Ghent University

Algoritmen en Datastructuren 3

Project Btree

Jan-Pieter Baert

Academiejaar 2019-2020

Inhoudsopgave

1 Implementatie

1.1 Algemene structuur

We definiëren eerst het aantal sleutels van de btree, dit moet een even getal zijn en noemen we *NUMBER_OF_BTREE_KEYS*.

De algemene structuur is een btree waarin we de ouder, het aantal elementen, zijn kinderen en zijn elementen bijhouden. De kinderen van een btree zijn een lijst van btree's (of een *NULL*-pointer bij bladeren), de lengte van deze lijst is *NUMBER_OF_BTREE_KEYS*+1 De elementen van een btree zijn een lijst van btreeElement's, de lengte van deze lijst is *NUMBER_OF_BTREE_KEYS*.

1.2 Keuzes

1.2.1 Grafstenen

In dit project hebben we gekozen voor grafstenen, deze grafstenen worden aangeduid door de waarde van een element gelijk te stellen aan een *NULL*-pointer. We hebben gekozen voor grafstenen omdat we in de cursus, oefening 19 (p 32 cursus AD3), zien dat het verschil in diepte vrij klein blijft. Dit zelfs voor een zeer laag aantal sleutels die geen grafsteen zijn.

1.2.2 Structuur

We hebben gekozen om de elementen rechtstreeks in de btree te steken en niet pointers ervan. De pointers in de btree steken was de initiële keuze bij de implementatie, maar we merkten dat de elementen rechtstreeks in de btree te steken sneller ging. De reden dat dit sneller is, is omdat alle elementen, van een laag in de btree, in 1 cache blok geladen kunnen worden, dit zorgt voor verbeterde lees- en schrijfsnelheden. Om dit optimaal te gebruiken is *NUMBER_OF_BTREE_KEYS* een berekening die het grootst mogelijk aantal sleutels in een btree zal nemen voor de cachegrootte meegegeven aan de compiler als *CACHESIZE*.

2 Caching

2.1 Dynamisch vs. statische btree's

Om statische btree's te simuleren hebben we een optie bij het genereren van testen om eerst al de toevoegoperaties te doen en pas daarna de andere operaties (waarbij de btree dus niet meer moet veranderen). We gebruiken een cache configuratie met *CACHESIZE*=8192, *CACHEASSOC*=8, *CACHELINESIZE*=64B.

grootte	100000	200000	300000	400000	500000
statisch	17.0%	18.3%	19.0%	18.6%	18.3%
dynamisch	15.1%	16.7%	17.3%	17.8%	18.1%

Tabel 1: Het verschil in cache miss % in statische tov. dynamische btree's

In tabel 1 zien we dat er weinig verschil op te merken is tussen het statisch en dynamisch zijn van een boom. Een dynamische input zorgt wel voor een gemiddeld kleinere boom bij de instructies, dit kan het kleine verschil tussen statisch en dynamisch verklaren.

2.2 Invloed van cachegrootte

Om de invloed van de cachegrootte te bekijken zullen we een dynamische en statische input (we bekijken voor beiden de testen van grootte 100000) kijken naar het % cache misses.

cachegrootte (in bytes)	512	1024	2048	4096	8192
statisch	14.9%(660M)	9.3%(236M)	8.7%(129M)	11.8%(938M)	16.9%(641M)
dynamisch	14.2%(316M)	8.9%(115M)	8.6%(67M)	10.9%(515M)	15.1%(368M)

Tabel 2: Het verschil in cache miss % (en totaal aantal refs) tussen verschillende cache sizes

In tabel 2 zien we dat 2048 bytes de beste configuratie blijkt te zijn om het aantal cache misses zo klein mogelijk te houden. We merken ook dat het totaal aantal refs ook het minst is bij 2048 bytes en dat het enorm hoog is bij 4096 bytes.