University of Washington
Department of Computer Science and Engineering
CSEP 521, Winter 2013

March 21, 2013

Exam Solution, Monday, March 18, 2013

NAME: ______________________

Instructions:

- Closed book, closed notes, no calculators
- Time limit: Two hours
- Answer the problems on the exam paper.
- If you need extra space use the back of a page
- Problems are not of equal difficulty, if you get stuck on a problem, move on.

1	/15
2	/15
3	/10
4	/15
5	/15
6	/15
7	/15
8	/10
Total	/110

Problem 1 (15 points) Short Answer:

a) How does the run time of finding a minimum cut compare with the run time for finding a maximum flow?

Solution:

To find a minimum cut, we first find the maximum flow, and then find the set of reachable vertices in the residual graph. This last step, can be done by a basic search algorithm in $O(n + m)$ time. Thus, if network flow can be solved in time $f(n, m)$, then the min-cut can be found in time $f(n, m) + O(n + m)$ time. Since the time to compute a flow is much larger than the other step, it is fair to say they can be computed in roughly the same time.

b) If you know problem X is NP-complete, and you want to show that problem Y is NP-complete, do you reduce X to Y, or reduce Y to X. Justify your answer.

Solution:

You reduce from X to Y. To show that Y is NP-complete, you want to show that it is "harder" than another NP-complete problem.

c) Yes, no, or maybe: There is a polynomial time algorithm for determining if an undirected graph has a hamiltonian circuit? Justify your answer.

Solution:

Maybe. Hamiltonian Circuit is NP-Complete. However, since we don't know if NP is different from P, the answer is maybe. If there were an polynomial time algorithm for Hamiltonian Circuit, then P=NP.

Problem 2 (15 points) Recurrences:

Give solutions to the following recurrences. Justify your answers.

a)

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + n^3 & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

Solution:

$O(n^3)$. The recurrence has a decreasing amount of work per level. The amount of work at the top level is n^3.

b)

$$T(n) = \begin{cases} 4T(\frac{n}{2}) + n^2 & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

Solution:

$O(n^2 \log n)$. The work remains the same on each level. The work per level is n^2 and there are $\log_4 n = \frac{1}{2}\log_2 n$ levels.

c)

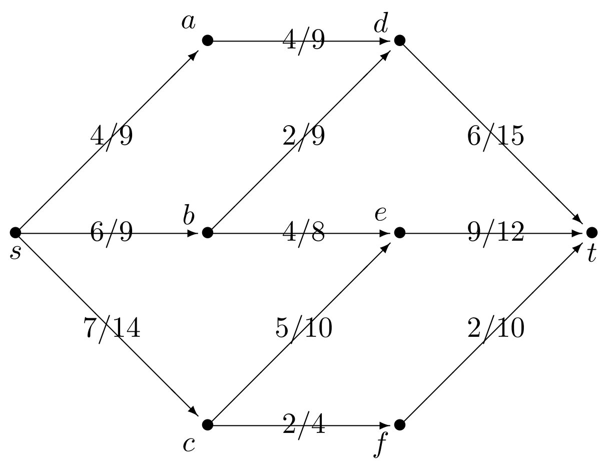
$$T(n) = \begin{cases} 3T(\frac{n}{2}) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

Solution:

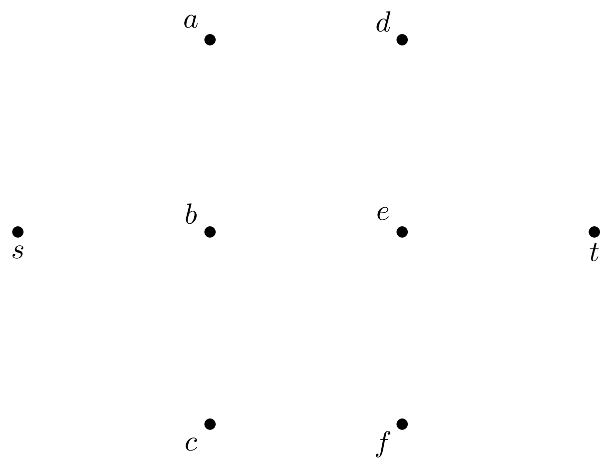
$O(n^{\log_2 3})$. The recurrence has an increasing amount of work per level, so the work is dominated by the last level. The work on the bottom level is $3^{\log_2 n} = 2^{\log_2 3 \log_2 n} = n^{\log_2 3}$.

Problem 3 (10 points) Residual Graph:

Consider the following flow graph G, with an assigned flow f. The pair x/y indicates that the edge is carrying a flow x and has capacity y.



a) Draw the residual graph G_f for the flow f.



Solution:

We apply the standard contruction for the residual graph. If there is a flow of f and capaicty c on the edge (u, v), we create an edge of capacity $c - f$ from u to v, and an edge of capacity f from v to u.

b) Show how the Ford-Fulkerson algorithm can find a maximum flow. List the paths that you use to augment the flow.

Solution:

There are muliple ways of doing this. One set of augmenting paths:

2 :	s, c, f, t
3 :	s, c, e, t
2 :	s, c, e, b, d, t
3 :	s, b, d, t
4 :	s, a, d, t

Problem 4 (15 points) How can you?:
Give short answers to the following. You do not need to discuss runtime. How can you:

a) Find a maximum weight spanning tree using an algorithm which computes a minimum weight spanning tree.

Solution:

We multiply the weight of every edge by -1, and then compute a minimum spanning tree.

b) Determine if there is a path from a vertex s to every other vertex in a directed graph using a network flow algorithm.

