# 1. Методы решения задачи коммивояжера.

## 1.1 Общее описание

---

Задача коммивояжера (в дальнейшем сокращённо - ЗК) является одной из знаменитых задач теории комбинаторики. Она была поставлена в 1934 году, и об неё, как об Великую теорему Ферма обламывали зубы лучшие математики. В своей области (оптимизации дискретных задач) ЗК служит своеобразным полигоном, на котором испытываются всё новые методы.

ЗК является NP-полной, и по тому имеет только один абсолютно точный класс алгоритмов – перебор вариантов. Этот класс вариантов решения самый долгий (по времени выполнения), и потому является самым невыгодным при решении задачи с большим количеством городов.

Постановка задачи следующая.

*Коммивояжер должен выйти из первого города, посетить по разу в неизвестном порядке города 2,1,3..n и вернуться в первый город. Расстояния между городами известны. В каком порядке следует обходить города, чтобы замкнутый путь (тур) коммивояжера был кратчайшим?*

Приведем задачу к математическому виду. Города перенумерованы числами $j \in T=(1,2,3..n)$. Тур коммивояжера может быть описан циклической перестановкой $t=(j_1,j_2,..,j_n,j_1)$, причём все $j_1..j_n$ – разные номера; повторяющийся в начале и в конце $j_1$, показывает, что перестановка зациклена. Расстояния между парами вершин $C_{ij}$ образуют матрицу C. Задача состоит в том, что необходимо найти такой тур t, чтобы минимизировать функционал

$$L = L(t) = \sum_{k=1}^{n} C_{j_k j_{k+1}}$$

Относительно данной формулировки ЗК уместно сделать два замечания.

Во-первых, в постановке $C_{ij}$ означали расстояния, поэтому они должны быть неотрицательными, т.е. для всех $j \in T$:

$$C_{ij} \geq 0; \; C_{jj} = \infty \qquad (2)$$

(последнее равенство означает запрет на петли в туре), симметричными, т.е. для всех i,j:

$$C_{ij} = C_{ji}. \qquad (3)$$

и удовлетворять неравенству треугольника, т.е. для всех:

$$C_{ij} + C_{jk} \geq C_{ik} \qquad (4)$$

В математической постановке говорится о произвольной матрице. Сделано это потому, что имеется много прикладных задач, которые описываются основной моделью, но всем условиям (2)-(4) не удовлетворяют. Особенно часто нарушается условие (3) (например, если $C_{ij}$ – не расстояние, а плата за проезд: часто туда билет стоит одну цену, а обратно – другую). Поэтому можно выделить два варианта ЗК: симметричную задачу, когда условие (3) выполнено, и несимметричную - в противном случае. Условия (2)-(4) по умолчанию мы будем считать выполненными.

Второе замечание касается числа всех возможных туров. В несимметричной ЗК все туры $t=(j_1,j_2,..,j_n,j_1)$ и $t'=(j_1,j_n,..,j_2,j_1)$ имеют разную длину и должны учитываться оба. Разных туров очевидно (n-1)!.

Зафиксируем на первом и последнем месте в циклической перестановке номер $j_1$, а оставшиеся n номеров переставим всеми (n-1)! возможными способами. В результате получим все несимметричные туры. Симметричных туров имеется в два раз меньше, т.к. каждый засчитан два раза: как t и как t'.

Можно представить, что C состоит только из единиц и нулей. Тогда C можно интерпретировать, как граф, где ребро (i,j) проведено, если $C_{ij}=1$ и не проведено, если $C_{ij}=0$. Тогда, если существует тур длины n+1, то он пройдёт по циклу, который включает все вершины по одному разу. Такой цикл называется гамильтоновым циклом. Незамкнутый гамильтонов цикл называется гамильтоновой цепью (гамильтоновым путём).

В терминах теории графов симметричную ЗК можно сформулировать так:

*Дана полная сеть с n вершинами, длина ребра (i,j)= $C_{ij}$. Найти гамильтонов цикл минимальной длины.*

В несимметричной ЗК вместо «цикл» надо говорить «контур», а вместо «ребра» - «дуги».

Некоторые прикладные задачи формулируются как ЗК, но в них нужно минимизировать длину не гамильтонова цикла, а гамильтоновой цепи. Такие задачи называются незамкнутыми. Некоторые модели сводятся к задаче о нескольких коммивояжерах, но мы в данной работе их рассматривать не будем.

### 1.2.1. Жадный алгоритм

*Жадный алгоритм* – алгоритм нахождения наикратчайшего расстояния методом выбора самого короткого, ещё не выбранного ребра, при условии, что оно не образует цикла с уже выбранными рёбрами. «Жадным» этот алгоритм назван потому, что на последних шагах приходится жестоко расплачиваться за жадность (последнее ребро, как правило самое большое или близко к нему по длине).

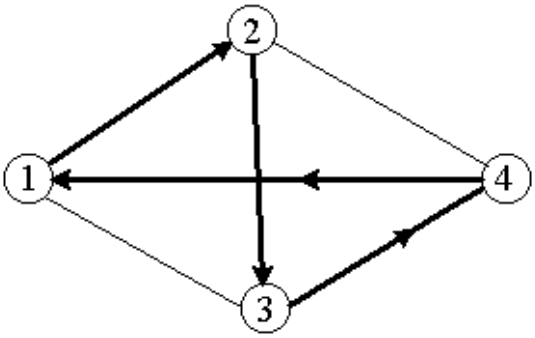

рис 2.

Стратегия: «иди в ближайший (в который еще не входил) город». Рассмотрим для примера сеть на рис. 2, представляющую узкий ромб. Пусть коммивояжер стартует из города 1. Алгоритм «иди вы ближайший город» выведет его в город 2, затем 3, затем 4; на последнем шаге придется платить за жадность, возвращаясь по длинной диагонали ромба. В результате получится не кратчайший, а длиннейший тур.

В пользу процедуры «иди в ближайший» можно сказать лишь то, что при старте из одного города она не уступит стратегии «иди в самый длинный».

