Algorithms and Datastructures assignment 2

Thomas Broby Nielsen (xlq119)
Tobias Overgaard (vqg954)
Christian Buchter (zvc154)

15. maj 2015

Indhold

1 Task 1

$$|N(c,i)| = \begin{cases} 0 & \text{if } c \leq 0 \\ 0 & \text{if } i = 0 \\ 1 + N(c, i-1) & \text{if } p_i = c \\ N(c - p_i, i-1) + N(c, i-1) & \text{otherwise} \end{cases}$$

2 task 2

In order to prove the correctness of our formula, for calculating the amount of unique combinations of unique beers you can buy with a specified amount of money, we will have to tjeck the three properties of a correct formula.

First we will first prove the correctness of the initialization of our formula.
This we have summarized into showing that for a given value of $1 \leq c$ and $1 \leq i$ that our base case holds true for the first iteration. For the example above, it is trivial to see that if $N(1,1)$ then the base cases for the formula, won't be triggered. From this we can see that initialization will hold for any value of $1 \leq c$ and $1 \leq i$.

The second property that our formula must hold, is that if the invariant before an iteration is true, must remain true after the iteration. To show that our

To further prove the correctness of our formula, we have see if the formula holds true before and after an iteration.

3 Task 3

Bellow we have written the pseudo-code for MemoizedBeerComp using the DP-method

```
MemoizedBeerComp(c, i, beerList, holderMatrix)
if c <= 0 or i==0
   return 0
if array(c, i) > -1
   return holderMatrix(c, i)
if c - beerList[i] == 0
    array(c,i) = 1 + MemoizedBeerComp(c, i - 1)
    return holderMatrix(c,i)
  else
      holderMatrix(c,i) =
         MemoizedBeerComp(c, i - 1, beerList, holderMatrix) +
         MemoizedBeerComp(c - beerList[i], i - 1, beerLis, holderMatrix)
      return holderMatrix(c, i)
```

Since we weren't able to calculate the running time for our memoizedBeerComp for task 4, with our top-down implementation of memoizedBeerComp, we decided to also implement our original formula as a bottom-up method instead, as seen below.

```
Bottom-Up-Beers (P,c) =
  let R[1,2, ... ,c][0,1,2,...,P.length] be a new Matrix
  for j=1 to c{
     R[j][0] = 0;
     for k = 1 to P.length {
        q = R[j][k-1];
        if (P[k] == j){
            q+=1}
        if (P[k] < j) {
           q += R[j-P[k]][k-1];
         }
     R[j][k] = q;
     }
  }
return R[c][P.length];
```

NOTE: we later figured out how to calculate the running time for the top-down method implementation after talking with a TA, but we decided to keep both of the implementations.

4 Task 4

For this task we went ahead and analysed the bottom-up implementation in task 3 made from our original formula in task 1. Because of the nested forloop nature in our implementation, it is easy to see that the running time for our implementation is $O(cn)$ where $n = p.length$ is the amount of individual beers. Therefore our running time, that is based off of the upper bound, is equal to $\theta(cn)$.

Since our matrix has to be stored for the algorithm to access it our memory usage is ||R|| = cn.

We prove the correctness by using the loop invariants:

- At the start each iteration of the inner loop the value of R[j][$k-1$] is the number of combinations of the p[1..$k-1$] beers achievable with j kr.
- At the start of each iteration of the outer loop the value of R[$j-1$][n], where n is P.length, is the number of combinations of all the beers achievable with $j-1$ kr.

Initialisation
Before the first inner loop the value of R[j][0] is 0, which is the number of combinations of 0 beers achievable with j kr

Before the first outer loop the value of R[0][n] is 0, which is the number of combinations of all the beers achievable with 0 kr.

Maintenance

If R[j][$k-1$] is true before the $k'th$ inner loop R[j][k] will be true after it since if $k = j$ we add that as a standalone combination to R[j][$k-1$], and otherwise if $j >$ P[k] we add the combinations from R[$j - P[k]$][$k-1$] which gives us all the combinations of R[j][k] as long as earlier iterations were Initialised correctly.

If R[$j-1$][n] is true before the $j'th$ iteration of the inner loop R[j][n] will be true after it since if the loop invariant for the inner loop holds after termination the $n'th$ iteration of the inner loop R[j][n] will contain the number of combinations of all the beers achievable with j kr.

Termination

The condition causing the inner loop to terminate is $k > n$. because each loop iteration increases k by one and because $n \neq \infty$ the loop must terminate. Since Maintenance of the inner loop gives us that R[j][n] will be true after the $n'th$ iteration we have that the loop will terminate and give us the loop invariant of the outer loop after the $n'th$ iteration.

The condition causing the outer loop to terminate is $j > c$. because each loop iteration increases j by one and because $c \neq \infty$ the loop must terminate. Since Maintenance of the outer loop gives us that R[c][n] will be true after the $c'th$ iteration we have that the loop will terminate after the $c'th$ iteration and give us the total number of combinations of the n beers achievable with c kr.