Общая схема рекурсивного перебора с возвратом такая:

```
def Перебор(Ситуация):
    if Ситуация конечная:
        Завершающая обработка
    else:
        for Действие in Множество всех возможных действий:
            Применить Действие к Ситуация
            Перебор(Ситуация)
            откатить Действие назад
```

В общем случае, алгоритм перебора с возвратом содержит тест, который по данной подзадаче быстро выдаёт один из трёх ответов:
неудача: подзадача не имеет решения; успех: найдено решение подзадачи; неопределённость.
Например в задаче о мирных ферзях неудача — новый ферзь бьёт одного из предыдущих, успех — ферзь успешно поставлен на последнюю линию, неопределённость — новый ферзь не бьёт ни одного из старых.
Нерекурсивный общий вид алгоритма перебора обычно удобно организовывать при помощи очереди задач:

```
начать с некоторой задачи P[0]
S = {P[0]} # {очередь активных подзадач}
пока S не пусто:
    выбрать подзадачу P из S и удалить её из S
    разбить P на меньшие подзадачи P[1], P[2], ..., P[k] (или сделать k возможных следующих шагов)
    для каждой P[i]:
        если тест(P[i]) = успех:
            вернуть найденное решение
        если тест(P[i]) = неудача:
            отбросить P[i]
        иначе:
            добавить P[i] в S
сообщить, что решения нет
```

При удачном подборе и реализации процедур `тест`, `выбрать` и `разбить` перебор с возвратами может быть вполне практичным.

Метод ветвей и границ

Этот метод может быть использован не только для задач поиска, но и для задач оптимизации. Допустим, мы решаем задачу минимизации (для задачи максимизации всё аналогично). Как и раньше, мы рассматриваем частичные решения. Нам необходимо понять, какие из них заведомо не приведут к оптимальному решению, чтобы их отбросить и сэкономить время. Отбросить подзадачу можно, если мы знаем, что на этом пути получится решение, худшее уже найденного в другой ветви. Но откуда мы это можем узнать? Для этого надо использовать нижнюю оценку стоимости решения.

```
начать с некоторой задачи P[0]
S = {P[0]} # {множество активных подзадач}
пока S не пусто:
    выбрать подзадачу (частичное решение) P из S и удалить её из S
    разбить P на меньшие подзадачи P[1], P[2], ... , P[k]
    для каждой P[i]:
        если P[i] является полным решением:
            обновить текущий_минимум с учётом P[i]
        иначе если нижняя_оценка(P[i]) < текущий_минимум:
            добавить P[i] в S
вернуть текущий_минимум
```

A. *Расстановки ферзей*

Известно, что на шахматной доске размером 8×8 можно расставить 8 ферзей не бьющих друг друга, причём сделать это можно 92 способами.

Дано натуральное $N \leqslant 12$. Определите сколькими способами на доске $N \times N$ можно расставить N мирных ферзей.

В этой задаче надо придумать способ проверки клетки на «занятость» за константное время (не зависящее от размера доски).

Input	Output
8	92

B. *Расстановки ферзей – 2*

Решите предыдущую задачу при ограничении $n \leqslant 10$, если расстановки ферзей, которые можно получить друг из друга поворотами и отражениями доски считать за одно.

Input	Output
8	12

C. *Грузоподъёмность*

Грузоподъёмность машины M килограмм. Есть N ящиков с грузами, масса i-го ящика равна m_i.

Определите, какую наибольшую массу грузов можно увезти на автомашине.

Программа получает на вход действительное число M, затем количество ящиков $N \leqslant 20$, затем N действительных чисел: массы ящиков.

Программа должна вывести единственное число: максимальную массу, которую можно увезти на машине.

Input	Output
10 4 1 3 5 8	9.0
1073.15936137 5 359.840622077 640.476657457 63.7703847126 345.949354785 635.448660233	1064.0876642465998

D. *Грузоподъёмность – 2*

Решите предыдущую задачу в ограничениях $N \leqslant 23$. Используйте следующие оптимизации:

(a) Отсортировать массы грузов по убыванию. Перебор начинать с более тяжелых грузов, сначала рассматриваем вариант, когда очередной груз берется.

