Spaceships 解説

Masaki Hara (qnighy)

2013年 情報オリンピック春期トレーニング合宿にて

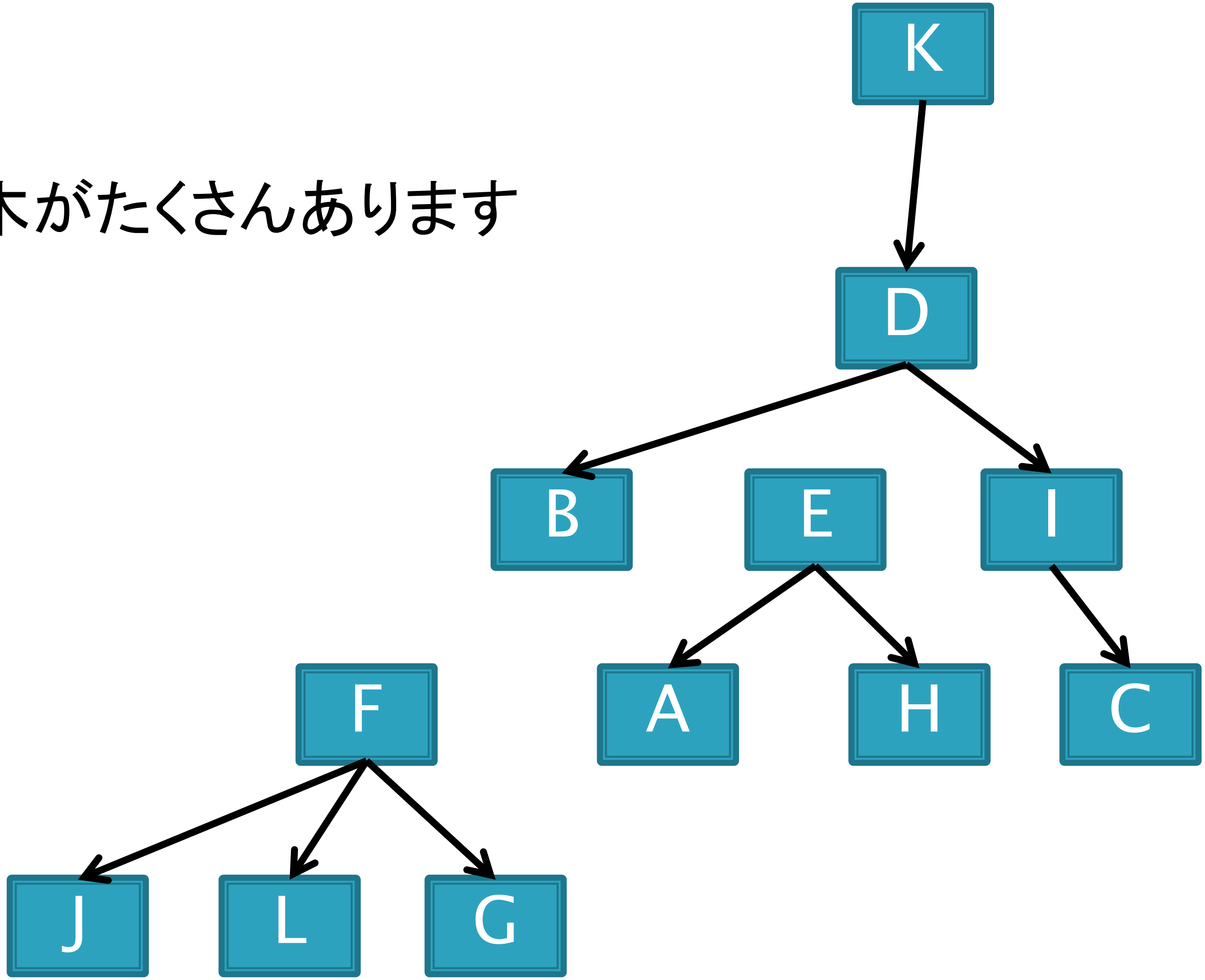
