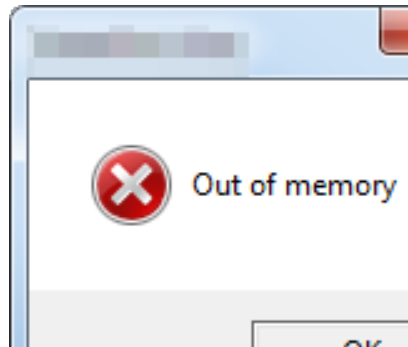


- Motivation
- Fibonacci revisited
- Matrix Powering revisited
- Graph Reachability revisited
- Methods/Cost of Saving Memory
- Memory  $\approx$  Parallel Time
- Streaming Algorithms
- (I/O-efficient/*external* memory algorithms)

# Motivation

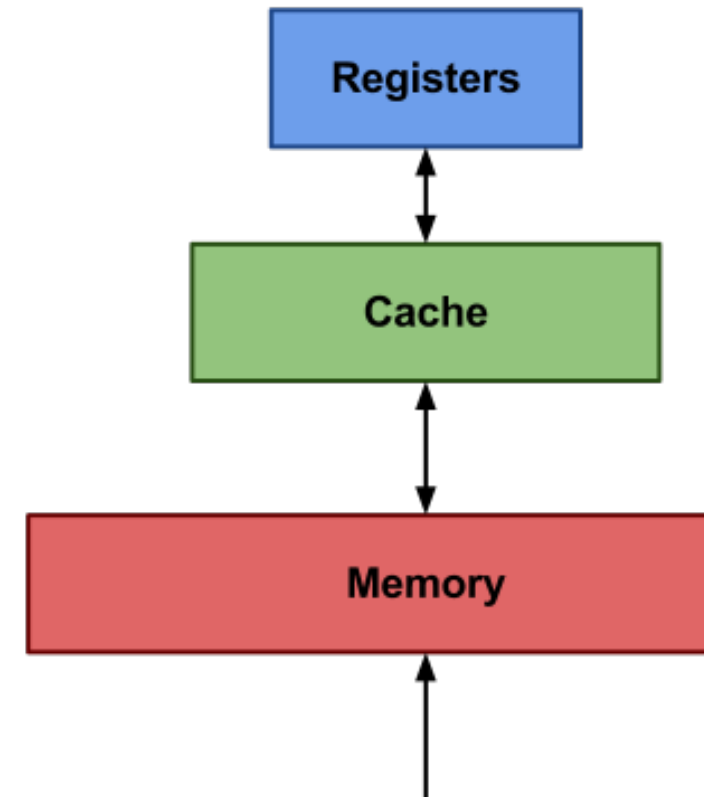
***Memory is limited, Time is unlimited.***



CPU Registers  
100 bytes  
< 1ns

Static RAM (SRAM)  
megabytes  
0.5-2.5ns

Dynamic RAM (DRAM)  
gigabytes  
50-70ns



- (I/O-efficient/*external* memory algorithms)

# Fibonacci, *Revisited*

- Algorithm **FibRek**( $n$ ):

if  $n=0$  return 0;  
 $n=1$  return 1;  
return FibRek( $n-1$ )  
+ FibRek( $n-2$ )

$$T(n) = T(n-1) + T(n-2) + O(n) = 2^{O(n)} \text{ bit ops.}$$

$$S(n) = \max\{S(n-1), S(n-2)\} + O(n) = O(n^2) \text{ bits}$$
- Algorithm **FibIter**( $n$ ):

$fib := 1; fibL := 0; \text{ while } n > 1$   
 $fibL := fib;$   
 $fib := fib + fibL; n := n - 1$

$$T(n) = O(n^2) \text{ bit ops.}$$

$$S(n) = O(n) \text{ bits}$$
- via  $2 \times 2$  integer matrix power:

$$\begin{vmatrix} F_n \\ F_{n-1} \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix} \cdot \begin{vmatrix} F_{n-1} \\ F_{n-2} \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix}^K \cdot \begin{vmatrix} F_{n-K} \\ F_{n-K-1} \end{vmatrix} \quad K := n-1$$

