Algorithms

DATE : 08/03/2022

GALE - SHAPELEY ALGORITHM

- Initially, each $a_i \in A$, and each $b_j \in B$, is FREE
- While there exists some FREE a_i that has not yet proposed to each $b_j \in B$

$O(1)$

- Let a_i be a FREE person that has not proposed to each $b_j \in B$ (1)
- Let $B' \subseteq B$ be the set of b_j such that a_i has not yet proposed to (2)
- Let $b_j \in B'$ be the person from B' that a_i likes the most

$$a_1 : b_1 > \boxed{b_2} > b_3$$
$$B' = \{\boxed{b_2}, b_3\}$$

b_2 is the most preferred in the B' set

- IF b_j is FREE : (3)
    - MATCH UP a_i and b_j // a_i & b_j get engaged
    - a_i and b_j are not free anymore
- ELSE :
    - Suppose that b_j is engaged to a_k (3)
    - IF b_j likes a_k more than a_i (4)
        - a_i REMAINS FREE
    - ELSE :
        - the match between b_j and a_k is broken
        - a_i and b_j are matched up
        - a_k becomes FREE

1. Identify a free a_i
2. Identify the b_j that is highest in a_i's preference list, that a_i hasn't yet proposed to
3. Given b_j, determine whether she's free, and - if not - identify her current partner a_k.
4. Determine whether b_j likes a_i more than a_k

To implement each of the 4 operations in $O(1)$ time, we use the following data structures:

(A) APREF, A $n \times n$ array, such that APREF[i][l] contains the index of the l^{th} most preferred b_j in a_i's ranking.

EX: if $a_i : b_{i_1} > b_{i_2} > \dots > b_{i_n}$

then $APREF[i][l] = i_l$

$l_3 : b_2 > b_1 > b_3$

$APREF[3][1] = 2$
$APREF[3][2] = 1$
$APREF[3][3] = 3$

(B) NEXT, an array of size n, such that next[i] contains the rank of the next b_j in a_i's list, that a_i is going to propose to.

At the outset, $next[i] = 1 \quad \forall i$.

When a_i is about to propose, he'll propose to b_j for $j = APREF[i][next[i]]$; after having proposed, $next[i] += 1$

↳ thus (2) can be solved in $O(1)$

(C) CURRENT, an array of size n, such that current[i] contains the index j of the a_j that b_i is currently engaged to; if b_i is currently free, then CURRENT[i] = None ($\perp$) ← simbolo per dire None

Thus, (3) can be solved in $O(1)$ time.

D) RANKING, an $n \times n$ array, such that RANKING[j][i] is the rank of a_i in b_j's preference list.

If $b_j : a_{j_1} > a_{j_2} > \dots > a_{j_n}$, then RANKING[j][$j_\ell$] = ℓ

EXAMPLE : if $b_3 : a_2 > a_1 > a_3$ then

RANKING[3][2] = 1
RANKING[3][1] = 2
RANKING[3][3] = 3

To decide whether b_j likes a_k more than a_i it is sufficient to check whether
RANKING[j][k] < RANKING[j][i]

Thus, 4 can be solved in $O(1)$ time

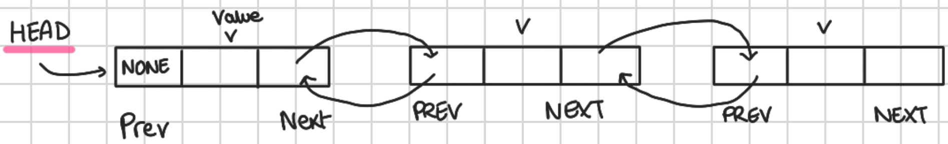
So, 2, 3 and 4 can be implemented in $O(1)$ time, if one has arrays A, B, C, D. (It is easy to check that A, B, C, D can be built $O(n^2)$ time.)

So, the only thing that remains to be done is to solve/implement 1 in $O(1)$ time. If we can do that, the total runtime G-S becomes:

$$O(n^2) + \underbrace{n^2}_{\text{nº of iterations}} \cdot (\underset{1}{O(1)} + \underset{2}{O(1)} + \underset{3}{O(1)} + \underset{4}{O(1)}) = O(n^2)$$

1 IDENTIFY A FREE a_i
HOW TO DO IT IN $O(1)$ TIME?

To do this, we're going to use doubly-linked lists.

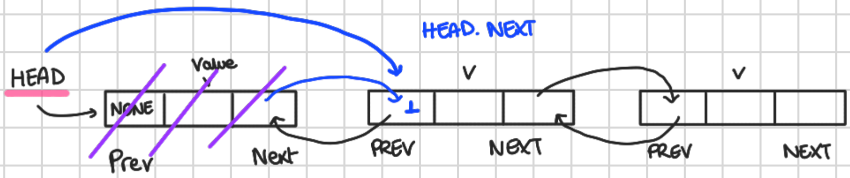


To get the first element of the list, I can just run.

HEAD. V.

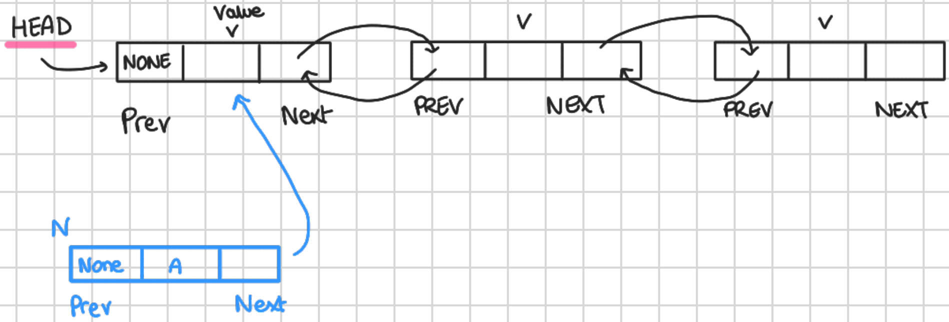
To remove the first element from the list

HEAD = HEAD. NEXT
HEAD. PREV = NONE



Now, we only need to attach to the list some a_i that is rejected by b_j.

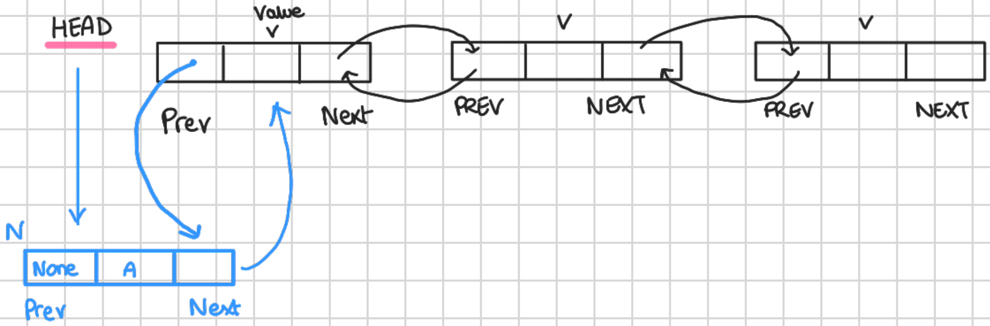
Whenever b_j has to choose between a_i and a_k, the "UNCHOSEN" one (let us say it is a_t for $t \in \{i, k\}$) will be added back to the list.



VISUAL REPRESENT.

N = List()
N. PREV = NON
N. NEXT = HEAD
N. V. = A
HEAD. PREV = N
HEAD = N

VISUAL REPRESEN.



THM: G-S can be implemented to run in $O(n^2)$ time.