Task statement

- 有向木がたくさんあります
- 以下のクエリに高速で答えてください

1. Aの親をBにする
   - Aは木の根で、BはAの子孫ではない
2. Aを親から切り離して木の根にする
3. A,Bが同じ木に属するか判定し、
   同じ木に属する場合はLCAを求める

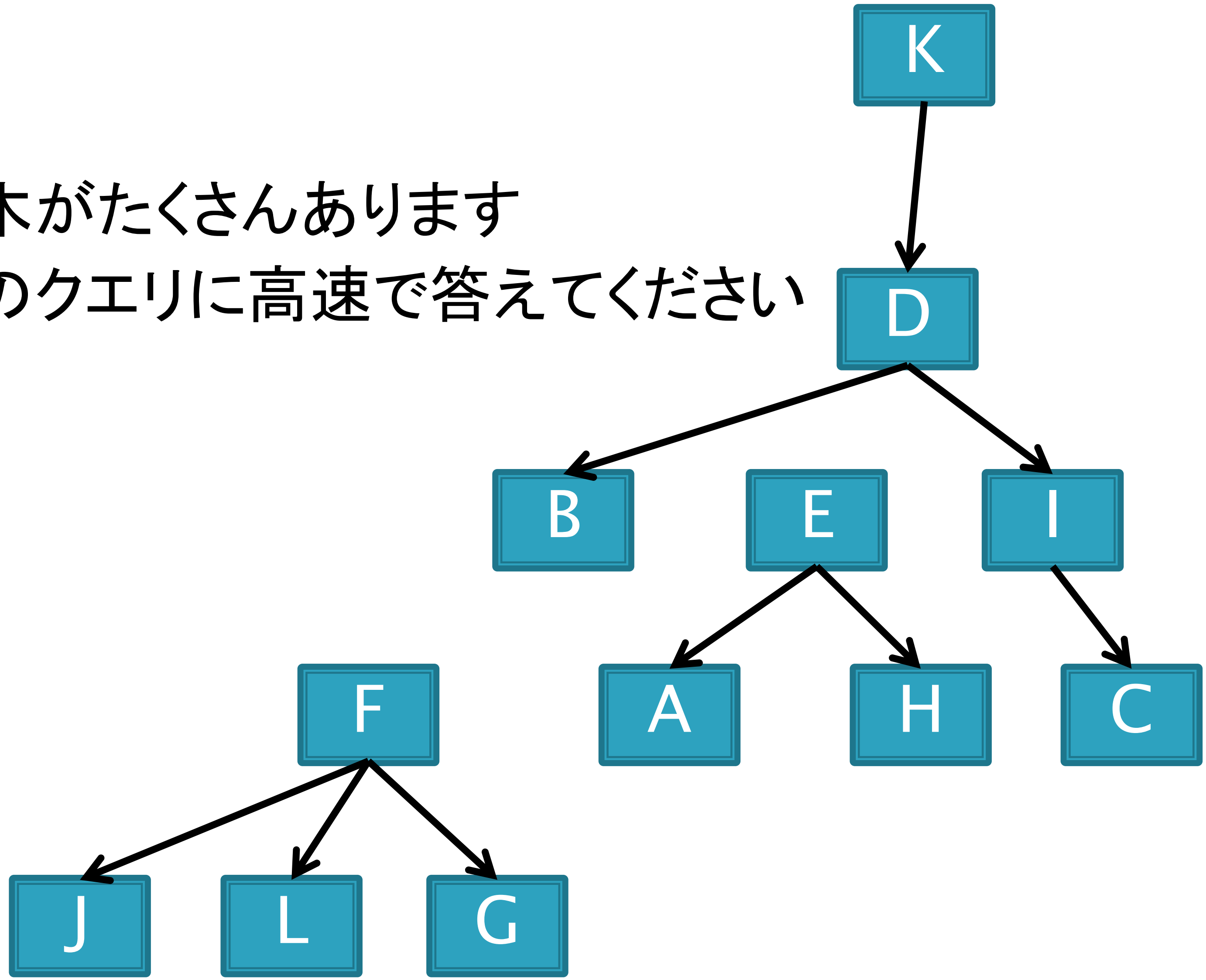
