Динамическое транзитивное замыкание

Божнюк Александр
371 группа

1. На лекции мы научились поддерживать инкрементальное транзитивное замыкание ориентированных графов. Придумайте алгоритм для декрементального транзитивного замыкания, работающий за $O(n^2(m+n))$ суммарно на все апдейты.

Решение.
Опишем основные идеи алгоритма.

- Как и в инкрементальном алгоритме, мы поддерживаем матрицу достижимости M. Она изначально строится при помощи транзитивного замыкания. Однако, помимо этого, мы строим списки смежности для данного графа и его реверснутой версии (в ней ребра направлены в противоположную сторону). Обозначим их $gList$ и $gList_{rev}$ соответственно.
- При удалении ребра (i,j) могут появиться вершины v, которые стали недостижимы из i, но все еще достижимы из j. Информацию о них и требуется обновить в M. Эти вершины мы можем получить при помощи запуска DFS на данном графе из вершины i (взяв те вершины, до которых не дошли при обходе) и проверки достижимости из j через матрицу M.
- При этом заметим, что вершина x достижима из y в данном графе тогда и только тогда, когда вершина y достижима из x в реверснутой версии. Пользуясь этим, для обработки вершин v, описанных выше, мы можем запустить DFS из них в реверснутом графе. Это даст информацию о достижимости из вершины v всех остальных вершин в реверснутом графе, а значит и о достижимости из остальных вершин вершины v в исходном графе. Эту информацию мы в итоге и вносим в M.

Algorithm 1

1: **function** REMOVE(i, j)
2: $\quad gList[i].remove(j)$
3: $\quad gList_{rev}[j].remove(i)$
4: $\quad reachedFromI := dfs(gList, i)$
5: $\quad$ **for** $v : reachedFromI[v] = 0 \ \wedge\ M[j, v] = 1$ **do**
6: $\qquad reachedFromVRev := dfs(gList_{rev}, v)$
7: $\qquad$ **for** $u \in V$ **do**
8: $\qquad\quad M[u, v] := reachedFromVRev[u]$

Докажем, что данный алгоритм работает за $O(n^2(m+n))$ суммарно за все апдейты.
Доказательство

1. Функция REMOVE вызовется максимум m раз на все апдейты, когда мы удалим все ребра.
2. Обход DFS работает за $O(n+m)$.
3. В строке 5 алгоритм в худшем случае перебирает порядка n вершин. Строки 6-8 работают за $O(n+m)$ из-за DFS. Получаем, что на одно удаление ребра строки 5-8 работают за $O(n(n+m))$.
4. Учитывая пункт 1, суммарно на все апдейты строки 5-8 отработают за $O(mn(n+m))$.

5. Если учесть стандартную для графов оценку $m <= n^2$, то получим, на первый взгляд, итоговую сложность $O(n^3(m+n))$ на все апдейты.

6. Сложность выше неверная. Вспомним о декрементальности алгоритма. Оно означает, что если $M[i,j]$ однажды стало 0, то оно уже никогда не станет 1, так как ребра мы только удаляем. В строке 5 мы делаем проверку $M[j,v] = 1$, однако в строке 8 мы делаем зануление значения в матрице M.

   Более того, в результате строк 6-8 хотя бы одна ячейка $M[u,v]$ точно обнулится. Так, если удаляется ребро (i,j), и вершина v подошла под условия в строке 5 и алгоритм исполняет строки 6-8, то гарантированно произойдет обнуление ячейки $M[i,v]$, ведь ранее сработало первое условие в строке 5.

   Значит, если вершина v прошла через проверку в строке 5, то при дальнейших вызовах функции пара i и v рассматриваться уже не будет, ведь их $M[i,v] = 0$, не пройдет условие в строке 5.

   Получается что условию в строке 5 будет удовлетворять не более n^2 вершин на все апдейты. А значит что и строки 6-8 будут исполняться не более n^2 раз на все апдейты.

7. Учитывая предыдущий пункт, суммарная сложность алгоритма на все апдейты будет равна $O(n^2(m+n))$.