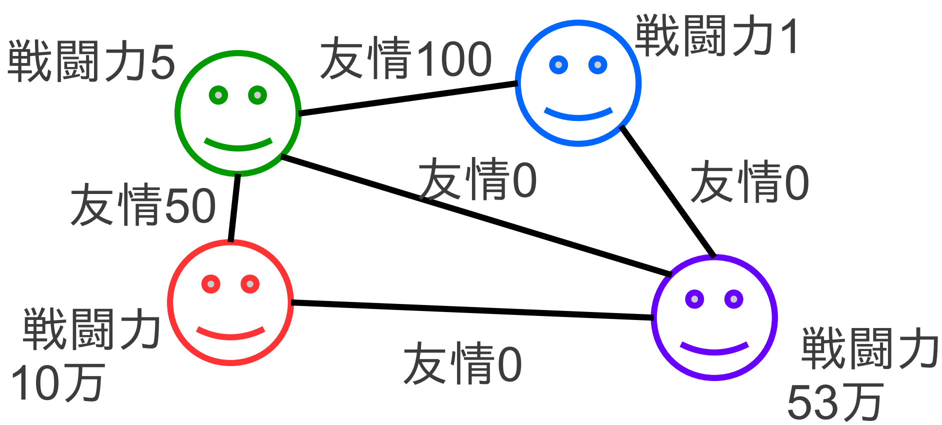
HCPC勉強会 2016/01/07

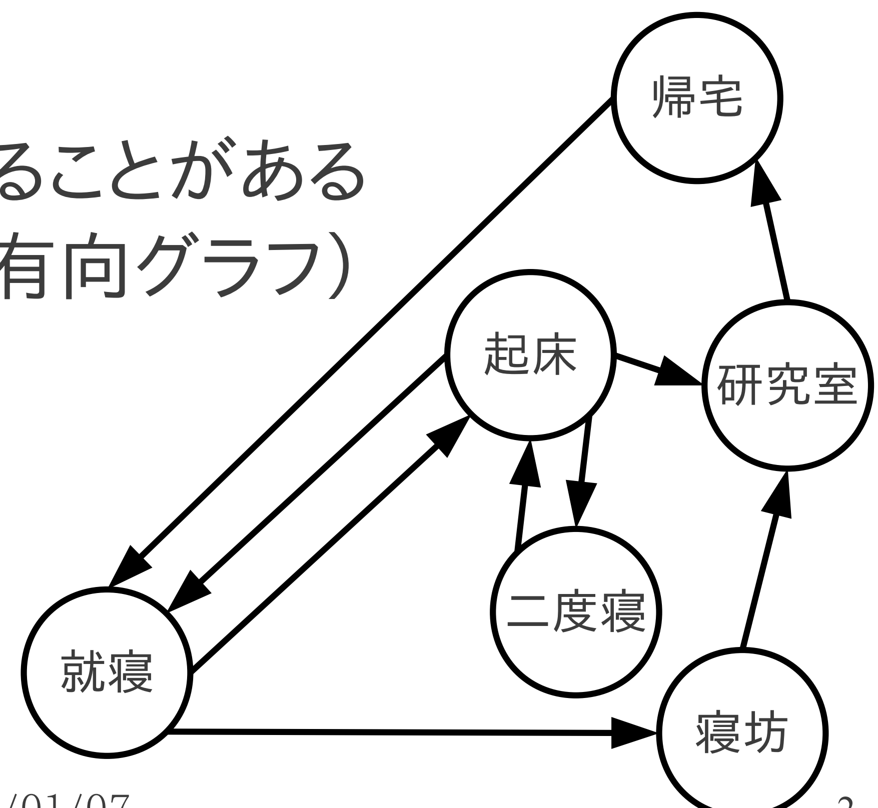
グラフマスターに、俺はなる！

M2　鈴木　浩史

さらっと事前知識：グラフとはなんぞや

- グラフ
  - 頂点（○ぽいやつ）を辺（線分）で結んだ図形
    - example
      - ○が人 → 友人関係の図
      - ○が状態 → 状態遷移図
  - 頂点や辺には値が付けられることがある
  - 辺は向きを持つことがある（有向グラフ）



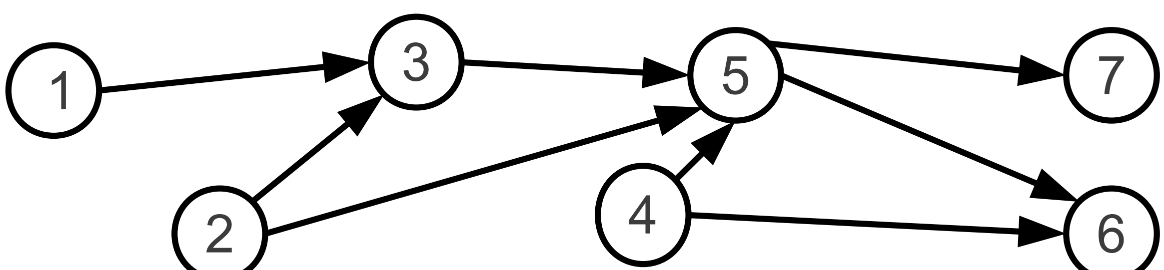


さらっと事前知識:グラフの重要な用語

- 閉路
  - ある頂点から開始して同じ辺を2度通らずに戻る路
- 連結
  - 辺に向きがないとして全ての頂点が互いに行き来できる
- 木
  - 閉路のない連結なグラフ
    - 今回は特に辺に向きがないことにする
  - 頂点に葉、子孫、先祖、根など特別な呼び方がある
- 強連結
  - 辺に向きがあるとき、全ての頂点が互いに行き来できる

さらっと事前知識：DAG

- 非巡回有向グラフ（Directed Acyclic Graph : DAG）
  - 辺に向きがあり閉路のないグラフ
- トポロジカル順序
  - 各頂点が自身より若い頂点に辿りつけないような順序
    - 求め方
      - 向けられている辺の数が0の頂点を消して番号付けることを繰り返す
      - DFSの帰りがけ順が遅いものから番号付けをする
  - 一意的ではない（大体の場合で気にしないけど）



