Параллельные алгоритмы

Надуткин Федор

January 2023

Pipelining

Идея `pipelining` - идея конвейера, делать всё поэтапно и для новой детали начинать выполнение первого этапа, пока старая деталь уже на втором.

Задача: Вставить в 2-3 дерево m элементов.

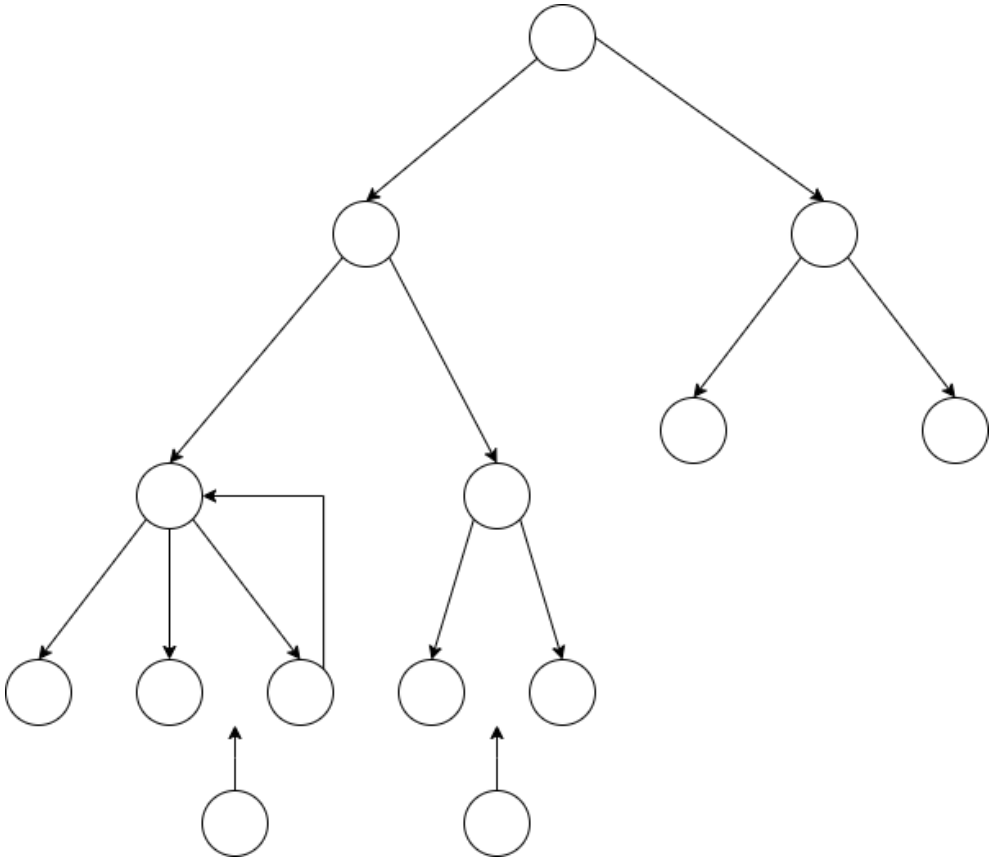


Рис. 1. Вставка в 2–3 дерево

1) Находим куда надо вставить элемент.
2) Вставляем, если у вершины 2 ребёнка, то останавливаемся, если 3, то делаем сплит и поднимаемся выше.
3) Если 2 или 3 потока пришли делать split, то выбираем главного и он уже идёт делить дальше.

$\texttt{Work} = m \cdot \log(n+m)$
$\texttt{Span} = \log m + \log n$

Проблема:

Если у нас на первом этапе вставляется множество элементов в одно место, то делать split не получится.

Решение:

1) Взять центральный элемент из массива тех, что нам надо вставить.

2) Вставить этот элемент в `2-3 дерево`.

3) Делать процедуру уже для 2 массивов, между которыми уже будет ключ.

