1 Tema Nr. 5: Căutarea în tabele de dispersie

Adresare deschisă prin verificare pătratică
Timp alocat: 2 ore

1.1 Implementare

Se cere implementarea **corectă** și **eficientă** a operațiilor de *inserare* și *căutare* într-o tabelă de dispersie ce folosește *adresarea deschisă* cu *verificare pătratică*.

Informații utile și pseudo-cod găsiți în notițele de curs sau în carte ([1]), în secțiunea *11.4 Open addressing*.

Noțiunile închis/deschis (closed/open) specifică dacă se obligă folosirea unei poziții sau structuri de date.

1.1.1 Hashing (se referă la tabela de dispersie (hash table))

- Open Hashing: pe o anumită poziție se pot stoca mai multe elemente (ex: chaining)
- Closed Hashing: se poate stoca doar un singur element pe o anumită poziție (ex. linear/quadratic probing)

Addressing (se referă la poziția finală a unui element față de poziția inițială)

- Open Addressing (Adresare Deschisă): adresa finală (poziția finală) nu este complet determinată de către codul hash. Poziția depinde și de elementele care sunt deja în tabelă. (ex: linear/quadratic probing - verificare liniară/pătratică)
- Closed Addressing (Adresare Închisă): adresa finală este întotdeauna determinată de codul hash (poziția inițială calculată) și nu exista probing (ex: chaining)

Pentru această temă tabela de dispersie va avea o structură similară cu cea de mai jos:

```
typedef struct {
    int  id;
    char name[30];
} Entry;
```

unde *adresa finală* în tabelă va fi calculată aplicând funcția de hashing pe câmpul *id* din structură. Câmpul *name* va fi folosit doar pentru demonstrarea corectitudinii operațiilor de căutare și ștergere și nu este necesară pentru evaluarea performanței (câmpul *name* va fi afișat în consolă dacă operația de căutare găsește *id*-ul respectiv, altfel afișați "negăsit").

1.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minimale (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo*: Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerinței, calitatea interpretărilor din comentariul bloc și* ***răspunsul corect dat de dumeavostră la întrebările puse de către profesor.***

1.3 Cerințe

1.3.1 Implementarea operației de inserare și căutare folosind structura de date cerută + *demo (dimensiune 10)* (5p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (ex. 10).

1.3.2 Evaluarea operației de căutare pentru un singur factor de umplere 95% (2p)

Se cere evaluarea operației de *căutare* în tabele de dispersie cu adresare deschisă și verificare pătratică, în cazul **mediu statistic** (nu uitați să repetați măsurătorile de 5 ori). Pentru a obține evaluarea, trebuie să:

1. Alegeți N, dimensiunea tabelei, un număr prim în jur de 10000 (e.g. 9973, sau 10007);
2. Pentru fiecare din următoarele valori pentru factorul de umplere $\alpha = \mathbf{0.95}$:
    a. Inserati în tabela n elemente aleator, astfel incat sa ajungeți la valoare lui α ($\alpha = n/N$)
    b. Căutați aleator, în fiecare caz, m elemente ($m = 3000$), astfel încât aproximativ jumătate din elemente sa fie *găsite*, iar restul sa *nu fie găsite* (în tabelă). *Asigurați-vă că elementele găsite sunt generate*

uniform, i.e. să căutați elemente care au fost introduse la moment diferite, cu probabilitate egală (există mai multe moduri în care se poate garanta acest lucru)

c. Numărați operațiile efectuate de procedura de căutare (i.e. numărul de celule accesate)

d. Atenție la valorile pe care le căutați, să fie într-o ordine aleatoare din cele introduse. *Dacă sunt primele 1500 adăugate, implicit efortul mediu găsite va fi 1.*

3. Generați un tabel de forma:

Table 1: Effort measurements at various filling factors

Filling factor	**Avg. Effort *(found)***	**Max Effort *(found)***	**Avg. Effort *(not-found)***	**Max Effort *(not-found)***
0.95	...	...	...	...

Efort mediu = efort_total / nr_elemente
Efort maxim = număr maxim de accese efectuat de o operație de căutare

1.3.3 Completarea evaluării pentru toți factorii de umplere (2p)

Respectând cerințele de la punctul 2 cu $\alpha \in \{0.8, 0.85, 0.9, 0.95, 0.99\}$, generați un tabel de forma:

Table 2: Effort measurements at various filling factors

Filling factor	**Avg. Effort *(found)***	**Max Effort *(found)***	**Avg. Effort *(not-found)***	**Max Effort *(not-found)***
0.80				
0.85				
...	...		...	

1.3.4 Implementare operației de ștergere într-o tabelă de dispersie și evaluarea operației de căutare după ștergerea unor elemente (1p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (ex. 10).

Pentru evaluarea operației de căutare după ștergere, umpleți tabela de dispersie până la factor de umplere 0.99. Stergeți elemente din tabela până ajungeți la factor umplere 0.8, după care căutați m elemente aleator (m ~ 3000), astfel încât aproximativ jumătate din elemente să fie *găsite*, iar restul să nu fie găsite (în tabelă). Măsurați efortul necesar pentru *căutare* și adăugați în tabelul generat anterior.

References

[1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.