Welcome to CSE 589 -- Applied Algorithms

Anna R. Karlin (karlin@cs)
Office Hours: Wed 9:20 - 10:00 pm
and by appointment

TA's: Nitin Sharma (nitin@cs)
Ka Yee Yeung (kayee@cs)
Office Hours: W 5:30-6:20pm

Administrivia

- Books, etc.
- Grading:
  - Weekly problem sets 40%
    - Can substitute JAVA applet for 2 problem sets.
    - Can substitute 20 minute presentation for 2 problem sets.
  - Quality of transcribed notes 20%
  - Final 40%

Other Announcements

- Please come to class next week (October 8).
  Free Pizza!!!!!!!!!
- No class on October 22.

Course Goals

- An appreciation for applications of algorithmic techniques in the real world.
- A larger toolbox.
- A better sense of how to model problems you encounter as well-known algorithmic problems.
- A deeper understanding of the issues and tradeoffs involved in algorithm design.
- FUN!!!!!!!!!!!!!!

Other

- Experimental, new course =>

  Very open to suggestions, feedback, constructive criticism. (E.g., topics/depth vs. breadth)
- Ask lots of questions! Participate! Offer opinions!
- Keeping you awake?

Plan For Today

- Whirlwind version of 421
  - Intro to algorithm design and analysis
  - Dynamic programming
  - Easy graph problems
  - Breakout
  - Hard graph problems

Algorithm Design Goals

- correctness
- efficiency

According to the *Oxford English Dictionary*

algorist "one skillful in reckonings or figuring"

Keeping Score: The RAM Model of Computation

- Each simple operation takes 1 time step.
- Loops and subroutines are not simple operations.
- Each memory access takes one time step, and there is no shortage of memory.

For a given problem instance:

- Running time of an algorithm = # RAM steps (ops)
- Space used by an algorithm = # RAM memory cells

useful abstraction => allows us to analyze algorithms in a machine independent fashion.

Types of complexity

Function from size of instance to real numbers

- Worst-case complexity
- Best-case complexity
- Average-case complexity

Big Oh Notation

- Goals :
  - ignore details that do not impact comparisons of algorithms
  - ignore constant factors
- **f(n) = O(g(n)) <=>** cg(n) is upper bound on f(n)

  <=> There exist c, N s.t. for n>=N, f(n) <= cg(n)
- **f(n) = Ω(g(n)) <=>** cg(n) is lower bound on f(n)

  <=> There exist c, N s.t. for n>=N, f(n) >= cg(n)
- **f(n) = Θ(g(n)) <=>** f(n) **= O(g(n)) and f(n) = Ω(g(n))**

  <=> There exist c_1, c_2, N s.t. for n>=N,

  c_1 g(n) <= f(n) <= c_2g(n)

Growth Rates, etc.

- Even by ignoring constant factors, can get an excellent idea of whether a given algorithm will be able to run in a reasonable amount of time on a problem of a given size.
- The "big Oh" notation and worst-case analysis are tools that greatly simplify our ability to compare the efficiency of algorithms.

Modeling the Problem

- **An art, but key to applying algorithm design techniques to real-world problems.**
- **Most algorithms designed to work on rigorously defined abstract structure.**

Two basic paradigms

Break problem down into smaller, more easily solved pieces.

- **Divide and conquer**
  typically, split problem in half, solve each half, combine results to get full solution

- **Dynamic programming**
  - **characterize structure of an optimal solution**
  - **recursively define value of an optimal solution**
  - **compute the value of an optimal solution in bottom-up fashion**
  - **construct an optimal solution from computed information**

Example DP Problem: Matrix Chain Multiplication

- Given a sequence $A_1, A_2, \ldots, A_n$, where matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product in a way that minimizes the number of scalar operations.

```
Matrix-Multiply(A,B)
  for i = 1 to rows [A]
    for j = 1 to columns [B]
      C[i,j] = 0;
      for k = 1 to columns [A]
        C[i,j] = C[i,j] + A[i,k]*B[k,j]
```

Dynamic Program for Matrix Chain Multiplication

- m[i,j] = minimum number of scalar multiplications needed to compute $A_{i..j}$
- Solution =
- m[i,j] =

Example: Approximate String Matching

- P -- a pattern string
- T -- a text string
- Edit distance between P and T is smallest number of changes to transform a substring of T into P, where changes are:
  - Substitution
  - Insertion
  - Deletion

Dynamic Program for AST

- D[i,j] = min # differences between P[1..i] and segment of T ending at j.