Solution:

We set this up as a flow problem by adding a sink t. We assign a capacity of ∞ to each of the original edges in the graph and then add an edge (v, t) of capacity 1 for each vertex v in the graph. If there is a flow of size n from s to t, then every vertex in the graph is reachable.

c) Find an optimal travelling salesman tour in a graph which may have negative length edges with an algorithm for the travelling salesman problem that requires all edges have positive length.

Solution:

Add a constant C to each edge to make it positive. Since the TSP tour as length n, this changes the cost of the tour by Cn. If $-m$ is the smallest edge in the graph, we can use $C = m + 1$.

Problem 5 (15 points) Product-sum:

Given a list of n integers, $v_1, \ldots, v_n$, the *product-sum* is the largest sum that can be formed by multiplying adjacent elements in the list. Each element can be matched with at most one of its neighbors. For example, given the list $1, 2, 3, 1$ the product sum is $8 = 1 + (2 \times 3) + 1$, and given the list $2, 2, 1, 3, 2, 1, 2, 2, 1, 2$ the product sum is $19 = (2 \times 2) + 1 + (3 \times 2) + 1 + (2 \times 2) + 1 + 2$.

a) Compute the product-sum of $1, 4, 3, 2, 3, 4, 2$.

$29 = 1 + (4 \times 3) + 2 + (3 \times 4) + 2$.

b) Give the optimization formula for computing the product-sum of the first j elements.

$$OPT[j] = \begin{cases} \max\{OPT[j-1] + v_j, OPT[j-2] + v_j * v_{j-1}\} & \text{if } j \geq 2 \\ v_1 & \text{if } j = 1 \\ 0 & \text{if } j = 0 \end{cases}$$

c) Give a dynamic program for computing the value of the product sum of a list of integers. (You do not need to determine which pairs are multiplies together - just the product.)

```
*Prod-Sum*(**int**[ ]$v, n$)
  **if** $(n == 0)$ **return** 0;
  **int** [ ] $OPT$ = **new int** $[n+1]$;
  $OPT[0] = 0$;
  $OPT[1] = v[1]$;
  **for int** $j = 2$ **to** $n$
    $OPT[j] = \mathbf{max}(OPT[j-1] + v[j], OPT[j-2] + v[j] * v[j-1])$;
  **return** $OPT[n]$;
```

Problem 6 (15 points) Electoral college:

The problem is to determine the set of states with the smallest total population that can provide the votes to win the electoral college. Formally, the problem is:

Let p_i be the population of state i, and v_i the number of electoral votes for state i. All electoral votes of a state go to a single candidate, so the winning candidate is the one who receives at least V electoral votes, where $V = \lfloor(\sum_i v_i)/2\rfloor + 1$. Our goal is to find a set of states S that minimizes the value of $\sum_{i \in S} p_i$ subject to the constraint that $\sum_{i \in S} v_i \geq V$.

a) The dynamic programming solution for this problem involves computing a function OPT where $OPT[i, v]$ gives the minimum populations of a set of states from $1, 2, \ldots, i$ such that their votes sum to exactly v. Give a recursive definition of OPT and an explanation as to why it is correct.

   $OPT[i, v] = \min\{OPT[i-1, v], OPT[i-1, v-v_i] + p_i\}$. The first term corresponds to state i not included in the vote total, and the second term corresponds to the state i included in the vote total.

b) What are the base cases for your function OPT.

   $OPT[0, 0] = 0$ and $OPT[0, v] = \infty$ for $v \geq 1$.

   Grading notes. Part A, 9 Points. Part B, 6 Points. The most common error was to give a base case of 0.

Problem 7 (15 points) Currency Conversion:

A group of traders are leaving Switzerland, and need to convert their Francs (the local currency) into various international currencies. There are n traders and m currencies. Trader i has T_i Francs to convert. The bank has B_j Francs worth of currency j. Trader i is willing to trade as much as C_{ij} of his Francs for currency j. (For example, a trader with 1000 Francs might be willing to convert up to 700 of his Francs for USD, up to 500 of his Francs for Japaneses Yen, and up to 500 of his Francs for Euros).

Assuming that all traders give their requests to the bank at the same time, describe an algorithm that the bank can use to satisfy the requests (if it can).

Solution:

This is set up as a flow network, with Francs flowing from the source s to the sink t. A row of vertices $t_1, \ldots, t_n$ represents the traders, and a row of vertices $b_1, \ldots, b_m$ represents the currency held by the bank. The edge (s, t_i) has capacity T_i which gives the amount trader i wants to change. The edge (t_i, b_j) has capacity C_{ij} giving the maximum number of Francs i wants to trade into currency j. Finally, the edge (b_j, t) with capacity B_j gives the limit on the amount of currency j that is available. If there is a flow f in the network with $|f| = \sum_i T_i$ then all traders are able to convert their currencies. (An alternate solution which reverses the source and sink, and has currency flow to from the banks to traders is also valid.)

Problem 8 (10 points) 4SAT:

Show that the 4SAT problem is NP-complete by giving a reduction from 3SAT. 4SAT is the satisfiability problem with exactly four literals per clause. You do not need to show that 4SAT is in NP.

Solution:

The simplest reduction from 3SAT to 4SAT is just to duplicate a literal in each clause, e.g.,

$$(x \vee y \vee \overline{z}) \quad \Longrightarrow \quad (x \vee y \vee \overline{z} \vee \overline{z}).$$

An alternate approach is to add a variable to each clause that is set to false:

$$(x \vee y \vee \overline{z}) \quad \Longrightarrow \quad (x \vee y \vee \overline{z} \vee a),$$

where the additional clause

$$(\overline{a} \vee \overline{a} \vee \overline{a} \vee \overline{a})$$

forces a to false. (Repeated literals can be avoided by using a group of eight additional clauses to false a to false).

Grading notes: Just adding the constant **F** to a clause was -2, since it is technically not a literal. Points were also lost for confusing $\vee$ and $\wedge$.