

HEIMADÆMI 3

TÖL203G Tölvunarfræði 2

Kári Hlynsson¹

Háskóli Íslands

2. febrúar 2023

Verkefni 1

Breytið `FourSum.java` í `FourSumFast.java` á sama hátt og er gert með `ThreeSumFast.java`. Skilið kóða fallsins `count` (sem texta, ekki skjáskoti) og skjáskoti af keyrslu `FourSum` og `FourSumFast` á gagnaskránni `1Kints.txt`. Þá eiga að finnast 13654 ferndir. Athugið að þið þurfið að laga kóðann aðeins, því `texttttFourSum.java` notar `long` fylki í stað `int` fylkis og innlestur gagnanna er aðeins ólíkur.

Lausn

Breytingarnar eru lítillegar og sjást fyrir neðan í forriti 1.

```
49     public static int count(long[] a) {
50         int N = a.length;
51         Arrays.sort(a);
52         int cnt = 0;
53         if (containsDuplicates(a))
54             throw new IllegalArgumentException("array contains duplicates");
55         for (int i = 0; i < N; i++) {
56             for (int j = i + 1; j < N; j++) {
57                 for (int k = j + 1; k < N; k++) {
58                     int l = Arrays.binarySearch(a, - (a[i] + a[j] + a[k]));
59                     if (l > k) cnt++;
60                 }
61             }
62         }
63         return cnt;
64     }
```

Forrit 1: Fallið `count` í `FourSumFast.java`

Mynd 1 sýnir keyrslu í skel á `FourSum.java` og síðan `FourSumFast.java`.

¹Slóð á Github kóða: <https://github.com/lvthnn/TOL203G/tree/master/HD3>

```
*[master] [~/Github/TOL203G/HD3/src]$ java FourSum < 1Kints.txt
13654
*[master] [~/Github/TOL203G/HD3/src]$ java FourSumFast < 1Kints.txt
13654
*[master] [~/Github/TOL203G/HD3/src]$ █

[0] 0:nvim- 1:zsh* "vr2t-dw200.rhi.hi.is" 11:21 02-Feb-23
```

Mynd 1: Keyrsla FourSum og FourSumFast í skel

Eins og má sjá fáum við 13654 sem er viðbúinn fjöldi fernda.

Verkefni 2

Framhald af æfingadæminu að ofan.

- (a) Finnið raunhæf neðri mörk á vaxtarhraða keyrslutíma reiknirits sem leysir þetta verkefni, þ.e. hversu margar aðgerðir þurfa öll reiknirit að nota til þess að leysa þetta verkefni (sem fall af N)? Rökstyðjið svarið í nokkrum orðum.
- (b) Það er hægt að leysa verkefnið á mun hraðvirkar hátt en gert var í æfingadæminu, með því að nýta sér fyrri útreikninga í B . Hugmyndin er þið eruð að reikna út $B[i, j]$ þá eruð þið nýbúin að reikna út $B[i, j-1]$. Er ekki hægt að nota það gildi? Útfærið þetta reiknirit í Java og keyrið það fyrir sömu gildi á N og gert var í æfingadæminu. Hver er vaxtarhraði þessa nýja reiknirits? Skilið kóðanum (sem texta, ekki skjáskoti) og svarinu.

Lausn

Hluti (a)

Látum $\sigma(i, j) := \sum_{k=i}^j a_k$.² Við getum sett upp töflu sem sýnir hvernig fylkið B lítur út fyrir gefna inntaksstærð N :

$\downarrow i \rightarrow j$	1	2	...	N
1	$\sigma(1, 1)$	—	...	—
2	$\sigma(2, 1)$	$\sigma(2, 2)$...	—
\vdots	\vdots	\vdots	\ddots	\vdots
N	$\sigma(N, 1)$	$\sigma(N, 2)$...	$\sigma(N, N)$

Tafla 1: Útlit fylkisins B .

Við miðum almenna kostanaðarlíkanið út frá fjölda fallakalla á $\sigma(i, j)$. Við sjáum að heildafjöldi kalla er $1 + 2 + \dots + N$ svo við fáum

$$T(N) = 1 + 2 + \dots + N = \frac{N(N+1)}{2} \sim \frac{1}{2}N^2$$

m.ö.o. er $T(N) \sim \Omega(N^2)$.

²Við notum hér bilið $1, \dots, N$ í stað $0, \dots, N-1$ eins og venja er fyrir í stærðfræðilegum rithætti fyrir runur. Þá er inntakið okkar heiltöluruna á forminu a_1, \dots, a_N .

Hluti (b)

Við skulum hefja umfjöllunina á upprunalega fallinu. Ef við útfærum sauðakóðann sem er gefinn í æfingadæminu í Java fáum við eftirfarandi forritsbút:

```
13  public static int[] [] arraysum(int[] A) {
14      int N = A.length;
15      int[] [] B = new int[N][N];
16
17      for (int i = 0; i < N; i++)
18          for (int j = i; j < N; j++)
19              for (int k = i; k <= j; k++)
20                  B[i][j] += A[k];
21      return B;
22  }
```

Forrit 2: Fallið arraysum (hægari útfærsla)

Í þessari aðferð útfærum við hlutsummufallið $\sigma(i, j)$ með því að ítra í gegnum bilið með $i \leq k \leq j$, sækja gildið a_k hverju sinni og leggja við b_{ij} . Þessi aðferð er línuleg, þ.e. kostnaður hennar er N á heildina litið og því er tímaflækja þessa forritsbúts $T(n) \sim \mathcal{O}(N^3)$.

