# Домашнее задание по Алгоритмам
# Второе домашнее задание

Группа 154-1
Чернышев Даниил

## Задача 1

Реализуйте линейный алгоритм, вычисляющий паросочетание в произвольном неориентированном графе без весов, отличающееся по размеру от наибольшего (по размеру) паросочетания в этом графе не более, чем вдвое. Напомним, что паросочетанием называется подмножество ребер графа, в котором ни у каких двух ребер нет одинаковых вершин. Докажите корректность алгоритма.

*Решение:* Эта задача решается с помощью жадного алгоритма. Возьмем список смежности графа и заведем массив used для пометки использованных вершин. Будем перебирать вершины в списке. Если вершина i, на которую мы смотрим не использована, то смотрим смежные с ней вершины. Если просматриваемая смежная вершина j тоже не использована, то добавляем в паросочетание ребро (i, j), а сами эти вершины пометим как использованные и закончим обход соседей. Если во время обхода все соседи оказались использованными или соседей попросту не оказалось, то просто переходим к i + 1. В итоге получим максимальное паросочетаение $M$.

**Лемма:** Пусть $A$ и $B$ - два максимальных паросочетания. Тогда $|A| < 2|B|$ и $|B| < 2|A|$.

**Доказательство:** Если ребро (u, v) $\in A$, то либо (u, $b_1$)$\in B\backslash A$ и (v, $b_2$)$\in B\backslash A$ либо (u,v) $\in B \cap A$. Но если ребро (u, v) $\in A\backslash B$, то тогда точно (u, $b_1$)$\in B\backslash A$ и (v, $b_2$)$\in B\backslash A$, в силу максимальности $B$. Получаем $|A\backslash B| \leq 2|B\backslash A|$ откуда простыми преобразованиями выводим $|A| = |A \cap B| + |A\backslash B| \leq 2|B \cap A| + 2|B\backslash A| = 2|B|$. Аналогично получается $|B| \leq 2|A|$. □

Из этой леммы получаем, что для наибольшего паросочетания $M_{max}$ и $\forall M$ верно $\frac{1}{2}|M_{max}| \leq |M|$.

**Код:**

```
AdjencyList edges;
vector<bool> used(edges.size(), false);
set <pair<size_t, size_t>> matching;
for (int i = 0; i <= edges.size(); ++i) {
    if (!used[i]) {
        for (auto j : edges[i]) {
            if (!used[j]) {
                matching.insert({ i, j });
                used[i] = true;
                used[j] = true;
                break;
            }
        }
    }
```

```
}
cout << matching.size() << endl;
```

Сложность алгоритма: Очевидно, что в худшем случае алгоритм пройдётся по всем ребрам поэтому сложность будет $O(E)$. □

## Задача 2

Будем считать расстояние между двумя битовыми строками равным числу позиций, в которых значения битов у этих строк различаются. Требуется разбить заданный массив строк на максимально возможное число кластеров так, чтобы расстояние между строками из разных кластеров было не меньше трех. Предложите и реализуйте эффективный алгоритм, решающий эту задачу, докажите его корректность и оцените время работы. Алгоритм должен вычислять число кластеров, сами кластеры возвращать не нужно.

*Решение:* Начнем с того, что расстояние Хэмминга между любыми двумя строками из разных кластеров должно быть более 3, а значит задача заключается в том, чтобы объединить все кластеры у которых это условие не выполняется. Есть несколько подходов к решению этой задачи. Первоначально все строки состоят в своих собственных кластерах и хранятся в хэш–структуре, чтобы сделать операцию поиска и удаления константной. Также, все кластеры хранятся в Disjoint-Set-Union с эвристикой сжатия пути и ранговой эвристикой. Такая структура позволяет выполнять операции объединения, поиска и добавления за почти константное время(обратная функция Аккермана $\alpha(n)$, которая растет с очень низкой скоростью, на практике она меньше 5). Поскольку сложность операций используемых структур константная, то опустим их в подсчете общей сложности алгоритмов.

Наивный подход - просто вычислить все расстояния Хэмминга и все строки, между которыми расстояние менее трех и состоящие в разных кластерах, объединить в общий кластер. Сложность такого решения составит $O(n^2)$. Такое решение будет неэффективным как по памяти так и по времени.

Поскольку мы знаем минимальное расстояние между строками, то мы просто можем просмотреть всех соседей(строки лежащие в радиусе 2) каждой из заданных строк и если эти соседи есть в списке заданных, то в случае принадлежности к разным кластерам объединяем их кластеры. Всего соседей будет $\binom{m}{1} + \binom{m}{2}$, что при $n \geq m$ будет меньше $n^2$. Тогда общая сложность алгоритма составит $O((\binom{m}{1} + \binom{m}{2}) \cdot n)$.

