18.337 PSET 3, Spring 2023

Due on Gradescope Wednesday 11:59pm 3/15/2023

Version 1.3 Tues March 14

Problem 1

Suppose that $f(A, B, C, D) = \mathrm{tr}(AB + CD)^{-1}$ taking four $n \times n$ matrices as inputs and a scalar output.

(1) Draw the DAG for this computation.

(2) Fill in the edges with operators that take derivatives. For example the operator for the derivative of $X \to X^{-1}$ can be written as a) dX →-X$^{-1}dXX^{-1}$ or b) $-X^{-T} \otimes X^{-1}$ or you can define your own notation. The operator for $\mathrm{tr}\, X$ could be a) $dX \to \mathrm{tr}\, dX$or b) $dX \to I \cdot dX$ (using the matrix dot product $A \cdot B \equiv tr(A^T B)$.

(3) Following the paradigm for forward mode automatic differentiation write down the gradient of $f(A, B, C, D)$ with respect to A, i.e. write down a matrix G such that $f(A + dA, B, C, D) \approx f(A, B, C, D) + G \cdot dA$.

(4) Check your answer numerically, take a screenshot or just copy code.

(5) (A little tricky) Show how to use reverse mode differentiation to obtain the same answer as in (3).

Problem 2

The length of the longest increasing subsequence of a permutation p of length n (or any sequence for that matter) is the largest number k such that $p[i_1] < p[i_2] < \ldots < p[i_k]$ for $i_1 < \ldots < i_k$. For example, if $p = [2, 3, 1, 5, 4]$, we have increasing subsequences $2, 3, 5$ and also $2, 3, 4$ but there is no increasing subsequence of length 4 so $k = 3$.

(1) Write your first program that computes the length of the longest increasing subsequence of a permutation. Call it `lis` Use any method you like: devise your own or look online. Run your program on `randperm(n)` (you will need the `Random.jl` package)

(2) Compare your program with `patiencesort1` in https://github.com/mitmath/18337/blob/master/hw3/patiencesort1.jl using `@btime` Say something about the time and allocations and show that you understand the comparison. Perhaps try a permutation of size 100 and a permutation of size 1000.

(3) Write a new routine `patiencesort2` that replaces the line `whichpile = 1 + sum` ... with a call to `searchsortedfirst` .

What can you say about timing now? Can you explain why?

(4) The `searchsortedfirst` uses a binary search algorithm. Write your own linear time algorithm. Around for what sized permutations is the linear time algorithm faster than the binary search?

Here is one possible linear algorithm:

```
function my_searchsortedfirst(pt, a)
   for i in 1:length(pt) #length(pt)
        if a < pt[i] return i end
```

```
    end
    return length(pt)+1
end
```

(5) Write a loop and gather $t = 10,000$ samples of random permutations of size $n = 6^6$ and measure the time. (Preallocate a vector of length 10,000 and store the results there)

(6) replace `randperm(n)` with `rand(n)`. Is this faster? Argue that statistically the answer is the same.

(7) Make sure you have as many threads as you can get, and parallelize the search with threads.

(8) Use the file https://github.com/mitmath/18337/blob/master/hw3/parallelhistogram.jl to compare with theory. Just use this as a blackbox – there is a so-called Fredholm determinant approach that draws the exact limiting curve when $n \to \infty$.

(9) (Optional) Speed contest, go crazy, reduce allocations, use a GPU, use a distributed memory machine, see how much data you can get in, say, 3 minutes. The winner will get a prize. Tell us what you did. Anything goes. (Judges will use whatever subjective criteria they wish.)