Rendezési algoritmusok

Felvételi felkészítő

2024.01.27.

Bevezető

Rendezési algoritmusok

Összehasonlítás

Áttekintés

- Bevezető
- Felvételi tematikában szereplő rendezési algoritmusok
  - Bemutatkozás - buborékrendezéssel (Bubble sort)
- Rendezési algoritmusok összehasonlítása
- Korábbi években adott (rendezéssel összefüggésbe hozható) feladatok megoldása

Egy algoritmussal kapcsolatos követelmények

helyesség

- a feladatot *helyesen* oldja meg minden megengedett bemeneti adatra
- lásd: sajátos esetek

hatékonyság

- minél kevesebb műveletet végezzen
- minél kevesebb tárhelyet igényeljen
- vegyük figyelembe a feladat sajátosságait

Rendezési algoritmusok

- Miben áll a rendezés feladata?
  Adott az (a1, a2, ..., an) sorozat.
  Rendezés: határozzuk meg a bemeneti sorozat egy olyan $(a1', a2', ..., an')$ permutációját, amelyben
  $a1' \leq a2' \leq ... \leq an'$.
- Mire használjuk?
- Mikor melyiket használjuk?

Összehasonlítási szempontok

- összehasonlítások száma
- mozgatások száma
- az algoritmus által igényelt tárhely mérete
- legjobb/legrosszabb/átlagos esetben

Rendezési algoritmusok (felvételi-tematika)

Összehasonlításon alapuló rendezések $O(n^2)$

- **buborékrendezés** (bubblesort) - sortare prin metoda bulelor
- **beszúró rendezés** (insert sort) - sortare prin inserție
- **kiválasztó rendezés** (select sort) - sortare prin selecție
- (összehasonlító) **(le)számláló rendezés** - sortare prin metoda numărării

Lineáris rendezés $O(n)$

- **ládarendezés/(le)számláló rendezés** (binsort/counting sort) - sortare prin metoda numărării

Oszd meg és uralkodj módszert alkalmazó rendezések $O(nlogn)$

- **összefésülő rendezés** (merge sort) - sortare prin interclasare
- **gyorsrendezés** (quicksort) - quicksort

Buborékrendezés (bubble sort)

működési elve:

- balról jobbra, páronként összehasonlítjuk az egymásmelletti elemeket, és ha a sorrend nem megfelelő, akkor felcseréljük az illető két elemet
- addig ismételjük a bejárást az elemek felcserélésével, amíg csere nélküli tömbvizsgálathoz jutunk

- bemutatkozás (név, honnan jelentkeztél be?, magasság (cm))

észrevételek

- ha végigvizsgáljuk a teljes sorozatot és a szükséges cseréket elvégeztük, akkor a **legnagyobb elem** biztosan az utolsó helyre kerül, második végigjáráskor helyre kerül az utolsó előtti elem, stb.
- vizsgáljuk meg a köv. fólián látható algoritmust: figyelembe vegyszi-e ezt az észrevételt?

Buborékrendezés (bubble sort)

```
Algoritmus buborékrendezés1(n, a)
{Bemeneti adatok: a - egészeket tartalmazó sorozat, n - sorozat mérete}
{Eredmény: a - rendezett sorozat}
  Ismételd
    rendben←igaz
    Minden i←1, n-1 végezd el:
      Ha a[i]>a[i+1] akkor
        x←a[i]
        a[i]←a[i+1]
        a[i+1]←x
        rendben←hamis
      Ha_vége
    Minden_vége
  ameddig rendben
Algoritmus_vége
```

észrevételek

- elég, ha a következő bejárás során csak az utolsó csere helyéig vizsgáljuk a sort (a köv. változat ezt is figyelembe veszi)

Buborékrendezés (bubble sort)

```
Algoritmus buborékrendezés2(n, a)
{Bemeneti adatok: a - egészeket tartalmazó sorozat, n - sorozat mérete}
{Eredmény: a - rendezett sorozat}
  k←n
  Ismételd
    nn←k-1
    rendben←igaz
    Minden i←1, nn végezd el:
      Ha a[i]>a[i+1] akkor
        rendben←hamis
        x←a[i]
        a[i]←a[i+1]
        a[i+1]←x
        k←i {legutóbbi csere helye}
      Ha_vége
    Minden_vége
  ameddig rendben
Algoritmus_vége
```

