Analiza Algoritmilor de Sortare

Student: Tavă Andrei-Daniel

Grupa: 133

Informatii

- Limbajul utilizat a fost C++(14), iar compilatorul a fost MVSC configurat pentru viteza.
- Algoritmii studiati au fost Merge Sort, LSD Radix Sort, Shellsort, Bucket Sort si Quicksort(+altii), implementati relativ eficient.
- A fost masurat efectul tuturor parametrilor, precum baza Radixului, gap sequence pentru Shellsort, algoritmul de sortare al Bucketului si metoda de alegere a pivotului pentru Quicksort.

Compiler Flags:
/permissive- /ifcOutput "x64\Release\" /GS /GL /W3 /Gy /Zc:wchar_t /Zi /Gm- /Ox /sdl /Fd"x64\Release\vc143.pdb" /Zc:inline /fp:precise /D "NDEBUG" /D "_CONSOLE" /D "_UNICODE" /D "UNICODE" /errorReport:prompt /WX- /Zc:forScope /Gd /Oi /MD /FC /Fa"x64\Release\" /EHsc /nologo /Fo"x64\Release\" /Ot /Fp"x64\Release\SDLab1.pch" /diagnostics:column

Testul general

- Pentru compararea generala dintre algoritmi, am selectat 9 teste variate.
- Algoritmii sunt configurati pentru a maximiza eficienta pe testele mari: Radixul este in baza 2^16, Shellsort foloseste secventa Ciura extinsa, Size/100 bucketuri sortate prin Insertion Sort pentru Bucket Sort, iar Quicksortul alege pivotul drept mediana din 3.
- Ulterior voi analiza fiecare algoritm si voi justifica alegerea.

Size Max Tip	10^4 10^6 Random	10^5 10^7 Random	10^6 10^8 Random	10^7 10^9 Random	10^4 10^12 Random	10^7 10^3 Random	10^8 10^10 Random	10^8 10^10 Cresc.	10^8 10^10 Desc.
STL	0.0004285s	0.0053549s	0.064943s	0.758069s	0.0004207s	0.305081s	8.69882s	1.33897s	1.50771s
Merge	0.0005979s	0.007256s	0.0958669s	0.992879s	0.0005851s	0.662503s	11.2714s	3.85066s	4.27372s
Quick Med3	0.0004357s	0.0057798s	0.0640137s	0.718356s	0.000443s	25.6452s	8.27098s	1.9521s	4.80267s
Shell Ciura Ext	0.0008017s	0.0110346s	0.151977s	1.90204s	0.0007858s	1.35215s	28.7031s	12.7747s	14.251s
Radix LSD 2^16	0.0004675s	0.0015796s	0.0180399s	0.191523s	0.0005823s	0.0577756s	4.08818s	4.17673s	4.13607s
Bucket Size/100 Ins	0.0004777s	0.0041856s	0.0480479s	0.72274s	0.0005068s	0.156892s	10.3454s	2.44205s	4.79915s

Interpretarea rezultatelor

- (Sper sa pot prezenta detaliat la laborator, aici o sa scriu sumar)
- Trendul general observat: Radix este, de obicei, cel mai rapid, iar Shell Sort este constant cel mai lent.
- Nu am reusit sa optimizez Shell Sortul mai mult de atat, dar nu cred ca poate fi mai rapid decat un Merge Sort optim(mai mult despre asta in sectiunea Shell Sort)
- Algoritmii bazati pe numarare(Radix) nu sunt influentati de ordinea inputului.

Merge Sort

- Complexitatea timp O(N*logN)
- Merge Sort a avut o performanta decenta, fiind constant cu aproximativ 30% mai lent decat STL Sort.
- Implementarea mea are complexitatea de memorie O(N)
- Merge Sort nu are parametri de modificat (maxim as fi putut seta o dimensiune de la care elementele sa fie sortate de alt algoritm si sa compar in functie de algoritmul folosit... dar probabil Insertion Sort ar fi fost optim)

Quicksort

- Complexitatea variaza mult in functie de datele de intrare si de alegerea pivotului, intre O(N*logN) si $O(N^2)$.
- Alegerea random este decenta in practica, dar este incerta din motive evidente.
- Alegerea mijlocului drept pivot pare buna, si este in cele mai multe cazuri, dar exista cazuri particulare pentru care este ineficienta.

