CSE 101- Winter ‘16
Final review session
March 13th 2016
Final Exam Outline

- Short Answers (~10 parts, 20 points)
- “Mechanical Execution” (10 points)
- Design / Analysis: DP (10 points)
- Design / Analysis: Greed (10 points)
- NP-completeness Reduction (10 points)

- Total: 60 points, 3 hours.
NP-completeness

• Classes - P, NP, NP-hard, NP-complete

• **NP-Hard:** Problem $X$ is NP-hard if *every* problem in NP is *polynomially reducible* to $X$

• **NP-Complete:** Problem $X$ is NP-complete if:
  – $X$ belongs to NP, and
  – $X$ is NP-hard

• Also, $Y$ is **NP-complete** if $Y \in NP$ and some NP-complete problem $X'$ is *polynomially reducible* to $Y$
NP-Completeness Proofs: Reductions

**Definition**: X is *polynomial-time reducible* to Y, written $X \leq_p Y$, if there exists an algorithm for solving X that would be polynomial if we took no account of the time needed to solve arbitrary instances of Y.

**What can we conclude:**
- If Y is easy $\rightarrow$ X must be easy
- If X is easy $\rightarrow$ NOTHING
- If X is hard $\rightarrow$ Y must be hard
- If Y is hard $\rightarrow$ NOTHING
NP-Completeness Proofs: Reductions

• Show that zero weight cycle (ZWC) problem is NP-complete

• Solution:
• Step 1: show that Y is in NP - can we check if a guess is a solution in polynomial time

• Decision problem: we can succinctly prove the “YES” decision (easier part – don’t need to be too formal)
• By going through edges we can check if a cycle is zero-weight
NP-Completeness Proofs: Reductions

• Show that zero weight cycle (ZWC) problem is NP-complete

• Solution:
  • Step 2: identify an NP-complete problem X which you will reduce (transform) to ZWC
  • Rudrata cycle / Hamiltonian cycle
NP-Completeness Proofs: Reductions

• Show that zero weight cycle (ZWC) problem is NP-complete

• **Solution:**

• **Step 3**: polynomial reduction

• For each edge $e$ of $G$, we build a graph $G'$ as follows:
  - $w(e) = -(|V| - 1)$
  - for each edge $e'$ different from $e$, $w(e') = 1$

• Run ZWC on $G'$

• If one run of ZWC returns YES, return YES for RC, otherwise NO.
NP-Completeness Proofs: Reductions

• Show that zero weight cycle (ZWC) problem is NP-complete

• Solution:

• Step 4: Proof: (1) (=>) show that a yes-instance of ZWC implies a yes-instance of RC. If a run of ZWC returns YES, it means that we have a cycle of weight zero. It must include the only negative edge, e such as \( w(e) = -(|V| - 1) \). As all the other edges have edge 1, the cycle must also include \(|V| - 1\) other unit-weight edge. Therefore, the cycle has \(|V|\), it is a RC.

• (2) (<=) vice versa. If there is a RC, then when in some iteration RC will contain an edge of weight \(-(|V| - 1)\) which implies weight of RC is zero \(=>\) ZWC
Dynamic programming

• Formulate problem recursively using subproblems
• Solve subproblems in an efficient order (not recursively - tabulation)

• When to think of DP / how to come up with the subproblem:
  – **Optimal substructure** – optimal solution to the problem contains within it optimal solutions to subproblems
  – **Overlapping subproblems** – unlike DQ
Dynamic programming

• What can a subproblem look like?
  – Longest common subsequence – $L(i,j)$ – LCS in $A[1..i]$ and $B[1..j]$
  – Knapsack – $F(k,y)$ - maximum value possible using only the first $k$ item types, when the weight limit is $y$.

