Федеральное государственное автономное учреждение
высшего профессионального образования

Московский Физико-Технический Институт
КЛУБ ТЕХА ЛЕКЦИЙ

Алгоритмы.
ИВТ.

III СЕМЕСТР

Лекторы: *А. И. Гришутин. И. Д. Степанов.*

Автор: *А. Ш. Ильдаров.*
Проект на overleaf
Проект на github

Осень 2020 года

Содержание

1 Паросочетания. Минимальное вершинное покрытие.

Определение 1. Двудольный граф – граф, множество вершин которого можно разбить на две части таким образом, что каждое ребро графа соединяет какую-то вершину из одной части с какой-то вершиной другой части, то есть не существует рёбер между вершинами одной и той же части.

Определение 2. Хроматическое число – минимальное число цветов, в которые можно раскрасить вершины графа так, чтобы концы любого ребра имели разные цвета.

Определение 3. Паросочетание (англ. matching) в двудольном графе — произвольное множество рёбер двудольного графа, такое что никакие два ребра не имеют общей вершины.
Мощность паросочетания – количество ребер в нем.
Максимальное паросочетание – мощность которого *наибольшая* среди всех возможных паросочетаний в данном графе.

Определение 4. Цепь – некоторый простой путь (т.е. не содержащий повторяющихся вершин или рёбер).
Чередующаяся цепь – цепь, в которой рёбра поочередно принадлежат/не принадлежат паросочетанию.
Увеличивающая цепь – чередующаяся цепь, в которой начальное и конечное ребра *НЕ* принадлежат паросочетанию.
Уменьшающая цепь – цепь, в которой начальное и конечное ребра принадлежат паросочетанию.

Определение 5. Вершины двудольного графа, инцидентные рёбрам паросочетания M, называются **насыщенными** или покрытыми.

Алгоритм Куна.

Теорема 1.1. *Паросочетание M в двудольном графе G – max $\iff$ в G нет увеличивающей цепи относительно M.*

Доказательство.
$\Rightarrow$:
От противного: Пусть в G с максимальным паросочетанием M существует увеличивающая цепь.

Тогда заменив в ней все рёбра, входящие в паросочетание, на невходящие и наоборот, мы получим большее паросочетание.
То есть M не являлось максимальным. Противоречие.
$\Leftarrow$:
Пусть M – не max, покажем что $\exists$ увеличивающая цепь.
Пусть M' – паросочетание: $|M'| > |M|$
Построим подграф $G' = M \bigoplus M'$, состоящий из ребер $\in$ только одному из паросочетаний.
M и M' – паросочетания $\Rightarrow$ нет вершин, которые смежны с двумя ребрами из паросочетания. То есть у каждой вершины подграфа есть не более одного ребра из M и не более одного из M'. $\Rightarrow \forall v \in G' \hookrightarrow deg(v) \leq 2$
Как известно, графы с таким свойством степеней вершин предстваляют из себя наборы цепей и циклов.
При этом длина цикла должна быть четной, ведь иначе мы будем иметь вершину, у которой два ребра, к ней смежных, принадлежат одному паросочетанию.
В циклах поровну ребер из каждого паросочетания, значит их вклад в отрыв M' от M по числу ребер – нулевой.
Значит обогнать M у M' получится только если в графе имеется цепочка нечетной длины, у которой начальное и конечное ребра лежат в M'. Вот эта вот цепочка и есть увеличивающая. □

1.1 Алгоритм Куна для поиска максимального паросочетания.

1. Фиксируем доли графа: L и R. Изначально считаем паросочетание пустым.
2. В доле L перебираем вершины в порядке увеличения номеров.
3. Если вершина насыщена, то пропускаем ее и идем дальше. В противном случае, пытаемся насытить вершину, запустив поиск увеличивающей цепи из этой вершины следующим образом:
   - 3.1. Стоя в текущей вершине v доли L, просмотрим все ребра из этой вершины.
   - 3.2. Возьмем текущее ребро (v, t_0): Если t_0 – ненасыщена, то одно ребро (v, t_0) задает нам увеличивающую цепь, просто увеличим паросочетание с его помощью и прекратим поиск. Если же t_0 насыщена каким-то ребром (t_0, p), то пойдем вдоль этого ребра, уже в поисках увеличивающейся цепи, проходящей через ребра (v, t_0) и (t_0, p). Для этого просто перейдем в вершину p и продолжим обход из нее.

