

CS3500 HW7 Solutions

1 (10 points each)

(a) We are trying to make change for n cents. The following greedy algorithm returns a multiset A on the set Q, D, N, P such that

$$25\mu_A(Q) + 10\mu_A(D) + 5\mu_A(N) + \mu_A(P) = n$$

where $\mu_A(Q)$ is the multiplicity of Q in A and similarly for $\mu_A(D)$, $\mu_A(N)$ and $\mu_A(P)$.

GREEDY-CHANGE(n)

1. $A \leftarrow \emptyset$
2. **while** $n \neq 0$
3. **do if** $n \geq 25$
4. **then** $A \leftarrow A + \{Q\}$
5. $n \leftarrow n - 25$
6. **elseif** $n \geq 10$
7. **then** $A \leftarrow A + \{D\}$
8. $n \leftarrow n - 10$
9. **elseif** $n \geq 5$
10. **then** $A \leftarrow A + \{N\}$
11. $n \leftarrow n - 5$
12. **else** $A \leftarrow A + \{P\}$
13. $n \leftarrow n - 1$
14. **return** A

Correctness: We will show how by induction on n that this algorithm yields an optimal solution - that is one which minimizes the number of coins, $|A|$. As the base case, for $n = 0$ we get $A = \emptyset$ which is clearly the optimal solution. Now consider making change for $n > 0$. If $n \geq 25$ then $A = \{Q\} + A'$ where A' is the change that would be returned by GREEDY-CHANGE($n - 25$), and therefore, by the induction hypothesis, A' is an optimal solution for $n - 25$ and $|A| = |A'| + 1$. Now let B be an optimal solution for n , and suppose that B does not contain any quarters ($\mu_B(Q) = 0$). If there exists some sub-multiset B' of B such that

$$10\mu_{B'}(D) + 5\mu_{B'}(N) + \mu_{B'}(P) = 25$$

then we could replace B' with $\{Q\}$ to yield $B'' = B - B' + \{Q\}$ with $|B''| < |B|$ which contradicts the optimality of B so no such B' can exist. The only other possibility (since $n \geq 25$) is that $\mu_B(D) \geq 3$, but in this case, we can replace three dimes with a quarter and a nickel to again produce a smaller solution which again is a contradiction. Thus, B must contain a quarter, so $B = \{Q\} + B'$ where B' is a solution for $n - 25$. Since A' is an optimal solution for $n - 25$, we have $|A'| \leq |B'|$, and therefore $|A| \leq |B|$, so A is an optimal solution for n . The cases for $10 \leq n < 25$, $5 \leq n < 10$, and $n < 5$ are all handled in a similar fashion.

(b) There are many examples, but consider having just quarters, dimes and pennies. Making change of 30 cents via a greedy strategy uses 6 coins (1 quarter and 5 pennies), yet the optimal solution uses 3 coins (3 dimes).

(c) Suppose we have k different coins with denominations d_1, \dots, d_k . Let $f(n)$ denote the optimal number of coins needed to make change for n cents. Then we can compute $f(n)$ for any $n > 0$ by

$$f(n) = \min_{1 \leq i \leq k} (f(n - d_i)) + 1$$

with the initial conditions $f(n) = \infty$ for $n < 0$, and $f(0) = 0$.

Solving this recursion iteratively, we can compute $f(n)$ in $O(kn)$ steps. To actually generate the set of coins, we can simply record the value of each j at each step, then trace back through the array of recorded values.

2 (10 points each)

(a) Let T be the $|V| \times |V|$ matrix representing the transitive closure, such that $T[i, j]$ is 1 if there is a path from i to j , and 0 if not. Initialize T (when there are no edges in G) as follows:

$$T[i, j] = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

T can be updated as follows when an edge (u, v) is added to G :

TRANSITIVE-CLOSURE-UPDATE(u, v)

```

for  $i \leftarrow 1$  to  $|V|$ 
  do for  $j \leftarrow 1$  to  $|V|$ 
    do if  $T[i, u] = 1$  and  $T[v, j] = 1$ 
      then  $T[i, j] \leftarrow 1$ 

```

- This says that the effect of adding edge (u, v) is to create a path (via the new edge) from every vertex that could already reach u to every vertex that could already be reached from v .
- Note that the procedure sets $T[u, v] \leftarrow 1$, because of the initial values $T[u, u] = T[v, v] = 1$.
- This takes $\Theta(V^2)$ time because of the two nested loops.

