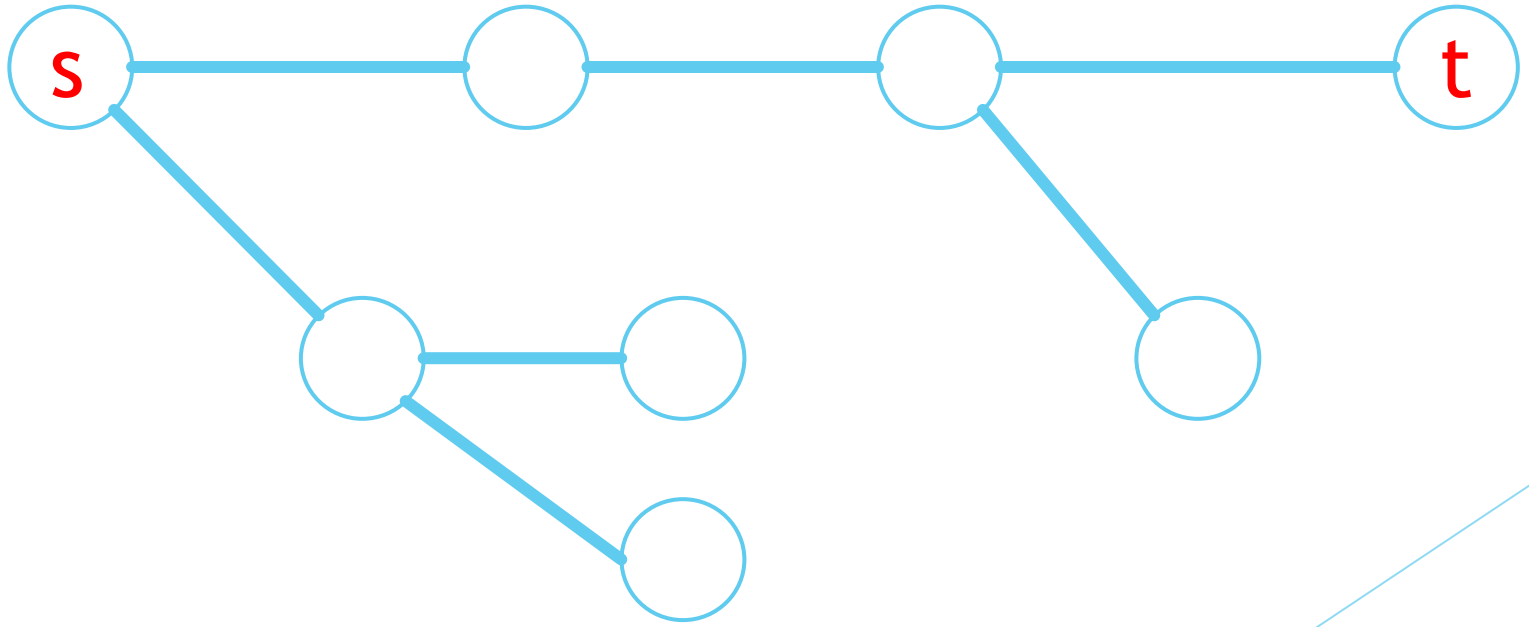
G: Detour

原案・解説　tardigrade（akTARDIGRADE13）

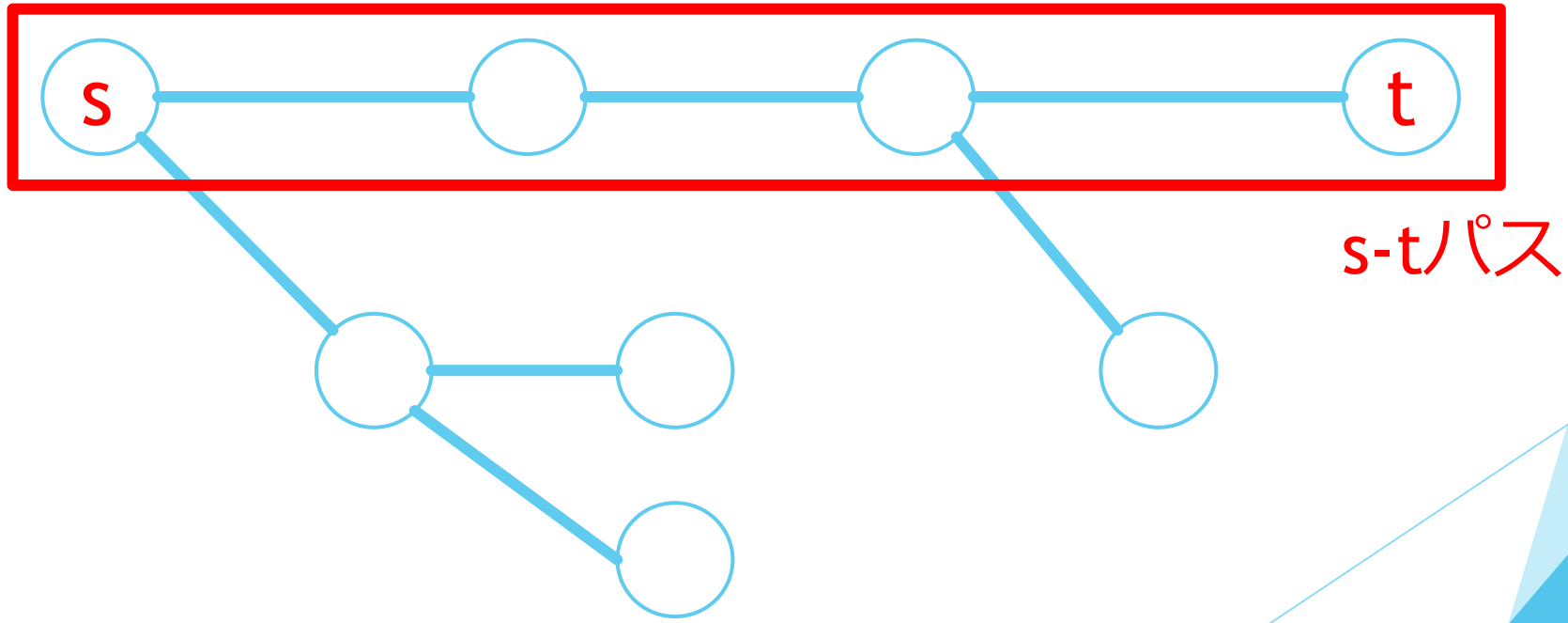
1. O(QN) 解法

