Problem 1

Problem 2

Problem 3

Part (a)

Supposing I have an optimal solution to some minimum-cost circulation problem, I show how I can re-optimize the solution after changing one edge cost by one unit.

Before the change, from the optimal solution, I can calculate a set of feasible prices and residual reduced edge costs, which are all non-negative. After changing one edge cost by one unit, I have at most one negative reduced edge cost of -1 in the residual network. Suppose that the negative residual reduced edge cost is between vertex v and w and has capacity u. If this edge is involved in a negative reduced cycle, all the other edges in the cycle must have a residual reduced cost of 0. I push as much flow from v to w as I can (but $\leq u$) around the cycle, through the path from w to v with edges of residual reduced cost 0. I do this as follows. I create a graph with only edges of residual reduced cost 0, add a source s' and an edge of capacity u from s' to w, add a sink t' and an edge of capacity u from v to t'. I look for the max flow between s' and t' in the created graph. I match this max flow with flow from v to w and add it all to the total flow, causing its cost to decrease.

Notice that I haven't introduced any new negative residual reduced edge costs by pushing along the edges with residual reduced cost of 0. Now, if the edge from v to w is saturated, there are no more negative residual reduced edge costs, and the solution is optimal again. However, even if the edge from v to w is not saturated, the solution is optimal again. Indeed, I can change the set of prices to make them feasible again. I do this by incrementing the price of v and then incrementing all nodes which have a path of reduced cost 0 to v in the new residual network. Since there is no path of cost 0 from w to v (because, if there were, I would have pushed more flow along it), the price of w will not increase. So now, the edge between v and w will have residual reduced cost 0. In addition, I am not introducing any negative residual reduced edge costs, because if a residual reduced edge cost decreases then its edge is from a node whose price isn't incremented to a node whose price is incremented, which implies that the residual reduced edge cost must have been positive, so decrementing it won't cause it to become negative. Therefore, in any case, the flow is optimal again.

Part (b)

I deduce a cost-scaling algorithm for minimum-cost flow that makes $O(m \log C)$ calls to my solution of part (a). I start with all costs at 0 and look for a maximum flow. I have $O(\log C)$ scaling phases, in which I shift bits of costs, from highest to lowest. At the beginning of a phase, I double the costs of each edge, which doesn't change the optimal flow. In a phase, for each edge in turn, if the bit of the edge cost is set for this phase, I increment or decrement the cost, depending on its sign, and then call my algorithm of part (a).

Problem 4

I create a graph that looks like a bipartite matching graph between results (old and new) and available slots. It has more structure than a bipartite graph to enforce the constraints that all clicked old results and at least k new results must appear. In addition, matching edges have costs in order to maximize the value of the merged result list (derived from values v_i and penalty changes p_{ij} for old results and values v'_i for new results).

Here is how I create the graph G. Unless explicitly specified, the cost of an edge is 0 and its capacity 1.

- Add a source s and a sink t, a node s_c (for clicked), a node s_n (for new), a node s_r (for rest).
- Add an edge of capacity c (number of clicked old results) from s to s_c.
- Add an edge of capacity k from s to s_n.
- Add an edge of capacity $n - c - k$ from s to s_r.
- For each result i in the old result list:
  - Add a node o_i.
  - Add a node o'_i.
  - Add an edge from s_r to o_i.
  - Add an edge from o_i to o'_i.
- For each clicked result i in the old result list: add an edge from s_c to o_i.
- For each result i in the new result list:
  - Add a node n_i.
  - Add a node n'_i.
  - Add an edge from s_r to n_i.
  - Add an edge from n_i to n'_i.
  - Add an edge from s_n to n_i.
- For each available slot j in the merged result list:
  - Add a node s_i.
  - Add an edge from s_i to t.
- For each pair of old result i and available slot j: add an edge of cost $p_{ij} - v_i$ from o'_i to s_i.
- For each pair of new result i and available slot j: add an edge of cost $-v'_i$ from n'_i to s_i.