Как можно видеть, жадный алгоритм ошибается. Можно ли доказать, что он ошибается умеренно, что полученный им тур хуже минимального, положим, в 1000 раз? Мы докажем, что этого доказать нельзя, причем не только для жадного алгоритма, а для алгоритмов гораздо более мощных. Но сначала нужно договориться, как оценивать погрешность неточных алгоритмов, для определенности, в задаче минимизации. Пусть $f_B$ - настоящий минимум, а $f_A$ - тот квазиминимум, который получен по алгоритму. Ясно, что $f_A/ f_B \geq 1$, но это – тривиальное утверждение, что может быть погрешность. Чтобы оценить её, нужно зажать отношение оценкой сверху:

$$f_A/f_B \geq 1+ n\varepsilon, \qquad (5)$$

где, как обычно в высшей математике, $\varepsilon \geq 0$, но, против обычая, может быть очень большим. Величина $\varepsilon$ и будет служить мерой погрешности. Если алгоритм минимизации будет удовлетворять неравенству (5), мы будем говорить, что он имеет погрешность $\varepsilon$.

Предположим теперь, что имеется алгоритм А решения ЗК, погрешность которого нужно оценить. Возьмем произвольный граф G (V,E) и по нему составим входную матрицу ЗК:

$$C[i,j]=\begin{cases} 1, & \text{если ребро } (i,j) \text{ принадлежит } E \\ 1+n\varepsilon & \text{в противном случае} \end{cases}$$

Если в графе G есть гамильтонов цикл, то минимальный тур проходит по этому циклу и $f_B = n+1$. Если алгоритм А тоже всегда будет находить этот путь, то по результатам алгоритма можно судить, есть ли гамильтонов цикл в произвольном графе. Однако, неперебoрного алгоритма, который мог бы ответить, есть ли гамильтонов цикл в произвольном графе, до сих пор никому не известно. Таким образом, наш алгоритм А должен иногда ошибаться и включать в тур хотя бы одно ребро длины $1+n\varepsilon$. Но тогда $f_A \geq n+(1+n\varepsilon)$ так что $f_A/f_B=1+n\varepsilon$ т.е. превосходит погрешность $\varepsilon$ на заданную неравенством (5) величину. О величине $\varepsilon$ в нашем рассуждении мы не договаривались, так что $\varepsilon$ может быть произвольно велик.

Таким образом доказана следующая теорема.

Либо алгоритм А определяет, существует ли в произвольном графе гамильтонов цикл, либо погрешность А при решении ЗК может быть произвольно велика.

Это соображение было впервые опубликовано Сани и Гонзалесом в 1980 г. Теорема Сани-Гонзалеса основана на том, что нет никаких ограничений на длину ребер. Теорема не проходит, если расстояния подчиняются неравенству треугольника (4).

Если оно соблюдается, можно предложить несколько алгоритмов с погрешностью 12. Прежде, чем описать такой алгоритм, следует вспомнить старинную головоломку. Можно ли начертить одной линией открытый конверт? Рис.3 показывает, что можно (цифры на отрезках показывают порядок их проведения). Закрытый конверт (рис.4.) одной линией нарисовать нельзя и вот почему. Будем называть линии ребрами, а их перекрестья – вершинами.

Когда через точку проводится линия, то используется два ребра – одно для входа в вершину, одно – для выхода. Если степень вершины нечетна – то в ней линия должна начаться или кончиться. На рис. 3 вершин нечетной степени две: в одной линия начинается, в другой – кончается. Однако на рис. 4 имеется четыре вершины степени три, но у одной линии не может быть четыре конца. Если же нужно прочертить фигуру одной замкнутой линией, то все ее

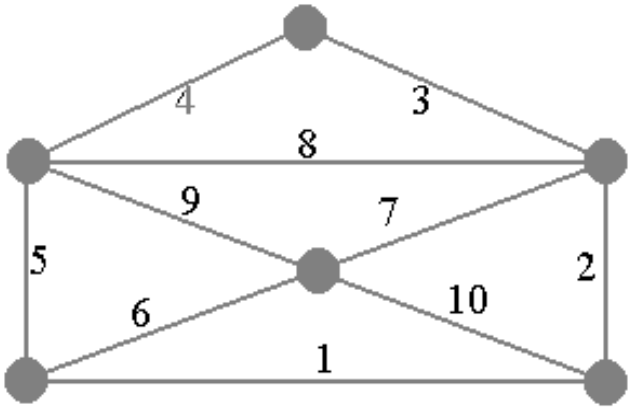

рис. 3

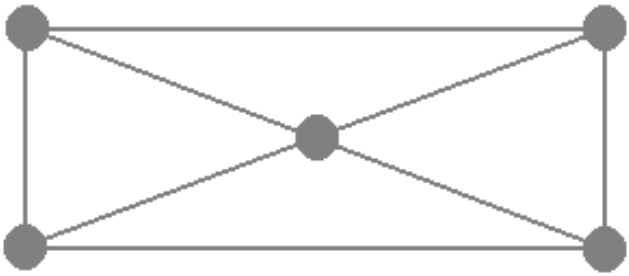
рис. 4

вершины должны иметь четную степень.

Верно и обратное утверждение: если все вершины имеют четную степень, то фигуру можно нарисовать одной незамкнутой линией. Действительно, процесс проведения линии может кончиться, только если линия придет в вершину, откуда уже выхода нет: все ребра, присоединенные к этой вершине (обычно говорят: инцидентные этой вершине), уже прочерчены. Если при этом нарисована вся фигура, то нужное утверждение доказано; если нет, удалим уже нарисованную часть G'. После этого от графа останется одна или несколько связных компонент; пусть G' – одна из таких компонент. В силу связности исходного графа G, G' и G'' имеют хоть одну общую вершину, скажем, v. Если в G'' удалены какие-то ребра, то по четному числу от каждой вершины. Поэтому G'' – связный и все его вершины имеют четную степень. Построим цикл в G'' (может быть, не нарисовав всего G'') и через v добавим прорисованную часть G'' к G'. Увеличивая таким образом прорисованную часть G', мы добьемся того, что G' охватит весь G.

Эту задачу когда-то решил Эйлер, и замкнутую линию, которая покрывает все ребра графа, теперь называю эйлеровым циклом. По существу была доказана следующая теорема.

