Space complexity

IIC3242

Conventions for space complexity

When dealing with space complexity we use Turing machines with input and output

The input string is not accounted in the complexity cost:

Definition

If M is a k-tape Turing machine with input and output, then the space used by M on input w is the total number of cells accessed by the heads of M on its work tapes (tapes 2 through $k-1$).

For a decider the output tape is not important, so sometimes we just use the input tape and the work tapes

We can use as many tapes as we wish (recall the proof that k-tape TM $=$ 1 one tape TM)

Space complexity

No we can define

Definition

For a k-tape deterministic TM with input/output the space complexity function $s_M(n)$ is defined as the maximum number of cells on work tapes that M scans when processing any input of length n.

For a non-deterministic machine this is the maximum over all branches of computation and all inputs of length n.

In both cases we say that the machine M runs in space $s_M(n)$, or that $s_M(n)$ is the space complexity of M.

Space complexity classes

Just as before:

Definition

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define:

$$\text{DSPACE}(f(n)) = \{L \mid L \text{ is decided by an } O(f(n)) \text{ space deterministic Turing machine with input/output}\}.$$

$$\text{NSPACE}(f(n)) = \{L \mid L \text{ is decided by an } O(f(n)) \text{ space non-deterministic Turing machine with input/output}\}.$$

Space complexity: examples

It is easy to see that SAT is in DSPACE(n):

$M =$ On input $\langle\varphi\rangle$, for a boolean formula φ:

1. For each truth assignment to variables of φ
2. Evaluate φ on this assignment
3. If we ever get 1 accept, otherwise reject

We reuse space for assignments (we only need to count up to $2^{\text{numer of variables}}$)

Space complexity: examples

$$\overline{\mathsf{ALL}_{\mathsf{NFA}}} = \{\langle A\rangle \mid A \text{ is an NFA and } L(A) \neq \Sigma^*\}$$

Note: A is not universal iff $\overline{A} \neq \emptyset$

- So we can just search if the powerset automaton for the complement is not empty

$M =$ On input $\langle A\rangle$, for an NFA A:

1. Write start state of $\overline{A}$ on the work tape
2. Repeat 2^q times, for q the number of states of A:
3. Nondeterministically pick a symbol of Σ
4. Write the next state (remember current and next state)
5. If we ever accept accept, otherwise reject

Clearly in NSPACE(n)

Space complexity classes

As in the time complexity case the most important classes are:

Definition

$$\mathrm{PSPACE} = \bigcup_k \mathrm{DSPACE}(n^k)$$

$$\mathrm{NPSPACE} = \bigcup_k \mathrm{NSPACE}(n^k)$$

For these guys the input tape does not matter. Why?

Is PSPACE=NPSPACE worth a million?

No, not really (recall in our examples that space can be reused)

Theorem (Savitch)

When $f(n) \geq logn$, then

$$NSPACE(f(n)) = DSPACE(f^2(n)).$$

Proof: A neat idea: *computation* = *graph*.

Consider an $f(n)$-space nondeterministic machine M (with input)

A configuration is defined as before

Savitch's theorem: configuration graph

How many configurations are there on input w of length n?

- We have k tapes, but input/output don't count
- A configuration: $\#u_1qv_1\#u_2qv_2\#\cdots\#u_kqv_k\#$
- Representing configuration: $(q, i, w_2, p_2, w_3, p_3, \ldots, w_{k-1}, p_{k-1})$
- So $|Q| \cdot n \cdot k \cdot f(n) \cdot |\Gamma|^{(k-2)\cdot f(n)} = 2^{O(f(n))}$ (recall $u_1 v_1$ is input)
- Above we use $f(n) \geq logn$

Configuration graph of M on w (notation $G(M, w)$):

- Nodes: Configurations of M on w
- If C_1 yields C_2 there is an edge between them

Savitch's theorem: configuration graph

No need to represent the graph as adjacency list/matrix

We just store the machine and input

And look up if there is an edge

From input string we just look up the position

So we only need to count

Savitch's theorem: configuration graph

From the definition we have:

w is accepted by M iff there is a path in the configuration graph $G(M, w)$ from an initial to an accepting state

So we only need to solve the reachability problem in this graph efficiently to get the desired result

Savitch's theorem: the meat

Recall:

$\mathsf{PATH} = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a path from } s \text{ to } t\}$

In fact this is what we want:

Theorem

PATH is in $DSPACE(log^2 n)$.