- k を全探索して愚直に計算するとークエリあたり O(N^2)
- 下のように木と s_i, t_i が与えられた与えられた例を考える



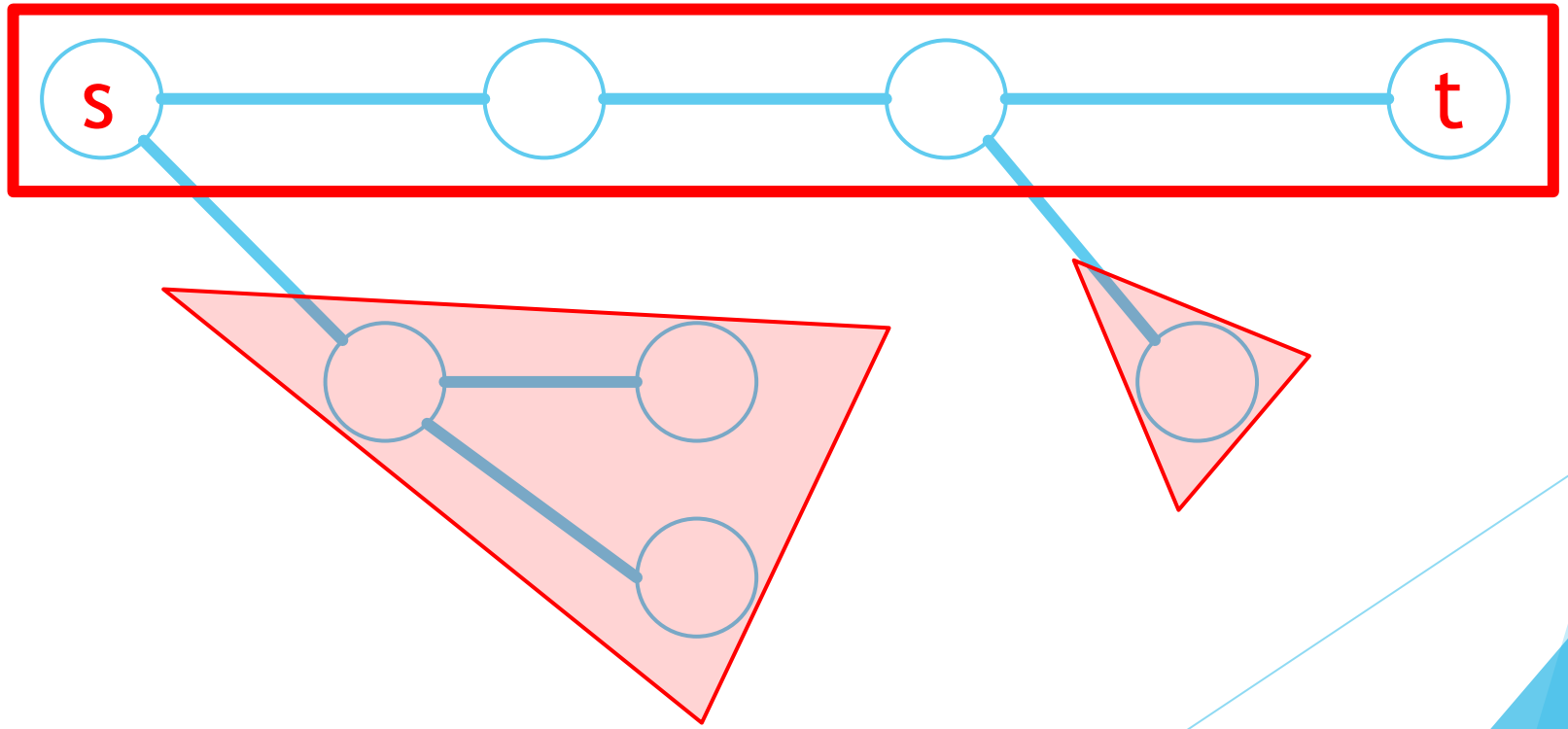
1. O(QN) 解法

- s-t パス上の頂点の重みは k の値に関わらず必ず加算される
