Проект за курс “Структури от данни 2”

“AVL дърво и skip list. Сравнение на ефективността на структури от данни”

Автор на проекта: Румен Георгиев Азманов

ФН: 82176 Специалност: Компютърни науки Курс: 2 Група: 3

Имейл: fn82176@g.fmi.uni-sofia.bg

Резюме

В следния проект се разглежда производителността на структурите от данни AVL дърво и skip list. Използват се стандартни реализации на структурите на езика C++, а за изследването се използват примерни входни данни, аналогични на реални ситуации при употребата на структурите.

Реализация и предмет на изследване

AVL дърво

AVL дървото е йерархична структура от данни, при която всеки елемент има два наследника - ляв и десен. В структурата е установена наредба, при която стойността на всеки възел е по-голяма от тази на левия наследник и по-малка от тази на десния, което е стандартна дефиниция на BST. Равенството между елементи не се разглежда в отделен случай, но имплементационно се използва равенство и с двата наследника. Всеки възел съдържа в себе си и своята височина в дървото. Чрез височините на двете поддървета за елемент се определя неговият баланс-фактор, който е разликата между височините на лявото и дясното поддърво. В случай, че тази стойност е извън интервала [-1, +1], то се извършват така наречените ротации (лява и дясна), над тази част от дървото докато не се постигне необходим баланс фактор.

Skip list

Skip list-ът е линейна структура от данни, използваща за основа свързан списък, при който всеки елемент е свързан логически със следващия. За всеки от елементите на списъка се определя по случаен начин добавяне на определен брой допълнителни указатели към последващи елементи. В реализацията на структурата се изисква елементите да са в нарастващ ред, за да се използва техника на преминаване с по-малко от линеен брой проверки до търсен елемент от списъка. При добавяне или премахване на стойност се извършва постъпкова редакция на сочещите към нея указатели. Вероятността даден елемент да е с определен брой указатели, както и максималният им брой са параметри на структурата, които влияят на нейната ефективност.

	AVL tree		Skip list	
	Средна	Най-лоша	Средна	Най-лоша
Добавяне	Θ(log(n))	O(log(n))	Θ(log(n))	O(n)
Търсене	Θ(log(n))	O(log(n))	Θ(log(n))	O(n)
Изтриване	Θ(log(n))	O(log(n))	Θ(log(n))	O(n)
Памет	Θ(n)	O(n)	Θ(n)	O(n)

Процедура и методи

За провеждането на изследването са създадени примерни тестове, които да извършат проверка за времето и ефективността на работа на структурите. Тестовете са разделени на няколко части: за вмъкване, търсене, изтриване на стойност от структурата, за заета памет и стандартни операции. За случайна структура приемаме структура от данни, която е създадена от случайно избран брой N целочислени стойности в интервала [0, N], като е възможно те са се повтарят, а при въвеждане не са сортирани.

За провеждането на опита се ползват множества с по 300 и 30000 елемента с различна наредба и брой повторения. Подобни опити се разглеждат, понеже данните имат различна структура за различен ред на добавяне. Търсенето и изтриването на елементи ползват само стандартни множества от случайно генерирани по време на тестването числа като се представят случаи, в които елементът се намира или не се намира в структурата.

Тестовете се състоят в два напълно идентични файла за AVL дърво и за skip list и използват стандартни за езика C++ методи за отчитане на времето и паметта.

Тестове:

