Counting problems

- \#SAT: Given a Boolean expression, count the number of distinct satisfying truth assignments
- \#HAM CYCLE: Given a graph or digraph, count the number of distinct Hamiltonian cycles
- \#Q: The counting problem associated with decision problem Q

Note: for each problem Q, we have Q \leq_T \#Q

Complexity class \#P

- \#Q: Given x, count how many distinct certificates y exist that show x \in Q
  - x \in SAT iff there exists a satisfying truth assignment y for Boolean expression x
  - x \in HAM CYCLE iff there exists a Hamiltonian cycle y for graph or digraph x
- \#P = \{\#Q : there exists a polynomial-time algorithm that determines whether or not given y is a certificate that shows x \in Q\}

\#P-completeness

- \#Q is \#P-complete if \#Q \in \#P and for each \#Q' \in \#P, \#Q' \leq_T \#Q
  - Note: Turing reduction is used here because mapping reductions apply to decision problems
- \#SAT is \#P-complete
- \#HAM CYCLE is \#P-complete
- For nearly all NP-complete problems Q, \#Q is \#P-complete
- There also exist problems Q \in P such that \#Q is \#P-complete

Matching

- Suppose G = (U,V,E) is a bipartite graph with |U|=|V| and E \subseteq U \times V
  - Example: U={boys}, V={girls}, E = \{(b,g) : b and g are compatible\}
- M \subseteq E is called a perfect matching of G if |M|=|U|=|V| and for all (u,v), (u',v') \in M, u=u' and v\neq v'
  - Example (continued): can all the boys and girls be paired with distinct compatible partners?

Matching (continued)

- MATCHING: Given a bipartite graph G, does G have a perfect matching?
  - MATCHING \in P
- \#MATCHING: Given a bipartite graph G, count the number of perfect matchings
  - \#MATCHING is \#P-complete
  - \#MATCHING is equivalent to computing the permanent of the adjacency matrix
Parity problems

- **⊕SAT**: Given a Boolean expression, does it have an odd number of distinct satisfying truth assignments?
- **⊕HAM CYCLE**: Given a graph or digraph, does it have an odd number of distinct Hamiltonian cycles?
- **⊕Q**: The parity problem associated with decision problem Q
  - ⊕Q = \{x : there are an odd number of certificates y that show x ∈ Q\}

Complexity class ⊕P

- ⊕P = \{Q : there exists a polynomial-time algorithm for ⊕Q\}
- For most problems #Q ∈ #P, we also have ⊕Q ∈ ⊕P
  - due to existence of "parsimonious reductions", which preserve the number of solutions

⊕P-completeness

- ⊕Q is ⊕P-complete if ⊕Q ∈ ⊕P and for all ⊕Q' ∈ ⊕P, ⊕Q' ≤m ⊕Q
- If #Q is #P-complete then ⊕Q is ⊕P-complete
  - ⊕SAT is ⊕P-complete
  - ⊕HAM CYCLE is ⊕P-complete
  - ⊕MATCHING is ⊕P-complete
  - again due to "parsimonious reductions"

Complement of ⊕P

- ⊕P = co-⊕P
  - Given a SAT instance over variables x₁,...,xₙ
  - Add new variable z to every existing clause
  - Also add new clauses (xₖ ∨ ¬z) for k=1,...,n
  - Each existing solution can be extended by letting z = false
  - Also, one extra solution is obtained by letting every variable = true
  - Number of solutions increases by exactly one (so changes odd to even, or even to odd)

Relationships to other classes

- P ⊆ NP ⊆ PH ⊆ P#P ⊆ PSPACE
  - Recall: #P contains non-decision problems
- P ⊆ ⊕P ⊆ P#P ⊆ PSPACE
- ⊕P is incomparable with NP and PH

Now for something completely different... looking inside P

- Define a Boolean circuit Cₙ
  - n inputs x₁,...,xₙ
  - one output
  - nodes are gates that compute Boolean functions (and, or, nand, nor, etc.)
  - directed acyclic graph (circuit is monotone, so no feedback)
Measures of circuit complexity

- Size = number of gates
- Depth = max number of gates along any path from input to output
- Fan-in = max in-degree of any gate
- Fan-out = max out-degree of any gate (not including an input line)