4. В конечном итоге, мы либо найдем увеличивающую цепь из вершины v и увеличим паросочетание этой цепью, тем самым насытив вершину, либо же покажем отсутствие увеличивающей цепи и невозможность насыщения вершины.

5. Возьмем следующую по порядку вершину доли L и повторим.

6. После того как мы просмотрим все вершины, пытаясь увеличить паросочетание цепью из них, мы получим максимальное паросочетание.

Минимальное вершинное покрытие.

Определение 6. Вершинное покрытие графа G=(V,E) - такое подмножество S множества вершин графа V, что любое ребро этого графа инцидентно хотя бы одной вершине из множества S.

Определение 7. Минимальное вершинное покрытие графа – вершинное покрытие, состоящие из *наименьшего* числа вершин.

Утверждение 1.2. $|M_{max}| \leq |C_{min}|$. *Чтобы покрыть все ребра, нам нужно вершин не меньше, чем мощность наибольшего паросочетания.*

Доказательство.
Паросочетание это, как известно, набор непересекающихся рёбер. Мы хотим взять набор вершин, эти ребра покрывающий. Ясно что каждая вершина может покрыть не более одного ребра, ведь они не пересекаются по концам. Оценка снизу доказана. □

Теорема 1.3. *(Кёнига)*
В двудольном графе $|M_{max}| = |C_{min}|$.

Доказательство. Явно предъявим покрытие, размер которого равен размеру максимального паросочетания.

1. Разделим граф на доли L и R.

2. Сориентируем ребра: Из M_{max} $\longleftarrow$, остальные $\longrightarrow$

3. Обойдем граф из *ненасыщенных* вершин $\in$ L.

4. Получим разбиение графа на четыре множества: L^+, R^+ – вершины, лежащие в L или R соответственно, которые доступны из ненасыщенных вершин, лежащих в L. L^-, R^- – аналогично *недоступные* из ненасыщенных вершин, лежащих в L.

5. Поймем какие ребра бывают между каждыми из четырех множеств:

   5.1. Покажем, что ребер $L^+ \to R^-$ не бывает, ведь если бы такое ребро имелось, мы из достижимой вершины, лежащей в L^+ добрались бы по этому ребру до вершины, по определению R^-, недостижимой.

   5.2. Из аналогичных рассуждений понятно что не бывает ребер $L^- \to R^+$. Это не отрицает существование ребер $(R^+ \to L^-)$ тт

   5.3. Покажем отсутствие ребер $R^- \to L^+$: От противного: Пусть (r^-, l^+) – такое ребро. Это ребро вида $\longleftarrow$, то есть принадлежащее паросочетанию.
   Вершина l^+ – насыщена, значит чтобы до нее дойти мы должны были начать обход из какой-то другой вершины l', из которой l^+ достижима.
   Так как l' и l^+ лежать в одной доле двудольного графа, маршрут из l' в l^+ в какой-то момент должен пойти справа налево, то есть имеется ребро вида (r', l^+), которое в силу своего направления также лежит в паросочетании. Значит из смежные ребра (r^-, l^+) и (r', l^+) лежат в одном паросочетании. Противоречие.

6. Получаем что любое ребро графа инцидентно или вершине из L^- или вершине из $R^+ \Rightarrow L^- \cup R^+$ – вершинное покрытие.

7. Покажем, что все вершины из $L^- \cup R^+$ насыщены ребрами из паросочетания:
   Это так, ведь если бы в L^- были ненасыщенные вершины, мы бы запускали из них обход и попадали бы в L^+, чего быть не может в силу отсутствия ребер $L^- \to R^+$ и если бы в R^+ была бы ненасыщенная вершина, то существовала бы увеличивающая цепь, в этой вершине заканчивающаяся, что противоречит максимальности паросочетания.

