Review session 2

Lecture 666
November 8, 2011
Dynamic Programming

- Find a “smart” recursion for the problem in which the number of distinct subproblems is small; polynomial in the original problem size.
- Eliminate recursion and find an iterative algorithm to compute the problems bottom up by storing the intermediate values in an appropriate data structure; need to find the right way or order the subproblem evaluation.
- Estimate the number of subproblems, the time to evaluate each subproblem and the space needed to store the value. Evaluate the total running time.
- Optimize the resulting algorithm further
Dynamic programming...

- Longest increasing subsequence.
- Computing the solution itself (not only its value).
- Maximum Weight Independent Set in Trees.
- Dynamic programs can be also solved as problems on DAGs.
- Edit distance: $O(nm)$ [but linear space!].
- Floyd-Warshall: $O(n^3)$.
- Knapsack: $O(nW)$ (pseudo-polynomial).
- TSP: $O(n^3 2^n)$ time and $O(n^2 2^n)$ space.
Greedy algorithms...

Greed has its place, but be careful not to be too greedy!

- Must prove correctness of greedy algorithms.
- Interval scheduling (so many variants that do not work). Proved correctness by showing that one can map the greedy solution to optimal.
- Interval Partitioning/Coloring. Proved the depth of instance was $\#$ colors used by greedy.
- Scheduling to Minimize Lateness.
Minimum spanning tree

- Algorithms can be interpreted as being greedy.
- **Prim**: $T$ maintained by algorithm will be a tree. Start with a node in $T$. In each iteration, pick edge with least attachment cost to $T$.
- **Reverse delete**: Delete edges keeping connectivity. Deleting edges from most expensive to cheapest.
- **Kruskal**: Add edges in increasing price. Add edge only if merges two trees in the current forest.
- **Borůvka’s**: Every vertex pick cheapest edge out of it. Collapse connected components of chosen edges. Repeat till have a single tree.
Why MST algorithms work?

**Definition**
An edge \( e = (u, v) \) is a **safe** edge if there is some partition of \( V \) into \( S \) and \( V \setminus S \) and \( e \) is the unique minimum cost edge crossing \( S \) (one end in \( S \) and the other in \( V \setminus S \)).

**Definition**
An edge \( e = (u, v) \) is an **unsafe** edge if there is some cycle \( C \) such that \( e \) is the unique maximum cost edge in \( C \).

**Proposition**
*If edge costs are distinct then every edge is either safe or unsafe.*

**Lemma**
*If \( e \) is a safe edge then every minimum spanning tree contains \( e \).*
Why MST algorithms work?

Even more

Lemma
Let $G$ be a connected graph with distinct edge costs, then the set of safe edges form a connected graph.

Corollary
Let $G$ be a connected graph with distinct edge costs, then set of safe edges form the unique MST of $G$.

Lemma
If $e$ is an unsafe edge then no MST of $G$ contains $e$. 
Data structures for MST

- Heap.
- Fibonacci heap.
- Union-find - path compression and union by rank. (Amazing running time - $O(\alpha(m, n))$ per operation,)
Randomized algorithms

- Basic concepts in discrete probability:
  Random variable, probability, expectation, linearity of expectation, independent events, conditional probability, indicator variables.
- Types of randomized algorithms: Las Vegas and Monte Carlo.
- Why randomization works - concentration of mass.
- Proved:

**Theorem**

Let $X_n$ be the number heads when flipping a coin independently $n$ times. Then

$$\operatorname{Pr} \left[ X_n \leq \frac{n}{4} \right] \leq 2 \cdot 0.68^{n/4} \text{ and } \operatorname{Pr} \left[ X_n \geq \frac{3n}{4} \right] \leq 2 \cdot 0.68^{n/4}$$
Randomized algorithms

- Proved **QuickSort** has $O(n \log n)$ expected running time.
- Proved **QuickSort** has $O(n \log n)$ running time with high probability.
- Proved **QuickSelect** has $O(n)$ expected running time.
- Hashing.
  - Why randomization is a must.
  - **2-universal** hash functions families.
  - Showed/proved a 2-universal hash family.
    - Guess two random numbers $\alpha$ and $\beta$. Hash function is $h(x) = (\alpha x + \beta) \mod p$. 
Network Flow

- Definitions.
- Edge flow $\Leftrightarrow$ path flow.
- Max-flow problem.
- Cuts and minimum-cut.
- $\text{flow} \leq \text{cut capacity}$.
- Max-flow Min-cut Theorem.
- Residual network.
- Augmenting paths.
- Ford-Fulkerson Algorithm.
- Proved correctness of Ford-Fulkerson Algorithm if capacities are integral.
Ford-Fulkerson running time is $O(mC)$.
Mentioned the strongly polynomial time algorithm by Edmonds-Karp.
Computing minimum cut from max-flow.
One can convert a flow to an acyclic flow.
A flow can be decomposed into paths from the source to the target + cycles.
Computing edge-disjoint paths using flow.
Computing vertex-disjoint paths using flow.
Menger’s theorem ($\# \text{ edge to cut} = \# \text{ edge disjoint paths}$).
Multiple sinks/sources.
Matching in bipartite graph.
Perfect matching.
Network Flow III

- Deciding if a specific team can win the Pennant using network flow.
- Project scheduling.
- Mentioned extensions to min-cost flow, and lower bounds on flow.
- Circulations.
- Survey design (using lower/upper bounds on flow).