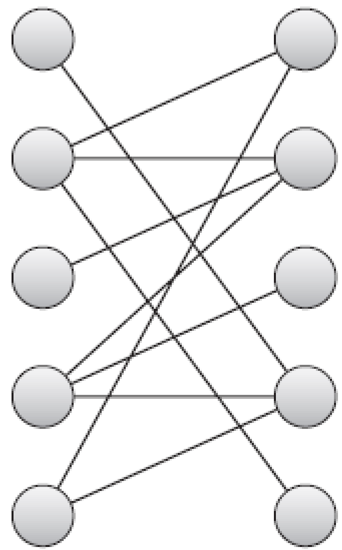
Complexity Theory



Agenda

Notation

- Big O
- Big Omega (Ω)
- Big Theta (Θ)

Big O Notation

- Suppose an algorithm's true run time performance is $f(N)$
- It has performance $O(g(N))$ if
$$f(N) < g(N) \times k$$
for some constant k and for N large enough
- In other words, the runtime function $f(N)$ is bounded above by $g(N)$

Big Omega Notation

- An algorithm has performance $\Omega(g(N))$ if

  $$f(N) > g(N) \times k$$

  for some constant k and for N large enough
- In other words, the runtime function f(N) is bounded below by g(N)

Big Theta Notation

- An algorithm has performance $\Theta(g(N))$ if it is $O(g(N))$ and $\Omega(g(N))$
- In other words, the run time function is bounded both above and below by $g(N)$

Notation Summary

- $O(g(N))$ Bounded above by $g(N)$
- $\Omega(g(N))$ Bounded below by $g(N)$
- $\Theta(g(N))$ Bounded above and below by $g(N)$

Complexity Classes

- Deterministic
- Nondeterministic

DTIME(f(N))

- Problems that can be solved by a deterministic computer in f(N) time
- For example, DTIME(N log N) includes problems that can be solved in O(N log N) time, such as sorting by using comparisons

P

- Problems that can be solved by a deterministic computer in polynomial time
- These are in some sense “solvable”

EXPTIME (or EXP)

- Problems that can be solved by a deterministic computer in exponential time ($O(2^{f(N)})$ for some polynomial function $f(N)$)

NTIME(f(N))

- Problems that can be solved by a nondeterministic computer in f(N) time
- For example, NTIME(N^2) includes problems in which an algorithm can guess the answer and verify that it is correct in O(N^2) time

NP

- Problems that can be solved by a nondeterministic computer in polynomial time
- For these problems, an algorithm guesses the correct solution and verifies that it works in polynomial time $O(N^P)$ for some power P

NEXPTIME (or NEXP)

- Problems that can be solved by a nondeterministic computer in exponential time
- For these problems, an algorithm guesses the correct solution and verifies that it works in exponential time $O(2^{f(N)})$ for some polynomial function $f(N)$

Space Classes

- DSPACE(f(N))
- PSPACE (polynomial space)
- EXPSPACE (exponential space)
- NPSPACE (nondeterministic polynomial space)
- NEXPSPACE (nondeterministic exponential space)

Class Relationships

- Clearly $P \subseteq NP$
- Less obviously:
  - PSPACE = NSPACE
  - EXPSPACE = NEXSPACE

- The big question:

Does $P = NP$?

Reductions

- Reduce solving one problem to solving another
- Polynomial time reductions are particularly important
- If you can reduce problem A to problem B in polynomial time, you write $A \leq_p B$

NP-Complete

- A problem is NP-complete if:
  - It is in NP
  - You can reduce every other problem in NP to it

SAT is NP-Complete

- The Cook-Levin theorem (or just Cook's theorem) proves that SAT is NP-complete
- For details, see: http://en.wikipedia.org/wiki/Cook-Levin_theorem
- Need to show:
  - SAT is in NP
  - All other problems in NP can be reduced to SAT

SAT is in NP

- SAT is in NP because you can guess the assignments for the variables and then, in polynomial time, verify that those assignments make the statement true

