7. Tabele

7.1. TDA Tabelă

- Noţiunea de **tabelă** este foarte cunoscută, fiecare dintre noi consultând tabele cu diverse ocazii.

- Ceea ce interesează însă în acest context se referă la evidenţierea acelor caracteristici ale tabelelor care definesc **tipul de date abstract tabelă**.

- În figura 7.1.a apare un exemplu de tabelă în două variante.

Nume Prenume	**An**	**Medie**
Antonescu Ion	3	7,89
Bărulescu Petre	2	9,20
Card Gheorghe	5	9,80
Mare Vasile	2	8,33
Suciu Horia	1	9,60

Nume Prenume	**An**	**Medie**
Suciu Horia	1	9,60
Bărulescu Petre	2	9,20
Mare Vasile	2	8,33
Antonescu Ion	3	7,89
Card Gheorghe	5	9.80

Fig.7.1.a. Exemple de tabele

- După cum se observă, tabela din figură, ca şi marea majoritate a tabelelor, este alcătuită din **articole**.

    - Acest lucru **nu** este însă obligatoriu, deoarece spre exemplu prima coloană a acestei tabele poate fi considerată la rândul ei o **tabelă** (tabela studenţilor înscrişi).

    - În prima variantă studenţii sunt aranjaţi în **ordine alfabetică**, în cea de-a doua în **ordinea crescătoare** a **anilor** de studiu.

- Se spune că **tabela** este **ordonată** după o "**cheie**" element care facilitează regăsirea informaţiilor pe care le conţine.

    - Astfel, o "**cheie**" este un câmp sau o parte componentă a unui câmp care **identifică** în mod unic o **intrare** în tabelă.

- **Nu** este necesar ca tabela să fie **sortată** după o cheie însă o astfel de sortare simplifică **căutarea în tabelă**.

- Dintre operaţiile frecvente care se execută asupra **tabelelor** se menţionează:

  - (1) **Căutarea** în tabelă în vederea localizării informaţiilor care satisfac una sau mai multe condiţii.

  - (2) **Prelucrarea integrală** a tabelei într-un proces de "**traversare**" a acesteia.

    - Prelucrarea unei anumite intrări a tabelei în cadrul procesului de traversare se numeşte **vizitare** ("**visiting**").

    - Vizitarea poate include toate genurile de prelucrări începând de la **afişare** şi terminând cu **modificarea** conţinutului intrării.

- În definitiv, **TDA Tabelă** poate fi descris în mod sintetic după cum urmează [7.1.a].

TDA Tabelă

Modelul Matematic: O secvenţă finită de elemente. Fiecare element are cel puţin o cheie care identifică în mod unic intrarea în tabelă şi care este un câmp sau un subcâmp al elementului.

Notaţii:

```
TipElement - tipul elementelor tabelei.
TipCheie       - tipul cheii.
t: TipTabela;
e: TipElement;
k: TipCheie;
b: boolean.                                    [7.1.a]
```

Operatori:

1. ***CreazăTabelaVidă(t: TipTabela);*** - operator care face tabela *t* vidă.

2. ***TabelăVidă(t: TipTabela): boolean;*** - operator boolean care returnează **true** dacă tabela *t* este goală.

3. ***TabelăPlină(t: TipTabela): boolean;*** - operator boolean care returnează **true** dacă tabela e plină.

4. ***CheieElemTabelă(e: TipElement): TipCheie;*** -operator care returnează cheia elementului *e*.

5. ***CautăCheie(t: TipTabela,k: TipCheie): boolean;*** - operator care returnează **true** dacă cheia *k* se găseşte în tabela *t*.

6. ***InserElemTabelă***`(t: TipTabela,e: TipElement);` -operator care inserează elementul *e* în tabela *t*. Se presupune că *e* nu există în tabelă.

7. ***SuprimElemTabelă***`(t: TipTabela,k: TipCheie);` - operator care suprimă din *t* elementul cu cheia *k*. Se presupune că există un astfel de element în *t*.

8. ***FurnizeazăElemTabelă***`(t: TipTabela,k: TipCheie): TipElement;` - operator care returnează elementul cu cheia *k*. Se presupune că elementul aparţine lui *t*.

9. ***TraverseazăTabelă***`(t: TipTabela, Vizitare (Listă Argumente))` – operator care execută procedura *Vizitare* pentru fiecare element al tabelei *t*.

- În anumite implementări este convenabil să se cunoască **numărul curent** de elemente conţinute în tabelă.
    - În acest scop se poate asocia tabelei un **contor de elemente** şi un operator care-l furnizează, spre exemplu, ***DimensiuneTabelă***`(t:TipTabela): TipContor`.
    - **Contorul** se iniţializează la crearea tabelei şi se actualizează ori de câte ori se realizează inserţii sau suprimări în tabelă.
- În unele aplicaţii este convenabil de asemenea să se definească operatorul ***Actualizare***`(t: TipTabela, e: TipElement)`.
    - Operatorul realizează **actualizarea** acelei intrări din **tabelă** care conţine elementul *e*.
    - În termenii operatorilor anterior definiţi, actualizarea poate fi implementată astfel [7.1.b]:

{TDA Tabelă - operatorul *Actualizare*}

SuprimElemTabelă`(t,CheieElemTabelă(e));`

InserElemTabelă`(t,e)` [7.1.b]

- **Structura tabelă** este strâns legată de **bazele de date relaţionale**.
    - De fapt **structura tabelă** este **componenta principală** a unei **baze de date relaţionale.**
- O **bază de date** în general, este o colecţie de date integrate, destinată unei mari varietăţi de aplicaţii, memorată într-o manieră eficientă.
- O **bază de date relaţională** este formată din mai multe **tabele** numite **relaţii**.

- Asupra acestor tabele pot fi realizate **prelucrări** extrem de diverse şi de fapt prin intermediul lor se realizează accesul eficient la elementele bazei de date.

