1. Нам требуется проверить достижимость вершины t из вершины s. Для этого запустим поиск в глубину из вершины t. Таким образом, мы и для ориентированного, и для неориентированного графа пройдем все вершины графа, и в случае успеха поиск в глубину найдет путь простой путь из t в s, иначе вернется в исходную вершину и сообщит о недостижимости. Сложность поиска в глубину равна $O(|V| + |E|)$.

2. Представим лабиринт в виде неориентированного графа, где каждая комната это вершина графа, а коридоры между ними - ребра графа. Для нахождения выхода из лабиринта нам нужно обойти в худшем случае все вершины графа. Для этого нужно с помощью поиска в глубину пройти все вершины графа и проверять каждую вершину на выход. Для того, чтобы запоминать, посещали ли мы данную комнату ранее, мы кладем в нее монетку, таким способом имитируя массив посещений вершин. То есть алгоритм такой: стартуем в какой-то комнате, кладем монетку перед входом в любой коридор, идем по нему в любую из смежных комнат, посещаем все смежные с ней комнаты (конечно, кроме исходной) (кладя монетку перед всеми коридорами, куда мы пойдем), возвращаемся в исходную, далее по возвращении в исходную посещаем все остальные смежные комнаты. Таким образом мы пройдемся по всем комнатам (то есть точно узнаем есть ли выход или нет), пройдя по всем m коридорам 2 раза (туда и обратно), то есть за $O(m)$ переходов.

3. Данная задача состоит из двух частей: 1) доказательство существования простого пути длины $n-1$ в турнире размера n, 2) построение алгоритма нахождения простого пути.

1. Проведем доказательство по ММИ. Заметим, что верно для $n = 2, n = 3$. Пусть верно для $n-1$. Проверим для n. По предположению индукции сущесвтует простой путь длины $n-2$ в турнире размера $n-1$. Обозначим этот путь как $\{V_1; ...; V_{n-1}\}$. Теперь докажем, что всегда найдется участок пути, куда можно будет вставить новую вершину V_n. Обозначим за A - массив смежности графа. Рассмотрим 3 случая:

   а) Если $A[n][V_1] == 1$, тогда просто добавляем вершину перед первой вершиной пути.

   б) Если $A[n][V_{n-1}] == 0$, тогда просто добавляем вершину после последней вершины пути.

   в) Если $A[n][V_1] == 0$ и $A[n][V_{n-1}] == 1$, то докажем что всегда найдется такая пара вершин нашего пути, таких что новую вершину можно вставить между ними. Воспользуемся техникой, похожей на метод противника: будем идти по нашему пути слева направо и не давать вставить новую вершину между существующими вершинами пути. $A[n][V_1] == 0 \Rightarrow A[V_1][n] == 1$, то есть существует прямая дорога без промежуточных пунктов между первой вершиной пути и новой вершиной. Нам надо сделать так, чтобы на было пути изновой вершины в вершину V_2, чтобы мы не могли вставить новую вершину между первой и второй. Тогда, мы просто берем и делаем так, чтобы стрелка шла не из новой вершины в V_2, а наоборот. И так далее. Тогда, когда мы дойдем до последней пары вершин $\{V_{n-2}; V_{n-1}\}$ мы сможем

вставить новую вершину между ними (так как $A[V_{n-2}][n] == 1, A[n][V_{n-1}] == 1$), иначе мы бы просто смогли бы вставить ее ранее.

2. Алгоритм поиска такого пути следующий:

4. Данная задача эквивалентна нахождению компонент сильной связности в ориентированном графе (так как в ксс - это связный подграф максимального размера, а тогда количество областей будет минимальным). Тогда алгоритм следующий: проходимся в глубину, транспонируем граф, проходимся в глубину еще раз в порядке убывания времени закрытия вершин. Сложность данного алгоритма равна $O(m+n)$. Корректность доказывалась на лекции 8.

Для второго пункта задачи мы считаем количество ксс, а также формируем для каждой ксс множество других ксс, в которые есть путь из данной. Тогда после формирования всех этих множеств ответ будет формироваться так: (количество всех ксс) * 2 - (суммарная мощность всех множеств). Умножение на 2 взялось из соображений пути туда-обратно из пары ксс.

5. В данной задаче я считаю, что вершины графа пронумерованы в порядке движения пути, то есть граф-путь это $\{V_1; V_2; ...; V_n\}$.

Будем рассматривать только граф на m ребрах. Заметим, что если ребро сонаправлено с направлением пути, то оно не влияет на количество ксс. Тогда уберем все пары значений $(i; i+k), k \in \mathcal{N}$.

Далее сортируем пары чисел по убыванию первой компоненты (первая компонента - начало ребра, вторая - конец).

Затем проходимся по полученным ребрам следующим образом: если начало текущего ребра меньше, чем конец предыдущего, то увеличиваем количество ксс на 1, иначе удаляем текущее ребро, а первую компоненту предыдущего ребра заменяем на минимум из двух чисел: 1) конец предыдущего ребра, 2) конец текущего ребра.

В конце считаем число одиночных ксс среди оставшихся вершин и добавляем их в счетчик.

Сложность данного алгоритма: $O(m)$ (проход в конце) + $O(m*\log_2(m))$ (сортировка)

6. Первым делом докажем, что в обеих долях нашего двудольного графа одинаковое число вершин.

По условию известно, что степень каждой вершины равна 2, тогда у каждой вершины левой доли есть 2 ребра. Общая сумма ребер, левый конец которых лежит в левой доле, тогда равен $2 * n$ (где n - число вершин в левой доле). Тогда, из принципа Дирихле следует, что количество вершин в правой доле также $2*n$ (данное утверждение следует из принципа Дирихле след. образом: если бы вершин в правой доле было бы меньше, чем n, то была бы хотя бы 1 вершина, чья степень была бы больше, чем 2, и наоборот, если бы в правой доле было бы больше вершин, то степень хотя бы одной вершины была бы меньше 2, что противоречит условию задачи).

Итак, мы доказали, что количество вершин в обеих долях графа одинаково. В таком случае, рассмотрим левую долю. Из каждой вершины исходит по 2 ребра. Тогда, подмножество ребер графа, имеющих попарно различные концы, будет состоять ровно из n ребер - количество равное половине всех вершин графа, то есть $\frac{|V|}{2}$.

Далее просто выбираем произвольную веришну из левой доли и идем по любому из рёбер в правую долю. Затем по оставшемуся ребру правой вершины идем в левую долю, и так далее. Таким способом, мы обойдем весь граф в глубину, и он либо будет являться одним большим деревом, либо будет лесом, состоящим из нескольких деревьев (циклов в графе). Мы сможем пройти так весь граф, задействовав все вершины, так как у всех вершин графа степень 2, то есть в случае одного большого дерева все вершины можно просто разместить по кругу и соединить между собой и, начав с любой из них, мы полностью обойдем граф. В случае нескольких деревьев граф можно будет представить в виде нескольких таких "кругов"благодаря тому, что степени всех вершин равны 2.

Итого, сложность такого обхода в глубину равна $O(|V| + |E|)$.

7. Алгоритм следующий: сначала выполняем топологическую сортировку, а затем делаем обход в глубину, подсчитывая максимальный путь.

Сложность: сортировка - $O(|V|+|E|)$, обход в глубину - $O(|V|+|E|)$. Итого, сложность равна $O(|V| + |E|)$.

Корректность: самый длинный путь в DAG должен начинаться в топологически самой старшей вершине, так как в противном случае, мы всегда сможем добавить как минимум одну вершину в путь (вершину, из которой есть путь в данную)