USACO NOV13 Problem 'nochange' Analysis

by Bruce Merry

The constraint K <= 16 should immediately catch the eye: it suggests that the intended solution will be exponential in K - possibly 2^K to consider all subsets of the coins. Indeed, if we can determine whether any given set of coins can pay for everything, we can just iterate over all sets and find the set whose complement has the largest sum. This complement will be the set of coins left over.

To tell whether a set of coins will work, we use dynamic programming to answer a more general question: what is the longest prefix of the payments that a set of coins can cover? Given a set S, we consider all the possibilities for the last coin c, use dynamic programming to determine what can be paid with S \ {c}, and then see how far we can get from there using c. Since we already have a factor of 2^K in the running time, we can't afford a factor of N as well; this rules out a linear search. However, if we precompute the partial sums of the payments, we can use a binary search to find the last payment that will be covered by c.

The total running time is thus $O(N+2^K K \log N)$.

Below is Bruce Merry's solution. Note that STL's upper_bound function nicely expresses the binary search.

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

using namespace std;

int main()
{
    ifstream in("nochange.in");
    ofstream out("nochange.out");

    int K, N;
    in >> K >> N;
    vector<int> coins(K);
    for (int i = 0; i < K; i++)
        in >> coins[i];
    vector<int> acc(N + 1);
    for (int i = 0; i < N; i++)
    {
        int v;
        in >> v;
        acc[i + 1] = acc[i] + v;
    }

    vector<int> dp(1 << K);
    int ans = -1;
    for (int b = 1; b < (1 << K); b++)
    {
        int best = 0;
        int rem = 0;
        for (int i = 0; i < K; i++)
        {
            int m = 1 << i;
            if (b & m)
            {
                int last = acc[dp[b ^ m]];
                int trg = upper_bound(acc.begin(), acc.end(), last + coins[i])
                       - acc.begin() - 1;
                if (trg > best)
                    best = trg;
            }
            else
                rem += coins[i];
        }

        dp[b] = best;
        if (best == N)
            ans = max(ans, rem);
    }

    out << ans << '\n';
    return 0;
}
```