例

- 有向木がたくさんあります



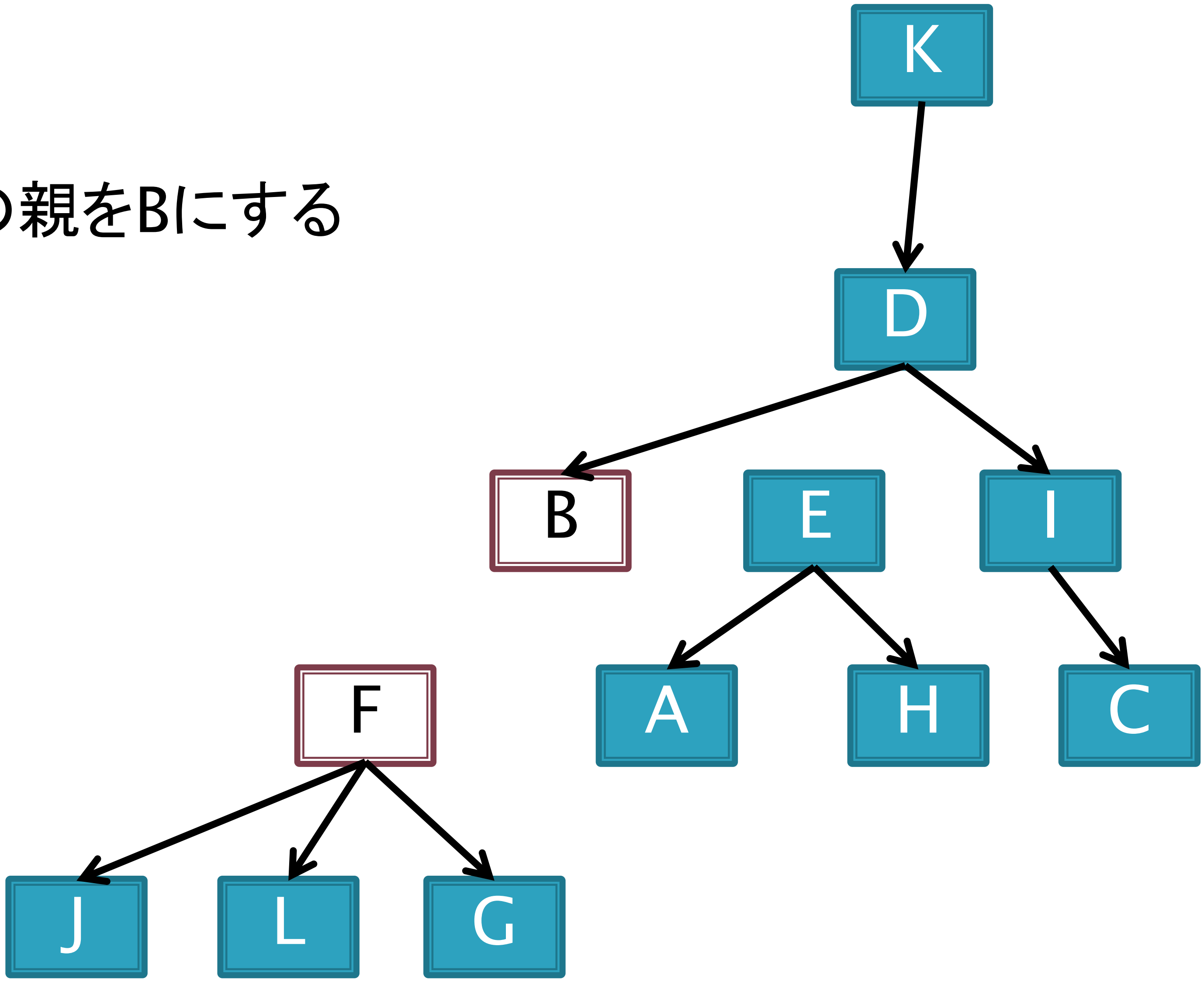
例

- 有向木がたくさんあります
- 以下のクエリに高速で答えてください