7.2. Tehnici de implementare a TDA Tabelă

- Există în principiu mai multe posibilităţi de **implementare** a **tipului de date abstract tabelă.**
    - Este vorba despre **tablouri**, **liste înlănţuite**, **tabele de dispersie**, **arbori** etc.
- Dintre aceste posibilităţi în cadrul paragrafului de faţă vor fi analizate cele bazate pe **structurile tablou** şi **listă înlănţuită**.
- Implementarea bazată pe **tehnica dispersiei** face obiectul secţiunii următoare iar cea bazată pe **arbori binari** va fi tratată ulterior.
- În general **listele înlănţuite** sau **tablourile**, ambele în variantă **ordonată** sau **neordonată** se constituie în **4 metode** posibile de implementare a tabelelor.
    - În continuare se va realiza o **analiză comparată** a acestor **metode**, detaliile de implementare fiind lăsate ca exerciţiu.
    - Aceasta **analiză comparată** are un caracter de **generalitate** care se răsfrânge şi asupra altor tipuri de date abstracte în a căror implementare sunt implicate structurile **tablou** respectiv **listă înlănţuită**.
    - În esenţă, în fiecare dintre cele 4 situaţii vor fi evidenţiate **avantajele**, **dezavantajele**, **concluziile** şi **recomandările** referitoare la utilizare.
- Observaţia **fundamentală** de la care porneşte orice implementare a unui **TDA** este aceea că, **particularităţile aplicaţiei** în care este el utilizat, **determină** cea mai potrivită **metodă de implementare**
- **Particularităţile** unei aplicaţii derivă direct din **natura** şi din **dinamica operaţiilor** efectuate asupra structurii de date asociate.

7.2.1. Implementarea TDA Tabelă cu ajutorul tabourilor ordonate

- **Avantaje:**
    - Posibilitatea utilizării tehnicii **căutării binare** *(*O*(log*$_2$ *n))*.
        - Acest avantaj se răsfrânge asupra **fazei de căutare** a cheii în operaţiile de inserţie, suprimare, furnizare element şi căutare cheie.
    - **Furnizarea** informaţiei şi **căutarea cheii** este performantă O*(log* $_2$ *n)*.

        - De fapt furnizarea constă din doi timpi: **faza de căutare** *(O(log 2 n))* şi din **faza de returnare** O*(1)*.

    - **Traversarea ordonată** a cheilor este liniară (O*(n)*).

- **Dezavantaje:**

    - **Faza de instalare** la **inserţie** este lentă (O*(n)*).

        - Necesită mutarea **tuturor intrărilor** începând cu punctul de inserţie pentru a face loc în tabelă.

    - **Inserţia** este lentă (O*(n)*).

        - Constă din **faza de căutare** care este rapidă (O*(log2 n)*) şi din **faza de instalare** care realizează inserţia propriu-zisă (O*(n)*).

    - **Faza de extragere** la **suprimare** este lentă (O*(n)*).

        - Necesită mutarea **tuturor intrărilor** începând cu locul de suprimare.

    - **Suprimarea** este lentă (O*(n)*).

        - Ea constă din **faza de căutare** (O*(log2 n)*) şi din **faza de extragere** (O*(n)*).

    - **Crearea** tabelei lentă (O*(n^2)*).

        - Pentru fiecare din cele n elemente se realizează o inserţie O*(n).*

    - Trebuie cunoscut apriori **numărul total** de intrări în tabelă.

        - Operatorul ***`TabelăPlină`*** trebuie executat înaintea fiecărei inserţii.

- **Concluzii:**

    - Utilizarea acestei implementări a tabelelor este **avantajoasă** atunci când:

        - Se realizează multe consultări, verificări sau rapoarte asupra tabelei.

        - Numărul de inserţii şi suprimări este relativ redus.

        - Tabela este de mari dimensiuni.

    - Trebuie cunoscută cu bună precizie apriori **dimensiunea maximă** a **tabelei**.

7.2.2. Implementarea TDA Tabelă cu ajutorul tablourilor neordonate

- **Avantaje:**
    - **Inserţia** este rapidă dacă se realizează la sfârşitul tabloului (O*(1)*).
    - **Faza de extragere** la **suprimare** este rapidă, dacă se mută ultima intrare a tabelei peste cea suprimată (O*(1)*).
    - **Construcţia** tabelei este rapidă (O*(n)*).
- **Dezavantaje:**
    - **Căutarea** se realizează **secvenţial** O*(n)*.
        - Acest dezavantaj se reflectă asupra suprimării, furnizării şi căutării cheilor.
    - **Suprimarea** este lentă (O*(n)*).
        - Ea constă din **faza de căutare** (O*(n)*) urmată de **faza de extragere** (O*(1)*).
    - **Furnizarea** este lentă (O*(n)*).
    - **Căutarea** unei chei este lentă (O*(n)*).
        - Ea constă din căutarea propriu-zisă (O*(n)*) urmată de **faza de returnare** (O*(1)*).
    - **Traversarea ordonată** a cheilor este lentă.
        - Necesită un efort de calcul cuprins între O*(n log₂ n)* şi O*(n²)* în dependenţă de metoda de sortare utilizată.
    - Trebuie cunoscut cu precizie **numărul total** de intrări în tabelă.
- **Concluzii:**
    - Acest mod de implementare este recomandat a fi utilizat atunci când:
        - Se execută multe traversări neordonate ale tabelei.
        - Se fac comparaţii sau calcule care afectează fiecare intrare.
    - Construcţia şi inserţia sunt simple.
    - Trebuie cunoscută apriori dimensiunea maximă a tabelei.
    - Implementarea **nu** este recomandabilă atunci când se efectuează frecvent suprimări şi căutări, sau traversări ordonate.

7.2.3. Implementarea tabelelor cu ajutorul listelor înlănţuite ordonate

- **Avantaje:**
  - **Faza de instalare** la **inserţie** este rapidă (O*(1)*).
  - **Faza de extragere** la **suprimare** este rapidă (O*(1)*).
  - **Traversarea** în manieră ordonată a tabelei este rapidă (O*(n)*).
  - **Nu** este necesară precizarea **dimensiunii maxime** a tabelei.
  - Operatorul ***`TabelăPlină`*** este întotdeauna fals.
- **Dezavantaje:**
  - **Căutarea** se realizează în manieră **secvenţială** (O*(n)*).
    - Acest dezavantaj se răsfrânge asupra **fazei de căutare** la inserţie, suprimare, furnizare element şi căutare cheie.
  - **Inserţia** este lentă (O*(n)*).
    - Constă de fapt din căutare (O*(n)*) şi instalare (O*(1)*).
  - **Suprimarea** este lentă (O*(n)*).
    - Constă din căutare (O*(n)*) şi extragere (O*(1)*).
  - **Furnizarea** este lentă (O*(n)*).
    - Presupune o căutare (O*(n)*) urmată de un retur (O*(1)*).
  - **Căutarea** cheii este lentă (O*(n)*).
    - Constă dintr-o căutare (O*(n)*) urmată de un retur (O*(1)*).
    - În situaţia în care cheia **nu** există în tabelă, căutarea se poate opri chiar după poziţia unde ar trebui să se găsească cheia.
  - **Construcţia** tabelei este lentă (O$(n \cdot log_2 n)$ – O(n^2)) funcţie de tehnica utilizată pentru sortare.
  - **Efortul de calcul** necesitat de **gestionarea înlănţuirilor**.
- **Concluzii:**
  - Structura este eficientă pentru situaţiile în care:
    - Se realizează **traversări** frecvente în manieră **ordonată** ale tabelei.

