CS29003 Algorithms Laboratory

Assignment No: 7

Last date of submission: 13–March–2018

Two pirates Punto and Qunto are going to retire in two private islands. Before that, they want to divide their collected treasure equally between them. The treasure consists of N items costing $a_0, a_1, a_2, \ldots, a_{N-1}$ (positive integers). Let $S = \sum_{i=0}^{N-1} a_i$. Ideally, each pirate should get treasure of worth $S/2$. But the items cannot be broken into pieces, so an exact amount of $S/2$ may be unachievable from the given items ($S/2$ need not even be an integer). Let $T = \lfloor S/2 \rfloor$. Punto is happy if he gets treasure of worth $P \leqslant T$ such that P is as large as possible. In this assignment, you help the pirates by computing Punto's maximum possible share P.

Part 1: Exhaustive search

Write a recursive function **`exhs`** to compute and return P. For each $i = 0, 1, 2, \ldots, N-1$, take two decisions: include/exclude a_i in Pinto's treasure. Recursively, solve the two subproblems, and when the two calls finish, return the maximum of the two returned values. Notice that a collection of worth larger than T is not a valid solution, so consider only those choices whose total costs are $\leqslant T$.

Part 2: Hash-table utilities

We are going to use a hash table H for *memoizing* intermediate partial solutions generated during the exhaustive search. Suppose that items $a_0, a_1, a_2, \ldots, a_i$ are so far considered, and a subcollection of these items is of worth σ. Let $flag$ be a Boolean value standing for whether a_i is included in the subcollection corresponding to σ. The hash table stores triples of the form $(i, \sigma, flag)$. Use the hash function

$$h(i, \sigma) = 100003i + 103\sigma \pmod{s},$$

where s is the size of the hash table H. Notice that the Boolean flag is not incorporated in the hash function. Its only purpose is to aid the construction of an actual solution in Part 4.

The hash table H should be capable of storing $s = NT$ triples. Use open addressing with linear probing. Write the following functions:

(a) **`htinit(s)`** to create an empty hash table of storage capacity s,

(b) **`htsearch(H,i,`σ`)`** to determine whether $(i, \sigma, flag)$ already resides in the hash table H for some $flag$ (notice that the hash function does not depend on the flag; the return value can be 0 on failure, or $k+1$ if (i, σ) is found at index k in H), and

(c) **`htinsert(H,i,`σ`,flag)`** to insert a triple $(i, \sigma, flag)$ to H (this function may assume that no triple storing (i, σ) resides in H before the insertion).

Here, you do not delete entries from the hash table, so you do not need to write a delete function for H.

Part 3: Search using hash tables

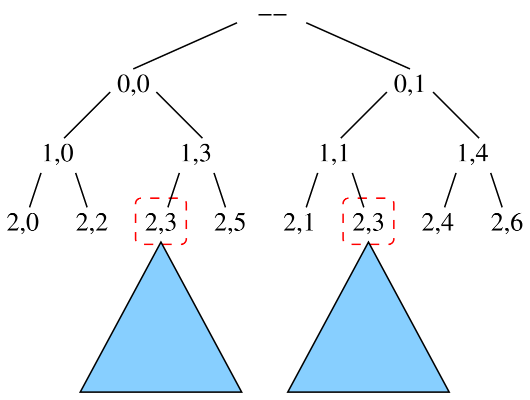
Write a hash-based search function **`hashs`** to compute and return Punto's share P. This is similar to the exhaustive search of Part 1 with the exception that the hash table H dictates the pruning of unnecessary search. The function takes as input an index $i \in \{0, 1, 2, \ldots, n-1\}$ and a sum σ of a subcollection of $a_0, a_1, a_2, \ldots, a_{i-1}$. The task of the call is to exercise two options:

(a) Exclude a_i: In this case, the recursive call will be on $(i+1, \sigma)$, and the Boolean $flag$ will be 0.

(b) Include a_i: In this case, the recursive call will be on $(i+1, \sigma + a_i)$, and $flag$ will be 1.

In the exhaustive search of Part 1, we blindly make both the calls. Now, we make the calls conditionally. In each of the two cases, we first search whether the current partial solution (that is, (i, σ) or $(i, \sigma + a_i)$ depending upon the case) already resides in the hash table H. If the search succeeds, we know that some other invocation of the function has already encountered this configuration, so there is no point re-exploring from this point. If the search fails, then this is a new configuration (for the moment), so we insert $(i, \sigma, 0)$ or $(i, \sigma + a_i, 1)$ to H depending upon which case it is, and make the recursive call. Let the two returned values be v_0, v_1. If a recursive call is not made, we take $v_0 = -1$ or $v_1 = -1$. Finally, $\max(v_0, v_1)$ is returned.

The following figure demonstrates the repetition of (i, σ) pairs in the search. Suppose that the first three item costs are $a_0 = 1$, $a_1 = 3$, and $a_2 = 2$. The left occurrence of $2, 3$ corresponds to a_1, and the right occurrence to $a_0 + a_2$. These two solutions are therefore achieved in two different ways. But since i and the partial sum σ are the same for these, the trees under these two nodes will be identical. The existence of $(2, 3)$ in H indicates that some invocation of the function has already explored this subtree. A second exploration is not needed irrespective of whether this subtree contains an optimal solution or not.



Part 4: Compute the solution

Write a function **`findsol`** that, given A, N, the maximum share P computed by **`hashs`**, and the hash table H as input, constructs a solution (a subcollection of the treasure items) that realizes the share P.

The *main*() function

- Read N and the item costs $a_0, a_1, a_2, \ldots, a_{N-1}$ from the user.
- Call **`exhs`**, and report the maximum share P.
- Call **`hashs`**, and report the maximum share P.
- Call **`findsol`** to print an optimal subcollection achieving the total value P of Punto's share.

Submit a single C/C++ source file. Do not use global/static variables.

Sample output

In a run of the first sample, you will probably not notice the difference between the running times of **`exhs`** and **`hashs`**.

```
        N = 16
        97926 95846 33555 87806 50758 26751 91574 52884 26587 42497 84104 64793
        52405 87805 98738 15616

        S = 1009645
        T = 504822

    +++ Exhaustive search
        Maximum P = 504804

    +++ Search with a hash table
        Maximum P = 504804
        Solution: 33555 + 26751 + 91574 + 52884 + 84104 + 64793 + 52405 + 98738
```

For the following larger sample, the effectiveness of hash-based pruning clearly shows up. On our laptop computer, **`exhs`** takes about 40 seconds, whereas **`hashs`** and **`findsol`** together finish within 3 seconds.

```
        N = 32
        38604 41509 61406 74764 49823 27743 62064 94358 16331 15622 79520 47685
        16574 10472 72027 42382 84860 43494 72273 78514 80934 19805 21113 46332
        65014 53259 50231 69886 60362 37064 42250 95318

        S = 1671593
        T = 835796

    +++ Exhaustive search
        Maximum P = 835796

    +++ Search with a hash table
        Maximum P = 835796
        Solution: 47685 + 72027 + 42382 + 84860 + 72273 + 78514 + 80934 + 19805 +
                  21113 + 65014 + 53259 + 60362 + 42250 + 95318
```