- Въвеждане
  - Въвеждане на 300 уникални сортирани числа
  - Въвеждане на 300 уникални сортирани в обратен ред числа
  - Въвеждане на 300 уникални числа
  - Въвеждане на 300 случайни числа
  - Въвеждане на 300 случайни повтарящи се числа
  - Въвеждане на 30000 уникални сортирани числа
  - Въвеждане на 30000 уникални сортирани в обратен ред числа
  - Въвеждане на 30000 уникални числа
  - Въвеждане на 30000 случайни числа
  - Въвеждане на 30000 случайни повтарящи се числа
  - Копиране на структура с 300 случайни числа
  - Копиране на структура с 30000 случайни числа
- Търсене
  - Търсене на 300 случайни числа
  - Търсене на 300 случайни числа (които са в структурата)
  - Търсене на 300 случайни числа (които не са в структурата)
  - Търсене на 30000 случайни числа
  - Търсене на 30000 случайни числа (които са в структурата)
  - Търсене на 30000 случайни числа (които не са в структурата)
- Изтриване
  - Изтриване на 300 случайни числа
  - Изтриване на 300 случайни числа (които са в структурата)
  - Изтриване на 300 случайни числа (които не са в структурата)
  - Изтриване на 30000 случайни числа
  - Изтриване на 30000 случайни числа (които са в структурата)
  - Изтриване на 30000 случайни числа (които не са в структурата)
  - Изтриване на структура с 300 случайни числа
  - Изтриване на структура с 30000 случайни числа
- Памет
  - Необходима памет за 300 случайни числа
  - Необходима памет за 30000 случайни числа

Резултати

След провеждането на описаните експерименти, резултатите са представени в следната таблица: (средна извадка от 10 опита):

Тест	Време / наносекунди		Време / процент от времето	
	AVL tree	Skip list	AVL tree	Skip list
Въвеждане на 300 уникални сортирани числа	148040	199410	74,24%	100,00%
Въвеждане на 300 уникални сортирани в обратен ред числа	137880	146180	94,32%	100,00%
Въвеждане на 300 уникални числа	151660	179160	84,65%	100,00%
Въвеждане на 300 случайни числа	145220	170130	85,36%	100,00%
Въвеждане на 300 случайни повтарящи се числа	140700	168340	83,58%	100,00%
Търсене на 300 случайни числа	32400	56420	57,43%	100,00%
Търсене на 300 случайни числа (които са в структурата)	28800	54840	52,52%	100,00%
Търсене на 300 случайни числа (които не са в структурата)	35990	53720	67,00%	100,00%
Изтриване на 300 случайни числа	138680	180970	76,63%	100,00%
Изтриване на 300 случайни числа (които са в структурата)	141710	191570	73,97%	100,00%
Изтриване на 300 случайни числа (които не са в структурата)	123910	132170	93,75%	100,00%
Изтриване на структура с 300 случайни числа	21960	34640	63,39%	100,00%
Копиране на структура с 300 случайни числа	57950	184320	31,44%	100,00%
Въвеждане на 30000 уникални сортирани числа	18445370	18137220	100,00%	98,33%
Въвеждане на 30000 уникални сортирани в обратен ред числа	18479130	9828600	100,00%	53,19%
Въвеждане на 30000 уникални числа	24408060	21336410	100,00%	87,42%
Въвеждане на 30000 случайни числа	25691090	21675160	100,00%	84,37%
Въвеждане на 30000 случайни повтарящи се числа	17206900	14155040	100,00%	82,26%
Търсене на 30000 случайни числа	8470730	18414770	46,00%	100,00%
Търсене на 30000 случайни числа (които са в структурата)	7259500	17183350	42,25%	100,00%
Търсене на 30000 случайни числа (които не са в структурата)	9260430	17261460	53,65%	100,00%
Изтриване на 30000 случайни числа	27272300	25945180	100,00%	95,13%
Изтриване на 30000 случайни числа (които са в структурата)	24014640	21135420	100,00%	88,01%
Изтриване на 30000 случайни числа (които не	26519660	28708350	92,38%	100,00%

са в структурата)				
Изтриване на структура с 30000 случайни числа	2814380	3136470	89,73%	100,00%
Копиране на структура с 30000 случайни числа	3897340	17430380	22,36%	100,00%
Тест	**Памет / байтове**		**Памет / процент от паметта**	
	AVL tree	**Skip list**	**AVL tree**	**Skip list**
Необходима памет за 300 случайни числа	7216	9652,8	74,76%	100,00%
Необходима памет за 30000 случайни числа	720016	960034,4	75,00%	100,00%

От таблицата се вижда, че за въвеждане на елементи, за skip list има особено значение (спрямо AVL дървото) дали данните са сортирани и в какъв ред. Списъкът създава структурата на данните по-бързо от дървото, само когато те са подредени в обратен ред и когато има повтарящи се данни.