• DP Table/Matrix
  – Each entry is the solution of a subproblem
  – Index of table entry eg. $(i,j)$, are the parameters of the subproblem
  – Fill the table starting from smaller subproblems (base cases, …)
  – Recursive formulation used to fill subsequent entries–
    • 1 or constant number of subproblems - LCS
    • Linear ($\sim n$) subproblems - Knapsack
DP – worked example

• Maximum-weight independent set in a path (MWISP) problem

• **Subproblem definition:**
  \( \text{OPT}(i) \) be the weight of the MWISP for the subgraph \{v_1, v_2, ..., v_i\}
  
  **Goal:** \( \text{OPT}(n) \)

• **Recursive formulation**

  \( \text{OPT}(i) = \max (\text{OPT}(i-2) + v_i, \text{OPT}(i-1)) \)
DP – worked example

• Maximum-weight independent set in a path (MWISP) problem

• **Pseudocode:**

  Base case: $\text{OPT}(0) = 0; \text{OPT}(1) = v_1$

  For $i = 2$ to $n$
    – $\text{OPT}(i) = \max (\text{OPT}(i-2) + v_i, \text{OPT}(i-1))$

• **Time complexity** – $O(n)$
Max-flow / Min-cut

- Given a source vertex s and a sink vertex t, you want to route as much flow as possible from s to t along the edges of the graph.

- Think of each edge as a pipe with capacity equal to its weight. An edge from s to v with weight 4 can route 4 flow from s to v.

- Edges are directed

- Conservation of flow

- Non-negativity of flow
Ford-Fulkerson Algorithm

Max flow
• Initialize $f = 0$
• REPEAT:
  – Construct the residual graph $G_f$
  – Find a path $P$ from $S$ to $T$ in $G_f$
  – If there is no such path, HALT
  – $c_p = \text{minimum } c_f\text{-capacity edge on path } P$
  – Increase $f$ by $c_p$ units along path $P$

Min cut
• Define an $(L, R)$ cut as follows:
  – $L = \text{nodes reachable from } S \text{ in final residual graph } G_f$
  – $R = \text{rest of nodes } = V - L$
  – The flow across this cut = sum of capacities of edges from $L$ to $R$ = the capacity of the cut
Worked example

```
S
  ↓
  ↓
A  B
  |  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓T
```

```
S
  ↓
  ↓
A  B
  |  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓  ↓
  ↓T
```
Worked example
Worked example
Worked example

Final residual graph

Min cut

Max flow = 3
Worked example

Capacity of cut = 3
Network Flow Can Be Solved as a LP

- Optimization variables: $f_{SA}, f_{BA}, \text{etc.} \quad (\text{or } f_{DT}, f_{ET})$
- Capacity constraints: $0 \leq f_{SA} \leq 3, \quad 0 \leq f_{ET} \leq 2, \text{ etc.} \quad \text{(for each edge)}$
- Conservation constraints: $f_{SA} + f_{BA} = f_{AD}, \text{ etc.} \quad \text{(for each node)}$
- Non-negative flow constraints: $f_e \geq 0$ for all $e$ in $E$

**LP**: maximize $F = f_{SA} + f_{SB}$ subject to constraints

- Observation: “Max Flow reduces to LP”
- Observation: “Matching reduces to Max Flow” $\Rightarrow$ “Matching reduces to LP”
Linear Programming

• Minimize or maximize some objective function subject to some restrictions
• Feasible set: set of variable assignments that satisfy all constraints
• Optimal assignment will always be at one of the corners of the feasible set
Greedy algorithms

• A greedy algorithm always makes the choice that looks best *at the moment*
• Greedy algorithms do *not* always yield optimal solutions, but for some problems they do

• When do we use greedy algorithms?
  – When we need a heuristic for a hard problem (2-opt TSP)
  – When the problem itself is “greedy” (MST)

• Concepts we saw
  – Greedy parameter
  – Proof of optimality – exchange argument
Minimum spanning trees

- **Minimum Spanning Tree**, subgraph of unweighted graph $G$
  - it is a tree (i.e., it is acyclic) and covers all the vertices $V$
  - Therefore, contains $|V| - 1$ edges
  - the total cost associated with tree edges is the minimum among all possible spanning trees
  - not necessarily unique (unique when all edge weights are distinct)
- **Kruskal’s algorithm**. Start with $T = \emptyset$. Consider edges in ascending order of cost. Insert edge $e$ in $T$ unless doing so would create a cycle.
- **Prim’s algorithm**. Start with some root node $s$ and greedily grow a tree $T$ from $s$ outward. At each step, add the cheapest edge $e$ to $T$ that has exactly one endpoint in $T$. 
Minimum spanning trees