        - Nu se cunoaşte **numărul maxim** de elemente.
    - Structura **nu** este propice investigaţiilor.
    - Inserţia şi suprimarea consumă timp în tabele de dimensiuni mari.

7.2.4. Implementarea tabelelor cu ajutorul listelor înlănţuite neordonate

- **Avantaje:**
    - **Inserţia** este rapidă.
        - Se realizează la începutul sau la sfârşitul listei (O*(1)*).
    - Faza de extragere la **suprimare** este rapidă (O*(1)*).
    - **Construcţia** tabelei este rapidă (O*(n)*)
        - Constă dintr-o succesiune de inserţii.
    - **Nu** este necesară precizarea dimensiunii maxime a tabelei.
    - Operatorul ***`TabelăPlină`*** este întotdeauna fals.
- **Dezavantaje:**
    - **Căutarea** este **secvenţială** (O*(n)*).
        - Dezavantajează inserţia, suprimarea, furnizarea şi căutarea propriu-zisă.
    - **Suprimarea** este lentă (O*(n)*).
        - Constă din **faza de căutare** (O*(n)*) urmată de **faza de extragere** (O*(1)*).
    - **Furnizarea** este lentă (O*(n)*).
        - Constă din căutare (O*(n)*) şi retur (O*(1)*).
    - **Căutarea** cheii este lentă (O*(n)*).
        - Similar ca şi la furnizare.
    - **Traversarea** în ordinea cheilor este lentă (O$(n \cdot log_2 n)$ – O(n^2)) funcţie de metoda de sortare utilizată.
    - Regia **legăturilor**.
- **Concluzii:**

- Această modalitate de implementare este indicată atunci când:
    - Se cere construcţia rapidă a tabelei.
    - Se realizează multe inserţii adiţionale.
    - Nu se cunoaşte dimensiunea maximă a tabelei.
- Investigările, suprimările sau traversările ordonate sunt consumatoare de timp.

7.3. Implementarea tabelelor prin tehnica dispersiei

- Una dintre metodele avansate de implementare a tabelelor o reprezintă **tehnica dispersiei**.
    - Această tehnică cunoscută şi sub denumirea de "**hashing**", reprezintă un exemplu de rezolvare elegantă şi deosebit de eficientă a problemei căutării într-un context bine precizat.

7.3.1. Consideraţii generale

- **Formularea problemei**:
    - Se dă o **mulţime** `S` de noduri identificate fiecare prin valoarea unei **chei**, organizate într-o **structură tabelă**.
    - Pe mulţimea cheilor se consideră definită o **relaţie de ordonare**.
    - Se cere ca tabela `S` să fie organizată de o asemenea manieră încât **regăsirea** unui nod cu o cheie precizată `k`, să necesite un **efort** cât mai redus.
- În ultimă instanţă, **accesul** la un anumit nod presupune **determinarea** acelei intrări din tabelă, la care el este **memorat**.
- Astfel, problema formulată se **reduce** la găsirea unei **asocieri specifice** `(H)` a **mulţimii cheilor** `(K)` cu **mulţimea intrărilor** `(A)` [7.3.1.a].

```
-------------------------------------------------------------
     H: K → A                                    [7.3.1.a]
-------------------------------------------------------------
```

- Problema **nu** este nouă, ea a mai fost abordată pe parcursul acestei lucrări.
    - Până la momentul de faţă astfel de asocieri au fost implementate cu ajutorul unor algoritmi de căutare în tablouri şi liste în diferite strategii de abordare şi în diverse moduri de organizare.
- În secţiunea de faţă se propune o altă **metodă**, simplă şi foarte eficientă în multe situaţii.

- Metoda se bazează pe **tehnica dispersiei** şi se aplică **structurilor tablou**.

- Deşi este o metodă care utilizează o **structură statică** de memorie, prin performanţele sale este un competitor serios al metodelor bazate pe structuri de date avansate.

- **Ideea** pe care se bazează **tehnica dispersării** este următoarea:

  - Se rezervă **static** un volum constant de memorie pentru **tabela dispersată**, care conţine **nodurile** cu care se lucrează.

  - **Tabela** se implementează în forma unei structuri de date **tablou** `T` având drept elemente nodurile în cauză.

  - Notând cu `p` **numărul elementelor tabloului**, indicele `a` care precizează un **nod oarecare**, ia valori între `0` şi `p-1`.

  - Se notează cu `K` **mulţimea tuturor cheilor** şi cu `k` o cheie oarecare.

    - Numărul cheilor este de obicei mult mai mare decât `p`. Un exemplu realist în acest sens este acela în care cheile reprezintă identificatori cu cel mult zece caractere, caz în care există aproximativ 10^{15} chei diferite în timp ce valoarea practică a lui `p` este 10^3.

  - Se consideră în aceste condiţii **funcţia** `H`, care defineşte o **aplicaţie** a lui `K` pe mulţimea tuturor indicilor, mulţime care se notează cu `L (L = {0,1,2,...,p-1})` [7.3.1.b].

`a = H(k)` *unde* `k ∈ K` *şi* `a ∈ L` [7.3.1.b]

- **Funcţia de asociere** `H` este de fapt o **funcţie de dispersie** şi ea permite ca pornind de la o cheie `k` să se determine indicele asociat `a`.

  - Este evident că `H` **nu** este o **funcţie bijectivă** deoarece numărul cheilor este mult mai mare decât numărul indicilor.

    - Practic există o mulţime de chei (clasă) cărora le corespunde acelaşi indice `a`.

  - Din acest motiv, o altă denumire uzuală sub care este cunoscută tehnica dispersării este aceea de **transformare a cheilor** ("**key transformation**"), întrucât **cheile** se transformă practic în **indici de tablou**.

- **Principiul metodei** este următorul.

  - Pentru **înregistrarea** unui nod cu cheia `k`:

    - (1) Se determină mai întâi **indicele asociat** `a=H(k)`.

    - (2) Se depune nodul la `T[a]`.

- Dacă în continuare apare o altă cheie `k'` care are acelaşi indice asociat `a=H(k')=H(k)`, atunci s-a ajuns la aşa numita **situaţie de coliziune**.
    - Pentru rezolvarea acestei situaţii trebuie adoptată o anumită **strategie de rezolvare** a coliziunii.
- La **căutare** se procedează similar.
    - (1) Dată fiind o cheie `k`, se determină `a=H(k)`.
    - (2) Se verifică dacă `T[a]` este nodul căutat, adică dacă `T[a].cheie=k`.
    - (3) Dacă da, atunci s-a găsit nodul, în caz contrar s-a ajuns la **coliziune**.

