Tue April 17 Space Complexity

Space complexity classes; Savitch's theorem; Quantified Boolean Formulas

What is the space complexity of a deterministic Turing Machine M?
A function f(N) → N, where f(n) is the max number of tape cells M scans on any input of length n.

For a non-deterministic TM, what does f(n) measure?
max number of cells on any branch

SPACE(f(n)) = ?
SPACE(f(n)) = { L | L is a language decided by an O(f(n)) space deterministic TM }
NSPACE(f(n)) = ?

Claim: SAT is in SPACE(n)
Why?
- loop over all possible truth assignments
- requires O(n) space (one cell per variable, and original formula)

ALL_NFA = { <A> | A is a NFA and L(A) = Sigma* }

Note: NOT known to be in NP or co-NP.

Claim: complement ALL_NFA is in NSPACE(n)
Idea: guess string that is rejected
1. Place marker on start state
2. repeat 2^q times
3 Non-deter select an input symbol and change the positions of the markers
to simulate reading that symbol
4. Accept iff you reach a point where none of the markers lie on an accept state

Why only need to repeat 2^q times? Because in any longer string that would be rejected, the locations of the markers would repeat.

what is SAVITCH's THEOREM?

NSPACE(f(n)) subseteq SPACE(f^2(n))

IE: you can simulate a ND TM by a TM using very little space.

Naive approach: try each branch sequentially.
Problem: branch that uses f(n) space may run for 2^O(f(n)) steps.
Why? → because that is all possible configurations of the space
Each step may be a ND choice
Therefore there could be 2^f(n) ND choices to record,
so keeping track of the choices could require O(2^f(n)) space.

Alternative approach: divide and conquer recursive algorithm.
configuration = what is on tape, location of head, finite control state

Yieldability problem: can a NTM get from c1 to c2 in t steps?

```
CANYIELD(c1, c2, t):
 if t = 1 then
      test whether c1=c2 or c1 yields c2 in one step
      accept or reject accordingly
 else
      if t>1, then for each configuration cm that uses space f(n)
      run CANYIELD(c1, cm, t/2)
      run CANYIELD(cm, c2, t/2)
```

CANYIELD(c_start, c_accept, 2^(d f(n)))
selecting d so that N has no more than 2^(d f(n)) configurations on f(n) tapes

space requirements: at most log(2^(d f(n)) = O(f(n)) depth of recursions
each requires at most O(f(n)) space, so O(f^2(n)) overall. THUS:

NSPACE(f(n)) subseteq SPACE(f^2(n))

Tue April 17 Space Complexity Continued

SAVITCH'S THEOREM TAKE 2

Concepts:
O(f(n)) space → 2^O(f(n)) possible configurations → 2^O(f(n)) time → 2^df(n) for some fixed d

Start with O(f(n)) space NTM.
Simulate it with a deterministic TM using recursive algorithm:

CANYIELD(c_start, c_accept, 2^df(n))

Each recursive call cuts the time bound in half. Therefore, the maximum depth of the recursion stack is O(log(2^df(n))) = O(df(n)) = O(f(n)).

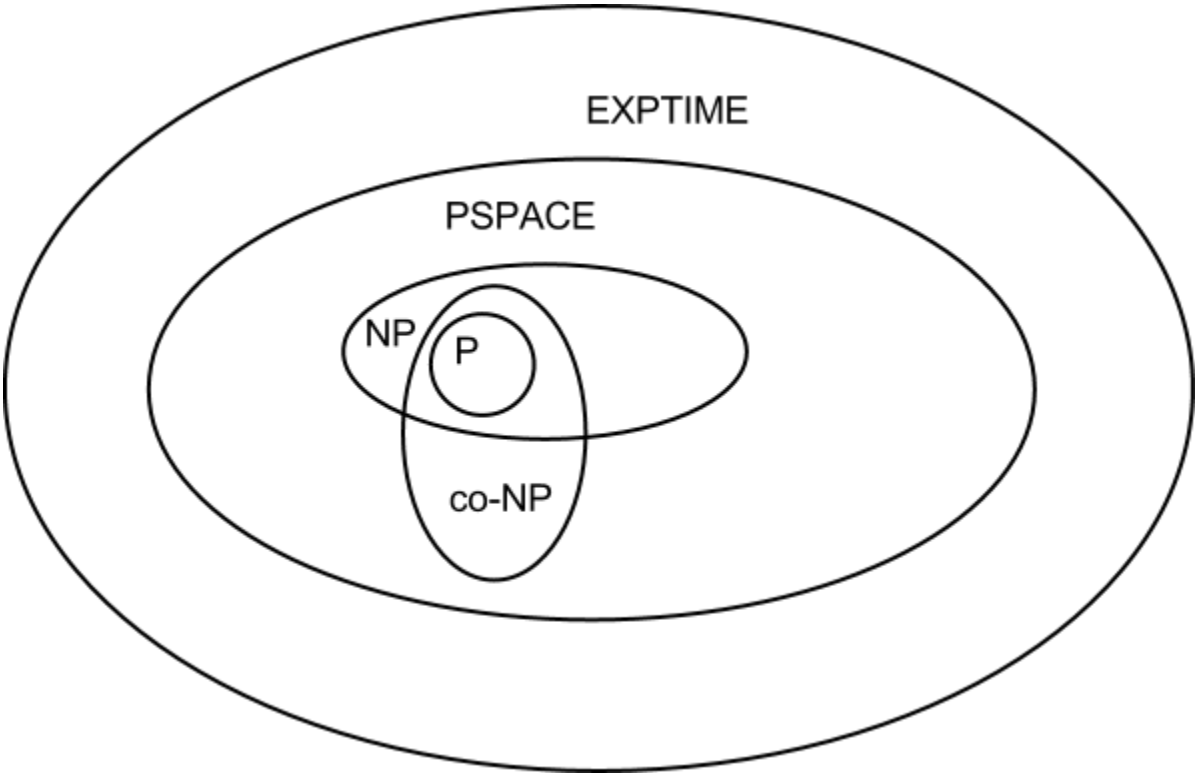
For each recursive call, the stack has to hold the current configuration, which requires O(f(n)) space.

