Lecture 8: Lower Bounds

so far, we hardly know any lower bounds for computational problem
Besides comparison based arguments ($\Omega(n \log n)$ for sorting) or about information distribution (LOCAL graph model), we only have trivial bounds (like $\Omega(n^2)$ for matrix multiplication).
Today: lower bounds for data structures

Cell Probe Model

- computation is for free (since we don't understand it anyways)
- pay for each memory access (word size $\log n$)

stronger than RAM & pointer machines (but only useful for lower bounds)

Dynamic Partial Sum Problem

- maintain an array $A[0 \dots n-1]$ (of values polynomial in n)
- update(i, x): $A[i] = x$
- query(i): prefix sum $A[0] + \dots + A[i]$ (or any other operation over large enough group G)

Pătraşcu / Demaine 2004: $\Omega(\log n)$ worst case / amortized per operation

Proof: $A = [0, 0, \dots, 0]$, n is a power of 2

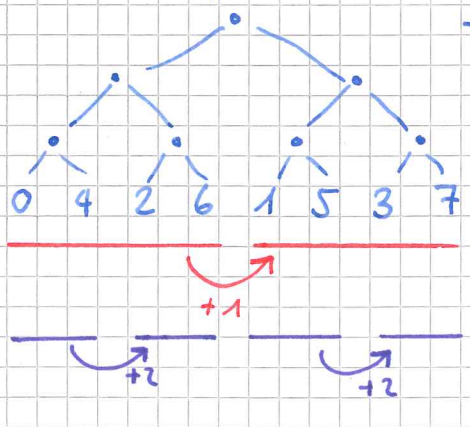
```
for t in 0...n-1:
    i = reverse-bit(t)
    query(i)
    update(i, x_t)
```

$x_t \in_{u.o.r.} G$

	reverse bit:		
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

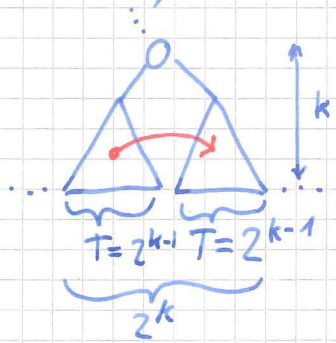
x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7

mentally build a binary tree over the access sequence (time)



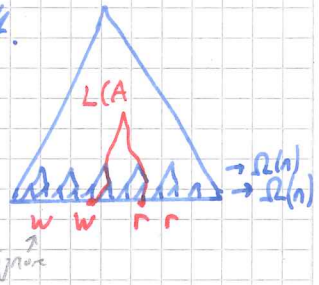
→ access sequence is „self-interleaving"
(self similar under constant shift)
but has non-trivial dependencies
(i.e. we can't just aggregate subtrees into a single value)

look at any node in the tree of time:



we want to count # cells written on the left & read on the right (and not changed in between)
↳ called <u>information transfer</u>

<u>Lemma</u>: $\Omega(T)$ memory cells are read on the right that were last written on the left.

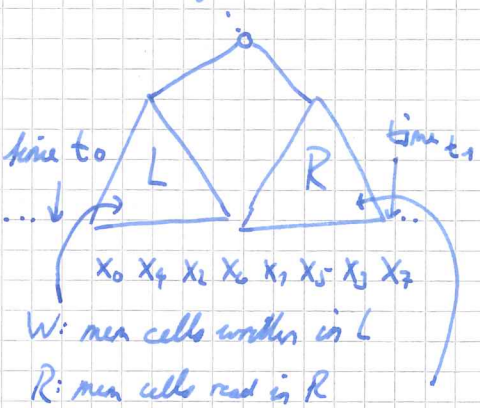


→ from this: $\Omega(n \log n)$ overall
↳ gives w.c./am. bound per op.

(we ignore the randomness here, i.e. all statements would need to be "in expectation" and more careful – but also hold for rand. alg.)

