SPb HSE, ПАДИИ, 1 курс, весна 2023/24
Практика по алгоритмам #9

DSU и Флойд

14 марта

Собрано 12 марта 2024 г. в 15:48

Содержание

DSU и Флойд

1. **DSU. Доказываем сжатие путей.**

   Почему одна эвристика «сжатие путей» будет работать за $\mathcal{O}((m+n)\log n)$?

2. **Online двудольность**

   Дан неорграф. В него в online добавляются рёбра.
   После каждого добавления говорить, двудолен ли граф?

   a) $\mathcal{O}(m \log n)$ через перекрашивание компонент.
   b) Быстрее через DSU.

3. **Чётность**

   В каждой клетке прямой записано число 0 или 1. Поступает информация: чётность количества единиц на отрезке $[L_i, R_i]$, найти первый запрос, после которого данные противоречивы.

4. **Казалось бы, причём здесь СНМ?**

   У нас есть массив длины n, мы хотим выполнить m запросов вида «покрасить отрезок $[l_i, r_i]$ массива в цвет c_i» и вывести, что получилось в конце. Решите за $\mathcal{O}(n \log m)$.

   Подсказка: попытайтесь выполнять запросы, начиная с последнего.

5. **Порог достижимости**

   Для каждой пары вершин в графе найти $w[a, b]$ – такой минимальный вес, что из a в b есть путь по рёбрам веса $\leqslant w[a, b]$.

6. **Флойд и битовое сжатие**

   Найдите транзитивное замыкание графа за $\mathcal{O}(\frac{V^3}{w})$.

7. **Найти отрицательный цикл**

   а) После Форда-Беллмана

   б) После Флойда

8. **(*) Форд-Беллман и число итераций**

   Пусть алгоритм Беллмана-Форда на каждой итерации рассматривает ребра в таком порядке:

   1. Рёбра из меньшей вершины в большую в порядке *возрастания* номера исходящей вершины.
   2. Рёбра из большей вершины в меньшую в порядке *убывания* номера исходящей вершины.

   Докажите, что алгоритм найдет все кратчайшие пути за $\frac{n}{2}$ итераций.

9. **(*) Форд-Беллман и число итераций – 2**

   Добавим рандомчика и получим $\frac{n}{3}$ итераций.

Разбор задач практики

1. **DSU. Доказываем сжатие путей.**

   Конспект 1к-ПМИ.

2. **Online двудольность**

   b) $\mathcal{O}((m+n)\alpha(n))$.

   Способ 1 (основной). У каждой компоненты связности есть одна или две доли. Храним доли в DSU. Если у компоненты две доли, их корни a_i, b_i, то будем хранить «ссылку на вторую долю» `link[a`$_i$`] = b`$_i$`, link[b`$_i$`] = a`$_i$. При добавлении ребра (x, y), три случая:

   1. `root(a) == root(b)` $\Rightarrow$ нечётный цикл.
   2. `link[root(a)] == root(b)` $\Rightarrow$ ребро между долями, ничего не делаем.
   3. Ребро между компонентами.
      У нас есть 4 доли – корни и их `link`-и, вызовем два `join`-а, обновим `link`-и.

   Способ 2. Храним цвета вершин и компоненты связности в «DSU на деревьях», перекрашиваем лениво. На рёбрах DSU указаны цвета 0 и 1.
   Цвет вершины v – `XOR` значений на пути от v до корня DSU.
   При `join` ставим на новое ребро 1, если нужно перекрашивание.
   При сжатии путей на новое ребро ставим цвет, равный цвету вершины.

3. **Чётность**

   Число единиц на отрезке $[L, R]$ чётно, то число единиц на префиксах $[1, L-1]$ и $[1, R]$ имеет одинаковую чётность, иначе разную
   $\Rightarrow$ видим граф, где вершины – префиксы, рёбра – отрезки
   $\Rightarrow$ задача аналогична предыдущей, только здесь нам сообщают, либо что концы ребра должны иметь одинаковый цвет (лежать в одной доли), либо разный (лежать в разных долях). Каждая компонента связности двудольна, можем, как и в решении выше хранить её, как «пара двух долей».

   Решение без двудольности. Будем проводить рёбра с весами 01. СНМ. Каждая компонента — множество в СНМ. Когда приходит ребро $a \xrightarrow{w} b$, если a и b уже в одной компоненте, проверяем, что $w_e =$ `xor` на пути $a \rightsquigarrow b$, а это ровно $up(a) \oplus up(b)$, где $up(a)$ – `xor` на пути от a до корня. $up(a)$ считается СНМ-ом также, как $get(a)$. Когда делаем `join`, проводим ребро не между a и b, а между $get(a)$ и $get(b)$ веса $up(a) \oplus up(b) \oplus w$.

   P.S. При больших координатах в offline можно сжать координаты, в online использовать хеш-таблицу (если `p[x]` еще не был определен, то `p[x] = x`).