При експериментите с търсене на стойност дървото винаги изпълнява функцията за около 40-60% от времето, необходимо на списъка. Тук особеността е, че търсенето на елементи, които не се намират в множеството е най-бавната задача за AVL tree, докато за skip list разликите във времето спрямо вида на търсения елемент са по-малки.

Понеже изтриването се базира главно на търсене на елемент, резултатите са аналогични на тези от предходните тестове. Особеност обаче е необходимостта на дървото да прилага ротации, за да запази баланс, което съществено забавя операциите.

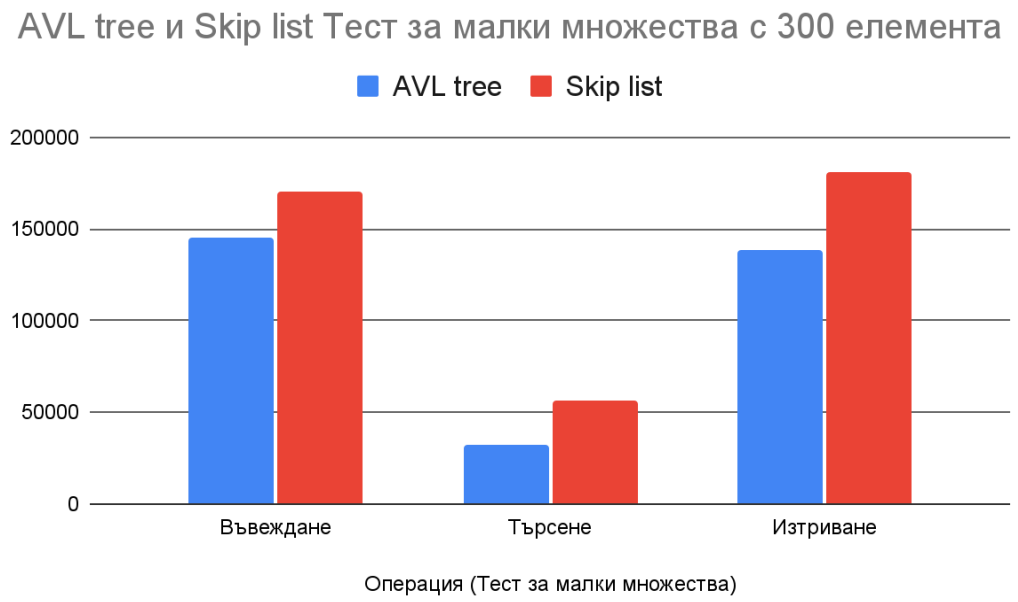
Независимо от броя елементи, AVL дървото се справя в пъти по-добре от листа в операциите за копиране и цялостно изтриване, понеже те са реализирани рекурсивно и се работи с целия елемент, за разлика от необходимостта да се обработва и масив с указатели от всеки елемент при skip list.