(b) Делается отсечение, если суммарная масса всех взятых грузов превышает вместимость машины.

(c) Делается отсечение, если суммарная масса всех взятых грузов и всех оставшихся грузов меньше наилучшего уже найденного результата (т.е. если заведомо не удастся улучшить результат).

(d) Для быстрого определения суммарной массы всех оставшихся грузов используется предподсчёт.

E. *Задача коммивояжёра*

На плоскости даны N точек. Соедините их замкнутой ломаной линией минимальной длины.

Программа получает на вход число $N \leqslant 10$. Далее по паре в строке N пар действительных чисел: координаты точек.

Выведите одно действительное число: минимальную длину замкнутой ломаной, проходящей через все эти точки. Ответ выводите с точностью в 15 знаков.

Input	Output
4 0 0 1 0 1 1 2 1	4.82842712474619

F. *Задача коммивояжёра – 2*

Решите задачу коммивояжера в ограничениях $N \leqslant 11$. Используйте следующую оптимизацию: Делать отсечение, если суммарная длина текущего маршрута больше или равна наилучшему известному замкнутому маршруту (т.е. заведомо не удастся улучшить результат).

G. *Задача коммивояжёра – 3*

Решите задачу коммивояжера в ограничениях $N \leqslant 12$. Используйте следующую оптимизацию: Будем для каждой точки перебирать вершины не по порядку номеров, а по возрастанию расстояний от данной точки, то есть сначала будут рассматриваться пути в ближайшие точки. В этом случае довольно быстро будет найден короткий цикл и за счет отсечения, реализованного в предыдущей задаче, решение будет работать быстрее.

H. *Равенство*

Дано выражение:

$$a_1?a_2?\ldots?a_n = S$$

Посчитайте, сколькими способами в этом выражении можно заменить вопросительные знаки на знаки операций сложения и умножения так, чтобы выполнялось равенство.

Программа получает на вход число N и S. Далее записано N натуральных чисел a_i.

Выведите одно число — количество расстановок знаков сложения и умножения, дающих верное равенство.

Input	Output
2 4 2 2	2

I. *Монеты*

В стране используются монеты достоинством $A_1, A_2, \ldots, A_M$. Покупатель пришёл в магазин и обнаружил, что у него есть ровно по две монеты каждого достоинства. Ему нужно заплатить сумму N. Напишите программу, определяющую, сможет ли он расплатиться без сдачи.

Сначала вводится число N ($1 \leqslant N \leqslant 10^9$), затем число M ($1 \leqslant M \leqslant 15$) и далее M попарно различных чисел $A_1, A_2, \ldots, A_M$ ($1 \leqslant A_i \leqslant 10^9$).

Выведите сначала K — количество монет, которое придется отдать покупателю, если он сможет заплатить указанную сумму без сдачи. Далее выведите K чисел, задающих достоинства монет. Если решений несколько, выведите вариант, в котором покупатель отдаст наименьшее возможное количество монет. Если таких вариантов несколько, выведите любой из них.

Если без сдачи не обойтись, то выведите одно число 0. Если же у покупателя не хватит денег, чтобы заплатить указанную сумму, выведите одно число -1 (минус один).

