Lecture 42
ECE 664

Purdue Univ.
A.C. Kak

Storage Considerations: The Classes PSPACE, PSPACE-Complete, and PSPACE-Hard of Problems

- So far we have only been concerned with the time it takes to solve a problem. But there is one more resource that is needed for solving problems — the amount of computer memory.
- All of the computational models we have used up to this point were based on different types of Turing machines. In Turing machine based computations, the time taken by an algorithm is the number of steps (with each step involving a state transition) of the Finite-State Control before the halt state is entered.
- The Turing machine based computational models are also convenient for analyzing the space requirements of an algorithm. (Space here means the same thing as computer memory.) We will define the space used by an algorithm as the number of distinct tape squares visited by the read/write head of the Finite-State Control Unit.

▶ Since the number of tape squares visited cannot exceed the total number of "time" steps in the computation, it follows that any problem that is solvable in polynomial time is also solvable in polynomial space.

In each state transition, the tape head can move at most one square either to the left or to the right.

▶ Since the computations in all of the problem classes NP, P^{NP}, NP^{NP}, and #P are supposed to be carried out in polynomial time on various types of Turing machines, (although admittedly with either random guesses, or with the help of oracles, or with a combination of both), the problems in all these problem classes should also be solvable in polynomial space.

- We use PSPACE to denote the class of all problems that can be solved in polynomial space. Evidently, PSPACE includes all of NP, P^{NP}, NP^{NP}, and #P.

- For a more precise definition:

> PSPACE is the class of all languages recognizable by polynomial-space bounded DTM programs that halt on all inputs.

Note that we place no constraints on the length of time it might take the DTM to solve the problem.

- Since PSPACE includes all of NP, P^{NP}, NP^{NP} and #P, it would be reasonable to suspect that the hardest problems in PSPACE would be very, very hard. We will refer to the hardest problems in PSPACE as PSPACE-Complete. A formal definition of PSPACE-Complete follows:

> A language L is PSPACE-Complete if $L \in$ PSPACE and if $L' \propto L$ for all $L' \in$ PSPACE.

- In order to discover problems that are PSPACE-Complete, we need at least one "seed" PSPACE-Complete problem. (That is, we need a problem that would play the same role in the class PSPACE as was played by SAT in class NP.) Our seed PSPACE-Complete problem is the problem of Quantified Boolean Formulas that we will describe next.

Quantified Boolean Formulas (QBF):

Instance: A well-formed Boolean formula

$$F = (Q_1 x_1)(Q_2 x_2) \cdots (Q_n x_n)\, E$$

where E is a Boolean expression involving the variables $x_1, x_2, \ldots x_n$ and where each quantifier Q_i is either "$\exists$" or "$\forall$". The Q_i's must alternate between "$\exists$" and "$\forall$"; any number of alternations are allowed.

Question: Is F true?

- Example of QBF: $F = \exists x\, \forall y\, \exists z\, P(x,y,z)$ where the predicate P tests whether z is the sum of x and y, and where we are provided with integer sets X, Y, and Z so that $x \in X$, $y \in Y$, $z \in Z$. The QBF question is whether F is true. So, basically, the question wants us to find out if for every $y \in Y$ there exists an $x \in X$ and an $z \in Z$ so that $x + y = z$.

- Because of the universal quantifiers, it is NOT possible to solve QBF on an NDTM in polynomial time. There is no alternative to testing the formula F for every value of all the universally quantified variables. Therefore, QBF $\notin$ NP

- Although QBF $\notin$ NP, we can easily establish that $\boxed{\text{QBF} \in \text{PSPACE}}$. This is a consequence of the fact that a copy of F can be evaluated in place by looping through the permissible values for all the variables in each of the quantified sets and terminating the evaluation process immediately if F evaluates to 'false' when it is not supposed to. In these evaluation loops, F must evaluate to 'true' for some value of an existentially quantified variable and every value of a universally quantified variable.

- Having established that QBF $\in$ PSPACE, in order to prove that QBF is also PSPACE-Complete, we must show that every language $L' \in$ PSPACE can be polynomially transformed to QBF. The proof of this will not be presented here; it is similar to the proof of the fact that every language $L' \in$ NP can be polynomially transformed to SAT.

