CS 170, Fall 1997
Second Midterm
Professor Papadimitriou

Problem #1

(10 Points) Remember the *change-maker problem*: We are given k integers $d_1, ..., d_k > 0$ (the *coin denominations*) and an integer n. We want to write n as the sum of denominations, with repetitions, with as few coins as possible. For example, for denominations 1, 5, 10, and 25, and $n = 83$, then the optimum solution is 25 + 25 + 25 + 5 + 1 + 1 + 1, with cost 7.

Give a dynamic programming algorithm for the change-maker problem. Suppose that c(i) is the minimum number of coins adding up to $i >= 0$.

A. Dynamic programming recurrence:

Basis (value at zero):

B. Justification of correctness:

C. Running time of the corresponding algorithm, as a function of n and k (you don't have to describe the algorithm). *Justification of the running time.*

D. Is this a polynomial-time algorithm? Why or why not?

Problem #2

(10 Points) (a) Write the change-maker problem (see the previous problem) as an *integer linear programming* problem:

choose your variables:

minimize this linear function:

subject to these constraints:

plus, all variables should be integers.

(b) Can we solve this problem by solving it as a linear programming problem with the certainty that the answer will come out integer, as in the bipartite matching problem? Either justify why this is the case, or give a counterexample in which the optimum of the linear program is not integer.

Problem #3

(10 points) STINGY SAT is the following problem: Given a set of clauses, and an integer *K*, is there a truth assignment that satisfies all clauses and has at most *K* TRUEs in it?

Prove that this problem is NP-complete.

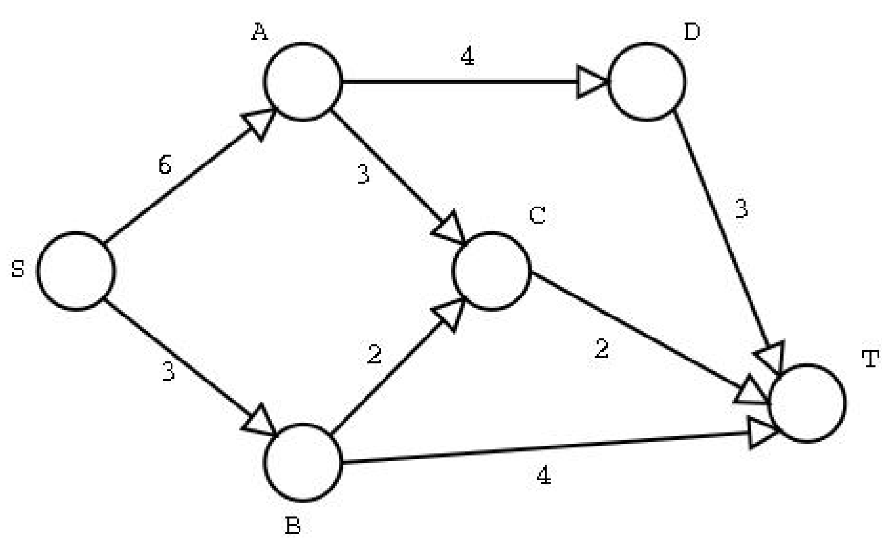
STINGY SAT is in NP:

Reduction form

Justification of the reduction.

Problem #4

(10 points) Consider the instance of the max-flow problem shown below.



A. What is the value of the maximum flow from S to T?

B. Indicate a minimum cut between S and T by drawing a line through the diagram above.

C. Suppose that you run the Ford-Fulkerson algorithm on this network. In the first iteration you find a path by depth-first search from S (breaking ties, as always, lexicographically), and augment along it. *Show the resulting residual network* (the network on which you will find an augmenting path next).

Problem #5

(Total of 33 points)

True or false? No explanation required, except for partial credit. Each question is worth 1.5 points, for a perfect score of 33. One point will be subtracted for wrong answers after the first two, so answer only if you are reasonably certain.

- The solution of $T(n) = 2T(n/2) + n$, $T(1) = 0$ is Theta($n \log n$).
- The solution of $T(n) = 7T(n/2) + n^3$, $T(1) = 0$ is Theta(n^3).
- The solution of $T(n) = 4T(n/2) + n^2$, $T(1) = 0$ is Theta(n^2).
- In Huffman coding, if all frequencies of symbols are distinct, then the most frequent symbol gets the shortest code.
- In Huffman coding, if all frequencies of symbols are distinct, then the second least frequent symbol gets the longest code.
- If all frequencies of symbols are distinct, the optimum Huffman code is unique.
- If the dynamic programming recurrence is

  $$C[i,j] = \min_{i<k<j} [C[i,k] + C[k,j] + k], \quad i <= j = 1, ..., n$$

  then the algorithm will take $O(n^2)$ time.
- If the dynamic programming recurrence is

  $$C[i,j] = \min \{2C[i-1, j], 2C[i, j-1], 4C[i-1, j-1]\}, \quad i <= j = 1, ..., n$$

  then the algorithm will take $O(n^2)$ time.
- FFT stands for "fast Fourier transform."

- If w is the nth root of unity, then w^2 is the $2n$th root of unity.
- If w is the nth root of unity, then w^2 is the $(n/2)$nd root of unity.
- To multiply two polynomials of degrees 16 and 13, respectively, we should use the FFT with 32nd roots of unity.
- If all capcities in a max-flow problem are integers, then there is an integer optimum.
- If all capcities in a max-flow problem are integers, then a flow with fractional values cannot be optimum.
- There is a known polynomial-time algorithm for linear programming.
- There is a known polynomial-time algorithm for integer linear programming.
- The simplex algorithm solves linear programming in polynomial time.
- If P = NP then all NP-complete problems are solvable in polynomial time.
- If one NP-complete problem is solvable in polynomial time then P = NP.
- There are decision problems that are not in NP.
- *one point extra credit* Write the *complete expansion* of the polynomial $(x - a) * (x - b) * (x - c) \ldots (x - z)$
- If a directed graph has a Hamilton cycle then it is strongly connected.
- If a directly graph is strongly connected then it has a Hamilton cycle.
