Computational Complexity

计算复杂性

Lecture 10
Space Complexity 1

- How do we measure space complexity of a Turing machine?
- The largest number of tape cells a Turing machine visits on all inputs of a given length n.
- DSPACE (s) the class of decision problems that are solvable in space complexity s.
- NSPACE (s) the class of decision problems that are solvable by nondeterministic TMs in space complexity s.
- logarithmic space complexity-L

Chapter 5 Space Complexity

- **General preliminaries and issues**
- **Log space classes: L, NL**

- **Savitch's Theorem**
  NSPACE(s)⊆DSPACE(s^2)
- **Immerman's Theorem： NL=co-NL**

- **PSPACE and QBF**

General preliminaries and issues

Multi-tape Turing machine.

Space complexity is meant to measure the amount of temporary stor (i.e. computer's memory) used when performing a computational tas

The standard definition of **space complexity** refers to the number of the work-tape scanned by the machine on each input.

The binary space complexity of a computation refers to the number o that can be stored in these cells, thus multiplying the number of cells logarithm of the finite set of work symbols of the machine.

For any function $s : \mathbb{N} \to \mathbb{N}$ we denote by $D_{SPACE}(s)$ the class of deci problems that are solvable in space complexity s.

A function $s : \mathbb{N} \to \mathbb{N}$ **space constructible** if there exists an algorithr on input n outputs $s(n)$ using at most $s(n)$ cells of the work-tape.

Space Complexity Classes

For any function $s:N \rightarrow N$, we define:

$DSPACE(s)=\{ S : S$ is decidable by a deterministic $s(n)$ space TM$\}$

$NSPACE(s)=\{ S : S$ is decidable by a non-deterministic $s(n)$ space TM$\}$

- Space complexity discards the use of the input and output devices.
- The input device is read-only and the output device is write-only.

  We assume that the input is read only, and that the output is write only, and we don't count either as part of the memory (space) usage.

- We will usually refer to the binary space complexity of algorithms, where the binary space complexity of a machine M that uses the alphabet Σ, finite state set Q, and has standard space complexity S_M is defined as $(\log_2 |Q|) + (\log -2|\Sigma|) \cdot S_M$ (Recall that S_M measures the number of cells of the temporary storage device that are used by M during the computation).
- We will assume that the machine does not scan the input device beyond the boundaries of the input.
- We will assume that the machine does not rewrite to locations of its output device (i. e., it write to each cell of the output device at most once).

The lowest space complexity class that we shall study in depth is logarithmic space

sublogarithmic space classes;

Logarithmic space is a critical bound in space complexity.

It is tempting to say that sub-logarithmic space machines are not more useful than constant space machines, because it seems impossible to allocate a sub-logarithmic amount of space(eg., the TMs working within sublogarithmic space cannot even record the the length of the input, and thus can be ‘fooled’ easily.)

In spite of the fact that some non trivial things can be done in sub-logarithmic space complexity, the lowest space complexity class that we shall study in depth is logarithmic space.

Time Versus Space

Unlike time, space can be reused, but, on the other hand, intermediate results of a computation cannot be recorded for free. These two conflicting aspects are captured in the following composition lemma:

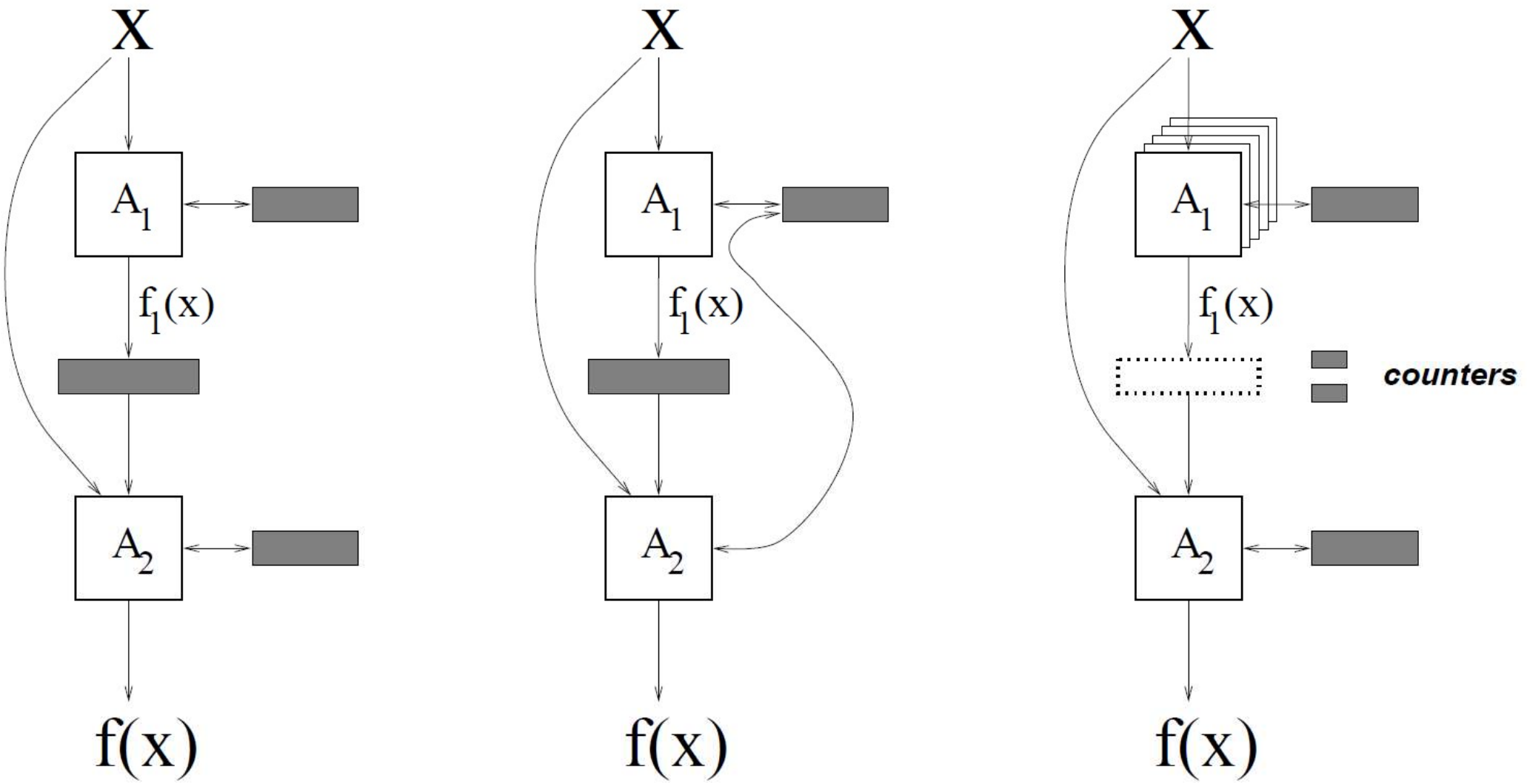
Naive composition:

