Biostatistics 615 - Statistical Computing

Lecture 5 Basic Algorithms II

Jian Kang

September 22, 2015

Summary of previous lectures

- Computer representation of numbers: C++ and R
- Introduction to C++: basic syntax
- Basic algorithms: Insertion sorting

Merge Sort

Divide and conquer algorithm

- **Divide** Divide the n element sequence to be sorted into two subsequences of $n/2$ elements each
- **Conquer** Sort the two subsequences recursively using merge sort
- **Combine** Merge the two sorted subsequences to produce the sorted answer

mergeSort.cpp - main()

```
#include <iostream>
#include <vector>
#include <climits>
void mergeSort(std::vector<int>& a, int p, int r); // defined later
void merge(std::vector<int>& a, int p, int q, int r); // defined later
void printArray(std::vector<int>& A); // same as insertionSort
// same to insertionSort.cpp except for one line
int main(int argc, char** argv) {
  std::vector<int> v;
  int tok;
  while ( std::cin >> tok ) { v.push_back(tok); }
  std::cout << "Before sorting: ";
  printArray(v);
  mergeSort(v, 0, v.size()-1); // differs from insertionSort.cpp
  std::cout << "After sorting: ";
  printArray(v);
  return 0;
}
```

mergeSort.cpp - mergeSort() function

```
void mergeSort(std::vector<int>& a, int p, int r) {
  if ( p < r ) {// terminating condition. nothing happens when p >= r
    int q = (p+r)/2;        // find a point to divide the problem
    mergeSort(a, p, q);     // divide-and-conquer
    mergeSort(a, q+1, r);   // divide-and-conquer
    merge(a, p, q, r);      // combine the solutions
  }
}
```

mergeSort.cpp - merge() function

```cpp
// merge piecewise sorted a[p..q] a[q+1..r] into a sorted a[p..r]
void merge(std::vector<int>& a, int p, int q, int r) {
  std::vector<int> aL, aR; //copy a[p..q] to aL and a[q+1..r] to aR
  for(int i=p; i <= q; ++i) aL.push_back(a[i]);
  for(int i=q+1; i <= r; ++i) aR.push_back(a[i]);
  aL.push_back(INT_MAX);//append additional value to avoid out-of-bound
  aR.push_back(INT_MAX);
  // pick smaller one first from aL and aR and copy to a[p..r]
  for(int k=p, i=0, j=0; k <= r; ++k) {
    if ( aL[i] <= aR[j] ) {
      a[k] = aL[i];
      ++i;
    }
    else {
      a[k] = aR[j];
      ++j;
    }
  }
}
```

Time Complexity of Merge Sort

If $n = 2^m$

$$
\begin{aligned}
T(n) &= \begin{cases} c & \text{if } n = 1 \\ 2\,T(n/2) + cn & \text{if } n > 1 \end{cases} \\
T(n) &= \sum_{i=1}^{m} cn = cmn = cn\log_2(n) = \Theta(n \log_2 n)
\end{aligned}
$$

`merge`

`mergesort`

Time Complexity of Merge Sort

If $n = 2^m$

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2\,T(n/2) + cn & \text{if } n > 1 \end{cases}$$

$$T(n) = \sum_{i=1}^{m} cn = cmn = cn\log_2(n) = \Theta(n\log_2 n)$$

For arbitrary n

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn & \text{if } n > 1 \end{cases}$$

$$cn\lfloor \log_2 n \rfloor \leq T(n) \leq cn\lceil \log_2 n \rceil$$

$$T(n) = \Theta(n\log_2 n)$$

Master Theorem

For recurrent equation

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

where

- n is the size of the problem.
- $a \geq 1$ is the number of subproblems in the recursion.
- n/b, $(b > 1)$, is the size of each subproblem. We assume that all the subproblems are of the same size.
- $f(n)$ is the cost of the work done outside the recursion calls.

http://en.wikipedia.org/wiki/Master_theorem

Master Theorem (cont.)

Case 1

if $f(n) = \Theta(n^c)$ where $c < \log_b a$, then

$$T(n) = \Theta\left(n^{\log_b a}\right).$$

Example

If $T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$, then $a = 8, b = 2, f(n) = 1000n^2 = \Theta(n^c)$, where $c = 2 < \log_b a = 3$. Therefore, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$. In fact, the exact solution to the recurrent equation is $T(n) = 1001n^3 - 1000n^2$ assuming $T(1) = 1$.