本スライドでの注意点

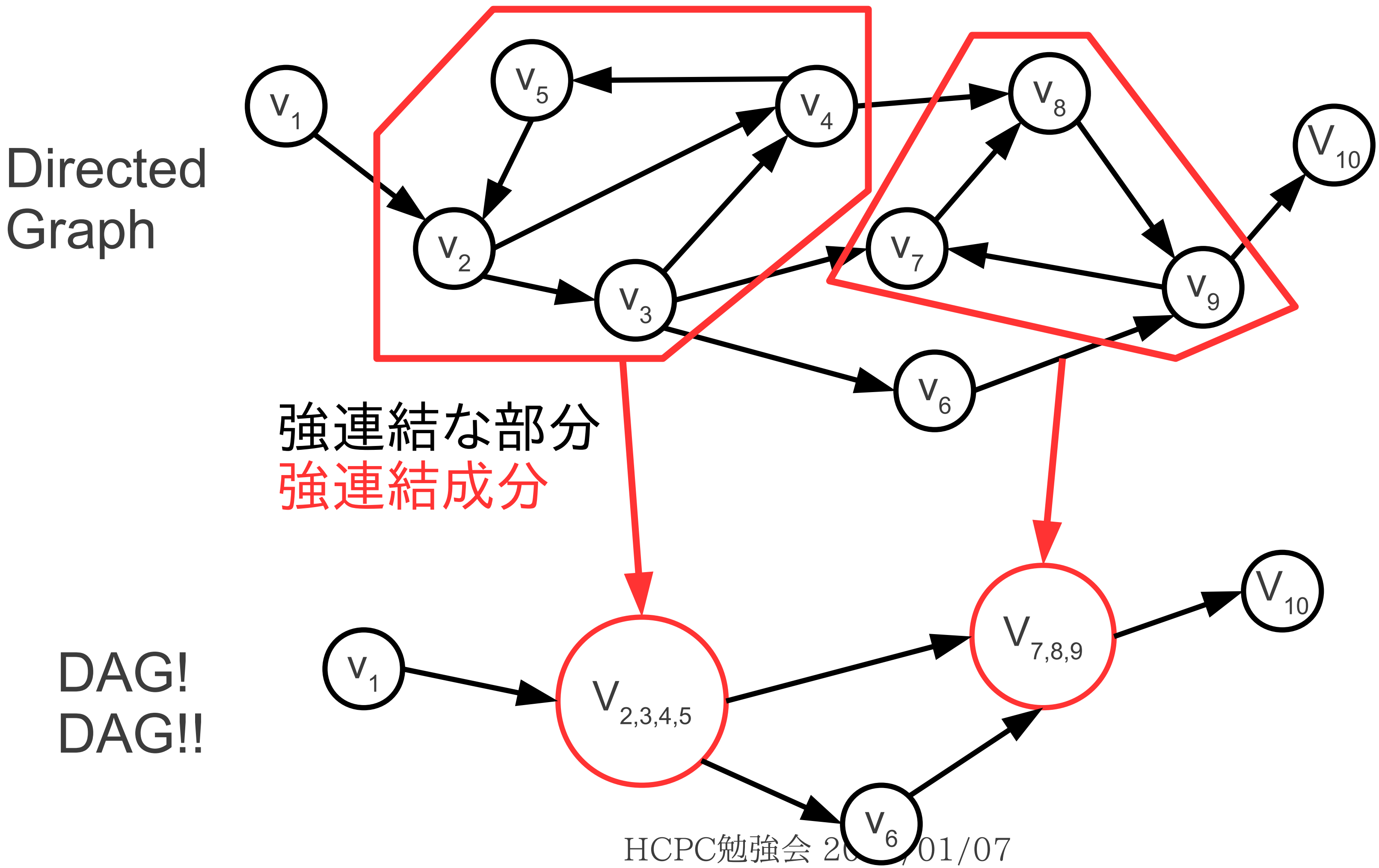
- 今後、グラフの頂点数をN、辺の数をMと記述する。
- 本スライドを見ただけでは、コードを書けるようにはなりません。頑張ってください。
  - 困ったら -tsukasa_diary github [検索]-
  - いや、蟻本読めよ
- 蟻本読め

本スライドの主な内容

- 強連結成分分解
  - 任意の有向グラフをDAGに変換するお話
- 最小共通先祖
  - 根を持つ木における重要なクエリのお話
- H/L分解
  - 根を持つ木に対する良い性質を持った分解のお話
  - ぶっちゃけ、競プロには必要ない
    - 知ってると無理やり問題を殴れるだけ
    - 趣味でしかない

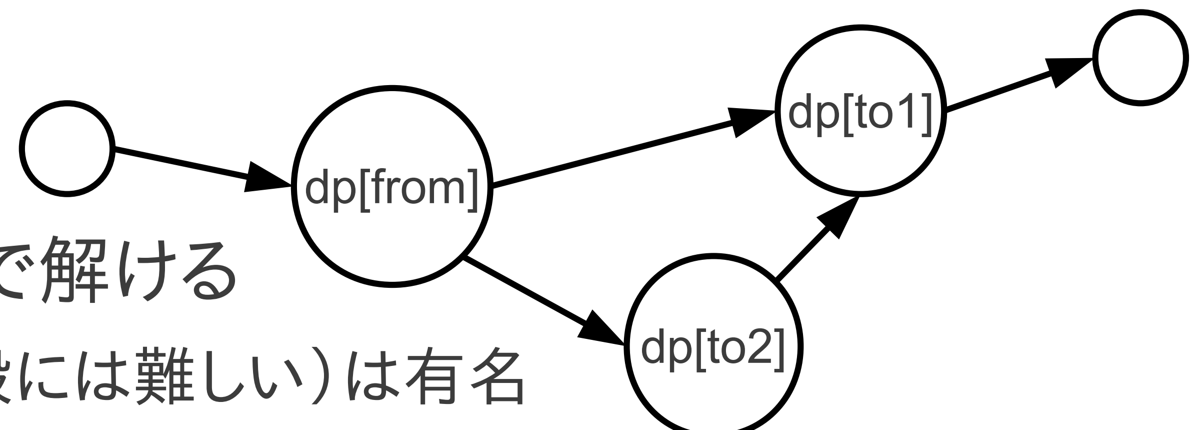
強連結成分分解

- 有向グラフの強連結な部分を潰してDAGにする



何が嬉しいの？

- 2-SATを爆速で解く（蟻本を読め）
  - 強連結成分を考えることで解ける問題はたくさん
- ~~DAGDAGできる！~~
- DAGは強い
  - 色々な問題がDPで解ける
    - 最長路問題（一般には難しい）は有名
    - 強連結成分ごとに前処理して、DAGに潰してDPするのは典型
  - DPはDAGをトポロジカル順にたどるアルゴリズムと考えることができる