Buborékrendezés (bubble sort) - ugyanaz, csupán más jelölésekkel

```
Algorithm buborékrendezés2(n, a)
{Bemeneti adatok: a - egészeket tartalmazó sorozat, n - sorozat mérete}
{Eredmény: a - rendezett sorozat}
  k←n
  Repeat
    nn←k-1
    rendben←igaz
    For i←1, nn execute
      If a[i]>a[i+1] then
        rendben←hamis
        a[i]↔a[i+1]
        k←i {legutóbbi csere helye}
      EndIf
    EndFor
  until rendben
EndAlgorithm
```

Buborékrendezés (bubble sort)

Jellemzés:

- ha a sorozat már rendezett, az első végigjárás után véget ér az algoritmus (azaz legjobb esetben O(n) a bonyolultsága)
- legrosszabb és átlagos esetben is $O(n^2)$ lépést igényel
- a cserék száma is $O(n^2)$ nagyságrendű
- **helyben rendez** - ha a bemeneti adatokon kívül konstans méretű többlet-memóriát használ
- **stabil** (az egyenlő értékek sorrendjét megtartja)
- mivel műveletigénye igen nagy, nem nagyon használják

Beszúró rendezés (Insert sort)

működési elv

- kezdetben a tömb első elemét rendezettnek tekintjük.
- a második elemet összehasonlítjuk az elsővel, s ha nagyobb, akkor a második elem is a helyén van, különben beszúrjuk az első elé úgy, hogy a második elemet elmentjük egy segédváltozóba, az első elemet eltoljuk 1-el jobbra, majd az első helyre tesszük az elmentett értéket
- ekkor az első két elem rendezve van, következik a harmadik elem elhelyezése a rendezett részben, és így tovább.
- egy i sorszámú elem beszúrásakor eldöntjük, hogy nem-e nagyobb az $i-1$ sorszámú elemnél, mert ha igen, akkor marad a helyén.
- különben elmentjük egy segédváltozóba, s a rendezett részben az i-1 sorszámú elemtől kezdve eltoljuk jobbra az összes nála nagyobb számot, végül az utolsónak eltolt szám helyére bemásoljuk az elmentett értéket

feladat

- vezessük végig az algoritmust a 6 4 9 5 2 7 számsorra

Beszúró rendezés (insert sort) - hibás

```
Algoritmus beszúróRendezés(n, a)
{Bemeneti adatok: a - egészeket tartalmazó sorozat, n - sorozat mérete}
{Eredmény: a - rendezett sorozat}
   Minden j←2, n végezd el:
    segéd←a[j]
    i← j-1
    Amíg i > 0 és a[i]>segéd végezd el:
      a[i+1]←a[i]
      i←i-1
    Amíg_vége
    a[i]←segéd
  Minden_vége
Algoritmus_vége
```

- Találjuk meg az elírást!

Beszúró rendezés (insert sort) - helyesen

```
Algoritmus beszúróRendezés(n, a)
{Bemeneti adatok: a - egészeket tartalmazó sorozat, n - sorozat mérete}
{Eredmény: a - rendezett sorozat}
   Minden j←2, n végezd el:
    segéd←a[j]
    i← j-1
    Amíg i > 0 és a[i]>segéd végezd el:
      a[i+1]←a[i]
      i←i-1
    Amíg_vége
    a[i+1]←segéd
  Minden_vége
Algoritmus_vége
```

Beszúró rendezés (Insert sort)

Jellemzés

- hatékony algoritmus kis számú elem rendezésére
- egyik fontos előnye, hogy rendezés közben képes új elemeket is felvenni a rendezni kívánt számsor végére
- az összehasonlítások száma a bemeneti adatok rendezettségétől függ
- helyben rendez, stabil
- ha a sorozat növekvő sorrendben van, az összehasonlítások száma n-1, legjobb esetben tehát $O(n)$ bonyolultágú
- az algoritmus bonyolultsága $O(n^2)$ nagyságrendű, a gyakorlatban mégis hatékonyabb a többi $O(n^2)$-es rendezésnél

Kiválasztó rendezés (Select sort)

feladat

- a megadott algoritmus alapján rekonstruáljuk a működési elvét (azaz mit végez az alábbi algoritmus)

```
Algoritmus kiválasztóRendezés(n, a)
{Bemeneti adatok: a - egészeket tartalmazó sorozat, n - sorozat mérete}
{Eredmény: a - rendezett sorozat}
   Minden i←1, n-1 végezd el:
    indMin←i
    Minden j←i+1, n végezd el:
      Ha a[indMin] > a[j] akkor
          indMin←j
      Ha_vége
    Minden_vége
    a[i]↔a[indMin]
  Minden_vége
Algoritmus_vége
```