Lemma 5.1 (naive composition): *Let* $f_1 : \{0,1\}^* \to \{0,1\}^*$ *and* $f_2 : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *be computable in space* s_1 *and* s_2*, respectively.*[4] *Then* f *defined by* $f(x) \stackrel{\text{def}}{=} f_2(x, f_1(x))$ *is computable in space* s *such that*

$$s(n) = max(s_1(n), s_2(n + l(n))) + l(n) + \delta(n)$$

where $l(n) = max_{x \in \{0,1\}^n}\{|f_1(x)|\}$ and
$\delta(n) = O(\log(l(n) + s_2(n + l(n)))) = o(s(n))$.

f(x)=f2(x, f1(x))



Emulative composition:

Lemma 5.2 (emulative composition): *Let f_1, f_2, s_1, s_2, ℓ and f be as in Lemma 5.1. Then f is computable in space s such that*

$$s(n) = s_1(n) + s_2(n + \ell(n)) + O(\log(n + \ell(n))) + \delta(n),$$

where $\delta(n) = O(\log(s_1(n) + s_2(n + \ell(n)))) = o(s(n))$.

Theorem 1. *If f and g are functions computable in DSPACE(S(n)), then $g \circ f = g(f(x))$ is in DSPACE(S(exp(S(n)))).*

$$DTIME(t(n)) \subseteq SPACE(t(n))$$

Trivial: A TM running in t steps can only touch t cells. So the TM deciding L in time t can also only touch t cells.

反之不成立。事实上:

Theorem 2. *If an algorithm A has binary space complexity s and halts on every input then it has time complexity t such that* $t(n) \leq n \cdot 2^{s(n)+\log_2 s(n)}$.

也就是:

$$SPACE(s(n)) \subseteq DTIME(2^{s(n)})$$

Basic idea: Assume TM M can decide L in space SPACE (s(n)), i.e., Given x, it will always halt with an accept or reject. Two positions on the tape cannot occur twice(Otherwise M would not halt)!

Proof: The proof refers to the notion of an *instantaneous configuration* (in a computation). Before starting, we warn the reader that this notion may be given different definitions, each tailored to the application at hand. All these definitions share the desire to specify *variable information* that together with some *fixed information* determines the next step of the computation being analyzed. In the current proof, we fix an algorithm A and an input x, and consider as variable the contents of the storage device (e.g., work-tape of a Turing machine as well as its finite state) and the machine's location on the input device and on the storage device. Thus, an instantaneous configuration of $A(x)$ consists of the latter three objects (i.e., the contents of the storage device and a pair of locations), and can be encoded by a binary string of length $\ell(|x|) = s(|x|) + \log_2 |x| + \log_2 s(|x|)$.[7]

The key observation is that the computation $A(x)$ cannot pass through the same computation twice, because otherwise the computation $A(x)$ passes through this configuration infinitely many times, which means that it does not halt. Intuitively, the point is that the fixed information (i.e., A and x) together with the configuration, determines the next step of the computation. Thus, whatever happens (i steps) after the first time that the computation $A(x)$ passes through configuration γ, will also happen (i steps) after the second time that the computation $A(x)$ passes through γ.

By the forgoing observation, we infer that the number of steps taken by A on input x is at most $2^{\ell(|x|)}$, because otherwise the same configuration will appear twice in the computation (which contradicts the halting hypothesis). The theorem follows. ■

SAT can be solved in linear(O(n)) space

SAT is NP-complete.