*Эйлеров цикл в графе существует тогда и только тогда, когда (1) граф связный и (2) все его вершины имеют четные степени.*

В данной работе мы исследуем именно Жадный алгоритм в действии и сравниваем его результаты с результатами лексического перебора.

### 1.2.2. Деревянный алгоритм.

Деревянный алгоритм - алгоритм решения ЗК с помощью построение кратчайшего остовного дерева. Для краткости будет называть этот алгоритм деревянным.

Вначале обсудим свойство спрямления. Рассмотрим какую-нибудь цепь, например, на рис.5. Если справедливо неравенство треугольника, то $d[1,3] \le d[1,2]+d[2,3]$ и $d[3,5] \le d[3,4]+d[4,5]$. Сложив эти два неравенства, получим $d[1,3]+d[3,5] \le d[1,2]+d[2,3]+d[3,4]+d[4,5]$. По неравенству треугольника получим. $d[1,5] \le d[1,3]+d[3,5]$. Окончательно

$$d[1,5] \le d[1,2]+d[2,3]+d[3,4]+d[4,5]$$

Итак, если справедливо неравенство треугольника, то для каждой цепи верно, что расстояние от начала до конца цепи меньше (или равно) суммарной длины всех ребер цепи. Это обобщение расхожего убеждения, что прямая короче кривой.

Вернемся к ЗК и опишем решающий ее деревянный алгоритм.

1. *Построим на входной сети ЗК кратчайшее остовное дерево и удвоим все его ребра. Получим граф G – связный и с вершинами, имеющими только четные степени.*
2. *Построим эйлеров цикл G, начиная с вершины 1, цикл задается перечнем вершин.*
3. *Просмотрим перечень вершин, начиная с 1, и будем зачеркивать каждую вершину, которая повторяет уже встреченную в последовательности. Останется тур, который и является результатом алгоритма.*

Пример 1. Дана полная сеть, показанная на рис.5. Найти тур жадным и деревянным алгоритмами.

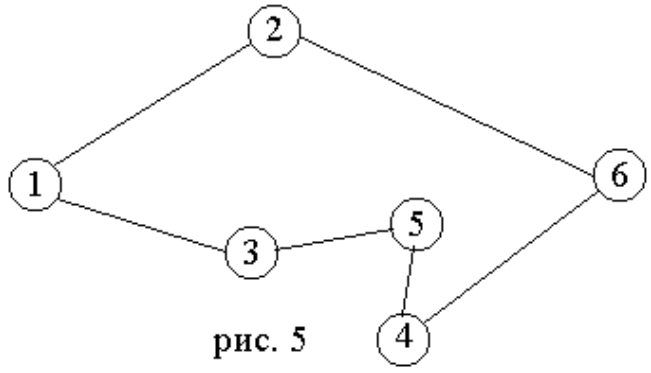

рис. 5

| - | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | 6 | 4 | 8 | 7 | 14 |
| 2 | 6 | - | 7 | 11 | 7 | 10 |
| 3 | 4 | 7 | - | 4 | 3 | 10 |
| 4 | 8 | 11 | 4 | - | 5 | 11 |
| 5 | 7 | 7 | 3 | 5 | - | 7 |
| 6 | 14 | 10 | 10 | 11 | 7 | - |

табл. 1

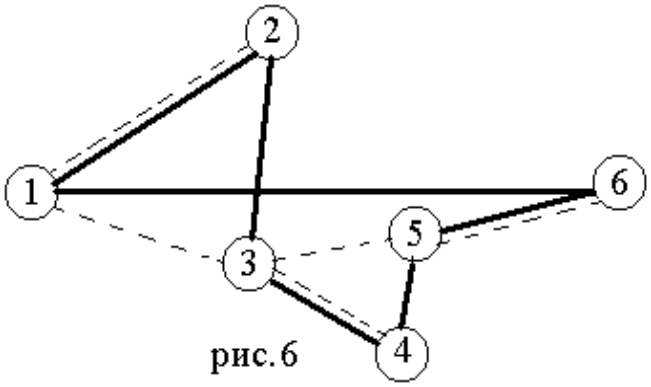

рис.6

**Решение**. Жадный алгоритм (иди в ближайший город из города 1) дает тур 1–(4)–3-(3)–5(5)–4–(11)–6–(10)–2–(6)–1, где без скобок показаны номера вершин, а в скобках – длины ребер. Длина тура равна 39, тур показана на рис. 5.

Деревянный алгоритм вначале строит остовное дерево, показанное на рис. 6 штриховой линией, затем эйлеров цикл 1-2-1-3-4-3-5-6-5-3-1, затем тур 1-2-3-4-5-6-1 длиной 43, который показан сплошной линией на рис. 6.

***Теорема***. *Погрешность деревянного алгоритма равна 1.*

Доказательство. Возьмем минимальный тур длины $f_B$ и удалим из него максимальное ребро. Длина получившейся гамильтоновой цепи $L_{HC}$ меньше $f_B$. Но эту же цепь можно рассматривать как остовное дерево, т. к. эта цепь достигает все вершины и не имеет циклов. Длина кратчайшего остовного дерева $L_{MT}$ меньше или равна $L_{HC}$. Имеем цепочку неравенств

$$f_B > L_{HC} \ge L_{MT} \qquad (6)$$

Но удвоенное дерево – оно же эйлеров граф – мы свели к туру посредством спрямлений, следовательно, длина полученного по алгоритму тура удовлетворяет неравенству

$$2L_{MT} > f_A \qquad (7)$$

Умножая (6) на два и соединяя с (7), получаем цепочку неравенств

$$2f_B > 2L_{HC} \ge 2L_{MT} \ge f_A \qquad (8)$$

Т.е. $2f_B > f_A$, т.е. $f_A/f_B > 1+\varepsilon$; $\varepsilon=1$.

Теорема доказана.

Таким образом, мы доказали, что деревянный алгоритм ошибается менее, чем в два раза. Такие алгоритмы уже называют приблизительными, а не просто эвристическими.

