**Задание.** Ваш друг не так успешен как вы и работает курьером. Он попросил вас помочь ему с его работой. Каждый день он получает заказы в разных частях города. Ему нужно получить оптимальный маршрут передвижения по клиентам. Более того, начальство требует от него указать приблизительное время доставки для каждого заказчика.

1. Для решения поставленной задачи опишем известную задачу коммивояжера, решение которой поможет получить ответ. Это одна из самых известных задач [комбинаторной оптимизации](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), заключающаяся в отыскании самого выгодного [маршрута](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)&action=edit&redlink=1), проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город только один раз — в таком случае выбор осуществляется среди [гамильтоновых циклов](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D0%BC%D0%B8%D0%BB%D1%8C%D1%82%D0%BE%D0%BD%D0%BE%D0%B2_%D1%86%D0%B8%D0%BA%D0%BB). Оптимизационная постановка задачи относится к классу очень NP-трудных задач. Задача коммивояжёра относится к числу [трансвычислительных](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0" \o "Трансвычислительная задача): уже при относительно небольшом числе городов (66 и более) она не может быть решена методом перебора вариантов никакими теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет.

Для возможности применения математического аппарата для решения проблемы, её следует представить в виде [математической модели](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C). Проблему коммивояжёра можно представить в виде модели на [графе](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), то есть, используя вершины и ребра между ними. Таким образом, вершины графа соответствуют городам, а рёбра{\displaystyle \left(i,j\right)} между вершинами{\displaystyle i} {\displaystyle j}- пути сообщения между этими городами. Каждому ребру {\displaystyle \left(i,j\right)}можно сопоставить критерий выгодности маршрута{\displaystyle c\_{ij}\geqslant 0}, который можно понимать как, например, расстояние между городами, время или стоимость поездки.

Рассмотрим алгоритмическую сложность задачи коммивояжера. Поскольку коммивояжёр в каждом из городов встает перед выбором следующего города из тех, что он ещё не посетил, существует (n - 1)!{\displaystyle (n-1)!} маршрутов для асимметричной и (n - 1)! / 2 {\displaystyle (n-1)!/2} маршрутов для симметричной задачи коммивояжёра. Таким образом, размер пространства поиска зависит [экспоненциально](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D1%81%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) от количества городов.

2. Опишем основные подходы к решению поставленной задачи. Для выбора алгоритма важно знать максимальное количество вершин и критерии оптимальности итогового решения.

Начнем с простейшего – жадный алгоритм. Это алгоритм, заключающийся в принятии локально [оптимальных решений](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5) на каждом этапе, допуская, что конечное решение также окажется оптимальным. Жадные алгоритмы не всегда приводят к оптимальным решениям, но во многих задачах дают нужный результат. Жадный метод обладает достаточной мощью и хорошо подходит для довольно широкого класса задач. Также к достоинству данного алгоритма можно отнести простоту реализации.

В качестве недостатка алгоритма можно отметить неприменимость к большому количеству вершин графа. В задачи коммивояжера, относящейся к классу NP, жадные алгоритмы не дают оптимального решения. Следовательно, данный метод не будет использоваться для решения поставленной задачи.

Рассмотрим второй метод для решения поставленной задачи – метод ветвей и границ. По сравнению с предыдущим методом, данный алгоритм сложнее в реализации. Можно найти точное решение задачи коммивояжёра, то есть «вручную» вычислить длины всех возможных маршрутов и выбрать маршрут с наименьшей длиной. Однако даже для небольшого количества городов решать задачу таким способом практически невозможно. Для простого варианта, симметричной задачи с{\displaystyle n} городами, существует (n - 1)! / 2 {\displaystyle (n-1)!/2}{\displaystyle (n-1)!/2}возможных маршрутов, то есть для 15 городов существует 43 миллиарда маршрутов и для 18 городов уже 177 триллионов. В этом и заключается недостаток данного метода. К достоинству метода ветвей и границ можно отнести быстроту реализации.

Рассмотрим третий метод для решения поставленной задачи – метод эластичной сети. В отличие от двух рассмотренных предыдущих методов, данный алгоритм применим к большому количеству вершин графа. По мере перебора путей строится оптимальный путь. Благодаря этому отбрасываются заведомо неоптимальные пути, уменьшая при этом количество проверок. Поэтому имеет смысл остановиться на этом методе для решения поставленной задачи. К недостаткам данного метода можно отнести сложность реализации. В 1987 году его использовали Дурбин и Уиллшоу, указавшие аналогию с механизмами установления упорядоченных нейронных связей.

3. Опишем выбранный метод – метод эластичной сети. Каждый из маршрутов коммивояжёра рассматривается как отображение окружности на плоскость (в каждый город на плоскости отображается некоторая точка этой окружности). Соседние точки на окружности должны отображаться в точки, по возможности ближайшие и на плоскости. Алгоритм следующий - начинается с установки на плоскость небольшой окружности. Она неравномерно расширяется, становясь кольцом, проходящим практически около всех городов и устанавливая таким образом искомый маршрут. На каждую движущуюся точку кольца оказывает действие две составляющие: перемещение точки в сторону ближайшего города и смещение в сторону соседей точки на кольце так, чтобы уменьшить его длину. Город в итоге связывается с определенным участком кольца по мере расширения. По мере расширения такой эластичной сети, каждый город оказывается ассоциирован с определенным участком кольца.