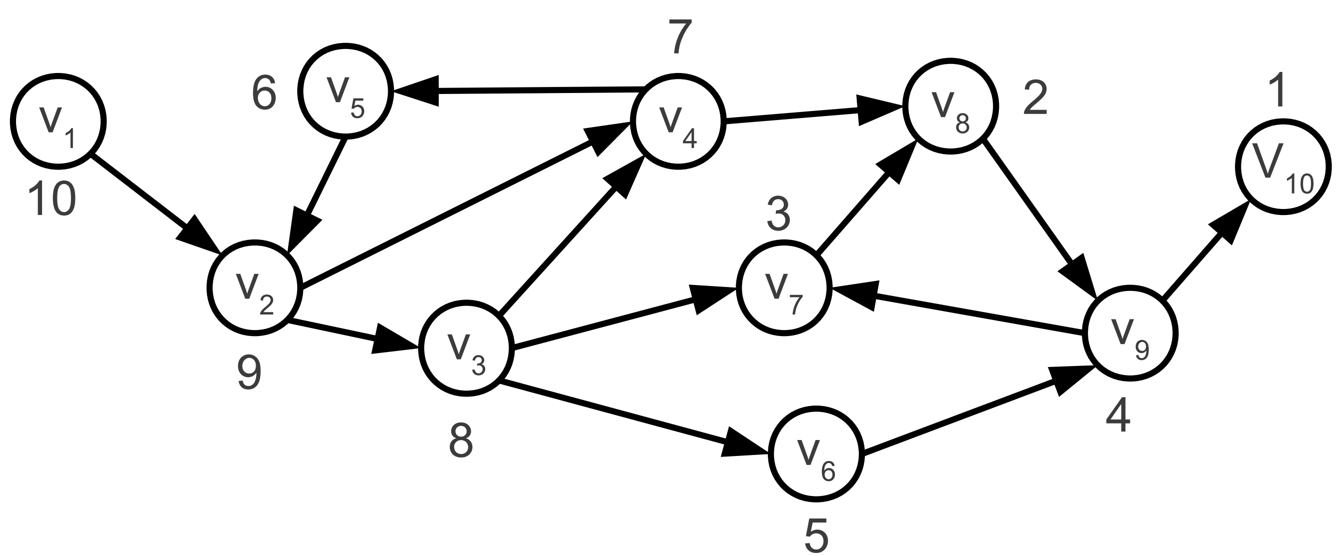


強連結成分分解のアルゴリズム
part-1

- DFSを2回するだけ
  - 1回目：帰りがけ順を記憶

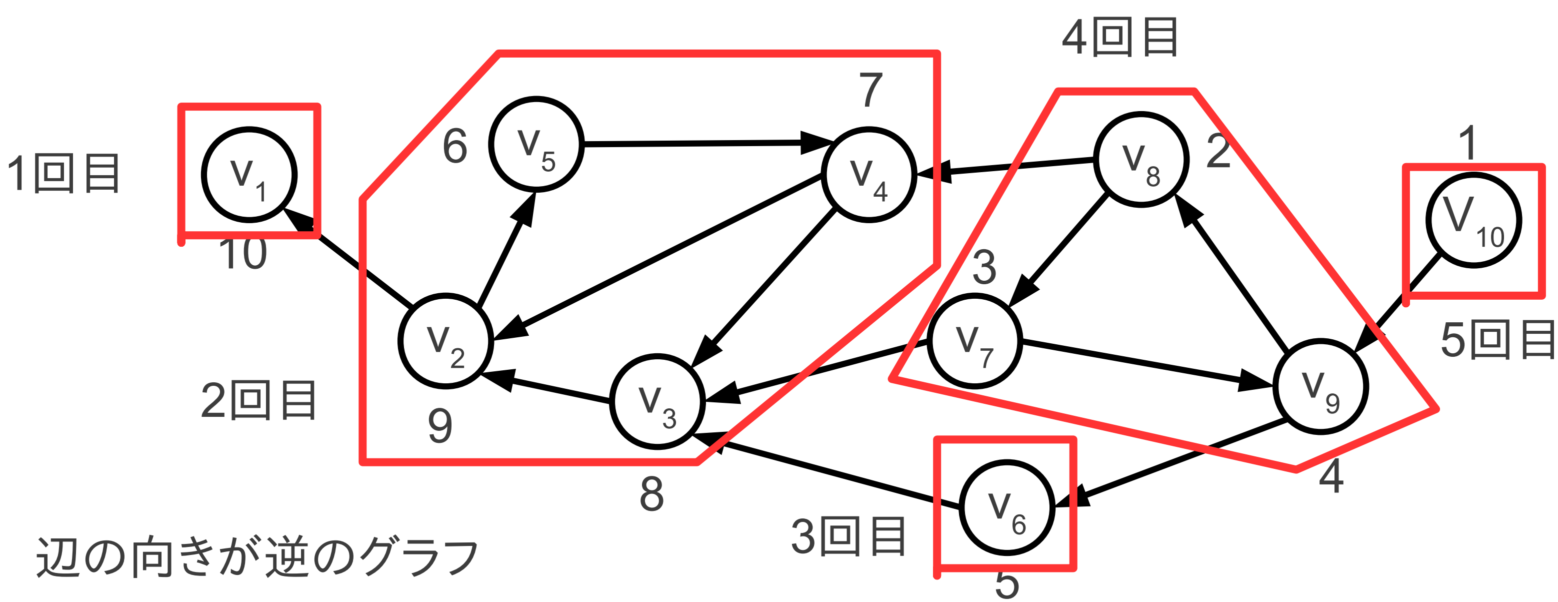
例えばこんなDFSをしたとする（赤はその頂点から帰るタイミング）

$v_1 \to v_2 \to v_3 \to v_6 \to v_9 \to v_{10} \to v_9 \to v_7 \to v_8 \to v_7 \to v_9 \to v_6 \to v_3 \to v_4 \to v_5 \to v_4 \to v_3 \to v_2 \to v_1$

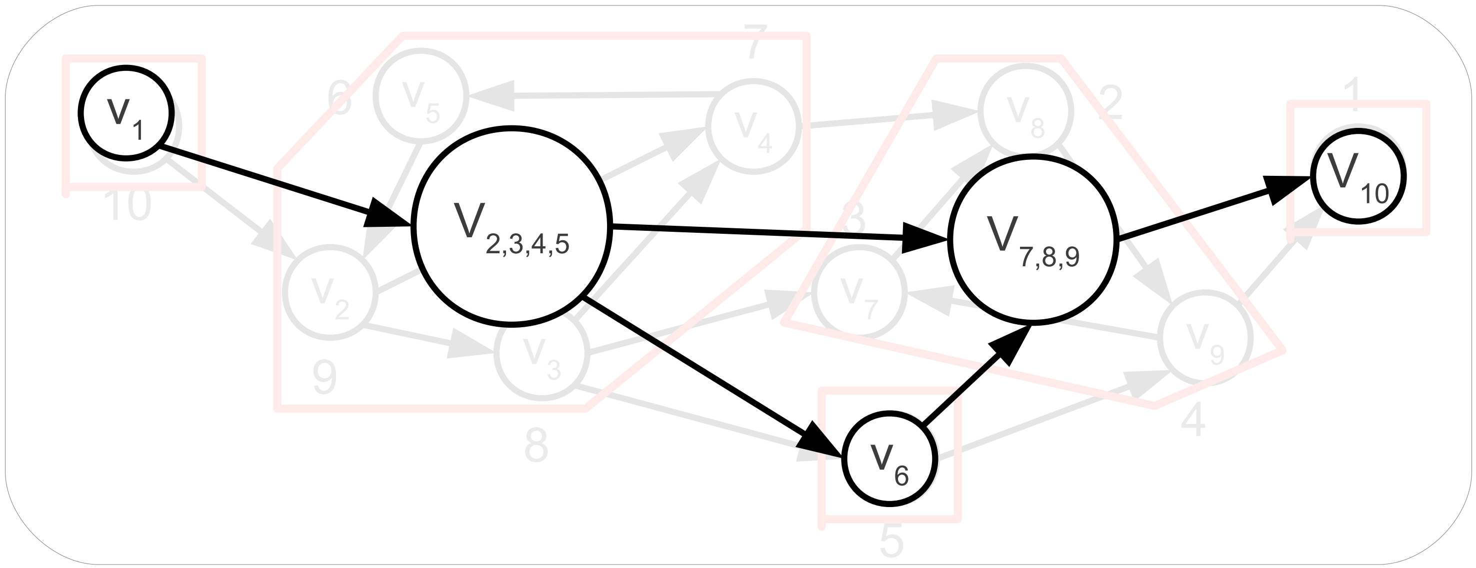


強連結成分分解のアルゴリズム
part-2

- DFSを2回するだけ
  - 2回目：帰りが遅かった頂点から順に逆辺を使ったDFS
  - 何が起こるのか？
    - すでに訪れた頂点を避けると、強連結成分ごとにDFSが停止



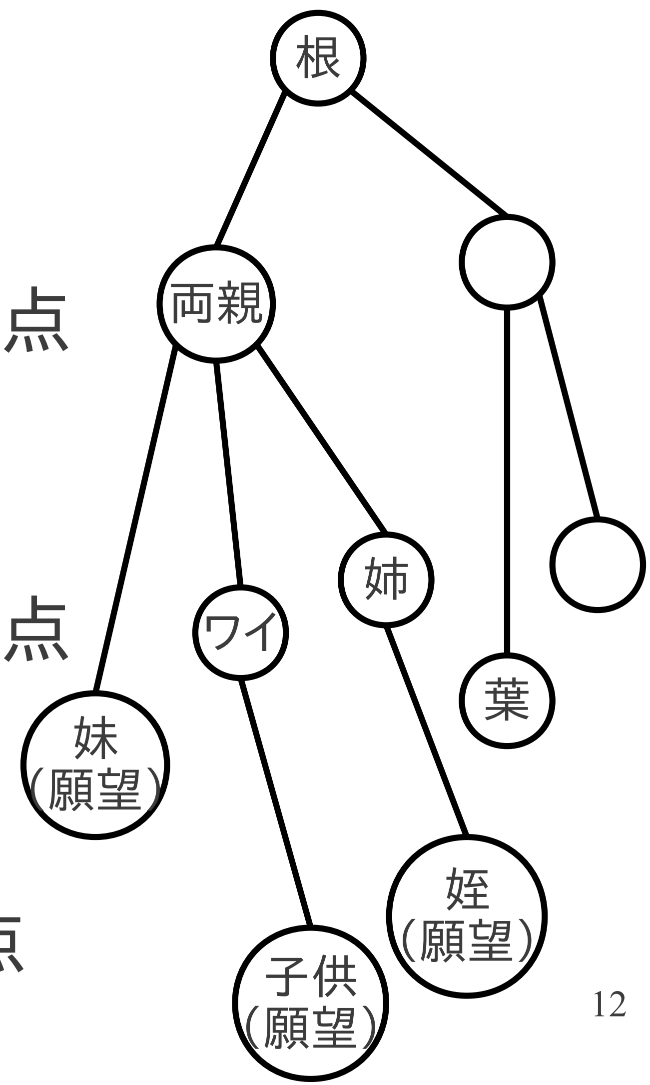
1度にたどった頂点をまとめれば完成



- DFSするだけなので計算量はO(N+M)
- なぜうまくいく？
  - 強連結な部分は辺を逆向きにしても強連結
    - 正方向で全部辿れるなら逆方向でも辿れる
  - DFSの帰りがけ順が遅い = 完成したDAGでトポ順が早い
    - 完成DAGでのDFS帰りがけ順を考えれば当然
  - 逆辺のDFSならトポ順で後の強連結成分を訪れない
  - つまり､トポ順に強連結成分を舐めることになる