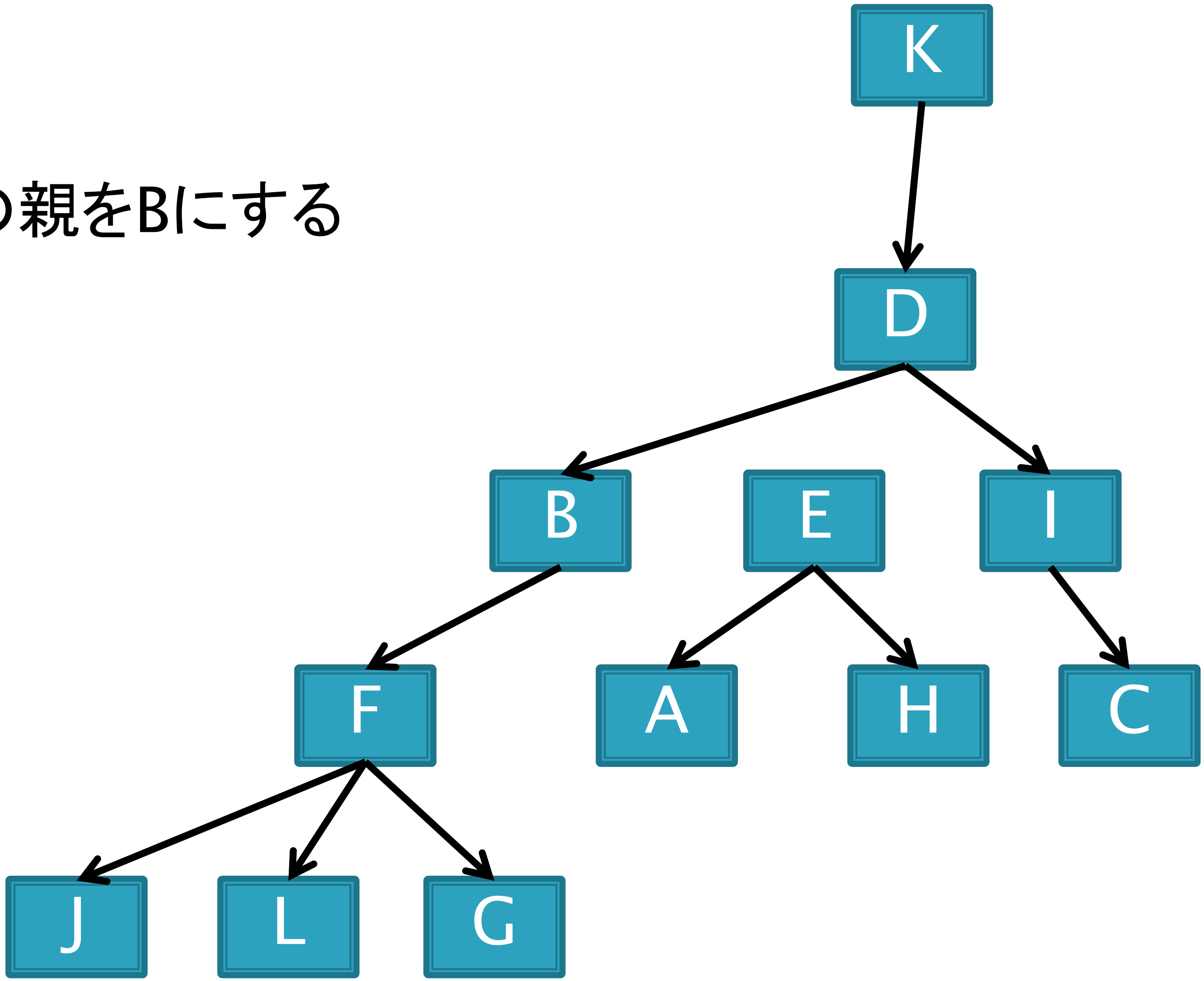
例

- 1. Fの親をBにする



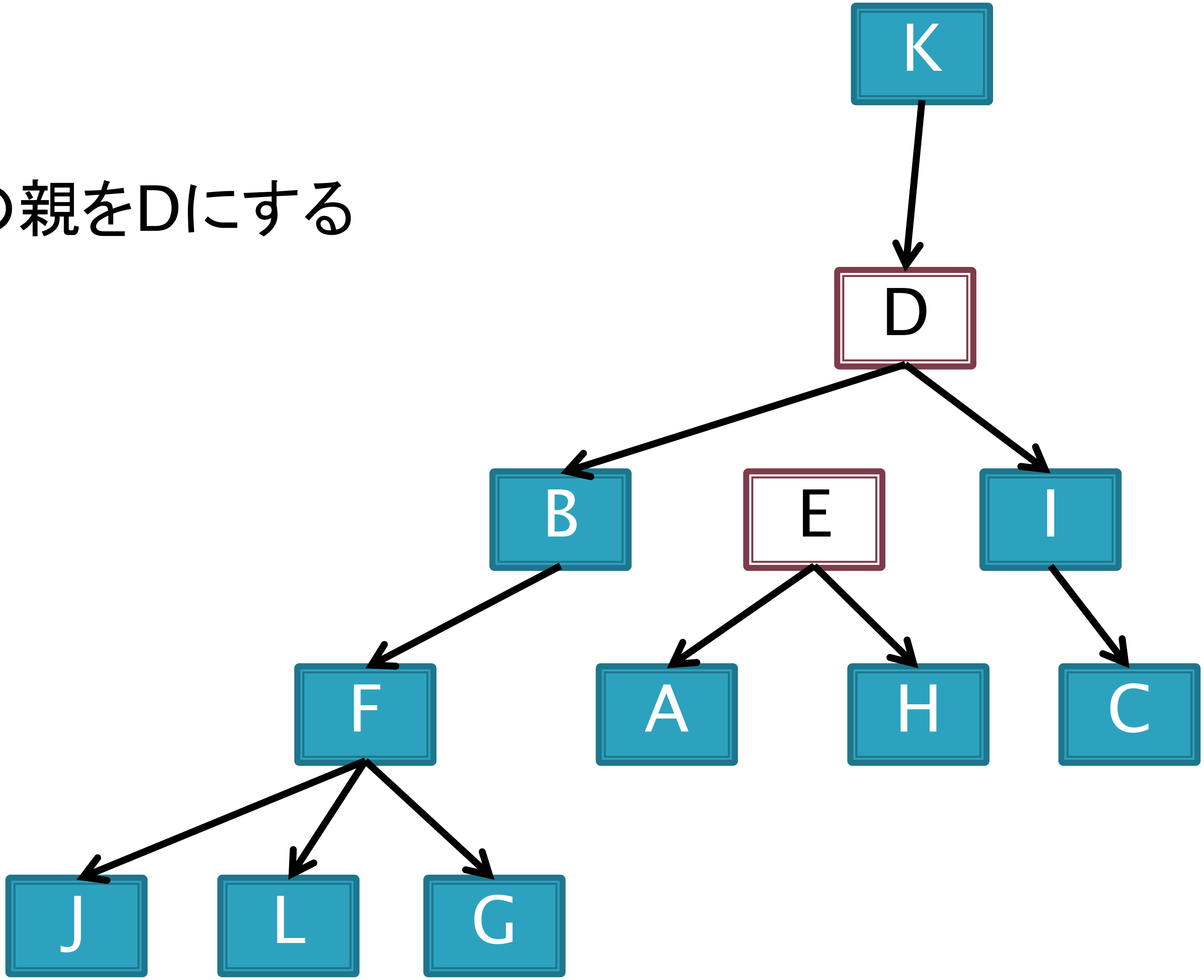