Kiválasztó rendezés (Select sort)

Működési elv:

- előbb megkeressük a számsor legkisebb elemét és felcseréljük az első számmal, majd eltekintünk az első helyen levő számtól és megkeressük a maradék számsorban a legkisebb elemet és felcseréljük a második számmal
- ezt az eljárást folytatva az utolsó lépésben az utolsó előtti helyről indul a minimumkiválasztás és a két utolsó közül a kisebb kerül az utolsó előtti helyre

feladat

- vezessük végig az algoritmust a
  9, 3, 6, 5, 1, 7, 2
  számsorra

Kiválasztó rendezés (Select sort)

Jellemzés

- az összehasonlítások száma legrosszabb, legjobb és átlag esetben is $O(n^2)$ nagyságrendű
- a cserék száma O(n) nagyságrendű
- helyben rendez
- stabil

Összehasonlító számláló rendezés

működési elv:

- a számsorozat minden eleme esetén megszámoljuk, hogy hány kisebb érték van az illető számnál, így megkapjuk ennek az elemnek sorszámát a rendezett sorozatban
- (ha egy vizsgált szám esetén k db. nála kisebb elem van, akkor ez a rendezett tömb $k+1$. helyére kerül)
- ezt viszont nem írhatjuk be az eredeti tömb $k+1$. helyére, mivel felülírnánk és elvesztenénk az ott tárolt értéket, szükségünk van egy *segédtömb* használatára, ebbe kerülnek a növekvő sorrendbe rendezett elemek

feladat:

- rendezzük számláló rendezéssel növekvő sorrendbe a következő számsort: 4, 9, 8 5, 6, 2
- mi történik, ha nem csak különböző értékeink vannak - hogyan oldjuk meg?
- módosítsuk a programot, hogy ilyen esetben is helyesen működjön

Összehasonlító számláló rendezés

```
Algoritmus összehasonlítóSzámlálóRendezés_1(a, n)
{Bemeneti adatok: a - egészeket tartalmazó tömb, n - a tömb mérete}
{Eredmények: b - rendezett tömb}
{Leírás: minden egyes tömbelemre megszámoljuk a nálánál kisebb elemek számát, így k-ban megkapjuk az a i. elemének b-beli sorszámát}
   Minden i←1,n végezd el
    k←1
    Minden j←1,n végezd el
      Ha a[i]>a[j] akkor
        k←k+1
      Ha_vége
    Minden_vége
    b[k]←a[i]
  Minden_vége
Algoritmus_vége
```

Összehasonlító számláló rendezés

```
Algoritmus összehasonlítóSzámlálóRendezés_2(a, n)
{Bemeneti adatok: a - egészeket tartalmazó tömb, n - a tömb mérete}
{Eredmények: b - rendezett tömb}
{Leírás: kevesebb összehasonlítás, nagyobb tárhely}
   Db[1..n]←1 {a tömb feltöltése 1-es értékekkel}
  Minden i←1,n-1 végezd el
    Minden j←i+1,n végezd el
      Ha a[i]>a[j]
        akkor Db[i]←Db[i]+1
        különben Db[j]←Db[j]+1
      Ha_vége
    Minden_vége
  Minden_vége
  Minden i←1,n végezd el
    b[Db[i]]←a[i]
  Minden_vége
Algoritmus_vége
```

Összehasonlító számláló rendezés

Jellemzés

- segédtömbre van szükség, így a helyfoglalás a memóriában $2 * n$ nagyságrendű
- bonyolultsága $O(n^2)$ nagyságrendű (legjobb, legrosszabb és átlagesetben is)
- stabil

- amennyiben a tömbértékek 1 és k közötti egész értékek, $O(n)$ időben megoldható a feladat (segédtömb használatával) - lineáris bonyolultság
- a segédtömbben megszámoljuk, hogy melyik elem hányszor szerepel $\rightarrow$ számláló rendezés/ládarendezés (Counting sort/binsort):
  - a segédtömb i. eleme azt tartja nyilván, hogy hány darab i-vel egyenlő elemet találtunk az eredeti tömbben.

