# Лекция 7 от 04.10.2016. Венгерский алгоритм решение задачи о назначениях

## Формальная постановка задачи

Оригинальная постановка задачи о назначениях выгдядит так: нам дана матрица $n \times n$

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix},$$

где строки отвечают за работников, которые требуют определенную сумму за то или иное задание (столбцы).

Требуется найти такую перестановку $p(i)$, что $\sum\limits_{i=1}^{n} a_{ip(i)} \to \min$. Обозначим эту задачу за $A'$.

## Немного линейного программирования

Выпишем задачу линейного программирования (назовём её $B'$).

$$\begin{aligned} \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_{ij} &\to \min \\ \forall i \in \overline{1\ldots n} \implies \sum_{j=1}^{n} x_{ij} &= 1, \\ \forall j \in \overline{1\ldots n} \implies \sum_{i=1}^{n} x_{ij} &= 1, \\ x_{ij} &\geqslant 0. \end{aligned}$$

Ясно, что если есть решение задачи $A'$, то оно является каким-то решением задачи $B'$, так как мы в каждой строке и столбце у нас ровно одно $x_{ij} = 1$, поэтому все такие решения подойдут под $B'$. Значит решение задачи $B'$ не хуже, чем у $A'$. Выпишем этой задаче двойственную (что, конечно же, было проделано много раз на курсе дискретной математики). Двойственные переменные $u_i$ отвечают за строки, $v_i$ за столбцы. Напомним пару двойственных задач:

$$\begin{cases} cx \to \max \\ Ax \leqslant b. \end{cases} \quad \text{двойственна} \quad \begin{cases} yb \to \min \\ yA = c, \\ y \geqslant 0. \end{cases}$$

Поэтому двойственная к $B'$ будет

$$\begin{aligned} \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j &\to \max \\ \forall i,j \in \overline{1\ldots n} \implies u_i + v_j &\leqslant a_{ij}. \end{aligned}$$

Это действительно так, потому что в строках матрицы $A$ задачи $B'$ будет лишь 2 элемента — отвечающее за строку и столбец, в котором переменная $x_{ij}$ находится.

Обозначим эту задачу за $C'$.

Как мы знаем из курса дискретной математики, двойственная и прямая задачи имеют одно и то же оптимальное значение, поэтому оптимальные значения $B'$ и $C'$ совпадают. Также легко убедиться, что каждая задача имеет хоть какое-то решение.

Заметим, что любое решение задачи $A'$ является решением задачи $C'$. Действительно,

$$\sum_{i=1}^{n} a_{ip(i)} \geqslant \sum_{i=1}^{n} u_i + v_{p(i)} = \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j.$$

То есть любое допустимая перестановка не меньше, чем любое значение целевой функции задачи $C'$, поэтому если мы предъявим хоть какое-то решение, что значение целевой функции задачи $C'$ совпадает с каким-то из задачи $A'$, то мы найдём оптимальное решение (достигается равенство выше). И тогда все оптимальные решения задач $A', B', C'$ совпадают. Предъявим конструктивно алгоритм построения решения $C'$.

## Венгерский алгоритм

Нарисуем двудольный граф, где левая доля отвечает переменным $u_i$, правая за $v_j$.

**Определение 1.** *Назовём ребро $(i, j)$* ***жёстким****, если $u_i + v_j = a_{ij}$.*

Заметим, что если мы найдём совершенное паросочетание на жёстких ребрах, то мы найдём какую-то перестановку $p(i)$, тогда мы найдём какое-то решение, где $u_i + v_{p(i)}$ будет входить в сумму целевой функции, как $a_{ip(i)}$, а значит мы найдём какое-то решение задачи $A'$. Поэтому надо всего лишь «подвигать значения переменных», чтобы получить совершенное паросочетание на жёстких ребрах.

Мы будем пытаться добавлять вершины левой доли по одной на каждом шаге, в предположении, что на предыдущих шагах мы смогли всё добавить. Пусть мы добавляем вершину $i$, а на всех $i - 1$ предыдущих максимальное паросочетание можно построить. Запустим алгоритм Куна из $i$-ой вершины. Если мы нашли паросочетание и на $i$-ой вершине, то отлично, мы увеличили паросочетание. Теперь допустим, что не смогли увеличить.

Обозначим за $R_+$ — все вершины, которые мы достигли алгоритмом Куна в правой доли из вершины $i$, $L_+$ в левой. Аналогично определяются множества вершин $R_-$ и $L_-$. В $R_+$ вершин меньше, так как для любой вершины из $R_+$ алгоритм Куна посетит вместе с ней какую-то вершину правой доли из паросочетания.

Заметим, что нет жёстких ребёр из $L_+$ в $R_-$, так как иначе мы могли бы найти удлиняющий путь (просто пройдя как-то по этой вершине).