D[i,j] =

Dynamic Programming Summary

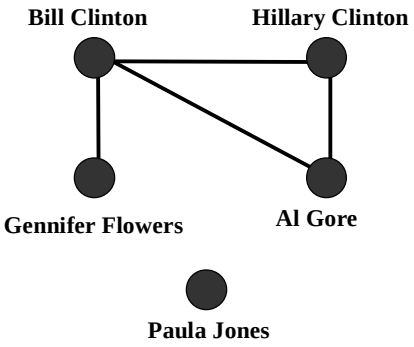
1. Formulate answer as recurrence relation or recursive algorithm
2. Show that number of values of recurrence bounded by poly
3. Specify order of evaluation for recurrence so have partial results when you need them.

- DP works particularly well for optimization problems with inherent left to right ordering among elements.
- Resulting global optimum often much better than solution found with heuristics.
- Once you understand it, usually easier to work out from scratch a DP solution than to look it up.

Graph Terminology

Graph G=(V,E) is set of vertices V, with set of edges E.

- undirected, directed
- weighted
- path
- connected
- strongly connected
- degree
- cycle
- tree



A friendship graph

Traversing a Graph

- Key idea: mark each vertex when it's visited, keep track of what's been explored. Vertex states:
  - undiscovered
  - discovered
  - fully-explored
- maintain structure of vertices discovered but not completely explored.
  - Queue
  - Stack
- Regardless of order, every edge considered twice.

Depth-First Search

```
DFS(G,u)
  state[u] = "discovered";
  for each v in Adj[u] do
      if state[v] ="undiscovered"
        parent[v] = u;
        DFS(G,v);
  state[u] = "fully-explored"
```

```
DFS-graph(G)
   for each vertex u in V[G] do
      state[u] = "undiscovered";
   for each vertex u in V[G] do
      if state[u] = "undiscovered"
          DFS[G,u];
```

Depth-First-Search

- Undirected graphs
  - DFS tree plus back edges
- Directed graphs
  - DFS tree plus
    - back edges
    - cross edges
    - forward edges

Applications of Graph Traversal

- Connected components
- Tree and Cycle Detection

Applications of Graph Traversal

- Two-coloring graphs

vertex-coloring -- assign color to each vertex so that no edge links two vertices of same color.

Use minimum number of colors.

- Topological Sort of DAG --

total order consistent with partial order

Topological Sorting

- DFS, label vertices in reverse order they are marked fully-explored.
- Consider edge u->v, encountered during exploration of u
  - v undiscovered => DFS(v) before continuing with u => v marked fully-explored before u
  - v discovered, not fully-explored =>
  - v fully-explored => labeled before u => u before v in topological order.

Applications of Graph Traversal

- Finding an articulation point -- a vertex whose deletion disconnects the graph
- More generally, connectivity of graph -- smallest number of vertices whose deletion disconnects graph.
- Strongly connected components of a directed graph -- a SCC of a graph is a maximal set of vertices U such that for every u,v in U, there is a path from u to v and a path from v to u.

Planarity Testing

- Given a graph G, can G be drawn in the plane such that no two edges cross?
- O(|V| + |E|), using depth-first search in subtle and clever way.

Minimum Spanning Trees

- Spanning tree of G -- subgraph of G that has same set of vertices as G and is a tree.
- MST of weighted graph G -- spanning tree of G whose edges sum to minimum weight
- greedy algorithms give optimal solution

Prim's algorithm

Start with arbitrary vertex
While there are vertices in tree connected to vertices outside
    select minimum weight edge between tree and outside
    add selected edge and vertex to tree

- Correctness ?
- Running Time -- trivial O(nm)
  Better?

Priority Queue

- Data structure for maintaining a set S of elements, each with an associated key
- Ops supported:
  - Insert(S,x)
  - Minimum(S)
  - Extract-Minimum(S)
- One implementation: heap
  - complete binary tree stored in an array A
  - satisfies heap property: A[*parent* [i]] <= A[i]

For MST, need Decrease-Key(x) as well.

Kruskal's algorithm

Sort edges in order of increasing weight
count = 0
While (count < n-1)
    get next edge (u,v)
    if u and v are in different components
        add edge to tree, and merge components

- Correctness ??
- Running Time -- trivial O(nm)
  Better?

Disjoint Set Data Structure Union/Find

- Data structure for maintaining a set of disjoint dynamic sets, each with representative
- Ops supported:
  - Make-Set(x)
  - Union(x,y)
  - Find-Set(x)
- One implementation:
  with union-by-rank and path compression, running time for m disjoint-set operations on n sets is
  $O(m\ \alpha(m,n))$ $\alpha(m,n)$ = inverse Ackermann f'n

Shortest Paths
Dijkstra's Algorithm

- For case where all edge weights are non-negative
- Finds shortest path from given vertex v to others
- At the beginning of an iteration, have a set of vertices S to which know the shortest path from v => grow set by one.
- Dynamic programming-like strategy

All-pairs shortest paths
Floyd-Warshall

- Slick dynamic programming algorithm
- $D[i,j]^k$ = length of shortest path from i to j using only vertices numbered 1,..,k as intermediate vertices

Matching

- Input: a weighted graph G=(V,E)
- Output: Largest-size set of edges M from E such that each vertex in V is incident to at most one edge of S

Graphs -- Summary

- can represent any relationship
- properly formulated, most applications of graphs can be reduced to standard graph properties and solved using well-known algorithms.