• **Cut property.** Let $S$ be any subset of nodes, and let $e$ be the min cost edge with exactly one endpoint in $S$. Then the MST contains $e$

• **Cycle property.** Let $C$ be any cycle, and let $f$ be the max cost edge belonging to $C$. Then the MST does not contain $f$. 
Example using cycle property

• Suppose you are given a connected graph $G = (V, E)$, with each edge having a unique weight (all edge weights are distinct). Edge $e$ is the heaviest edge in some cycle of $G$ but not necessarily the heaviest edge in $G$. Claim – Edge $e$ cannot be part of any MST of $G$

Follows from Cycle property…
Pre-midterm
Divide and Conquer

- Divide – the problem (instance) into one or more subproblems
- Conquer – each subproblem recursively
- Combine – separate solutions

**Master theorem:** $T(n) \leq a \cdot T(n/b) + O(n^d)$

1. $T(n) = O(n^d)$ if $a < b^d$
2. $T(n) = O(n^d \log n)$ if $a = b^d$
3. $T(n) = O(n\log_b a)$ if $a > b^d$

- Example: $T(n) = 9T(n/2) + O(n^2)$
Graph traversal/shortest paths

• Depth first search – aggressive $O(V+E)$
  – Types of edges – tree, forward, back, cross
  – DAG – topological ordering
  – Directed graph – scc, meta graph

• Breadth first search – layer by layer $O(V+E)$
  – Shortest paths in unweighted graphs

• Shortest paths in weighted graphs
  – Dijkstra’s $O((V+E) \log |V|)$
  – Bellman-Ford $O(V*E)$
Dijkstra’s Algorithm: Running time

\begin{verbatim}
procedure dijkstra(G, l, s)
    for all u ∈ V:
        dist(u) = ∞
        prev(u) = nil
    dist(s) = 0

    H = makequeue(V) (using dist-values as keys)

    While H is not empty:
        u = deletemin(H)
        for all edges (u, v) ∈ E:
            if(dist(v)) > dist(u) + l(u, v) :
                dist(v) = dist(u) + l(u, v)
                prev(v) = u
                decreasekey(H, v)

Priority queue operations = |V|+|E| insert/decreasekey + |V| deletemin
\end{verbatim}
Bellman-Ford Algorithm: Pseudo-Code

procedure update((u, v) ∈ E)
  \( \text{dist}(v) = \min\{\text{dist}(v), \text{dist}(u) + l(u,v)\} \)

procedure shortest-paths(G, l, s)

  for all u ∈ V:
    \( \text{dist}(u) = \infty \)
    \( \text{prev}(u) = \text{nil} \)

  \( \text{dist}(s) = 0 \)

  Repeat \(|V| - 1\) times:
    for all e ∈ E:
      update(e)  

(Ref: Algorithms, Fig 4.8)
Big-O

• \( f(n) = O(g(n)) \) if there exist positive constants \( c \) and \( n_0 \) such that \( 0 \leq f(n) \leq c(g(n)) \) for all \( n \geq n_0 \).

• Alternate definition:
  \(-\ f(n) = O(g(n)) \Rightarrow 0 \leq \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty\)

• Claim: \(5n^2 + n = O(n^2)\)

• Proof:
  \(-\) Consider \( \lim_{n \to \infty} \frac{5n^2 + n}{n^2} \)
  \(= \lim_{n \to \infty} 5 + \frac{1}{n} \)
  \(= 5 < \infty\)

  \(-\) Hence proved!
L'Hôpital's Rule

- Claim: \( \log n = O(n) \)
- Proof:
  - Consider \( \lim_{n \to \infty} \frac{\log n}{n} \)
    \[
    = \lim_{n \to \infty} \frac{d(\log n)/d\ n}{d(n)/d\ n} \quad \text{(L'Hôpital's Rule)}
    \]
    \[
    = \lim_{n \to \infty} \frac{1/n}{1} = 0 < \infty
    \]
  - Hence proved!
Two pieces of paper ("cheat sheets") allowed for final exam.