Теперь введём обозначение $d = \min(a_{ij} - u_i - v_j), i \in L_+, j \in R_-$. Как мы только что показали, $d > 0$, так как ни одно ребро не жёсткое.

Теперь применим данные действия к алгоритму:

- Увеличим все $u_i$ на $d$ для $i \in L_+$,
- Уменьшим все $v_j$ на $d$ для $j \in R_+$.

Заметим, что целевая функция увеличилась хотя бы на $d$, так как в $L_+$ больше вершин, чем в $R_+$. Осталось проверить, что все неравенства линейной задачи $C'$ сохранятся:

- На ребрах из $L_+ \to R_+$ ничего не изменится, так как мы добавили $d$ и вычли $d$, в частности жёсткие ребра останутся жёсткими.
- На ребрах из $L_- \to R_+$ мы уменьшим $u_i + v_j$ на $d$, но уменьшать нам не запрещено, так как неравенство сохранится,
- На ребрах из $L_+ \to R_-$ мы по определению не получим противоречия в неравенствах $u_i + d + v_j \leqslant a_{ij}$, так как $d \leqslant a_{ij} - u_i - v_j$,
- Ребра из $L_- \to R_-$ мы никак не модифицировали, поэтому всё корректно будет и тут.

Поэтому из $L_+$ после пересчёта переменных будет ещё одно жёсткое ребро, которое ведёт в $R_-$, а значит алгоритмом Куна мы строго увеличим количество посещенных вершин.

Бесконечно этот процесс не может происходить, так как мы не можем строго увеличивать бесконечно количество посещённых вершин. Максимум через $n$ действий мы получим, что $R_-$ состоит из одной вершины, а все вершины левой доли мы посетили, поэтому мы сможем найти удлиняющую цепочку.

## Понижение асимптотики алгоритма до $\mathcal{O}(n^3)$

Сейчас наш алгоритм работает за $\mathcal{O}(n^4)$. Действительно, мы должны добавить $n$ вершин, каждый раз запуская алгоритм Куна от вершины не более $n$ раз, пересчет переменных происходит за $\mathcal{O}(n^2)$ и за $\mathcal{O}(n^2)$ будет работать одна фаза алгоритма Куна, поэтому асимптотика $\mathcal{O}(n^4)$, но, можно и быстрее, давайте поймём как проапгрейдить наш алгоритм.

Ключевая идея: теперь мы будем добавлять в рассмотрение строки матрицы одну за одной, а не рассматривать их все сразу. Таким образом, описанный выше алгоритм примет вид:

- Добавляем в рассмотрение очередную строку матрицы $a$.
- Пока нет увеличивающей цепи, начинающейся в этой строке, пересчитываем переменные.
- Как только появляется увеличивающая цепь, чередуем паросочетание вдоль неё (включая тем самым последнюю строку в паросочетание), и переходим к началу (к рассмотрению следующей строки).

Чтобы достичь требуемой асимптотики, надо реализовать шаги 2-3, выполняющиеся для каждой строки матрицы, за время $\mathcal{O}(n^2)$.

Заметим, что если вершина была достижима из $i$-ой, то после пересчётов переменных она останется достижимой, так как жёсткие ребра остаются жесткими, и пересчет переменных выполняется $\mathcal{O}(n)$ раз.

Также заметим, что для проверки наличия увеличивающей цепочки нет необходимости запускать алгоритм Куна заново после каждого пересчёта переменной. Вместо этого можно оформить light версию обхода Куна: после каждого пересчёта переменной мы просматриваем добавившиеся жёсткие рёбра и, если их левые концы были достижимыми, помечаем их правые концы также как достижимые и продолжаем обход из них.

Введём также величину $c_j$ для всех $j$:

$$c_j = \min_{i \in L_+}(a_{ij} - u_i - v_j)$$

Тогда наша $d = \min\limits_{j \in R_-} c_j$. Это и есть в точности те вершины правой, которые мы ещё не посетили, соединенных с вершинами левой доли $L_+$. Это делается за $\mathcal{O}(n)$.

Теперь легко изменять $c_j$ при расстановке потенциалов — надо просто вычесть $d$ из всех $c_j, j \in R_-$. Если мы добавляем в левой доли вершину $k$, надо лишь обновить все $c_j = \min(c_j, a_{kj} - u_k - v_j)$. Это делается за $\mathcal{O}(n)$.

А инициализировать $c_j$ надо при добавлении $i$-ой вершины, как $c_j = a_{ij} - u_i - v_j$, так как пока только 1 вершина посещена — эта вершина на $i$-ой фазе алгоритма.