Now, I look for a min-cost max-flow in G from s to t. I simply look at the edges with positive flow between result nodes (o'_i's and n'_i's) and slot nodes (s_i's) in order to construct the best merged result list.

Note: a simple $O(n \log n)$ greedy algorithm

It is possible to solve the problem using a greedy approach. Hold onto the c clicked items in the old result list. Sort the new result list by value, and hold onto the k most valued new items. Merge the remaining old result list (without the c clicked items) and the remaining new result list (without the k most valued items) and sort the merged list. Hold onto the $n - k - c$ most valued remaining items in the merged list. Now, put each old item, which was hold onto, into the best merged result list at the same position as its position in the old result list. Put the new items, which were hold onto, into the remaining positions in the merged result list. Voilà!

Problem 5

I consider the following linear programming problem: minimize cx s.t. $x_1+x_2 \geq 1$, $x_1 + 2x_2 \leq 3$, $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$. For each of the following objectives c, I give the optimum value and the set of optimum solutions.

(a) $c = (-1, 0, 0)$ The optimium value is -3 and the set of optimum solutions is $x_1 = 3$, $x_2 = 0$, $x_3 \in [0, +\infty)$.

(b) $c = (0, 1, 0)$ The optimium value is 0 and the set of optimum solutions is $x_1 \in [1, 3]$, $x_2 = 0$, $x_3 \in [0, +\infty)$.

(c) $c = (0, 0, -1)$ The optimum value is $-\infty$ and the set of optimum solutions is $x_1 \in [0, 3]$, $x_2 \in [\max(0, 1 - x_1), \frac{3-x_1}{2}]$, $x_3 = +\infty$.

Problem 6

Part (a)

I represent this problem as a directed graph, where each currency x is a node, each order i is an edge from node a_i to node b_i with capacity u_i, rate r_i and flow f_i (the flow values f_i will be our linear programming variables).

My constraints are as follows:

- For each order i:
$$0 \le f_i \le u_i$$
- For each currency x except Dollar:
$$\sum_{\forall i,\text{ s.t. } a_i = x} f_i \le \sum_{\forall i,\text{ s.t. } b_i = x} f_i r_i$$
- For the currency Dollar (d):
$$\sum_{\forall i,\text{ s.t. } a_i = d} f_i \le \sum_{\forall i,\text{ s.t. } b_i = d} f_i r_i + D$$

My objective is to maximize the amount in the currency Yen (y):

$$-\sum_{\forall i,\text{ s.t. } a_i = y} f_i + \sum_{\forall i,\text{ s.t. } b_i = y} f_i r_i$$

The second and third constraints guarantee that we're never "creating" money, and that all money starts at the node of the currency Dollar.

Part (b)

Because $\prod r_i < 1$ for any cycle, cycles just waste money and so I can get rid of them (see part (c)). Once I have a solution with no cycle, I can order the edges topologically by the flow, and process each edge at most once. Since each edge represents an order, I will be processing each order at most once. Because of the topological ordering, I will have enough money in the necessary currency to respond to an order at once when I process it.

Part (c)

By running the linear program of part (a), I will get the optimal amount Y in the currency Yen (y). I add this optimum as a new constraint to the linear program from part (a):

$$-\sum_{\forall i,\text{ s.t. } a_i = y} f_i + \sum_{\forall i,\text{ s.t. } b_i = y} f_i r_i = Y$$

My objective is now to minimize the amount in the currency Dollar (d) that I spend:

$$\sum_{\forall i,\text{ s.t. } a_i=d} f_i - \sum_{\forall i,\text{ s.t. } b_i=d} f_i r_i$$

By minimizing the amount of dollars that I spend, I am sure that I won't be wasting money in cycles. In addition, I won't convert money into another currency unless it will eventually be converted into yens, since, otherwise, I am better off keeping my dollars. Therefore, the new linear program above guarantees that I will have the optimum amount of Yen and no other currency except dollars.