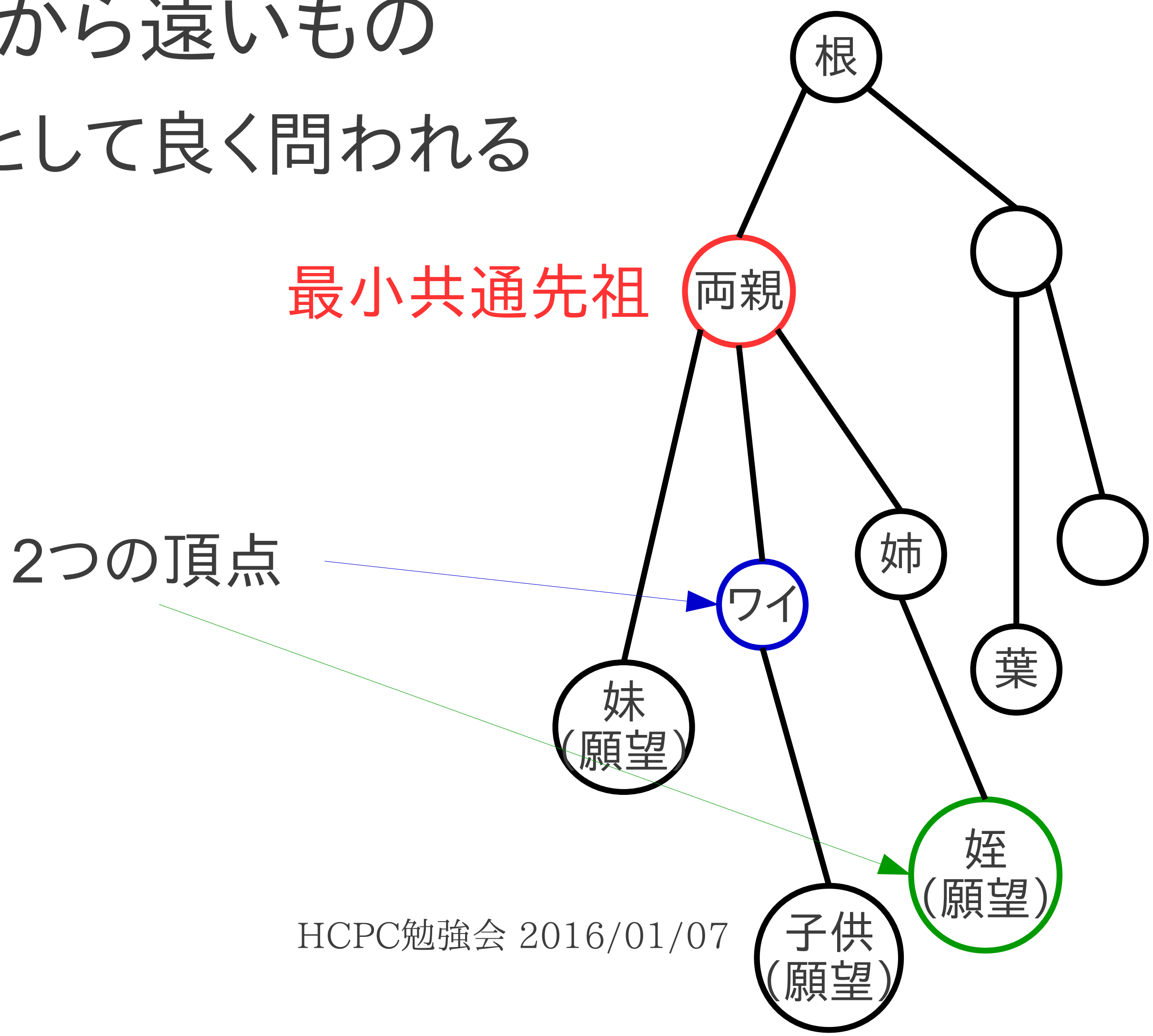
さらっと事前知識：木について

- 根
  - 一番上の頂点
  - これを定めた木を根付き木と言う
- 先祖
  - 根を定めたとき自分より上にいる頂点
  - 自分自身も含むことに注意
- 子孫
  - 根を定めたとき自分より下にいる頂点
- 葉
  - 辺が1本しか出ていない頂点
  - 根を定めたとき子孫を持たない頂点



最小共通先祖(LCA)

- 根付き木のある2頂点について、共通する先祖のうち最も根から遠いもの
  - クエリとして良く問われる

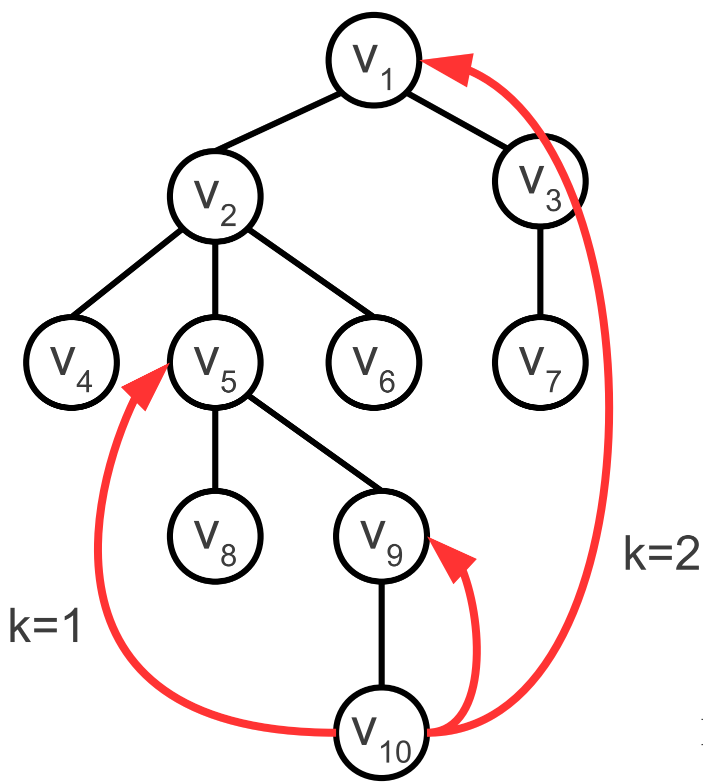


二分法を用いて求める

- 着眼点
  - uとvの高さが同じなら、それぞれから「ある値だけ登った位置」以降にu,vの全ての共通先祖がある
  - 「ある値だけ登った位置」がLCA
    - uとvが初めてぶつかる位置がLCAということ
- やること
  - 前処理：各頂点について深さと2^k個上の先祖を記憶
  - クエリ処理：二分法で一気に登る

前処理

- DFSで各頂点の深さと1つ上の先祖を記憶
- 各頂点の2^k個上の先祖を求める
  - kは$2^k > max_{depth}$となる最小のkまで考えれば良い
    - だいたいは雑に k = log(N)とする
  - 2^k個上の先祖 = 2^{k-1}個上の先祖の2^{k-1}個上の先祖



k = 0 はDFSによってわかっている
kを1から初めて以下のDPをすれば良い！

parent[v][k] = parent[parent[v][k-1]][k-1]

