pa3

Kevin O'Connor, Gurbir Dhillon

April 21, 2012

Throughout we maintain the notations introduced in the assignment. Namely, we consider a set of n nonegative integers a_i, and we set $b := \sum_i a_i$.

1 Dynamic Programming

We first present a dynamic programming solution to the number partition problem, in $O(nb)$ time. Consider first the related 'subset problem' $S(k)$ of whether there exists a subset $I \subseteq \{1, \ldots, n\}$ such that

$$\sum_{i \in I} a_i = k$$

that is, there exists a subset which sums to k, for $0 \leqslant k \leqslant b$.

Lemma 1.1. *$S(k)$ can be solved in $O(nb)$ time, simultaneously for all k, $0 \leqslant k \leqslant b$.*

Proof. Write $S(i, k)$ for the problem restricted to the first i numbers: does there exists a subset $J \subseteq \{1, \ldots, i\}$ such that

$$\sum_{j \in J} a_j = k$$

with output T or F.

We have the initial condition:

$$S(1, k) = \begin{cases} T & a_1 = k \\ F & a_1 \neq k \end{cases}$$

and the recursion

$$S(i, k) = \begin{cases} T & S(i-1, k) = T \vee (a_i \leqslant k \wedge S(i-1, k - a_i) = T) \\ F & o.w. \end{cases}$$

To see the recursion, given a solution condition on whether a_i (the last number) was used or not: if it was not, then $S(i-1, k)$ was solvable, and if was, we have $a_i \leqslant k$, (recall $a_j \geqslant 0$), and hence a subset of the first $i-1$ numbers sum to $k - a_i$. It's easy to see both arguments are in fact if and only if.

We have that $S(n, k) = S(k)$, the desired solutions; in running time, initialization and each step of the recursion takes constant time, and we perform $O(nb)$ calculations, one for each $S(i, k), 1 \leqslant i \leqslant n, 0 \leqslant k \leqslant b$, as desired.

□

With the lemma, to solve the number partition problem, produce all the $S(k)$ as in the lemma. I claim that minimizing the residue, is equivalent to finding the k closest to $\frac{b}{2}$ for which $S(k)$ is true (k not necessarily unique).

To see this:

Lemma 1.2. *There exists a partition with residue $i \Leftrightarrow S(\frac{b+i}{2}) = T$.*

Proof. For a partition A_1, A_2 with residue i, write

$$s := \sum_{i \in A_1} a_i \qquad l := \sum_{i \in A_2} a_i$$

WLOG set $s \leqslant l$; informally these are the heights of the smaller and bigger piles, respectively. Note $l + s = b$

$$l - s = i \Leftrightarrow l - (b - l) = i \Leftrightarrow 2l - b = i \Leftrightarrow l = \frac{b+i}{2}$$

as desired. □

With the lemma, having computed the $S(k)$, let k_0 denote the least $k \in \{\frac{b}{2}, \ldots, b\}$ such that $S(k_0)$ is true, the corresponding minimal residue is $2k_0 - b$, by the lemma.

2 Karmarkar-Karp

We first briefly describe how to implement Karmarkar-Karp on $\{a_1, \ldots, a_n\}$ in $O(n \log n)$ time.

Create a binary heap $Leftovers$ and an array $Untouched$, initialize $Leftovers$ to empty and $Untouched$ to $\{a_1, \ldots, a_n\}$. Sort the a_i in $O(n \log n)$ time.

At the i^{th} step, find the maximal 2 elements of $Leftovers \coprod Untouched$, it takes constant time to extract the 2 greatest elements of $Leftovers$ and $Untouched$ each, as $Leftovers$ is a binary heap.

Remove m_1, m_2 from $Leftovers$ and/or $Untouched$, and then add $|m_1 - m_2|$ to $Leftovers$.

We perform $n - 1$ steps, observe recursively that the size of $Leftovers$ is bounded above by n, and we perform maximally 2 removals and 1 insert from it in each step, each step takes $O(\log n)$ time; at the end, we return the unique element remaining of $Leftovers$, hence this takes

$$O(n \log n) + nO(\log n) = O(n \log n)$$

time, as desired.