SAT Reducability, Part 1

- Suppose problem A is in NP. Then you can make a nondeterministic Turing machine with internal states that let it solve A.
- The idea behind the proof is to build a boolean expression that says:
  - The inputs are passed into the Turing machine
  - The states work correctly
  - The machine stops in an accepting state

SAT Reducability, Part 2

- Three kinds of variables:
  - T_{ijk} is true if tape cell i contains symbol j at step k
  - H_{ik} is true if the machine's read/write head is on tape cell i at step k
  - Q_{qk} is true if the machine is in state q at step k of the computation

SAT Reducability, Part 3

- Other terms represent how a Turing machine works
- For example, the tape can hold only 0s and 1s
- This statement:

  $(T_{001} \text{ AND NOT } T_{011}) \text{ OR } (\text{NOT } T_{001} \text{ AND } T_{011})$

  means that cell 0 at step 1 contains a 0 or a 1 but not both

SAT Reducability, Part 4

- Other parts mean:
  - The read/write head is in a single position at each step
  - The machine starts in state 0
  - The read/write head starts at tape cell 0
  - Etc.
- The full boolean expression is equivalent to the original Turing machine for
- the problem in NP. In other words, if you set the values of the variables Tijk to

SAT Reducability, Part 5

- If you set the values of the variables T_{ijk} to represent a series of inputs, the truth of the boolean expression tells you whether the original Turing machine would accept those inputs
- This reduces problem A to the problem of determining whether the boolean expression can be satisfied

Showing Other Problems are NP-Complete

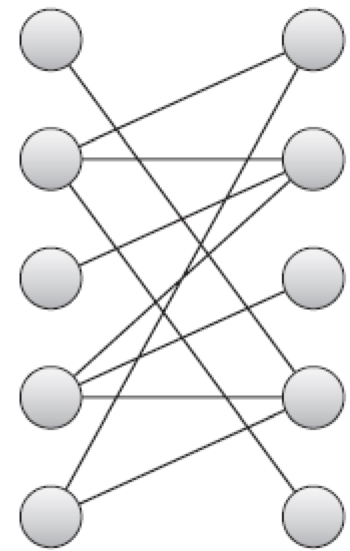
- Now that you know SAT is NP-complete, you can reduce it to other problems to show they are NP-complete
- Once you have other NP-complete problems, you can use them, too

3SAT

- 3SAT
  - Three-term conjunctive normal form (3CNF)
  - (A OR B OR NOT C) AND (C OR NOT A OR B)

Bipartite Matching

- A matching is a set of links, no two of which share a common end point.
- Given a bipartite graph and a number k, is there a matching that contains at least k links?



NP-Hardness

- A problem is NP-hard if every other problem in NP is polynomial-time reducible to it

Detection, Reporting, and Optimization Problems

- For the subset sum problem:
  - Detection—Is there a subset of the numbers that adds up to a specific value k?
  - Reporting—Find a subset of the numbers that adds up to the specific value k (if such a subset exists)
  - Optimization—Find a subset of the numbers with a total as close to the specific value k as possible

Detection $\leq_p$ Reporting

- (Detection is reducible to reporting)
- Suppose algorithm Report(k) returns a subset with values that add up to k
- Detect(k) calls Report(k) and returns true if Report(k) finds such a subset

Reporting $\leq_p$ Optimization

- (Reporting is reducible to optimization)
- Suppose algorithm Optimize(k) returns a subset with sum as close as possible to k
- Report(k) calls Optimize(k) and returns the subset if its total value is k

Reporting $\leq_p$ Detection

- (Reporting is reducible to detection)
- Suppose algorithm Detect(k) returns true if there is a subset with sum k

Reporting $\leq_p$ Detection Algorithm

1. Call Detect(k) on the whole set to see if a solution is even possible
2. If a solution is possible, for each value V_i in the set:
   a. Remove V_i from the set and call Detect(k) for the remaining set to see if there is still a subset with total value k
   b. If Detect(k) returns false, restore V_i to the set, and continue the loop at Step 2
   c. If Detect(k) returns true, leave V_i out of the set, and continue the loop at Step 2

When the loop finishes, the remaining values make a set with total value k

Optimization $\leq_p$ Reporting

- (Optimization is reducible to reporting)
- Suppose algorithm Report(k) returns a subset with total value k (if one exists)

1. For i = 0 To N, where N is the number of items in the set:
   a. If Report(k + i) returns a subset, Optimize(k) returns that subset.
   b. If Report(k – i) returns a subset, Optimize(k) returns that subset.
   c. Continue the loop in Step 1.