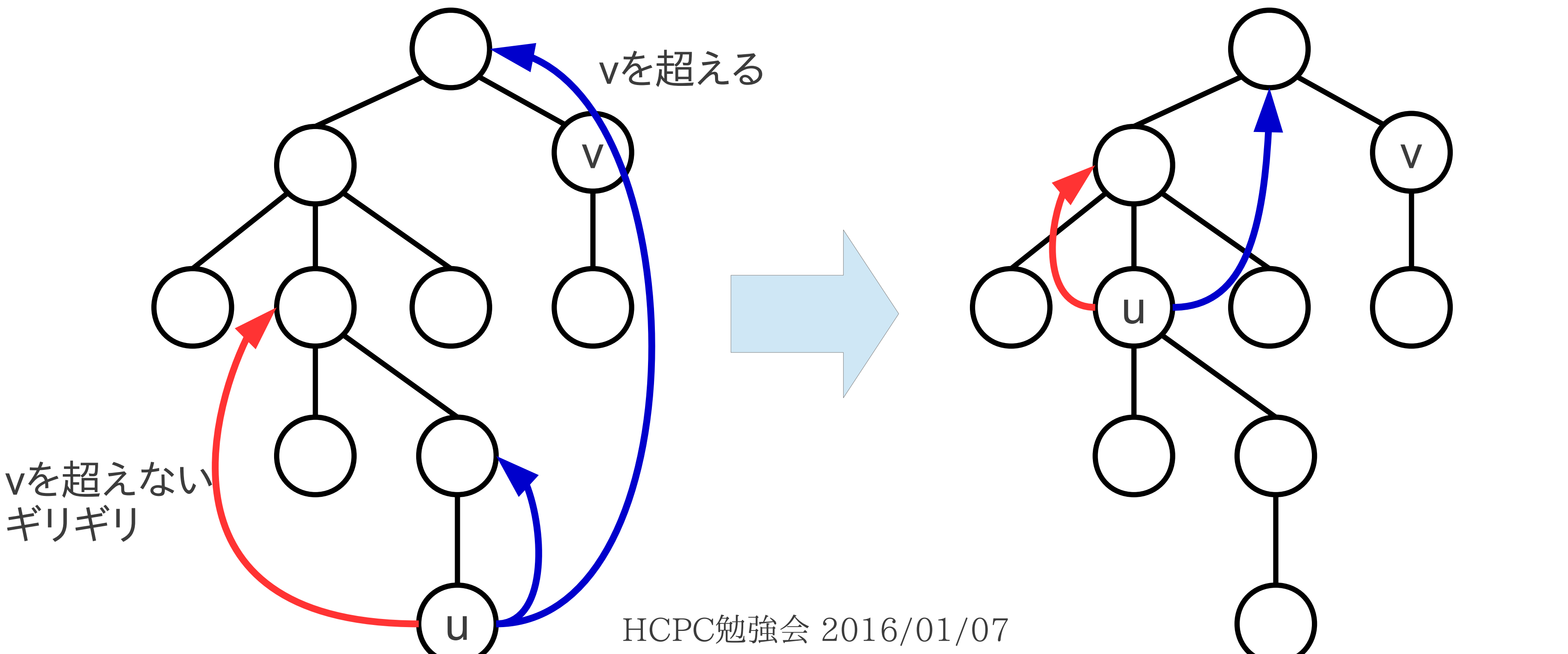
前処理の計算量
DFS-part ：O(N)
DP-part ：O(N log(N))
全体 ：O(N log(N))

クエリ処理

- 2頂点u,vのLCAを二分法により求める
- 一気に登るアルゴリズム
  - uとvの高さが合うように深い方を登らせる
    - これを二分法によりO(log(N))で行う
  - uとvが初めてぶつかる直前まで一気に登る
    - これも二分法によりO(log(N))で行う
  - 全体でO(log(N))で処理

クエリ処理 part-1

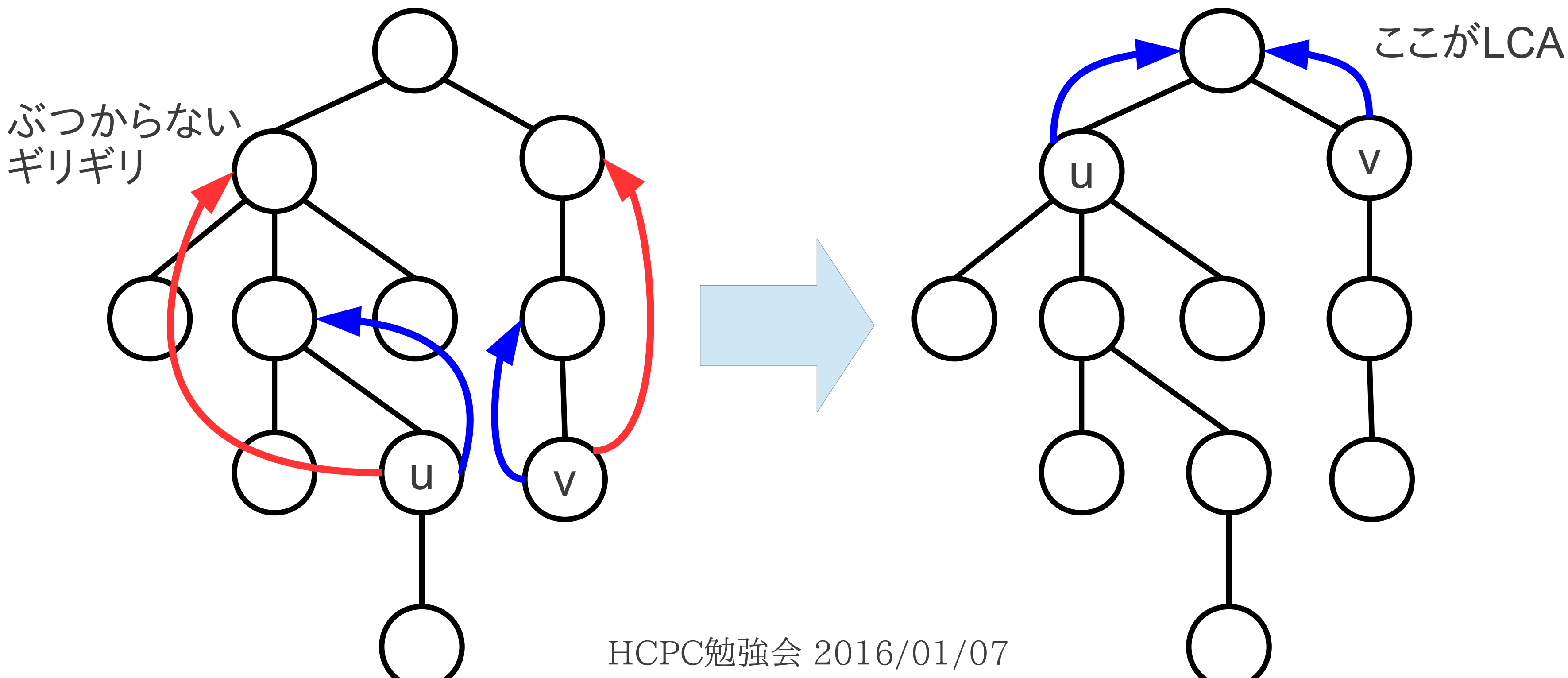
- uが深いとして、vとの高さが合うまでuを登らせる
  - vの高さを超えない範囲で一気に登る
    - 2^k登ってもvの高さを超えないなら u = parent[u][k]
    - 次に登る値の最大値が半分($2^k \rightarrow 2^{k-1}$)になっていく O(log(N))