- 同上はークエリあたり O(N) で計算可



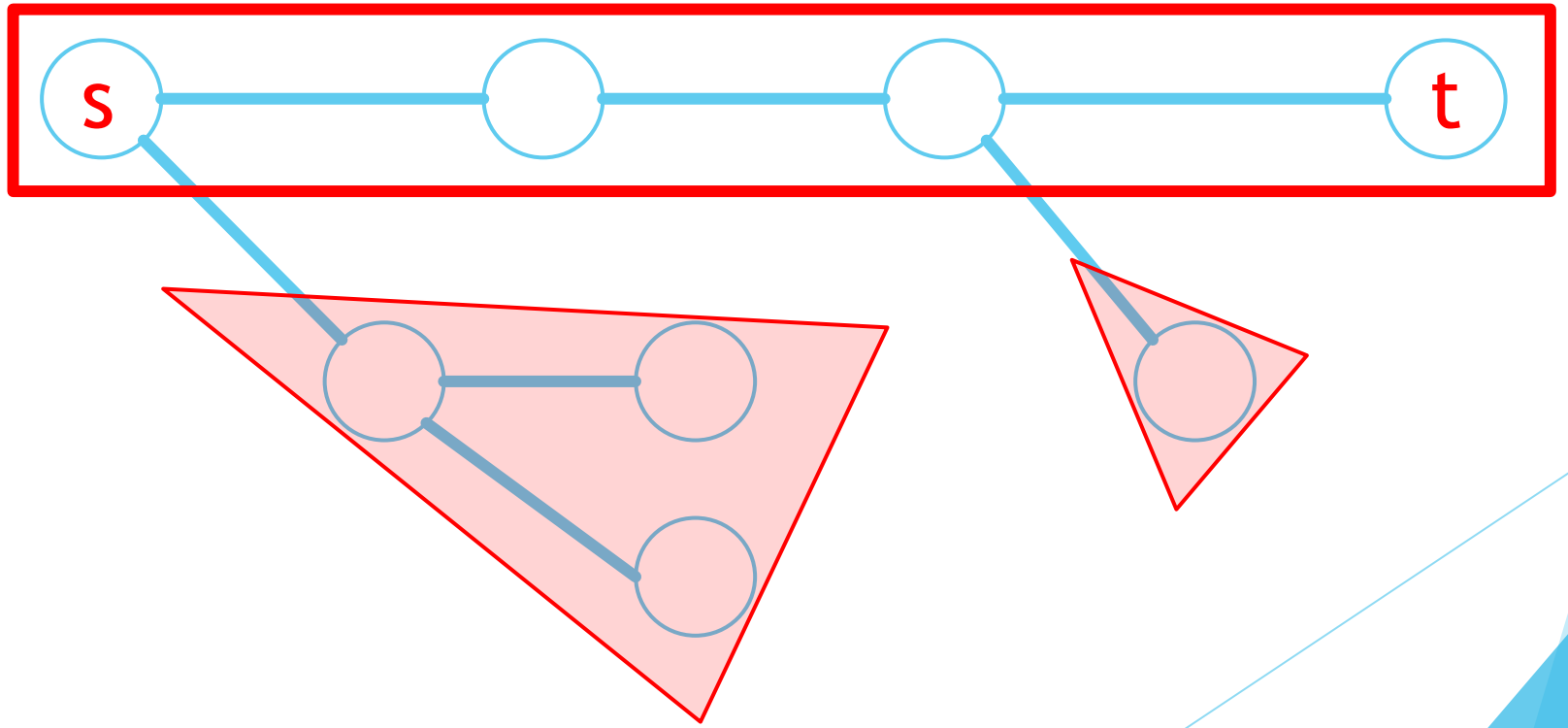
1. O(QN) 解法

- k を選んで新たに加算される重みの最大値がわかればよい
- s-t パス上の頂点を k として選んだ時、新たに加算される値は0
- **△の部分木**に着目する



