# 1 Алгоритм Йена

1. Формулировка: пусть все простые пути из $s$ в $t$ отсортированы по весу, нужно найти первые (минимальные) $K$.

2. Для начала научимся искать второй минимальный путь. Найдем первый алгоритмом Дейкстры. Второй где-то отличается от первого. Т.е. он сперва повторяет его префикс, а теперь сворачивает, а дальше как-то идет к $t$. Давайте переберем место, в котором второй путь будет отличаться от первого. Пусть первый путь $= v_1, v_2, \cdots, v_k$. Скажем, что вершины $v_1, v_2, \cdots, v_i$ мы прошли, а затем пошли в любую другую вершину, но не $v_{i+1}$. Для этого запустим Дейкстру из вершины $v_i$, запретив ее проходить по вершинам $v_1, v_2, \cdots, v_i$ (там мы уже были) и проходить по ребру $(v_i, v_{i+1})$. Из полученных путей выберем минимум.

3. На самом деле в предыдущем пункте мы разбили все простые пути из $s$ в $t$ на классы и в каждом классе выбрали минимальный элемент. Класс путей — все пути начинающиеся на $v_1, v_2, \cdots, v_i$ для фиксированного $i$ и не проходящие по ребру $(v_i, v_{i+1})$. Классы не пересекаются и в объединении дают все пути кроме минимального (его мы уже выбрали). Чтобы находить $K$-й путь нужно в каждый момент времени хранить все классы, на очередном шагу выбирать минимальный путь, и класс, к которому относится выбранный путь, разбивать на более мелкие классы (это делается так же, как когда мы искали второй путь). При этом запрещенными могут оказаться уже несколько ребер.

4. Всего за $K$ итераций у нас появится $O(KV)$ классов. Оценка времени работы алгоритма: $O(KV \cdot Dijkstra)$ Оценка памяти: $O(KV \cdot V)$. Т.к. для хранения одного класса нужно исользовать $O(V)$ памяти. Память можно сократить до $O(K \cdot V)$, т.к. на самом деле не нужно хранить все классы, достаточно хранить только $K$ минимальных.

5. Применение Йена для алгоритма Краскала (MST). Отсортируем один раз в самом начале ребра (сортируем по возрастанию веса). Теперь остовное дерево — последовательность ребер. Мы какие-то $i$ первых можем точно взять, $i+1$-е запретить брать, а теперь сперва насильно добавить ребра, которые мы точно берем, а для оставшихся запустить Краскала (ребро берем, если оно не запрещено и не образует цикл).

6. Применение Йена для алгоритма Эдмондса (matching). Занумеруем все ребра от 1 до $E$. Теперь в любом паросочетании ребра можно упорядочить по номерам ребер. Применяем ту же идею: первые $i$ берем, $i+1$-е ребро запрещаем, для оставшегося графа запускаем Эдмондса.

# 2 Динамика на ацикличном графе

1. Динамическое программирование (*Динамика*) это всегда задача на ацикличном графе. Если граф содержит циклы, то применяют или Дейкстру, или Форд-Беллмана, или поиск в ширину. Исключением являются графы с простой структурой, где наличие циклов можно обойти простыми хаками.

2. Стандартные задачи, которые решает динамическое программирование, с точки зрения графов таковы: суммарное число путей, минимальный путь, максимальный путь, все из $s$ в $t$, все за время $O(E)$.

3. В прошлом пункте мы предположили, что $s$ и $t$ — вершины. Бывает много конечных и начальных состояний — множества вершин $S$ и $T$. Алгоритм от этого сложнее не становится, меняется база динамики.

4. Пусть на каждом ребре написана буква, тогда каждому пути соответствует строка. Лексикографически минимальный (*LexMin*) путь из $s$ в $t$ можно легко найти за $O(VE)$ (функция динамики = строка).

5. Если для каждой вершины символы, написанные на ее исходящих ребрах различны, то можно жадно перебирать символы в порядке возрастания, а строку целиком не хранить, только ссылку на первое ребро. Получится решение за $O(E)$.

6. Пусть мы хотим найти путь минимальной длины (веса = 1, `bfs`), а из уже таких — LexMin. В этом случая также есть решение за $O(E)$, а именно такое: `bfs` разбил граф на слои (слой — это вершины с одинаковым расстоянием до конца). Кратчайший путь — любой путь по слоям вперед. Будем вести динамику по слоям. Наша цель — занумеровать вершины целыми числами от 1 до $N$ так, чтобы сравнение на больше, меньше, равно строк вершин было равносильно сравнению номеров вершин. Пусть для слоя $i+1$ мы уже знаем номера вершин. Чтобы посчитать номера для $i$-го слоя, нужно увидеть, что строка из вершины в $t$ это пара [`первый символ`, `номер вершины из слоя i+1`]. Отсортируем и занумеруем эти пары. Конец.

7. В произвольном случае задача LexMin имеет решение за время $O(E \log V)$. Первая часть решения: не хранить для вершины всю строку, а только ссылку на первый символ. Сравнивать строки мы умеем все еще только за $O(V)$, а вот лишней памяти мы уже не используем. К тому же *ссылки на первый символ* образует дерево, сходящееся в вершину $t$. Чтобы быстро сравнивать строки, будем хранить для каждой вершины $v$ хэш строки от $v$ до $t$, а также таблицу двоичных подъемов ($p[v,k]$ = вершина, в которой мы окажемся, если пройдем от $v$ по ссылкам вперед $2^k$ шагов). Если предположить $p[t,0] = t$ (корень ссылается на себя самого), то крайних случаев не будет, и для пересчета двоичных подъемов достаточно одной формулы: $p[v,k] = p[p[v,k-1],k-1]$. Теперь мы умеем сравнивать строки на больше, меньше, равно за $O(\log N)$ и решение работает за обещанное время $O(E \log V)$.