M_1="On input <ϕ>, where ϕ is a Boolean formula:

1. For each truth assignment to the variables $x_1,\ldots x_n$ of ϕ.
2. Evaluate ϕ on that truth assignment.
3. If ϕ ever evaluate to 1, ACCEPT; if not, REJECT."

x_1	x_2	x_3	…	x_n	
0	0	0	…	0	

Two models for Non-Deterministic

- **Non-deterministic time-bounded computations were dened via two equivalent models:**
  Off-line model (NP as a proof system)
  On-line model (traditional denition of NP)

In the standard model, called the on-line model, the machine makes non-deterministic choices "on the fly" (or, alternatively, reads a non-deterministic input from a special read-only tape *that can be read only in a uni-directional way*). Thus, if the machine needs to refer to such a non-deterministic choice at a latter stage in its computation, then it must store the choice on its storage device (and be charged for it). In contrast, in the so-called off-line model the non-deterministic choices (or the bits of the non-deterministic input) are read from a read-only device (or tape) *that can be scanned in both directions like the main input.*

$\mathrm{NSPACE}_{\text{on-line}}(s)$ is in $\mathrm{NSPACE}_{\text{off-line}}(s)$

We denote by $\mathrm{NSPACE}_{\text{on-line}}(s)$ (resp., $\mathrm{NSPACE}_{\text{off-line}}(s)$) the class of sets that are acceptable by an on-line (resp., off-line) non-deterministic machine having space complexity s. We stress that, as in Definition 2.7, the set accepted by a non-deterministic machine M is the set of strings x such that there exists a computation of $M(x)$ that is accepting. Clearly, $\mathrm{NSPACE}_{\text{on-line}}(s) \subseteq \mathrm{NSPACE}_{\text{off-line}}(s)$. On the other hand, not only that $\mathrm{NSPACE}_{\text{on-line}}(s) \neq \mathrm{NSPACE}_{\text{off-line}}(s)$ but rather $\mathrm{NSPACE}_{\text{on-line}}(s) = \mathrm{NSPACE}_{\text{off-line}}(\Theta(\log s))$, provided that s is at least linear. For

- let us justify the focus on the online model

Low Space Classes

Definitions (logarithmic space classes):

- L = DSPACE(logn)
- NL = NSPACE(logn)

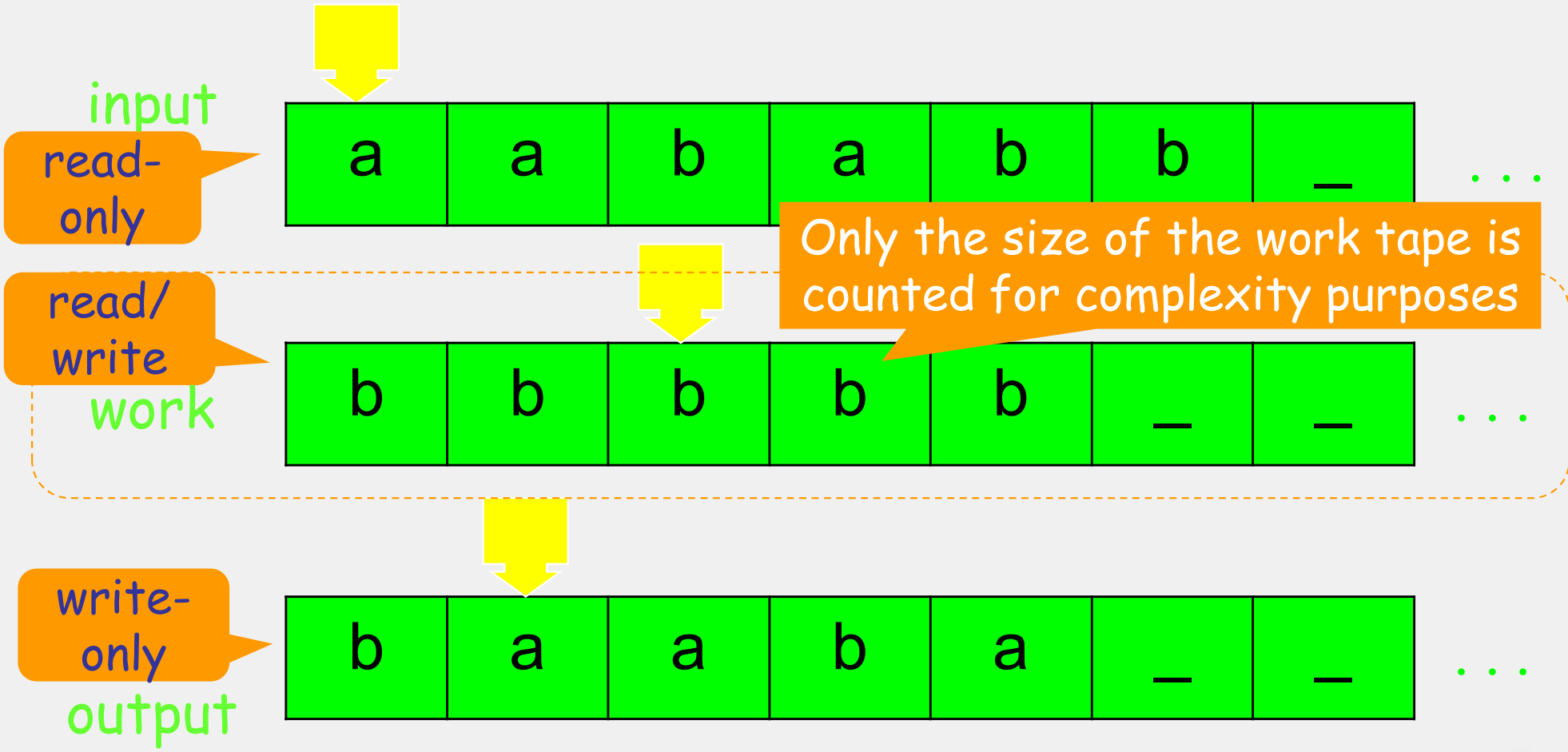
Specifically, $\mathcal{NL} = \cup_c \text{NSPACE}(\ell_c)$, where $\ell_c(n) = c \log_2 n$.