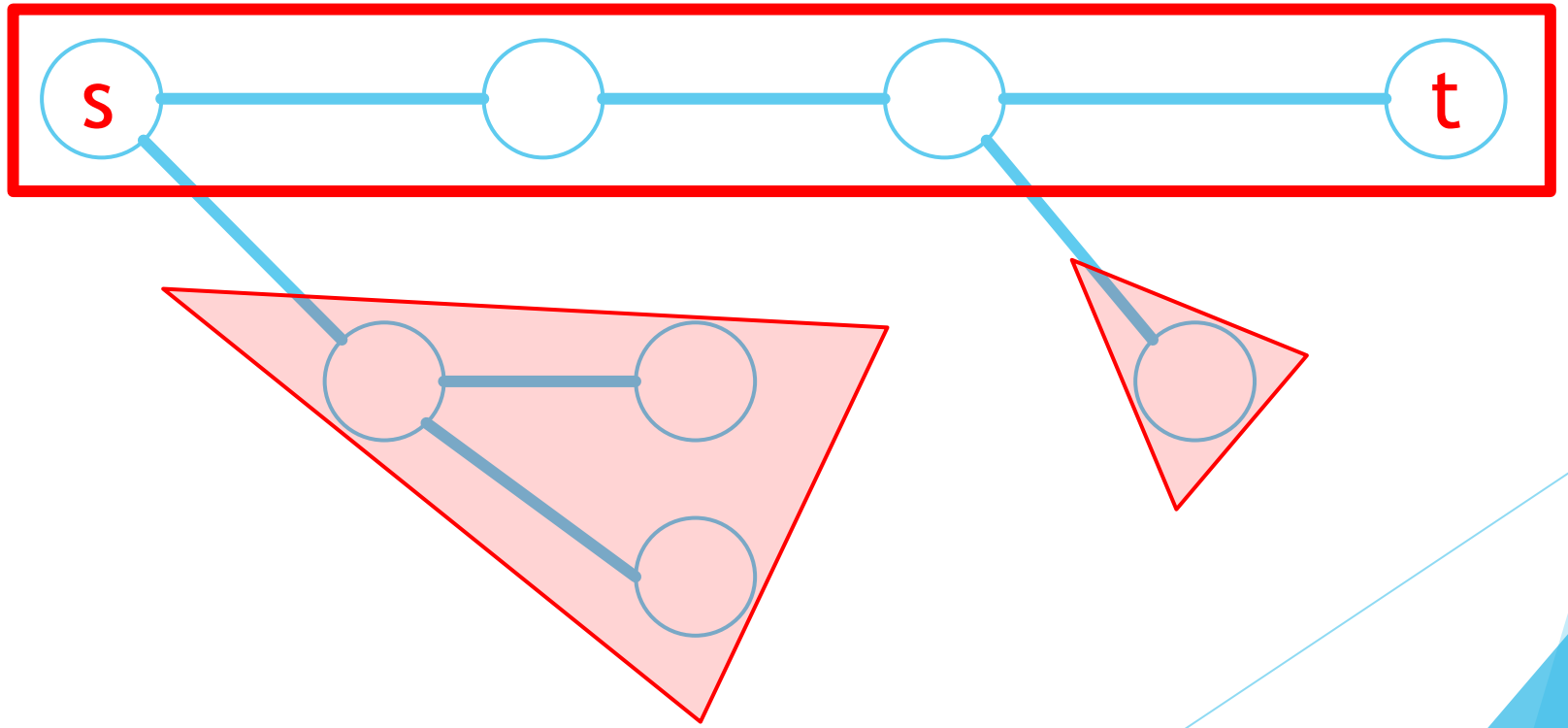
1. O(QN) 解法

- **△の部分木**の根からのパスを考える
- そのようなパスの重みの最大値は木 DP によって求まる
- 前計算で全方位木 DP をすると O(1) で値を持ってこれて嬉しい



1. O(QN) 解法

- **△の部分木**の値の最大値が、 新たに加算される重みの最大値
- 次数が多いと計算量がやばそうだが、各頂点について、値が大きい部分木 top3 を持っておけば十分



1. O(QN) 解法

- まとめ
- 前計算で全方位木 DP をする
  - 比較的単純なものなので詳細は割愛。頂点の重みが負の値を取る可能性があることに注意
- 各クエリごとに独立に計算
- s-t パス上の頂点の重みの総和は O(N) で計算可
- 経由地として k を追加したときに新たに加算される値を考える
  - 新たに追加される値は0以上になることに注意
- 図で表したような△**の部分木**を列挙してその値の最大値を取る
  - top3 だけ管理しておくと O(N)
- これで一クエリあたり O(N) 、全体で O(QN)