(b) Consider inserting the edge $v_n \rightarrow v_1$ into the straight-line graph $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$, where $n = |V|$. Before this edge is inserted, only $n(n+1)/2$ entries in T are 1 (the entries on and above the main diagonal). After the edge is inserted, the graph is a cycle in which every vertex can reach every other vertex, so all n^2 entries in T are 1. Hence $n^2 - (n(n+1)/2) = \Theta(n^2) = \Theta(V^2)$ entries must be changed in T , so any algorithm to update the transitive closure must take $\Omega(V^2)$ time on this graph.

(c) This algorithm in part (a) would take $\Theta(V^4)$ time to insert all possible $\Theta(V^2)$ edges, so we need a more efficient algorithm in order for any sequence of insertions to take only $\Theta(V^3)$ time.

To improve the algorithm, notice that the loop over j is pointless when $T[i, v] = 1$. That is, if there is already a path $i \rightsquigarrow v$, then adding the edge $u \rightarrow v$ would not make any new vertices reachable from i . The loop to set $T[i, j]$ to 1 for j such that there is a path $v \rightsquigarrow j$ is just setting entries that are already 1. Eliminate this redundant processing as follows:

TRANSITIVE-CLOSURE-UPDATE(u, v)

1. **for** $i \leftarrow 1$ **to** $|V|$
2. **do if** $T[i, u] = 1$ **and** $T[i, v] = 0$
3. **then for** $j \leftarrow 1$ **to** $|V|$
4. **do if** $T[v, j] = 1$
5. **then** $T[i, j] \leftarrow 1$

We show that this takes $O(V^3)$ time to update the transitive closure for any sequence of n insertions:

- There can't be more than $|V|^2$ edges in G , so $n \leq |V|^2$.
- Summed over n insertions, the time to execute lines 1 and 2 is $O(nV) = O(V^3)$.

- Lines 3-5, which take $\Theta(V)$ time, are executed only $O(V^2)$ times for n insertions. To see this, notice that lines 3-5 are executed only when $T[i, v] = 0$, and in that case line 5 sets $T[i, v] \leftarrow 1$, so the number of 0 entries in T is reduced by at least 1 each time lines 3-5 run. Since there are only $|V|^2$ entries in T , lines 3-5 can run at most $|V|^2$ times.
- Hence the total running time over n insertions is $O(V^3)$.

3 (10 points each) a) The reduction here is simple. We just modify the graph slightly, by adding two vertices s and t , and we add edges between every $v \in V$ to s and to t . s remains disconnected from t . We will call this graph G' . Now consider the instance $\langle G', s, t, |V| + 1 \rangle = y$. Assume that $x = \langle G \rangle \in \text{HAMILITON-PATH}$. That mean there is a path of length $|n - 1|$ from two vertices a and b , thus there is a path of length at least $n + 1$ in G' , i.e. from $s \rightarrow a \rightsquigarrow b \rightarrow t$. So $x \in \text{LPATH}$. Conversely assume that $y \in \text{LPATH}$. That means there exists a path of length $n + 1$ from s to t in G' . A path can have no repeated vertices, thus there exists a path of length $n - 1$ in G and thus $x \in \text{HAMILITON-PATH}$. Futhermore, HAMILITON-PATH is clearly in NP , given a path use can check that that path is Ham. in polynomial time. Thus HAM-PATH is NP -Complete

b) Notice the set of undirected graphs is the set of instances both for CLIQUE and INDEPENDENT-SET. Given a graph we want to determine if it has a CLIQUE of size k . To reduce to INDEPENDENT-SET do the following. If $(u, v) \in V^2$ is an edge remove it and if is not an edge add it. Call this new graph G' . Clearly G' has an independent set iff and G has a CLIQUE. Furthermore, given a set of k vertices we can determine in poly time that no vertex in that set has an edge to another vertex in that set. Thus INDEPENDENT-SET is NP -Complete.