例

- 1. Fの親をBにする



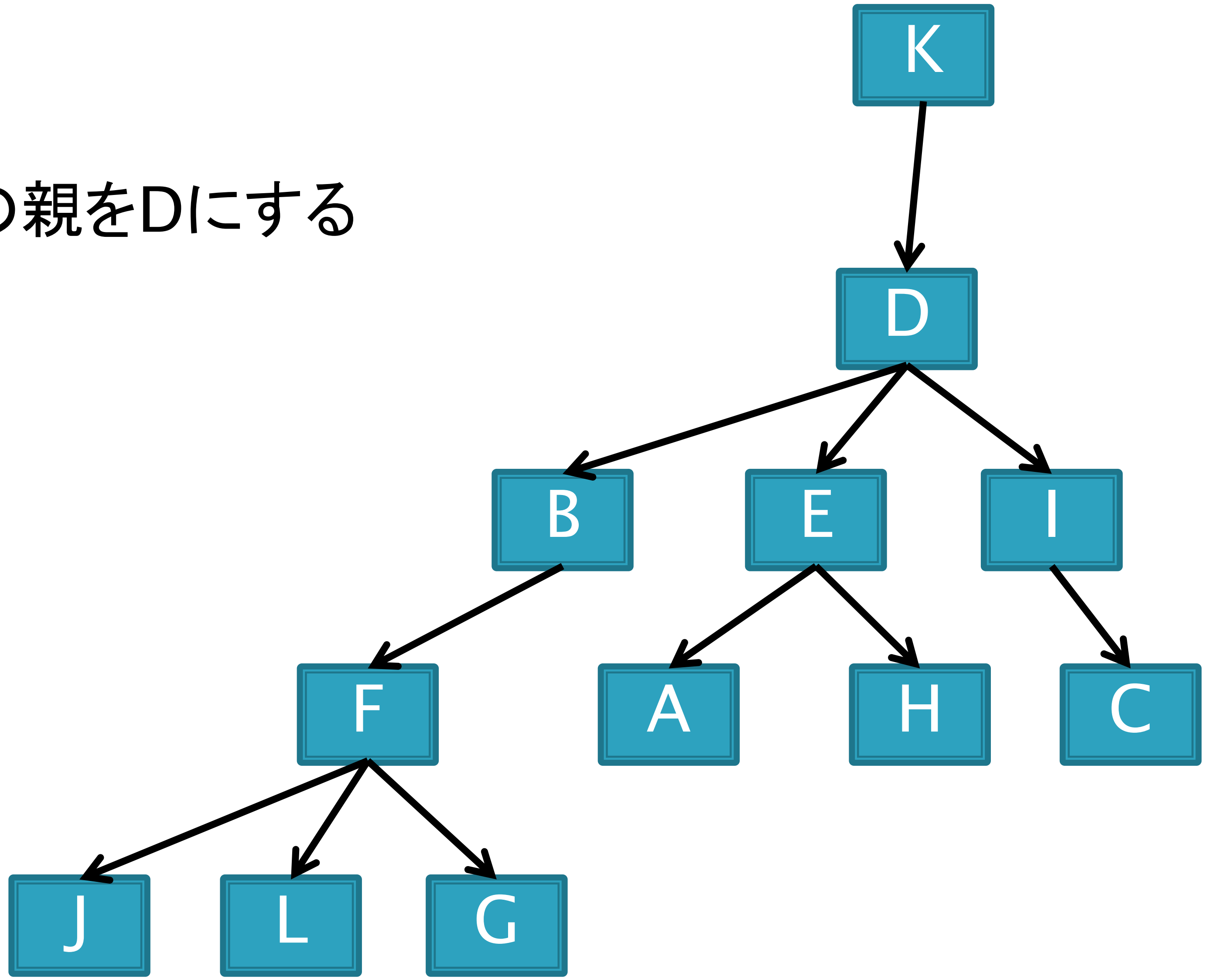
例

- 1. Eの親をDにする