Approximate Optimization

- Detection – Is there a solution?
- Reporting – Find a solution.
- Optimization – Find the closest solution.

NP-Complete Problems

- More than 3,000 known NP-complete problems

- Art gallery problem—Find the minimum number of guards needed
- Bin packing—Pack objects in the fewest bins possible
- Bottleneck TSP—Find a Hamiltonian path with minimum largest link cost
- Chinese postman problem (route inspection problem)—In a network, find the shortest circuit that visits every link
- Chromatic number (or vertex coloring)—Given a graph, find the smallest
- number of colors needed to color the graph’s nodes. (The graph is not
- necessarily planar.)
- Clique—In a graph, find the largest clique (mutually connected nodes)
- Clique cover problem—Given a number k, find a way to partition a graph into k cliques.

NP-Complete Problems, Part 2

- Degree-constrained spanning tree—Find a spanning tree with a given maximum degree
- Dominating set—Given a graph, find a set of nodes S so that every other node is adjacent to one of the nodes in the set S
- Feedback vertex set—Given a graph, find the smallest set S of vertices that you can remove to leave the graph free of cycles
- Hamiltonian completion—Find the minimum number of edges you need to add to make a graph Hamiltonian (contains a Hamiltonian path).
- Hamiltonian cycle—Determine whether there is a path through a graph that visits every node exactly once and then returns to its starting point
- Hamiltonian path (HAM)—Determine whether there is a path through a graph that visits every node exactly once

NP-Complete Problems, Part 3

- Job shop scheduling—Given N jobs and M identical machines, schedule the jobs for the machines to minimize the total time to finish all the jobs
- Knapsack—Given a knapsack with a capacity and a set of objects with weights and values, find the set of objects with the largest possible value that fits in the knapsack.
- Longest path—Find the longest path that doesn’t revisit any nodes
- Maximum independent set—Find the largest set of nodes where no two nodes in the set are connected by a link
- Maximum leaf spanning tree—Find a spanning tree that has the maximum possible number of leaves
- Minimum degree spanning tree—Find a spanning tree with the minimum possible degree

NP-Complete Problems, Part 4

- Minimum k-cut—Given a number k, find the minimum weight set of edges that you can remove to divide the graph into k pieces
- Partitioning—Given a set of integers, find a way to divide the values into two sets with the same total value
- Satisfi ability (SAT)—Given a boolean expression containing variables, find an assignment of true and false to the variables to make the expression true
- Shortest path—Given a (not necessarily planar) network, find the shortest path between two given nodes
- Subset sum—Given a set of integers, find a subset with a given total value
- Three-partition problem—Given a set of integers, find a way to divide the set into triples that all have the same total value
- Three-satisfiability (3SAT)—Given a boolean expression in 3CNF, find an assignment of true and false to the variables to make the expression true

NP-Complete Problems, Part 5

- Traveling salesman problem (TSP)—Given a list of cities and the distances between them, find the shortest possible route that visits all the cities and returns to the starting city
- Unbounded knapsack—Similar to the knapsack problem, except that you
- can select any item multiple times
- Vehicle routing—Given a set of customers and a fleet of vehicles, find the most efficient routes for the vehicles to visit all the customers
- Vertex cover—Find a minimal set of vertices so that every link in the graph touches one of the selected vertices

Summary

- Notation
- Complexity Classes
- Reductions
- NP-Hardness
- Detection, Reporting, and Optimization Problems
- NP-Complete Problems

Exercises

- Chapter 17 Exercises 1 – 6, 9, 10.
- Bonus: Chapter 17 Exercises 11, 12.
- Double Bonus: Chapter 17 Exercises 7, 8, 13.
- Read *Essential Algorithms, 2e* Chapter 18 pages 561 – 594. (All of Chapter 18.)