Prof. Sergio A. Alvarez
21 Campanella Way, room 569
Computer Science Department
Boston College
Chestnut Hill, MA 02467 USA

http://www.cs.bc.edu/∼alvarez/
alvarez@cs.bc.edu
voice: (617) 552-4333
fax: (617) 552-6790

CS383, Algorithms
Spring 2009
HW7 Selected Solutions

1. Consider the computational task stated in Algorithm 1. Notice that the only constraint on the coefficients d_i is non-negativity. In particular, there is no predetermined upper bound either on the total number of d_i nor on their individual magnitudes. As an example, both $d_0 = 35,\ d_1 = 4$ and $d_0 = 25,\ d_1 = 5$ are valid representations of the same value $n = \sum_{i=0,1} d_i 10^i$.

   Algorithm 1: Powers of Ten
   Input: Non-negative integer, n.
   Output: Non-negative integers $d_0...d_k$ (number k not specified in advance) such that $n = \sum_{i=0...k} d_i 10^i$, and $\sum_{i=0...k} d_i \leq \sum_{i=0...k'} c_i$ whenever $n = \sum_{i=0...k'} c_i 10^i$.

   (a) Cast the computational task in Algorithm 1 as an optimization task of the sort discussed in class in the context of greedy algorithms. Specify each of the following elements in detail: set of candidates C, notion of solution, notion of feasible set, and objective function. State whether the task involves maximization or minimization. Your answers should be specific to the particular computational task in Algorithm 1.

   Solution

   - The (multi)set C of candidate elements from which any solution is to be constructed consists of unlimited numbers of tokens with the values 10^i for each integer $i \geq 0$.
   - Given the input (target value) n, a subcollection S of C is a solution if and only if the values of the tokens contained in S add up to n. If S contains d_i copies of token 10^i for each $i = 0...k$, then this is equivalent to saying that n is equal to the sum $\sum_{i=0...k} d_i 10^i$.
   - Given the input (target value) n, a subcollection S of C is a feasible set if and only if the values of the tokens contained in S add up to a value that is no greater than n. If S contains d_i copies of token 10^i for each $i = 0...k$, then this is equivalent to saying that n is greater than or equal to the sum $\sum_{i=0...k} d_i 10^i$.
   - The objective function is a quantity to be optimized that is expressed in terms of the elements of a proposed solution set S. In the present case, this quantity is the sum of the d_i, assuming that the proposed solution set S consists of d_i copies of 10^i, for $i = 0...k$. The statement makes it clear that this quantity should be *minimized* by an optimal solution $d_0...d_k$.

   (b) Using your anwers from 1a, provide detailed pseudocode for a greedy computational procedure for Algorithm 1 that solves the stated computational task.

Solution

See Algorithm 2.

Algorithm 2: Greedy decimal expansion
Input: Non-negative integer, n.
Output: Non-negative integers $d_0...d_k$ (number k not specified in advance) such that $n = \sum_{i=0...k} d_i 10^i$, and $\sum_{i=0...k} d_i \leq \sum_{i=0...k'} c_i$ whenever $n = \sum_{i=0...k'} c_i 10^i$.
GREEDYTENS(n)

```
(1)  if n > 0
(2)      k = ⌊log10 n⌋
(3)  else
(4)      k = 0
(5)  foreach i = k...0
(6)      d_i = ⌊n/10^i⌋
(7)      n = n − d_i 10^i
(8)  return d_0...d_k
```

(c) Show the details of the computation performed by your algorithm from 1b on the input $n = 395$. What is the optimal value of the objective function for this particular input instance? Explain.

Solution

Algorithm 2 sequentially finds the values $k = 2$, $d_2 = 3$, $d_1 = 9$, $d_0 = 5$. The objective function for this solution has the value $3 + 9 + 5 = 17$, which is in fact the optimal (smallest possible) value. See the next subtask.

(d) Give a careful argument that shows that your algorithm in 1b returns a solution that is optimal in the sense of optimizing the objective function from 1a.

Solution

We can prove by induction in k that if Algorithm 2 outputs $d_0...d_k$ for a given target integer n, then this solution has the smallest possible digit sum among all decimal decompositions of n.

- If $k = 0$, then the target value n is an integer between 0 and 9 (since otherwise Algorithm 2 would use a larger value of k). Therefore, in this case it is not possible for *any* decimal decomposition of n to use any tokens other than unit tokens. There is a unique decomposition of n that uses unit tokens only, so this solution is optimal by default.
- Now assume that, for some k, if Algorithm 2 outputs $d_0...d_k$ for a given target integer n, then this solution has the smallest possible digit sum among all decimal decompositions of n. Consider an input n for which Algorithm 2 outputs a sequence of token counts of the form $d_0...d_{k+1}$. Let $c_0...c_{k'}$ be any other decimal decomposition of n. First, notice that k' must be less than or equal to k. This is because k is chosen by Algorithm 2 to be the largest exponent of 10 for which 10^k does not exceed n.