`Span` $= \log 1 \cdot \log n + \log 2 \cdot \log n + \cdots + \log m \cdot \log n = \mathcal{O}(\log^2 m \cdot \log n)$

Ускорение:

Стоит заметить, что нам не нужно доводить элементы до самого верха, чтобы вставить новые. По прошествии 2 итераций новые элементы никак не будут взаимодействовать со старыми, а значит не помешают. Поэтому алгоритм изменится следующим образом:

1) Взять центральный элемент.

2) Вставить элемент в `2-3 дерево`.

3) Проделать 2 итерации для всех незавершённых элементов.

4) Сделать процедуру для 2 массивов.

`Span` $= \log m \cdot (\log n + \log m) = \log m \cdot \log n + \log^2 m$

Symmetry breaking

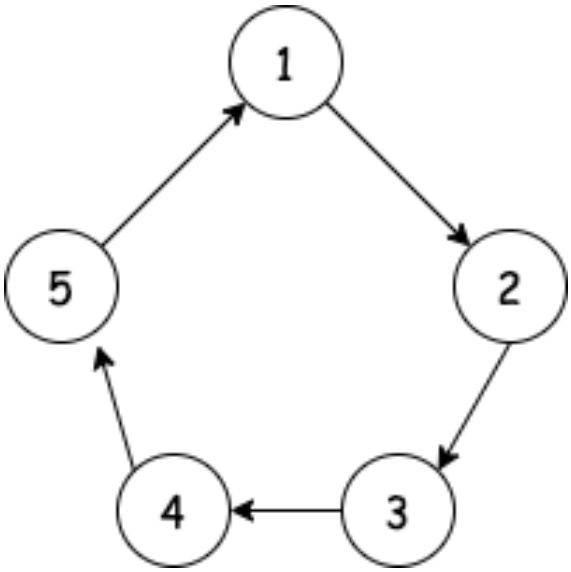


Рис. 2. Symmetry breaking

Для каждой вершины дано `next[id]`, нужно найти `prev[id]`. Делаем `parallel for` и устанавливаем `prev[next[id]] = id`.

Задача

Раскрасить в минимальное число цветов граф.

Решение

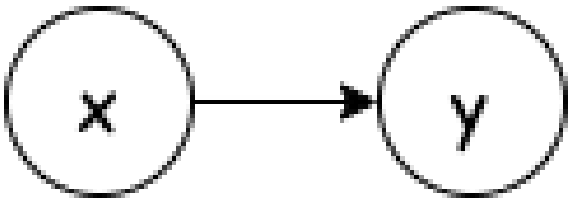


Рис. 3. Берём x и next[x]

Переведём x и y в двоичную форму. $x =$ `001101`, $y =$ `000101`. `color[y]` $= 2 \cdot \min(bit_{color[x] \neq color[y]}) + bit_{color[y]} = 2 \cdot 3 + 0$. Изначально `color` каждой вершины равен её id.

Стоит заметить, что $\forall$ двух соседних вершин, их цвета не равны. Если `color[id]` с `color[id + 1]` и `color[id + 1]` с `color[id + 2]` различаются в разных битах (например `color[id]` с `color[id + 1]` меньше), то и разница между ними будет как минимум 2. Если же они различаются в одной и той же позиции, то тогда `color[id + 1]` и `color[id + 2]` имеют разный бит на этой позиции.

Каждый раз количество цветов уменьшается в log раз, поэтому можно дойти до момента $n \to 2 \cdot \log n \to 2 \cdot \log(\log n) \to \ldots C$, где C - некоторая константа. Однако C может оказаться больше, чем 3. Для решения этой проблемы, для каждого цвета составим массив из вершин этого цвета.