Известно еще несколько простых алгоритмов, гарантирующих в худшем случае $\varepsilon=1$. Для того, чтобы найти среди них алгоритм поточнее, зайдем с другого конца и для начала опишем «brute-force enumeration» - «перебор животной силой», как его называют в англоязычной литературе. Понятно, что полный перебор практически применим

только в задачах малого размера. Напомним, что ЗК с n городами требует при полном переборе рассмотрения (n-1)!/2 туров в симметричной задаче и (n-1)! Туров в несимметричной, а факториал, как показано в следующей таблице, растет довольно быстро:

| 5! | 10! | 15! | 20! | 25! | 30! | 35! | 40! | 45! | 50! |
|---|---|---|---|---|---|---|---|---|---|
| $\sim 10^2$ | $\sim 10^6$ | $\sim 10^{12}$ | $\sim 10^{18}$ | $\sim 10^{25}$ | $\sim 10^{32}$ | $\sim 10^{40}$ | $\sim 10^{47}$ | $\sim 10^{56}$ | $\sim 10^{64}$ |

Чтобы проводить полный перебор в ЗК, нужно научиться (разумеется, без повторений) генерировать все перестановки заданного числа m элементов. Это можно сделать несколькими способами, но самый распространенный (т.е. приложимый для переборных алгоритмов решения других задач) – это перебор в лексикографическом порядке.

Пусть имеется некоторый алфавит и наборы символов алфавита (букв), называемые словами. Буквы в алфавите упорядочены: например, в русском алфавите порядок букв а∝б∝я (символ ∝ читается «предшествует»). Если задан порядок букв, можно упорядочить и слова. Скажем, дано слово $u=(u_1,u_2,..,u_m)$ – состоящее из букв $u_1,u_2,..,u_m$ - и слово $v=(v_1,v_2,..,v_b)$. Тогда если $u_1 \propto v_1$, то и $u \propto v$, если же $u_1=v_1$, то сравнивают вторые буквы и т.д. Этот порядок слов и называется лексикографическим. Поэтому в русских словарях (лексиконах) слово «абажур» стоит раньше слова «абака». Слово «бур» стоит раньше слова «бура», потому что пробел считается предшествующим любой букве алфавита.

Рассмотрим, скажем, перестановки из пяти элементов, обозначенных цифрами 1..5. Лексикографически первой перестановкой является 1-2-3-4-5, второй – 1-2-3-5-4, …, последней – 5-4-3-2-1. Нужно осознать общий алгоритм преобразования любой перестановки в непосредственно следующую.

Правило такое: скажем, дана перестановка 1-3-5-4-2. Нужно двигаться по перестановке справа налево, пока впервые не увидим число, меньшее, чем предыдущее (в примере это 3 после 5). Это число, $P_{i-1}$ надо увеличить, поставив вместо него какое-то число из расположенных правее, от $P_i$ до $P_n$. Число большее, чем $P_{i-1}$, несомненно, найдется, так как $P_{i-1} < P_i$ . Если есть несколько больших чисел, то, очевидно, надо ставить меньшее из них. Пусть это будет $P_j$,$j>i-1$. Затем число $P_{i-1}$ и все числа от $P_i$ до $P_n$, не считая $P_j$ нужно упорядочить по возрастанию. В результате получится непосредственно следующая перестановка, в примере – 1-4-2-3-5. Потом получится 1-4-2-5-3 (тот же алгоритм, но упрощенный случай) и т.д.

Нужно понимать, что в ЗК с n городами не нужны все перестановки из n элементов. Потому что перестановки, скажем, 1-3-5-4-2 и 3-5-4-2-1 (последний элемент соединен с первым) задают один и тот же тур, считанный сперва с города 1, а потом с города 3. Поэтому нужно зафиксировать начальный город 1 и присоединять к нему все перестановки из остальных элементов. Этот перебор даст (n-1)! разных туров, т.е. полный перебор в несимметричной ЗК (мы по-прежнему будем различать туры 1-3-5-4-2 и 1-2-4-5-3).

Данный алгоритм описан на языке Паскаль в качестве образца для исследований.

Пример 2. Решим ЗК, поставленную в Примере 1 лексикографическим перебором. Приведенная выше программа напечатает города, составляющие лучший тур: 1-2-6-5-4-3 и его длину 36.

### 1.2.3. Метод ветвей и границ

К идее метода ветвей и границ приходили многие исследователи, но Литтл с соавторами на основе указанного метода разработали удачный алгоритм решения ЗК и тем самым способствовали популяризации подхода. С тех пор метод ветвей и границ был успешно применен ко многим задачам, для решения ЗК было придумано несколько других модификаций метода, но в большинстве учебников излагается пионерская работа Литтла.

Общая идея тривиальна: нужно разделить огромное число перебираемых вариантов на классы и получить оценки (снизу – в задаче минимизации, сверху – в задаче максимизации) для этих классов, чтобы иметь возможность отбрасывать варианты не по одному, а целыми классами. Трудность состоит в том, чтобы найти такое разделение на классы (ветви) и такие оценки (границы), чтобы процедура была эффективной.

Изложим алгоритм Литтла на примере 1 предыдущего раздела.. Повторно запишем матрицу:

| - | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | 6 | 4 | 8 | 7 | 14 |
| 2 | 6 | - | 7 | 11 | 7 | 10 |
| 3 | 4 | 7 | - | 4 | 3 | 10 |
| 4 | 8 | 11 | 4 | - | 5 | 11 |
| 5 | 7 | 7 | 3 | 5 | - | 7 |
| 6 | 14 | 10 | 10 | 11 | 7 | - |

табл. 2

Нам будет удобнее трактовать $C_{ij}$ как стоимость проезда из города i в город j. Допустим, что добрый мэр города j издал указ выплачивать каждому въехавшему в город коммивояжеру 5 долларов. Это означает, что любой тур подешевеет на 5 долларов, поскольку в любом туре нужно въехать в город j. Но поскольку все туры равномерно подешевели, то прежний минимальный тур будет и теперь стоить меньше всех. Добрый же поступок мэра можно представить как уменьшение всех чисел j-го столбца матрицы С на 5. Если бы мэр хотел спровадить коммивояжеров из j-го города и установил награду за выезд в размере 10 долларов, это можно было бы выразить вычитанием 10 из всех элементов j-й той строки. Это снова бы изменило стоимость каждого тура, но минимальный тур остался бы минимальным. Итак, доказана следующая лемма.