Pad the sequence $c_0...c_{k'}$ with 0's at the left if necessary, so that we may take k' to be equal to k with no loss of generality.
We claim that if the most significant digits differ in the two expansions: $c_k \neq d_k$, then the competing solution $c_0...c_k$ has a token count $\sum_{i=0...k} c_i$ that is strictly larger than that of Algorithm 2's solution $d_0...d_k$, and therefore $c_0...c_k$ cannot be optimal. We already know that because Algorithm 2 chooses d_k to be as large as possible, c_k either equals d_k, or else c_k is smaller than d_k. So, assume that c_k is strictly smaller than d_k. Since the sums $\sum_{i=0...k} c_i 10^1$ and $\sum_{i=0...k} d_i 10^i$ must both equal n, this means that the positive nonzero deficit $(d_k - c_k)10^k$ associated with the most significant digits must be accounted for by the remaining digits $c_{k-1}...c_0$. Because less significant digits correspond to lower powers of 10, at least 10 tokens will be needed to recover each 10^k token in the c solution as compared with the d solution. This means that the total token count for c will be larger than that for d, so that c cannot be optimal. Hence, we conclude that c_k must in fact be exactly equal to d_k if c has a token count that is no larger than that of d. The equality of the leading digits now implies that the remaining digits nust together provide a decimal expansion of the remaining value $n - d_k 10^k$ in both the c and d expansions. By the induction hypothesis, we know that the shorter subsequence $d_{k-1}...d_0$ has minimum token count for this new target value, which completes the optimality proof.

2. (From exercise 5.22 in the book by Dasgupta, Papadimitriou, and Vazirani)

   (a) Use the Cut Property as discussed in class to prove the property of undirected weighted graphs stated below.

   > Given any cycle of the graph, there is a minimal spanning tree that does *not* contain the heaviest edge of that cycle.

   Hint: given a cycle, start by picking any MST T of the graph. There are two cases depending on whether the heaviest edge of the cycle forms part of T or not.

Solution

Let C be a cycle of a weighted connected graph G, and let T be any MST of G. Let e be the heaviest edge of the cycle C. If e does not form part of T, then T is a MST of G that does not contain e, so T satisfies the requirements of the statement,
Suppose, then, that e *does* form part of T. We will construct a different MST of G that does not contain e. Note that removing e from T disconnects T into two connected components S and S'. This is because T, being a tree, has exactly one fewer edge than vertices. Since T spans G, the two components S and S' together contain all of the vertices of the graph G, and hence $\{S, S'\}$ for a cut of G. No edges of T other than e cross this cut – otherwise, $T \setminus \{e\}$ would still be connected. Also, $T \setminus \{e\}$ can obviously be extended to a MST of G, namely T. Hence, by the Cut Property discussed in class, adding to $T \setminus \{e\}$ the lightest edge e' of G that crosses the cut will result in a set that can still be extended to a MST T' of G. Since $T \setminus \{e\}$ already spans all of the vertices of G and is only one edge short of being a tree, the resulting enlarged set $T \setminus \{e\} \cup \{e'\}$ will be a spanning tree and therefore must in fact already be a MST.

I claim that we may assume without loss of generality that the lightest edge e' that crosses the cut $\{S, S'\}$ is *not* the same edge as e. This is because since C is a cycle, some edge e'' of C other than e must cross the cut – removing a cycle edge cannot disconnect G. This other edge e'' can be no heavier than e by definition of e as the heaviest edge of C. If e'' is strictly lighter than e, then clearly e cannot be the lightest edge of G across the cut. Otherwise, we may simply replace e by e'' without changing the overall edge weight. The resulting MST T' does not contain e.

(b) Prove carefully that Algorithm 3 works correctly, that is, that it returns a MST of the input graph.

Algorithm 3: MST By Pruning
Input: Weighted undirected graph G.
Output: A MST of G.
PRUNEMST(G)
(1) sort the edges of G in decreasing order of weight
(2) **foreach** edge e (starting with the heaviest)
(3) **if** e is part of a cycle of G
(4) remove e from G
(5) **return** G

Solution

First, prove the following loop invariant by induction in the number of iterations of the main loop, using 2a.

> There exists a MST of the original input graph that contains only edges selected from among those that remain.

Second, we must guarantee that the Algorithm 3 does not terminate prematurely. To this end, notice that the main loop is exited only after no cycles remain in the pruned graph. Thus, the pruned graph at this point will be acyclic. By the above loop invariant, this pruned graph will contain a MST of the original graph. The latter point implies that the edges of the pruned graph span all of the vertices of the original graph. Therefore, the pruned graph that remains when the loop is exited is a spanning tree of the original graph. Since the pruned graph contains (by the loop invariant) a MST of the original graph, it (the pruned graph) must in fact be a MST.