We first note that the proof of Theorem 5.3 can be easily extended to the (on-line) non-deterministic context. The reason being that moving from the deterministic model to the current model does not affect the number of instantaneous configurations (as defined in the proof of Theorem 5.3), whereas this number bounds the time complexity. Thus, $\mathcal{NL} \subseteq \mathcal{P}$.

Problem!

How can a TM use only logn space if the input itself takes n cells?!

3Tape Machines

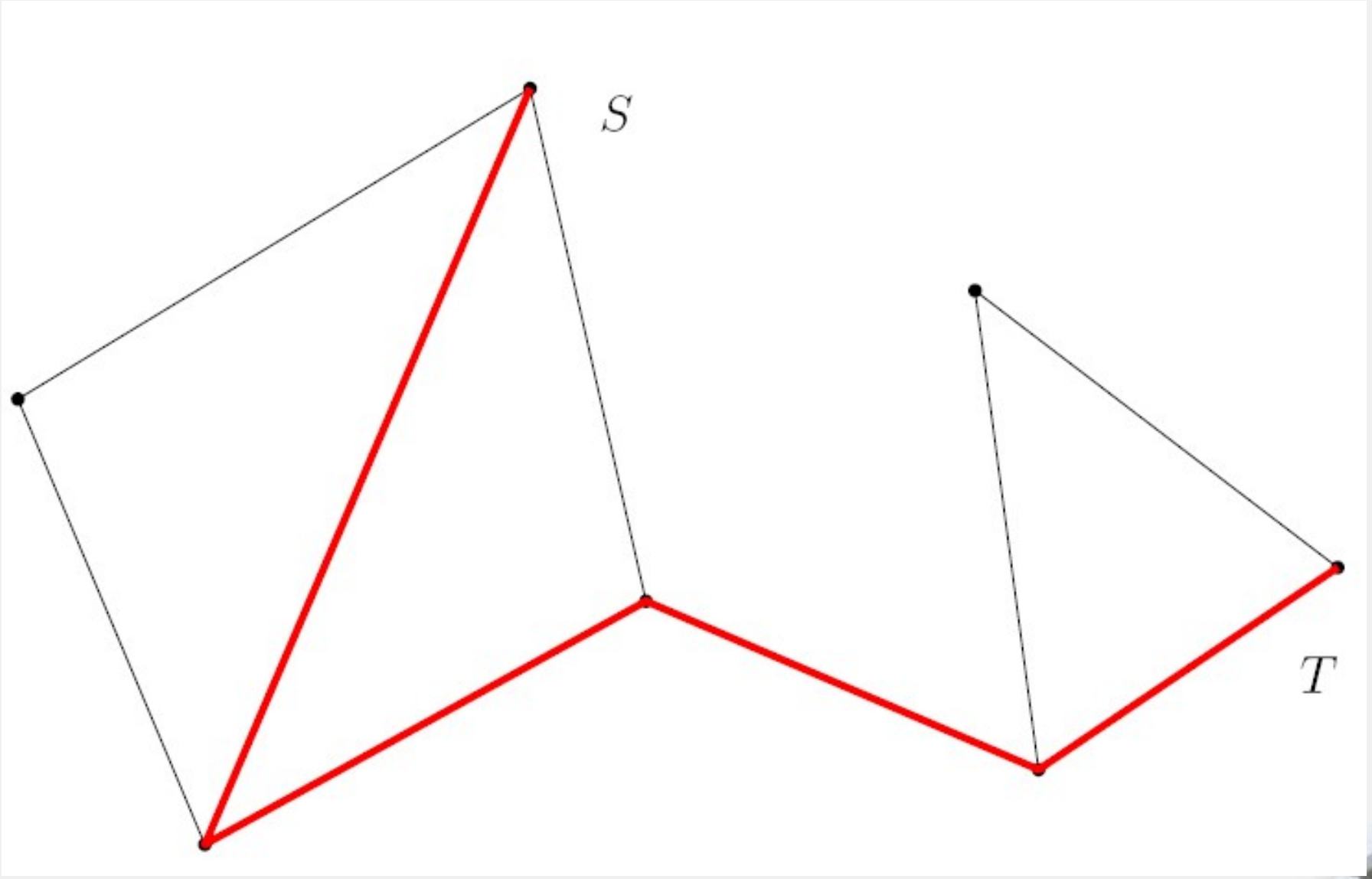


Graph Connectivity(连通性)