Case 2

If $f(n) = \Theta(n^c \log^k n)$ where $c = \log_b a$, $k \geq 0$, then

$$T(n) = \Theta\left(n^c \log^{k+1} n\right).$$

Example

If $T(n) = 2T\left(\frac{n}{2}\right) + 10n$, then $a = 2, b = 2, f(n) = 10n = \Theta(n^c \log^k n)$, where $c = 1 = \log_b a$, $k = 0$. Therefore, $T(n) = \Theta(n^{\log_b a} \log^{k+1} n) = \Theta(n \log n)$. In fact, the exact solution to the recurrent equation is $T(n) = n + 10n \log_2 n$ assuming $T(1) = 1$.

Master Theorem (cont.)

Case 3

if $f(n) = \Theta(n^c)$ where $c > \log_b a$, then

$$T(n) = \Theta\left(f(n)\right).$$

Example

If $T(n) = 2T\left(\frac{n}{2}\right) + n^2$, then $a = 2, b = 2, f(n) = n^2 = \Theta(n^c)$, where $c = 2 > \log_b a = 1$. Therefore, $T(n) = \Theta(f(n)) = \Theta(n^2)$. In fact, the exact solution to the recurrent equation is $T(n) = 2n^2 - n$ assuming $T(1) = 1$.

Akra-Bazzi Method

For recurrent equation

$$T(x) = g(x) + \sum_{i=1}^{k} a_i T(b_i x + h_i(x)), \text{ for } x \geq x_0$$

where

- $a_i > 0$ and $0 < b_i < 1$ are constants for all i.
- $|g(x)| = O(x^c)$, where c is a constant.
- $|h_i(x)| = O\left(\frac{x}{(\log x)^2}\right)$ for all i.
- x_0 is a constant.

Then

$$T(x) = \Theta\left(x^p\left(1 + \int_1^x \frac{g(u)}{u^{p+1}} du\right)\right),$$

where p is the value satisfying $\sum_{i=1}^{k} a_i b_i^p = 1$.
http://en.wikipedia.org/wiki/Akra-Bazzi_method

Akra-Bazzi Method (Example)

If

$$T(n) = \begin{cases} n^2 + \frac{7}{4} T\left(\lfloor \frac{1}{2} n \rfloor\right) + T\left(\lceil \frac{3}{4} n \rceil\right), & \text{for integers } n > 3 \\ 1, & \text{for integers } 0 \leq n \leq 3 \end{cases}$$

then, solving

$$\frac{7}{4}\left(\frac{1}{2}\right)^p + \left(\frac{3}{4}\right)^p = 1$$

we have $p = 2$, therefore

$$\begin{aligned} T(n) &= \Theta\left(n^p \left(1 + \int_1^x \frac{g(u)}{u^{p+1}} du\right)\right) \\ &= \Theta\left(n^p \left(1 + \int_1^x \frac{u^2}{u^3} du\right)\right) \\ &= \Theta(n^2(1 + \ln n)) \\ &= \Theta(n^2 \log n). \end{aligned}$$

Running time comparison

Running example with 200,000 elements

```
user@host:~$ time sh -c 'seq 1 200000 | shuf | ./insertionSort > /dev/null'
0:17.42 elapsed, 17.428 u, 0.017 s, cpu 100.0% ...

user@host:~$ time sh -c 'seq 1 200000 | shuf | ./stdSort > /dev/null'
0:00.36 elapsed, 0.346 u, 0.042 s, cpu 105.5% ...

user@host:~$ time sh -c 'seq 1 200000 | shuf | ./mergeSort > /dev/null'
0:00.46 elapsed, 0.465 u, 0.019 s, cpu 102.1% ...
```

Summary: Merge Sort

- Easy-to-understand divide and conquer algorithm
- $\Theta(n \log n)$ algorithm in worst case
- Need additional memory for array copy
- Slightly slower than other $\Theta(n \log n)$ algorithms due to overhead of array copy

Quicksort

- Worst-case time complexity is $\Theta(n^2)$
- Expected running time is $\Theta(n \log_2 n)$.
- But in practice mostly performs the best

Quicksort

- Worst-case time complexity is $\Theta(n^2)$
- Expected running time is $\Theta(n \log_2 n)$.
- But in practice mostly performs the best