**Доказательство корректности:** Пусть алгоритм некорректен. Тогда найдутся два таких кластера, в которых существуют две строки, расстояние между которыми менее 3 и при этом не лежащих в едином кластере. Это также означает, что эти строки соседи, а значит перебор соседей от любой из них выдал другую. Но по алгоритму если две строки соседи и они не лежат в одном кластере, то их кластеры объединяются. Этого не произошло, а значит перебора для этих строк не было. Поскольку перебор выполняется для всех заданных строк, то получаем, что эти две строки не были заданны. Но это значит, что их не существует ни в одном кластере. Противоречие. □

Можно ускорить предыдущий алгоритм. Если между двумя строками расстояние 2, то у них есть общий сосед, до которого расстояние 1 и через который лежит путь–ребро длины 2. Очевидно, что этот сосед может быть не задан. Тогда для каждой заданной строки рассмотрим соседей в радиусе 1 и для всех не заданных соседей отметим эту строку как корень(то есть при просмотре этого соседа мы будем получать указатель на кластер первой заданной строки, которая его нашла, таким образом давая ему мнимую заданность), а если сосед задан, то просто объединим кластеры. В итоге нам надо будет просмотреть только $\binom{m}{1} = m$ соседей, что дает нам сложность $O(m \cdot n)$. Но стоит отметить, что возрастет

потребление памяти, так как нужно будет использовать дополнительную структуру основанную на хэшах (сохраняет ассимтотическую сложность) для хранения всех найденных соседей и их корней (это дает прибавку к потреблению памяти в $O(\text{хэш–структура}(n \cdot m))$).

**Доказательство корректности:** Пусть алгоритм некорректен. Тогда найдутся два таких кластера, в которых существуют две строки, расстояние между которыми менее 3 и при этом не лежащих в едином кластере. Это означает, что у них есть общий сосед. Этот сосед должен содержать ссылку на корень, полученную при первом просмотре, то есть считаться как член кластера одной из этих вершин, что означает, что при втором просмотре кластер смотрящей строки должен был объединится с кластером корня. Но этого не произошло, а значит, что либо второго либо первого просмотра не было. Это приводит к тому, что либо одна из этих вершин не просматривала соседей либо обе, но поскольку просмотр соседей запускается от всех заданных первоначально строк, то получаем, что либо они мнимозаданны либо они вообще не были заданны. Но тогда выходит, что ни один из кластеров не содержит их. Противоречие. □

Существует реализация без использования Disjoint-Set-Union. Создадим очередь и добавим в неё первый элемент хэш–структуры. Запустим от него перебор соседей. Если сосед оказался заданной строкой, то добавим его в очередь на обработку. Закончив перебор удалим первый элемент из хэш–структуры и запустим перебор от следующей строки в очереди. Когда очередь закончится, добавим в неё первый элемент хэш–структуры и продолжим перебор соседей. В итоге количество кластеров будет равняться количеству добавлений элементов из хэш–структуры в очередь или количеством запусков PseudoBFS'а(так можно назвать весь выполняемый процесс от начала до конца очереди), поскольку каждый запуск будет обрабатывать все вершины компоненты связности, то есть все строки в одном кластере. Сложность такого алгоритма составит $O(((\binom{m}{1} + \binom{m}{2}) \cdot n)$, так как мы все равно переберём для каждой заданной строки всех возможных соседей.

**Доказательство корректности:** Пусть алгоритм некорректен. Тогда найденное число кластеров либо больше настоящего ответа либо меньше. Если больше, то это означает, что в нескольких PseudoBFS'ах компонента была исследована не полностью. Это означает, что не все ребра длины меньше 3 были обработаны. Так как перебор соседей гарантировано выдает все соседние вершины, то получается, что он просто на просто был вызван не для всех. Но он вызывается для каждого найденного соседа. В итоге по индукции по итерациям PseudoBFS'а приходим к тому, что он не вызывался для стартовой вершины перебора. Но перебор первоначально вызывается для элементов из хэш–структуры заданных строк, то есть получается, что он вызывается для всех заданных строк. Тогда выходит, что стартовая вершина перебора не является заданной строкой, а значит этого кластера не существует. Противоречие.

Теперь, если ответ меньше реального, то это означает, что PseudoBFS обошел больше вершин чем в компоненте связности. Но это невозможно так как он исследует компоненту по её ребрам, то есть он не сможет попасть в строку, ребро до которой имеет длину более 2. □

Важно заметить, что быстрее всего алгоритмы работают если строки переводить в целые числа, а потом хэшировать, так как инвертирование битов лучше работает с целочисленными данными чем со строками(там вообще получается очень затратный процесс перебора). □