*Вычитая любую константу из всех элементов любой строки или столбца матрицы С, мы оставляем минимальный тур минимальным.*

| - | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | 0 | 0 | 3 | 3 | 6 |
| 2 | 0 | - | 1 | 4 | 1 | 0 |
| 3 | 1 | 2 | - | 0 | 0 | 3 |
| 4 | 4 | 5 | 0 | - | 1 | 3 |
| 5 | 4 | 2 | 0 | 1 | - | 0 |
| 6 | 7 | 1 | 3 | 3 | 0 | - |
|  |  | 2 |  | 1 |  | 4 |

табл. 4

Для алгоритма нам будет удобно получить побольше нулей в матрице С, не получая там, однако, отрицательных чисел. Для этого мы вычтем из каждой строки ее минимальный элемент (это называется приведением по строкам, см. табл. 3), а затем вычтем из каждого столбца матрицы, приведенной по строкам, его минимальный элемент, получив матрицу, приведенную по столбцам, см. табл. 4).

Прочерки по диагонали означают, что из города i в город i ходить нельзя. Заметим, что сумма констант приведения по строкам равна 27, сумма по столбцам 7, сумма сумм равна 34.

| - | 1 | 2 | 3 | 4 | 5 | 6 |  |
|---|---|---|---|---|---|---|---|
| 1 | - | 2 | 0 | 4 | 3 | 10 | 4 |
| 2 | 0 | - | 1 | 5 | 1 | 4 | 6 |
| 3 | 1 | 4 | - | 1 | 0 | 7 | 3 |
| 4 | 4 | 7 | 0 | - | 1 | 7 | 4 |
| 5 | 4 | 4 | 0 | 2 | - | 4 | 3 |
| 6 | 7 | 3 | 3 | 4 | 0 | - | 7 |

табл. 3

Тур можно задать системой из шести подчеркнутых (выделенных другим цветом) элементов матрицы С, например, такой, как показано на табл. 2. Подчеркивание элемента означает, что в туре из i-го элемента идут именно в j-тый. Для тура из шести городов подчеркнутых элементов должно быть шесть, так как в туре из шести городов есть шесть ребер. Каждый столбец должен содержать ровно один подчеркнутый элемент (в каждый город коммивояжер въехал один раз), в каждой строке должен быть ровно один подчеркнутый элемент (из каждого города коммивояжер выехал один раз); кроме того, подчеркнутые элементы должны описывать один тур, а не несколько меньших циклов. Сумма чисел подчеркнутых элементов есть стоимость тура. На табл. 2 стоимость равна 36, это тот минимальный тур, который получен лексикографическим перебором.

Теперь будем рассуждать от приведенной матрицы на табл. 2. Если в ней удастся построить правильную систему подчеркнутых элементов, т.е. систему, удовлетворяющую трем вышеописанным требованиям, и этими подчеркнутыми элементами будут только нули, то ясно, что для этой матрицы мы получим минимальный тур. Но он же будет минимальным и для исходной матрицы С, только для того, чтобы получить правильную стоимость тура, нужно будет обратно прибавить все константы приведения, и стоимость тура изменится с 0 до 34. Таким образом, минимальный тур не может быть меньше 34. Мы получили оценку снизу для всех туров.

Теперь приступим к ветвлению. Для этого проделаем шаг оценки нулей. Рассмотрим нуль в клетке (1,2) приведенной матрицы. Он означает, что цена перехода из города 1 в город 2 равна 0. А если мы не пойдем из города 1 в город 2? Тогда все равно нужно въехать в город 2 за цены, указанные во втором столбце; дешевле всего за 1 (из города 6). Далее, все равно надо будет выехать из города 1 за цену, указанную в первой строке; дешевле всего в город 3 за 0. Суммируя эти два минимума, имеем 1+0=1: если не ехать «по нулю» из города 1 в город 2, то надо заплатить не меньше 1. Это и есть оценка нуля. Оценки всех нулей поставлены на табл. 5 правее и выше нуля (оценки нуля, равные нулю, не ставились).

Выберем максимальную из этих оценок (в примере есть несколько оценок, равных единице, выберем первую из них, в клетке (1,2)).

Итак, выбрано нулевое ребро (1,2). Разобьем все туры на два класса – включающие ребро (1,2) и не включающие ребро (1,2). Про второй класс можно сказать, что придется приплатить еще 1, так что туры этого класса стоят 35 или больше.

Что касается первого класса, то в нем надо рассмотреть матрицу на табл. 6 с вычеркнутой первой строкой и вторым столбцом.

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | $0^1$ | 0 | 3 | 3 | 6 |
| 2 | $0^1$ | - | 1 | 4 | 1 | 0 |
| 3 | 1 | 2 | - | $0^1$ | 0 | 3 |
| 4 | 4 | 5 | $0^1$ | - | 1 | 3 |
| 5 | 4 | 2 | 0 | 1 | - | 0 |
| 6 | 7 | 1 | 3 | 3 | $0^1$ | - |

табл. 5

| | 1 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 2 | $0^1$ | 1 | 4 | 1 | 0 |
| 3 | 1 | - | $0^1$ | 0 | 3 |
| 4 | 4 | $0^1$ | - | 1 | 3 |
| 5 | 4 | 0 | 1 | - | 0 |
| 6 | 7 | 3 | 3 | $0^1$ | - |

табл. 6

| | 1 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 2 | $0^1$ | 1 | 4 | 1 | 0 |
| 3 | $0^3$ | - | $0^1$ | 0 | 3 |
| 4 | 3 | $0^1$ | - | 1 | 3 |
| 5 | 3 | 0 | 1 | - | 0 |
| 6 | 6 | 3 | 3 | $0^1$ | - |

табл. 7

| | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| 2 | 1 | 4 | 1 | 0 |
| 4 | $0^1$ | - | 1 | 3 |
| 5 | 0 | 1 | - | 0 |
| 6 | 3 | 3 | $0^1$ | - |

табл. 8

Дополнительно в уменьшенной матрице поставлен запрет в клетке (2,1), т. к. выбрано ребро (1,2) и замыкать преждевременно тур ребром (2,1) нельзя. Уменьшенную матрицу можно привести на 1 по первому столбцу, так что каждый тур, ей отвечающий, стоит не меньше 35. Результат наших ветвлений и получения оценок показан на рис.6.