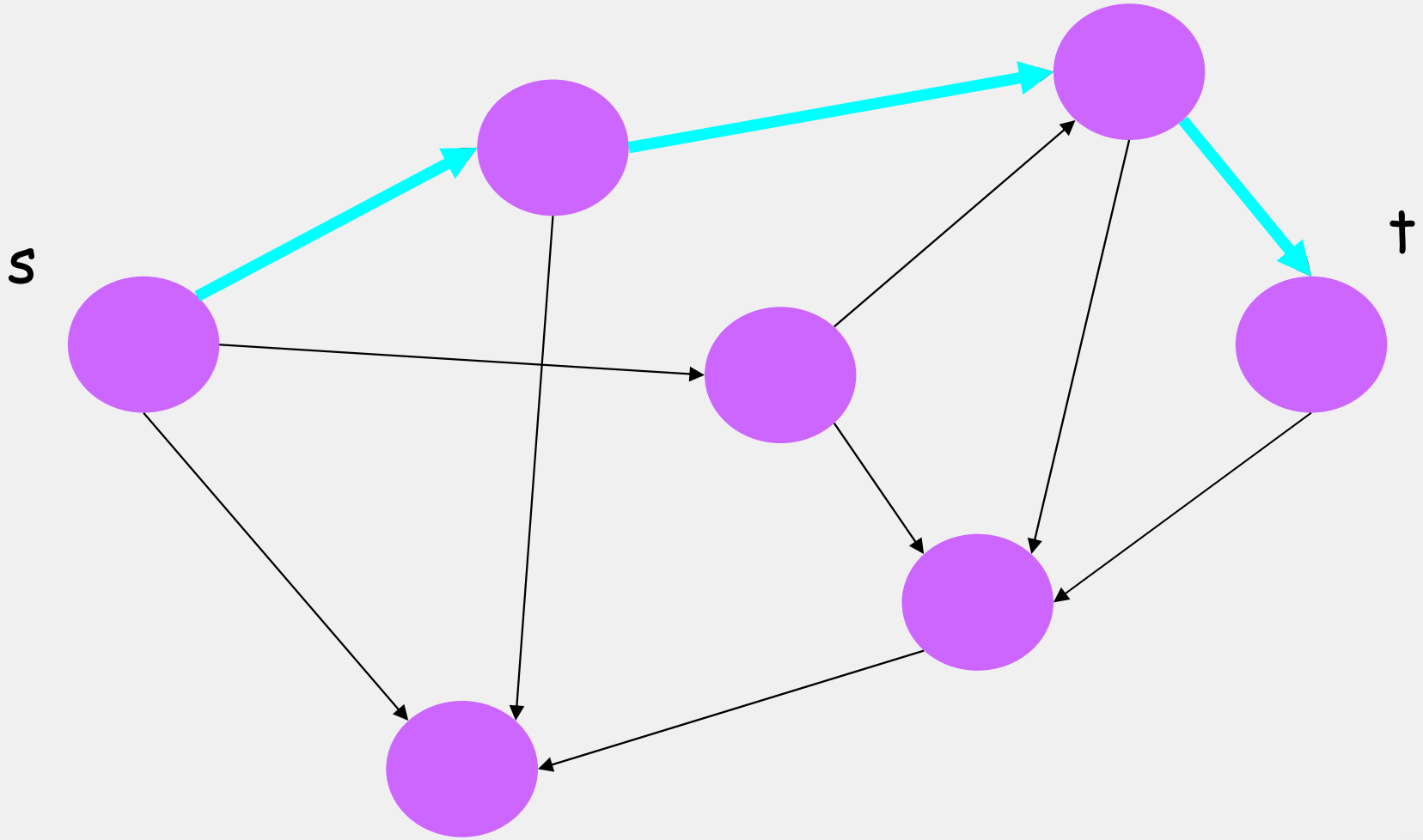
CONN

- Instance: a directed(or undrected) graph G=(V,E) and two vertices $s,t \in V$
- Problem: To decide if there is a path from s to t in G?

Undirected connectivity(UCONN)



st-CONN



CONN is in NL

- Start at s
- For i = 1, .., |V| {
  - Non-deterministically choose a neighbor and jump to it
  - Accept if you get to t

  }
- If you got here – reject!

- Counting up to |V| requires log|V| space
- Storing the current position requires log|V| space

Log-Space Reductions

Definition:

A is log-space reducible to B, written $A \leq_L B$,

if there exists a log space TM M that, given input w, outputs f(w) s.t.

$$w \in A \text{ iff } f(w) \in B$$

NL Completeness

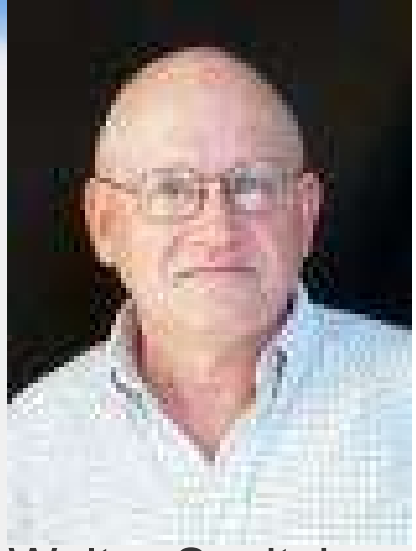
Definition:

A language B is NL-Complete if

1. $B \in NL$
2. For every $A \in NL$, $A \leq_L B$.

If (2) holds, B is NL-hard

Savitch's Theorem



Walter Savitch

Theorem: (5.12)

$\forall S(n) \geq \log(n)$

$$NSPACE(S(n)) \subseteq DSPACE(S(n)^2)$$

Proof:

