Aufgabe 1a)

Am 29.04.2015 um 20:21 schrieb Flo:

Guten Abend,   
  
gestern als ich hier reingeschaut habe, hat sich das, was hier drin stand mal überhaupt nicht mit meinen Gedanken vertragen. Jetzt läufts aber langsam in die gleiche Richtung.   
  
Ich habe die Aufgabe auch so gelesen, dass das neue Array erstellt wird, sobald 2^k\*m - 1 Felder im Array belegt sind. Das letzte Element zeigt auf das nächste freie Feld, was ja dann im nächstgrößeren Array erst zu finden wäre.   
  
Warum sollte der BestCase sein, wenn alles vollgeschrieben ist? Pro Eintrag fallen doch Kosten an. Auch wenn die Zugriffzeit pro Eintrag O(1) ist, muss jedes Feld im Array beschrieben werden. Bei einem sehr großen Array braucht es auch bei O(1) sehr lange, die einzelnen Felder zu lesen, denn neuen Speicher zu allokieren und die bisherigen Felder in die neuen Felder des neuen Arrays zu schreiben.   
  
Meiner Meinung nach ist der BestCase für n=0 und fertig. Alle anderen n sind schlecht, da dort dann Arbeit anfällt. Und das schlimmste ist genau dann, wenn n = 2^k\*m -1 , da dann das nächste Array aufgebaut werden muss. Somit sind aber dann auch {(2^(k+1)\*m)/2} +1 Elemente im neuen Array frei, sofern für das alte Array k und für das neue Array dan k+1 gilt.

muss mich korrigieren. Es sind natürlich {2^(k+1)\*m}/2  -1 Felder frei.

Hallo, das klingt schon sehr gut...   
  
Den worst case haben Sie richtig beschrieben:   
  
Gehen wir mal davon aus, dass in alle array Plätze Daten aufgenommen werden. Man also nicht, wie bei Ihnen beschrieben, noch eine Zeigerstelle verwaltet. Alle diese Daten müssen umkopiert werden, wenn das array voll ist. Also tritt der worst case doch dann ein, wie von Ihnen beschrieben, wenn man das letzte mal verdoppelt hat und nur noch ein weiteres Element in das neue array eingefügt wird. D.h. N ist analog zu Ihrer Argumentation als 2^k\*m +1 darstellbar, bei k+1 Verdoppelungen.   
Entsprechend ergibt sich der Betrag der nicht genutzten Plätze gemäß Ihrer Argumentation.

--   
Mit freundlichen Grüßen   
  
Holger Helmut Hennings   
Kursbetreuung

OK, dann bleibt nun noch die Berechnung der Kostenfunktion...die scheint ja noch nicht ganz richtig zu sein....also nochmal neu:   
  
Entgegen meiner ersten Annahme wird das Array erst dann verdoppelt, wenn das 2^k+m-te Element eingefügt wird.   
Bei nochmaligem Durchlesen der Aufgabenstellung denke ich außerdem, dass jeder Zuweisungsvorgang eines Elements S kostet, genauso jeder Auslesevorgang enes Elements. Ein Umkopiervorgang eines Elementes kostet also 2S.   
  
Damit berechne ich zuerst mal die Gesamtkosten für die Verdoppelungen KDoppel in Abhängigkeit von der Zahl der Verdoppelungen k und komme auf   
KDoppel(k) = 2\*(2^k-1)mS   
  
N ist im worst case direkt nach einer Verdoppelung N=2(k-1)m+1, das forme ich nach k um und setze das in die KDoppel-Funktion ein, um die Gesamtkosten für die Verdoppelungen in Abhängigkeit von N zu bekommen, und das ist dann   
KDoppel(N) = (4N-2m-4)\*S   
  
Zum Schluss muss ich noch die Kosten für das erstmalige Einfügen jeden Elementes in das Array berücksichtigt habe, die wären N\*S, d.h. Kges=(5N-2m-4)\*S.   
(Jetzt frage ich mich, warum da eine 4 drinsteht, wie erklärt sich die? Vermutlich irgendwo einmal k statt k-1 als Exponent?!?)   
  
Größenordnungsmäßig passt das aber gut zu meinem ersten Ansatz weiter oben in diesem Thread, da kam ich auf K(N)=2N, wobei ich die initialen Einfügekosten nicht berücksichtigt hatte und für das Umkopieren eines Elementes die Kosten 1S statt 2S gesetzt hatte.