So, the total space required to hold the stack is O(f(n)) x O(f(n)) = O(f^2(n)).

PSPACE = U SPACE(n^k)

Venn diagram:

P subseteq NP subseteq PSPACE = NPSPACE subsetq EXPTIME



PSPACE completeness:
1. B is in PSPACE
2. every A is PSPACE is **polytime** reducible to B

PSPACE hard → no need to check condition (1)

Quantified Boolean Formulas

all x exists y . ((x v y) & (~x v ~y))

A fully quantified formula is either TRUE or FALSE.

Exercise: true or false?

all x . exists y . (x v y)

all x . all y . (x v y)

all x . exists y . (x → y)

all x exists y . ((x v y) & (~x v ~y))

exists x all y . ((x v y) & (~x v ~y))

TQBF = < <phi> | phi is true fully quantified Boolean formula >

Theorem: TQBF is PSPACE-complete.

Why is it in PSPACE?
Simple backtracking recursive algorithm, try T or F for each quantifier.

Next: need to show that every language A in PSPACE reduces to TQBF in poly time. Given polyspace bounded TM for A, define a polytime mapping function from w (input to machine) to a QBF that encodes a simulation of the machine on w -- it is true iff machine accepts w.

Can we use a table of Configurations x Time to generate the encoding, as in Cook-Levin theorem?
No, because table could have exponential number of rows (time).

Idea: use approach inspired by SAVITCH theorem to construct a encoding by divide and conquer.

Given: M that runs in n^k for some fixed k.

Define:

$\phi_{c1,c2,t} =$ formula evaluates to true iff M can go from c1 to c2 in at most t steps.

The overall formula for the problem is $\phi_{cstart,caccept,2^{df(n)}}$ where f(n) = n^k

As in proof of Cook's theorem, formula encodes the contents of tape cells.
Each configuration has n^k cells, so each configuration can be encoded by O(n^k) variables.

For t=1, formula $\phi_{c1,c2,1}$ asserts that c1=c2 or c1 is followed by c2 in one step.

For t>1, construct the formula recursively:

$$\phi_{c1,c2,t} = \exists m\,[\phi_{c1,m,t/2} \wedge \phi_{m,c2,t/2}]$$

where $\exists m$ is shorthand for $\exists x1 \exists x2 \exists x3 \ldots \exists xl$ where the O(n^k) variables encode a configuration.

This is close to the solution, but the formula is too big - it doubles in size with each recursion.
Trick to get a small formula is to the universal quantifier to replace the two recursions with one:

$$\phi_{c1,c2,t} = \exists m\ \forall c3,c4\,.\,[(c3 = c1 \wedge c4 = m) \vee (c3 = m \wedge c4 = c2)] \rightarrow [\phi_{c3,c4,t/2}]$$

How big is the resulting formula $\phi_{cstart,caccept,2^{df(n)}}$? Each subformula is linear in the size of a configuration, so it is of size O(f(n)). There are log(2^df(n)) = O(f(n)) recursions. Thus, the whole formula is size O(f^2(n)).

Games

Relationship between quantifiers and games:

E x1 A x2 E x3 Ax4 F

Players E and A.
Each chooses a value for a variable in order from left to right.
E wins if F is true,
A wins if F is false.

A winning strategy = player can win when both sides play optimally.

E x1 A x2 E x3 [(x1 v x2) & (x2 v x3) & (~x2 v ~x3)]

Winning strategy for E: pick 1, then select x3 to the negation of whatever A selects for x2.

Formula-Game(F | Player E has a winning strategy for F) = TQBF = PSPACE complete.

Observation: Exactly one of the players always has a winning strategy in a formula game!
Why? (Every quantified formula evaluates to T or F!)

Other PSPACE complete games.

Geography: two players take turns naming cities, where next city begins with last letter of previous city, with no repetitions. Win if a player is stuck - no more unused cities with letter.
Peoria -> Amerst -> Tuscon -> New York -> Kansas City

How modeled as a graph?

each node = city
arrow A → B iff B starts with last letter of A

Simple path in the graph = does not repeat a node (city)

Generalized geography: arbitrary directed graph with designated start node.
CG = {<G,b> | Player I has a winning strategy for generalized geography played on G starting at node b}

Theorem: Generalized Geography is PSPACE complete.

Reduction with FORMULA-GAME -- which way?

FORMULA-GAME <=p GENERALIZED GEOGRAPHY

E x1 A x2 E x3 . F