**Задача «Автобусы»**

В городке Урюполе только один автобусный маршрут, соединяющий вокзал с главной местной достопримечательностью — продуктовым рынком, славящимся на всю округу большим ассортиментом и низкими ценами.

В Урюполь недавно пришел поезд из соседнего городка Крыжопинска, и на автобусной остановке возле вокзала образовалась очередь из *N* человек, желающих попасть на рынок.

В связи с этим, для развозки пассажиров к остановке собираются подать *M* автобусов вместимостью *D* каждый. Известно, что если пронумеровать людей от 1 до *N* в порядке очереди, то *i*-й из них при посадке в автобус займет *L(i)* единиц объема.

Однако автобус — не единственный транспорт в Урюполе: если человек устал ждать в очереди, он может выйти из очереди, сесть на такси и тут же уехать. При этом относительный порядок оставшихся в очереди людей не меняется.

Посадка в автобусы происходит следующим образом. Автобус подъезжает к остановке, открывает переднюю дверь, и в нее заходят люди в порядке очереди. Как только для очередного человека не хватает места, автобус закрывает дверь и уезжает, после чего к остановке подходит следующий автобус (если он есть).

Поскольку зарплата водителя автобуса зависит от количества перевезенных пассажиров, водители хотят знать, какое наибольшее суммарное количество людей из очереди они могут перевезти. Помогите им.

Формат ввода

Первая строка содержит число *M* (≤ 100).

Вторая строка — *D* (≤ 300).

Третья строка — *N* (≤ 300).

Четвёртая строка — *L(1)* *L(2)* … *L(N)*. Все *L(i) ≤ D*.

Все входные параметры — натуральные числа.

Формат вывода

Необходимо вывести единственное число — искомое количество людей.

**Решение.**

1. Создадим массивы isVisited и partialAnswer, в которых будем хранить флаг, посчитан ли ответ для подзадачи, и, если посчитан, то сам ответ соответственно, а так же массив peopleVolume, где будем хранить объемы людей в очереди.
2. Считаем входные данные, а именно – количество автобусов (busNumber), вместимость автобусов (busCapacity), количество людей в очереди (peopleNumber) и объем каждого человека (peopleVolume).
3. Вызовем функцию findMaximumProfit() с исходными параметрами 0,0,0, которая найдет ответ на задачу, т.е. максимальное количество пассажиров, которое могут увести автобусы.
4. Выведем полученный ответ.

Функция findMaximumProfit() рекурсивно находит ответ для подзадачи, характеризуемой следующими параметрами:

* currentBus – номер автобуса (начиная с 0), который стоит в данный момент на остановке;
* currentPassenger – номер пассажира (начиная с 0), который стоит в данный момент в начале очереди;
* busySpace – сколько объема уже занято в текущем автобусе currentBus.

Рассмотрим подробнее работу этой функции:

1. Проверяем, закончились ли автобусы (currentBus == busNumber) или люди в очереди (currentPassenger == peopleNumber). Если да, то возвращаем 0, т.к. мы не можем никого посадить в несуществующий автобус, а так же не можем никого увезти, если у нас не осталось в очереди людей.
2. Проверяем, посчитали ли мы раньше ответ для текущей подзадачи. Если посчитали, то просто возвращаем его.
3. В зависимости от текущей подзадачи мы можем перейти к следующим:
   1. Текущий пассажир уходит из очереди. Тогда номер автобуса и объём занятого в нем места останется прежним, а номер пассажира увеличится. Отразим это вызовом функции от следующей подзадачи: findMaximumProfit(currentBus, currentPassenger + 1, busySpace)
   2. Если в автобусе достаточно свободного места для текущего пассажира, то он садится в этот автобус. Тогда номер автобуса остается прежним, номер текущего пассажира увеличивается на 1, а объем занятого места в текущем автобусе увеличивается на объем посаженного в него пассажира. Количество отвезенных пассажиров увеличивается при этом на единицу. Отразим это вызовом функции от следующей подзадачи:

findMaximumProfit(currentBus, currentPassenger + 1, busySpace + peopleVolume[currentPassenger])

* 1. Если же места для текущего пассажира недостаточно, то текущий автобус отъезжает и подъезжает следующий. То есть номер пассажира остается прежним, номер автобуса увеличивается на 1, а объём занятого места обнуляется, т.к. новый автобус подъезжает пустым. Отразим это вызовом функции от следующей подзадачи: findMaximumProfit(currentBus + 1, currentPassenger, 0)

1. Используя значения, возвращенные рекурсивными вызовами подзадач, определяем максимальное количество человек, которое мы можем увезти, опираясь на текущие параметры. Т.е. вычисляем max(a, b + 1) или max(a, c) в зависимости от текущих параметров. Значения a, b и c подставлены вместо вызовов функции из соответствующих пунктов выше.
2. Сохраняем полученный ответ и отмечаем флаг того, что ответ для текущей подзадачи посчитан. Возвращаем ответ.

Корректность алгоритма очевидна, за исключением одного момента – мы получаем ответ на задачу, используя оптимальные ответы на подзадачах. Неочевидным является факт, что такой подход приведет нас к оптимальному ответу на всей задаче. Подзадача характеризуется 3мя параметрами – номером пассажира в начале очереди, номером автобуса, стоящего на остановке, и количеством занятого места в этом автобусе. Ответ на подзадачу – максимальное количество людей из оставшейся очереди, которое можно посадить в текущий и последующие автобусы. Докажем, что для достижения оптимального ответа на задачу это количество должно быть максимальным:

Воспользуемся доказательством от противного.

Пусть, максимальный ответ на текущую подзадачу – k человек. Предположим, что выгодней взять p человек, где p < k. Тогда ответ на всю задачу можно разбить на 2 части:

* количество людей, которые уехали, начиная с текущей подзадачи и до конца очереди, т.е. p человек.
* количество людей, которые уехали до текущей подзадачи, пусть их будет r человек.

Тогда ответом на всю задачу будет (r + p) человек, т.е. все уехавшие на автобусах люди. Однако, если бы мы для подзадачи взяли k человек, то на всю задачу получили бы ответ (r + k) человек, а т.к. p < k, то и (r + p) < (r + k). Следовательно, ответ (r + p) не является оптимальным и наше предположение не верно. Значит, при расчете ответа на задачу нам всегда выгодно брать максимальный ответ на подзадаче, что и требовалось доказать.

Сначала может показаться, что время работы алгоритма экспоненциальное, так как мы на каждом шаге рекурсивно вызываем еще два, однако, благодаря меморизации, мы фактически заполняем трехмерный массив partialAnswer, причем высчитываем каждую его ячейку не более одного раза. Поэтому итоговая временная сложность алгоритма составляет O(busNumber × busCapacity × peopleNumber) или O(M×N×D) в терминологии условия задачи.

Затраты по памяти так же определяются размерами массивов partialAnswer, isVisited и peopleVolume, то есть O(M×N×D + M×N×D + N), что эквивалентно O(M×N×D).