**Reminder:**  $F_n \approx \varphi^n$  of bit-length  $O(n)$ ,  $\varphi \approx 1.6$

# Integer Matrix Powering

- Algorithm **FibRek(n)**:

$$T(n) = T(n-1) + T(n-2) + O(n) = 2^{O(n)} \text{ bit op.s}$$

$$S(n) = \max \{ S(n-1), S(n-2) \} + O(n) = O(n^2) \text{ bits}$$

*Fast(est)  
integer  
multipl.*

[Harvey &  
van der  
Hoeven'19]

- Algorithm **FibIter(n)**:

$$T(n) = O(n^2) \text{ bit op.s}$$

$$S(n) = O(n) \text{ bits}$$

- via **2×2 integer matrix power**  $A \rightarrow A^K$ ,  $K := n-1$

- $A \rightarrow A^K$  **iteratively**:  $T = O(K^2 \log K)$   $S = O(K)$  **bits**

- $A \rightarrow A^2, A^4, A^8, \dots$  **repeated squaring**:

$$d=2 \quad T = O(K \cdot \log^2 K) \text{ bit op.s, } S = O(K) \text{ bits}$$

$$\log \max |A^K| \leq O(K \cdot \log d + K \cdot \log \max |A|) \leq O(K)$$

# Boolean Matrix Power, entrywise

**Given:** Boolean  $N \times N$  matrix  $A$

**Input:** integers  $K, I, J < N$

**Output:**  $(A^K)_{I,J}$

does *not* count as  
working memory use

only  $O(\log N)$   
bits of mem.

**Given:** Boolean  $N \times N$  matrices  $X, Y$

**Input:** integers  $I, J$

**Output:**  $(X \cdot Y)_{I,J} = \bigvee_L X_{I,L} \wedge Y_{L,J}$

**Repeated Squaring, recursively:**

$$A^K = (A^{K/2})^2 \quad \text{if } K \text{ even}$$

$$A^K = (A^{(K-1)/2})^2 \cdot A \quad \text{if } K \text{ odd}$$

$$S(K) = S(\lfloor K/2 \rfloor) + O(\log N) \text{ bits}$$

Use only  $O(\log K \cdot \log N)$  **bits** of working memory!

# Graph Reachability Revisited

**Given** directed graph  $G=(V,E)$  with  $N=|V|$  vertices.

**Input** vertices  $s,t \in V$  and distance  $K \in \mathbb{N}$ :

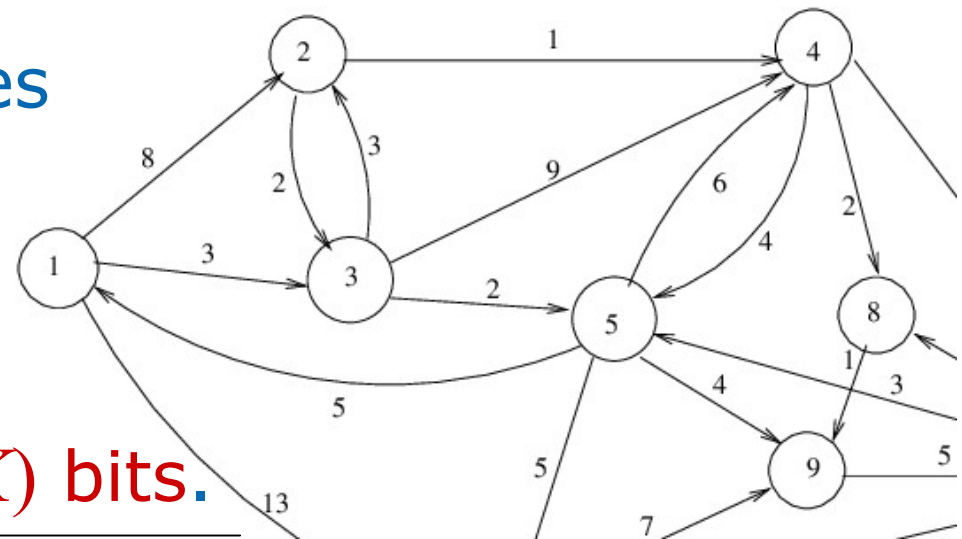
Is  $t$  reachable from  $s$  in  $G$  within at most  $K$  steps?