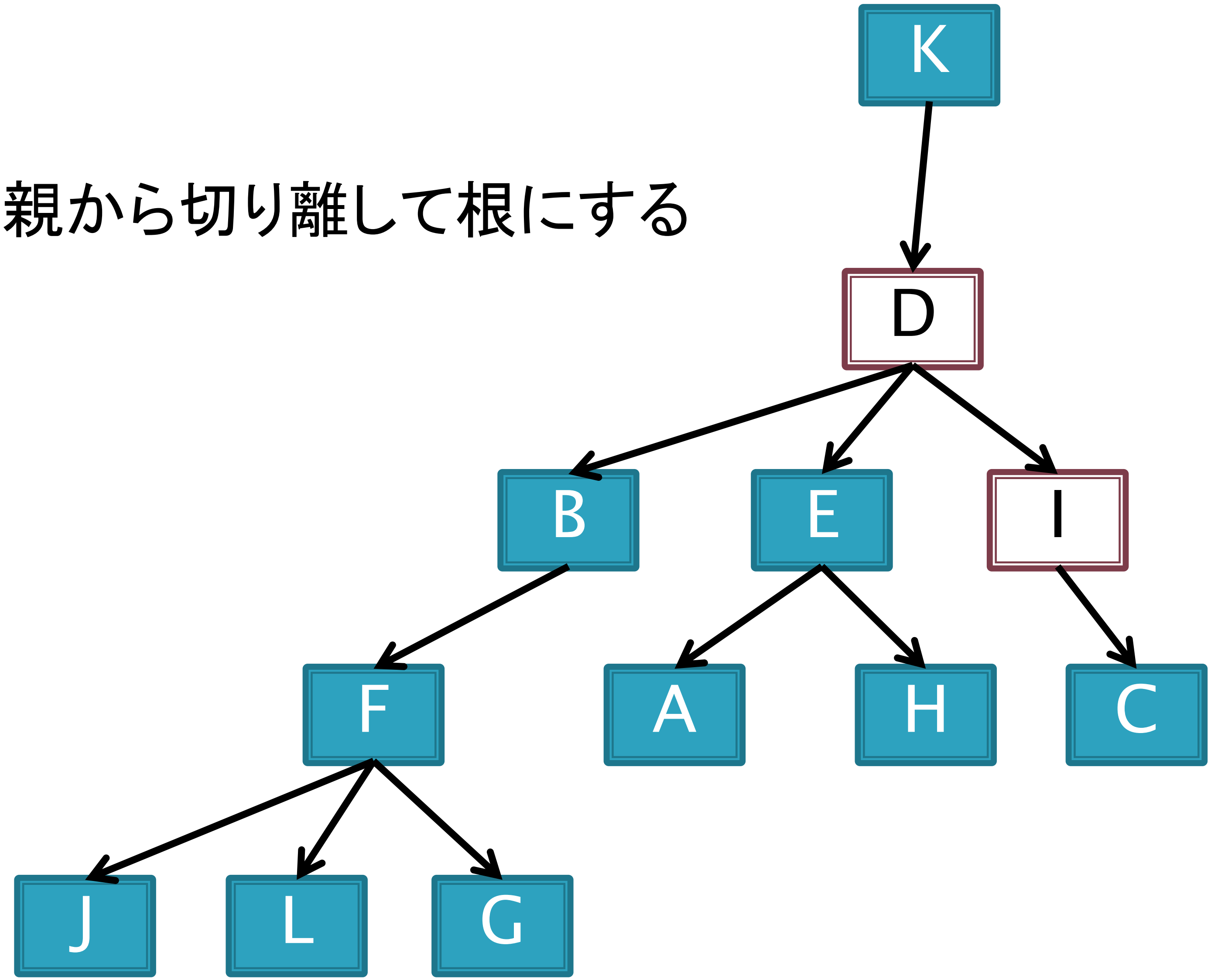
例

- 1. Eの親をDにする



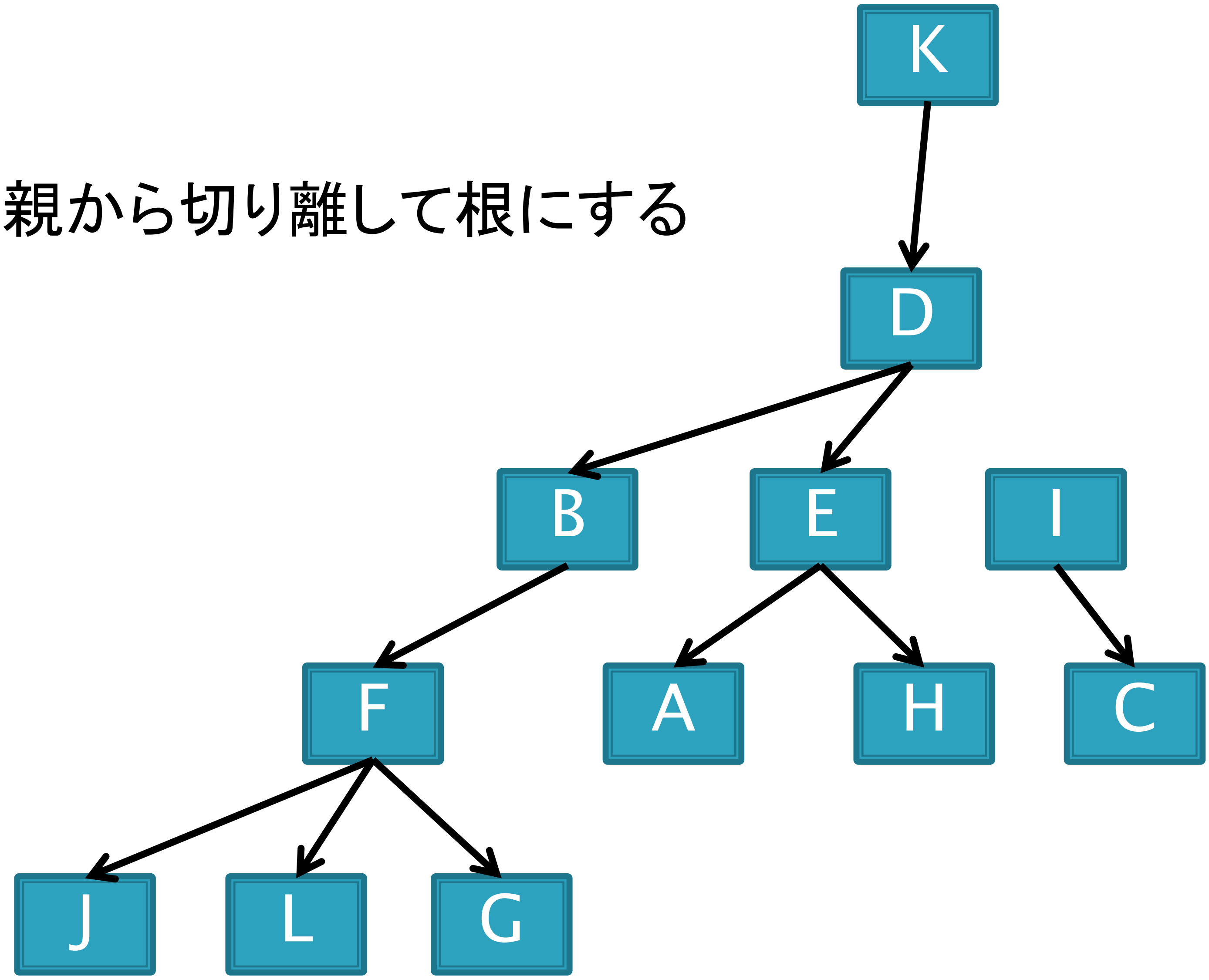
