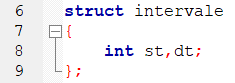
**PROBLEMA 4**

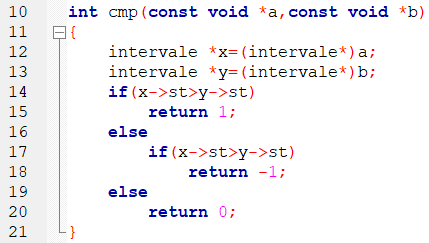
Date de intrare

* i -pentru parcurgerea tabloului unidimensional
* j -pentru parcurgerea tabloului unidimensional atunci când efectuăm ștergerea
* v-tabloul unidimensional care memorează capetele intervalelor cu ajutorul unei structuri denumită „intervale”
* n-numărul de intervale care trebuie reunite

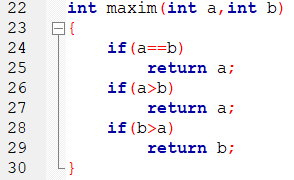
Rezolvarea problemei este efectuată folosind următoarele funcții:



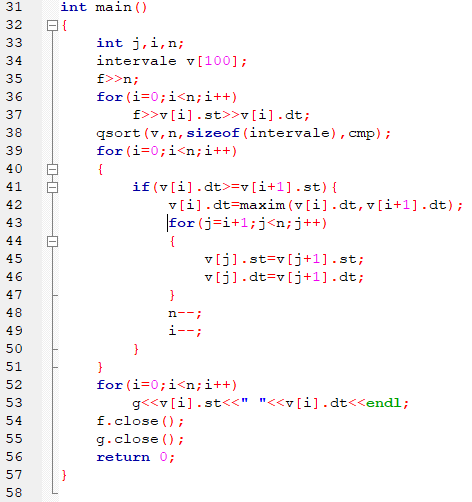
* structura „intervale” este definită pentru a ne ajuta să reținem capătul din stânga respectiv cel din dreapta al unui interval



* funcția „cmp” ne va ajuta pentru sortarea elementelor ,ea funcționează astfel:
  + are ca parametrii două variabile a,b de tip void care vor fi convertite la liniile 12 și 13 în tipul de date pentru structura definită anterior.
  + pentru a ordona intervalele crescător în funcție de capătul din stnga al intervalului vom compara care este cel mai mare și vom returna astfel : 1-dacă x->st >y->st ; (-1)- dacă x->st >y->st ; 0-în caz contrar.
  + funcția este creată pentru a ne ajuta mai târziu în folosirea funcției quicksort.



* funcția „maxim” determină maximul dintre 2 variabile de tip int ,a și b.



* funcția main:
  + în primă fază ea va citii n și capetele intervalelor la liniile 35-37.
  + la linia 38 funcția qsort este folosită pentru a ordona crescător elementele în funcție de capătul din stnga al intervalului el având însuși o complexitate O(n\*log[n]).
  + facem parcurgerea elementelor cu ajutorul for-ului de la linia 39
  + comparăm dacă extremitatea dreaptă a intervalului curent este mai mare sau egală cu extremitatea stângă a intervalului următor pentru a verifica daca cele 2 intervale pot fi reunite sau nu ,iar dacă această condiție este adevarată vom actualiza extremitatea dreaptă a intervalului curent calculând maximul dintre extremitatea dreaptă curentă și extremitatea dreaptă următoare folosind funcția „maxim” precizată anterior.Apoi ștergem următorul element pentru că deja am făcut reuniunea celor 2 intervale iar intervalul următor devine inutil.
  + ștergerea este efectuată astfel:
    - parcurgem elementele de la poziția i+1 pana la n-1 folosind un for
    - mutăm toate elementele începând cu i+1 spre stânga
    - scădem numărul de elemente doarece am șters un element
    - pe i îl scădem cu o unitate pentru a verifica dacă un interval din urmă a rămas neverificat.
  + în cele din urmă afișăm vectorul nostru care are toate intervalele reunite.

Complexitatea problemei:

|  |  |  |
| --- | --- | --- |
| Operație | Cost | Nr. repetări |
| 1(citire) | c1 | n |
| 2(qsort) | c2 | n\*log[n] |
| 3(parcurgere și ștergere elemente) | c3 | n\*n |
| 4(afișare) | c5 | n |

* după tabelul de complexitate putem observa că avem o complexitate o(n+ n\*log[n]+ n\*n+n),iar cel mai nefavorabil caz este cel de la operația 3 deci putem aproxima complexitatea la O(n2).

Exemple date de intrare și de ieșire:

|  |  |
| --- | --- |
| Date de intrare | Date de ieșire |
|  |  |
|  |  |

Rezolvarea problemei cu ajutorul tehnicii de programare numită „Greedy” constă în faptul că ea caută cea mai ușoară metodă prin ordonarea intervalelor și apoi compararea lor pe rând apoi actualizând capetele intervalelor în funcție de reuniunea dorită.