Given graph  $G=(V,E)$  not stored/input,

but accessed via queries " $(u,w) \in E$ ?"

**Dijkstra** maintains/updates  
"wavefront" of (tentative)  
distances  $d_s(u) \in \{1 \dots K\}$  to  $s$   
for each  $u \in V$ :

Stores  $N$  integers à  $O(\log K)$  bits.



**Entrywise  $N \times N$  Boolean Matrix Powering**  $A \rightarrow A^K$

Use only  $O(\log K \cdot \log N)$  **bits** of working memory!

# Graph Reachability Summary

---

**Given** directed graph  $G=(V,E)$  with  $N=|V|$  vertices.

**Input** vertices  $s,t \in V$  and distance  $K \in \mathbb{N}$ :

Is  $t$  reachable from  $s$  in  $G$  within at most  $K$  steps?

Graph  $G=(V,E)$  *not* stored/input,

but accessed via queries " $(u,w) \in E$  ?"

**Reingold'08:** *UN*directed Reachability in  $O(\log N)$  bits!

**Directed Graph Reachability** in  $O(\log N \cdot \log K)$  bits!

**Dijkstra** stores  $N$  integers à  $O(\log K)$  bits.

---

**Entrywise  $N \times N$  Boolean Matrix Powering**  $A \rightarrow A^K$

Use only  $O(\log K \cdot \log N)$  **bits** of working memory!

# Means / Cost of Memory Saving

*"Forget and recalculate intermediate results."*

**Memory-Saving Approach**  $A^K = (A^{K/2})^2$ ,  $K$  even

Matrix squaring  $(X \cdot X)_{I,J} = \sum_L X_{I,L} \wedge X_{L,J}$

now time  $T(K) = O(N) \cdot (T(K/2) + T(K/2))$

**time  $O(N^{\log K} \cdot K)$**

**Directed Graph Reachability** in  $O(\log N \cdot \log K)$  bits!

**Dijkstra** stores  $N$  integers à  $O(\log K)$  bits



# Memory $\approx$ Parallel Time

**Theorem (Borodin'77):**

*decision*

Any algorithm with seq. time  $T(n)$  using  $S(n)$  mem bits can be parallelized to a circuit of depth  $O(S(n) \cdot \log T(n))$



**Graph Reachability, distance  $K$  among  $N$  vertices:**

**Recall §9 : parallel Depth =  $O(\log N \cdot \log K)$**

# Reachability Method

**Theorem (Borodin'77):**

*decision*

Any algorithm with *seq.* time  $T(n)$  using  $S(n)$  mem bits can be parallelized to a circuit of *depth*  $O(S(n) \cdot \log T(n))$ .