Input	Output
5 2 1 2	3 2 2 1
7 2 1 2	-1
5 2 3 4	0

Использование эвристик для оптимизации перебора

Если уж дело дошло до полного перебора, то использование эвристик позволяет ускорить его в некоторых случаях на порядки. Представим себе, что мы можем численно оценить «перспективность» конкретного хода или даже всей позиции целиком. В этом случае позиции для следующего шага нужно перебирать в порядке убывания «качества» этого шага или «качества» получаемой позиции. Если есть возможность оценить качество шага, то удобно использовать перебор с возвратом в виде

```
def Перебор(Ситуация):
    if Ситуация конечная:
        Завершающая обработка
    else:
        for Действие in Множество_всех_возможных_действий,_упорядоченное_по_убыванию_перспективности:
            Применить Действие к Ситуация
            Перебор(Ситуация)
            откатить Действие назад
```

Если есть возможность оценить «перспективность» позиции целиком, то может оказаться удобно использовать очередь заданий. Соберём все возможные пары (перспективность, задание) в список и отсортируем его. Каждый раз будем удалять из списка самое перспективное задание. Варианты продолжения будем класть назад, поддерживая сортировку (например, используя бинпоиск или `bisect.insort`, или даже кучу). Итоговый псевдокод получается таким:

```
задания = [(перспективность_начального_условия, начальное_условие)]
while задания:
    текущее = задания.pop()
    разбить текущее на меньшие подзадачи P[1], P[2], ... , P[k] (или перебрать все возможные продолжения)
    для каждой P[i]:
        если тест(P[i]) = успех:
            вернуть найденное решение
        если тест(P[i]) = неудача:
            отбросить P[i]
        иначе:
            добавить пару (перспективность_P[i], P[i]) в задания
сообщить, что решения нет
```

Самое сложное — это найти подходящую эвристическую оценку перспективности хода или позиции. Например, в задаче D (Грузоподъёмность – 2) для оценки хода использовалась масса груза: перебор начинался с более тяжёлого. В задаче G (Задача коммивояжера – 3) для оценки хода использовалась расстояние до следующей точки: перебор начинался с ближайшей.
Если мы пытаемся пройти через все вершины в графе, то полезно начинать с вершин, из которой меньше всего выходов.
Если мы пытаемся найти короткий путь в графе, то в качестве оценки позиции можно использовать сумму пройденного пути и эвристической оценки оставшейся части пути.
Если мы пытаемся решить головоломку за минимальное число шагов, то в качестве оценки позиции можно использовать сумму числа уже выполненных ходов и эвристической оценки качества получаемой позиции. При этом для оценки качества позиции можно использовать число шагов для решения задачи по каким-нибудь другим более простым и наглым правилам.

J. *Игра Пятнашки*

Головоломка «игра в пятнашки» состоит из 15 квадратиков, уложенных в рамку размером 4×4, при этом одно место остается незанятым. За один ход можно передвинуть один квадратик, соседний с незанятым местом, в незанятое место.

Цель головоломки: упорядочить квадратики до такого положения:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Вам дано некоторое начальное положение головоломки. Известно, что её можно решить не более, чем за 20 ходов. Найдите решение этой головоломки.

Программа получает на вход 4 строки, содержащие по 4 числа: значения, записанные на квадратиках в начальном расположение. Число 0 соответствует пустой клетке.

Программа должна вывести последовательность перемещений, решающих головоломку в виде строки из букв `L`, `R`, `U`, `D` решающую головоломку, где буква `L` соответствует движению квадратика с числом влево, `R` — вправо, `U` — вверх, `D` — вниз.

Input	Output
1 2 3 4 5 6 7 8 9 10 15 11 13 14 0 12	DLU

K. *Игра Пятнашки – 2*

В условиях предыдущей задачи найдите и выведите кратчайшее решение (т.е. решение содержащее наименьшее число перемещений). Гарантируется, что головоломку всегда можно решить не более чем за 20 перемещений.

L. *Кабинет министров*

Премьер-министр некоторого государства хочет составить новый кабинет министров. К составу нового кабинета есть следующие пожелания:

- Министров должно быть как можно меньше
- Для каждой области государственной деятельности (строительство, финансы, внешняя политика и т.д.) должен быть хотя бы один министр, который в ней разбирается.

На рассмотрение Премьер-министра поступило N кандидатур. Определите, сколько и каких людей должны получить министерские посты, с учетом пожеланий.

