1. Задача о размене.
2. Непрерывный рюкзак.
3. Задача о выборе заявок.
4. (*) Разбор задачи с олимпиады
5. Задача про укладывание спать гномиков

1. Размен монет

Монетная система некоторого государства состоит из монет достоинством $a_1 = 1 < a_2 < ... < a_n$. Требуется выдать сумму S наименьшим возможным количеством монет.

Жадный алгоритм решения этой задачи таков. Берётся наибольшее возможное количество монет достоинства a_n: $x_n = \lfloor S/a_n \rfloor$. Таким же образом получаем, сколько нужно монет меньшего номинала, и т. д.
Для данной задачи жадный алгоритм не всегда даёт оптимальное решение, а только для некоторых, называемых каноническими, монетных систем, вроде используемых в США (1, 5, 10, 25 центов).
Неканонические системы таким свойством не обладают. Так, например, сумму в 24 (11) копейки монетами в 1, 5 и 7 коп. жадный алгоритм разменивает так: 7 коп. — 3 шт., 1 коп. — 3 шт., в то время как правильное решение — 7 коп. — 2 шт., 5 коп. — 2 шт.

Одним из достаточных условий каноничности можно назвать удваивание достоинства как минимум в 2 раза по сравнению с предыдущим.

Доказательство.
Допустим не выгодно взять хотя бы одну из монет максимального номинала. Тогда их количество уменьшается, а количество каких-то других (меньших) монет увеличивается. Однако чтобы набрать ту же сумму, что и макс. номинал, понадобится как минимум две монеты, что больше чем одна.

2. Задача о рюкзаке

Пусть у тебя есть набор предметов из n предметов, у каждого из них есть цена c[i] и целый вес w[i] килограмм. Но у тебя есть только рюкзак который вмещает m килограмм. Придумай алгоритм, который решает, какие предметы нужно брать, чтобы унести набор предметов с максимальной стоимостью.

Непрерывный рюкзак. Можешь подумать, а что делать если предметы можно делить. То есть у тебя вместо предметов “золотая каша”. В смысле, в рюкзак

ты можешь класть не обязательно весь предмет целиком, а только его часть. (Отложить часть каши.)
Берем кашу с максимальной плотностью (c[i]/w[i]) и берем максимальное возможное количество такой каши.
Доказательство. Очевидно. Если есть вариант лучше, заменим часть каши на более плотную и получим больше денег.

3. Выбор заявок

Формулировка № 1. Даны n заявок на проведение занятий в некоторой аудитории. В каждой заявке указаны начало и конец занятия (s_i и f_i для i-й заявки). В случае пересечения заявок можно удовлетворить лишь одну из них. Заявки с номерами i и j совместны, если интервалы $[s_i, f_i)$ и $[s_j, f_j)$ не пересекаются (то есть $f_i \leq s_j$ или $f_j \leq s_i$). Задача о выборе заявок состоит в том, чтобы набрать максимальное количество совместных друг с другом заявок.
Формулировка № 2. На конференции, чтобы отвести больше времени на неформальное общение, различные секции разнесли по разным аудиториям. Учёный с чрезвычайно широкими интересами хочет посетить несколько докладов, проходящих в разных секциях. Известно начало s_i и конец f_i каждого доклада. Определить, какое максимальное количество докладов можно посетить.
Приведём жадный алгоритм, решающий данную задачу. При этом полагаем, что заявки упорядочены в порядке возрастания времени окончания. Если это не так, то можно отсортировать их за время $O(n \log n)$; заявки с одинаковым временем конца располагаем в произвольном порядке.
Activity-Selector(s,f)
1. $n \leftarrow$ length[s]
2. $A \leftarrow \{1\}$
3. $j \leftarrow 1$
4. for $i \leftarrow 2$ to n do
5. if $s_i \geq f_j$ then
6. $A \leftarrow A \cup \{i\}$
7. $j \leftarrow i$
8. return A

На вход данному алгоритму подаются массивы начала и окончания занятий. Множество A состоит из номеров выбранных заявок, а j — номер последней заявки. Жадный алгоритм ищет заявку, начинающуюся не ранее окончания j-той, затем найденную заявку включает в A, а j присваивает её номер. Таким

образом, каждый раз мы выбираем то (ещё не начавшееся) занятие, до конца которого осталось меньше всего времени.

Алгоритм работает за $O(n \log n + n)$, то есть сортировка плюс выборка. На каждом шаге выбирается наилучшее решение. Покажем, что в итоге получится оптимум.

Доказательство. Заметим, что все заявки отсортированы по неубыванию времени окончания. Заявка номер 1, очевидно, входит в оптимум (если нет, то заменим самую раннюю заявку в оптимуме на неё, от этого хуже не станет). Выкинув все заявки, противоречащие первой, получим исходную задачу с меньшим количеством заявок. Рассуждая по индукции, аналогичным образом приходим к оптимальному решению.

```
s[i] f[i]
d[i] = [i for i in range(n)]
d.sort(key=lambda x: f[x])
count = 0
j = 0
for i in range(1, n):
  if s[d[i]] >= f[d[j]]:
    j = i
    count += 1
print(count)
```

5. Гномики

Гномики укладываются спать. Чтобы уложить i-го гномика, требуется u[i] минут. После этого он спит sl[i] минут. Можно ли уложить всех гномиков спать?

Решение. Будем укладывать гномиков в убывающем порядке суммы sl[i] + u[i].

Доказательство. Пусть жадный порядок - 1, 2, …, n. И пусть существует какое-либо другой порядок, в котором гномиков можно уложить спать. Рассмотрим ситуацию, когда в этом новом порядке поменяны местами только гномы 1 и 2. Если в жадности другой порядок, значит sl[1] + u[1] >= sl[2] + u[2] (1). Тогда для новой ситуации верны два неравенства (что следует из возможности их уложить):

sl[2] >= u[1] + u[3] + … + u[n] (2)

sl[1] >= u[3] + u[4] + … + u[n] (3)

Преобразуем:

sl[2] + u[2] >= u[1] + u[2] + … + u[n] (2*)

sl[1] + u[1] >= u[1] + u[3] + … + u[n] (3*)

Из (1) и (2*) следует:

sl[1] + u[1] >= sl[2] + u[2] >= u[1] + u[2] + … + u[n]

Так как u[i] неотрицательны, из правой части (2*) можно выкинуть элемент, и неравенство не изменится:
sl[2] + u[2] >= u[2] + u[3] + … + u[n]
Преобразуем:
sl[1] >= u[2] + u[3] + … + u[n]
sl[2] >= u[3] + u[4] + … + u[n]
Эти неравенства означают как раз то, что если гномиков можно уложить в порядке 2, 1, 3, …, n, то их можно уложить и в порядке 1, 2, …, n. То есть, жадный алгоритм тоже работает.
Теперь мы умеем менять местами двух соседних гномиков, а значит можем попарно менять местами гномиков, для какого бы порядке ни существовало бы решение.