First we'll prove $NL \subseteq DSPACE(\log^2 n)$

then, show this implies the general case

Savitch's Theorem

Theorem:

$$NSPACE(\log n) \subseteq DSPACE(\log^2 n)$$

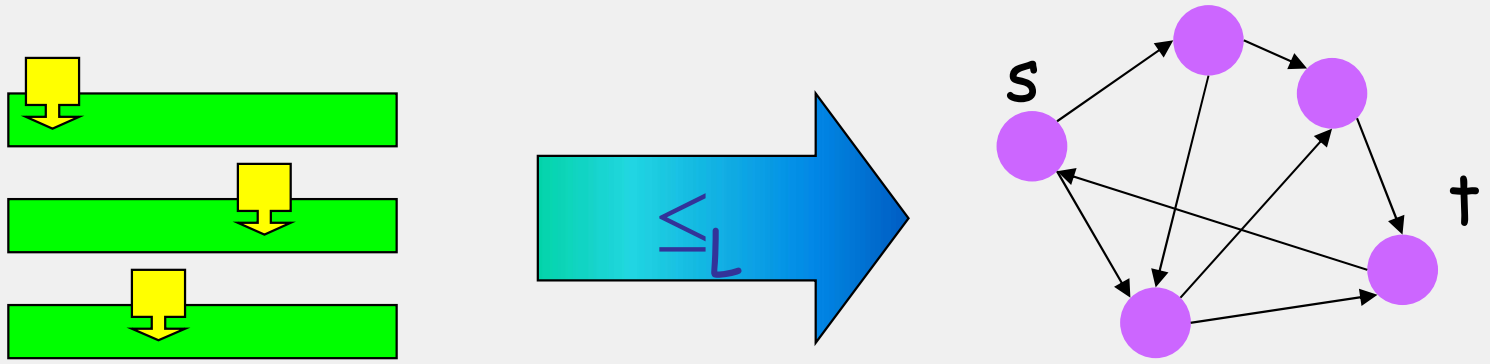
Proof:

1. First prove CONN is NL-complete (under log-space reductions)
2. Then show an algorithm for CONN that uses $\log^2 n$ space

CONN is NL-Complete

Theorem: CONN is NL-Complete

Proof: by the following reduction:



"Does M accept x?"

"Is there a path from s to t?"

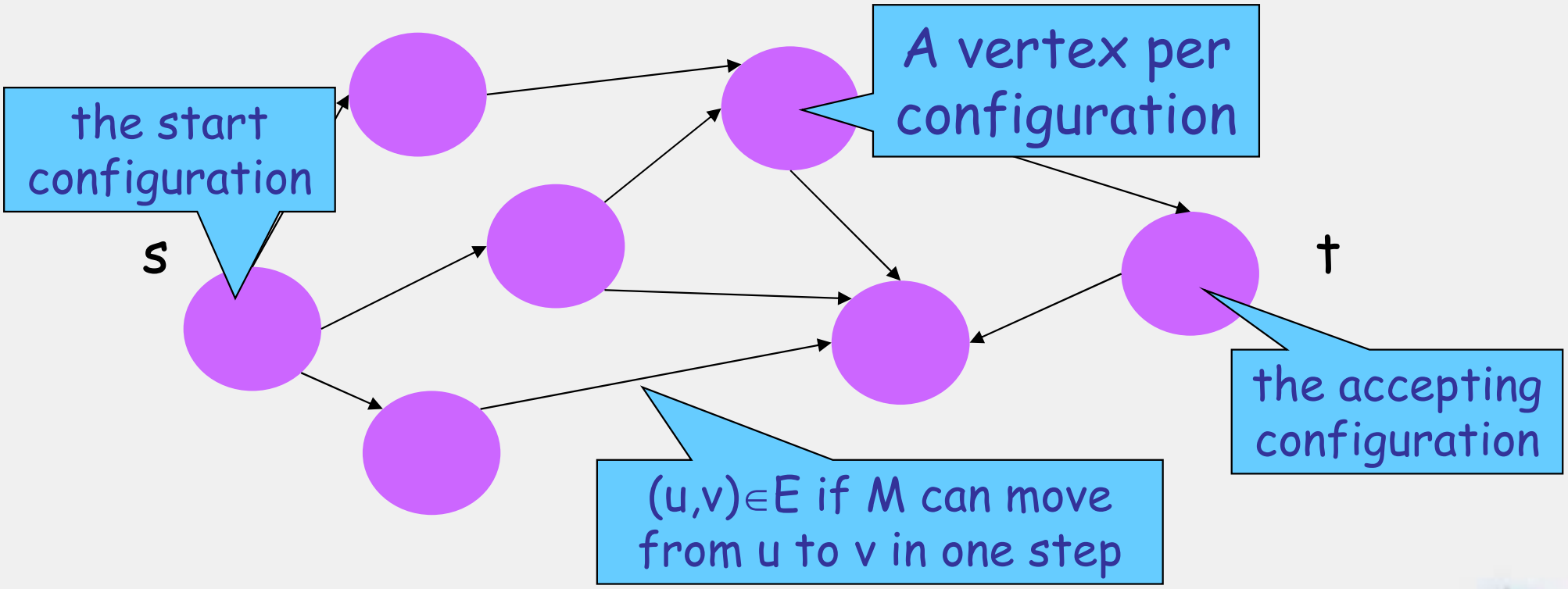
Technicality

Observation:

Without loss of generality, we can assume all NTM's have exactly one accepting configuration.