- În concluzie, aplicarea **tehnicii dispersiei** presupune soluţionarea a două **probleme**:
    - (1) Definirea **funcţiei de dispersie** `H`.
    - (2) Rezolvarea **situaţiilor de coliziune.**
- Rezolvarea favorabilă a celor două probleme conduce la rezultate remarcabil de eficiente, cu toate că metoda prezintă şi anumite dezavantaje, asupra cărora se va reveni.

7.3.2. Alegerea funcţiei de dispersie

- **Funcţia de dispersie** trebuie:
    - Pe de o parte **să repartizeze** cât mai **uniform** cheile pe mulţimea indicilor deoarece în felul acesta se reduce probabilitatea coliziunilor.
    - Pe de altă parte trebuie să fie **uşor şi rapid calculabilă**.
- Proprietăţile acestei funcţii sunt aproximate în literatura de specialitate de termenul "**hashing**" (amestec) motiv pentru care funcţia se notează generic cu `H` fiind denumită funcţie de **amestec** ("**hash function**").
- Din acelaşi motiv **tehnica dispersiei** se mai numeşte şi **tehnică hashing.**
    - În literatură se definesc diverse funcţii de dispersie fiecare cu avantaje şi dezavantaje specifice, activitate care conturează un domeniu de cercetare extrem de activ [Kn76].
- În cele ce urmează se va prezenta o variantă simplă a unei astfel de funcţii.
    - Fie `ORD(k)` o valoare întregă unică ataşată cheii `k`, valoare care precizează **numărul** de ordine al cheii `k` în **mulţimea ordonată** a tuturor cheilor (§1.3.2.).

- Funcţia H se defineşte în aceste condiţii astfel [7.3.2.a]:

```
-------------------------------------------------------------
     H(k)= ORD(k) MOD p                              [7.3.2.a]
-------------------------------------------------------------
```

- Această funcţie care asigură distribuţia cheilor pe mulţimea indicilor (0,p-1) stă la baza mai multor metode de repartizare.

- S-a demonstrat **experimental** că în vederea repartizării cât mai uniforme a cheilor pe mulţimea indicilor este **recomandabil** ca p să fie un număr **prim**.

    - Cazul în care p este o putere a lui 2 este extrem de favorabil din punctul de vedere al eficienţei calculului funcţiei, dar foarte nefavorabil din punctul de vedere al coliziunilor, mai ales în cazul în care cheile sunt secvenţe de caractere [Wi76].

7.3.3. Tratarea situaţiei de coliziune

- Prezenţa unei situaţii de coliziune presupune **generarea unui nou indice** în tabelă pornind de la cel anterior, numit **indice secundar**.

- Există două modalităţi principiale de generare a indicilor secundari:

    - (1) O modalitate care presupune un **spaţiu suplimentar** asociat tabelei şi care prefigurează **metoda dispersiei deschise**.

    - (2) O a doua modalitate care exploatează doar spaţiul de memorie alocat tabelei şi care prefigurează **metoda dispersiei închise**.

7.3.3.1. Metoda dispersiei deschise

- Metoda **dispersiei deschise** presupune înlănţuirea într-o **listă înlănţuită** a tuturor nodurilor ale căror indici primari sunt identici, metodă care se mai numeşte şi **înlănţuire directă.**

    - Elementele acestei liste pot sau **nu** aparţine tabloului iniţial: în ultimul caz memoria necesară alocându-se din aşa numita "**zonă de depăşire**" a tabelei.

- Structurile de date aferente acestei metode sunt prezentate în [7.3.3.1.a]. Ele prefigurează de fapt **un tablou ale cărui elemente sunt liste înlănţuite**.

```
-------------------------------------------------------------
{Dispersia deschisă - Structuri de date}

CONST DimensiuneTabela = p   {număr prim}

TYPE TipIndex = 0..DimensiuneTabela;
     TipInfo = ... ;
     TipCheie = ... ;
```

```
     TipElement = RECORD
       cheie: TipCheie;
       info: TipInfo
     END;

     TipReferinta = ^TipNod;                    [7.3.3.1.a]

     TipNod = RECORD
       element: TipElement;
       urm: TipReferinta
     END;

     TipTabela = ARRAY[TipIndex] OF TipReferinta;

VAR t: TipTabela;
-------------------------------------------------------------
```

- Această metodă are **avantajul** că permite **suprimarea** elementelor din tabelă, operaţie care **nu** este posibilă în toate implementările.

- Dintre **dezavantaje** se evidenţiază două:

    - Necesitatea menţinerii unei **liste secundare**.

    - Prelungirea fiecărui nod cu un spaţiu de memorie pentru pointerul (indexul) necesar înlănţuirii.

7.3.3.2. Metoda dispersiei închise

- Metoda **dispersiei închise** utilizează, după cum s-a precizat anterior, **strict** spaţiul de memorie alocat tabelei.

    - În cazul unei **coliziuni** se realizează parcurgerea după o anumită **regulă** a tabloului dispersat `T` până la găsirea primului **loc liber** sau a **cheii căutate**.

- Structurile de date aferente acestei metode apar în [7.3.3.2.a].

```
-------------------------------------------------------------
{Dispersia închisă - Structuri de date}

CONST DimensiuneTabela = p;    {număr prim}

TYPE TipIndex = 0..DimensiuneTabela;
     TipInfo = ... ;
     TipCheie = ... ;

     TipElement = RECORD
       cheie: TipCheie;
       info: TipInfo                            [7.3.3.2.a]
     END;

     TipTabela = ARRAY[TipIndex] OF TipElement;
```

Schiţa de principiu a algoritmului care realizează **căutarea** în tabelă este prezentat în secvenţa [7.3.3.2.b].

```
{Metoda dispersiei închise - căutarea unui element în tabelă}

VAR t: TipTabela;
    l: TipIndex;

a:= H(k); i:= 0;

REPEAT                                                    [7.3.3.2.b]
  IF t[a].cheie=k THEN
      *element găsit
    ELSE
      IF t[a].cheie=liber THEN
          *elementul nu este în tabelă
        ELSE
          BEGIN{coliziune}
              i:= i+1;
              a:= a+g(i)
          END
UNTIL (găsit) OR (nu este în tabelă);
```

- Funcţia **g(i)** este aceea care precizează **regula** după care se parcurge tabela în caz de coliziune.