Your turn!!!!

To try your hand at modeling/solving problems using what we've seen today ….

Problems

1. Speech recognition problem -- distinguishing between words that sound alike (e.g. to, two, too). You're given a list of words and the likelihood of transitions between them. For each word spoken into the microphone, your program determines the set of words that sound like it. How would you figure out what sentence was spoken?
2. Given a set of processes that need to be scheduled, and dependencies that tell you which processes need to be completed in order for a given process to execute, how would you schedule the processes so as not to violate any of the dependencies?
3. How would you find your way out of a maze?

Problems

4. Network reliability -- how would you determine if there is a single point of failure in your network?
5. In an optical character recognition system, a method for separating lines of text is needed. Although there is some white space between the lines, problems like noise and the tilt of the page make the separation difficult to find. How would you go about doing line segmentation?
6. DNA sequences -- given experimental data consisting of small fragments. For each fragment f, know that certain other fragments are forced to lie to the left of f, certain fragments forced to lie to right, and rest either way. How do you find a consistent ordering of the fragments from left to right that satisfies all the constraints?

Problems

7. Multicast -- How would you decide what routes to use to send a message to a subset of the processors in your network? You can assume that you have information about the bandwidth and congestion on the various links in the network.
8. How would you design natural routes for a video-game character to follow through an obstacle-filled room?
9. How might you test if a certain set of conditions in your program can cause deadlock? For every pair of processes, A and B, it is possible to determine if A might ever wait for B.
10. Circuit simulation -- Given a circuit layout and input/output information for each gate in the circuit, (where its outputs go and where its inputs come from) how would you schedule the simulation of the gates in the circuit?

Problems

10. Video-on-demand servers --You have a set of storage devices, each storing some number of movies. Suppose the movies are replicated to a limited extent across the storage devices, and that each storage device has a certain bandwidth (number of movie requests it can serve per unit time) . Given a set of requests, i.e., a set of movies clients wish to watch, how would you go about figuring out how to satisfy a maximum number of requests at the same time?
11. Protein Function Determination -- When biologists discover a new protein, they usually try to figure out its function by finding known proteins (with known function) that have similar amino acid sequences. How would you go about assessing how similar two amino acid sequences are?

Hard Graph Problems

- Provably NP-complete problems --

  If any one can be solved in polynomial time, then all of them can.
- Right now, no known efficient algorithm known. Biggest open problem in CS:

  P = NP?
- Heuristics/approximation algorithms typically used.

Most notorious hard graph problem

- **Traveling Salesman Problem**
  - Input description: A weighted graph G
  - Output description: Find the cycle of minimum cost that visits each of the vertices exactly once.
- Example: optimization of tool path for manufacturing equipment. E.g., robot arm assigned to solder all connections on printed circuit board.

Clique

- Input description: A graph G=(V,E)
- Problem description: What is the largest subset S of V such that for all x,y in S, (x,y) in E.
- Example:

  IRS detection of organized tax fraud. Graph has vertices corresponding to submitted tax forms, and edges between pairs that are suspiciously similar.

Independent Set

- Input description: A graph G=(V,E)
- Problem description: What is the largest subset S of V such that no pair of vertices in S has an edge between them.
- Example:
  - Identifying location for a new franchise service such that no two locations are close enough to compete with each other.
  - Highest capacity code for given communication channel.

Hamiltonian Cycle

- Input description: A graph G=(V,E)
- Problem description: Find an ordering of the vertices such that adjacent vertices are connected by an edge and each vertex is visited exactly once.
- Example:

  Pattern recognition -- longest path in graph.

Graph Partition

- Input description: A weighted graph G=(V,E) and integers j,k
- Problem description: Partition vertices into two subsets such that each subset has size at most j, and the weight of edges connecting the two subsets is at most k.
- Example:
  VLSI layout

Vertex Coloring

- Input description: A graph G=(V,E)
- Problem description: Color vertices using the minimum number of colors such that for each edge (i,j) in E, vertices i and j have different colors
- Example:
  Register allocation for compilers.

Steiner Tree

- Input description: A graph G=(V,E) and a subset T of the vertices V
- Problem description: Find smallest tree connecting all the vertices of T
- Example:

  Network design and wiring layout.