<u>Proof of the Lemma</u>

each x_i has $\Theta(\log n)$ bits of entropy
gives the state of the data structure at time t_0 and queries on R, updates on R
we can recover the updates on L



W: mem cells written in L
R: mem cells read in R

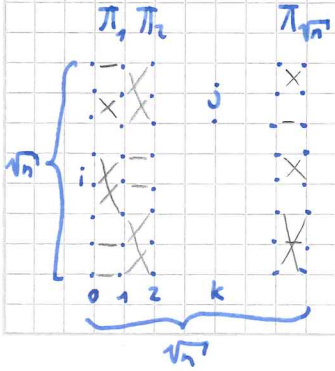
query (1): x_0 → recover x_0
query (5): $x_0 + x_2 + x_4 + \cancel{x_1}$ → recover x_4
query (3): $x_0 + x_2 + \cancel{x_1}$ → recover x_2
query (7): $x_0 + x_2 + x_4 + x_6 + \cancel{x_1} + \cancel{x_3} + \cancel{x_5}$ → recover x_6

If we would encode $R \cap W$ explicitly (as addresses & values)
we have a $|R \cap W| \cdot \log n$ encoding that would allow us to jump from state at t_0 to the state at t_1 using a simulator (we know all we need)
But as this encodes $T \cdot \log n$ bits of entropy, we get: $T \log n \leq |R \cap W| \log n$ □

Lower Bound for Dynamic Connectivity

$\Omega(\log n)$ per operation (update or query) Patrascu & Demaine 2004
(even if all the components are paths)

<u>Proof:</u> $\sqrt{n} \times \sqrt{n}$ grid with perfect matchings between columns



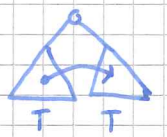
composition of permutations:
- $\pi_i \in S_{\sqrt{n}}$ (each needs $\sqrt{n} \log n$ encoding)
- path $v_{1,i}$ to $v_{k,j}$ iff $\underbrace{\pi_k(\pi_{k-1}(\dots \pi_2(\pi_1(i))\dots))}_{\pi_1 \circ \pi_2 \circ \dots \circ \pi_{k-1} \circ \pi_k} = j$

↳ just applying the previous result is not possible, as extracting $\pi_1 \circ \dots \circ \pi_k$ would take $\Theta(n)$ graph queries not $\Theta(\sqrt{n})$.

Instead: just support a verification query:

- maintain $\pi_1, \dots, \pi_{\sqrt{n}}$
- query(i, x): verify $\pi_1 \circ \dots \circ \pi_i = x$
- update(i, x): $\pi_i = x$

We will always query with $x = \pi_1 \circ \dots \circ \pi_i$ which we can check with $\sqrt{n}$ graph queries



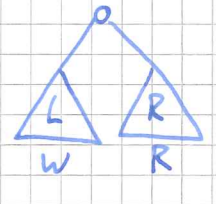
Apply same 'tree of tries' technique:

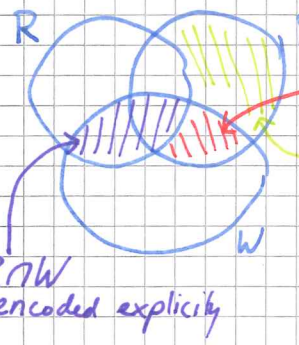
$\Omega(T\sqrt{n}\log n)$ bits of entropy } per node
↳ $\Omega(T \cdot \sqrt{n})$ cells

$\Omega(\sqrt{n} \cdot \sqrt{n} \log n)$ cells overall → $\Omega(\log n)$ per graph operation

Why does the simulation argument work?

for query(i, x) without knowing the x that will give answer 'yes'
we just try out all $x \in S_{\sqrt{n}}$ to see which one returns 'yes'
↳ But do we know all memory contents for this simulation?





problematic part: we don't know its content
old value from t_0 ok

either way: we want to abort the simulation as soon as it accesses $R' \setminus R$ → but how to detect?

idea: use a separator S between $R \setminus W$ & $W \setminus R$
→ such a separator exists with a small encoding (more math)