2. 高速化

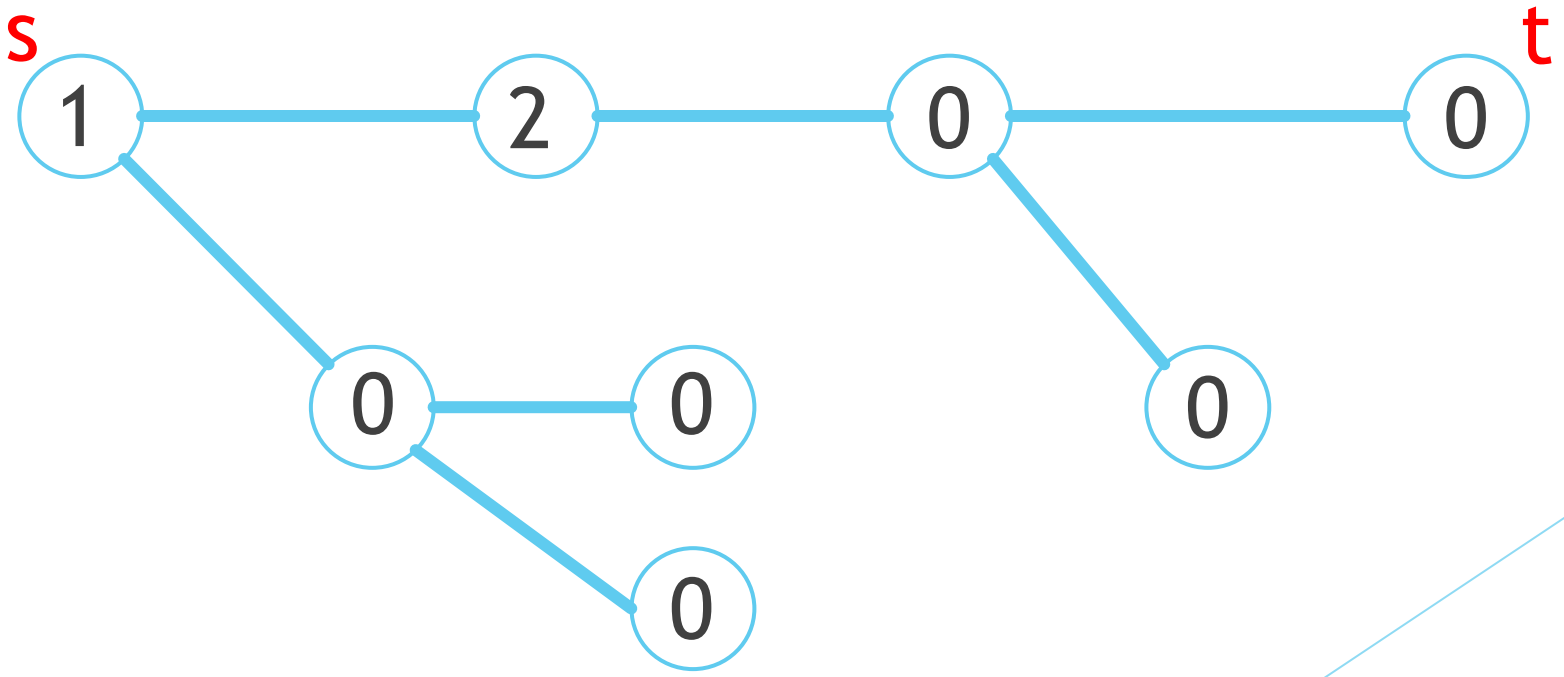
- s-t パス上の頂点の重みの総和は、HLD と累積和を用いて O(logN) で求めることができる
  - BITやセグメント木を用いると O((logN)^2)　（本問題の制約だとこちらでもOK）

- **△の部分木**の値の最大値も、似た感じで高速に取得できない？
- HLD と static RMQ を用いて O(logN) で求められたら嬉しい
  - セグ木を用いると O((logN)^2)　（本問題の制約だとこちらでもOK）
  - static RMQ だと長いので、以下セグ木で統一します

