Final Exam Review

CMSC 451, Summer 2009

Final Exam Topics

- Greedy Algorithms (Interval Scheduling, Shortest Path, Minimum Spanning Tree, Union-Find, Clustering, and Huffman Coding)
- Divide and Conquer (Merge Sort, Inversion Counting, Closest Pair of Points, Multiplication, and Matrix Multiplication)
- Dynamic Programming (Weighted Interval Scheduling, Knapsack Problem, and Sequence Alignment)
- Network Flow (Max Flow/Min Cut, Capacity Scaling, Bipartite Matching, Extensions, and Airline Scheduling)
- Intractability (Poly-time reductions, NP, NP-Complete)
- Approximation Algorithms (Load Balancing, k-Center Problem, Pricing Method, Integer and Linear Programming)
- Randomized Algorithms (Global Minimum Cut and Closest Pair of Points)
- Registrar’s Problem

LPs

- Given a flow network G(V,E) with the set of lower bounds l(v,w) and upper bounds u(v,w) for each edge, and demands d(v) for each vertex, give an LP (with no objective function) to create a valid circulation.

Google/Microsoft Interview Questions

- Given a number, describe an algorithm to find the next larger number that is prime.

Google/Microsoft Interview Questions

- You are given a list of numbers. When you reach the end of the list you will come back to the beginning of the list (a circular list). Write the most efficient algorithm to find the minimum # in this list. Find any given # in the list. The numbers in the list are always increasing but you don’t know where the circular list begins, ie: 38, 40, 55, 89, 6, 13, 20, 23, 36.

Divide and Conquer

- Provide a divide-and-conquer algorithm for determining the largest and second largest values in a given unordered set of numbers. Provide a recurrence equation expressing the time complexity of the algorithm, and derive it’s solution.
NP-Complete Reductions

- Dominating set: Given a graph G=(V,E) and integer 0<k≤|V|, is there a subset D of V such that |D|≤k and every vertex in V\D is joined to at least one member of D by an edge in E?
- Set cover: Given a universe U, subsets S_1, S_2, ... , S_m, and integer k, are there ≤k subsets S_i such that their union is U?
- Reduce dominating set to set cover

Google/Microsoft Interview Questions

- Assume you have an array that contains a number of strings (perhaps char * a[100]). Each string is a word from the dictionary. Your task, described in high-level terms, is to devise a way to determine and display all of the anagrams within the array (two words are anagrams if they contain the same characters; for example, tales and slate are anagrams.)

Network Flow

- Let M be an n x n matrix with each entry equal to either 0 or 1. Let mij denote the entry in row i and column j. A diagonal entry is one of the form mii for some i. Swapping rows i and j of the matrix M denotes the following action: we swap the values mik and mj for k=1,2,...,n. Swapping two columns is defined analogously.
- M is rearrangeable if it is possible to swap some of the pairs of rows and some of the pairs of columns (in any sequence) so that after all the swapping, all the diagonal entries of M are 1.
- Is M rearrangeable? Poly-time algorithm: yes/no

Interval Scheduling: Analysis

- Analysis: O(n log n) time
  - Sorting O(n log n)
  - Check compatibility in O(1) by remembering last finish time

```c
Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n.
set of jobs selected
A ← φ
for j = 1 to n {
  if (job j compatible with A)
    A ← A ∪ {j}
}
return A
```
Dijkstra’s Algorithm: Time Analysis

**Input:** edge-weighted graph \( G = (V, E, \delta) \), source \( s \), sink \( t \)

**Let** \( S \) be the set of explored nodes

**For each** \( u \in S \), store a distance \( d(u) \)

**Initialize** \( S = \{s\} \) and \( d(s) = 0 \)

**While** \( S \neq V \)

- **Select** a node \( u \) not in \( S \) with at least one edge from \( S \) such that \( d(u) = \left( \min_{v \in S} \{d(v) + \delta(u, v)\} \right) \) if minimized

- **Add** \( v \) to \( S \) and \( d(v) = d(u) + \delta(u, v) \)

**EndWhile**

**Post-condition.** Every \( v \in V \) is reached.
**Pre-condition.** Every \( v \in V \) is reachable.

**Frequencies:**

- \( f_1 = 0.32 \)
- \( f_2 = 0.20 \)
- \( f_3 = 0.18 \)
- \( f_4 = 0.05 \)