Languages of circuits

- \( L(C_n) = \) language of \( C_n = \{ x_1...x_n : C_n \text{ yields output 1 when fed input values } x_1,...,x_n \} \)
- Family of circuits \([C_n] = \bigcup \{ C_n : n \geq 1 \}\)
  - \([C_n]\) is called “uniform” if algorithms exist that can answer these questions in \(O(\log n)\) time:
    - Is there a path from node \(i\) to node \(j\) in \(C_n\)?
    - What type of gate is node \(j\) of \(C_n\)?
  - To allow infinite languages, \(L([C_n]) = \) language accepted by \([C_n] = \bigcup \{ L(C_n) : n \geq 1 \}\)

Circuit complexity classes

- \( NC_k = \{ L([C_n]) : [C_n] \text{ is uniform, polynomial size, } O(\log^k n) \text{ depth, and contains only unary } \neg, \text{ binary } \land, \text{ binary } \lor \text{ gates} \}\)
- \( AC_k = \{ L([C_n]) : [C_n] \text{ is uniform, polynomial size, } O(\log^k n) \text{ depth, and contains only unary } \neg, \text{ unbounded fan-in } \land, \text{ unbounded fan-in } \lor \text{ gates} \}\)
- \( TC_k = \{ L([C_n]) : [C_n] \text{ is uniform, polynomial size, } O(\log^k n) \text{ depth, and contains only unary } \neg, \text{ unbounded fan-in MAJORITY gates} \}\)

Relationships

- \( NC_k \subseteq AC_k \)
  - Obviously
- \( AC_k \subseteq TC_k \)
  - Simulate unbounded fan-in \land, \lor \text{ gates using a MAJORITY gate and extra “dummy” wires}

Relationships (continued)

- \( TC_k \subseteq NC_{k+1} \)
  - Simulate a MAJORITY gate using binary \land, \lor \text{ gates with depth } O(\log n)
  - Assume fan-in is } 2^{m-1}, \text{ otherwise add equal numbers of “dummy” 0 inputs and “dummy” 1 inputs}
  - Compute the sum and check if the high-order bit is 1

General circuit complexity

- \( NC = \bigcup \{ NC_k : k \geq 0 \} = \text{Nick’s Class} \)
- \( AC = \bigcup \{ AC_k : k \geq 0 \} = \text{Alternating Class} \)
- \( TC = \bigcup \{ TC_k : k \geq 0 \} = \text{Threshold Class} \)
- \( NC = AC = TC \subseteq P \)

- Equivalent definition: \( NC = \{ \text{problems solvable in polylog time using polynomial number of processors} \} = \{ \text{problems that can be efficiently parallelized} \} \)
### More complexity classes
- \(SC_k = \{\text{problems solvable in } O(\log^k n) \text{ space using polynomial time}\}\)
- \(SC = \bigcup \{SC_k : k \geq 0\} = \text{Steve's Class}\)
- Equivalently: \(SC = \{\text{problems solvable on polynomial-size circuits having polylog "width"}\}\)
- Open problem: Does \(NC = SC\)?

### Brief history
- Nick Pippinger defined a useful class but he did not give it any name
  - So Steve Cook began referring to it as \(NC\) in Pippinger's honor
- Later, Steve Cook defined a class and named it PLOPS
  - But Nick Pippinger didn't like that name, so he returned the favor and began referring to this class as \(SC\)

### Yet more complexity classes
- \(L = \text{DSPACE}(\log n)\)
- \(NL = \text{NSPACE}(\log n)\)
- \(NC_1 \subseteq L = SC_1 \subseteq NL = \text{co-NL} \subseteq AC_1\)
- Open problem: Does \(L = NL\)?
- \(\text{DTIME}(\log n)\) and \(\text{NTIME}(\log n)\) have apparently not been studied much
  - They can only contain problems that don't look at their complete input

### NL-complete problems (using logspace reductions)
- Directed graph reachability
  - Instance: Digraph \(G=(V,E)\), vertices \(x\) and \(y\)
  - Question: Does \(G\) contain a path from \(x\) to \(y\)?
- \(2\)-Satisfiability
  - Instance: Boolean expression \(E\) in CNF with exactly 2 terms per clause
  - Question: Does \(E\) have a satisfying truth assignment?

### P-complete problems (using logspace reductions)
- Circuit value problem
  - Instance: Circuit \(C\), input values \(x_1, \ldots, x_n\), target output \(z\)
  - Question: Does \(C\) yield output \(z\) when fed inputs \(x_1, \ldots, x_n\)?
- Non-empty context-free grammar
  - Instance: Context-free grammar \(G\)
  - Question: Does \(G\) generate any strings?

### Next week
- Randomized classes \(RP, \ ZPP, \ PP, \ BPP\)
- Interactive and zero-knowledge proofs
- Non-approximability (if time permits)