- セグ木に何を載せるかが重要になる

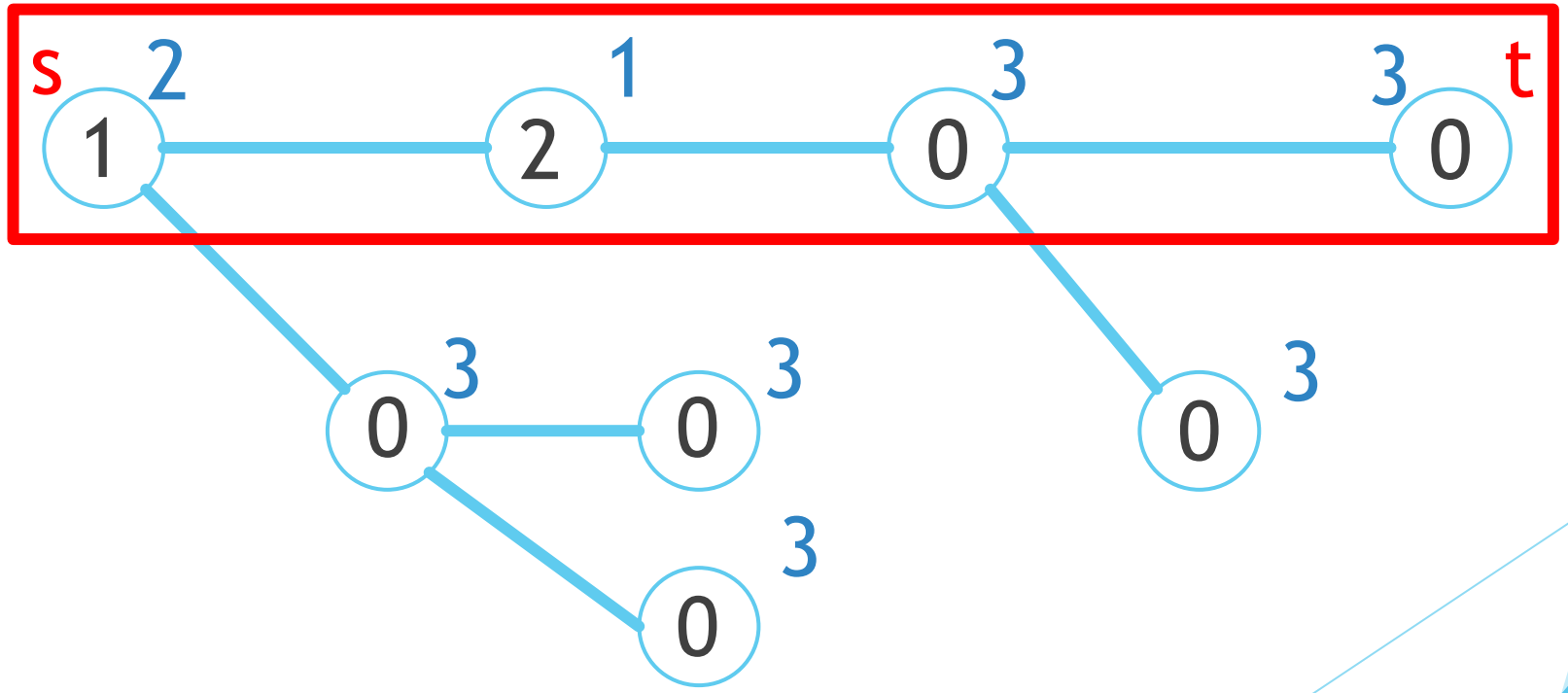
2. 高速化

- ダメな例
- 「各頂点と隣接する部分木の値の最大値」を載せる
- 下のように各頂点に重みがつけられているとする



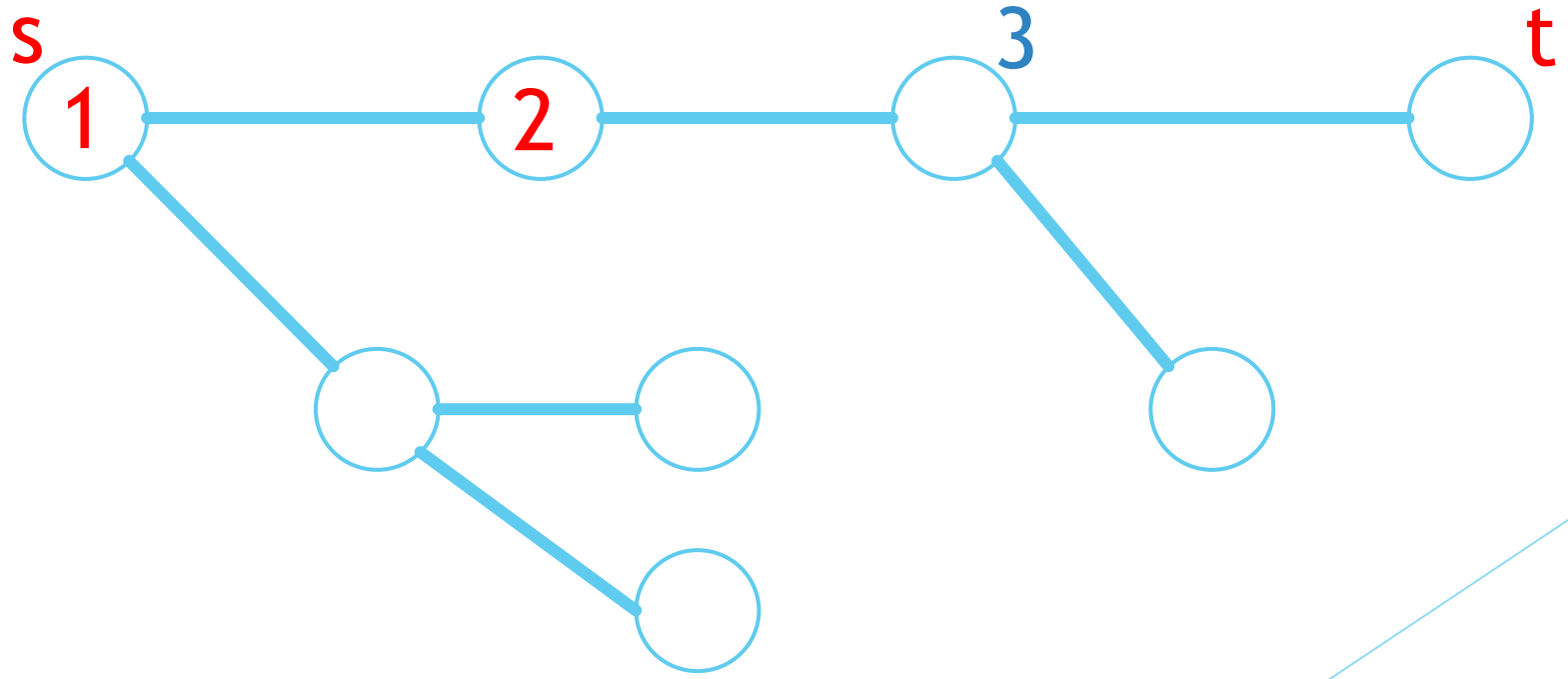
2. 高速化

- 青い値がセグ木に載っている値
- s-t パス上の最大値は 3 だけど…



2. 高速化

- 新たに加算される値に、s-t パス上の頂点の重みも含めてしまっている
- これだと正しい値が計算できない



2. 高速化

- そもそもの話
- HLD ってなんで / どんな時にパスクエリを高速に処理できるの？

- HLD は各辺を「Heavy な辺」と「Light な辺」に分ける
