SDA – Seminar 5 - Ansamblu

- **Conținut**: Materialul curent exemplifică aplicabilitatea structurii de date **Ansamblu** (*Heap*) pentru rezolvarea unei probleme, prezentându-se și comparându-se din perspectiva eficienței timp soluții multiple.

Enunțul problemei

Determinați suma celor mai mari *k* elemente dintr-un vector conținând *n* elemente numerice distincte.

Exemplu

De exemplu, dacă vectorul conține următoarele 10 elemente: [6, 12, 91, 9, 3, 5, 25, 81, 11, 23] și k = 3, rezultatul așteptat este: 91 + 81 + 25 = 197.

Soluții posibile

I. Determinând maximul de k ori

Observații:

- Dacă, pentru exemplul considerat, aplicăm funcția de determinare a maximului de 3 ori, aceasta va returna 91 de fiecare dată. Așadar, este necesară impunerea unei limite superioare impuse maximului calculat, căutându-se astfel elementul maxim care este mai mic decât limita superioară impusă.
- Generalizând, începând cu cel de-al doilea pas, limita superioară impusă este maximul determinat la pasul anterior. La primul pas se poate alege ca limită o valoare mai mare decât marginea superioară a domeniului valorilor posibile.

- **Complexitatea** acestei soluții este: **Θ(k*n)** – găsirea maximului are loc în Θ(n) și se repetă de k ori.

II. Sortarea șirului în ordine descrescătoare și însumarea primelor k elemente.

- Sortarea poate fi efectuată în **$\Theta(n*\log_2 n)$** (folosind sortarea prin interclasare sau *HeapSort*) (OBS: dacă domeniul valorilor șirului ar respecta condițiile necesare aplicării *BucketSort,* s-ar putea efectua în timp liniar, însă neavând această asigurare, nu o vom presupune)
  - BC - HeapSort: **$\Theta(n*\log_2 n)$** daca toate elementele sunt distincte

III. Prin adăugarea tuturor celor n elemente într-un ansamblu binar maximal (*binary max-heap),* urmată de eliminarea și însumarea primelor k (celor mai mari) elemente.

Exemplu:

Ansamblu:

creeaza(“>=”)

adauga(Ansamblu, Intreg)

sterge(Ansamblu): Intreg

```
Funcție sumaCelorMaiMariK(sir, n, k):
//sir este un vector (tablou unidimensional) de numere întregi distincte
//n este dimensiunea vectorului sir (adică numărul de elemente pe care acesta le conține)
//k este numărul de elemente pe care dorim să le însumăm, verificându-se inegalitatea k <= n

        creeaza(a, “≥”) //presupunând că avem o structură de date Ansamblu implementată, a fiind o instanță a ei, inițializăm relația impusă ansamblului cu “≥”, obținând un ansamblu maximal (max-heap)
        Pentru i ← 1, n execută //fiecare dintre cele n elemente din șir
                adaugă(a, sir[i]) //este adăugat în ansamblu
        SfPentru
        suma ← 0 //inițializăm suma
        Pentru i ← 1, k execută //de k ori
                elem ← șterge(a) //eliminăm maximul curent din ansamblu
                suma ← suma + elem //și îl adăugăm sumei
        SfPentru
        sumaCelorMaiMariK ← suma //stabilim rezultatul algoritmului (funcției)
SfFuncție
```

Care este **complexitatea adăugării** unui element într-un ansamblu cu **m elemente**?

- Complexitatea adăugării unui element într-un ansamblu cu m elemente este **O($\log_2$m)**.

Care este **complexitatea ștergerii** unui element dintr-un ansamblu cu **m elemente**?

- Complexitatea eliminării unui element dintr-un ansamblu cu m elemente este **O($\log_2$m).**

Care este **numărul maxim de elemente conținut de ansamblul în care adăugăm / din care ștergem elemente**?

- Ansamblul în care adăugăm și din care ștergem elemente conține cel mult **n** elemente (anterior primei ștergeri conține exact n elemente).

Câte adăugări se efectuează? Dar ștergeri?

- Se efectuează **n adăugări** (fiecare dintre cele n elemente fiind adăugat în ansamblu) și **k ștergeri** (sau, eventual, **k-1 ștergeri + accesarea maximului** dintre elementele rămase în ansamblu și adăugarea lui sumei).

Care este complexitatea totală obținută?

- Obținem complexitatea totală: **O(n*log$_2$n) + O(k*log$_2$n)**. Având în vedere că n >= k, se obține **O(n*log$_2$n)**.

Putem reduce complexitatea soluției anterioare, O(n*log$_2$n), printr-o soluție mai eficientă ?

