1 Остовные деревья

1.1 Определение

Остовным деревом (spanning tree) связного неориентированного графа G на n вершинах будем называть подмножество его ребер размера $n-1$, не содержащее циклов. Понятно, что тогда по этим ребрам можно добраться из любой вершины до любой, и эти ребра образуют дерево. Если граф G взвешенный, суммарный вес ребер остовного дерева будем называть *весом* остовного дерева. Классическая задача — построить остовное дерево минимального веса *(minimal spanning tree, или MST).*

1.2 Важное утверждение

Утверждение. Пусть F — некоторое множество ребер, которое можно достроить (т.е. добавить сколько-то ребер) до MST. Пусть S — некоторая компонента связности по ребрам F и e — любое из минимальных ребер, ведущих из множества S наружу (т.е. в вершину, не лежащую в S). Тогда множество $F' = F + e$ тоже можно достроить до некоторого MST.

Доказательство. Рассмотрим T — MST, содержащее в себе F. Если оно содержит также и e, то все хорошо. Иначе добавим к T ребро e, от этого в нем появился ровно один простой цикл. Пойдем вдоль этого цикла, начиная с ребра e. На первом шаге мы выйдем из множества S наружу, но должны когда-то туда вернуться. Пусть f — первое ребро, по которому мы возвращаемся в S. $w(f) \geqslant w(e)$, поскольку e — минимальное из ребер, ведущих из S наружу. Удалим из графа f, получили множество T'; оно является деревом, поскольку мы удалили ребро на единственном цикле. Кроме этого, $w(T') = w(T) - w(f) + w(e) \leqslant w(T)$, и T — MST, значит, T' тоже MST и содержит в себе множество $F + e$. Доказали.

□

Применяя это утверждение на разные лады, будем получать разные алгоритмы для построения MST.

2 Алгоритм Прима

Общая схема алгоритма Прима такова: изначально в F нет ребер. Выберем произвольную вершину v, ее компонента связности S сначала состоит из нее самой. На каждом шаге мы имеем право добавить любое минимальное ребро, идущее из S наружу, при этом другой конец ребра добавится в S; будем повторять этот процесс, пока S не станет равно всему множеству вершин. В конечном итоге F будет состоять из ребер MST.

Чтобы каждый раз не искать минимальное ребро заново, сделаем следующее: для каждой вершины u, не лежащей в S, будем поддерживать $dist[u]$ — вес минимального ребра, ведущего из S в u, либо ∞, если такого ребра нет; кроме этого, будем хранить $parent[u]$ — вершина в S, которая является концом минимального

ребра, ведущего в u (если такое есть). Тогда на каждом шаге надо просто выбрать вершину u, не лежащую в S, с минимальным $dist[u]$, добавить u в S и ребро $(u, parent[u])$ в множество F. После добавления вершины u в множество S, у нас могли поменяться $dist[v]$ для остальных вершин, т.к. ребра, исходящие из u, там еще не учтены. Пройдем по всем таким ребрам и поменяем $dist[v]$ для тех вершин, где новые ребра являются минимальными.

На каждом шаге множество S расширяется на одну вершину, значит, после $n-1$ шага алгоритм завершится. На каждом шаге мы должны выбрать вершину с минимальным $dist[u]$. Можно делать это, например, простым проходом по массиву $dist$, что потребует $O(n)$ операций на каждом шаге, и $O(n^2)$ операций суммарно. Кроме этого, требуется рассмотреть все ребра для пересчитывания $dist[v]$, но легко заметить, что каждое ребро будет рассмотрено не более двух раз (в каждую из сторон). Таким образом, суммарное время работы такой реализации алгоритма Прима составляет $O(n^2 + m) = O(n^2)$ (как обычно, n и m — количество вершин и ребер в графе соответственно)

Узкое место здесь — это поиск следующей вершины для добавления в S. Воспользуемся для этой цели какой-либо структурой данных, которая поддерживает операции добавления элемента и извлечения минимума за время $O(\log n)$, где n — количество элементов в структуре; в качестве такой структуры можно взять, например, кучу или `std::set`. Теперь операция выбора новой вершины требует $O(\log n)$ операций в худшем случае, но каждое изменение $dist[v]$ требует изменения элемента в куче, поэтому суммарное время работы стало равно $O(n \log n + m \log n) = O(m \log n)$. Мы видим, что в случае неплотных графов (т.е. m гораздо меньше n^2) реализация алгоритма Примы с кучей позволяет существенно выиграть по времени, но в случае плотных графов (m порядка n^2) реализация с кучей проигрывает простой реализации ($O(n^2 \log n)$ против $O(n^2)$).

3 Алгоритм Краскала

Теперь сделаем по-другому. Будем рассматривать ребра графа G в порядке неубывания веса и добавлять в F те из них, после добавления которых в графе не образуется циклов. Мы можем так делать, поскольку если ребро e соединяет на данном шаге различные компоненты связности, одна из которых S, то оно является минимальным ребром, идущим наружу из S; если бы это было не так, мы бы рассмотрели минимальное ребро раньше и добавили его в F, однако этого не произошло. Опять-таки, в конце получим некоторое MST графа G.