- Există numeroase astfel de funcţii **g** cărora le corespund modalităţi diverse în conformitate cu care se realizează parcurgerea tabelei.

- Dintre acestea cele mai cunoscute sunt **adresarea deschisă liniară** şi **adresarea deschisă patratică**.

7.3.3.3. Adresarea deschisă liniară

- **Adresarea deschisă liniară** prefigurează cea mai simplă metodă de parcurgere a tabelei în cadrul **tehnicii dispersiei închise**.

- Conform **adresării deschise liniare**, intrările tabelei se parcurg **secvenţial**, tabela considerându-se **circulară**. Cu alte cuvinte **g(i)=i**, iar modalitatea de generare a intrărilor în tablelă este precizată în [7.3.3.3.a].

---

a_0 = H(k)
a_i =(a_0+i) **MOD** p *i=1...p-1* [7.3.3.3.a]

- În fragmentul de program din secvenţa [7.3.3.3.b], se prezintă procesul de **căutare în tabelă** a intrării cu cheia k, bazat pe **adresarea deschisă liniară**.

- Dacă intrarea se găseşte, se asignează variabila de ieşire v cu conţinutul său.

```
-------------------------------------------------------------
{Adresarea deschisă liniară - căutarea unui element în
tabelă}

VAR  t: TipTabela
     a,i: TipIndex;
     gasit,absent: boolean;
     v: TipElement;

a:= H(k); i:= l; gasit:= false; absent:= false;

REPEAT
  IF t[a]=liber THEN
      absent:= true
    ELSE
      IF t[a].cheie=k THEN
          BEGIN
            v:= t[a]; gasit:= true
          END                                    [7.3.3.3.b]
        ELSE
          BEGIN{coliziune}
            a:= a+1;
            IF a=p THEN a:= 0;
            IF a=i THEN absent:= true
          END
UNTIL gasit OR absent;
-------------------------------------------------------------
```

- Acest **algoritm** are drept principal **dezavantaj** tendinţa de a **îngrămădi** nodurile în continuarea celor deja înregistrate.

    - Dezavantajul **nu** este inerent metodei de **adresare deschisă liniară**, el datorându-se **regulii** de parcurgere a elementelor tabelei.

- Ideal ar fi ca în caz de coliziune să se aplice o regulă care distribuie încercările, cu **probabilităţi** egale, pe **locurile libere** rămase în tabelă.

    - Realizarea practică a acestui deziderat este însă deosebit de complexă.

7.3.3.4. Adresarea deschisă pătratică

- **Adresarea deschisă pătratică** reprezintă o altă soluţie de compromis relativ simplă şi eficientă de parcurgere a tabelei în cazul **metodei dispersiei închise**, soluţie care rezolvă unele dintre deficienţele **adresării deschise liniare**.

- În cazul **adresării deschise pătratice** funcţia **g** are expresia $g(i)=i^2$, iar modalitatea de generare a intrărilor în tablelă este precizată în [7.3.3.4.a].

```
-------------------------------------------------------------
   a0 = H(k)
   ai = (a0+i²) MOD p        i=1,2,3,...          [7.3.3.4.a]
```

- O metodă simplă de **implementare** a acestei **funcţii de parcurgere** este următoarea:

    - Dacă un indice `a` conduce la **coliziune**, atunci proxima încercare se face la indicele `a+r` unde `r` este o mărime care se iniţializează pe `1` şi care după fiecare încercare nereuşită se incrementează cu `2`.

    - Astfel valorile succesive ale lui `r` formează **şirul numerelor fără soţ**, şir a cărui **sumă** este întotdeauna un **pătrat perfect.**

- În caz de coliziuni repetate, numărul încercărilor se va limita prin impunerea condiţiei formale `r < p`.

- Algoritmul de **inserţie** a unui nou element `v` în tabelă, bazat pe **adresarea deschisă patratică** este prezentat în secvenţa [7.3.3.4.b].

```
{Adresarea deschisă patratică - inserţia unui element în tabelă}

VAR  t: TipTabela
     a,r: TipIndex;
     depus: boolean;
     v: TipElement;

a:= H(k); r:= 1; depus:= false;

REPEAT
  IF t[a]=liber THEN
      BEGIN
        t[a]:= v; depus:= true
      END                                          [7.3.3.4.b]
    ELSE
      BEGIN
        a:= a+r;
        IF a>=p THEN a:= a-p;
        r:= r+2
      END
UNTIL depus OR (r=p);
```

- Un **dezavantaj** minor al acestei metode este acela că se poate întâmpla să **nu** se găsească nici un element liber, ajungându-se la depăşirea tabelei, cu toate că în realitate în tabelă mai există locuri libere.

    - Se demonstrează însă că **probabilitatea** acestui eveniment este **foarte mică** şi că în general metoda este **foarte performantă**.

- **Traversarea** ordonată a tabelei este greu de realizat.

- **Dezavantajul major** al implementărilor **dispersiei închise** în ambele variante, se referă la faptul că ea **nu permite suprimarea elementelor din tabelă.**

- Suprimarea unui element oarecare al tabelei **întrerupe** secvenţa **liniară** sau **pătratică** de parcurgere a tabelei fapt care poate conduce la **pierderea** iremediabilă a unor intrări.

- Pentru a remedia acest dezavantaj major se poate recurge la următoarea **stratagemă**.

  - Se asociază fiecărei intrări a tabelei un câmp de tip enumerare numit `stare` care poate lua valorile `liber, ocupat` sau `şters`.

  - În consecinţă `TipElement` din secvenţa [7.3.3.2.a] se modifică după cum urmează [7.3.3.4.c].

```
-------------------------------------------------------------
TYPE    ... ;

     TipElement = RECORD
       cheie: TipCheie;
       info: TipInfo;                          [7.3.3.4.c]
       stare: (liber,ocupat,şters)
     END;
-------------------------------------------------------------
```

  - Iniţial la **crearea tabelei** se trec **toate** locaţiile sale în starea `liber`.

  - Pe măsură ce se **inserează** elemente, locaţiile corespunzătoare se trec în starea `ocupat`.

- În aceste condiţii:

  - **Suprimarea** se poate realiza simplu fără îndepărtarea efectivă a elementului prin simpla trecere a stării câmpului asociat în `şters`.

  - În **procesul de căutare** a unei chei se parcurg **şi** câmpurile aflate în starea `şters`, secvenţa de adresare creată rămânând nemodificată.

  - Câmpurile marcate cu `şters` pot fi însă **reutilizate** la introducerea unor noi elemente în tabelă.