Сначала вводится число N (натуральное, не превышает 10) — количество кандидатов в списке, затем вводится число K (натуральное, не превышает 20) — общее количество областей, в которых министры должны разбираться. Затем идет N строк следующего формата: в начале строки вводится число P_i (натуральное, не превышает K) — количество областей, в которых разбирается i-й кандидат, потом вводятся номера этих областей (натуральные числа, не превышают K).

Выведите количество министров, которое планируется назначить, исходя из требований задачи, затем перечислите номера подходящих кандидатов, в порядке возрастания. Если решений несколько, то выберите из них то, в котором участвуют кандидаты, идущие раньше по списку (то есть полученный список должен быть минимальным в лексикографическом порядке из всех возможных списков минимальной длины). Гарантируется, что решение существует (то есть в каждой области разбирается хотя бы один кандидат).

Input	Output
3 2 2 1 2 2 1 2 2 1 2	1 1
4 3 1 1 1 2 1 3 2 1 2	2 3 4

M. *Быстрые шахматы*

На шахматной доске размером 8×8 клеток расставлены фигуры. За один ход разрешается взять одной фигурой другую. Цвет фигур значения не имеет, ходы без взятия делать нельзя. Требуется найти последовательность ходов, после которой на доске останется одна фигура.

Ладья ходит по горизонтали или вертикали на любое число клеток. Слон ходит по диагонали на любое число клеток. Ферзь ходит по горизонтали, вертикали или диагонали на любое число клеток. Эти фигуры не могут перепрыгивать через другие фигуры. Конь ходит на две клетки по горизонтали или вертикали и на одну клетку в перпендикулярном направлении (например, на 2 клетки вверх и на одну клетку вправо и т.п.), при этом он может перепрыгивать через другие фигуры.

В восьми строках входных данных записаны по 8 символов, описывающих шахматную доску: `R` — ладья, `B` — слон, `K` — конь, `Q` — ферзь, точка обозначает пустую клетку. На доске изначально стоит не менее двух и не более десяти фигур.

Выведите возможную последовательность в следующем формате. Для каждого хода указывается сначала клетка, с которой делается ход, затем двоеточие, затем клетка, на которую делается ход. Клетка задается столбцом и строкой: столбцы нумеруются слева направо строчными латинскими буквами a, b, c, d, e, f, g, h; строки — снизу вверх цифрами $1, 2, 3, 4, 5, 6, 7, 8$.

Если решений несколько, приведите любое из них. Если решений нет, выведите `NO SOLUTION`.

Input	Output
.... .B...... ....K...	b6:e3
K......K .... .K...... ......Q. .... K......K	NO SOLUTION
..K..... ..K..... BR...... .B......	c7:a6 b6:a6 b5:a6 a6:c8

N. *Обход доски*

Дана шахматная доска размером $N \times M$. Необходимо построить обход всей доски ходом коня так, чтобы конь побывал во всех клетках доски ровно по одному разу.

Вы должны сдать на проверку текстовый файл, содержащий код вашей программы, строчку-разделитель из 20 символов #, а затем ответ на задачу: $N \cdot M$ строчек. Каждая строчка должна содержать координаты ровно одной клетки. Две соседние координаты должны быть связаны ходом коня и каждая из $N \cdot M$ клеток доски должна встречаться в этом файле ровно один раз. Каждая клетка записывается в виде «a1», где сначала записывается одна из первых N букв латинского алфавита, затем число от 1 до M. Например, для доски 4×5 сданный файл может быть таким:

```
код решения
####################
a1
c2
d4
b5
a3
b1
d2
c4
a5
b3
c1
a2
b4
d5
c3
d1
b2
a4
c5
d2
```

Теперь интересное: сдайте в тестовую систему текстовый файл с кодом вашей программы и путём коня на доске 20×20. Ваша программа может работать сколь угодно долго, хоть неделю. Хотя у задачи есть решение на питоне для любой доски в пределах 50×50, работающее меньше, чем за секунду.