Вначале все города оказывают приблизительно одинаковое влияние на каждую точку маршрута. В последующем, большие расстояния становятся менее влиятельными и каждый город становится более специфичным для ближайших к нему точек кольца. Такое постепенное увеличение специфичности, которое напоминает метод обучения сети Кохонена, контролируется значением некоторого эффективного радиуса.

Представим алгоритм наглядно на схемах.

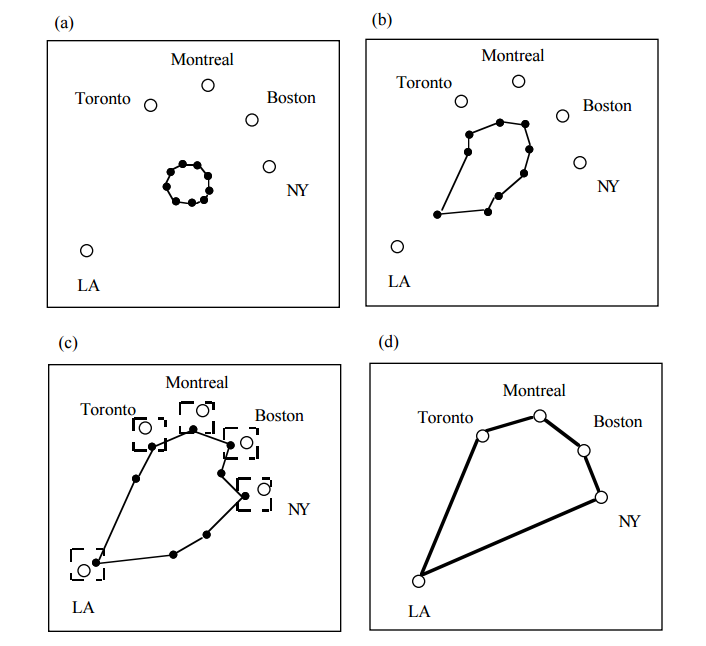


Рис.1 Постепенное изменение эластичной сети с течением времени (a) (b) (c) и конечная стадия (d).

Дурбин и Уиллшоу показали, что для задачи с 30 городами, рассмотренной Хопфилдом и Танком, метод эластичной сети генерирует наикратчайший маршрут примерно за 1000 итераций. Для 100 городов найденный этим методом маршрут лишь на 1 % превосходил оптимальный.

4. В данной задаче предполагается использование двусвязных списков, массивов, красно – черных деревьев.

Двусвязный список состоит из элементов данных, каждый из которых содержит ссылки как на следующий, так и на предыдущий элементы. Наличие двух ссылок вместо одной предоставляет несколько преимуществ. Вероятно, наиболее важное из них состоит в том, что перемещение по списку возможно в обоих направлениях. Это упрощает работу со списком, в частности, вставку и удаление. Помимо этого, пользователь может просматривать список в любом направлении. Еще одно преимущество имеет значение только при некоторых сбоях. Поскольку весь список можно пройти не только по прямым, но и по обратным ссылкам, то в случае, если какая-то из ссылок станет неверной, целостность списка можно восстановить по другой ссылке.

Массив - [тип](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) или [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) в виде набора компонентов (элементов массива), расположенных в памяти непосредственно друг за другом. При этом доступ к отдельным элементам массива осуществляется с помощью [индексации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), то есть через ссылку на массив с указанием номера (*индекса*) нужного элемента. За счёт этого, в отличие от [списка](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), массив является структурой данных, пригодной для осуществления произвольного доступа к её ячейкам.

Красно-чёрное дерево - это одно из самобалансирующихся [двоичных деревьев поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0), гарантирующих [логарифмический](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B0%D1%80%D0%B8%D1%84%D0%BC) рост [высоты дерева](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%B0%D1%80%D1%8C_%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%BE%D0%B2_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2#.D0.92) от числа [узлов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%B0%D1%80%D1%8C_%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%BE%D0%B2_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2#.D0.A3) и быстро выполняющее [основные операции](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)#.D0.9E.D1.81.D0.BD.D0.BE.D0.B2.D0.BD.D1.8B.D0.B5_.D0.BE.D0.BF.D0.B5.D1.80.D0.B0.D1.86.D0.B8.D0.B8_.D0.B2_.D0.B4.D0.B2.D0.BE.D0.B8.D1.87.D0.BD.D0.BE.D0.BC_.D0.B4.D0.B5.D1.80.D0.B5.D0.B2.D0.B5_.D0.BF.D0.BE.D0.B8.D1.81.D0.BA.D0.B0) дерева поиска: добавление, удаление и поиск узла. Сбалансированность достигается за счёт введения дополнительного атрибута узла дерева - «цвета». Этот атрибут может принимать одно из двух возможных значений - «чёрный» или «красный».