CI583: Data Structures and Operating Systems More probabilistic and non-deterministic algorithms

Probabilistic strategies

As well as algorithms that return an *approximate* answer, there are other significant categories of probabilistic algorithm.

These can often improve on the running time of a deterministic algorithm, or can be used to address problems for which there is no known efficient, deterministic solution:

1. Monte Carlo algorithms,
2. Las Vegas algorithms, and
3. Sherwood algorithms.

Monte Carlo algorithms

A Monte Carlo algorithm *always gives a result* but it might be wrong, like our primality test.

The probability that the result is correct increases the longer the algorithm is allowed to run.

We are only interested in those algorithms that return a correct answer more than half the time.

Monte Carlo algorithms

If there are several choices that can be made during execution, an algorithm that will always make the same choices is called consistent.

As well as allowing the algorithm to run for a longer time, we can improve a consistent algorithm by calling it multiple times and choose the answer that appears most often.

Monte Carlo algorithms

Suppose we want to discover whether an array contains a “majority element”, one which appears in more than half of the locations.

The most obvious solution, comparing every element to every other element, takes $O(n^2)$ steps.

A Monte Carlo solution would pick an element at random and check whether it appears in more than half of the array.

If the algorithm returns `true`, it is definitely correct. If it returns `false`, it might just have picked the wrong element.

Monte Carlo algorithms

However, if there *is* a majority element, the chance of picking it is more than 50% and increases the more often the element appears.

If we call `Majority` 5 times, the likelihood of it being right increases to 97% at a cost of 5n steps, making it $O(n)$.

```
procedure Majority($array$, $n$)
    $x \leftarrow$ Random$(1, n)$        ▷ Random number between 1 and n
    $count \leftarrow 0$
    for $i$ from 0 to $n-1$ do
        if $array[i] = array[choice]$ then
            $count \leftarrow count + 1$
        end if
    end for
    return $count > n/2$
end procedure
```

Las Vegas algorithms

A Las Vegas algorithm will never return the wrong answer but it might return no answer at all.

A program that uses a Las Vegas algorithm would call it repeatedly until it succeeds or gives up.

Recall the N-Queens problem, for which we presented a backtracking solution in lecture 4a.

We need to place n queens on an n $\times$ n chessboard so that none of the queens is attacked.

A Las Vegas solution would place queens randomly, row by row, and give up if there is nowhere to put a queen on the next row.

Probabilistic N-Queens

For each row, inspect each column location.

If it is not attacked, add it to a list of possible places for the next queen. If this list remains empty, give up.

If there are m possible places for this queen in the current row, place it in one of them so that each place has a $1/m$ chance of being picked.

If we get all the way to the last row, return the solution.

Probabilistic N-Queens

```
procedure LVQueens($result$) ▷ an empty array that will hold column positions
    for $row$ from 0 to 7 do
        $places \leftarrow 0$
        for $col$ from 0 to 7 do
            if $board(row, col)$ is not attacked then
                $places \leftarrow places + 1$
                if $Random(1, places) = 1$ then
                    $try \leftarrow col$
                end if
            end if
        end for
        if $places > 0$ then
            $result[row] = try$
        else
            return FAIL
        end if
    end for
    return $result$
end procedure
```

Probabilistic N-Queens

A full statistical analysis will show that the chance of success is just under 13% for 8 queens and that, on average, a failure will be detected in fewer than 7 passes.

This means that we could usually find a solution in about 55 passes.

An optimised backtracking solution to 8-Queens is better than this, but a recursive solution, for instance, uses about twice as many passes.

See `http://penguin.ewu.edu/~trolfe/QueenLasVegas/QueenLasVegas.html` for an analysis of this algorithm and a faster Las Vegas solution.

Sherwood algorithms

A Sherwood algorithm always returns the correct answer. So far, sounds deterministic!

Sherwood algorithms introduce a coin toss to deterministic algorithms in which the best, average and worst complexity differ by a large amount depending on the input.

Consider BinarySearch, in which we usually start by picking an element in the middle of the list.

In the average and best cases, this is the most sensible place to start.

Sherwood algorithms

A Sherwood BinarySearch would pick the starting position at random.

Sometimes this would pay off, meaning that we discard more than half of the elements straight away, and sometimes it wouldn't.

The name comes from Robin Hood (of Sherwood Forest), who famously robbed the rich to pay the poor: we reduce the chance of the occurrence of the worst case, at the cost of making the best case less likely.

Probabilistic strategies

We have described algorithms that return an approximate answer, such as “maybeprime”, and which improve the longer they run.

These are Monte Carlo algorithms – as long as the approximate answer, the *maybe*, is right more than half the time, we can improve the likelihood of a correct answer.

Las Vegas algorithms might not give an answer at all, but will not return the wrong one.

Sherwood techniques can be applied to any deterministic algorithm without affecting its correctness but can reduce the chance of worst-case behaviour.

Next time

Algorithms for parallel computing and Big Data.