Hin útfærslan er mun hraðvirkari. Hún gengur þannig fyrir sig að við ítrum sem áður í gegnum fylkið. Ef við erum í hornalínustaki er það fyrsta stakið sem leggur eitthvað til summunnar og því setjum við einfaldlega $B_{ij} = A_i$ ef $i = j$.³ Ef svo er ekki þá setjum við $B_{ij} = B_{i,j-1} + A_j$ því við höfum þegar reiknað $B_{i,j-1}$. Forritsbúturinn er gefinn fyrir neðan:

```
31  public static int[] [] arraysum_fast(int[] A) {
32      int N = A.length;
33      int[] [] B = new int[N][N];
34
35      for (int i = 0; i < N; i++)
36          for (int j = i; j < N; j++)
37              B[i][j] = (i == j) ? A[i] : B[i][j - 1] + A[j];
38      return B;
39  }
```

Forrit 3: Fallið arraysum_fast (hraðari útfærsla)

³Þetta getur allt að eins verið A_j því $A_i = A_j$ því $i = j$.

```

*[master] [~/Github/TOL203G/HD3/src/V2]$ java -Xmx13G ArraySum 100000 10 fast > data_fast.csv
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at ArraySum.arraysum_fast(ArraySum.java:35)
    at ArraySum.timeFunc(ArraySum.java:83)
    at ArraySum.main(ArraySum.java:103)
*[master] [~/Github/TOL203G/HD3/src/V2]$ cat data_fast.csv
    N      T
    1      0.0
    2      0.0
    4      0.0
    8      0.0
   16      0.0
   32      0.0
   64      0.0
  128      0.0
  256      0.0
  512      0.0
 1024      0.0
 2048      0.0
 4096      0.0
 8192      0.1
16384      0.2
32768      1.9
*[master] [~/Github/TOL203G/HD3/src/V2]$

```

[1] 0:nvim- 1:zsh* "Karis-MacBook-Air-2.1" 21:57 02-Feb-23

Mynd 2: Keyrsla á ArraySum.java og söfnun gagna

Útfærslan í forriti 3 tryggir $\sigma(i, j) \sim \mathcal{O}(1)$ þ.e. summuaðgerðin er fasti hverju sinni svo við búumst við því að $T(N) \sim \mathcal{O}(N^2)$ fyrir þessa útfærslu. Við göngum úr skugga um þetta með mælingum.

Látum $T_s(N)$ tákna tímaflækju meintu hægari útfærslunnar en $T_f(N)$ tákna meintu hraðari útfærsluna. Við spáðum fyrir að $T_s(N) \sim \mathcal{O}(N^3)$ og að $T_f(N) \sim \mathcal{O}(N^2)$. Við keyrum notendaforritið í skel og beinum staðalúttakinu í csv skrá til frekari úrvinnslu, eins og mynd fyrir neðan sýnir.

Mynd fyrir neðan sýnir keyrsluna á ArraySum.java. Notuð var stillingin -Xmx13G til að gefa forritinu 13 GB til keyrslu en þá getum við fengið aðeins meiri gögn. Hið sama var endurtekið á nýjan leik fyrir hægari útfærsluna.

Verkefni 3

Tiltekið hótél hefur N herbergi, sem eru í röð á löngum gangi. Herbergi 0 er næst móttökunni, en herbergi $N - 1$ er lengst í burtu. Öll herbergin frá 0 til $F - 1$, en herbergi F til $N - 1$ eru laus. Við viljum sjálf vera í herbergi F , en við vitum ekki gildið á F (aðeins að $F < N$). Til þess að finna fyrsta lausa herbergið getum við aðeins kannað eitt herbergi í einu með því að banka á hurðina og kíkja inn. Við viljum lágmarka fjölda skipta sem við bönkum á hurðir í versta tilfalli.

- (a) Hver er versta tilfellis tími (sem fall af N) á reikniriti sem byrjar á herbergi 0 og rekur sig út eftir ganginum þar til fyrsta lausa herbergið er fundið?
- (b) Lýsið reikniriti sem notar í versta falli $\log N$ tíma til að finna fyrsta lausa herbergið.
- (c) Ef N er mikið stærra en F , þá er hægt að gera betur og nota aðeins $\sim 2 \log F$ tíma til þess að finna fyrsta lausa herbergið. Lýsið þessari aðferð og rökstyðjið vaxtarhraða hennar.

Lausn

Hluti (a)

Ef við gefum okkur að aðferðin er að fara hurð eftir hurð eftir ganginum fæst versta tilfallið þegar $F = N - 1$. Þá er tímaflækjan nokkurn veginn $\sim N$.

Hluti (b)

Til þess að útfæra þetta notum við afbrigði af binary search.