4 (20 points)

Lets call the problem given in the question Π . In order to prove that Π is NP-Hard all we need to do is show that there is a polynomial time reduction from 3-SAT to Π (as opposed to NP-Complete where you must also show that Π is in NP as well, we are just asked for NP-Hard here). In order to give a reduction from SAT to Π we must on receiving an instance of the 3-SAT problem, i.e. a Boolean formula F in 3-conjunctive normal form, be able to define a matrix A and a column vector b such that each entry in

these is an integer. Furthermore the system $Ax \geq b$ must have an integer solution iff F is satisfiable.

The Reduction Function: In what follows we assume that a_{ij} is the element in row i column j of A , and b_i is the i^{th} element of b . Notice that asking if there exists an x such that $Ax \geq b$ where A is a $n \times m$ matrix is equivalent to asking does there exist an assignment to the set of variables x_0, x_1, \dots, x_{m-1} such that the following system of linear inequalities is satisfied:

$$\begin{aligned} a_{00}x_1 + a_{01}x_2 + \dots + a_{0m}x_m &\geq b_1 \\ a_{10}x_1 + a_{11}x_2 + \dots + a_{1m}x_m &\geq b_2 \\ &\vdots \\ a_{n0}x_1 + a_{n1}x_2 + \dots + a_{nm}x_m &\geq b_n \end{aligned}$$

Given a boolean formula F having n literals and k clauses, all we need to do is define a set of linear inequalities such that each a_{ij} and b_i is an integer and such that F has a solution iff there are integer assignments to the x_i s such that the system of inequalities is satisfied.

So assume F has k clauses and n literals. We will have $2n$ variables, two for each literal, i.e. for the i^{th} literal define two variables x_i and \bar{x}_i (intuitively one for the literal and the other is for its negation). For each clause $(\bar{w}_h, w_i, \bar{w}_j)$ we give a constraint

$$\bar{x}_h + x_i + \bar{x}_j \geq 1. \tag{1}$$

Notice that if w_j is negated in a clause then in the corresponding inequality we have an \bar{x}_j . Also for each literal w_i we give four constraints

$$x_i \geq 0, \tag{2}$$

$$-x_i \geq -1, \tag{3}$$

$$\bar{x}_i \geq 0, \tag{4}$$

and

$$-\bar{x}_i \geq -1. \tag{5}$$

Notice that this forces every x_i and \bar{x}_i to be 1 or 0. Finally we set for each literal we have two additional constraints

$$x_i + \bar{x}_i \leq 1 \tag{6}$$

and

$$-x_i - \overline{x_i} \geq -1. \quad (7)$$

Notice that constraints 6 and 7 imply $x_i + \overline{x_i} = 1$, which along with constraint 2-5 mean that one of x_i and $\overline{x_i}$ is equal to 1 and the other is 0.

So now we have defined the reduction, and we must prove that it works. Assume that F is satisfiable. Is there an assignment to the variables $x_1 \dots x_n$ such that the inequality constraints defined above are also satisfied? Yes, there is. Let w be a satisfying assignment to F . If a variable w_i is TRUE we let $x_i = 1$ and thus $\overline{x_i} = 0$ and if it is FALSE the reverse holds. Notice that this satisfies constraints 2-7. For constraints 1 and 2, notice that if we have w_i in the a clause and $w_i = FALSE$ then in the corresponding constraint we have $x_i = 0$. Alternatively if we have $\overline{w_i}$ in a clause and $\overline{w_i} = TRUE$ then in the corresponding constraint we have $\overline{x_i} = 0$. Thus if a clause is false its corresponding constraint is equal to 0. Given our assumption that every clause is satisfiable every constraint is greater than or equal to 1.

The argument for the reverse direction is very simple. In any solution to this integer programming instance, all variables must be set to either 0 or 1. If $x_i = 1$, then set literal $w_i = TRUE$. If $x_i = 0$, then set literal to FALSE. No Boolean variable and its complement can both be true, so it is a legal assignment, which must also satisfy all the clauses.

This completes the proof:

Several of you had the following idea: Given a formula in 3-CNF with m clauses and n variables, (1) let the vector $b = (-1, \dots, -1)$, (2) for the matrix A , have 1 row for each clause C_i , the ij th entry of A is 1 if C_i contains x_j , -1 if C_i contains $\overline{x_j}$, and 0 otherwise. Well, some of you did this because I told you too. However this solution is not correct because not correct yet, as just assigning $x = (0, \dots, 0)$ satisfies the system of linear equations whether or not the given formula is satisfiable.