7.3.4. Analiza performanţei dispersiei închise

- Pentru **analiza performanţei dispersiei închise** se presupune pentru început că:

  - (1) Toate cheile au **probabilităţi** de apariţie **egal posibile**.

  - (2) Funcţia de distribuţie `H` **repartizează uniform** cheile pe mulţimea intrărilor tabelei.

- Se presupune de asemenea că o cheie oarecare se inserează într-o tabelă de dimensiune `n`, care conţine `k` elemente, adică `k` intrări din tabelă sunt **deja ocupate**.

- În consecinţă:
  - Probabilitatea de a nimeri o **locaţie ocupată** este `k/n`.
  - Probabilitatea de a nimeri o **locaţie liberă** la prima încercare este `1-k/n`.
  - Aceasta este de asemenea şi probabilitatea notată cu p_1 ca să fie necesară o **singură încercare** pentru a găsi o **intrare liberă** într-o tabelă cu `n` intrări, dintre care `k` sunt deja ocupate [7.3.4.a].

$$p_1 = 1 - \frac{k}{n} = \frac{n-k}{n} \qquad [7.3.4.a]$$

- Probabilitatea p_2 ca să fie necesară exact **o a doua încercare** pentru a găsi o intrare liberă în tabelă este egală cu produsul dintre:
  - (1) Probabilitatea apariţiei unei **coliziuni** în **prima încercare**.
  - (2) Probabilitatea de a nimeri o locaţie **liberă** în a **doua încercare**.
- În manieră similară se poate calcula probabilitatea p_3 de a găsi o intrare liberă **după 3 încercări** şi prin generalizare probabilitatea p_i valabilă pentru `i` **încercări** [7.3.4.b].

$$p_2 = \frac{k}{n} \cdot \frac{n-k}{n-1} \quad , \qquad p_3 = \frac{k}{n} \cdot \frac{k-1}{n-1} \cdot \frac{n-k}{n-2}$$

$$\ldots \qquad [7.3.4.b]$$

$$p_i = \frac{k}{n} \cdot \frac{k-1}{n-1} \cdot \frac{k-2}{n-2} \cdot \ldots \cdot \frac{k-(i-2)}{n-(i-2)} \cdot \frac{n-k}{n-(i-1)}$$

- Prin urmare, **numărul total** de **încercări** E_k aşteptat a fi necesar pentru **inserţia celei de-a k chei** este o sumă de termeni de forma `i*`p_i pentru `i=1,...,k`, precizată de formula [7.3.4.c] unde:
  - `i` reprezintă **numărul de încercări** .
  - p_i **probabilitatea** ca să fie necesare exact `i` **încercări**.

$$E_k = \sum_{i=1}^{k} i \cdot p_i = 1 \cdot \frac{n-k}{n} + 2 \cdot \frac{k}{n} \cdot \frac{n-k}{n-1} + \ldots +$$

$$+ k \cdot \frac{k}{n} \cdot \frac{k-1}{n-1} \cdot \frac{k-2}{n-2} \cdot \ldots \cdot \frac{k-(k-2)}{n-(k-2)} \cdot \frac{n-k}{n-(k-1)} = \frac{n+1}{n-k+2} \qquad [7.3.4.c]$$

- **Numărul de încercări** necesar pentru a **insera** un element este identic cu **numărul de încercări** necesar pentru a-l **găsi**.

- Ca atare, valoarea lui E_k reprezintă:
    - **Numărul de încercări** aşteptat a fi necesar pentru a **insera** cea de-a `k` cheie în tabelă.
    - **Numărul de încercări** aşteptat a fi necesar pentru a **extrage** o cheie dintr-o tabelă care conţine deja `k` elemente.

- Valoarea E_k poate fi utilizată pentru calculul **numărului mediu de încercări** `E`, necesar realizării **inserţiei** sau **accesului la o cheie oarecare** din **tabelul dispersat** care conţine `m` elemente [7.3.4.d].

$$E = \frac{1}{m} \cdot \sum_{k=1}^{m} E_k = \frac{n+1}{m} \cdot \sum_{k=1}^{m} \frac{1}{n-k+2} = \frac{n+1}{m} \cdot (H_{n+1} - H_{n-m+1}) \qquad [7.3.4.d]$$

- În formula de mai sus H_n este **suma seriei armonice** care poate fi aproximată prin relaţia [7.3.4.e] în care γ este constanta lui Euler [Wi74].

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \approx ln(n) + \gamma \qquad [7.3.4.e]$$

- Dacă se face substituţia $\alpha = m/(n+1)$ se obţine relaţia [7.3.4.f].

$$E = \frac{1}{\alpha} \cdot \left[ln(n+1) - ln(n-m+1) \right] = \frac{1}{\alpha} \cdot ln \frac{n+1}{n+1-m} = -\frac{1}{\alpha} \cdot ln(1-\alpha) \qquad [7.3.4.f]$$

- α aproximează **raportul** dintre **locaţiile ocupate** şi cele **disponibile** în cadrul **tabelei dispersate**.
    - El se mai numeşte şi **factor de umplere**.
        - $\alpha = 0$ precizează **tabela vidă**.
        - $\alpha = n/(n+1)$ precizează **tabela plină**.

- Numărul **probabil** de încercări necesar găsirii sau inserţiei unei chei `k` funcţie de factorul de umplere α, apare tabelat în figura 7.3.4.a.

- Rezultatele numerice sunt surprinzătoare.
    - Ele **nu depind de dimensiunea tabelei**.
    - În acelaşi timp explică **performanţa excepţională** a acestei metode.

- Chiar dacă tabela este plină în proporţie de 90 %, în medie sunt necesare numai 2.56 încercări pentru a localiza o cheie sau a găsi un loc liber în ea.