IV. Observăm nu avem nevoie de toate cele n elemente în ansamblu, întrucât ne interesează doar cele mai mari k.

- Dacă reconsiderăm exemplul (șirul este [6, 12, 91, 9, 3, 5, 25, 81, 11, 23] și k = 3), **putem păstra în ansamblu, la fiecare pas, doar cele mai mari k elemente** întâlnite până la momentul curent, procedând după cum urmează:
- Inițial adăugăm primele k elemente în ansamblu. Deci, pentru exemplul considerat, în momentul în care ajungem cu parcurgerea șirului la elementul 9, ansamblul conține 6, 12 și 91.
- Când întâlnim 9, putem renunța la 6, știind sigur că nu va face parte din cele mai mari 3 numere, întrucât avem deja 3 numere mai mari decât acesta. Așadar, păstrăm 12, 91 și 9.
- Când întâlnim 3, știm că 3 nu va fi printre cele mai mari 3 numere, întrucât avem deja 3 elemente mai mari decât 3 în ansamblu. Conținutul ansamblului rămâne, în continuare: 12, 91 și 9. Similar pentru elementul 5.
- Când întâlnim 25, putem renunța la 9, continuând cu 12, 91 și 25.

Ce se întâmplă la următorii 3 pași și care vor fi cele 3 elemente conținute în final de ansamblu?

- Când întâlnim 81, putem renunța la 12, continuând cu 91, 25 și 81.
- Când întâlnim 11, continuăm cu 91, 25 și 81.
- Când întâlnim 23, păstrăm în ansamblu 91, 25 și 81.
- Astfel, ansamblul conține la finalul parcurgerii șirului elementele 91, 25 și 81, care sunt cele mai mari 3 elemente din întreg șirul.

La ce se reduce rezolvarea problemei ulterior parcurgerii întregului șir și actualizării corespunzătoare a conținutul ansamblului?

- Ulterior parcurgerii șirului și construirii ansamblului format din cele mai mari k elemente, rezolvarea problemei se reduce la însumarea elementelor componente ale ansamblului.

Ansamblul în care păstrăm cele mai mari k elemente din șir ar trebui să fie un **ansamblu maximal (*max-heap*) sau unul minimal (*min-heap*)?**

- Având în ansamblu, la un moment dat, cele mai mari k elemente anterioare elementului curent , suntem interesați de elementul minim dintre acestea, astfel încât să îl comparăm cu elementul curent. Dacă elementul curent este mai mare decât minimul elementelor din ansamblu, atunci îl va înlocui. Altfel, conținutul ansamblului rămâne nemodificat (știind că toate elementele din ansamblu sunt mai mari decât elementul curent). Prin urmare, avem nevoie de un **ansamblu minimal (*min-heap*).**

Ansamblu:

creeaza(“>=”)
adauga(Ansamblu, Intreg)
sterge(Ansamblu): Intreg
prim(Ansamblu): Intreg

```
Functie sumaCelorMaiMariK_2(sir, n, k):
//sir este un vector (tablou unidimensional) de numere întregi distincte
//n este dimensiunea vectorului sir (adică numărul de elemente pe care acesta le conține)
//k este numărul de elemente pe care dorim să le însumăm, verificându-se inegalitatea k <= n
```

```
     init(a, "≤")  //presupunând că avem o structură de date Ansamblu implementată, a fiind o instanță a ei, inițializăm relația impusă ansamblului cu "≤", obținând un ansamblu minimal(min-heap)
      Pentru i ← 1, k execută //primele k elemente ale șirului sunt adăugate în ansamblu în mod implicit
            adaugă(a, șir[i])
      SfPentru
      Pentru i ← k+1, n execută //fiecare element rămas în șir va fi comparat cu elementul minim din ansamblu
            Dacă sir[i] > prim(a) atunci //dacă este mai mare decât elementul minim din ansamblu, returnat de funcția prim, atunci
                  șterge(a) //, elementul minim din ansamblu este eliminat
                  adaugă(a, șir[i]) //, iar elementul curent este adăugat în ansamblu
            SfDacă
      //calculăm suma celor k elemente conținute de ansamblu la finalul parcurgerii șirului
    SfPentru
    Pentru i ← 1, k execută //de k ori
                        elem ← șterge(a)  //eliminăm minimul curent din ansamblu
                        suma ← suma + elem //și îl adăugăm sumei
    SfPentru
    sumaCelorMaiMariK_2 ← suma
SfSFuncție
```

Care este **numărul maxim de elemente conținut de ansamblul în** care adăugăm / din care ștergem elemente?

- Ansamblul în care adăugăm și din care ștergem elemente conține cel mult **k** elemente.

Câte sunt complexitățile operațiilor de adăugare și ștergere ?

