AD - Assigment 2

pwn274, npd457, kgt356

February 2022

Contents

1 Group part **2**
1.1 Give a greedy algorithm that solves this problem in O(n) time 2
1.2 Give a formal proof of correctness of your algorithm from task 1 2
1.2.1 Optimal Substructure 2
1.2.2 Greedy Choice Property 2
1.3 Using the procedure from task 1 give an O(n log B) algorithm to find the maximum value $\hat{b}$ 3

2 Individual parts **3**
2.1 pwn274 - Jakob Hallundbæk Schauser 3
2.2 npd457 - Sebastian Ø. Utecht 3
2.3 kgt356 - Christoffer A. Ankerstjerne 4

1 Group part

1.1 Give a greedy algorithm that solves this problem in O(n) time

After some careful consideration, we have come up with the following recursive algortihm:

```
function run(beers, places, b, i):
    if (i = len(beers))
        return min(beers) == b

    surplus = beers[i] - b

    distance = places[i+1] - places[i]

    if (d < 0)
        beers[i+1] -= surplus + distance
        beers[i] -= surplus

    if (d > 0)
        forward = surplus - distance
        if (forward > 0)
            beers[i+1] += forward
            beers[i] -= surplus


    return run(beers, places, b, i+1)
```

The main idea is as follows: Every bar looks at the difference between their stock and the required stock $\bar{b}$. If they have a deficit, they request beer from the bar to the right. If they on the other hand have a surplus, they send as much as possible to the next bar. This is run iteratively until the final bar is reached.

1.2 Give a formal proof of correctness of your algorithm from task 1

1.2.1 Optimal Substructure

Assume a global optimal solution of the problem, meaning all bars have at least $\bar{b}$ beers in stock. This means that each individual bar has at least $\bar{b}$ beers in stock. Thus the problem exhibits optimal substructure, as the global optimal solution consists of local optimal solutions.

1.2.2 Greedy Choice Property

Our algorithm takes the greedy choice of ensuring that the bar in question (at position p_i) is always adequately stocked, as if it is not, it will request beers from the bar at position p_{i+1}. This means, that after the algorithm is run for this bar, it has at least $\bar{b}$ beers in stock. In a global optimal solution (all bars being stocked with at least $\bar{b}$ beers) this bar would also have at least $\bar{b}$ beers, and it is thus contained in such a solution. This proofs that the problem exhibits the greedy choice property.

1.3 Using the procedure from task 1 give an O(n log B) algorithm to find the maximum value $\hat{b}$

In order to achieve we $\mathcal{O}(n \log B)$ running time we implemented a binary search, which searches for the optimal value of $\hat{b}$. Our initial algorithm runs in linear time, as it only considers each bar once, and binary search runs in $\mathcal{O}(\log B)$ time, as the recursion tree bottoms out at $\log_2 B$ levels. This means that our final algorithm will run in $\mathcal{O}(nlogB)$ time.

```
function search(low, hi):
    if (low + 1 = hi)
        return low

    mid = floor((hi + low)/2)

    if (run(mid))
        return search(mid, hi)
    else
        return search(low, mid)
```

2 Individual parts

2.1 pwn274 - Jakob Hallundbæk Schauser

- Explain greedy algorithms in general
  - special case of dynamic programming.
  - $OPTIMAL(P) = OPTIMAL(P') + x$
- Greedy Choice Property
- Optimal substructure.
- Example: Activity selection.
  - Greedy Choice Property: Ends first because of sorted assumption
  - Optimal substructure: A_{ij} must include A_{ik} and A_{kj}
- Argue for optimal ($\mathcal{O}(N)$) running time
- Round off and talk pitfalls

2.2 npd457 - Sebastian Ø. Utecht

Greedy Algorithms Disposition

- Optimal substructure
- Greedy Choice property
- Combination: An optimal solution can be found via recursive use of a greedy algorithm
- **Example:** Activity Selection

  - Problem: To select activities such that the largest amount of activities can be held within a given timespan.
  - Optimal substructure.
  - Greedy Choice Property
  - Solution

2.3 kgt356 - Christoffer A. Ankerstjerne

- Explain greedy algorithms
- Greedy Choice Property
- Optimal substructure.
- Example: Activity selection.