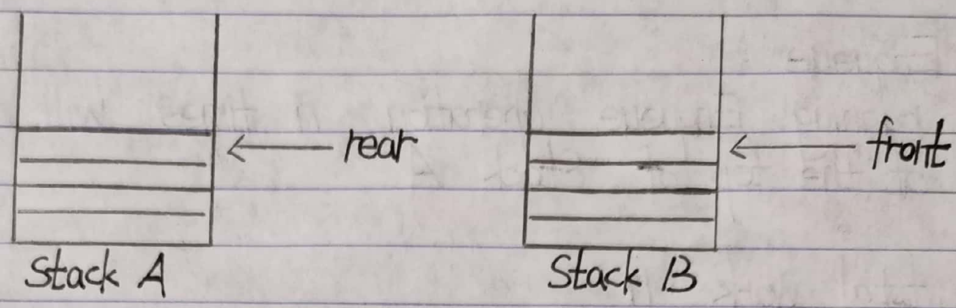
1.(a) use two stacks to implement a queue :



Enqueue :

insert the new element at the top of Stack A

Dequeue :

- if Stack B is empty :
  - if Stack A is empty :
    - do no operation and return null
  - else , Stack A is not empty :
    - while Stack A is not empty :
      - remove the element at the top of Stack A and insert it at the top of Stack B
    - remove the element at the top of Stack B and return it
- else , Stack B is not empty :
  - remove the element at the top of Stack B and return it

(b) aggregate method :

Enqueue :
running Enqueue operation n times will cost n insertions at the top of Stack A

total work : n
on average : $\frac{n}{n} = 1 \in O(1)$ amortized runtime

Dequeue :
running Dequeue operation n times will cost n removals from the top of Stack A, n insertions at the top of Stack B, and n removals from the top of Stack B

total work : $3n$
on average : $\frac{3n}{n} = 3 \in O(1)$ amortized runtime

(c) accounting method :

we assign each element 4 rubles at first

Enqueue :
1 ruble will pay for the insertion at the top of Stack A

Dequeue :
1 ruble will pay for the removal from the top of Stack A
1 ruble will pay for the insertion at the top of Stack B
1 ruble will pay for the removal from the top of Stack B

This covers all of the necessary work, meaning that we have $1 \in O(1)$ amortized runtime for Enqueue and $3 \in O(1)$ amortized runtime for Dequeue

(d) potential method:

$$T_{amortized} = T_{actual} + C(\Phi_{after} - \Phi_{before})$$

here we define $\Phi = n$; n is the number of elements in Stack A. $C = 2$

Enqueue:

1 insertion at the top of Stack A

$T_{actual} = 1$

$\Phi_{before} = n$, $\Phi_{after} = n+1$

$T_{amortized} = 1 + 2 \times (n+1-n) = 3 \in O(1)$

Enqueue has $O(1)$ amortized time

Dequeue:

if Stack B is empty:

if Stack A is empty:

no operation, $T_{amortized} = 0 \in O(1)$

else, Stack A is not empty:

n ~~insertions~~ removals from the top of Stack A, n insertions at the top of Stack B, 1 removal from the top of Stack B

$T_{actual} = 2n + 1$

$\Phi_{before} = n$, $\Phi_{after} = 0$

$T_{amortized} = 2n+1+2 \times (0-n) = 1 \in O(1)$

else, Stack B is not empty:

1 removal from the top of Stack B

$T_{actual} = 1$

$\Phi_{before} = n$, $\Phi_{after} = n$

$T_{amortized} = 1+2 \times (n-n) = 1 \in O(1)$

Dequeue has $O(1)$ amortized time

2. 3 Color Problem $\leq_p$ Fast Food Problem

Suppose given an instance of the 3 Color Problem, now we have:

G : the undirected, unweighted graph with vertices on it
color1 : the first color
color2 : the second color
color3 : the third color

Create an instance of the Fast Food Problem from the instance of the 3 Color Problem:

$k=3$; 3 different fast food chains in total
$f1 = color1$: use color1 to represent the first food
$f2 = color2$: use color2 to represent the second food
$f3 = color3$: use color3 to represent the third food
get the graph of this instance, Gf, from G:
each vertex in G is a town in Gf
if two vertices are adjacent in G, then the weight of the edge between them in Gf is d
if two vertices are not adjacent in G, then the weight of the edge between them in Gf is larger than d, we just use $d+1$ here

Now we have created an instance of the Fast Food Problem in polynominal time, given an instance of the 3 Color Problem

If my Fast Food Problem gives an optimal solution, to construct a solution for the 3 Color Problem:
for each vertex in Gf that has f1, coloring the same vertex in G with color 1
for each vertex in Gf that has f2, coloring the same vertex in G with color 2
for each vertex in Gf that has f3, coloring the same vertex in G with color 3
now we have constructed a solution for the 3 color problem in polynominal time
Since the vertices are the same in G and Gf, and we have 3 colors in 3 Color Problem and 3 foods in Fast Food Problem, now we have an optimal solution from 3 foods, the solution we get in 3 colors is also optimal

If my Fast Food Problem does not give an optimal solution, then there is not a solution for the 3 color Problem.
To prove this, use contrapositive:
If my 3 Color Problem gives an optimal solution, to construct a solution for the Fast Food Problem:
for each vertex in G that is in color 1, set f1 on the same vertex in Gf
for each vertex in G that is in color 2, set f2 on the same vertex in Gf
for each vertex in G that is in color 3, set f3 on the same vertex in Gf
now we have constructed a solution for the Fast Food Problem in polynominal time
Similarly, since vertices are the same in G and Gf, now we have an optimal solution from 3 colors, the solution we get for 3 foods is also optimal

Till now, we have proved that :
optimal solution in Fast Food Problem → optimal solution in 3 Color Problem

no solution in Fast Food Problem → no solution in 3 color Problem

Fast Food Problem is at least as difficult to solve as 3 Color Problem

Since 3 Color Problem is NP Complete

⇒ Fast Food Problem is NP Complete