Savitch's theorem: the meat

Recall:

$\text{PATH} = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a path from } s \text{ to } t\}$

In fact this is what we want:

Theorem

PATH is in DSPACE($\log^2 n$).

Savitch's theorem: fast reachability

Proof: Let G be a graph with n nodes and $x, y \in G$.

We define a predicate $REACH(x, y, i)$ which is true iff there is a path in G from x to y of length *at most* 2^i

If we solve $REACH(x, y, \lceil logn \rceil)$ we solve PATH

Savitch's theorem: fast reachability

To solve $REACH(x, y, i)$ we use a DTM with input and two work tapes:

- Input is the adjacency matrix
- The first work tape has triples of the form (x, y, i)
- Note that each triple is of the size $3logn$ roughly
- The second work tape is just for maintaining indices (counting up to n^2)

Key observation:

- A path from x to y of length $\leq 2^i$
- Has a midpoint z with:
- A path from x to z of length $\leq 2^{i-1}$
- And a path from z to y of length $\leq 2^{i-1}$

Savitch's theorem: fast reachability

The algorithm is recursive:

$REACH(x, y, i)$:

1. If $i = 0$ check if $x = y$, or there is an edge between them
2.   If yes return true, else false
3. If $i \geq 1$ then *for all* $z \in G$:
4.   Run recursively $REACH(x, z, i - 1)$ and $REACH(z, y, i - 1)$
5.   If both return true return true

Savitch's theorem: fast reachability

How to run recursive calls efficiently?

- Idea: reuse the space
- Generate the nodes z one after another
- Each time new z is used we reuse the space
- For a new z:
  - Add $(x, z, i-1)$ to the first work tape
  - Start working on this problem
  - If $REACH(x, z, i-1)$ is false go to the next z (erase the triples for this one)
  - If $REACH(x, z, i-1)$ is true, obtain y from the triple to the left and work on $REACH(z, y, i-1)$
  - If $REACH(z, y, i-1)$ is false move to next z else return true

First work string = activation record stack

Second work string = maintain indices

Savitch's theorem: fast reachability

How much does this take?

Recursion depth is $log n$ (path of length n)

Each triple is $3log n$, so the first work tape uses $log^2 n$

The second one just counts up to $log n^2$ (number of edges)

We get the desired bound $\square$

Savitch's theorem: proof done

We get Savitch's theorem by using the previous algorithm on $G(M, w)$

Observe:

- Size of $G(M, w)$ is $2^{O(f(n))}$
- So we run $REACH(start, accept, O(f(n)))$
- So it runs in $O(f^2(n))$
- Note that we have to repeat this for every (accepting) configuration
- But for this we reuse space again: we just need to count up to $2^{O(f(n))}$

$\square$

Still no $$$

As a corollary we get:

Corollary

PSPACE= NPSPACE.

But at least we can play games (in a few slides)

PSPACE-completeness

As before we are interested in complete problems.

Definition

A language B is **PSPACE-complete** if:

1. $B \in \text{PSPACE}$ and;
2. Every language $A \in \text{PSPACE}$ is reducible to B.

So which reduction do we have to use?

In this case it can be either one, so we use the easier.

Could we use PSPACE-reductions?

PSPACE-completeness: TQBF

A quantified boolean formula is of the form:

$$Q_1 x_1 Q_2 x_2 \dots Q_k x_k \varphi$$

where:

- $Q_i \in \{\forall, \exists\}$; and
- φ is a propositional formula using the variables $x_1, \dots, x_k$.

Semantics is defined as for first-order formulas.

Which one is true:

- $\exists y \forall x (x \vee y) \wedge (\neg x \vee \neg y)$
- $\forall y \exists x (x \vee y) \wedge (\neg x \vee \neg y)$

PSPACE-completeness: TQBF

$$\mathsf{TQBF} = \{\langle \phi \rangle \mid \phi \text{ is a true quantified boolean formula}\}$$

Theorem

TQBF is PSPACE-complete.