Lásd:animáció (a stabil változat)

Számláló rendezés/ládarendezés (Counting sort/binsort)

```
Algoritmus számlálóRendezés(a, n, k)
{Bemeneti adatok: a - egészeket tartalmazó tömb, melynek elemei
1 és k közötti értékek, n - a tömb mérete}
{Eredmények: b - a tömb elemei rendezve}
  Minden i←1,k végezd el
    segéd[i]←0
  Minden_vége
  Minden j←1,n végezd el
    segéd[a[j]]← segéd[a[j]]+1 {hany i ertek van}
  Minden_vége
  q← 0
  Minden i←1,k végezd el
    Minden j←1,segéd[i] végezd el
      q←q+1
      a[q]←i
    Minden_vége
  Minden_vége
Algoritmus_vége
```

Számláló rendezés/ládarendezés (Counting sort/binsort) - **stabil** változat

ha a rendezéshez használt kulcsokhoz más adatok is kapcsolódnak

```
Algoritmus számlálóRendezés2(a, n, k)
{Bemeneti adatok: a - egészeket tartalmazó tömb, melynek elemei
1 és k közötti értékek, n - a tömb mérete}
{Eredmények: a - rendezett tömb}
  Minden i←1,k végezd el
    segéd[i]←0
  Minden_vége
  Minden j←1,n végezd el
    segéd[a[j]]← segéd[a[j]]+1 {hány i érték van}
  Minden_vége
  Minden i←2,k végezd el
    seged[i]←seged[i-1]+seged[i]
  Minden_vége
  Minden i←n,1,-1 végezd el
    b[seged[a[i]]]←a[i]
      seged[a[i]]←seged[a[i]]-1
  Minden_vége
Algoritmus_vége
```

$O(n \log n)$-es rendezési algoritmusok

Összefésüléses rendezés (Merge Sort)

- az **oszd meg és uralkodj** (divide et impera) elvet használó rendezés
- a számsort felosztjuk két részre, majd ezeket is felezzük mindaddig, amíg ezek hossza 1 lesz. Ekkor biztos, hogy az illető részsorozatok rendezve vannak, és ezeket a rendezett részsorozatokat kell **összefésülni** rekurzívan.
- **összefésülés**: két rendezett sorozatból előállítunk egy harmadik növekvő sorrendű sorozatot.

Összefésüléses rendezés

```
Alprogram rendez (x, bal, jobb)
  Ha bal<jobb akkor
      közép←(bal+jobb) / 2;
      rendez(x, bal, közép);
      rendez(x, közép+1, jobb);
      összefésül(x, bal, közép, jobb);
  Ha_vége
Alprogram_vége
```

ahol

- az `összefésül (x, bal, közép, jobb)` alprogram összefésüli az `x` tömb két rendezett részsorozatát:
  - az egyik az `x` *bal* indexű elemétől tart a középsőig
  - a másik pedig a középső utáni elemtől a *jobb*-ig
- a két rendezett részsorozatot átmásoljuk az `a` és `b` segédtömbökbe, és az `x`-be fogja összefésülni őket

Az `n` elemű `x` tömb rendezéséhez meghívjuk a `rendez(x, 1, n)` alprogramot.

Összefésüléses rendezés

```
Alprogram összefésül (x, bal, közép jobb)
  Minden i←bal,közép végezd el:
    a[i]←x[i]
  Minden_vége
  Minden i←közép+1,jobb végezd el:
    b[i]←x[i]
  Minden_vége
  a[közép+1]←végtelen
  b[jobb+1]←végtelen
  i←bal
  j←közép+1
  Minden k←bal,jobb végezd el:
    Ha a[i]<b[j] akkor
      x[k]←a[i] ; i←i+1
    különben
      x[k]←b[j] ; j←j+1
    Ha_vége
  Minden_vége
Alprogram_vége
```

Gyorsrendezés (Quicksort)

- szintén **oszd meg és uralkodj** elvet használja
- *strázsának* kiválaszt egy elemet (általában az elsőt) és úgy rendezi a többi elemet, hogy a nála kisebbek az illető elemtől balra kerüljenek, a nagyobbak pedig tőle jobbra.
- a strázsa elem ekkor a helyén van, tőle balra illetve jobbra két (egyelőre rendezetlen) részsorozatot kaptunk
- a kettéosztást tovább folytatjuk mindkét részsorozattal, amíg 1 elemű részsorozatokat kapunk
- a strázsa meghatározása kulcskérdés az algoritmus végrehajtása során
- gyakran az első elemet választják strázsának
- más megközelítés: véletlenszerű elem kiválasztása, amit felcserélünk az első elemmel

