ダイクストラ法の高速化について
On speeding up Dijkstra's algorithm

2021/3/27 kaage（@ageprocpp）
@JOI春季ステップアップセミナー

もくじ

・ダイクストラ法 with std::priority_queue が無駄という話
・二分ヒープの話
・高速化

〜以下後日やります〜
・フィボナッチヒープでオーダー改善する話
・フィボナッチヒープへの prioritize 実装

もくじ

・ダイクストラ法 with std::priority_queue が無駄という話
・二分ヒープの話
・高速化

〜以下後日やります〜
・フィボナッチヒープでオーダー改善する話
・フィボナッチヒープへの prioritize 実装

みなさんは

みなさんは
ダイクストラ法を知っていますか？

ダイクストラ法

・けんちょん本 14.6 読んでください

・BFS みたいな感じ

・グラフ上の単一始点最短路問題を解く

・計算量は $O(|V|^2)$ か $O(|E|\log|V|)$（実装により異なる）

・一般的な実装では std::priority_queue を使う

・ただ、std::priority_queue での実装は無駄がある

ダイクストラ法

- けんちょん本 14.6 読んでください
- BFS みたいな感じ
- グラフ上の単一始点最短路問題を解く
- 計算量は $O(|V|^2)$ か $O(|E|\log|V|)$（実装により異なる）
- 一般的な実装では std::priority_queue を使う
- **ただ、std::priority_queue での実装は無駄がある**

ダイクストラ法

```cpp
using IP = std::pair<int, int>;

std::vector<std::vector<IP>> edges;
std::vector<int> dist;

void dijkstra(int s) {
    std::priority_queue<IP, std::vector<IP>, std::greater<IP>> que;
    que.push({0, s});
    while (!que.empty()) {
        auto p = que.top();
        que.pop();
        if (dist[p.second] < p.first) continue;
        for (const auto& e : edges[p.second]) {
            if (chmin(dist[e.first], dist[p.second] + e.second)) {
                que.push({dist[e.first], e.first});
            }
        }
    }
}
```

ダイクストラ法

```cpp
using IP = std::pair<int, int>;

std::vector<std::vector<IP>> edges;
std::vector<int> dist;

void dijkstra(int s) {
    std::priority_queue<IP, std::vector<IP>, std::greater<IP>> que;
    que.push({0, s});
    while (!que.empty()) {
        auto p = que.top();
        que.pop();
        if (dist[p.second] < p.first) continue;
        for (const auto& e : edges[p.second]) {
            if (chmin(dist[e.first], dist[p.second] + e.second)) {
                que.push({dist[e.first], e.first});
            }
        }
    }
}
```