8. Как известно, не существует ребер $L^- \to R^+$, которые бы принадлежали паросочетанию, а значит каждому ребру паросочетания инцидентна ровно одна вершина из покрытия $L^- \cup R^+$, а значит $|M| = |L^- \cup R^+|$, и это вершинное покрытие является наименьшим из оценки.

□

2 Потоки

Определение 8. Сеть – ориентированный граф $G = (V, E)$, с двумя выделенными различными вершинами: s – исток и t – сток, а также с функцией $cap : E \to \mathbb{Z}_{>0}$.

Определение 9. Поток в сети – функция $f : V \times V \to \mathbb{Z}$:

1. $\forall u, v \in V \hookrightarrow f(u, v) \leq cap(u, v)$
2. $\forall v \in V \setminus \{s, t\} \hookrightarrow \sum\limits_{\substack{u \\ (u,v)\in E}} f(u, v) = \sum\limits_{\substack{w \\ (v,w)\in E}} f(v, w)$
3. $\forall u, v \in V \hookrightarrow f(u, v) = -f(v, u)$

Замечание. Свойства 2 и 3 можно объединить в следующем виде $\forall v \in V \hookrightarrow \sum\limits_{u\in V} f(v, u) = 0$

Определение 10. Остаточная сеть G_f сети G с потоком f – сеть, у которой модифицируются пропускные способности: $cap_f(u, v) = cap(u, v) - f(u, v)$, и удаляются ребра с нулевой cap_f.

Определение 11. Величина потока $|f|$– то сколько "вытекает" из s – сумма исходящих из s потоков.

Определение 12. Разрез сети G – пара подсетей (S, T): $s \in S, t \in T, S \cap T = \emptyset, S \cup T = G$
$cap(S, T) = \sum\limits_{\substack{u\in S \\ v\in T}} cap(u, v)$
$f(S, T) = \sum\limits_{\substack{u\in S \\ v\in T}} f(u, v)$

Лемма 2.1. (S, T) – *разрез* $\Rightarrow |f| = f(S, T)$

Доказательство. На семинаре. □

Лемма 2.2. (S, T) – *разрез* $\Rightarrow f(S, T) \leq cap(S, T)$

Доказательство. $\forall u, v \in V \hookrightarrow f(u, v) \leq cap(u, v) \Rightarrow$ чтд. □

Теорема 2.1. *(Форда – Фалкерсона) Следующие утверждения эквивалентны:*

1. f *– max поток в* G
2. *в* G_f *нет пути из* s *в* t
3. $|f|$ *= cap некоторого разреза.*

Доказательство.
$1 \Rightarrow 2$: От противного:
Пусть f – max, но в G_f есть путь $s \to t$.
Рассмотрим $x = min(cap_f) > 0$ на этом пути.
Значит вдоль пути можно пустить x единиц потока $\Rightarrow$ в исходной сети можно было пустить на x единиц больше $Rightarrow$ f – не max.

$2 \Rightarrow 3$:
Пусть $S =$ множество вершин, доступных из s в G_f, $t \notin S$
$T = V \setminus S$.
По определению (S, T) – разрез.
Покажем что $|f| = cap(S, T) = \sum\limits_{\substack{u \in S \\ v \in T}} cap(u, v)$.
Рассмотрим ребро (u, v): $u \in S, v \in T$
u – доступно в G_f, v – нет $Rightarrow$ оно удаляется в остаточной сети $Rightarrow$
$f(u, v) = cap(u, v) Rightarrow \sum\limits_{\substack{u \in S \\ v \in V}} cap(u, v) = \sum\limits_{\substack{u \in S \\ v \in V}} f(u, v) \Rightarrow$
$f(S, T) = |f| = cap(S, T)$.

$3 \Rightarrow 1$:
$\exists (S, T) : \ |f| = cap(S, T)$.
При этом $\forall (S, T)$ – разрез $\hookrightarrow f(S, T) \leq cap(S, T) \Rightarrow f$ – max. □

2.1 Алгоритм Форда – Фалкерсона для поиска максимального потока.

1. Изначально поток равен 0.
2. Пока в G_f есть путь из s в t:
   2.1. $x = min(cap_f$ на этом пути $) > 0$
   2.2. Увеличим поток f на x и изменим остаточную сеть соответствующим образом.