Gyorsrendezés (Quicksort)

```
Algoritmus quickSort (a, bal, jobb)
  Ha bal<jobb akkor
    feloszt(a, bal, jobb, m)
    quickSort(a, bal, m-1)
    quickSort(a, m+1, jobb)
  Ha_vége
Algoritmus_vége
```

- feloszt(a, bal, jobb, m): meghatározzuk azt az *m* helyet, ahol a sorozatot két részsorozatra bontjuk, miközben egy elem a végleges *m* helyére kerül
- algoritmus meghívása: quickSort(a,1,n)

Gyorsrendezés (Quicksort)

```
Algprogram feloszt (a, bal, jobb, m)
  strázsa←x[bal]
  i←bal-1
  j←jobb+1
  Ismételd
    Ismételd
      j←j-1
    ameddig a[j]≤strázsa
    Ismételd
      i←i+1
    ameddig a[i]≥strázsa
    Ha i<j akkor
      a[i]↔a[j]
    Ha_vége
  ameddig i≥j
  m ←j
Alprogram_vége
```

Rendezési algoritmusok összehasonlítása

- n tetszőleges elem összehasonlítás alapú rendezése leggyorsabban $O(nlogn)$ összehasonlítással végezhető el.
- az $O(n^2)$-es algoritmusok nagy bemenetre jóval lassabbak
- ennek ellenére az $O(n^2)$-es algoritmusok előnye:
  - egyszerűségük
  - rövid bemenetre gyorsabb futásidőt eredményeznek, mint az $O(nlogn)$-es társaik
- sajátos tulajdonságokkal rendelkező sorozat esetén megoldható a rendezés lineáris időben (lásd: ládarendezés)

Rendezési algoritmusok összehasonlítása

algoritmus	**legjobb**	**átlag**	**leg-rosszabb**
buborékrendezés (Bubble sort)	$O(n)$	$O(n^2)$	$O(n^2)$
kiválasztó rendezés (Selection sort)	$O(n^2)$	$O(n^2)$	$O(n^2)$
beszúró rendezés (Insertion sort)	$O(n)$	$O(n^2)$	$O(n^2)$
gyorsrendezés (Quicksort)	$O(n\log n)$	$O(n\log n)$	$O(n^2)$
összefésülő rendezés (Merge sort)	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$

animáció + leírás

- rendezési algoritmusok vizuális összehasonlítása, különböző típusú bemenetekre
- néhány vizuálisan megjelenített rendezési algoritmus

Látványos/hangzatos/vicces rendezés

Eltáncolt rendezési algoritmusok

- buborékrendezés – csángó tánc
- beszúró rendezés – román néptánc
- kiválasztó rendezés – cigány tánc
- gyorsrendezés (Quicksort) – Küküllő menti legényes
- összefésüléses rendezés (Merge sort) – szász tánc

más eltáncolt rendezési algoritmusok:

- kagyló-rendezés (Shell sort) – székely tánc

Vizualizáció+hangzás (pl. nagyobb érték, magasabb hang)

- egy órányi rendezési algoritmusok által generált “zene”

Vicces rendezési algoritmusok (angol nyelven)

- Strangest sorting algorithms

Feladatok

1. Generáljunk egy egészeket tartalmazó n elemű tömböt és rendezzük az elemeket úgy, hogy elöl legyenek a páros számok növekvő sorrendben, majd a páratlanok csökkenő sorrendben.
2. Olvassunk be egy 12 elemű tömböt, mely egy bolt havi bevételeit jelképezi. Írassuk ki csökkenő sorrendben, hogy melyik a legjobb hat hónap. Mely hónapok bevétele kisebb az átlagbevételnél?
3. Egy csupa különböző egészekből álló sorozat bitonikus, ha először nő, utána pedig csökken, vagy fordítva: először csökken, utána nő. Például az (1; 3; 7; 21; 12; 9; 5), (9; 7; 5; 4; 6; 8) és (1; 2; 3; 4; 5) sorozatok bitonikusak. Adjunk rendező algoritmust n elemű bitonikus sorozatok rendezésére!