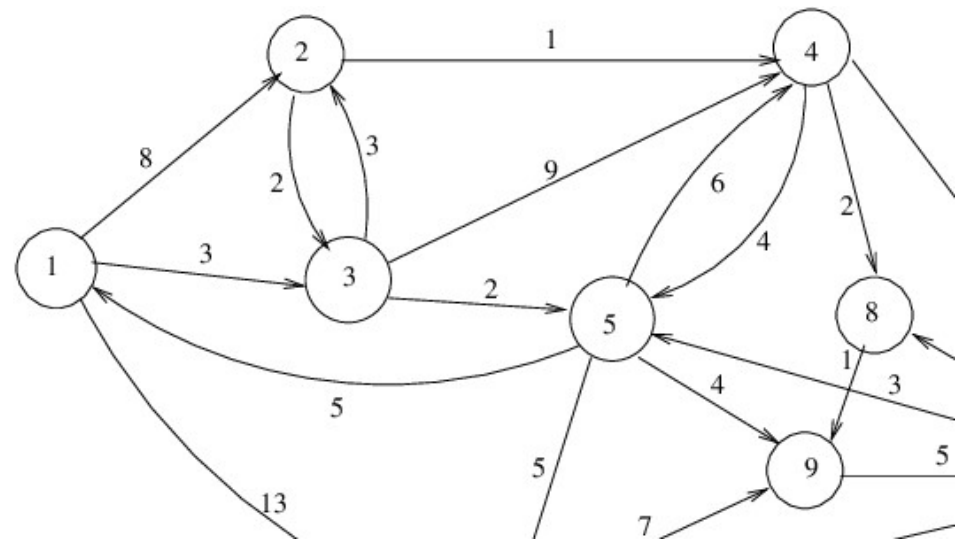
## Proof Sketch:

Configuration graph

has  $N := 2^{O(S(n))}$  vertices.

Unique (sic!) accepting configuration reached

within  $K := T(n)$  steps. ■

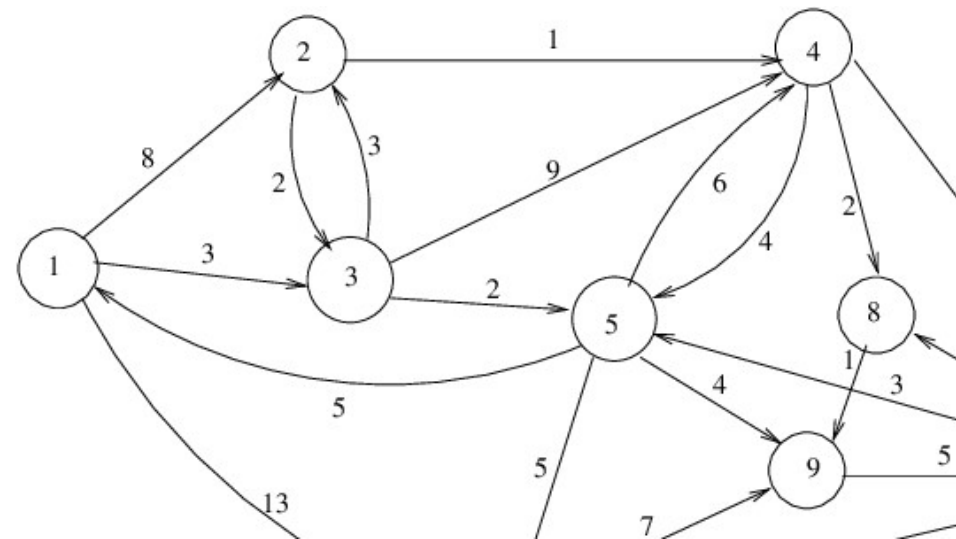


**Recall §9 : parallel Depth =  $O(\log N \cdot \log K)$**

# Configuration Graph

- *Configuration* of a program/algorithm:  
"snapshot" of its memory/variables + prog.counter
- *Start configuration* contains stored input,
- *End configurations* w.l.o.g. after erasing memory:  
one accepting, one rejecting (decision problem)

Configuration graph  
has  $N := 2^{O(S(n))}$  vertices.  
Unique (sic!) accepting  
configuration reached  
within  $K := T(n)$  steps.