Divide Partition (rearrange) the array $A[p..r]$ into two subarrays

- Each element of $A[p..q-1] \leq A[q]$
- Each element of $A[q+1..r] \geq A[q]$

Compute the index q as part of this partitioning procedure

Conquer Sort the two subarrays by recursively calling quicksort

Combine Because the subarrays are already sorted, no work is needed to combine them. The entire array $A[p..r]$ is now sorted

http://www.sorting-algorithms.com/quick-sort

Quicksort Algorithm

Algorithm $\textsc{Quicksort}$

Data: array A and indices p and r
Result: $A[p..r]$ is sorted
if $p < r$ **then**
  $q = \textsc{Partition}(A,p,r)$;
  $\textsc{Quicksort}(A,p,q-1)$;
  $\textsc{Quicksort}(A,q+1,r)$;
end

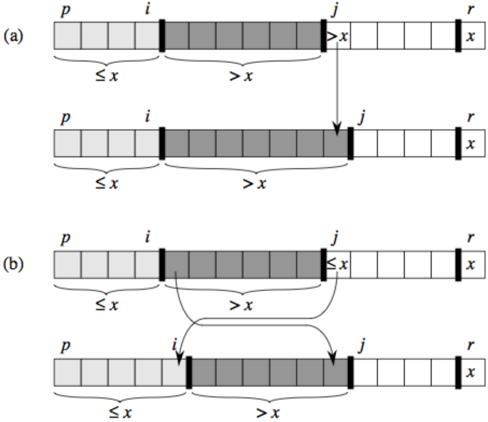
Quicksort Algorithm

Algorithm PARTITION

Data: array A and indices p and r
Result: Returns q such that $A[p..q-1] \leq A[q] \leq A[q+1..r]$

$x = A[r]$;
$i = p - 1$;
for $j = p$ **to** $r - 1$ **do**
  if $A[j] \leq x$ **then**
    $i = i + 1$;
    EXCHANGE($A[i]$,$A[j]$);
  end
end
EXCHANGE($A[i+1]$,$A[r]$);
return $i + 1$;

How PARTITION Algorithm Works



Implementation of QUICKSORT Algorithm

```
// quickSort function
// The main function is the same to mergeSort.cpp except for the function name
void quickSort(std::vector<int>& A, int p, int r) {
  if ( p < r ) { // immediately terminate if subarray size is 1
    int piv = A[r];  // take a pivot value
    int i = p-1;     // p-i-1 is the # elements < piv among A[p..j]
    int tmp;
    for(int j=p; j < r; ++j) {
      if ( A[j] < piv ) { // if smaller value is found, increase q (=i+1)
        ++i;
        tmp = A[i]; A[i] = A[j]; A[j] = tmp; // swap A[i] and A[j]
      }
    }
    A[r] = A[i+1]; A[i+1] = piv;             // swap A[i+1] and A[r]
    quickSort(A, p, i);
    quickSort(A, i+2, r);
  }
}
```

Running time comparison

Running example with 200,000 elements

```
user@host:~$ time sh -c 'seq 1 200000 | shuf | ./insertionSort \\
             > /dev/null'
0:17.42 elapsed, 17.428 u, 0.017 s, cpu 100.0% ...

user@host:~$ time sh -c 'seq 1 200000 | shuf | ./stdSort > /dev/null'
0:00.36 elapsed, 0.346 u, 0.042 s, cpu 105.5% ...

user@host:~$ time sh -c 'seq 1 200000 | shuf | ./mergeSort \\
             > /dev/null'
0:00.46 elapsed, 0.465 u, 0.019 s, cpu 102.1% ...

user@host:~$ time sh -c 'seq 1 200000 | shuf | ./quickSort \\
             > /dev/null'
0:00.35 elapsed, 0.353 u, 0.018 s, cpu 102.8%...
```

Summary: Quicksort

- $\Theta(n \log n)$ algorithm on average (and most case)
- $\Theta(n^2)$ algorithm in worst case
- Divide and conquer algorithms based on partitioning
- Slightly faster than other $\Theta(n \log n)$ algorithms

Summary

- Divide and conquer is a powerful tool for solving difficult problems.
- The master theorem and the Akra-Bazzi method are useful for time complexity analysis.
- There are many other famous divide and conquer algorithms: matrix multiplication and inversion algorithms (will be discussed later), fast Fourier transformation, eigenvalue algorithm, etc.