クエリ処理 part-2

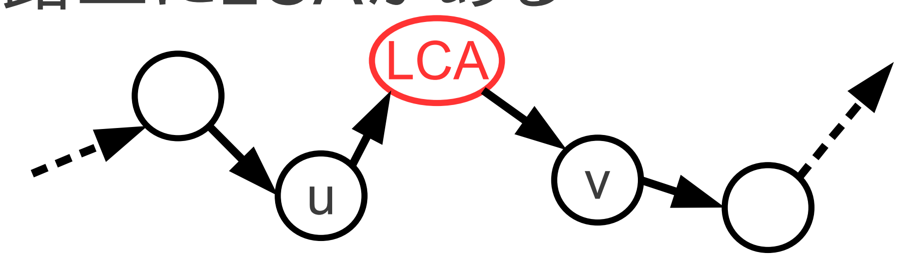
- 2^k個上の先祖情報を用いて一気に登る
  - 2^k個上の先祖が違うならu,v共に2^k個登る
    - If (parent[u][k] != parent[v][k]) then 登る;
    - これもまた登る値の最大値が半分($2^k \rightarrow 2^{k-1}$)になっていく

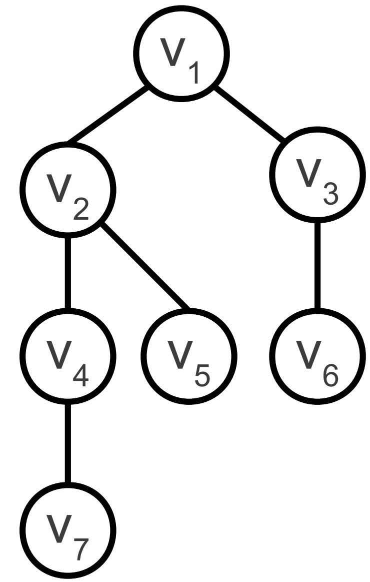
O(log(N))



RMQ(Range Minimum Query)を用いて求める

- DFSの訪問順序を記憶
  - $v_1 \to v_2 \to v_4 \to v_7 \to v_4 \to v_2 \to v_5 \to v_2 \to v_1 \to v_3 \to v_6 \to v_3 \to v_1$
- 2頂点間の訪問経路上にLCAがある
  - v_5とv_7 のLCAは?
    - Answer. v_2





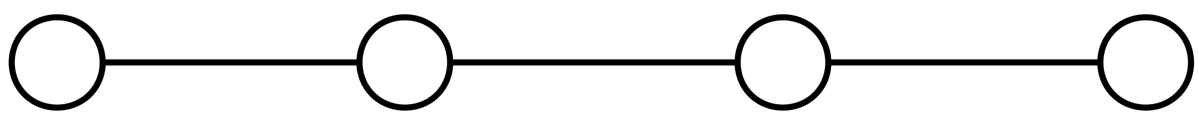
  - $v_1 \to v_2 \to v_4 \to v_7 \to v_4 \to v_2 \to v_5 \to v_2 \to v_1 \to v_3 \to v_6 \to v_3 \to v_1$
- 2頂点間の訪問経路上で最も深さの小さい頂点がLCA
  - 深さに関するRMQでLCAの位置がわかる
  - $v_1 \to v_2 \to v_4 \to v_7 \to v_4 \to v_2 \to v_5 \to v_2 \to v_1 \to v_3 \to v_6 \to v_3 \to v_1$
  - $0 \to 1 \to 2 \to 3 \to 2 \to 1 \to 2 \to 1 \to 0 \to 1 \to 2 \to 1 \to 0$

※行きがけ順のインデクスで考える

SegmentTreeを使う場合
前処理DFS+Seg木構築:O(N)
クエリ処理:O(log(N))

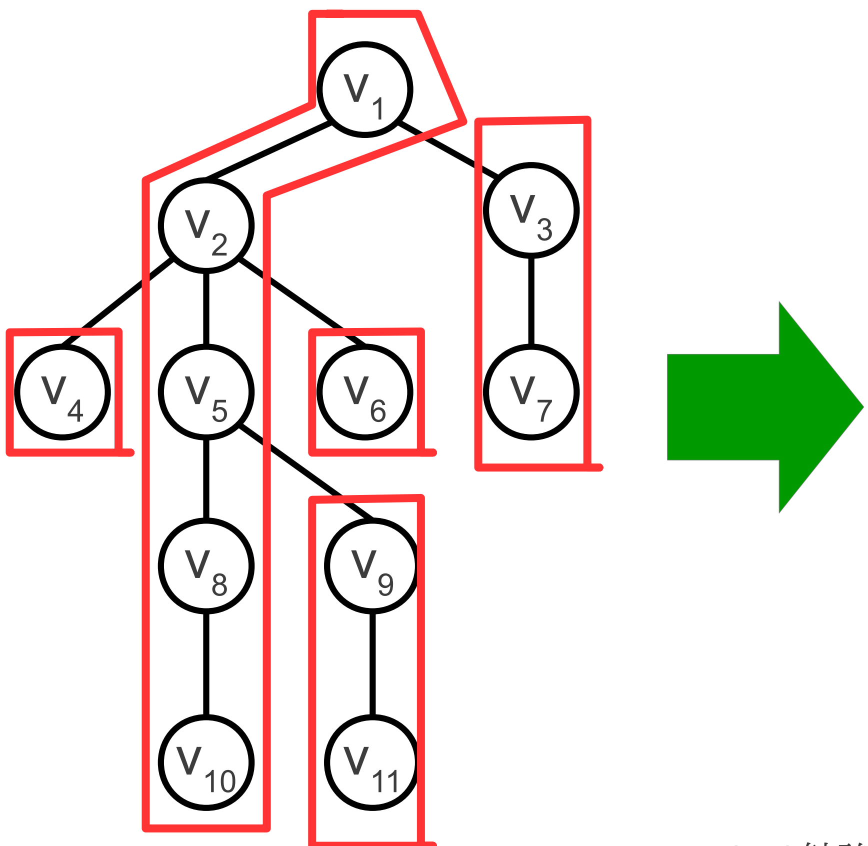
事前知識:パス

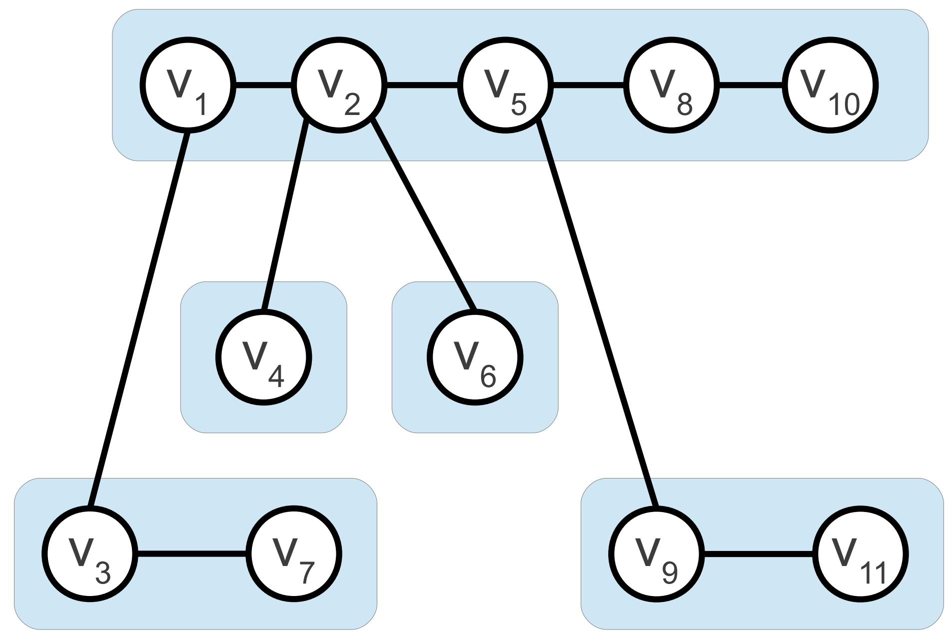
- 一直線に書くことができるグラフ
  - 閉路がない
  - 木の特殊系とも見れる



