Freestyle Engineering Challenge

Justin Wei

Assumptions:

In creating a solution to this challenge, I have made the assumptions that...

1. This is a dinner party, therefore:
    a. Food is more important than drinks. One unit of food utility has more value than one unit of drinks utility: $\text{Utility}_{\text{food}} > \text{Utility}_{\text{drink}}$.
        i. More specifically, I define $\text{Utility}_{\text{food}} = 2 * \text{Utility}_{\text{drink}}$.
    b. Optimizing for one entree and one drink per person is optimal.
2. Utility functions are the same across all guests (not realistic, but a necessary simplification).
3. The "unit cost" of a food or drink refers to the cost of buying the item for a single person.
4. Drink and food files are listed in order of preference of the guest
    a. These entries probably came from the guests themselves, therefore it is natural to assume that they listed their drink and food preferences in order of decreasing utility.
        i. $\text{Utility}_{\text{pref1}} > \text{Utility}_{\text{pref2}} > \text{Utility}_{\text{pref3}} \dots > \text{Utility}_{\text{pref(N)}}$
5. Assumptions about the utility functions of drinks and food...
    a. $\text{Utility}_{\text{prefItem}} = 1 - (\text{preferenceRank}_{\text{item}} / \# \text{ of preferences})$
        i. i.e. if preferences are [rice, pasta, chicken, salad], $\text{preferenceRank}_{\text{rice}}=0$, $\text{preferenceRank}_{\text{salad}}=3$, and `# of preferences=4`. Note that a guest's first preference maximizes their utility ($\text{Utility}_{\text{firstPref}}=1$)
    b. The utility difference from preference 1 to preference 2 is inversely proportional to the number of preferences:
        i. $\Delta\text{Utility}_{\text{pref1} \rightarrow \text{pref2}} = 1 / (\# \text{ of preferences})$, and utility difference between two consecutive preferences is independent of what the preferenceRanks are (i.e. $\Delta\text{Utility}_{\text{pref1} \rightarrow \text{pref2}} = \Delta\text{Utility}_{\text{pref2} \rightarrow \text{pref3}} = \dots = \Delta\text{Utility}_{\text{pref(N-1)} \rightarrow \text{pref(N)}}$).
    c. Drinks and food add utility independently (practically, this means there's no such thing as "pairings" of food and drinks), and utilities are weighted to account for the fact that twice as much utility is derived from food as is derived from drinks:
        i. $\text{Utility}_{\text{Total}} = (⅓) * \text{Utility}_{\text{Drink}} + (⅔) * \text{Utility}_{\text{Food}}$
        ii. $\text{Utility}_{\text{Max}} = 1$
6. A reasonable budget is given for the event, such that there is enough money for everyone to have their cheapest preferences of food and drink.
7. Generally, input files follow the format specified in challenge description (I do account for whitespace errors).

Overall Algorithm:
Any allocation of budget to multiple people requires that we try to maximize fairness between how much money is spent on each person. Therefore, we begin by allocating the same amount of money to each person ($\texttt{Budget}_{\texttt{single-guest}} = \texttt{Budget}_{\texttt{total}}$ `/ # of guests`).

Then, we calculate each guest's optimal preference combination (drink, food) independently, using their allotted budget. This calculation is done with the following algorithm and considerations:

Algorithm for determining (drink, food) combination for a single guest:
With the assumptions we've made, the bulk of the problem is an optimization one over two fields (drinks and food), where both utility and price of items are considerations.

One way to calculate the optimal combination for a guest is by calculating the utility of every possible (drink, food) combination for a guest (using the equation $\texttt{Utility}_{\texttt{Total}} = (\frac{1}{3}) * \texttt{Utility}_{\texttt{Drink}} + (\frac{2}{3}) * \texttt{Utility}_{\texttt{Food}}$), eliminating the pairings that the guest can't afford, and finding the pair with the maximum utility. The runtime of this algorithm would be `O(d*f + (d+f))` → `O(d*f),` where `d = # drink preferences, and f = # of food preferences`. This solution would suffice for the problem at hand, since the size of the party, number of drink preferences, and number of food preferences is realistically limited to a reasonable number, where runtime would not really be a consideration. However, if we want a solution that is scalable to similar, much larger problems, we need to improve the runtime of this algorithm.

The bottleneck here is the need to create pairings of every possible (drink, food) combination of a guest. Is there a way we can make this process more efficient by not having to consider all combinations of (drink, food)? Perhaps we should consider combining the items into a single list and traversing it while keeping track of the overall utility. After all, drink utility can be expressed as weighted food utility, thus putting them into same list is a natural idea. As we traverse the list of drinks and foods, if the new item (either a drink or food) we are considering would raise the overall guest's utility, then we replace either the guest's current best drink or food with the new item. The issue with this algorithm, however, is that we won't know when a trade-off is worth it. For example, if the food item we are currently considering raises the overall utility of the customer, how do we know it's worth purchasing as opposed to saving the money and purchasing a drink later in the list. This issue leads us to consider a sorted list that can help indicate which items to purchase. An algorithm that uses sorting and then traverses the list only once would result in a runtime of `O(nlog(n) + n)` → `O(nlog(n)),` where `n = d+f ≈ d` (because `d ≈ f` when both are very large). The tricky part of sorting is that we cannot only sort by utility, though, as we need to consider the price tradeoff, and vice versa. We need to sort using a *relationship* between utility and price.

In order to find the maximal utility given a certain amount of money (and to achieve the desired runtime), we need to consider both utility and price of items in a single pass of the list. The way we can solve this issue is by using a value that relates both utility and price: `Happiness per Dollar (HPD) = Utility`$_{\texttt{Item}}$` / Price`$_{\texttt{Item}}$. If we sort our list according to HPD, we then have a list that is sorted with both the utility and price of an item in mind simultaneously.

By sorting the list by HPD in descending order, we solve the issue we had earlier when we were unable to determine whether we should hold off on purchasing an item because it "might not be worth it." Now, the best item to purchase will always be the first item in the list. However, just choosing the item with the maximum HPD does not necessarily maximize overall happiness. For example, if the list is [rice (10), pasta (9), chicken (3), salad (0.01)], with HPD shown in parentheses, then salad has the highest HPD (because of its cheap cost). But consider if the budget was $100.00 per person, then our optimized preference for the guest should still be rice, despite not having the highest HPD (for simplification purposes, this is only optimizing for food preference).

So, the algorithm also needs to account for situations where the maximum HPD might not necessarily mean best total utility. Solving this issue is simple: we just traverse the list from highest HPD to lowest HPD, checking to make sure we can afford the next item, and if so, would it increase our overall utility? Doing so, we know that the new item we are currently considering will have a better "value" than any of the items later on (so there is no concern about "saving money" for a later item), so all we have to focus on is the strict utility gained from a new drink/food item.

When we are done traversing the list, we will have the (drink, food combination) that best optimizes utility while accounting for value (HPD).

Overall Algorithm Conclusion:

Now that we have an algorithm that calculates the optimal preferences of a single guest, we can repeat this process for every guest (again, assuming that all guests can afford at least their cheapest drink and food options. Once all guests have received their optimal preferences within budget, in addition to outputting the counts of each food/drink that the organizer must order, "orders.txt" also returns the amount of leftover money, the specific orders of each guest, the happiness rating of each guest, and the money each guest spent.