4. **Казалось бы, причём здесь СНМ?**

   Обрабатывая запросы с конца, идём слева направо и красим ещё непокрашенные клетки.
   Как быстро пропускать непокрашенные?
   Сжатие путей: изначально `p[i]=i+1`, после покраски слева направо путь сжимается.
   Сжатие путей, как и СНМ с одной эвристикой даст суммарное время $\mathcal{O}((m+n)\log n)$.

Полноценный СНМ. В СНМ храним области «что-то покрашенное и, возможно, крайняя правая не покрашенная». Для корня множества помним крайнюю правую клетку. Тогда

```
for (int l = right[get(l)]; l <= r; l = right[get(l)]):
    if (color[l] == -1) color[l] = c;
    join(l, l+1);
```

5. **Порог достижимости**

Флойд: `relax(d[i,j], max(d[i,k], d[k,j]))`. $\mathcal{O}(V^3)$.
«Джонсон». Заметим, что если вес пути равен max ребру, то Дейкстра работает даже с отрицательными ребрами. Т.е. никаких потенциалов не надо, просто Дейкстра из каждой вершины. $\mathcal{O}(VE \log V)$ (в теории $\mathcal{O}(VE + V^2 \log V)$).

Ещё решение (снм): добавляем ребра в пустой граф в порядке возрастания.
Если ребро e соединяет разные компоненты C_1, C_2, то его вес – ответ
для всех пар $a \in C_1, b \in C_2$. Выставим им ответ перебором пар за $|C_1| \cdot |C_2|$.
Поскольку каждой паре ответ запишется только один раз, это даст в сумме $\mathcal{O}(V^2)$.
Итого $\mathcal{O}(sort(E) + V^2 + E \cdot T(\text{dsu}))$. DSU изучим в будущем.

6. **Флойд и битовое сжатие**

Как выглядит обычный флойд для транзитивного замыкания?

```
for k = 0..n-1:
    for i = 0..n-1:
        for j = 0..n-1:
            a[i][j] |= (a[i][k] and a[k][j])
```

Перепишем код, так чтобы использовать битовое сжатие (`a[i]` это теперь битсет).

```
for k = 0..n-1:
    for i = 0..n-1:
        if (a[i][k])
            a[i] |= a[k] // a[i][j] |= (a[i][k] and a[k][j])
```

7. **Найти отрицательный цикл**

а) После Форда-Беллмана: $\forall v$ обновлённой на n-й фазе, откатываемся по ссылкам $p[v]$, откаты дадут предпериод и период, период – искомый цикл.

б) После Флойда: в первый момент времени, когда $d[v,v] < 0$, начинаем любое стандартное восстановление ответа (или $k[i,j]$, или $next_i[i,j]$).

8. **(*) Форд-Беллман и число итераций**

Худший случай – дерево кратчайших путей есть путь длины n. За одну итерацию мы будем прыгать в следующий локальный минимум. Локальных минимумов больше всего на пилообразной последовательности, $\frac{n}{2}$.

9. **(*) Форд-Беллман и число итераций – 2**

Алгоритм: добавим `randomshuffle` номеров вершин.

Вероятность для каждой вершины, что она – локальный минимум ровно $\frac{1}{3}$ (она должна быть меньше обоих соседей). Итого матожидание числа локальных минимумов $\frac{n}{3}$.

Домашнее задание

1. **(2) Матрица расстояний с уменьшением ребер**

   На запрос «уменьшился вес ребра» за $\mathcal{O}(n^2)$ пересчитывать матрицу расстояний.

2. **(2) Насколько сжатие путей быстрое?**

   У нас есть СНМ на деревьях со сжатием путей.

   Пусть с какого-то момента выполняются только запросы `get`. Доказать, что все эти запросы `get` отработает за $\mathcal{O}(n+m)$, где n – число элементов, m – суммарное число запросов.

3.1. Дополнительная часть

1. **(2) Random и СНМ**

   Рассмотрим реализацию СНМ:

```
1 int get(int v) { return v == p[v] ? v : get(p[v]); } // без сжатия путей!
2 void join(int a, int b):
3     a = get(a), b = get(b);
4     if (rand() & 1) swap(a, b);
5     p[a] = b;
```

   Временем работы на тесте назовём матожидание по случайным битам. Какое максимальное время работы этой реализации по всем возможным тестам?
   Докажите оценку сверху, приведите тест, на котором она достигается.

2. **(3) Dynamic connectivity in directed graph**

   Дан ацикличный орграф. Нужно за $\mathcal{O}(n^2)$ обрабатывать запросы «добавить ребор» и «удалить ребро». Гарантируется, что граф всегда остается ацикличным. Также нужно за $\mathcal{O}(1)$ отвечать на запрос «есть ли путь из a в b»?

   Подсказка: наш алгоритм будет вероятностным.