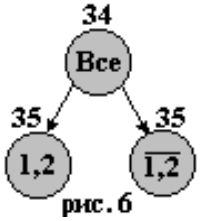

рис. 6

Кружки представляют классы: верхний кружок – класс всех туров; нижний левый – класс всех туров, включающих ребро (1,2); нижний правый – класс всех туров, не включающих ребро (1,2). Числа над кружками – оценки снизу.

Продолжим ветвление в положительную сторону: влево - вниз. Для этого оценим нули в уменьшенной матрице C[1,2] на табл. 7. Максимальная оценка в клетке (3,1) равна 3. Таким образом, оценка для правой нижней вершины на рис. 7 есть 35+3=38. Для оценки левой нижней вершины на рис. 7 нужно вычеркнуть из матрицы C[1,2] еще строку 3 и столбец 1, получив матрицу C[(1,2),(3,1)] на табл. 8. В эту матрицу нужно поставить запрет в клетку (2,3), так как уже построен фрагмент тура из ребер (1,2) и (3,1), т.е. [3,1,2], и нужно запретить преждевременное замыкание (2,3). Эта матрица приводится по столбцу на 1 (табл. 9), таким образом, каждый тур соответствующего класса (т.е. тур, содержащий ребра (1,2) и (3,1)) стоит 36 и более.

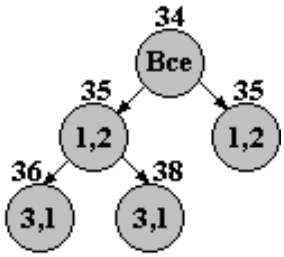

рис. 7

Оцениваем теперь нули в приведенной матрице C[(1,2),(3,1)] нуль с максимальной оценкой 3 находится в клетке (6,5). Отрицательный вариант имеет оценку 38+3=41. Для получения оценки положительного варианта убираем строчку 6 и столбец 5, ставим запрет в клетку (5,6), см. табл. 10. Эта матрица неприводима. Следовательно, оценка положительного варианта не увеличивается (рис.8).

| | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| 2 | 1 | 3 | 1 | 0 |
| 4 | $0^1$ | - | 1 | 3 |
| 5 | 0 | $0^2$ | - | 0 |
| 6 | 3 | 2 | $0^3$ | - |

табл. 9

Оценивая нули в матрице на табл. 10, получаем ветвление по выбору ребра (2,6), отрицательный вариант получает оценку 36+3=39, а для получения оценки положительного варианта вычеркиваем вторую строку и шестой столбец, получая матрицу на табл. 11.

В матрицу надо добавить запрет в клетку (5,3), ибо уже построен фрагмент тура [3,1,2,6,5] и надо запретить преждевременный возврат (5,3). Теперь, когда осталась матрица 2x2 с запретами по диагонали, достраиваем тур ребрами (4,3) и (5,4). Мы не зря ветвились, по положительным вариантам. Сейчас получен тур: 1→2→6→5→4→3→1 стоимостью в 36. При достижении низа по дереву перебора класс туров сузился до одного тура, а оценка снизу превратилась в точную стоимость.

| | 3 | 4 | 6 |
|---|---|---|---|
| 2 | 1 | 3 | $0^3$ |
| 4 | $0^3$ | - | 3 |
| 5 | 0 | $0^3$ | 0 |

табл. 10

| | 3 | 4 |
|---|---|---|
| 4 | 0 | - |
| 5 | 0 | 0 |

табл. 11

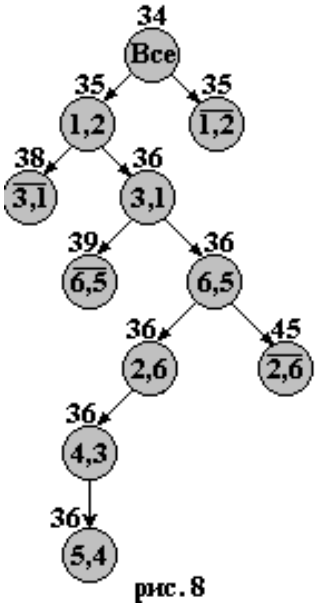

рис. 8

Итак, все классы, имеющие оценку 36 и выше, лучшего тура не содержат. Поэтому соответствующие вершины вычеркиваются. Вычеркиваются также вершины, оба потомка которой вычеркнуты. Мы колоссально сократили полный перебор. Осталось проверить, не содержит ли лучшего тура класс, соответствующий матрице C[Not(1,2)], т.е. приведенной матрице C с запретом в клетке 1,2, приведенной на 1 по столбцу (что дало оценку 34+1=35). Оценка нулей дает 3 для нуля в клетке (1,3), так что оценка отрицательного варианта 35+3 превосходит стоимость уже полученного тура 36 и отрицательный вариант отсекается.

Для получения оценки положительного варианта исключаем из матрицы первую строку и третий столбец, ставим запрет (3,1) и получаем матрицу. Эта матрица приводится по четвертой строке на 1, оценка класса достигает 36 и кружок зачеркивается. Поскольку у вершины «все» убиты оба потомка, она убивается тоже. Вершин не осталось, перебор окончен. Мы получили тот же минимальный тур, который показан подчеркиванием на табл. 2.

Удовлетворительных теоретических оценок быстродействия алгоритма Литтла и родственных алгоритмов нет, но практика показывает, что на современных ЭВМ они часто позволяют решить ЗК с n = 100. Это огромный прогресс по сравнению с полным перебором. Кроме того, алгоритмы типа ветвей и границ являются, если нет возможности доводить их до конца, эффективными эвристическими процедурами.

### 1.2.4. Алгоритм Дейкстры

Одним из вариантов решения ЗК является вариант нахождения кратчайшей цепи, содержащей все города. Затем полученная цепь дополняется начальным городом – получается искомый тур.