Nr. crt.	α	E(α)
1	0.10	1.05
2	0.25	1.15
3	0.50	1.39
4	0.75	1.85
5	0.90	2.56
6	0.95	3.15
7	0.99	4.66

Fig.7.3.4.a. Tabelarea numărului probabil de încercări funcţie de factorul de umplere al tabelei (cazul ideal)

Nr. crt.	α	E(α)
1	0.1	1.06
2	0.25	1.17
3	0.5	1.50
4	0.75	2.50
5	0.9	9.90
6	0.95	10.50

Fig.7.3.4.b. Tabelarea relaţiei experimentale E(α) funcţie de factorul de umplere α

- Rezultatele prezentate presupun însă **cazul ideal**, respectiv o metodă de tratare a coliziunilor care repartizează în mod perfect uniform cheile pe mulţimea locaţiilor rămase libere.
    - În realitate, acest lucru este practic irealizabil.
- Analiza **experimentală** a comportamentului real a **dispersiei închise** bazate pe **adresarea deschisă liniară** a condus la relaţia experimentală [7.3.4.g].

$$E = \frac{1-\alpha/2}{1-\alpha} \qquad [7.3.4.g]$$

- Reprezentarea grafică ale valorilor sale numerice funcţie de factorul de umplere α apar reprezentate în figura 7.3.4.b [Wi74].
- Rezultatele obţinute chiar pentru metoda cea mai puţin elaborată de tratare a coliziunilor (adresarea deschisă liniară) sunt atât de bune încât ar putea exista tentaţia de a considera metoda dispersiei drept soluţie pentru orice situaţie.
- Performanţele metodei, cel puţin din punctul de vedere al numărului de comparaţii necesare pentru găsire sau inserţie, sunt superioare celor mai sofisticate metode bazate pe structuri arbore.
- Desigur, metoda are şi o serie de **dezavantaje**.
- Primul şi cel mai important, este acela că, **dimensiunea tabelei** fiind fixă aceasta **nu** poate fi ajustată conform cererii curente.
    - O estimare a priori a dimensiunii sale maxime este de regulă greu de realizat şi conduce fie la o slabă utilizare a memoriei fie la performanţe scăzute prin depăşirea tabelei dispersate.
    - Chiar dacă numărul de elemente este cunoscut exact – lucru care se întâmplă foarte rar – în vederea obţinerii unei bune performanţe, dimensiunea tabelei trebuie aleasă cam cu 10-20 % mai mare.

- O soluţie pentru rezolvarea situaţiei în care tabela se apropie de un factor de umplere de 100%, o reprezintă aşa numita **tehnică rehashing** care presupune reconstrucţia integrală a tabelei de dispersie pentru o valoare crescută corespunzător a lui `p`.

- O a doua deficienţă majoră se referă la **suprimarea cheilor**.
  - În **dispersia închisă clasică**, suprimarea elementelor este imposibilă.
  - Această deficienţă poate fi însă remediată prin strategia prezentată anterior, conform căreia, fiecarei locaţii a tabelei i se ataşează un câmp de stare care poate lua valorile `liber, ocupat` sau `şters`.
  - În **dispersia deshisă** suprimarea cheilor este posibilă întrucât aceasta se bazează pe metoda înlănţuirii care utilizează un spaţiu de memorie suplimentar pentru înlănţuirea directă, elemente care au fost deja abordate în cadrul capitolului de faţă.

- În concluzie, **tehnica dispersiei nu** este recomandabil a fi utilizată dacă:
  - Volumul de date este cunoscut cu **mică probabilitate**.
  - Volumul de date este **puternic variabil**.
  - Volumul de date **creşte în timp**.

7.3.5. Exemplu. Construcţia unui tablou de referinţe încrucişate

- În cadrul acestui paragraf se prezintă un program pentru construirea unei aşa numite **tabele de referinţe încrucişate** ("**cross reference index**") asociată unui text dat.

- **Specificaţia programului** este următoarea:
  - Programul citeşte ca intrare **un text**, pe care îl afişează inserând la începutul fiecărui rând numărul curent al rândului (*n* = 1, 2, 3, ...).
  - În acelaşi timp, programul selectează şi memorează **toate cuvintele textului** precum şi **numerele rândurilor** în care acestea au fost întâlnite.
  - La terminarea parcurgerii textului, programul poate furniza o **tabelă** conţinând **toate cuvintele ordonate alfabetic** şi **lista liniilor** în care apare fiecare cuvânt (tabela de referinţe încrucişate).

- În implementare se utilizează metoda **dispersiei închise** bazată pe **adresare deschisă pătratică.**

- Un element al **tabelei dispersate** (`TipCuvant`) este o **structură articol** care pe lângă câmpul `cheie` care memorează cuvântul ca atare, are asociată o listă înlănţuită a **numerelor liniilor** în care este depistat cuvântul respectiv în procesul de baleere.

- Această listă este precizată de variabilele `inceput` şi `sfarsit` de tip referinţă.

- De asemenea odată cu introducerea lor în tablela dispersată, elementele sunt înlănţuite într-o listă liniară simplă prin intermediul câmpului `leg`, listă care va fi utilizată ulterior în procesul de **traversare ordonată** a tabelei.

- Programul integral asociat apare în secvenţa [7.3.5.a].

    - Metoda de tratare a **coliziunilor** este metoda **adresării pătratice**.

    - Câmpul `leg` înregistrează indici referitori la tabela `T` prin intermediul cărora elementele efectiv utilizate ale tabelei sunt înlănţuite într-o listă specială utilizată ulterior la afişare.

    - **Metoda dispersiei** conform căreia se construieşte şi se exploatează cu mare eficienţă tabela `T`, are dezavantajul că elementele depuse **nu** sunt ordonate alfabetic.