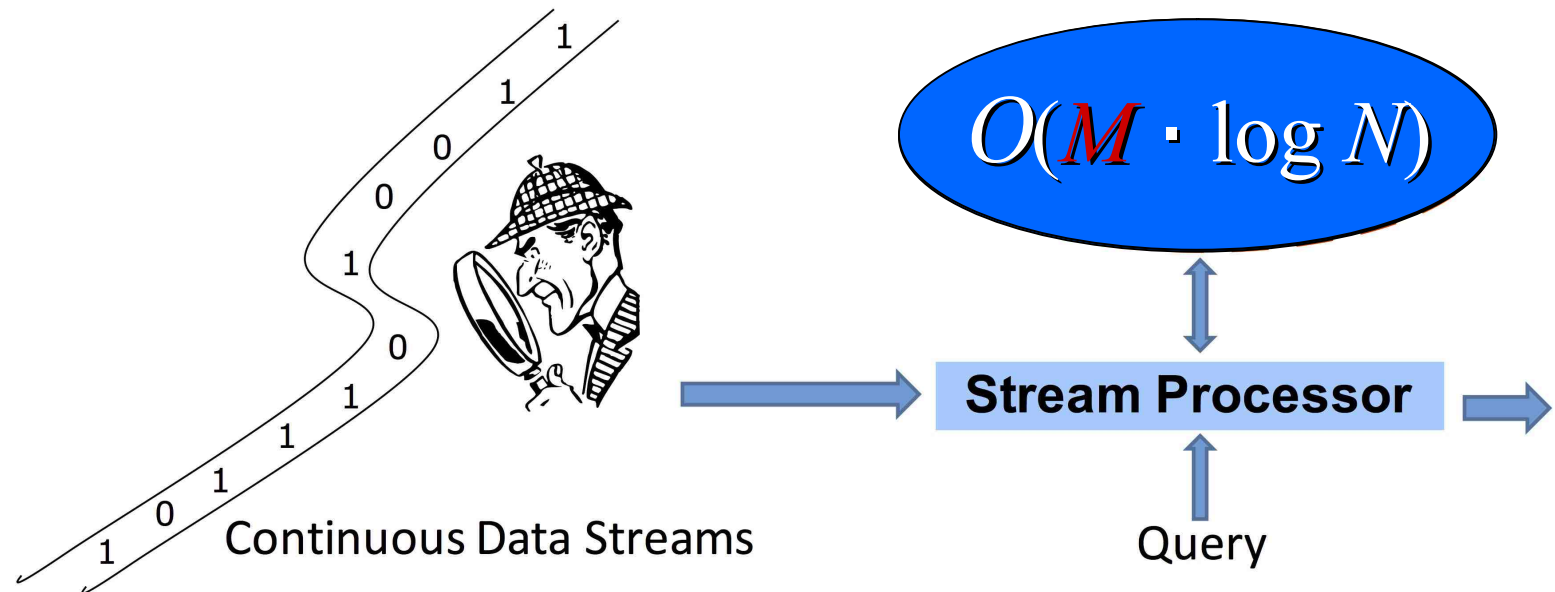


- *Step* of progr./algorithm = *directed* edge in graph

# Streaming Problems

**Example:** Input sequence  $x_n \in \{0, \dots, M-1\}$ ,  $n=0..N-1$

- a) **Count:** Given  $y \in \{0, \dots, M-1\}$ , output  $\#\{n: x_n=y\} =: C_y$
- b) **Sum:** output  $\sum_n x_n^k$
- c) **Counting *Distinct*:** output  $F_0 = \#\{y: C_y > 0\}$
- d) **Frequency *Moment*:** Given  $k$ , output  $F_k = \sum_y C_y^k = N$



