ,ezért csomó mással most itt nem foglalkozom

- balra forgatás
    - y legyen x jobb gyereke
    - csekkold hogy y nem nullptr e;
    - x jobb gyereke y bal gyereke lesz
    - ha az y left nem nullptr, akkor legyen az y balgyerekének a szülője x
    - y apja legyen x apja
    - ha az x gyökere nullptr, akkor legyen a root y
    - ha x bal gyerek (x == x->parent->left)
        - legyen y is balgyereke ennek (x->parentnek)
    - minden más esetben y legyen a jobb gyereke a szülőnek
    - végül pedig állítsuk be az x - y szülő gyerek kapcsolatot
    - y->left = x;
    - x->parent = y;
- jobbra forgatás
    - tök ugyanaz csak kicsit át kell írni (végiggondolod)
- más itt szerintem nem kell mert túl HC lenne

PFF

hát itt is forgatás lehet még talán, mert rebalancolás az túl durva talán

- balra forgatás
    - y legyen x jobb gyereke
    - cskkolod hogy y nem empty_leaf e
    - x jobb gyereke y bal gyereke lesz
    - ha az y legt nem empty_leag, akkor legyen az y balgyerekének a szülője x
    - y apja legyen x apja
    - ha az x apja empty_leaf
    - BAZMEG SONTRA UGYANAZ CSAK AZ EGYIKBEN NULLPTR A MÁSIKBAN EMPTYLEAF VAN

Hashtábla

a hashtábla ugye úgy működik hogy kell bele egy mátrix tulajdonképpen, tehát egy listákból álló vektor, aminek indexei vannak, van egy kapacitása, hogy hány hashe lehet, és van egy actualsize-a, amiben azt tartjuk niylván, hogy jelenleg hány elem van belecsapva

- konstruktor
    - table(capacity) a tábla mérete legyen akkora mint a kapacitás
    - a kapacitás legyen a kapacitás
    - az aktuális méret pedig kezdetben nulla
- size
    - simán visszaadja az acctualsizeot
- empty
    - ha az actualsize==0
- clear
    - végig megyünk az összes elemen és mindet vleareljük
    - az aktuális méretet nullára állítom a végén
- insert
    - először a beadott keyre meghívom a hash függvényt, kapok egy intet ai mondjuk myhash
    - ezen a helyen kell lennie, tehát ezen az indexen megnézem benne van e a listában a valahol ez az elem, ha igen semmi
    - ha nem push_back
    - majd növeld eggyel a méretet
- find
    - ugyanígy máködik
- erase
    - törléshez dettó (eraset használsz)
- print
    - nagyon logikus
- gethash
    - hát ez picsa, de neked kell kitalálni
    - legegyszerűbb beadott érték % capacity

Rendezők

Buborék

A buborék rendezővel a legegyszerűbb az élet

- ezt szerintem összehozod magadtól

Maxsort

Ezzel is még elég egyszerű dolgod van : megkeresed a legnagyobbat : bekúrod a végére

- ez úgy működik, hogy először végig megyek a tömbön megkeresem a maximumot, megkeresem a leghátsó elemmel
- aztán a maradékból a maxot
- SZERINTEM EZT IS ÖSSZEHOZOD

Beszúró

na ez már télleg nehezebb egy fokkal, de szerintem ezt is össze tudnád hozni

Gyorsrendező

na ez picut hc-bb

- rekúrzívan működik tehát van három beneet egy tömb, egy alső index és egy felső index
    - amíg az alsó index télleg kisebb mint a felső
    - hozzunk létre egy q felosztást felosztó fv-ről később
    - majd quickosrt down-tól q-1-it
    - majd quicksort q+1-tól upig
- a felosztó fv úgy néz ki, hogy
    - kinevezünk egy pivotot A[down] pl
    - aztán a left index legyen down, a right legyen up
    - ciklust amíg left < right
        - ciklus amúg A[left] kisebb mint a pivot
            - növelem a legtet
        - a,míg A[right] nagyobb mint a picot
            - csökkentem rightot
        - ha a left < right
            - csere A[left] A right
    - végén az A[down] = A[right]
    - az A [right] pedig a pivot
    - majd dobb vissza rightot.

Kupac

a kupac olyan fa, amiben csak annyi a megkötés hogy a gyökér nagyobb mint a gyerekei, van benne egy fektor, amiben tárolom az elemeket

-