Можно предложить много процедур решения этой задачи, например, физическое моделирование. На плоской доске рисуется карта местности, в города, лежащие на развилке дорог, вбиваются гвозди, на каждый гвоздь надевается кольцо, дороги укладываются верёвками, которые привязываются к соответствующим кольцам. Чтобы найти кратчайшее расстояние между i и k, нужно взять I в одну руку и k в другую и растянуть. Те верёвки, которые натянутся и не дадут разводить руки шире и образуют кратчайший путь между i и k. Однако математическая процедура, которая промоделирует эту физическую, выглядит очень сложно. Известны алгоритмы попроще. Один из них – алгоритм Дейкстры, предложенный Дейкстрой ещё в 1959г. Этот алгоритм решает общую задачу:

*В ориентированной, неориентированной или смешанной (т. е. такой, где часть дорог имеет одностороннее движение) сети найти кратчайший путь между двумя заданными вершинами.*

Алгоритм использует три массива из n (= числу вершин сети) чисел каждый. Первый массив a содержит метки с двумя значениями: 0 (вершина ещё не рассмотрена) и 1 (вершина уже рассмотрена); второй массив b содержит расстояния – текущие кратчайшие расстояния от $v_i$ до соответствующей вершины; третий массив c содержит номера вершин – k-й элемент $c_k$ есть номер предпоследней вершины на текущем кратчайшем пути из $v_i$ в $v_k$. Матрица расстояний $D_{ik}$ задаёт длины дуг $d_{ik}$; если такой дуги нет, то $d_{ik}$ присваивается большое число Б, равное «машинной бесконечности».

Теперь можно описать:

**Алгоритм Дейкстры**

**1**(инициализация).

В цикле от одного до n заполнить нулями массив a; заполнить числом i массив c: перенести i-тую строку матрицы D в массив b;

a[i]:=1; c[i]:=0; {i-номер стартовой вершины}

**2**(общий шаг).

Найти минимум среди неотмеченных (т. е. тех k, для которых a[k]=0); пусть минимум достигается на индексе j, т. е. $b_j \le b_k$; a[j]:=1;

если $b_k > b_j + d_{jk}$ то ($b_k := b_j + d_{jk}$; $c_k := j$) {Условие означает, что путь $v_i..v_k$ длиннее, чем путь $v_i..v_j, v_k$ . Если все a[k] отмечены, то длина пути $v_i..v_k$ равна b[k]. Теперь надо перечислить вершины, входящие в кратчайший путь}

**3**(выдача ответа).

**{**Путь $v_i..v_k$ выдаётся в обратном порядке следующей процедурой:**}**

3.1. z:=c[k];

3.2. Выдать z;

3.3. z:=c[z]; Если z = 0, то конец, иначе перейти к 3.2.

Для выполнения алгоритма нужно n раз просмотреть массив b из n элементов, т. е. алгоритм Дейкстры имеет квадратичную сложность. Проиллюстрируем работу алгоритма Дейкстры численным примером (для большей сложности, считаем, что некоторые города (вершины) i,j не соединены между собой, т. е. D[i,j]=∞). Пусть, например, i=3. Требуется найти кратчайшие пути из вершины 3. Содержимое массивов a,b,c после выполнения первого пункта показано на табл. 12:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 23 | 12 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 23 | 0 | 25 | ∞ | 22 | ∞ | ∞ | 35 |
| 12 | 25 | 0 | 18 | ∞ | ∞ | ∞ | ∞ |
| ∞ | ∞ | 18 | 0 | ∞ | 20 | ∞ | ∞ |
| ∞ | 22 | ∞ | ∞ | 0 | 23 | 14 | ∞ |
| ∞ | ∞ | ∞ | 20 | 23 | 0 | 24 | ∞ |
| ∞ | ∞ | ∞ | ∞ | 14 | 24 | 0 | 16 |
| ∞ | 35 | ∞ | ∞ | ∞ | ∞ | 16 | 0 |

табл. 12

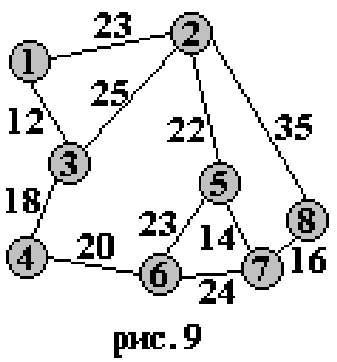


рис. 9

Очевидно, содержимое таблицы меняется по мере

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| a | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| b | 12 | 25 | 0 | 18 | ∞ | ∞ | ∞ | ∞ |
| c | 3 | 3 | 0 | 3 | 3 | 3 | 3 | 3 |

табл. 13

выполнения общего шага. Это видно из следующей таблицы:

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **min $b_k$=12** | **a** | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **0** |
| | **b** | **12** | **25** | **0** | **18** | ∞ | ∞ | ∞ | ∞ |
| | **c** | **3** | **3** | **0** | **3** | **3** | **3** | **3** | **3** |
| **min $b_k$=18** | **a** | **1** | **0** | **1** | **1** | **0** | **0** | **0** | **0** |
| | **b** | **12** | **25** | **0** | **18** | ∞ | **38** | ∞ | ∞ |
| | **c** | **3** | **3** | **0** | **3** | **3** | **4** | **3** | **3** |
| **min $b_k$=25** | **a** | **1** | **1** | **1** | **1** | **0** | **0** | **0** | **0** |
| | **b** | **12** | **25** | **0** | **18** | **47** | **38** | ∞ | **60** |
| | **c** | **3** | **3** | **0** | **3** | **2** | **4** | **3** | **2** |
| **min $b_k$=38** | **a** | **1** | **1** | **1** | **1** | **0** | **1** | **0** | **0** |
| | **b** | **12** | **25** | **0** | **18** | **47** | **38** | **62** | **60** |
| | **c** | **3** | **3** | **0** | **3** | **2** | **4** | **6** | **2** |
| **min $b_k$=47** | **a** | **1** | **1** | **1** | **1** | **1** | **1** | **0** | **0** |
| | **b** | **12** | **25** | **0** | **18** | **47** | **38** | **61** | **60** |
| | **c** | **3** | **3** | **0** | **3** | **2** | **4** | **5** | **2** |
| **min $b_k$=60** | **a** | **1** | **1** | **1** | **1** | **1** | **1** | **0** | **1** |
| | **b** | **12** | **25** | **0** | **18** | **47** | **38** | **61** | **60** |
| | **c** | **3** | **3** | **0** | **3** | **2** | **4** | **5** | **2** |

Таким образом, для решения ЗК нужно n раз применить алгоритм Дейкстры следующим образом.