$|f|$ увеливается на целое положительное число и ограничен сверху, значит алгоритм закончится. Асимтотика – $O(ans \times |E|)$

2.2 Алгоритм Эдмондса – Карпа для поиска максимального потока.

Та же самая идея, только теперь вместо произвольного пути будем выбирать кратчайший путь по ребрам. По сути, DFS меняется на BFS.

1. Изначально поток равен 0.
2. Пока в G_f есть *минимальный* путь из s в t:
   2.1. $x = min(cap_f$ на этом пути $) > 0$.
   2.2. увеличим поток f на x и изменим остаточную сеть соответствующим образом.

Алгоритм будет иметь асимптотику $O(|V| \times |E|^2)$

Докажем это.

Определение 13. $dist(u, v)$ – минимальное число ребер между вершинами.

Лемма 2.3. *Пусть поток f' получается из потока f после одной итерации алгоритма Эдмондса – Карпа.*
Пусть $dist'(u, v)$ – кратчайшее расстояние между ребрами в $G_{f'}$, тогда $\forall v \in V\{s, t\} \hookrightarrow dist'(s, v) \geq dist(s, v)$.
То есть с каждой итерацией расстояние не убывает.

Доказательство. От противного:
Пусть $\exists v \in V \setminus \{s, t\} : \ dist'(s, v) < dist(s, v)$ и при этом $dist(s, v)$ – минимальное возможное
Пусть u – предыдущая вершина перед v на кратчайшем пути из s в v в $G_{f'}$
Тогда $dist'(s, u) = dist'(s, v) - 1$ (по определению dist).
Также $dist'(s, u) \geq dist(s, u)$ ведь мы выбрали минимально удаленное v.
Рассмотрим 2 случая:
$(u, v) \in E_f$, тогда:
$dist(s, v) \leq dist(s, u) + 1 \leq dist'(s, u) + 1 \leq dist'(s, v)$
Но при этом $dist'(s, v) < dist(s, v)$
Противоречие.

$(u, v) \notin E_f$, но известно что $(u, v) \in E_{f'}$.
Появление ребра (u, v) после увеличения потока означает увеличение потока по обратному ребру (v, u). Увеличение потока производится вдоль кратчайшего пути, а значит вершина

v – предыдущая перед u на пути из s в u
Это значит что $dist(s,v) = dist(s,u) - 1 \leq dist'(s,u) - 1 = dist'(s,v) - 2$, но ведь $dist'(s,v) < dist(s,v)$. Противоречие.

□

Определение 14. Насыщенное – ребро, вдоль которого идет поток, равный его capacity.

Лемма 2.4. *Любое ребро сети насыщается не более $O(|V|)$ раз, то есть в алгоритме $O(|V||E|$ итераций, то есть каждая итерация насыщает хоть 1 ребро.*

Доказательство.
Рассмотрим ребро (u,v) в момент его насыщения.
Если оно насыщается, то оно лежит на кратчайшем пути от s до v (так работает наш алгоритм).
Значит $dist(s,u) + 1 = dist(s,v)$.

Теперь посмотрим, когда оно могло насытиться еще один раз:
Оно сначала должно перестать быть насыщенным, для этого нужно отменить поток вдоль (u,v), то есть пустить поток вдоль (v,u)
Пусть $dist'$ – расстояние в момент, когда пускается поток вдоль (v,u).
$dist'(s,v) + 1 = dist'(s,u)$ (Так как проталкивание происходит вдоль кратчайшего пути).
Со временем расстояния только увеличиваются, значит $dist'(s,v) \geq dist(s,v) + 1 = dist(s,u) + 2$
Таким образом от момента насыщения, до момента повторного насыщения расстояние $dist(s,u)$ должно вырасти по меньшей мере на 2.
Однако все расстояния ограничены $O(|V|)$, значит и насыщений может быть только $O(|V|)$. □

3 Продложение потоков. Алгоритм Диница, теоремы Карзанова

3.1 Детали реализации.

```
struct Edge {
  int from = 0,
      to   = 0;

  long long cap = 0;
  long long flow = 0;
};

auto edges = vector<Edge>();
auto outbound = vector<vector<Edge>>();
/* v-th element -- numbers of edges from v. */
```

Листинг 1: Удобная реализация ребер

Рядом с каждым ребром всегда есть обратное, изначально с нулевой capacity. Будем присваивать обратному ребру номер исходного, увеличенный на 1. К примеру номера $2n$ и $2n + 1$. Тогда при добавлении нового ребра в граф, мы будем добавлять его и сразу же добавлять обратное, а потом сохранять номера обоих ребер в массив edges.