同じ頂点を複数回 push する可能性がある！

ダイクストラ法

priority_queue の中に入るのは高々 $|E|$ 個

*各辺の走査を考えると高々1回の追加で済む

ダイクストラ法

priority_queue の中に入るのは高々 $|E|$ 個

*各辺の走査を考えると高々1回の追加で済む

⬇

$|E| \leq |V|^2, \log|E| \leq 2\log|V|$ から、$O(|E|\log|V|)$ で OK

ダイクスト

priority_que 入るのは高々

*各辺 加で済む

$|E| \leq |V|^2, \log|E| \leq 2$ $(|E|\log|V|)$ で OK

ダイクスト…

priority_que… 入るのは高々…

*各辺の… 追加で済む

$|E| \le |V|^2, \log |E| \le 2$ … $(|E| \log |V|)$ で OK

もったいない！

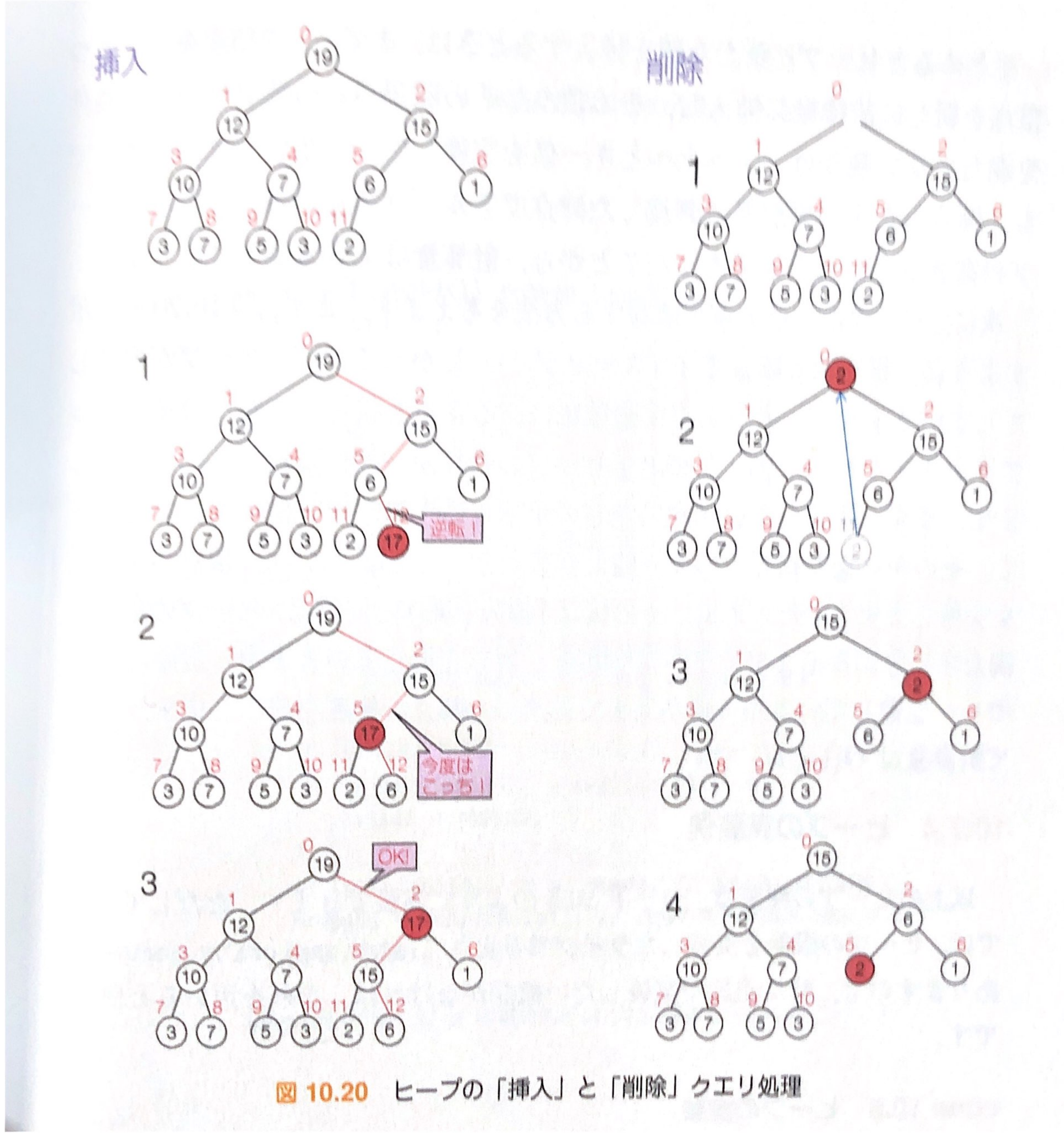
もくじ

・ダイクストラ法 with std::priority_queue が無駄という話
・二分ヒープの話
・高速化

〜以下後日やります〜
・フィボナッチヒープでオーダー改善する話
・フィボナッチヒープへの prioritize 実装

二分ヒープ is 何

けんちょん本を読もう



もくじ

・ダイクストラ法 with std::priority_queue が無駄という話
・二分ヒープの話
・高速化

〜以下後日やります〜
・フィボナッチヒープでオーダー改善する話
・フィボナッチヒープへの prioritize 実装

高速化

最短距離が更新されたとき、queue の中にすでに値があると queue が膨らむ

高速化

最短距離が更新されたとき、queue の中にすでに値があると queue が膨らむ

⬇

queue の大きさは $|V|$ で本来は十分だから、これを超えたくない

高速化

最短距離が更新されたとき、queue の中にすでに値があると queue が膨らむ

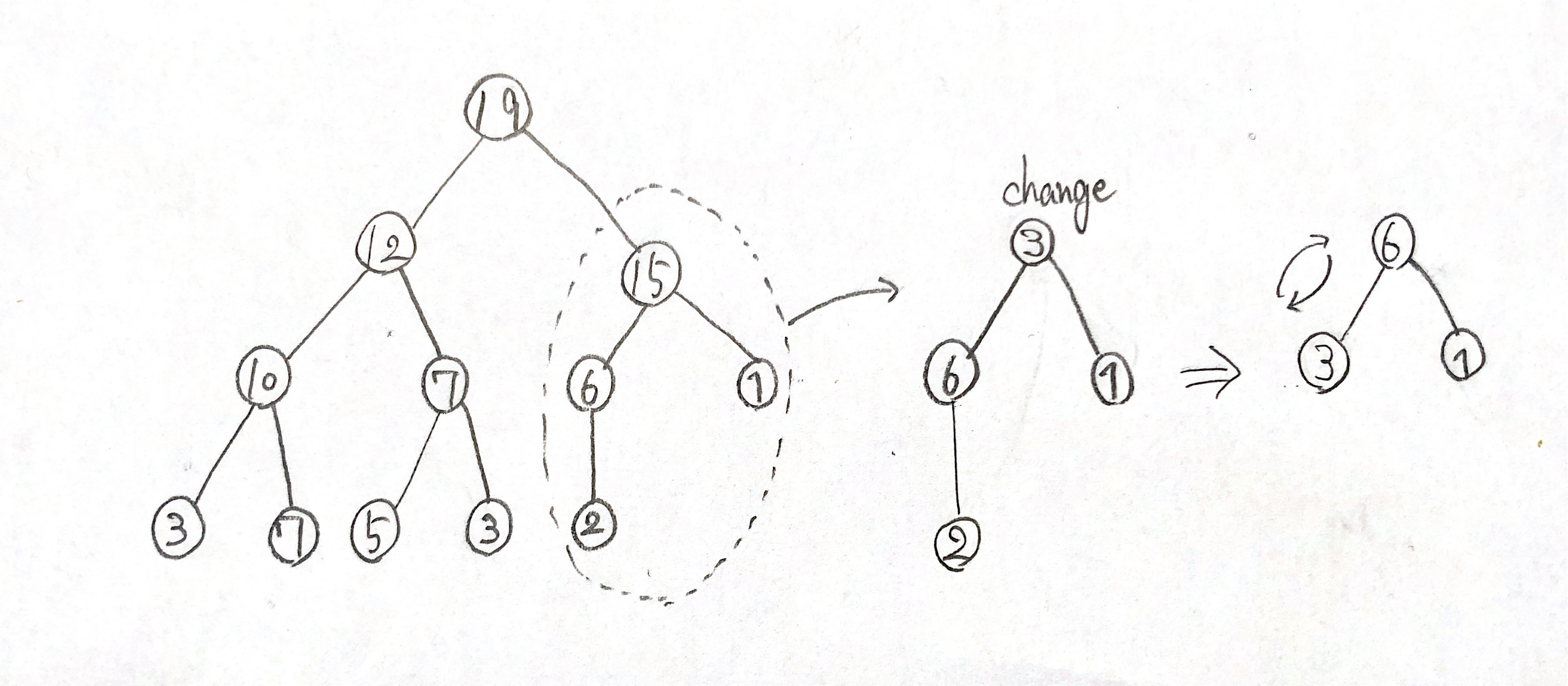
queue の大きさは $|V|$ で本来は十分だから、これを超えたくない

queue の中の値を queue に入れっぱなしのまま更新 (prioritize) できれば良い！

二分ヒープ is 何（再）

変えたい頂点の位置がわかれば簡単に変更可能

頂点の位置は、毎回の操作と同時に更新すれば良い



要するに…

- 二分ヒープを実装する
- 頂点番号→ノードのリンクを保持して一緒に更新する
- $O(\log N)$ で更新可能（そのぶん pop が減るからよい）
