A short topic: Min Contig Sum 09/18

We may have a stream of data

- input is presented as a sequence of items which can be examined in only a few passes, typically 1 pass
- goal may be to compute something about the data within certain constraints

eg - single pass

- constant space, or logarithmic space, or sometimes linear

Noga Alon, Yossi Matias, Mario Szegedy 1996

Seminal paper... led to 2005 Gödel prize.

- Databases
- Networking
- Natural Language Processing

Things we might want to compute — Space

max - input is stream of integers — O( )

min - " " " — O( )

frequency

Assumption: the integers are small enough to fit in O(1) space

Min Contig Sum = "Minimum Contiguous Sum"

8 4 -1 -6 2 7 -1 2 -9 18 -3 0....

- use a Dynamic Programming approach.

Can you do it in linear space?
logarithmic space?
* Constant space?

(Assume the values can each be stored in O(1) space)

Graph streaming is an interesting area of streaming algorithm. Can be fixed $n = |V|$, with

edge-insertion-only ... stream of edges that get added to E.

edge-insertion and deletion = "Dynamic Graph Stream"

Lots of good stuff here, like Graph Connectivity
- often results are "whp" = "with high probability".

Review of Dynamic Programming

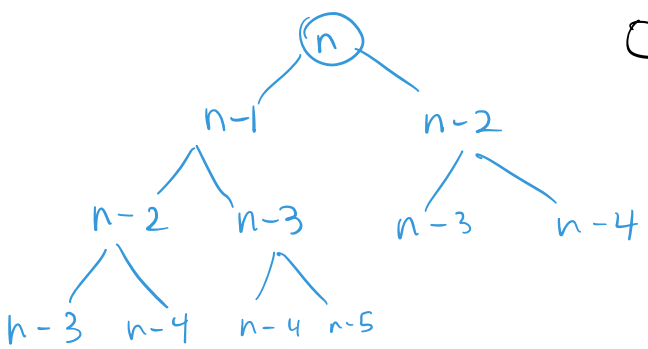
"Memoize" = remember (write down) stuff rather than recomputing it.

Eg. fib(n)

if $n < 2$ return 1

return fib(n-1) + fib(n-2)

Running time is huge ... Exponential in n



$O(\phi^n)$

↖ golden ratio

$\frac{\sqrt{5}+1}{2} \approx 1.618$

n	1.618^n
4	6.9
8	
12	322
16	
20	
24	103,629
⋮	
48	10.7 billion.

Note ∃ 2 subtrees labelled n-2
3 subtrees labelled n-3
⋮
etc

and each time we recompute the whole subtree.

DP says lets be smarter, and just
"note down" the stuff we might use later
memoize

↑ herein lies the art of DP!

```
fib(n)
    if  n == 0  or  n == 1
        Fib[n] = 1
        return Fib[n]
    else if  Fib[n] == unassigned
        Fib[n] = fib(n-2) + fib(n-1)
        return Fib[n]
    else return Fib[n]
```

DP is characterized by recursive formulas of the type:

$$X(i,j,k) = \begin{cases} X(i-1,j,k) + \dots & \text{if } \_\_\_ \\ X(i,j-1,k) + \dots & \text{if } \_\_\_ \\ \dots & \text{otherwise} \end{cases}$$

Some mix of indices

- the evaluation must be linearizable.

(can't have a cycle, like $X(3,2,1)$ depends on $X(2,2,1)$ which depends on $X(3,1,1)$ which depends on $X(3,2,1)$.)