Configurations Graph

A Computation of a NTM M on an input x can be described by a graph $G_{M,x}$:



Correctness

Claim: For every non-deterministic log-space Turing machine M and every input x,

M accepts x iff there is a path from s to t in $G_{M,x}$

CONN is NL-Complete

Corollary: CONN is NL-Complete

Proof: We've shown CONN is in NL. We've also presented a reduction from any NL to CONN which is computable in log space (Why?) ■

A Byproduct

Claim: NL⊆P

Proof:

- Any NL is log-space reducible to CONN
- Thus, any NL is poly-time reducible to CONN
- CONN is in P
- Thus any NL is in P. ■

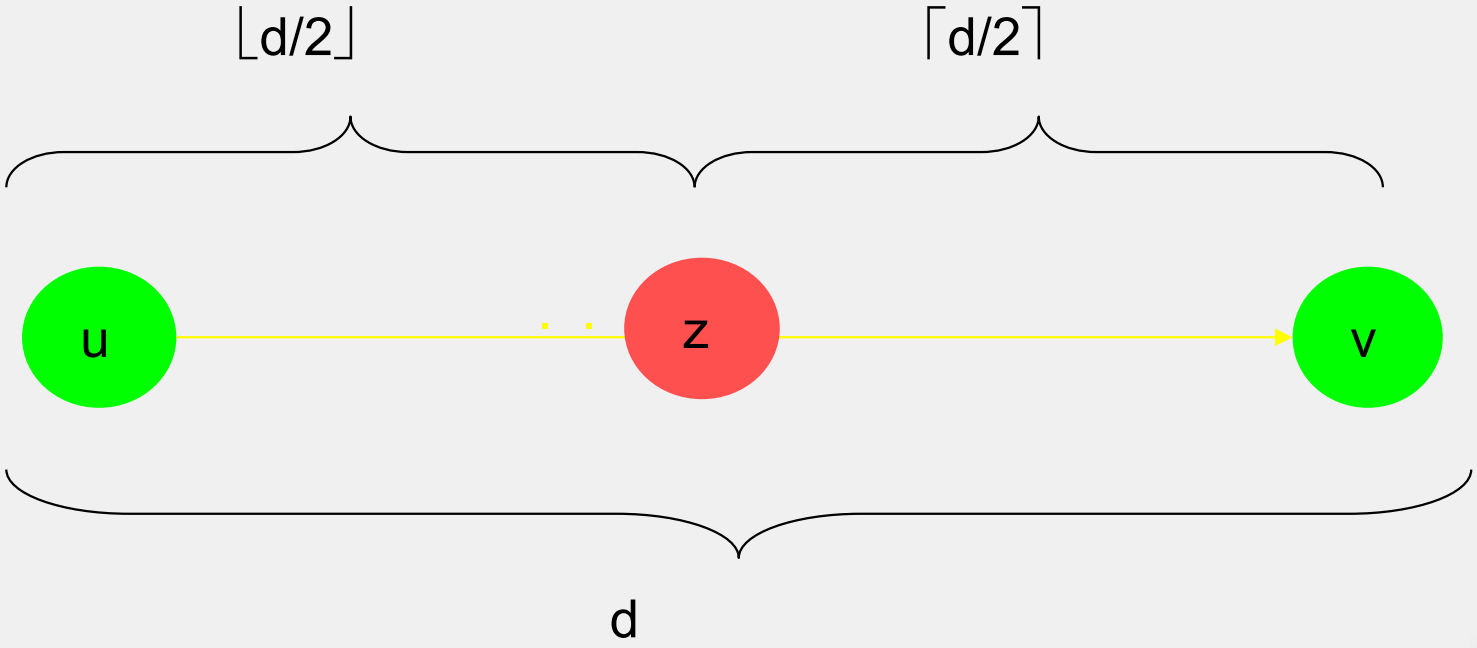
What Next?

We need to show CONN can be decided by a deterministic TM in $O(\log^2 n)$ space.

This problem has received a lot of attention in the past few decades and the complexity of USTCON has been well studied. The first randomized log-space algorithm for USTCON was shown in 1979 by Aleliunas, Karp, Lipton, Lovász and Rackoff. In 1970, Savitch demonstrated a simulation of a non-deterministic space S machine by a deterministic space S^2 machine. Thus USTCON $\in$ SPACE $(\log^2 n)$. Nisan, Szemerdi and Wigderson in 1989 showed that USTCON $\in$ SPACE$(\log^{3/2} n)$. Armoni, Ta-Shma, Wigderson and Zhou in 2000 proved that USTCON $\in$ SPACE$(\log^{4/3} n)$. In 2005, Reingold used expander graphs to show that USTCON is in **L** and **SL** collapses to **L**.

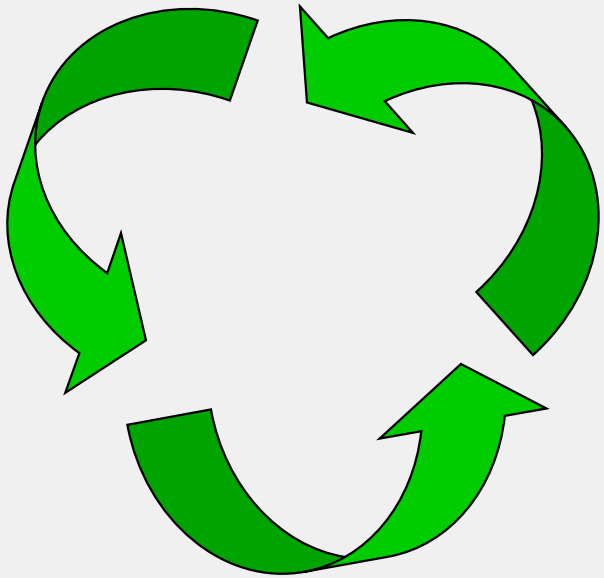
The Trick

"Is there a vertex z, so there is a path from u to z of size ⌊d/2⌋ and one from z to v of size ⌈d/2⌉?"