PSPACE-completeness: TQBF

Proof: Here is a PSPACE-machine for TQBF:
$M =$ On input $\langle\phi\rangle$, for ϕ a quantified boolean formula:

1. If there are no quantifiers evaluate the expression remaining
2. If $\phi = \forall x \varphi$ recursively call M on φ where each occurrence of x is replaced first by 1 then by 0. If both accept accept, otherwise reject
3. If $\phi = \exists x \varphi$ recursively call M on φ where each occurrence of x is replaced first by 1 then by 0. If either accepts accept, otherwise reject

Lower bound uses the idea from Cook-Levin to code configurations using variables

PSPACE-completeness: TQBF

Let A be a language decided by a DTM M running in space n^k

For a word w we construct a QBF that is true iff M accepts w

We use variables $x_{i,s}$ with i a tape position and s a tape symbol a, or a symbol a_q (recall HORNSAT)

A configuration c can be encoded using the variables $x_{i,s}$

If c_1 and c_2 are sets of variables and $t > 0$:

- We construct $\phi_{c_1,c_2,t}$
- Which is true iff M can go from c_1 to c_2 in at most t steps
- When c_1 and c_2 encode actual configurations of M

PSPACE-completeness: TQBF

A DTM using space $f(n)$ has at most $h = 2^{df(n)}$ configurations on input of length n

So we just use $\phi_{c_{init}, c_{accept}, h}$ for our reduction

For $t = 1, \phi_{c_1, c_2, t}$ is easy to construct (how?)

For $t > 1$ we want to split the formula in 2:

$$\phi_{c_1, c_2, t} = \exists m[\phi_{c_1, m, \lceil \frac{t}{2} \rceil} \wedge \phi_{m, c_2, \lceil \frac{t}{2} \rceil}]$$

Here $\exists m$ is a shorthand for $\exists x_{m,1} \exists x_{m,2} \ldots \exists x_{m,l}$ (m represents a configuration, so $l = n^k$)

Why does this not work?

PSPACE-completeness: TQBF

We use:

$$\phi_{c_1,c_2,t} = \exists m \forall (c_3, c_4) \in \{(c_1, m), (m, c_2)\}[\phi_{c_3,c_4,\lceil \frac{t}{2} \rceil}]$$

Here $\forall x \in \{y, z\}[\ldots] = \forall x[(x = y \lor x = z) \rightarrow \ldots]$

In each step the formula grows by the size of a configuration

And we have $log(2^{df(n)})$ steps, so total poly-size

$\square$

TQBF as a game

Consider the formula:

$$\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Q x_k [\varphi]$$

- Quantifiers are alternating
- ϕ is a game between Player E and Player A
- E selects values for $\exists$ and A for $\forall$ variables
- This defines a valuation of variables
- E wins if ϕ is true, A if it is false under this valuation

This is a game associated with ϕ

PSPACE and games

Example:

$$\phi = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})]$$

One play: E picks $x_1 = 1$, A $x_2 = 0$, E $x_3 = 1$; E wins

In fact: E can always win ($x_1 = 1$ and $x_3 = \neg x_2$)

E has a **wining strategy**

Winning strategy for E = E wins no matter how A plays

PSPACE and games

Example:

$$\phi = \exists x_1 \forall x_2 \exists x_3 [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3})]$$

Here A has a winning strategy ($x_2 = 0$)

PSPACE and games

$$\text{FORMULA-GAME} = \{\langle\phi\rangle \mid \text{ Player E has a winning strategy in a game associated with } \phi\}$$

Theorem

FORMULA-GAME is PSPACE-complete.

Proof: Show that this problem is the same as TQBF.

Hint: if the quantifiers do not alternate add ones that do and dummy clauses with variables they use

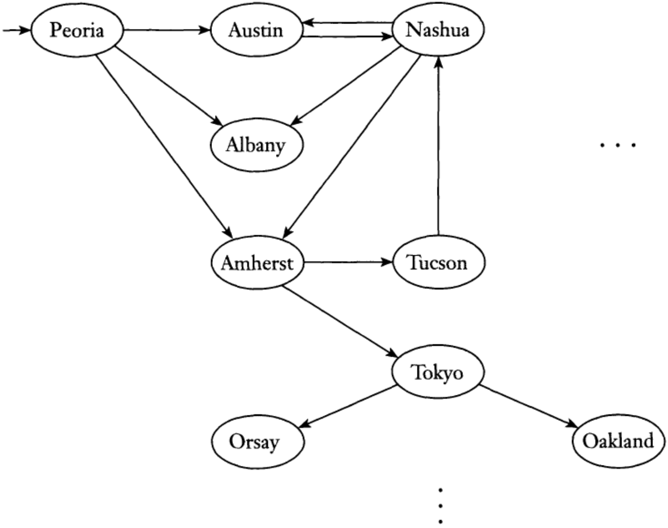
PSPACE and games: generalized geography

Game of geography:

- Player I names a city c
- Player II names a city d starting with the last letter of c
- Player I does the same for d
- They carry on
- No repetitions allowed
- If a player can't make a move he/she loses

PSPACE and games: generalized geography

Visually we are traversing the graph:

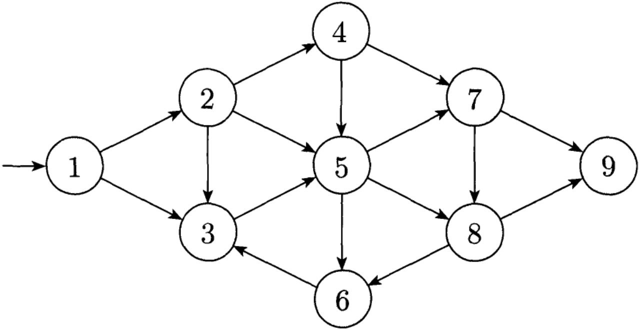


PSPACE and games: generalized geography

Abstraction of this:

- Take a graph with a designated node
- Player I moves from this node
- Then Player II, etc.
- No repeated nodes are allowed (a simple path)
- The goal is to force a player in a position with o further moves

PSPACE and games: generalized geography



Here Player I has a winning strategy

If the edge from 3 to 6 is reversed Player II has a winning strategy

PSPACE and games: generalized geography

$\mathsf{GG} = \{\langle G, b\rangle \mid$ Player I has a winning strategy for the generalized geography played on graph G starting at node $b\}$

Theorem

GG is PSPACE-complete.

Proof: Membership in PSPACE is similar as for TQBF.

How would this go?

PSPACE and games: generalized geography

PSPACE-hardness is more interesting

We do a reduction from FORMULA-GAME

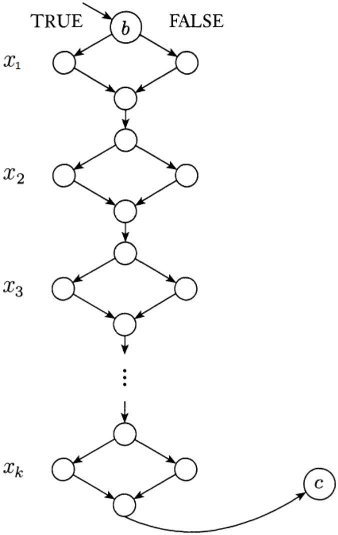
Take $\phi = \exists x_1 \forall x_2 \exists x_3 \ldots Q x_k [\varphi]$

Wlog a formula ϕ *starts and ends* with $\exists$ and alternates between $\exists, \forall$

Wlog φ is in conjunctive normal form

We construct a graph G such that Player I has a winning strategy iff ϕ is true

Generalized geography: left part of G



Generalized geography: left part of G

- One diamond per variable
- Player I starts (left means $x_1 = 1$, right 0)
- Then Player II goes down, then Player I
- Now Player II goes to diamond of x_2, etc.
- At the end we have $\exists x_k$
- So Player I's last move in this part is to c

This defines a valuation of $x_1, \ldots, x_k$

Now we move to the right side of the graph (checking if ϕ is true for this valuation)

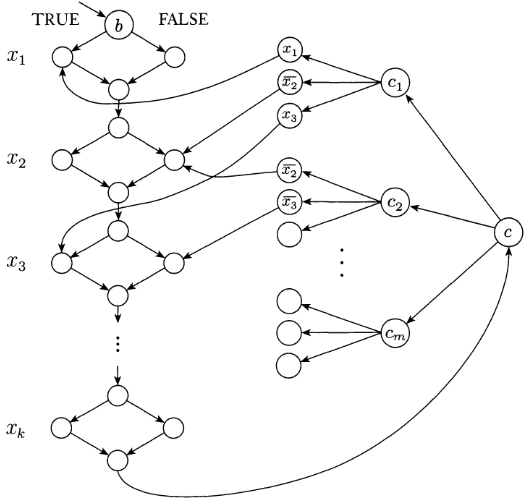
Generalized geography: right part of G

The right path has:

