Lec 6 More on sequences + techniques

Filter $W = O(n)$ $S = O(\log n)$

→ notation $\langle x \in A \mid F(x) \rangle$

```
filter f A =
  F = map (fn x => 1 if f(x) else 0) A
  (X, l) = scan op+ 0 F
  R = alloc(l)
  pFor i = 0..(n-1):
    if (F[i] = 1) then R[X[i]] = A[i]
  ret R
```

scan to get which index the element go

F	1	0	1	1	1	0	0	1	1	1	
X	0	1	1	2	3	4	4	4	5	6	$l=7$
R	␣	␣	␣	␣	␣	␣	␣				

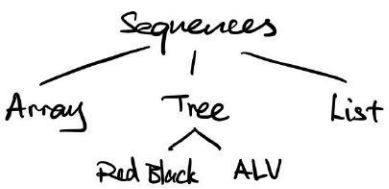
Flatten

```
flatten A =
  L = ⟨|s| | s ∈ A⟩
  (X, l) = scan op+ 0 L
  R = alloc l
  pFor i = 0..(|A|-1)
    pFor j = 0..(L[i]-1)
      R[X[i] + j] = (A[i])[j]
  ret R
```

	$\langle\langle 2,3\rangle, \langle 7,8,1\rangle, \langle 4\rangle\rangle$
map length	$\langle 2, 3, 1 \rangle$
scan (op+)	$\langle 0 2, 5 \rangle, 6$
R indexes	$\langle 0, 1, 2, 3, 4, 5 \rangle$

Sequences Abstractions

Definition



Cost model

└ A[i] work	$O(1)$	$O(\log n)$
└ append A B	$O(\|A\|+\|B\|)$	$O(\log(\|A\|+\|B\|))$

Impl detail : acth whatever

Math abstraction

Problem solving ← toolbox, connections, search

Alg design techniques

- Reduction
- Brute force
- Divide and conquer
- Contraction
- Greedy alg
- Dynamic programming

Reduction

Def Problem A is reducible to problem B if ∀ instance of A we can:
- transform the instance to some instance of B
- solve it
- transform the solution back to a solution for A

Ex. Find max in input sequence

Bad reduction: sort it, grab last elem

convert → sort → convert
O(1) O(n) O(1)

Brute-force

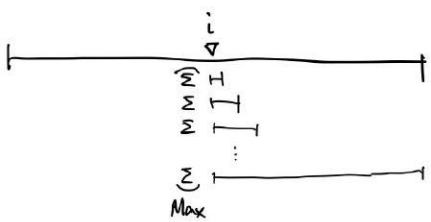
more confident, good parallelisation, use as baseline

— "Consider and check all possible solutions"

Ex. MCSS

→ Check all possible subseqs O(n³) work
then check redundant work
But wasted lots of work summing independently

Reduction: MCSS with start position i problem (MCSSwSP)

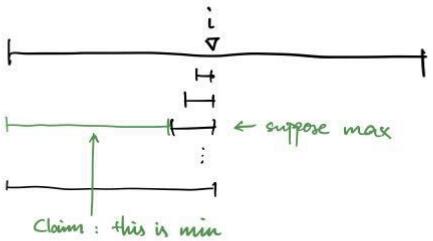


Scan then reduce — O(n) work
O(lg n) span

Then solve MCSSwSP for all i, then reduce.
O(n²) work, O(lg n) span
↑ Better! ... but still some wasted work across different start positions

(MCSSwEP)

Further reduction: MCSS with end position i problem




```
MCSSwEP A i = let
  (b, v) = scan op+ A[0, i]
  prefixSum = append b ⟨v⟩
  minPrefix = reduce min ∞ prefixSum
in
  v - minPrefix
end
```

O(n) work O(log n) span

Now redundant work have overlapping scans.

New alg

```
MCSS A = let
  (b, v) = scan op+ 0 A
  prefixSums = append b ⟨v⟩
  (minPrefixes, _) = scan min ∞ prefixSum
  maxForEnds =
      ⟨ prefixSums[i] - minPrefix[i]  | 0 ≤ i ≤ |A| ⟩
in
  reduce max -∞ maxForEnds
end
```

Yay O(n) work O(log n) span

↑

This sol took 9 years to find