**Key idea.** Multiply 2-by-2 blocks with only 7 multiplications.

**Fast Matrix Multiplication**

**Key idea.** Multiply 2-by-2 blocks with only 7 multiplications.

\[
\begin{align*}
C_{00} &= A_1 \times (B_2 - B_3) \\
C_{01} &= (A_2 + A_3) \times B_3 \\
C_{10} &= A_2 \times (B_1 - B_3) \\
C_{11} &= A_3 \times B_1
\end{align*}
\]

- **7 multiplications:**
- **10 + 10 additions and subtractions.**
Weighted Interval Scheduling: Memoization

**Memoization**. Store results of each sub-problem in a cache; lookup as needed.

**Input**: n, s₁,...,sₙ, f₁,...,fₙ, v₁,...,vₙ

Sort jobs by finish times so that f₁ ≤ f₂ ≤ ... ≤ fₙ.

Compute p(1), p(2), ..., p(n)

for j = 1 to n

if (M[j] is empty)

M[j] = max(vⱼ + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))

return M[j]

Memoization. Store results of each sub-problem in a cache; lookup as needed.

Knapsack Problem: Bottom-Up

**Input**: n, W, w₁,...,wₙ, v₁,...,vₙ

for w = 0 to W

M[0, w] = 0

for i = 1 to n

for w = 1 to W

if (wᵢ > w)

M[i, w] = M[i-1, w]

else

M[i, w] = max {M[i-1, w], vᵢ + M[i-1, w-wᵢ]}

return M[n, W]
Load Balancing: LPT Rule

Longest processing time (LPT). Sort n jobs in descending order of processing time, and then run list scheduling algorithm.

```c
LPT-List-Scheduling(m, n, t_1, t_2, ..., t_n) {
  Sort jobs so that t_1 > t_2 > ... > t_n.
  for i = 1 to n {
    J(i) = \{ jobs assigned to machine i \}
    J(i) = J(i) \cup \{ job with maximum processing time \}
    update load of machine i
    return J(1), ..., J(m)
  }
}
```

Pricing Method

Set prices and find vertex cover simultaneously.

```c
Weighted-Vertex-Cover-Approx(G, w) {
  C = \{ s \}
  repeat k times {
    select a site s with maximum distance to closest site.
    add s to C
    for s \in C {
      Each c \in C except s
      dist(c, s) = distance to closest site
      φ(C) = \{ s \}
    }
  }
  return C
}
```

Weighted Vertex Cover: LP Relaxation

Formulate LP to find fractional values.

```c
(LP) \min \sum \sum \pi_i \pi_j
\text{s.t.} \sum \pi_i \pi_j = 1 \forall (i,j) \in E
\pi_i = 0 \forall i \notin V
```

Observation. Optimal value of (LP) is \( \frac{1}{2} \) optimal value of (IP).

Pf. LP has fewer constraints.

Example Knapsack

Knapsack ATAS. Round up all values: \( \frac{b_i}{\pi_i} \) \( \frac{b_j}{\pi_j} \)

```c
Knapsack-Approx(G, (v, b)) {
  S = solve Knapsack with values \( \frac{b_i}{\pi_i} \), \( \frac{b_j}{\pi_j} \)
  return S
}
```

Contraction Algorithm

Contraction algorithm. [Karger 1999]

* Pick an edge \( e = (u, v) \) uniformly at random.
* Contract edge \( e \).
  - replace \( u \) and \( v \) by single new super-node \( w \)
  - preserve edges, updating endpoints of \( u \) and \( v \) to \( w \)
  - keep parallel edges, but delete self-loops
  - repeat until graph has just two nodes \( v_k \) and \( v_{k+1} \)
* Return the cut (all nodes that were contracted to form \( v_k \)).
Recall: Closest Pair Algorithm

\[
\text{Closest-Pair}(p_1, \ldots, p_n) \{
\begin{align*}
\text{Compute separation line } L \text{ such that half the points are on one side and half on the other side.} \\
\delta_1 &= \text{Closest-Pair(left half)} \\
\delta_2 &= \text{Closest-Pair(right half)} \\
\delta &= \min(\delta_1, \delta_2) \\
\text{Delete all points further than } \delta \text{ from separation line } L \\
\text{Sort remaining points by } y\text{-coordinate.} \\
\text{Scan points in } y\text{-order and compare distance between each point and next 11 neighbors. If any of these distances is less than } \delta, \text{ update } \delta. \\
\text{return } \delta.
\end{align*}
\]

\[
O(n) \\
2T(n/2) + O(n) + O(n \log n) \\
O(n \log n)
\]

\[
O(n) \\
O(n \log n) \\
O(n \log n)
\]