Recycling Space

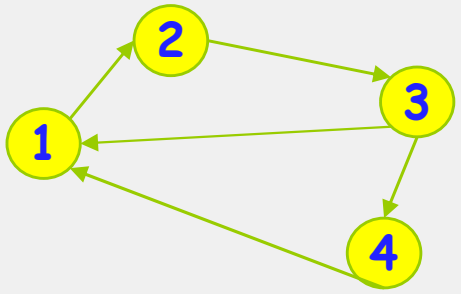
The two recursive invocations can use the same space



The Algorithm

```
Boolean PATH(a,b,d) {
    if there is an edge from a to b then
        return TRUE
    else {
        if d=1 return FALSE
        for every vertex v {
            if PATH(a,v,⌈d/2⌉) and PATH(v,b,⌊d/2⌋)
            then return TRUE
        }
        return FALSE
    }
}
```

Example of Savitch's algorithm




```
boolean PATH(a,b,d) {
    if there is an edge from a to b then
        return TRUE
    else {
        if (d=1) return FALSE
        for every vertex v (not a,b)  {
            if PATH(a,v, ⌈d/2⌉) and
                PATH(v,b, ⌊d/2⌋) then
                return TRUE
        }
        return FALSE
    }
}
```

(a,b,c)=Is there a path from a to b, that takes no more than c steps.

(1,4,3) TRUE

$3\text{Log}_2(d)$

$O(\log^2 n)$ Space DTM

Claim: There is a deterministic TM which decides CONN in $O(\log^2 n)$ space.

Proof:

To solve CONN, we invoke PATH(s,t,|V|)

The space complexity:

$S(n)=S(n/2)+O(\log n)=O(\log^2 n)$ ■

Conclusion

Theorem:

$$NSPACE(\log n) \subseteq DSPACE(\log^2 n)$$

How about the general case
$NSPACE(S(n)) \subseteq SPACE(S^2(n))$?

Formally

$s_i(n)$ can be computed with space $s_i(n)$

Claim: For any two space constructible functions $s_1(n), s_2(n) \geq \log n$, $f(n) \geq n$:

simulation overhead

$$NSPACE(s_1(n)) \subseteq SPACE(s_2(n))$$

$$\Downarrow$$

$$NSPACE(s_1(f(n))) \subseteq SPACE(s_2(f(n)))$$

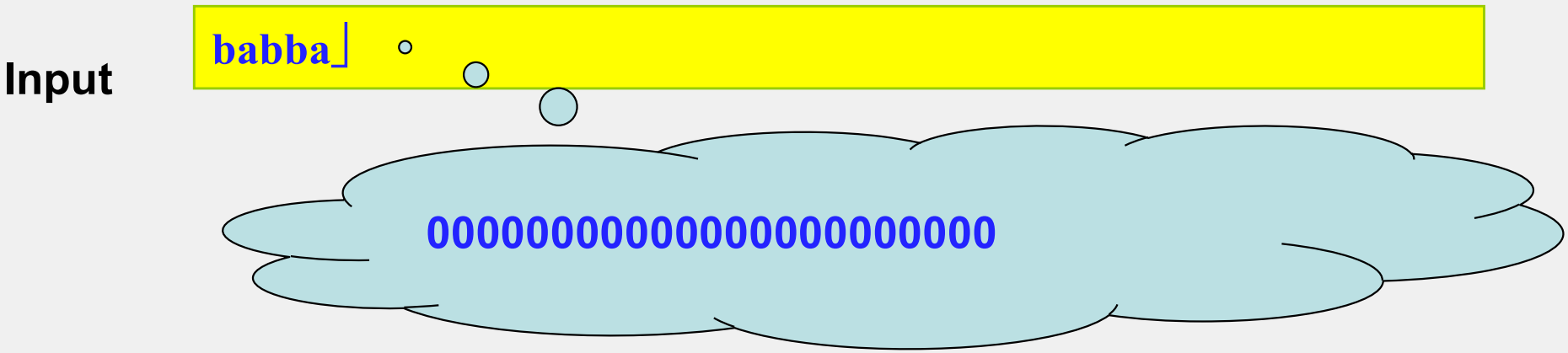
E.g $NSPACE(n) \subseteq SPACE(n^2) \Rightarrow NSPACE(n^2) \subseteq SPACE(n^4)$

Padding Argument

- We started with $L \in NSPACE(s_1(f(n)))$
- We showed: $L' \in NSPACE(s_1(n))$
- Thus, $L' \in SPACE(s_2(n))$
- Using the DTM for L' we'll construct a DTM for L, which will work in $O(s_2(f(n)))$ space.

Padding Argument

- The DTM for L will simulate the DTM for L' when working on its input concatenated with zeros



Padding Argument

- When the input head leaves the input part, just pretend it encounters 0s.
- maintaining the simulated position (on the imaginary part of the tape) takes $O(\log(f(|x|)))$ space.
- Thus our machine uses $O(s_2(f(|x|)))$ space.
- $\Rightarrow NSPACE(s_1(f(n))) \subseteq SPACE(s_2(f(n)))$

Savitch: Generalized Version

Theorem (Savitch):

$\forall S(n) \geq \log(n)$

$$NSPACE(S(n)) \subseteq SPACE(S(n)^2)$$

Proof: We proved $NL \subseteq SPACE(\log^2 n)$. The theorem follows from the padding argument. ■

Corollary

Corollary: PSPACE = NPSPACE

PSPACE is the class of problems that are decidable in polynomial space on a det. TM.

PSPACE= U_k SPACE(n^k).

Proof:

Clearly, PSPACE$\subseteq$NPSPACE.

By Savitch's theorem, NPSPACE$\subseteq$PSPACE. ■