H/L分解

- Heavy−Light Decompositionのこと
- 根付き木をいくつかのパスに分解する

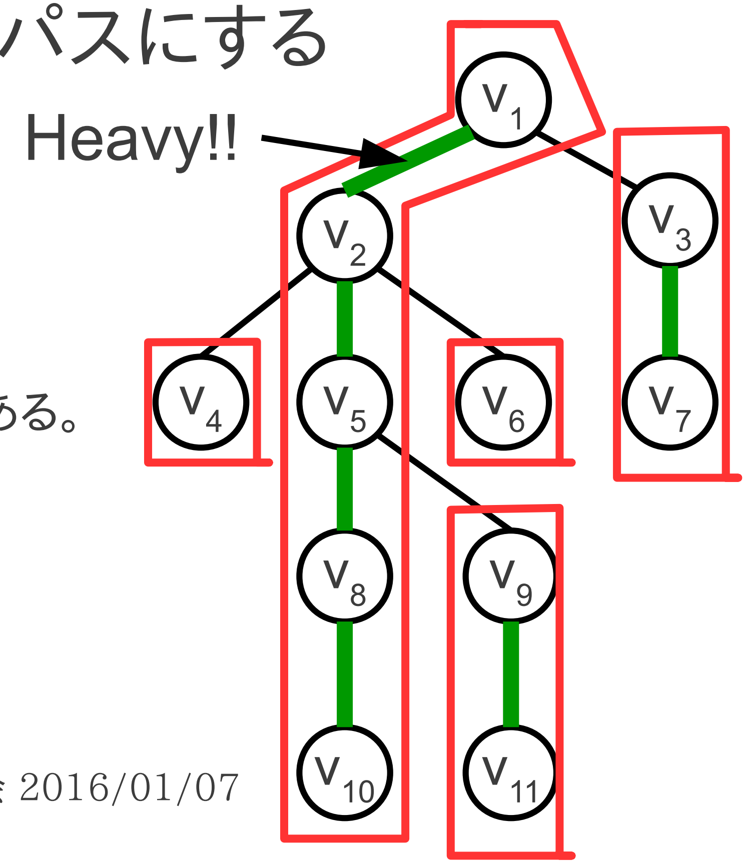




1つのパスを1つのノードに潰すと、
より小さな木として考えることができる！

H/L分解の規則

- 規則
  - 枝をHeavy-EdgeとLight-Edgeに分ける
    - 子孫の半分以上を持つ枝をHeavyで他はLightとするだけ
  - Heavy-Edgeをつないでパスにする



性質がほぼ同じ分解として、
子孫の最も多い枝をHeavyと考える分解もある。
（僕はいつもこっちを使っている）

H/L分解のうまみ

- 新しい木は高さがO(log(N))
  - 潰した木に残っている辺はLight-Edgeのみ
  - Light-Edgeで結ばれた先は |子孫| <= N/2
  - Light-Edgeをたどった先では頂点数が半分になる
- 新しい木の頂点はパスを保持する
  - パスに対する高速なアルゴリズムと組合せられる
  - 1次元配列を用いたデータ構造で色々管理できる

H/L分解のうまい利用法（雑）

- O(log(N))でLCAを求められる
- データ構造との組合せ
  - BIT
  - SegmentTree
    - コイツデナグルルトキノモンダイダイタイトケルツヨイ
  - 他にも色々あるんだろうなぁ。。。
- 結局のところ
  - 根付き木に関する多くのクエリをO(log(N))とか $O(\log^2(N))$とかで解決

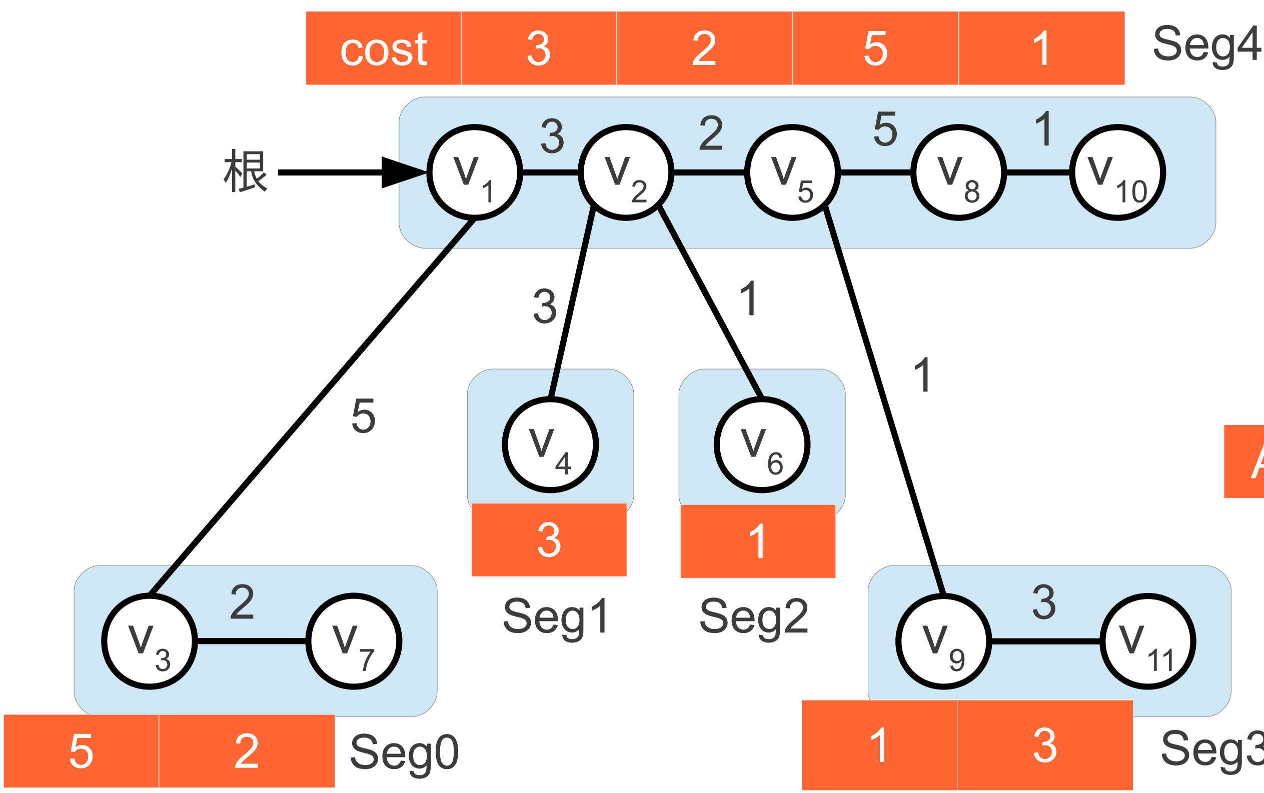
HL分解〜SegmentTreeを添えて〜

- 実際の問題で使い方を雑に説明します
- AOJ 2667 Tree を参照してください
  - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2667

