Homework 4

Thomas Kim tsk389 51835

David Munoz dam2989 51840

CS331 Algorithms and Complexity

Problem Q1. Suppose you're consulting for a company that manufactures PC equipment and ships it to distributors all over the country. For each of the next n weeks, they have a projected $supplys_i$ of equipment (measured in pounds), which has to be shipped by an air freight carrier.

Each week's supply can be carried by one of two air freight companies, A or B.

- Company A charges a fixed rate r per pound (so it costs $r \cdot s_i$ to ship a week's supply s_i)
- Company B makes contracts for a fixed amount c per week, independent of the weight. However, contracts with company B must be made in blocks of four consecutive weeks at a time.

Give a polynomial-time algorithm that takes a sequence of supply values $s_1, s_2, \ldots, s_n$ and returns a *schedule* of minimum cost. $OPT(i)$

1: **if** $i = 0$ **then**
2: **return** 0
3: **end if**
4: **if** $i < 0$ **then**
5: **return** $INVALID$
6: **end if**
7: **return** $min(rs_i + OPT(i-1), 4c + OPT(i-4))$

Memoization: Store an array of $[OPT(0), OPT(1), \ldots, OPT(n)]$
Each element can be accessed in constant time by hashing based on i **Correctness**:

Proof. The schedule of using only Company A is explored by this algorithm.
At every step of that schedule, the algorithm explores the possibility of using Company B.
Therefore, the entire space of possible schedules is explored by the algorithm.
By the last line of the algorithm, it will find the minimum cost schedule in the space of explored schedules.

□

Runtime:

Proof. OPT could possibly be computed for n different values of i.
Computing $OPT(i)$ will use two constant time lookups for $OPT(i-1)$ and $OPT(i-4)$
Therefore the worst case complexity is $O(n)$

□

Problem Q2(a). Assume that you have to make change for N, and that you have an infinite supply of each $C = c_1, c_2, \ldots, c_n$ valued coins where $1 \leq c_i \leq N-1$. Compute the minimum number of coins required to make the change. Provide an algorithm which solves the problem using dynamic programming, prove correctness and runtime. $OPT(C, N)$

1: $C = c_1, c_2, \ldots, c_k$
2: **if** $C = \{\} \wedge N \neq 0$ **then**
3: **return** $INVALID$
4: **end if**
5: **if** $N = 0$ **then**
6: **return** 1
7: **end if**
8: **if** $N < 0$ **then**
9: **return** $INVALID$

10: **end if**
11: **return** $min(OPT(C - \{c_k\}, N), 1 + OPT(C, N - c_k))$

Memoization Store a collection of k sparse arrays containing $OPT(C, N)$
Choose the array by keeping track of the number of elements in C, which can be done in amortized constant time.
This is achieved by counting during the first iteration and keeping track of it through successive calls to OPT.
Choose the index into the array using N, so the total lookup is amortized constant time.

Proof of correctness

Proof. Suppose some particular combination of coins C' is not explored.
Let $C' = \{c_{1,1}, \ldots, c_{1,k_1}, \ldots, c_{j,k_j}\}$ where the second subscript of each coin describes the individual instance of coin being used.
Consider the following execution:
First, OPT line 11 uses case 2 k_j times.
Next, OPT line 11 uses case 1.
Next, OPT line 11 uses case 2 k_{j-1} times.
Next, OPT line 11 uses case 1.
This repeats until all coins in C' have been used.
During this execution, the combination of coins C' has been explored.
This contradicts the original assumption. □

Proof of runtime

Proof. There are n subsets of C and N values of N which could be computed.
We see that OPT could be computed N^2 times in the worst case, since $1 \leq c_i \leq N - 1$
Therefore the worst case runtime of this algorithm is $O(N^2)$, which is polynomial in N.

□

Problem Q2(b). Solve the same problem, but this time assume you have a limited supply p_i of each coin c_i Provide a dynamic programming algorithm, do not prove correctness or runtime. $Setup(C)$

1: $C' = \emptyset$
2: **for** $\forall c_i \in C$ **do**
3: $\quad c_{i,1}, c_{i,2}, \ldots, c_{i,p_i} = c_i$
4: $\quad C' = C' \cup \{c_{i,1}, c_{i,2}, \ldots, c_{i,p_i}\}$
5: **end for**

$OPT(C, N)$
1: $C = c_1, c_2, \ldots, c_k$
2: **if** $C = \{\} \wedge N \neq 0$ **then**
3: $\quad$ **return** $INVALID$
4: **end if**
5: **if** $N = 0$ **then**
6: $\quad$ **return** 1
7: **end if**
8: **if** $N < 0$ **then**
9: $\quad$ **return** $INVALID$
10: **end if**

11: **return** $min(OPT(C - \{c_k\}, N), 1 + OPT(C - \{c_k\}, N - c_n))$

Memoization: The memoization is same as $Q2(a)$
The solution is $OPT(Setup(C), N)$

Problem Q2(c). Assume $C = 1, 2, 4, \ldots, 2^m$ and $N < 2^{m+1}$. For this special case, give an $O(m)$ time algorithm that solves the problem in (a).

1: $count = 0$
2: Express N as a binary number $n_1 n_2 n_3 \ldots n_k$
3: **for** $1 \leq i \leq 2^m$ **do**
4: **if** $n_i = 1$ **then**
5: $count := count + 1$
6: **end if**
7: **end for**

Correctness

Proof. Thm: For every element $x \in Q_p$, where Q_p is a field of $p - adic$ integers, there exists a unique representation $x = \sum_{i=0}^{k} a_i p^i, a_k \neq 0$
By expressing N as a 2-adic integer, it is guaranteed that there exists some representation of N in the form shown above.
Note that k in the above sum is bounded by $k < m + 1$, since $2^{m+1} > N$
$\forall 0 \leq i \leq m, 2^i \in C$ Therefore, there exists a unique representation of N as a sum of the elements in C. □

Runtime

Proof. Expressing N as a binary number is $O(m)$ time, since N expressed in binary has at most m digits.
Counting the 1's in the binary representation of N takes $O(m)$ time, for the above reason.
The complexity is therefore $O(2m) = O(m)$ □

Problem Q3. An opportunity cycle is one where the product of the ratios along the cycle is greater than 1. Give a polynomial-time algorithm to find an opportunity cycle in a graph, if one exists.

1: Let $G(V, E)$ be the graph of possible trades
2: Let $w(e_i)$ be the weight of edge $e_i \in E$
3: Let $G'(V', E')|V' = V$
4: **for** $\forall e_i(u, v) \in E$ **do**
5: $e'_i = (u, v)|w(e'_i) = -log(w(e_i))$
6: $E' = E' \cup \{e'_i\}$
7: **end for**
8: **for** $\forall v' \in V'$ **do**
9: Run Bellman-Ford on G' starting on v' to detect negative cycles, return TRUE if one is detected
10: **end for**

Correctness:

Proof. Suppose there is some cycle $e_1, e_2, \ldots, e_k$ such that $\Pi_{i=1}^{k} e_i > 1$ $log\left(\Pi_{i=1}^{k} e_i\right) > log(1)$
$\implies \sum_{i=1}^{k} log(e_i) > 0$
$\implies \sum_{i=1}^{k} -log(e_i) < 0$
Therefore, by line 9, a negative cycle corresponds to an opportunity cycle.

□

Runtime:

Proof. Constructing G' takes $O(|V| + |E|)$ time.
Running Bellman-Ford takes $O(|V||E|)$ time.
Bellman-Ford is run $|V|$ times.
Therefore, the total complexity is $O(|V|^2|E|)$ which is polynomial in the number of vertices/edges. □