- A node for each clause (for II to pick)
- A node for each literal in the clause (for I to pick)
- An edge from c to clause
- An edge from clause to its literals
- An edge from literal to left side of G
- The last ones connect x_i to left side of the diamond for x_i
- And $\overline{x_i}$ with the right side of the diamond

Generalized geography: the graph G

$$\phi = \exists x_1 \forall x_k \dots \exists x_k [(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee \dots) \wedge \dots]$$



Generalized geography: the graph G

Show that ϕ is true iff Player I has a winning strategy in G, b

Note that the reduction is both poly-time and logspace

Pick either one

$\square$

More games

Standard boar games (chess, go) are easy for complexity

Note: board = fixed size, so no input variability

Complexity is dumb about this (DeepBlue)

But if we generalize we get PSPACE-complete problems

Which shows that the reasoning behind these games is difficult

A very instructive PSPACE-reduction

Notation

$L(r)$ is the language of a regular expression r.

A regular expression r_1 is contained in a regular expression r_2 if $L(r_1) \subseteq L(r_2)$.

- Example: $(01)^*$ is contained in $(0+1)^*$

Notation

$r_1 \subseteq r_2$: r_1 is contained in r_2.

A very instructive PSPACE-reduction

Let CONT-REG be the following problem:

$$\text{CONT-REG} = \{(r_1, r_2) \mid r_1 \text{ and } r_2 \text{ are regular expressions such that } r_1 \subseteq r_2\}$$

Important applications:

- Query optimisation over XML databases

Theorem (Meyer & Stockmeyer)

CONT-REG is PSPACE-complete.

Proof in Marcelo's slides (you know the upper bound)

Two important space complexity classes

Definition

LOGSPACE is the class of languages decided by a deterministic machine running in logarithmic space:

$$\text{LOGSPACE} = \text{DSPACE}(logn).$$

NLOGSPACE is the class of languages decided by a nondeterministic machine running in logarithmic space:

$$\text{NLOGSPACE} = \text{NSPACE}(logn).$$

The input tape is crucial here

Two important space complexity classes

These two classes are exact (unlike PTIME and NP):

- Why are they important?
- With logarithmic space we can maintain pointers to the input
- Of course, only a constant number of pointers
- But this is what the actual code usually does

Exercise

Show that $A = \{0^k 1^k \mid k \geq 0\}$ is in LOGSPACE.

An important NLOGSPACE problem

$\text{PATH} = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a path from } s \text{ to } t\}$

Algorithm for PATH:

1. Write s on the first work tape
2. Write 0 on the second work tape (counting)
3. Repeat until you reach t or have $n = |G|$ on second work tape:
4. Guess a node (non deterministically)
5. If it is not reachable from the one on tape 1 reject
6. If it is replace tape 1 with this node
7. Increase the counter on tape 2 by 1
8. If t was reached accept

NLOGSPACE-completeness

For NLOGSPACE-completeness we use LOGSPACE reductions

That is, B is NLOGSPACE-complete if:

1. $B \in$ NLOGSPACE and;
2. For every $A \in$ NLOGSPACE we have that $A \leq_L B$

Would polynomial time reductions work?

Can we define LOGSPACE-completeness in the same way?

NLOGSPACE-completeness: PATH

Theorem

PATH is NLOGSPACE-complete.

Proof: We know the upper bound.

We actually also know the lower bound (Savitch's theorem).

That is, we only need to construct the graph $G(M, w)$

Recall: in our reductions we use machines with output tapes, so the fact that $G(M, w)$ is huge will not matter

NLOGSPACE-completeness: PATH

Let M be a NLOGSPACE machine and w an input

Our reduction first lists all the nodes of $G(M, w)$:

- Each node is a configuration (of M on w)
- So is of size $c \cdot log|w|$, for some constant c
- We generate all strings of this length (one by one)
- And output the ones that are valid configurations of M

Wlog there is only one accepting configuration (how to achieve this?)

This is clearly using only LOGSPACE

NLOGSPACE-completeness: PATH

Next we generate all the edges:

- Try each pair (c_1, c_2) of configurations
- If M can move from c_1 to c_2 output the edge
- We check this by comparing the head position in c_1 with position in c_2

Start node = initial configuration; end node = accepting configuration

$\square$

NLOGSPACE-completeness: PATH

This gives us:

Corollary

NLOGSPACE $\subseteq$ *PTIME*.

Next, we will explore connections between complexity classes in detail