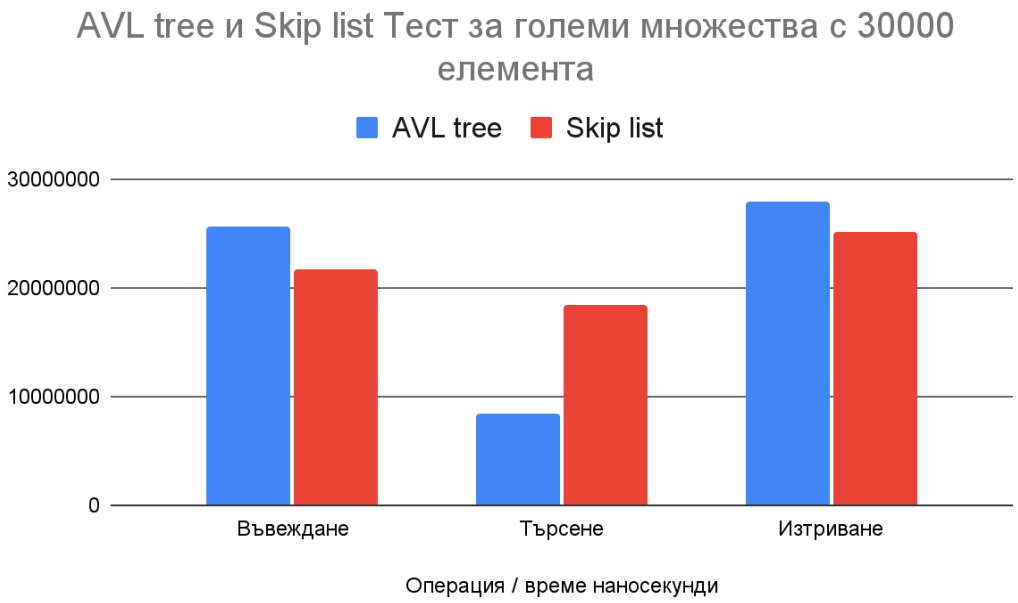
За големи множества можем да твърдим, че разликите във времето за добавяне и премахване на данни са пренебрежимо малки и двете структури се справят еднакво добре, но листът има малко преимущество.

Необходимата памет за работа на структурите е пропорционална на размера на данните. В случая е особено зависима от използваната имплементация, но и за двете структури се знае, че имат необходимост от два пъти повече указатели към други елементи спрямо броя на стойностите. За skip list е необходима повече памет, понеже структурите пазят и указател към списък с указатели.

Заключение

В заключение и двете структури имат особености - добри и лоши страни. При добавяне и изтриване на елементи можем да твърдим, че и двете са равностойни, особено ако предварително не е известна информация за тях или не са често използвани операции. Несъмнено дървовидната структура има особено преимущество при търсенето на стойности поради свойството си винаги да е балансирана и логаритмичната си сложност. Препоръчва се използването на AVL дървото, когато ще се съхраняват малък брой данни, необходимо е да се ползва по-малко памет или ще се изисква често търсене. Ако обаче се ползва голям обем стойности и главно ще се добавят и изтриват други, то skip list е структурата за предпочитане.





Бъдещи идеи

- Да се разгледа възможността за ползване на други структури от данни и алгоритми, които да са част от текущите и да подобрят тяхната ефективност. Например хеш-филтър, който частично ограничава търсенето на елементи, които не са в структурата.
- Създаване на още тестове за ефективност.
- Тестове, базирани на стандартни задачи, например за множества или статистически.
- Проучване на алгоритми за генериране на случайни числа и как те влияят на skip list.

Източници

- Лекции и упражнения УП, ООП, СДП и СД2
- cppreference.com
- cplusplus.com
- wikipedia.com
- geeksforgeeks.org

Бележки към проверяващите

- Кодът е изцяло авторски, нищо от предаденото не е копирано дело на друг автор. Повечето от алгоритмите и функциите са аналогични на изучаваните в курсовете по СДП и СД2 кодове за дървета, BST и Linked list.
- За създаването на *template* структури от данни и същевременно спазване на добрия ООП стил за разделяне на кода в отделни файлове се използва изученият по СДП метод за разполагане на кода в *.imp* файл, който се вмъква директно чрез *include* след дефиницията на класа.
- Към файловете е приложен и *catch.hpp* ползван за тестване на кода.
- В *makefile* в зависимост от използваната задача се изпълняват следните:
    - *MainTask* - пояснения по проекта, кратък пример
    - *Catch2Tests* - тестове за качеството на кода
    - *TestStructures* - тестове за ефективност на структурите, извлича информацията на екрана
    - *TestStructuresAndCreateResults* - тестове за ефективност на структурите, извлича информацията в *Results.csv* файл