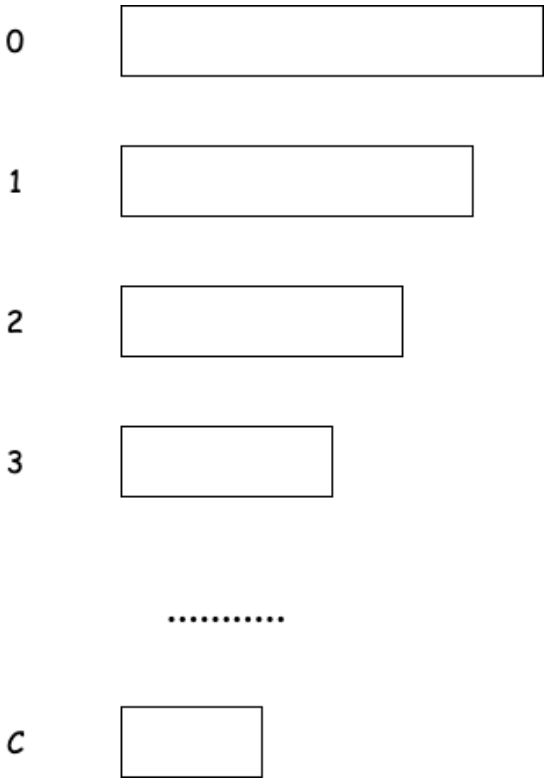


Рис. 4. Массивы с вершинами.

1) Вставляем все вершины из `0`, `1`, `2` (не получатся одинаковыми по прошлому рассуждению).

2) Для всех последующих списков, берём вершину из списка и красим её в минимальный цвет (0, 1, 2), которого не было у соседей, так как соседей 2, а цветов 3 это возможно (если соседи не покрашены, то делаем min с цветами, которые есть).

`Work` $= n \cdot \log^2 n$
`Span` $= polylog(n)$

List Ranking

У нас есть одно связанный список, для каждой вершины известно `next[id]`, надо найти расстояние от каждой вершины до конца.

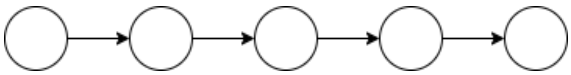


Рис. 5. Односвязанный список

Решения

Двоичные подъёмы

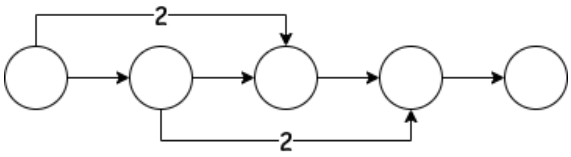


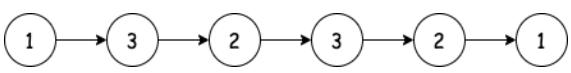
Рис. 6. Двоичные подъёмы.

```
pfor i=1...n
    next'[i] = next[i] + next[next[i]]
    len'[i] = len[i] + len[next[i]]
```

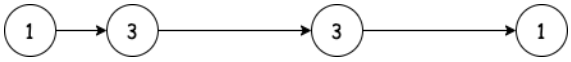
`Work` $= n \cdot \log n$
`Span` $= \log^2 n$

Уменьшение работы при помощи Symmetry breaking

1) При помощи `Symmetry breaking` раскрашиваем вершины в 3 цвета.



2) Удаляем вершины с номером 2.



3) Когда количество вершин станет равным $\frac{n}{\log n}$ запускаем алгоритм `List Ranking`.

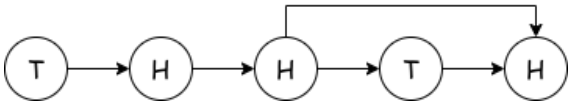
4) После этого начинаем возвращать вершины, мы знаем, что у вершины цвета 2 нет соседа цвета 2 $\Rightarrow$ мы знаем расстояние от её соседа до конца, а значит `len[id] = len[next[id]] + 1`.

`Work` $= \mathcal{O}(n)$
`Span` $= = \mathcal{O}(\log^2 n \cdot \log\log n)$

Randomized List Ranking

Метод позволяющий сократить количество вершин перед `List Ranking`.

1) Для каждой вершины подбрасываем монетку `Head` или `Tail`.



2) Удалим все вершины `H` у которых `next` это `T`. Ожидаемое количество удалённых вершин $= \frac{n}{4}$. Попытка удачная, если нам удалось удалить $\frac{7}{8}$ вершин.

3) Дальше всё так же как и в `List Ranking c Symmetry Breaking`.