USACO NOV06 Problem 'plank' Analysis

by Richard Peng

This problem is the classical Huffman code problem, which asks for the minimum cost of placing all the vertices on the leaves of a binary tree where the cost is defined as weight of node* depth of node. It's quite clear these two problems are equivalent as each combining of two boards is equivalent to moving the two nodes up one level in their position in the tree.

The algorithm used to find the minimum cost of the Huffman tree is quite simple: take the two nodes with the least weight, combine them and insert the new node back into the list of nodes. Repeat until one node is left in the tree. Under C++, the algorithm can be implemented using the STL priority_queue in O(NlogN) time. With C and Pascal, the easiest way to implement it is as follows: sort all the weights in incremental order, keep two lists, the first containing all the uncombined weights and the second containing all the weights combined. Note that as the algorithm processes, the weight of the combined node is always increasing, so we could insert the new node to the end of the 2nd list in constant time. When we want to find the minimum pair of nodes, it suffices to check the 3 combinations of where the two nodes come from (1st, 2nd of one of the lists, 1st of both lists). This algorithm takes O(NlogN) for the sorting phase and O(N) in the combining phase.

Code (C++, by Bruce Merry):

```
#include <fstream>
#include <algorithm>
#include <queue>
#include <vector>

using namespace std;

typedef long long ll;

int main() {
    priority_queue<ll, vector<ll>, greater<ll> > q;
    ifstream in("plank.in");
    ofstream out("plank.out");
    int N;

    in >> N;
    for (int i = 0; i < N; i++) {
        ll x;
        in >> x;
        q.push(x);
    }

    ll cost = 0;
    for (;;) {
        ll a = q.top();
        q.pop();
        if (q.empty()) break;
        ll b = q.top();
        q.pop();
        cost += a + b;
        q.push(a + b);
    }
    out << cost << "\n";
    return 0;
}
```