# LiU Challenge 23 Solutions

**LoweK** **HermanL**

## Skeleton Array

Let array $B$ have $B_1 = 0$ initially. Always let $B_{i+1} = B_i - A_i$, then $S(B) = A$ always holds. Note that if we add some constant $c$ to $B_1$ then the sum of $B$ is affected by $c(N+1)$. We want to minimize the absolute value of the sum of $B$, therefore we would like to solve $0 = \sum B + x(N+1)$ for $x$, which gives $x = -\frac{\sum B}{N+1}$. Setting $B_1 = x$ gives the absolute value of the sum of $B$ equal to zero, but since $x$ is not guaranteed to be an integer, try the nearest integer solutions.

## Desire For Coffee

Distance sum of stack is independent of stack ordering as addition is commutative.

Let $S$ be the an optimal stack where higher indices of $S$ denotes higher up on the stack.

**Claim:** If $W_i + C_i \leq W_{i+1} + C_{i+1}$ holds for turtles $t_i$ and $t_{i+1}$ of $S$, swapping positions of turtles $t_i$ and $t_{i+1}$ maintains an atleast as good remaining minimum carrying capacity as not swapping.

**Proof:** Note that, up until index $i - 1$ (if such exist) of stack $S$, the conditions; the least remaining carrying capacity $R_S$ of stack $S$, is equal between alternative $t_i$ and $t_{i+1}$ placements. Comparing the options, the new minimum remaining carrying capacity of the stack becomes:

$t_i$ remains under $t_{i+1}$: $\Rightarrow \min(C_i - W_{i+1}, C_{i+1})$

$t_i$ on top of $t_{i+1}$: $\Rightarrow \min(C_{i+1} - W_i, C_i)$

(OR possibly $R_S - W_i - W_{i+1}$, in which case both alternatives are as good).

We have $W_{i+1} + C_{i+1} \geq W_i + C_i \Leftrightarrow C_{i+1} - W_i \geq C_i - W_{i+1}$ and therefore both $C_{i+1} > C_{i+1} - W_i \geq C_i - W_{i+1}$ and $C_i > C_i - W_{i+1}$ holds. Therefore the least term is sure to be $C_i - W_{i+1}$ and thus $\min(C_{i+1} - W_i, C_i) \geq \min(C_i - W_{i+1}, C_{i+1})$. The remaining carrying capacity of the stack will always be as large or larger if we place $t_i$ on top of $t_{i+1}$ i.e. arrange the stack in descending order with respect to $W + C$ for higher positions on the stack. □

It follows from the claim that there exist an optimal stack where $W_i + C_i \geq W_{i+1} + C_{i+1}$ holds for all turtles in the stack. The problem can be solved with DP over the turtles sorted in descending order with respect to $W + C$. Let $d[i]$ be the largest crawling distance constructable with $i$ remaining carrying capacity, $i \leq 500$. Iterating over sorted turtles, the transition step $d[i - W] = \max(d[i - W], d[i] + L)$ evaluates whether using this turtle can construct a better solution.

## Tidy Racks

$n$ - number of racks
$k$ - rack size
$w_1, w_2, w_3, w_4$ - weight types 5, 10, 15, 20 respectivly
$x_1, x_2, x_3, x_4$ - $x_i$ denotes number of weights of type $w_i$
$x$ - total number of weights ($x = x_1 + x_2 + x_3 + x_4$)
$r_1, r_2, r_3, r_4$ - $r_i$ denotes the number of racks required to hold all weights of type $w_i$ subject to rack size $k$, $r_i = \lceil \frac{x_i}{k} \rceil$
$r$ - total racks required ($r = r_1 + r_2 + r_3 + r_4$)

The first condition, condition (1), $r \leq n$ must hold, else there cannot exist a solution since there are too few racks.

Next consider case $k = 1$, then everything is solved from start.

Next consider special cases $n = 1$ and $n = 2$. If $n = 1$ then (1) is sufficient for there to be a solution, else no. If $n = 2$, we may concatenate the stacks top-to-top and check whether $a[i] \neq a[i + 1]$ holds at most one time, together with condition (1).

Next do case work for $n \geq 3$ and $k \geq 2$ over $0 \leq x \leq (n-1)k$ and $(n-1)k < x \leq nk$

**case 0 <= x <= (n-1)k:**

Note that $x \leq (n-1)k$ implies that we can create an empty rack by moving all weights from one rack into the others.

