DT019G Objektbaserad programmering i C++

Laboration: Undersökning av sorteringsalgoritmer

Martin Kjellqvist

sorts.tex 985 2013-04-29 13:48:24Z martin

Innehåll

1 Introduktion **1**

2 Syfte **1**

3 Terminologi **2**

4 Uppgift **2**

5 Jämförelsemetod **2**

6 Examination **2**

1 Introduktion

Sortering är ett välbeökt bryderi i datatekniken. Sortering kommer i två huvudsakliga varianter, sortering baserad på jämförelser, och sortering baserad på klassificering.

Jämförelsebaserad sortering är vanligast och baseras på en enkelt beskriven ordning

Algorithm 1 Ordna två element

if $a < b$ **then**
  a läggs före b i strukturen
else
  b läggs före a i strukturen
end if

Till jämförelsebaserad sortering hör Bubblesort, Selection sort, Insertion sort, Quicksort, Mergesort, Introsort och många andra välkända sorteringsmetoder.

Alternativet till en jämförelsebaserad sortering utnyttjar andra sätt att beskriva storleken hos det som ska sorteras. Det metoderna har gemensamt är att de inte utnyttjar jämförelser. Vanliga metoder är Counting sort, Radix sort och Bucket sort.

2 Syfte

Vi kommer i labben att i detalj undersöka hur olika jämförelsebaserade sorteringsmetoder beter sig för olika datastrukturer.

En beskrivning av sorteringsalgoritmerna finner du både i kurslitteraturen och på **?**].

Notera att det redan finns olika sorteringsmetoder implementerade i standardbiblioteket, bland annat `std::sort`. Om du behöver sortera data i ett eget program är det troligast att du vill använda någon av de metoder som finns i standardbiblioteket. `std::sort`.

Innan du fortsätter med instruktionen bör du läsa igenom en beskrivning av sorteringsmetoderna.

3 Terminologi

För att beskriva sorteringsmetoder används flera olika termer.

container En struktur som innehåller element som ska sorteras.

inplace/in-situ sortering Sorteringsmetoder som arrangerar elementen direkt i containern.

sorterad En container med ett eller flera element. Om en container innehåller ett element är den sorterad. Om en container innehåller flera element gäller $e_i <= e_{i+1}$ för samtliga element.

tidskomplexitet Ett mått på det antal operationer en sorteringsmetod kräver för att ordna alla element. Här använder man O-notationen för att ge en övre gräns som beskriver antalet operationer som krävs. N beskriver antalet element i containern. Exempelvis kan man undersöka om en container är sorterad i $O(N)$ operationer. En jämförelse per element.

4 Uppgift

Du skall jämföra två egenhändigt skrivna sorteringsmetoder med standardbibliotekets `std::sort`. Du ska implementera en av metoderna med länkade listan från föregående lab och en med den givna IntArray-klassen. Du har att välja mellan följande sorteringsmetoder:

Selection sort Urvalssortering eller Sök minsta-byt. En enkel och intuitiv in-situ sorteringsmetod som har tidskomplexitet $O(N^2)$. IntArray.

Insertion sort Insticksortering. En enkel sortering som brukar liknas vid ett intuitivt sätt att sortera kort i en korthand. Effektiv för få element som är någorlunda ordnade.i In-situ, $O(N^2)$. Länkad lista.

Heap-sort En effektiv sorteringsmetod som på ett finurligt sätt arrangerar elementen i containern i en heap-struktur. Kan göras in-situ. $O(NlogN)$. IntArray.

Merge-sort En effektiv sorteringsmetod som använder sig av extra utrymme för att göra en merge-operation. $O(NlogN)$. Länkad lista.

std::sort En sorteringsmetod som har tidskomplextitet. $O(NlogN)$. Kräver en array-liknande struktur.

IntArray och Länkad lista givet efter varje sorteringsmetod är containerförslag. Det går utmärkt att implementera samtliga metoder i båda strukturerna. Heapsort kan bli lite extra knölig med en länkad lista.

Din kod ska vara strukturerad så att funktioner är samlade i relaterade headerfiler med separata cpp-filer. Koden ska vara lättläst med förtydligande kommentarer både för funktionsdeklarationer och programkod.

5 Jämförelsemetod

I huvudprogrammet ska du jämföra hur snabbt de olika metoderna tar att sortera olika stora containers. Välj olika storlekar N så att du får tidsmätningar som ger vettiga resultat. Lämpliga storlekar kan vara $N = 1000, 10000, 100000$ element.

Fyll dina containers med slumpvärden. `rand()%10000` kan vara lämpligt. Använd bifogade Timer.h och Timer.cpp [**?**] för tidsmätningarna. Tänk på att bara mäta tiden för själva sorteringen. Förberedelser och utskrifter ska inte tas med. Resultatet av programmet ska vara en utskrift i tabellform som redovisar de tre metodernas tidsåtgång för tre olika storlekar.

För `std::sort` fyller du en `std::vector<int>` med element. Sorteringen görs sedan med `std::sort( v.begin(), v.end() );`.

6 Examination

Din lösning ska redovisas muntligen för en lärare vid något av kursens redovisningstillfällen. När du redovisat och fått godkänt laddar du upp din källkod (och byggskript) till inlämningslådan i lärplattformen.