Возьмём произвольную пару вершин

j,k. Исключим непосредственное ребро C[j,k]. С помощью алгоритма Дейкстры найдём кратчайшее расстояние между городами j..k. Пусть это расстояние включает некоторый город m. Имеем часть тура j,m,k. Теперь для каждой пары соседних городов (в данном примере – для j,m и m,k) удалим соответственное ребро и найдём кратчайшее расстояние. При этом в кратчайшее расстояние не должен входить уже использованный город.

Далее аналогично находим кратчайшее расстояние между парами вершин алгоритмом Дейкстры, до тех пор, пока все вершины не будут задействованы. Соединим последнюю вершину с первой и получим тур. Чаще всего это последнее ребро оказывается очень большим, и тур получается с погрешностью, однако алгоритм Дейкстры можно отнести к приближённым алгоритмам.

## *1.3 Практическое применение задачи коммивояжера*

Кроме очевидного применения ЗК на практике, существует ещё ряд задач, сводимых к решению ЗК.

***Задача о производстве красок***. Имеется производственная линия для производства n красок разного цвета; обозначим эти краски номерами 1,2… n. Всю производственную линию будем считать одним процессором.. Будем считать также, что единовременно процессор производит только одну краску, поэтому краски нужно производить в некотором порядке. Поскольку производство циклическое, то краски надо производить в циклическом порядке $\pi=(j_1,j_2,..,j_n,j_1)$. После окончания производства краски i и перед началом производства краски j надо отмыть оборудование от краски i. Для этого требуется время C[i,j]. Очевидно, что C[i,j] зависит как от i, так и от j, и что, вообще говоря,$C[i,j]\neq C[j,i]$. При некотором выбранном порядке придется на цикл производства красок потратить время

$$f = \sum_{(j,i)\in\pi} C_{ij} + \sum_{k=1}^{n} t_k$$

Где $t_k$ - чистое время производства k-ой краски (не считая переналадок). Однако вторая сумма в правой части постоянна, поэтому полное время на цикл производства минимизируется вместе с общим временем на переналадку.

Таким образом, ЗК и задача о минимизации времени переналадки – это просто одна задача, только варианты ее описаны разными словами.

***Задача о дыропробивном прессе***. Дыропробивной пресс производит большое число одинаковых панелей – металлических листов, в которых последовательно по одному пробиваются отверстия разной формы и величины. Схематически пресс можно представить в виде стола, двигающегося независимо по координатам x, y, и вращающегося над столом диска, по периметру которого расположены дыропробивные инструменты разной формы и величины. Каждый инструмент присутствует в одном экземпляре. Диск может вращаться одинаково в двух направлениях (координата вращения z). Имеется собственно пресс, который надавливает на подвешенный под него инструмент тогда, когда под инструмент подведена нужная точка листа.

Операция пробивки j-того отверстия характеризуется четверкой чисел $(x_j,y_j,z_j,t_j)$,, где $x_j,y_j$- координаты нужного положения стола, $z_j$ - координата нужного положения диска и $t_j$ - время пробивки j-того отверстия.

Производство панелей носит циклический характер: в начале и конце обработки каждого листа стол должен находиться в положениях $(x_0, y_0)$ диск в положении $z_0$ причем в этом положении отверстие не пробивается. Это начальное состояние системы можно считать пробивкой фиктивного нулевого отверстия. С параметрами $(x_0,y_0,z_0,0)$.

Чтобы пробить j-тое отверстие непосредственно после i-того необходимо произвести следующие действия:

1. Переместить стол по оси x из положения $x_i$ в положение $x_j$, затрачивая при этом время $t^{(x)}(|x_i-x_j|)=t_{i,j}^{(x)}$
1. Проделать то же самое по оси y, затратив время $t_{i,j}^{(y)}$
2. Повернуть головку по кратчайшей из двух дуг из положения $z_i$ в положение $z_j$, затратив время $t_{i,j}^{(z)}$ .
3. Пробить j-тое отверстие, затратив время $t_j$.

Конкретный вид функций $t^{(x)}$, $t^{(y)}$, $t^{(z)}$ зависит от механических свойств пресса и достаточно громоздок. Явно выписывать эти функции нет необходимости

Действия 1-3 (переналадка с i-того отверстия j-тое) происходит одновременно, и пробивка происходит немедленно после завершения самого длительного из этих действий. Поэтому

$$C[i,j] = \max(t^{(x)}, t^{(y)}, t^{(z)})$$

Теперь, как и в предыдущем случае, задача составления оптимальной программы для дыропробивного пресса сводится к ЗК (здесь - симметричной).

# 2. Литература


1. О. Оре  Графы и их применение. Пер. с англ. под ред. И.М. Яглома. - М., «Мир», 1965, 174 с.
2. В. П. Сигорский. Математический аппарат инженера. - К., «Техніка», 1975, 768 с.
3. Ю. Н. Кузнецов, В. И. Кузубов, А. Б. Волощенко. Математическое программирование: учебное пособие. 2-е изд. перераб. и доп. - М.; Высшая школа, 1980, 300 с., ил.
4. Е. В. Маркова, А. Н. Лисенков. Комбинаторные планы в задачах многофакторного эксперимента. – М., Наука, 1979, 345 с.
5. Е. П. Липатов. Теория графов и её применения. - М., Знание, 1986, 32 с.
6. В. М. Бондарев, В. И. Рублинецкий, Е. Г. Качко. Основы программирования. – Харьков, Фолио; Ростов на Дону, Феникс, 1998, 368 с.
7. Ф. А. Новиков Дискретная математика для программистов. - Санкт-Петербург, Питер, 2001, 304 с., ил.