- Adăugările și ștergerile se efectuează în **$O(\log_2 k)$**.

Câte adăugări și câte ștergeri se efectuează?

- Adăugăm și ștergem de cel mult **n** ori (de pildă, dacă șirul este ordonat crescător).
  - ➢ Toate cele n elemente ale șirului vor fi adăugate în ansamblu
  - ➢ Primele n-k elemente vor fi eliminate din ansamblu până la construirea ansamblului final, odată cu parcurgerea șirului, restul de k elemente fiind eliminate pentru calculul sumei

Care este complexitatea totală obținută?

- Obținem complexitatea totală: **O(n*log$_2$k)**.

Observații:

Dacă am lucra direct pe reprezentare, în ultima structură repetitivă *Pentru* în care eliminăm elemente din ansamblu și le însumăm, am putea doar să însumăm elementele din ansamblu. În acest caz, clasa de complexitate nu se schimbă, dar numărul efectiv de operații elementare efectuate se reduce.

V. Ne amintim (din Cursul 7) că putem să **transformăm șirul într-un ansamblu** în O(n). Prin urmare, putem proceda similar soluției III, dar în loc să adăugăm elementele unul câte unul în ansamblu, putem să transformăm șirul într-un ansamblu maximal **și apoi să eliminăm k elemente din el**.

Care este complexitatea acestei soluții?

- Complexitatea acestei soluții este: **O(n + k*log$_2$n)**.

Problema 2:

Găsiți un algoritm O(n*log2k) pentru a interclasa k liste ordonate, unde n este numărul total de elemente din listele de intrare.

Reprezentarea listelor este ascunsă, acestea se parcurg folosind iteratori.

Indicație

- generalizăm ideea de la interclasarea a două liste ordonate
- în fiecare dintre cele k liste, avem un element curent (indicat de un iterator)

a) - pentru a extrage minimul/maximul dintre cele k liste, folosim un ansamblu O(log2k)

Ansamblul va conține maxim *k* elemente în orice moment, *k* fiind numărul de liste, $1 \le k \le m$ => înălțimea ansamblului e $O(\log_2 k)$ **Observație:** Ansamblul va conține k elemente atâta timp cât mai sunt elemente de parcurs în fiecare dintre cele k liste care se interclasează. Pe măsură ce listele se epuizează (parcurgerea lor, în cadrul interclasării, se finalizează), numărul de elemente memorate în ansamblu scade.

- în lista din care a fost găsit minimul/maxim, deplasăm iteratorul
- elementul indicat de iterator, îl adăugăm în heap

b) repetăm pasul a) de n ori (pentru numărul de elemente din toate listele) => O(nlog2k)

Așadar, deducem complexitatea interclasării ca fiind:

- $O(n \log_2 k)$, dacă $k > 1$
- $\Theta(n)$, dacă $k = 1$ (interclasarea se reduce la copierea singurei liste)

LO:
iterator(LO, IteratorLO)
vida(LO): A/F
Lista:
creeaza(Lista)
adaugaFinal(Lista, TElement)
Ansamblu:
creeaza(Relatie, Ansamblu)
adauga(Ansamblu, TComparabil)
sterge(Ansamblu): TComparabil
prim(Ansamblu): TComparabil
IteratorLO:
element(IteratorLO): TComparabil
urmator(IteratorLO)
valid(IteratorLO): A/F

```
Pre: LO_j: LO, k:Întreg, k > 0, !vida(LO_j), R: Relație: IteratorLO x IteratorLO -> {A,F}
Post: rez: Lista, rez = lista obținută prin interclasarea LO_j, j = 1,k
Subalgoritm interclaseaza(LO_j, n, k, R, rez), j = 1,k
      creează(R,ans) //se creează ansamblul
      Pentru i ← 1,k executa //pentru fiecare dintre cele k liste de interclasat
            iterator(LO_i,it) //se creează un iterator
            adaugă(ans,it) //și se adaugă în ansamblu
      SfarșitPentru
      Pentru i ← 1,n execută //de un număr de ori egal cu numărul total de elemente din cele k liste
            it ← șterge(ans) //se șterge din ansamblu, deci iteratorul referind elementul minim curent
            elem ← element(it) //se obține elementul referit de iterator
            adaugăFinal(rez,elem) //și se adaugă la finalul listei rezultat
            următor(it)//se avansează în lista din care s-a selectat minimul curent
            Dacă valid(it) atunci //, iar dacă iteratorul rămâne valid (adică nu s-a epuizat lista)
                  adaugă(ans,it) //se adaugă iteratorul referind următorul element în ansamblu
            SfarsitDacă
      SfarsitPentru
SfârșitSubalgoritm
```