Теперь поймём, почему теперь алгоритм работает за $\mathcal{O}(n^3)$. Есть, как и в прошлой версии, внешняя фаза — добавление вершин левой доли за $\mathcal{O}(n)$. Теперь обновление переменных выполняется за $\mathcal{O}(n)$, обновление массива $c_j$ происходит при каждом обновлении за $\mathcal{O}(n)$, что даёт $\mathcal{O}(n^2)$ на каждом шаге. Плюс ещё мы должны во внешней фазе запустить алгоритм Куна, чтобы найти паросочетание (мы уже уверены, что оно есть), что тоже $\mathcal{O}(n^2)$. В итоге $\mathcal{O}(n^3)$. Приведем псевдокод:

**Algorithm 1** Венгерский алгоритм

```
1:  M ← ∅                                   ▷ Поддерживающееся паросочетание на жёстких рёбрах
2:  for i ← 1 to n do
3:      c_j ← a_ij − u_i − v_j                                          ▷ По всем j
4:      flag ← true
5:      while flag do
6:          d ← min c_j                          ▷ По всем j ∈ R_− — надо найти это самое d
7:          u_k ← u_k + d                                  ▷ По всем k ∈ L_+ — см. выше
8:          v_k ← v_k − d                                  ▷ По всем k ∈ R_+ — см. выше
9:          c_k ← c_k − d                                  ▷ По всем k ∈ R_− — см. выше
10:         for k ∈ R_− do
11:             if c_k = 0 then                    ▷ Значит вершина стала достижимой
12:                 if k ∈ Right(M) then   ▷ Вдруг, наша вершина оказалась уже в паросочетании, то мы ещё не можем увеличить
13:                     L_+ ← L_+ + Left_neighbour(k)   ▷ Добавляем посещенную в L_+, которая насыщена паросочетанием M
14:                     R_+ ← R_+ + k          ▷ Суммарно добавлений будет не более O(n)
15:                     q ← Left_neighbour(k)
16:                     c_p ← min(c_p, a_qp − u_q − v_p)   ▷ надо обновить после добавления по всем p ∈ R_−
17:             else
18:                 flag ← false     ▷ уже точно знаем, что можем увеличить паросочетание
19:                 break                ▷ надо выйти вообще, чтобы запустить алгоритм Куна
20:     M ← kunh_algo(i)
```

Действительно, в $L_+$ и $R_+$ мы добавляем не больше, чем $\mathcal{O}(n)$ раз, значит и 16 строка выполнится не более $\mathcal{O}(n)$ раз, значит в цикле с 5 строки мы в целом будем выполнять не более $\mathcal{O}(n^2)$ действий. Также алгоритм Куна будет выполняться не более $\mathcal{O}(n^2)$ действий на каждой итерации. А значит общая асимптотика равна $\mathcal{O}(n^3)$.

Можно немного сэкономить на практике и не писать алгоритм Куна, а запоминать, из какой вершины левой доли мы пошли в правую, а вместо алгоритма Куна делать всего $\mathcal{O}(n)$ действий, идя обратно по пути.

**Байка от Глебаса:** Была одна команда на ACM, вроде из Саратова. На финал можно было принести с тобой несколько листов заготовленного кода. Одна из задач была на венгерский алгоритм, и у команды была распечатка по этому алгоритму. В итоге она у них не заходила, потому что когда они тестили у себя в контесте, видимо, тесты были слабые. Мораль: ставьте asserты везде, где можно, чтобы убедиться, что этот алгоритм работает корректно.

## Венгерский алгоритм на прямоугольной матрице

Будем считать, что $n \leqslant m$, иначе просто транспонируем матрицу.

Здесь всё тоже самое, только давайте теперь оценим время работы, если у нас матрица размера $n \times m$. Заметим, что в алгоритме мы будем добавлять не больше, чем $\min(n, m) = n$ вершин, так как мы либо ничего не делаем, либо добавляем по одному элементу к $L_+$ и $R_+$ одновременно. Поэтому тут $\mathcal{O}(n)$, но 16 строку мы будем обновлять за $\mathcal{O}(m)$, поэтому внешний цикл с 5-ой строки будет выполняться не более $\mathcal{O}(nm)$ действий, алгоритм Куна тоже будет выполняться за $\mathcal{O}(nm)$ (максимальное количество рёбер столько). Поэтому асимптотика будет равна $\mathcal{O}(n^2m)$, что может быть применимо для больших $m$ и $n$ поменьше.

Также для совсем маленьких $n$, можно оставить лучших $n$ в каждой строке (ведь если есть решение, что в данной строке кто-то не из $n$ лучших, то остальные работники забрали не более $n - 1$ столбец в матрице на свои лучшие ответы, а значит по принципу Дирихле мы можем взять получше ответ для этой строки). Поэтому матрица осталась максимум $n \times n^2$, значит такой алгоритм будет работать за $\mathcal{O}(n^4 + nm)$.