- Heavy な辺で連結している頂点たちについては、セグ木を使って高速に値を取得できる（高速に辿れる）
- 連結成分は O(logN) 個（そうなるように分解する）だから嬉しい

- セグ木で高速に計算できて、答えが壊れないような値を載せたい

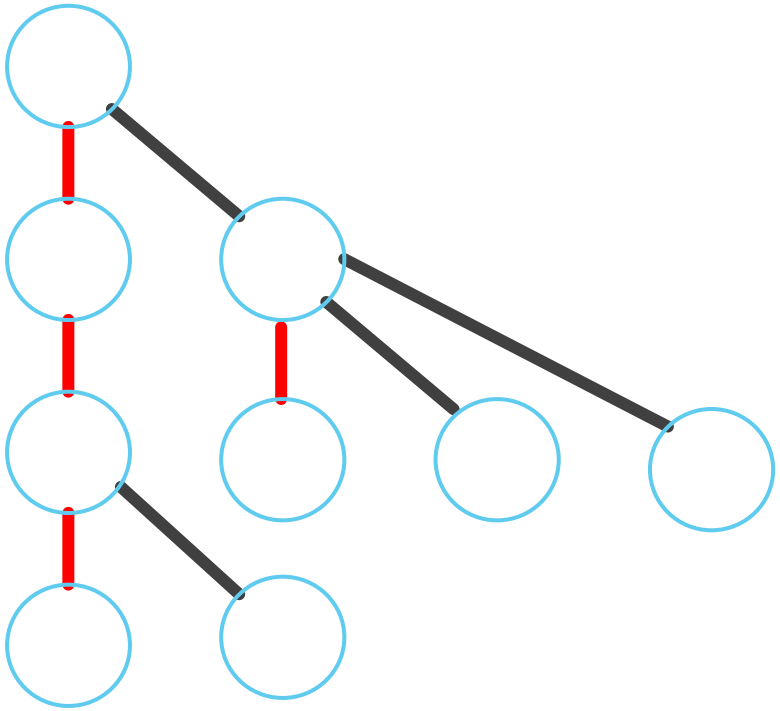
2. 高速化

- 結論
- 「各頂点と隣接する部分木の値の最大値」を載せる
- **※但し、親方向の部分木と Heavy な 辺で連結している方向の部分木は除く**

- 補足
- 「Light な辺で連結している親方向の部分木は含める」とする実装方針もあると思います
- おそらくどの方針を選んでも、場合分けはそこそこ必要です
- **赤字**よりも簡単な方針を思いついた方がいたら教えてください