Теперь если мы пускаем поток по ребру с номером i, то можем удобно уменьшить поток по противоположному ребру, как имеющее номер $i \oplus 1$

```
edges[i].flow  += delta_flow;
edges[i ^ 1].flow -= delta_flow;
```

При реализации алгоритма Форда-Фалкерсона будем просто выполнять DFS на ребрах, у которых cap $>$ flow.

3.2 Блокирующий поток.

Определение 15. Блокирующий – поток f в сети G, такой что в сети G (но вовсе не обязательно G_f) больше нет пути из s в t, вдоль которого можно протолкнуть поток.

Пример. Даю установку: Вы видите рисунок графа.

Замечание. Блокирующий поток совершенно не обязательно наибольший.

3.3 Поиск блокирующего потока

За $O(nm)$

Забудем что у нас бывают обратные ребра.

Идея проста, будем пускать поток пока получается, а для оптимизации учтем тот факт, что если мы уже однажды пытались пойти вдоль ребра, и узнали что после прохода по нему дойти до t не получится, то нам не стоит идти по этому ребру снова. Это аналог метки used в DFS.

```
auto first_worthy_edge_num = vector<int>();
/*first_worthy_edge[v] -- relative number of the first edge
              which is worth considiring from v. */

long long dfs(int v, long long flow){
//If we've followed the path and we can push the flow, we do it.

  if (v == t) return flow;

//While there is at least one worthy edge.
  while (first_worthy_edge_num[v] < outbound[v].size()){
    auto cur_e = g[v][first_worthy_edge_num[v]];

  /*If there is some additional flow we can push and the edge is
          presented in initial network (is not reverse). */

  //We will code edge presence checker in Dinic's algorithm.
  if (edges[e].capacity > edges[e].flow && e is not reverse edge){
      auto next_flow = dfs(edges[e].to,
                           min(flow, edges[e].cap - edges[e].flow));


      if (next_flow > 0){
        edges[e].flow  += next_flow;
        edges[e ^ 1].flow -= next_flow;

        return next_flow;
      }
    }

    ++first_worthy_edge_num[v];
  }

```

```
34    return 0;
35  }
```

Тогда сам поиск блокирующего потока будет иметь вид

```
1   while ((auto x = dfs(s, inf)) != 0){
2     blocking_flow += x;
3   }
```

Алгоритм работает за $O(nm)$, ведь если главный вызов dfs (из main) суммарно сдвинул все номера интересных вершин на k, то dfs работал $n + k$, ведь каждое ребро либо нас устроило, и мы просто увеличили номер, либо оно нас устроило, и мы пошли в новую вершину, а сдвигать указатели долго мы не сможем, суммарно сдвигов будет не больше чем m, а значит время работы $= n*$ кол-во запусков $+m$, а каждый запуск насыщает хоть 1 ребро, при этом из-за отсутствия обратного ребра, ребра не рассыщаются. Значит имеем асимптотику $O(nm)$

3.4 Алгоритм Диница для поиска наибольшего потока

За $O(n^2m)$

Пусть G – сеть.
$dist(s, v)$ – кратчайшее расстояние между вершинами.

Определим слоистую сеть:
Вершина s.
Вершины с $dist$ 1 от s.
...
Вершины на расстоянии k от s.
Вершина t.

При этом оставим только ребра, идущие из меньшего уровня в следующий по порядку, то есть кратчайшие пути.

Даю установку: Вы видите рисунок слоистой сети.

Алгоритм
Вспомним что у нас бывают обратные ребра.
Пока не найден наибольший поток:

1. Построим слоистую сеть.