    - Această ordonare se realizează **artificial**, cu eficienţă mai redusă, în momentul listării tabelei de referinţe încrucişate cu ajutorul listei înlănţuite mai sus amintite.

```
------------------------------------------------------------
program ReferinteIncrucisate;
uses Strings;

const C1 = 10; {Lungime maximă cuvânt}          [7.3.5.a]
      C2 = 8;  {Numărul maxim de elemente afişate pe o
                                                      linie}
      C4 = 500; {Număr maxim de linii}
      P = 997; {Lungime tablou dispersie}
      LIBER = '          ' ;

type TipIndice = 0..P;

     RefNumRand = ^NumRand;

     TipCuvant = record {Element tabelă de dispersie}
       cheie: String;
       inceput,sfarsit: RefNumRand;
       leg: TipIndice;
     end;

     NumRand = record  {Nod listă numere linii}
        nr: 0..C4;
        urm: RefNumRand;
     end;

     TabelaDispersie = array [0..P] of TipCuvant;

var  I,Init: TipIndice;
     Litere,Cifre: set of char;
     K,K1: integer;
     N: integer;    {Număr curent al rândului}
```

```
    A: String;     {Tampon pentru cuvântul curent}
    Car: char;     {Caracterul curent}
    T: TabelaDispersie;
    Depasire: boolean;
    NumeIntrare: String;
    Intrare: Text;

function H(Id: String): indice;
  var I,S: Integer;
  begin
   S:= 0;
   for I:= 1 to length(Id) do
     S:= S + ord(Id[I]) * trunc(exp(I * ln(2.0)));
   H:= S mod P;
  end;{H}

procedure Cauta;
  var L,R: Integer;
      X: RefNumRand;
      F: boolean;
  begin
    L:= H(A); F:= false; Depasire:= false; R:= L;
    new(X); X^.nr:= N; X^.urm:= NIL;
    repeat
      if T[L].cheie = A then {s-a găsit}
        begin
          F:= true;
          T[L].sfarsit^.urm:= X; T[L].sfarsit:= X;
        end
          else
            if T[L].cheie = LIBER then {se creează un
                                        nou element}
              begin
                F:= true;
                with T[L] do
                  begin
                    cheie:= A; inceput:= X;
                    sfarsit:= X; leg:= Init;
                  end;
                Init:= L;
              end
            else
              begin {coliziune}
                L:= L + R; R:= R + 2;
                if L >= P then
                  L:= L - P;
                if (R = P) then
                  begin
                    writeln ('Depasire tabel');
                    Depasire:= true;
                  end
              end;
    until F or Depasire;
  end;{Cauta}
```

```
procedure Imprimare;
  var I, J, M: Indice;

  procedure ImprCuvant(R:TipCuvant);
  var L: Integer;
      X: RefNumRand;

    begin
      write (' ', R.cheie);
      X:= R.inceput; L:= 0;
      repeat
        write(' ');
        if (L=C2) then
          begin
            writeln;
            for L:= 1 to C1+1 do
              write(' ');
            L:= 0
          end;
        L:= L + 1; write (X^.nr); X:= X^.urm;
      until X=nil;
      writeln;
    end;{ImprCuvant}

  begin {Imprimare}
    I:= Init;
    while I<>P do {parcurge T şi caută elementul cu cheia
                   minimă}
      begin
        M:= I; J:= T[I].leg;
        while J<>P do
          begin
            if T[J].cheie<T[M].cheie then
              M:= J;
            J:= T[J].leg;
          end; {s-a găsit elementul cu cheia minimă}
        ImprCuvant(T[M]);
        if M<>I then
          begin
            T[M].cheie:= T[I].cheie;
            T[M].inceput := T[I].inceput;
            T[M].sfarsit := T[I].sfarsit
          end;
        I:= T[I].leg;
      end
    end;{Imprimare}

  begin {Program principal};
    litere:= ['A'..'Z', '_']; cifre:= ['0'..'9'];
    write('Nume fisier'); readln(NumeIntrare);
    assign (Intrare, NumeIntrare); reset(Intrare);
    N:= 0; K1:= C1; Init:= P; Depasire:= false;
    read(Intrare, Car);
    Car:= UpCase(Car); {rezolva caracterele 'a'..'z'}
    for I:= 0 to P-1 do
```

```
    T[I].Cheie := LIBER;
  while not eof(Intrare) or Depasire do
    begin
      if N=C4 then
        N:= 0;
    N:= N + 1;
    write(N, ' ');
    while not (Car=chr(13)) or Depasire do
      begin
        if Car in Litere then
          begin
            K:= 0; A:=  '';
            repeat
              if K<C1 then
                begin
                  K:= K + 1; A:= A + car;
                end;
              write(Car); read(Intrare, Car);
              Car:=UpCase(Car);
            until not (Car in Litere) or (Car in Cifre);
            if (K>=K1) then
                K1 := K
              else
                repeat
                  A[K1]:= ' '; K1:= k1 - 1;
                until K1=K;
            cauta
          end
        else
          begin
            if Car='''' then
                repeat
                  write(Car); read(Intrare, Car);
                  Car:= UpCase(Car);
                until Car=''''
              else
                if Car='[' then
                  repeat
                    write(Car); read(Intrare, Car);
                    Car:= UpCase(Car);
                  until Car=']';
            write(Car); read(Intrare, Car);
            Car:= UpCase(Car);
          end{else}
      end;{while}
      if (Car = chr(13)) then
      read(Intrare, Car); {cerut de DOS}
      writeln; read(Intrare, Car); Car:= UpCase(Car);
    end;{while}
  Imprimare
end.
```

7.4. Aplicaţii propuse

Aplicaţia 7.4.1

Se cere să se implementeze **TDA Tabelă** precizat în cadrul capitolul, în fiecare dintre modalităţile sugerate, respectiv utilizând:

- Tablouri neordonate.
- Tablouri ordonate.
- Liste înlănţuite neordonate.
- Liste înlănţuite ordonate.

Pentru fiecare dintre operatorii ***CautaCheie***, ***InserElemTabela, SuprimElemTabela,*** să se realizeze un studiu comparativ al performanţelor implementărilor în baza construcţiei profilului algoritmului corespunzător.

Aplicaţia 7.4.2

Se cere să se implementeze **TDA Tabelă** cu ajutorul **tehnicii dispersiei deschise.** Să se construiască profilul algoritmului de căutare în acest context şi să se comenteze performanţele obţinute.

Aplicaţia 7.4.3

Se cere să se implementeze **TDA Tabelă** cu ajutorul **tehnicii dispersiei închise** pentru cele două variante descrise:

- Adresarea deschisă liniară.
- Adresarea deschisă pătratică.

Să se construiască profilul algoritmului de căutare în cele două situaţii şi să se realizeze o analiză comparată a performanţelor celor două implementări.

Aplicaţia 7.4.4

Se cere să se construiască o **tabelă de dispersie** care conţine cuvintele unui text, fiecare cuvânt având asociat contorul propriu de apariţii (problema concordanţei). Să se compare performanţele acestei implementări cu cea bazată pe liste înlănţuite.

Aplicaţia 7.4.5

Pentru fiecare dintre implementările tehnicii dispersiei realizate anterior (aplicaţiile 7.4.2 şi 7.4.3) să se studieze corespondenţa dintre numărul de încercări necesar pentru găsirea unei locaţii libere şi factorul de umplere al tabelei. Se se realizeze o analiză comparată a rezultatelor obţinute.

Aplicaţia 7.4.6

Pentru fiecare dintre implementările tehnicii dispersiei realizate anterior (apl. 7.4.2 şi 7.4.3) se cere să se implementeze operatorul de suprimare. Să se studieze influenţa modificărilor necesare asupra performanţelor de ansamblu ale metodei.

Aplicaţia 7.4.7

Principalul dezavantaj al tehnicii dispersiei este acela că dimensiunea tabelei este fixă, ea neputând fi ajustată în raport cu cererea curentă. În prezenţa unui mecanism de alocare dinamică a memoriei, în momentul în care tabela de dispersie T este plină, se poate genera un tablou mai mare T' în care se transferă toate intrările din tabela T. Această tehnică se numeşte **redispersie** ("**rehashing**"). Se cere să se realizeze un program care implementează **tehnica redispersiei**.