**Claim:**
We can rearrange the weights however we want subject to the constraints if we have an empty rack with $n \geq 3$ and $k \geq 2$.

**Proof:**

(here higher indices means lower positions in the stack)

1) Wlog. denote racks A,B,E where E is empty initially, and both A and B are non-empty racks. We can swap weights A[i] and A[j] $(i < j)$ of stack A all else equal with the following steps:

```
Move 1 weight from B to E.
Move weights from A to E until A[i] is on top of A.
Move 1 weight (A[i]) from A to B.
Move weights from A to E until A[j] is the last to be placed on E.
Move 1 weight (A[i]) from B to A.
Move 1 weight (A[j]) from E to B.
Move weights from E to A until next position of A is A[i]'s old position.
Move 1 weight (A[j]) from B to A.
Move weights from E to A until one weight is left on E
Move 1 from E to B.
```

2) Wlog. denote racks A,B,E where E is empty initially, and both A and B are non-empty racks. We can swap any two weights from different stacks i.e. swap A[i] with B[j].

```
Use 1) to put A[i] on top of A.
Use 1) to put B[j] on top of B.
Move 1 (A[i]) from A to E
Move 1 (B[j]) from B to A
Move 1 (A[i]) from E to B
```

The given operation (move one topmost weight from one stack to another), together with 1) swap weights internally and 2) move weights between stacks, enables us to rearrange the weights however we want. □

If condition (1) holds, then a target configuration that solves the problem exists. Then, since we can rearrange the weights however we want we construct the solution. This case is **always** solvable.

**case (n-1)k < x <= nk:**

This case is similar to the previous case only that we can never reach the $x - (n-1)k$ lowest weights in each rack. Simply, check if the bottom-most $x - (n-1)k$ weights of each stack are of the same type, and that the locked configuration encloses our requirements. i.e. if our given configuration has

$r_1 = X, r_2 = Y, r_3 = Z$ and $r_4 = C$, then we must have atleast $X$ bottom locked stacks of type $w_1$ , $Y$ of $w_2$, $Z$ of $w_3$ and $C$ of $w_4$, else it is impossible. If we do then we can omit thinking about the locked bottom $x - (n-1)k$ weights and use the algorithm described for previous case to order all the weights correctly.

## Choir Of Birds

The problem can be solved with DP.

Let $v[f, n, d]$ be the the total volume from the branch starting with node $n$, excluding the edge from $f$ to $n$, where the distance from the starting node to $n$ is $d$. The goal is to find the node $m$ that maximizes $v[-1, m, 0]$. ($-1$ denotes an invalid node, to stop $v$ from excluding any branch at the starting node). $v$ can be calculated using the formula

$v[f, n, d] = \left\lfloor \frac{A_n}{B_n + d^2} \right\rfloor + \sum_k v[n, k, d + 1]$

Where the sum is over every node $k$ connected to $n$, except the case where $k = f$. The total number of states is bounded by $2N^2$ (number of edges, in both directions, times maximum distance), and the time to evaluate $v$ is bounded by $O(N)$ (maximum degree of the nodes), making the total time complexity $O\left(N^3\right)$.

Problem 1: Large distances. Because $\left\lfloor \frac{A_i}{B_i + d^2} \right\rfloor = 0$ when $d^2 > A_i - B_i$, it is unnecessary to calculate $v$ for distances larger than $\sqrt{\max_i A_i}$. More formally: $v[f, n, d] = 0$ where $d^2 < \max_i A_i$. This brings the upper bound of the total number of states down to $N\sqrt{\max_i A_i}$ and the total time complexity down to $O\left(N^2\sqrt{\max_i A_i}\right)$.

Problem 2: Star graphs. The total time spent per node is proportional to the degree of the node squared (number of edges to the node times the number of edges from the node). However, this can be reduced by noting that, when $f$ is connected to $n$, $v[f, n, d] = v[-1, n, d] - v[n, f, d + 1]$ (the total of the edges from $n$ excluding $f$ is the total all edges from $n$ minus the edge to $f$). This means that we only have to compute the sum once per node and distance, reducing the total time to $O\left(2d_{\max}N + d_{\max}\sum_i \deg(i)\right) = O(2d_{\max}N + 2d_{\max}N) = O(N \max_i A_i)$.