2. Пустим в ней произвольный блокирующий поток.

Утверждение 3.1. *Алгоритм Диница находит наибольший поток.*

Доказательство.
Теорема Форда-Фалкерсона учит нас что поток максимален $\iff$ в остаточной сети нет увеличивающего пути. Если наш алгоритм завершился, то есть не сумел пустить блокирующий поток в слоистой сети, значит в слоистой сети нет пути из s в t, а по построению слоистой сети, это значит что и в исходной сети пути из s в t. □

Утверждение 3.2. *Алгоритм Диница делает не более n итераций.*

Доказательство.
Покажем что после каждой итерации $dist'(s,t) > dist(s,t)$.
На каждом пути из s в t есть хоть одно насыщенное ребро, ведь мы пустили блокирующий поток. (По определению блокирующего потока)
После того как мы пустили блокирующий поток, у в слоистой сети исчезли некоторые ребра слева направо, а обратные ребра справа налево наоборот появились, значит расстояние в слоистой сети увеличилось, а значит увеличилось и расстояние в исходной сети. □

3.5 Теоремы Карзанова.

Зачастую асимптотика алгоритма Диница завышена, так как сеть в задаче имеет специфичный вид. Теоремы Карзанова помогают точнее оценить число итераций.

Определение 16. $C_out(v) = \sum\limits_{u \in V} cap(v,u)$ $C_in(v) = \sum\limits_{u \in V} cap(u,v)$

Замечание. Понятно что тогда через любую вершину течет не более чем $min(C_{in}(v), C_{out}(v))$

Определение 17. Общий потенциал сети $P = \sum\limits_{v \in V, v \neq s,t} min(C_{out}(v), C_{in}(v))$

Лемма 3.1. *В сети G $l = dist(s,t) \leq \frac{P}{F} + 1$, где F – максимальный поток, P – общий потенциал.*

Доказательство.
Пусть $V_i = \{v : dist(s,v) = i\}$
Получилась слоистая сеть, в которой ребра есть только из V_i в V_{i+1}

Таким образом, если разрез и пересекает какое-либо ребро, то оно идет между слоями
А сумма capacity ребер между слоями не меньше чем максимальный поток.
$cap(\cup_0^i V_k, \cup_{i+1}^l V_k) = \sum\limits_{x \in V_i, y \in V_{i+1}} cap(x, y) \geq F$
Просуммируем это неравенство по всем i:
$\sum\limits_{i=0}^{l-1} \sum\limits_{x \in V_i, y \in V_{i+1}} cap(x, y) \geq (l-1)F$
А сверху эту сумму можно оценить общим потенциалом сети, ведь сумма capacity ребер между слоями не может превышать сумму capacity ребер в одном слое, которая, в свою очередь, не превышает сумму потенциалов вершин в V_i.
$P \geq (l-1)F$ □

Лемма 3.2. *После проталкивания потока f в сети G, общий потенциал остаточной сети G_f равен общему потенциалу исходной сети.*

Доказательство.
Когда мы проталкиваем поток f вдоль ребер (u, v) и (v, w) по обратным ребрам проходит $-f$.
Таким образом уменьшение $C_{in}(v)$ за счет потока по (u, v) будет скомпенсировано увеличением $C_{in}(v)$ за счет увеличения потока по обратному ребру (w, v). □

Теорема 3.3. *Число итераций алгоритма Диница в сети G с целочисленными capacity – $O(\sqrt{P})$*

Доказательство.
Сделаем ровно $\sqrt{P}$ итераций, тогда $l \geq \sqrt{P}$, ведь, как мы уже знаем, каждая итерация увеличивает расстояние между s и t хоть на 1.
Запишем утверждение первой леммы для остаточно сети после $\sqrt{P}$ итераций:
$l \leq \frac{P}{F} + 1$, а $F = F - f$, где f – то сколько потока удалось протолкнуть за прошедшие итерации.
При это $f \geq \sqrt{P}$, ведь все capacity целочисленные, и мы не могли проталкивать меньше 1 единицы потока за итерацию.
Таким образом получаем $F \leq 2\sqrt{P}$ □