CSE 421: Introduction to Algorithms

Complexity and Representative Problems

Winter 2005
Paul Beame

Measuring efficiency: The RAM model

- RAM = Random Access Machine
- Time = # of instructions executed in an ideal assembly language
  - each simple operation (+,*,=,if,call) takes one time step
  - each memory access takes one time step

Complexity analysis

- Problem size $N$
  - Worst-case complexity: max # steps algorithm takes on any input of size $N$
  - Best-case complexity: min # steps algorithm takes on any input of size $N$
  - Average-case complexity: avg # steps algorithm takes on inputs of size $N$

Stable Matching

- Problem size
  - $N$=2N$^2$ words
    - 2n people each with a preference list of length n
    - $2n\log n$ bits
    - specifying an ordering for each preference list takes $n\log n$ bits
  - Brute force algorithm
    - Try all $n!$ possible matchings
  - Gale-Shapley Algorithm
    - $n^3$ iterations, each costing constant time
      - For each man an array listing the women in preference order
      - For each woman an array listing the preferences indexed by the names of the men

Complexity

- The complexity of an algorithm associates a number $T(N)$, the best/worst/average-case time the algorithm takes, with each problem size $N$.

- Mathematically,
  - $T$ is a function that maps positive integers giving problem size to positive real numbers giving number of steps.

Efficient = Polynomial Time

- Polynomial time
  - Running time $T(N) \leq cN^k+d$ for some $c,d,k>0$
- Why polynomial time?
  - If problem size grows by at most a constant factor then so does the running time
    - E.g. $T(2N) \leq c(2N)^k+d \leq 2^k(cN^k+d)$
    - Polynomial-time is exactly the set of running times that have this property
  - Typical running times are small degree polynomials, mostly less than $N^3$, at worst $N^6$, not $N^{100}$
Complexity

O-notation etc

5 Representative Problems

Interval Scheduling

Interval scheduling

Interval Scheduling

5

9

10

11

12

Given two positive functions \( f \) and \( g \):
- \( f(N) \) is \( O(g(N)) \) iff there is a constant \( c > 0 \) so that \( f(N) \) is eventually always \( \leq c \cdot g(N) \)
- \( f(N) \) is \( o(g(N)) \) iff the ratio \( f(N)/g(N) \) goes to 0 as \( N \) gets large
- \( f(N) \) is \( \Omega(g(N)) \) iff there is a constant \( c > 0 \) so that \( f(N) \geq c \cdot g(N) \) for infinitely many values of \( N \)
- \( f(N) \) is \( \Theta(g(N)) \) iff \( f(N) \) is \( O(g(N)) \) and \( f(N) \) is \( \Omega(g(N)) \)

Note: The definition of \( \Omega \) is the same as “\( f(N) \) is not \( o(g(N)) \)”

In Interval Scheduling:
- Single resource
- Reservation requests
- Of form “Can I reserve it from start time \( s \) to finish time \( f \)?”
- \( s < f \)
- Find: maximum number of requests that can be scheduled so that no two reservations have the resource at the same time

Formally:
- Requests 1, 2, ..., \( n \)
  - request \( i \) has start time \( s_i \) and finish time \( f_i \)
  - \( s_i < f_i \)
- Requests \( i \) and \( j \) are compatible if either:
  - request \( i \) is for a time entirely before request \( j \)
  - \( f_i \leq s_j \)
  - request \( j \) is for a time entirely before request \( i \)
  - \( f_j \leq s_i \)
- Set \( A \) of requests is compatible if every pair of requests \( i, j \in A \), \( i \neq j \) is compatible
- Goal: Find maximum size subset \( A \) of compatible requests
Weighted Interval Scheduling
- Same problem as interval scheduling except that each request $i$ also has an associated value or weight $w_i$
  - $w_i$ might be
    - amount of money we get from renting out the resource for that time period
    - amount of time the resource is being used
- Goal: Find compatible subset $A$ of requests with maximum total weight

Bipartite Matching
- A graph $G=(V,E)$ is bipartite iff
  - $V$ consists of two disjoint pieces $X$ and $Y$ such that every edge $e$ in $E$ is of the form $(x,y)$ where $x \in X$ and $y \in Y$
  - Similar to stable matching situation but in that case all possible edges were present
  - $M \subseteq E$ is a matching in $G$ iff no two edges in $M$ share a vertex
  - Goal: Find a matching $M$ in $G$ of maximum possible size

Independent Set
- Given a graph $G=(V,E)$
  - A set $I \subseteq V$ is independent iff no two nodes in $I$ are joined by an edge
  - Goal: Find an independent subset $I$ in $G$ of maximum possible size
- Models conflicts and mutual exclusion

Weighted Interval Scheduling
- Ordinary interval scheduling is a special case of this problem
  - Take all $w_i = 1$
- Problem is quite different though
  - E.g. one weight might dwarf all others
  - “Greedy algorithms” don’t work
- Solution: “Dynamic Programming”
  - builds up optimal solutions from smaller problems using a compact table to store them

Bipartite Matching
- Models assignment problems
  - $X$ represents jobs, $Y$ represents machines
  - $X$ represents professors, $Y$ represents courses
  - If $|X|=|Y|=n$
    - $G$ has perfect matching iff maximum matching has size $n$
- Solution: polynomial-time algorithm using “augmentation” technique
  - also used for solving more general class of network flow problems

Independent Set
- Generalizes
  - Interval Scheduling
    - Vertices in the graph are the requests
    - Vertices are joined by an edge if they are not compatible
  - Bipartite Matching
    - Given bipartite graph $G=(V,E)$ create new graph $G'=(V',E')$ where
      - $V'=E$
      - Two elements of $V'$ (which are edges in $G$) are joined if they share an endpoint in $G$
**Bipartite Matching vs Independent Set**

\[ G = (U \cup V, E) \]

\[ G' = (V', E') \]

**Independent Set**

- No polynomial-time algorithm is known
- But to convince someone that there was a large independent set all you’d need to do is show it to them
- They can easily convince themselves that the set is large enough and independent
- Convincing someone that there isn’t one seems much harder
- We will show that Independent Set is NP-complete
  - Class of all the hardest problems that have the property above

**Competitive Facility Location**

- Two players competing for market share in a geographic area
  - e.g. McDonald’s, Burger King
- Rules:
  - Region is divided into \( n \) zones, 1,...,\( n \)
  - Each zone \( i \) has a value \( b_i \)
  - Revenue derived from opening franchise in that zone
  - No adjacent zones may contain a franchise i.e., zoning regulations limit density
  - Players alternate opening franchises
- Find: Given a target total value \( B \) is there a strategy for the second player that always achieves \( \geq B \)?

**Competitive Facility Location**

- Model geography by
  - A graph \( G=(V,E) \) where
    - \( V \) is the set \{1,...,\( n \}\) of zones
    - \( E \) is the set of pairs \((i,j)\) such that \( i \) and \( j \) are adjacent zones
- Observe:
  - The set of zones with franchises will form an independent set in \( G \)

**Competitive Facility Location**

- Checking that a strategy is good seems hard
  - You’d have to worry about all possible responses at each round!
    - a giant search tree of possibilities
- Problem is PSPACE-complete
  - Likely strictly harder than NP-complete problems
  - PSPACE-complete problems include
    - Game-playing problems such as \( n \times n \) chess and checkers
    - Logic problems such as whether quantified boolean expressions are always true
    - Verification problems for finite automata