Compararea dintre linked lists și skip lists

Ciuperceanu Vlad-Mihai
Grupa 151

1 Aspecte ale proiectării codului

Înainte a implementa cele două structuri de date, vom începe prin a scrie clasele necesare ambelor tipuri de noduri. Apoi, vom trece la construirea structurilor de date. În ambele cazuri, nodurile vor reține perechi de tipul $(key, value)$, unde cheia este un string, iar valoarea este un număr, pentru a putea să testăm performanțele celor două structuri.

Pentru *Linked List*, fiecare nod va avea un pointer *next* către următorul nod din listă. În cadrul unei liste simplu înlănțuite, vom păstra capătul listei, toate funcțiile pornind de la acesta. Totodată, vom mai păstra și capătul de final al listei, pentru a asigura inserarea în $O(1)$. Ca funcționalități, avem *insert* și *remove*, ce adaugă un nod la sfârșitul listei, respectiv elimină un nod după o cheie dată. De asemenea, mai avem funcțiile *search*, ce caută un nod după o cheie dată, precum și *print* (ce afișează toată lista) și *sort* (ce sortează lista după chei). Din punctul de vedere al complexității, operațiile descrise au complexitățile: $insert - O(1)$, $remove - O(n)$, $search - O(n)$, $print - O(n)$ și $sort - O(n^2)$, unde n este numărul de noduri din listă.

Pentru *Skip List*, vom reține într-un nod, pe lângă perechea $(key, value)$, și un vector de pointeri $forward$, folosit pentru a face legătura nodului curent cu alte noduri de pe alte niveluri din structură. În cadrul unui *Skip List*, vom reține numărul de niveluri, precum și punctul de plecare al listei. De asemenea, am setat numărul maxim de niveluri la 16 printr-o constantă definită anterior. Funcționalitățile oferite sunt aceleași ca și cele descrise mai sus, având în plus funcția *getRandomLevel*, care oferă un nivel random pentru un momentul în care ne dorim să inserăm un nod, bazându-se pe probabilitățile descrise în curs. De asemenea, nu mai avem nevoie de funcția de sortare, întrucât ordinea cheilor este păstrată în cadrul operațiilor. Diferența majoră față de *Linked List* se află în complexitatea acestor operații. Într-un *Skip List*, operațiile au complexitățile: $insert - O(\log n)$, $remove - O(\log n)$, $search - O(\log n)$, $print - O(n)$, cu precizarea că valorile pentru *insert*, *remove* și *search* sunt valorile medii, putând ajunge la $O(n)$ în cel mai rău caz.

2 Experimente

Pentru a testa diferențele de complexitate dintre cele două structuri de date, am început prin a seta prin două variabile(*numbers*, *numbersChosen*) numărul de noduri pe care le vor avea listele, precum și numărul de chei pe care le vom căuta și șterge. Apoi, am generat doi vectori cu elemente random între 0 și *numbers*, respectiv între 0 și *numbersChosen*, care vor reprezenta datele pentru operațiile de testat.

Mai întâi, am contorizat timpii pentru operațiile de inserare. Apoi, am făcut același lucru și pentru operațiile de căutare, respectiv ștergere. După câteva rulări, timpii afișați au fost:

- Elapsed time for insertion in linked list: 0.0015271 s Elapsed time for insertion in skip list: 0.0102612 s

  Elapsed time for search in linked list: 0.0868046 s Elapsed time for search in skip list: 0.0009136 s

  Elapsed time for remove in linked list: 0.0882978 s Elapsed time for remove in skip list: 0.0010878 s

- Elapsed time for insertion in linked list: 0.0009567 s Elapsed time for insertion in skip list: 0.0103072 s

  Elapsed time for search in linked list: 0.0811359 s Elapsed time for search in skip list: 0.0007813 s

  Elapsed time for remove in linked list: 0.0771205 s Elapsed time for remove in skip list: 0.000839 s

- Elapsed time for insertion in linked list: 0.0015664 s Elapsed time for insertion in skip list: 0.0118372 s

  Elapsed time for search in linked list: 0.0787872 s Elapsed time for search in skip list: 0.0008217 s

  Elapsed time for remove in linked list: 0.0744296 s Elapsed time for remove in skip list: 0.0008813 s

- Elapsed time for insertion in linked list: 0.0012084 s Elapsed time for insertion in skip list: 0.0125752 s

  Elapsed time for search in linked list: 0.078769 s Elapsed time for search in skip list: 0.0008478 s

  Elapsed time for remove in linked list: 0.0751729 s Elapsed time for remove in skip list: 0.0009162 s

- Elapsed time for insertion in linked list: 0.0009474 s Elapsed time for insertion in skip list: 0.0091014 s

  Elapsed time for search in linked list: 0.0839115 s Elapsed time for search in skip list: 0.0008677 s

  Elapsed time for remove in linked list: 0.0794266 s Elapsed time for remove in skip list: 0.0009458 s

Astfel, putem spune că *Skip List* este aproximativ cu 98% mai rapid decât *Linked List* la căutare, cu 98% mai rapid la ștergere, dar cu 570% mai încet la inserare.