- PrioritizableBinaryHeap と呼んでいます
- ヒープが小さくなって速くなる！（うれしい）

実装

こんな感じ（コード一部）
全体は github に置いてあります

```cpp
void down_heap(int id = 1) {
    while ((id << 1) < heap.size()) {
        int il = id << 1, ir = id << 1 | 1, swap = -1;
        auto &vl = heap[il], &vx = heap[id];
        if (comp(vx.second, vl.second)) swap = il;
        if (ir < heap.size()) {
            auto& vr = heap[ir];
            if (comp(vx.second, vr.second)) {
                if (swap == -1 || comp(vl.second, vr.second))
                    swap = ir;
            }
        }
        if (swap == -1) return;
        std::swap(rev[vx.first], rev[heap[swap].first]);
        std::swap(vx, heap[swap]);
        id = swap;
    }
}
```

ベンチマーク

3回提出して平均をとりました

	std::priority_queue	PrioritizableBinaryHeap	$O(\|V\|^2)$
SHIPC2018-D	254ms	250ms	-
ARC064-E	131ms	51ms	60ms
yosupo judge	423ms	424ms	-

まとめ

- 密グラフだと明らかに速い（それはそう）
- 疎グラフでもそんなに遅くなってはいなさそう
- 遅くなるということはなさそうなので、差し替えて損はない

さらなる高速化

- フィボナッチヒープを使う
- prioritize が $O(1)$ でできる
- gcc の Policy Based Data Structures にあるらしい
（tree<int, null_type, less<int>…> で有名なアレ）
- オーダーが改善できる
- $O(|E|\log|V|)$ から $O(|E|+|V|\log|V|)$ に
- 時間がなかったのでやってません

ご静聴ありがとうございました