# 3 Комментарии к лекции Саши Миланина

1. Простой рандомизированный алгоритм дает уже почти максимальное паросочетание. Его проблема в том, что он может ошибиться на $O(1)$ и последние, самые сложные, дополняющие пути просто не найти.

2. Чтобы восполнить эту проблему, запустим в конце любую, самую простую, реализацию алгоритма Эдмондса (сжатие соцветий). Даже если один дополняющий путь мы будем искать за время $O(V^3)$, поскольку дополняющих осталось мало, общее время работы будет также $O(V^3)$.

# 4 Комментарии к лекции Миши Дворкина: Cover

1. Алгоритм с оценкой два: построим насыщенное паросочетание $M$ (такое, что никакое ребро нельзя добавить в него просто так, ничего не перестраивая). Возьмем все $2|M|$ концов. Это покрытие. Также понятно, что размер минимального покрытия $\geqslant |M|$.
2. `min Cover` = `max Independent Set` (т.к. дополнением к любому покрытию является независимое множество вершин и наоборот). Мы получили возможность, получив хорошее решение одной из двух задач, сразу применять его ко второй.
3. Очень хорошо работает следующая простая жадность: брать каждый раз вершину, покрывающую максимальное число еще не покрытых ребер (т.е. имеющую максимальную степень в остаточном графе).
4. Улучшим нашу жадность. Если есть вершина степени один, то нужно обязательно взять или ее, или ее единственного соседа. Очевидно, что выгодно брать соседа. Если мы в какой-то момент можем применить это отсечение, применяем.
5. Чтобы наша жадность работала совсем хорошо, нужно превратить ее в перебор. Т.е. брать не вершину max степени, а перебирать все вершины в порядке убывания степени. И так пока время работы не превысит одну секунду. Сам по себе перебор уже лучше жадности, но есть последнее улучшение: отсечение по ответу. Пусть сейчас мы уже умеем строить покрытие размера $Best$. На текущем уровне рекурсии мы уже взяли $X$ вершин, но еще не покрыли все ребра. Очевидно, что если есть $K$ независимых (не пересекающихся по концам) ребер, то нужно взять хотя бы еще $K$ вершин. Отсечение: `if` $Best \leqslant X + K$ `then return`

# 5 Комментарии к лекции Миши Дворкина: Salesman Problem

1. Чтобы найти путь, нужно найти цикл и выкинуть одно самое дорогое ребро.
2. На практике решение этой задачи нужно для больших $N$. Т.е. если алгоритм имеет плохую асимптотику и не работает даже для $N = 2000$, он нам слабо интересен.
3. Алгоритм построения: будем из цикла длины $k$ получать цикл длины $k+1$. Изначально у нас есть цикл длины 1. Для того, чтобы увеличить цикл, выберем вершину и вставим ее между двумя соседними вершинами цикла. Выберем вершину и место для вставки так, чтобы длина цикла увеличилась как можно меньше.
4. Оценка времени. Если реализовывать описанную выше идею в лоб, то получится время $O(N^3)$. Наша задача быстро выбирать пары (вершина, место). Давайте также как в алгоритмах Дейкстры и Прима для каждой вершины помнить оптимальное место вставки и пересчитывать, если цикл меняется. Таким образом можно получить время

$O(N^2)$ (если вы увидели только $O(N^2 \log N)$ — это нормально, но подумайте еще, $O(N^2)$ тоже существует).ж

5. Локальные оптимизации: на самом деле любой хороший алгоритм решения задачи коммивояжера состоит из двух фаз — поиск хорошего начального приближения (это мы уже умеем делать) и собственно улучшение текущего ответа. Т.е. локальные оптимизации. Существует два вида простых оптимизаций. Первый: для любой пятерки подряд идущих вершин мы можем перебрать все 120 вариантов и выбрать оптимальный, если что-то улучшилось — хорошо. Второй: пусть мы хотим решить задачу для точек на плоскости. Тогда ребра это отрезки. Если какие-то два отрезка найденного цикла пересекаются, можно их развернуть так, что пересечение пропадет, а длина уменьшится.

# 6 Комментарии к лекции Миши Дворкина: Рюкзак

1. Будем решать чуть другую задачу: даны $K$ рюкзаков, нам нужно уложить в них **все** вещи. Задача, обсуждавшаяся на Мишиной лекции сводится к данной бинпоиском по $K$. Мы также можем предположить, что рюкзаки имеют разные размеры. Размер $i$-го рюкзака $W_i$.

2. Общий алгоритм решения: отсортировали вещи в каком-то порядке, отсортировали рюкзаки в каком-то порядке, жадно перебираем рюкзаки и в каждый рюкзак каждую вещь или кладем (если влезает), или не кладем (вещи мы, конечно, перебираем всегда в одном и том же порядке).

3. Чтобы получить классное решение, сделаем так: будем запускаться четыре раза (можно и вещи, и рюкзаки сортировать как по убыванию размера, так и по возрастанию). Каждый из четырех запусков — это перебор, основанный на описанной выше жадности. Каждому из четырех переборов дадим по 0.25 секунд.