HL分解～SegmentTreeを添えて～

- とりあえずH/L分解してSeg木を付ける

図はいくつかクエリ処理した後だと
思ってくださいオナシャス

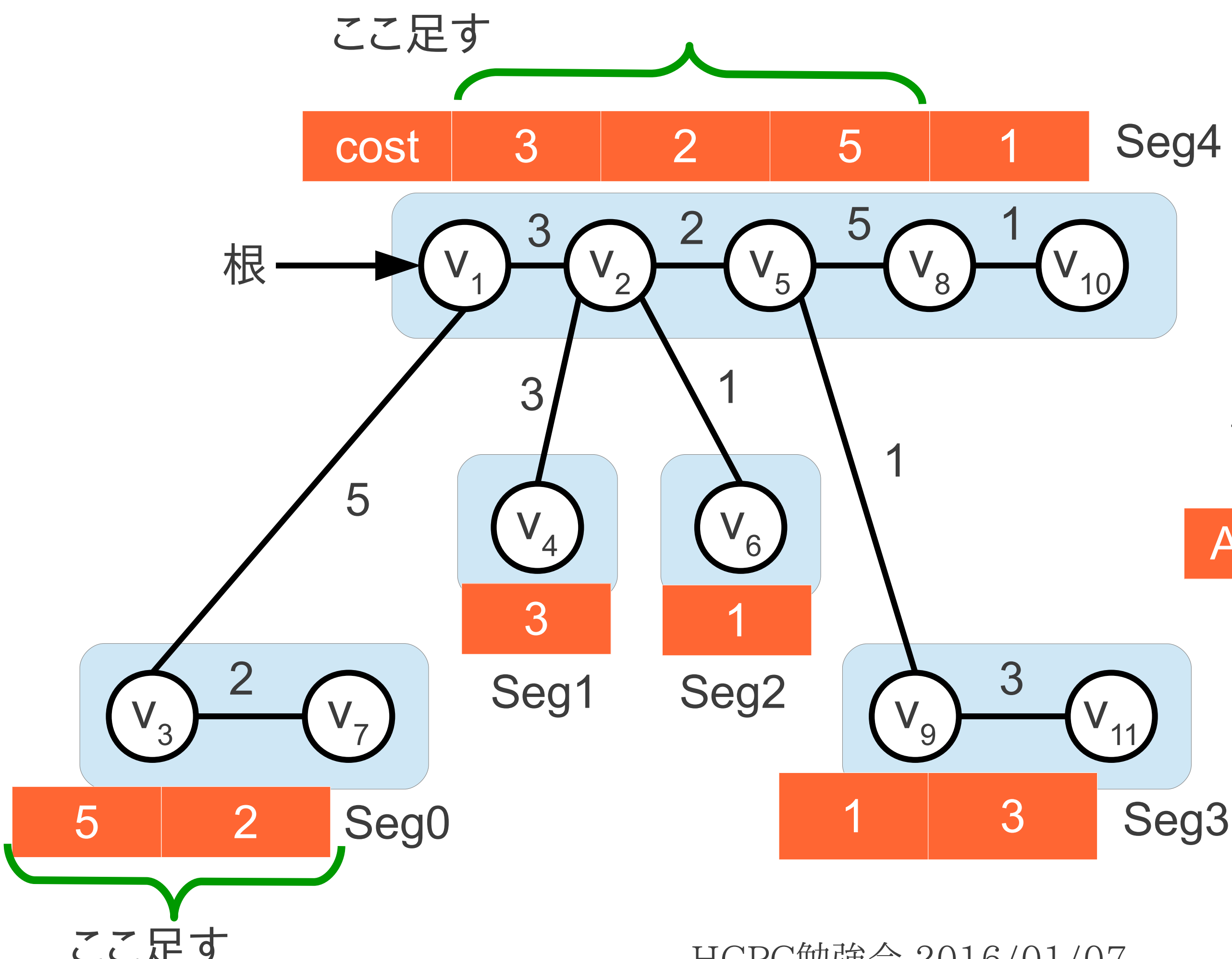


木をDFSするとき、親子関係の
区間表現も作っておくことで、
各Seg木に対する
<u>全範囲への追加コスト</u>を、
別のSeg木で管理する

All	3	2	3	3	2
	Seg0	Seg1	Seg2	Seg3	Seg4

HL分解～SegmentTreeを添えて～

v_5からv_7へのcostは？



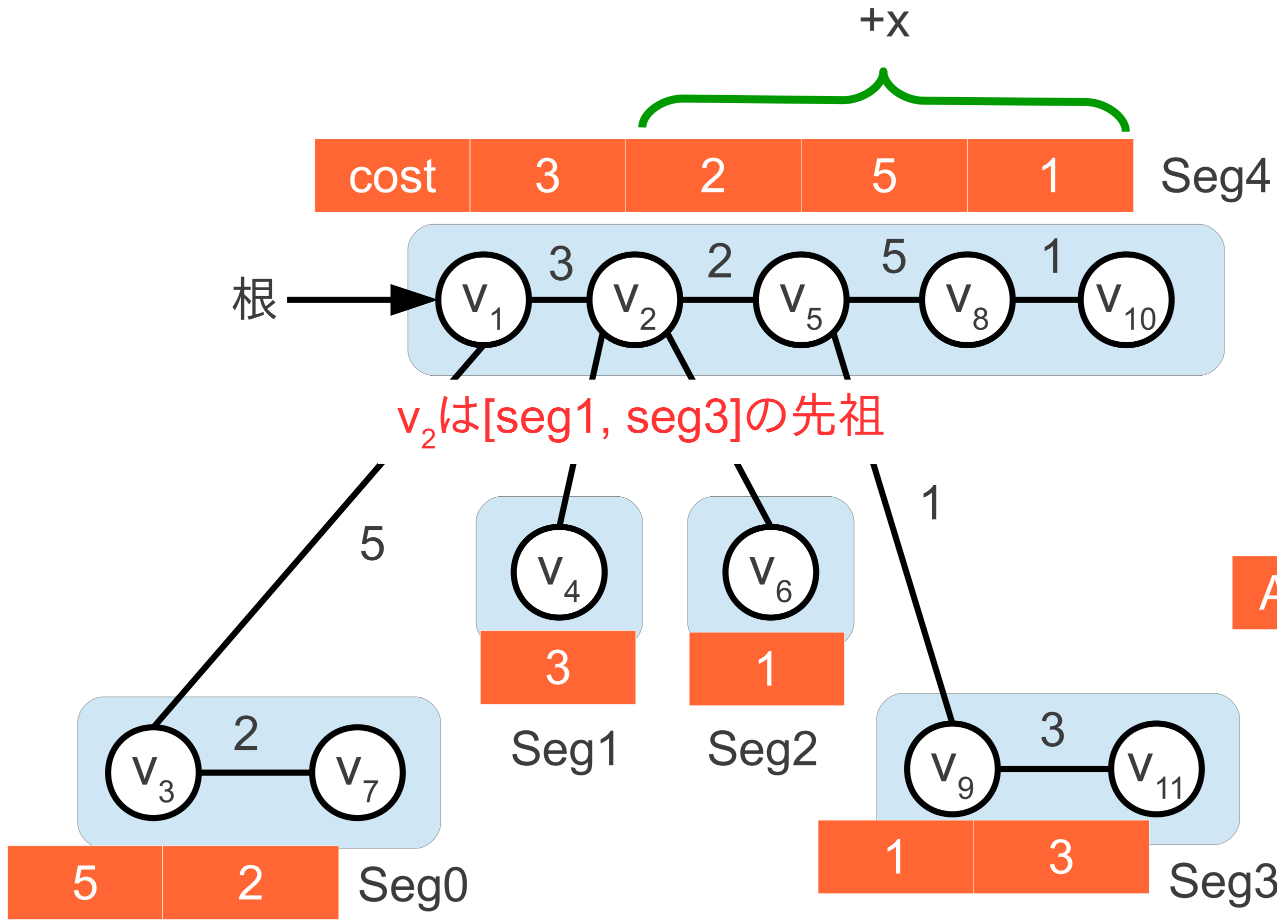
木をDFSするとき、親子関係の
区間表現も作っておくことで、
各Seg木に対する
<u>全範囲への追加コスト</u>を、
別のSeg木で管理する

高さがlog(N)なので
$O(\log^2(N))$

HL分解～SegmentTreeを添えて～

v_2より下の辺のcostを +x

※範囲更新可能な
SegmentTreeでやるべき



木をDFSするとき、親子関係の
区間表現も作っておくことで、
各Seg木に対する
<u>全範囲への追加コスト</u>を、
別のSeg木で管理する

All	3	2	3	3	2
	Seg0	Seg1	Seg2	Seg3	Seg4

+x

O(log(N))で更新できた！

本スライドはここまで

蟻本とかWeb検索の方が圧倒的情報量