Сложность заключает в том, как проверять лежат ли сейчас концы выбранного ребра в одной компоненте связности. Можно сделать так: для каждой вершины хранить номер компоненты связности, в которой она находится. Если для текущего ребра номера компонент для его концов не совпадают, пройдемся по всем вершинам и перекрасим вершины таким образом, чтобы две компоненты слились в одну. Такая процедура требует $O(m \log n + n^2)$ времени на сортировку ребер и процедуры перекраски вершин (их будет не более $n - 1$).

Здесь узким местом является перекраска вершин и проверка, лежат ли они в одной компоненте. Нам бы очень пригодилась структура, поддерживающая набор

непересекающихся множеств на заданном наборе элементов и умеющая быстро выполнять следующие операции:

- проверить для двух элементов, лежат ли они в одном множестве
- объединить два множества

4 СНМ

Реализуем такую структуру следующим образом: будем хранить ориентированный граф, в котором из каждой вершины исходит не более одного ребра. Граф всегда будет состоять из подвешенных деревьев, в каждом из которых ребра ориентированы по направлению к корню дерева. Будем считать, что два элемента лежат в одном множестве, если они находятся в одном таком дереве. Для двух данных элементов это можно проверить, если для каждой вершины пройти до корня соответствующего дерева и сравнить корни. Чтобы объединить два множества, достаточно подвесить корень одного из деревьев к корню другого.

Временные затраты на операцию проверки зависят от «высоты» получающихся деревьев. Легко построить последовательность объединений, после которой деревья будут достигать высоты порядка n, и такой же порядок будет иметь количество операций на каждую проверку.

Чтобы уменьшить высоту деревьев, будем использовать следующие оптимизации.

4.1 Ранговая эвристика

Если мы хотим объединить два дерева, то интуитивно нам более выгодно подвешивать менее глубокое дерево в более глубокому. Реализуем это следующим образом: каждой вершине припишем «ранг» — число, означающее максимальную глубину ее поддерева. При объединении двух множеств, выберем из двух корней тот, который имеет меньший ранг, и подвесим его к другому. Если при этом ранги равны, то ранг корня нового дерева увеличивается на 1, в противном случае ничего не происходит.

Теорема. *В СНМ с ранговой эвристикой после выполнения n операций все деревья имеют высоту $O(\log n)$.*

Доказательство. Для того, чтобы получить дерево ранга как минимум k, необходимо объединить два дерева ранга как минимум $k-1$. Если обозначить минимальное количество операций для получения дерева ранга k за f_k, то выполняется соотношение $f_k = 2f_{k-1} + 1$, откуда $f_k = 2^k - 1$. За n операций можно успеть получить дерево ранга k, если $2^k - 1 \leqslant n$, т.е. $k \leqslant \log_2(n+1) = O(\log n)$.

□

4.2 Эвристика переподвешивания к корню

Мы можем заметить, что если переподвесить какую-либо вершину куда-либо в другое место в том же дереве, структура множеств останется той же. При этом нам стоит стремится подвешивать как можно больше вершин к корню соответствующего дерева, поскольку тогда пути до корня будут короче и поиск корня будет работать быстрее.

При поиске корня для заданной вершины мы должны пройти путь от этой вершины до корня дерева. После нахождения корня мы можем переподвесить все вершины на пути сразу к корню.

Теорема. *В СНМ с эвристикой переподвешивания выполнение n операций требует времени $O(m \log n)$.*

Доказательство. Разделим ребра на три типа: ребра, ведущие в корень, а также «легкие» и «тяжелые» ребра. Тяжелое ребро — это ребро, на котором висит как более половины вершин в поддереве, а легкие — все остальные. Заметим, что в вершину не может входить более одного тяжелого ребра. Посчитаем количество переходов по ребрам каждого типа отдельно. Пусть мы сделали суммарно m запросов. Запрос на объединение состоит из двух запросов нахождения корня и одного переподвешивания. Тогда количество переходов по ребрам корень составляет $O(m)$, а суммарное количество переходов по легким ребрам составляет $O(m \log n)$.

С тяжелыми ребрами сложнее. Если мы переходим вверх по тяжелому ребру, то его нижний конец будет переподвешен в корень, поэтому размер поддерева уменьшится как минимум в два раза, поэтому в каждой вершине не может произойти более $O(\log n)$ удалений тяжелых ребер. Операция подвешивания создает не более одной тяжелой вершины, значит, всего тяжелых вершин за все время выполнения m операций не может быть создано более m вершин. Таким образом, всего может быть не более $O(m \log n)$ переходов по тяжелым ребрам. Итак, суммарное количество переходов во ребрам (а значит, и количество операций) не может превышать $O(m \log n)$.

□

Без доказательства отметим, что если применить обе эвристики одновременно, высота каждого дерева не будет превышать $O(\alpha(n))$, где $\alpha(n)$ — обратная функция Аккермана. Эта функция не превосходит 4 для всех значений $n \leqslant 2^{2^{2^{2^{16}}}}$, т.е. во всех практических случаях ее можно считать константой.