# Random Approximate Counting

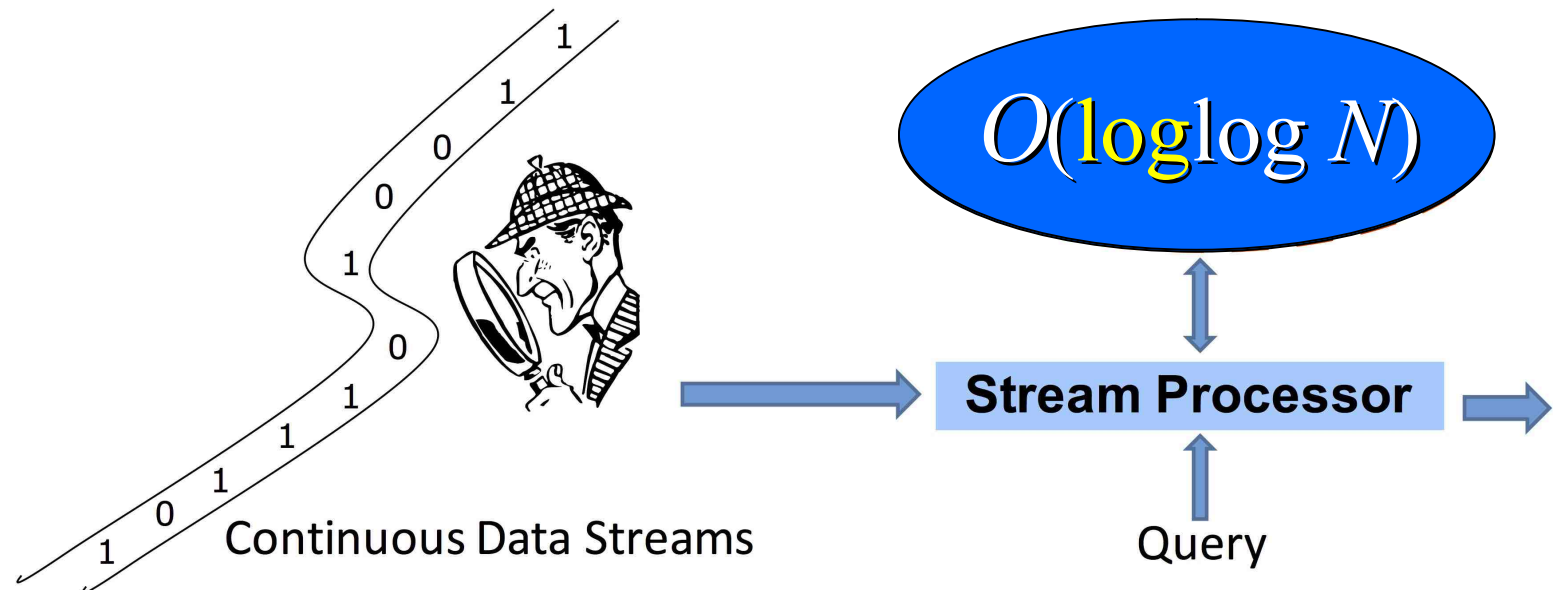
**Example:** Input sequence  $x_n \in \{0, \dots, M-1\}$ ,  $n=0..N-1$

**a) Count:** Given  $y \in \{0, \dots, M-1\}$ , output  $\#\{n: x_n=y\} =: C_y$

**Idea:** Store and maintain  $c_y := \log C_y$  instead of  $C_y$

Increment  $c_y$  with every  $2^{c_y} = C_y$ -th occurrence of  $y$ ,

on average: Flip  $c_y$  coins; if all up *Heads*, then  $++c_y$



# §10 Summary

- Motivation
- Fibonacci revisited
- Matrix Powering revisited
- Graph Reachability revisited
- Methods/Cost of Saving Memory
- Memory  $\approx$  Parallel Time
- Streaming Algorithms
- (I/O-efficient/*external* memory algorithms)

# QUIZ

Let  $A$  denote a  $d \times d$  matrix.

**Prove: a)**  $\max |A^2| \leq d \cdot (\max |A|)^2$

**Prove: b)**  $\max |A^K| \leq d^{K-1} \cdot (\max |A|)^K$

$\max |A^2| \leq d \cdot (\max |A|)^2, \quad \max |A^K| \leq d^{K-1} \cdot (\max |A|)^K$