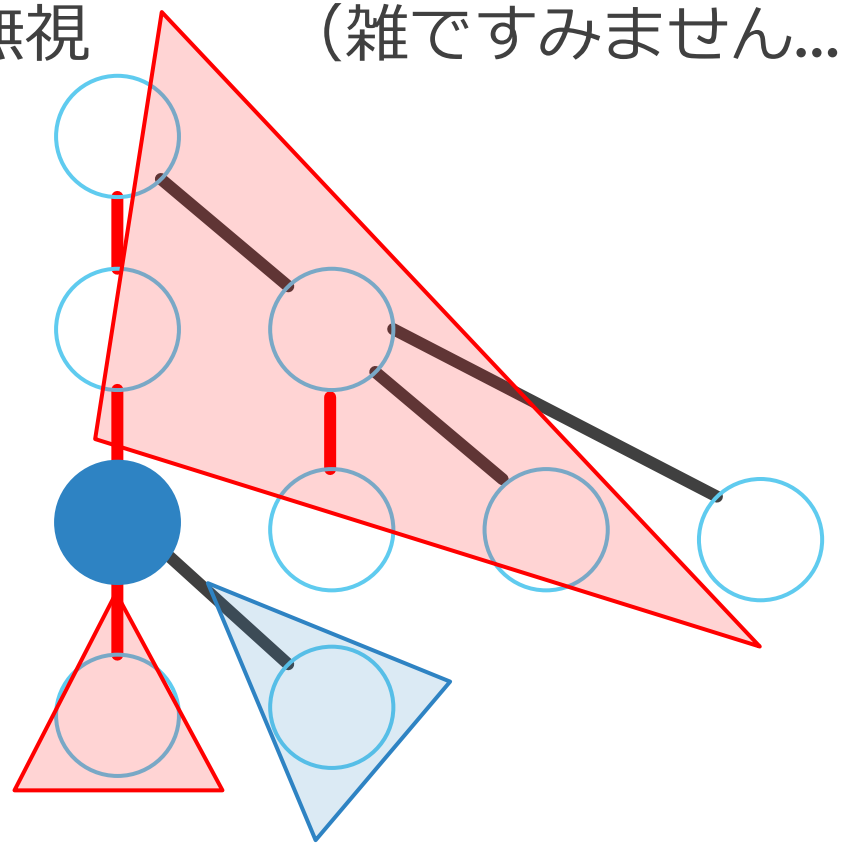
2. 高速化

- HLD の図解
- 赤線が Heavy な辺、黒線が Light な辺



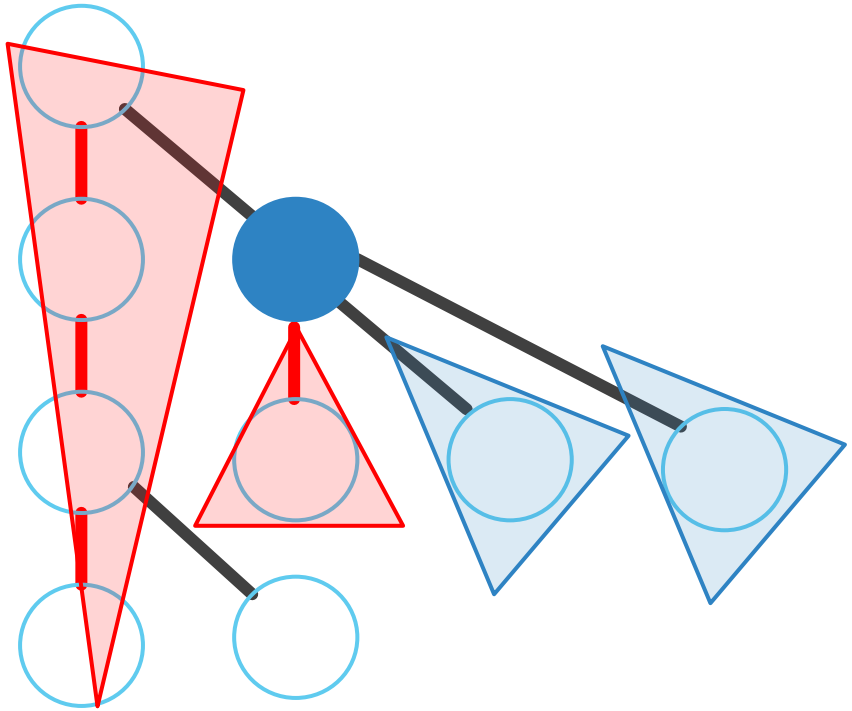
2. 高速化

- HLD の図解
- 青の頂点には△の部分木の値の最大値を載せる
- △の部分木は無視　　（雑ですみません…）



2. 高速化

- HLD の図解
- 青の頂点には△の部分木の値の最大値を載せる
- △の部分木は無視　　（雑ですみません…）



2. 高速化

- まとめ
- 前計算で全方位木 DP をする
  - 比較的単純なものなので詳細は割愛。頂点の重みが負の値を取る可能性があることに注意
- 各クエリごとに独立に計算
- s-t パス上の頂点の重みの総和は、HLD と累積和を用いて O(logN) で求めることができる
- 経由地として k を追加したときに新たに加算される値を考える
  - 新たに追加される値は0以上になることに注意
- これも HLD と static RMQ を用いてO(logN) で求められる
- 全体で O(N+QlogN)（データ構造によってはO(N+Q(logN)^2) ）
- 本問題の制約ではlog2つでも通ります

3. 実装上の注意など

- セグ木の値を使えない場合があることに注意
  - Light な辺を辿って到達した頂点
  - 実装の方針によってはもっといろいろあると思います
- LCA(s,t)・パスの端に注意
  - セグ木の値が使えない場合と、使えるけどそれだけでは不十分な場合が存在します
  - 上の頂点と同様の場合分けを行うのがわかりやすいと思います
- Top3 だけを管理しよう
  - 場合分けの際に、隣接するすべての部分木を調べようとすると、ウニグラフでTLEします
- Heavy な辺で連結している子ってどうやって持ってくるの？
  - 一般的なHLD実装だと、親はもともと管理していることが多いと思います
  - 子も実は簡単で、セグ木上で隣接している頂点を見てあげればよいです