Size Max Tip	10^6 10^8 Random	10^6 10^8 Ascending	10^6 10^8 Descending
Mediana 3	0.0052586s	0.0012945s	0.002376s
Random	0.008311s	0.0054751s	0.0049876s
Mijlocul	0.0052946s	0.0013017s	0.0014071s
Primul	0.0050314s	2.36364s	2.47347s
Ultimul	0.0050831s	2.58358s	2.47309s

- Nici Mediana din 3 (in cazul meu dintre inceput, mijloc si final) nu este perfecta.
- Quicksort a avut o performanta buna, fiind aproape de si uneori intrecand STL Sort, cu exceptia cazului in care datele au multe repetitii (size>>>max), caz in care performanta este oribila.

Shell Sort

- Complexitatea depinde foarte mult de gap sequence-ul folosit, doar cel mai bun caz este O(N*logN).
- Ciura este {1,4,10,23,57,132,301,701,1750}.
- Ext Ciura este extinderea sa prin relatia $A_{k+1}=floor(2.25*A_k)$, de la 1750.
- Tokuda este definita prin relatia $A_{k+1}=ceil(2.25*A_k+1)$ cu $A_1=1$.
- Knuth este definita prin $A_k=(3^k-1)/2$ cu $A_1=1$.
- Shells este definita prin $A_k=2^{k-1}$ cu $A_1=1$.
- None corespunde Insertion Sortului.
- Toate secventele in afara de Ciura au fost generate pana la ultimul termen mai mic decat 10^8.

Size Max Tip	10^6 10^8 Random	10^7 10^8 Random
Ext. Ciura	0.154556s	1.94731s
Tokuda	0.148425s	2.01622s
Knuth	0.146782s	2.18383s
Shells	0.856102s	28.2799s
Ciura	0.218568s	30.8363s
None	MULT	MULT

- Dintre algoritmii studiati, Shell Sort este cel mai lent. Acest lucru este de asteptat, dat fiind ca Shell Sort are un average case mai lent decat restul O(N^a) cu 1<a<2.
- Implementarea este generala si deci nu e perfecta pentru fiecare sequence(ex: secventa Shell ar fi mai rapida daca ar fi implementata prin injumatatirea size-ului, dar nu ar depasi celelalte secvente).

LSD Radix Sort

- Radix Sort are o complexitate de timp de O(N*logM) si de spatiu de O(N + B), unde M este maximul si B este baza. Baza de numarare aleasa determina baza logaritmului si influenteaza semnificativ viteza, dar si memoria folosita.
- De asemenea, bazele ce sunt puteri ale lui 2 pot fi implementate cu operatii pe biti, fiind astfel si mai rapide.
- Nesurprinzator, Radix Sortul are cea mai buna performanta de obicei.

Size Max Tip	10^8 10^11 Random	10^8 10^13 Random	10^8 10^18 Random
Baza 10	14.3433s	18.0087s	28.2093s
Baza 16	12.7714s	15.2921s	23.5696s
Baza 2^{16}	6.3174s	6.76301s	9.45439s
Baza 2^{16}(biti)	4.74732s	4.75406s	6.26087s

Bucket Sort

- Bucket Sort este probabil cel mai variat algoritm de sortare prezentat, putand fi modificate atat numarul de buckets cat si algoritmul folosit pentru sortarea bucketurilor.
- Calculul complexitatii depinde mult de algoritmul de sortare utilizat.
- Un numar prea mic de buckets duce la un bucket-size mare ceea ce incetineste algoritmul. Un numar prea mare induce overhead.
- Worst-case pentru Bucket Sort este cand elementele nu sunt uniform distribuite si astfel ajung in acelasi bucket, caz in care complexitatea este dominata de algoritmul folosit.

- In urma testelor mele, numarul ideal de buckets pare sa fie size/100.
- Algoritmul cu cea mai buna performanta in bucket sort este Insertion Sort.
- Acest lucru este de asteptat, un numar de buckets de size/100 cu elemente distribuite uniform ar duce la un size de aproximativ 100 per bucket, dimensiune ideala pentru Insertion Sort.

Size Max Tip	10^6 10^10 Random	10^7 10^10 Random	10^8 10^10 Random
10^5 Insertion	0.180678s	0.742835s	13.7474s
10^6 Insertion	0.323179s	2.42305s	11.0841s
10^7 Insertion	0.484992s	3.90739s	38.0561s
Size/100 Insertion	0.0516189s	0.809323s	11.0024s
Size/100 Shell	0.0706704s	0.908055s	13.3601s
Size/100 Merge	0.0691892s	0.886398s	12.6808s
Size/100 Quick	0.0592756s	0.809832s	12.0019s

Aici se incheie analiza mea.
Multumesc de vizionare.