Combinatorial games are a rich source of PSPACE-Complete problems

These are typically two-person games. The game itself can be described as a set of all possible "configurations" of a structure. Each player makes a move to change the current configuration to his/her advantage and/or to the adversary's disadvantage. The set of configurations includes an initial configuration that corresponds to the start of the game and possibly two subsets called winning configurations for each of the players.

- That QBF can be used to reason about the winning strategies can be seen from the following formal definition of "PLAYER$_1$ as the first player forces a win in n moves":

  - there is a move for PLAYER$_1$ from configuration C_0 to C_1 such that
  - for all moves of PLAYER$_2$ from C_1 to C_2
  - there is a move for PLAYER$_1$ from configuration C_2 to C_3 such that
  - for all moves of PLAYER$_2$ from C_3 to C_4
  - ⋮
  - there is a move for PLAYER$_1$ from C_{n-2} to C_{n-1} such that
  - for all moves of PLAYER$_2$ from C_{n-1} to C_n
  - configuration C_n is a winning configuration for PLAYER$_1$

  Note the same alternation between the existential and universal quantifiers that made QBF PSPACE-Complete.

- Presented next is a specific example of a combinatorial game that is played on a graph $G = (V, E)$: Given two vertices s and t, the idea of the game is for PLAYER$_1$ to select one vertex in each move so that, when the game is over, the set of vertices collected by him/her

contains a path between s and t. PLAYER$_2$ also selects a vertex at each move but with the intention of making it difficult for PLAYER$_1$ to find the vertices he/she needs. The game ends when there are no more vertices to be chosen by either player. PLAYER$_1$ wins if there exists a path between s and t in his/her share of the vertices. The problem of ascertaining whether or not PLAYER$_1$ will win the game is called GENERALIZED HEX:

Generalized Hex (GENHEX):

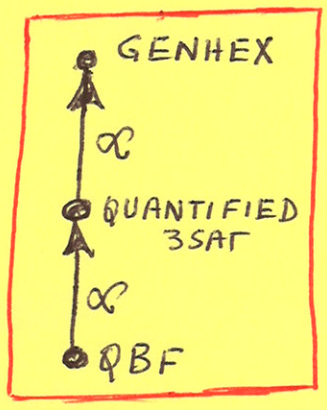
Hex is a board game, called so because it is played on a hexagonal grid, in which you form a connected path, usually from one side of the board to the opposite side, using tiles assigned to you.

Instance: An undirected graph $G=(V,E)$ and two specified vertices $s,t \in V$

Question: Using the game description below, can Player$_1$ force a win?

Game: The game configurations are partitions of V into three sets (V_1, V_2, V_3) with $\{s,t\} \subset V_3$. The initial configuration is (ϕ, ϕ, V) and all partitions of the form $(V_1, V_2, \{s,t\})$ are final. That is, we start with V_3 containing all vertices and we end the game when V_3 has just the vertices s and t. In a configuration (V_1, V_2, V_3) with $V_3 \neq \{s,t\}$, it is Player$_1$'s move if $|V_1|+|V_2|$ is even; otherwise it is Player$_2$'s move. Player$_1$ makes a move by choosing a vertex $v \in V_3 - \{s,t\}$ and changing the game configuration to $\boxed{(V_1 \cup \{v\},\ V_2,\ V_3 - \{v\})}$. Player$_2$ moves analogously. A final position $(V_1, V_2, \{s,t\})$ is a win for Player$_1$ if and only if the subgraph of G induced by $V_1 \cup \{s,t\}$ contains a path from s to t.

- GENHEX is in PSPACE because the game can be played in-place by associating markers with the vertices, one marker for each player, to indicate vertex ownership. GENHEX was proved to be PSPACE-Complete by a polynomial transformation from a problem called 'QUANTIFIED 3SAT' that, in turn was proved to be PSPACE-Complete by a polynomial transformation from QBF:→



- A problem Π is PSPACE-Hard if there exists another problem Π' that is PSPACE-Complete such that $\Pi' \propto_T \Pi$. As with NP-Hard problems, a PSPACE-Hard problem may not even be in PSPACE.