例

- 2. Iを親から切り離して根にする



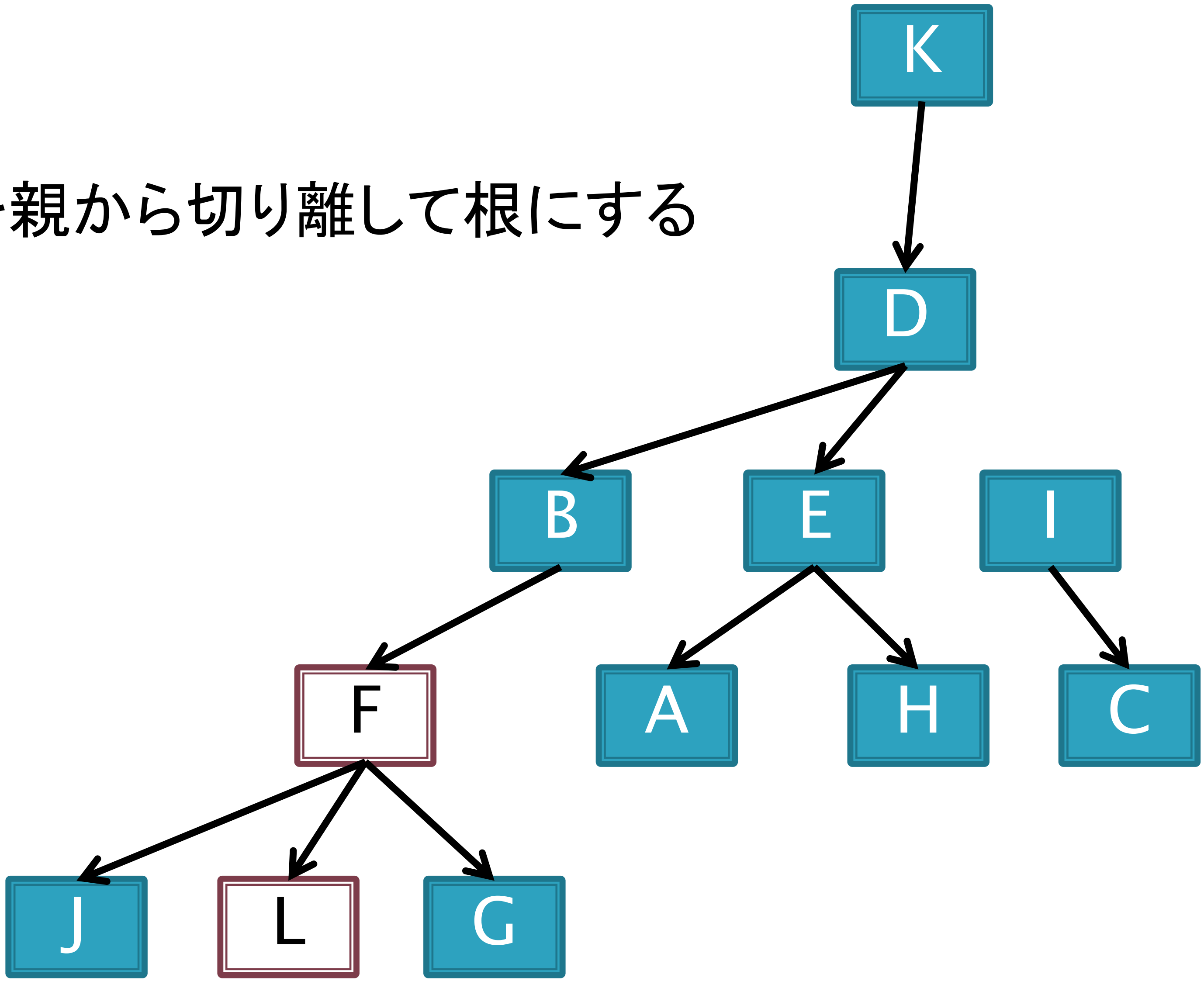
例

- 2. Iを親から切り離して根にする



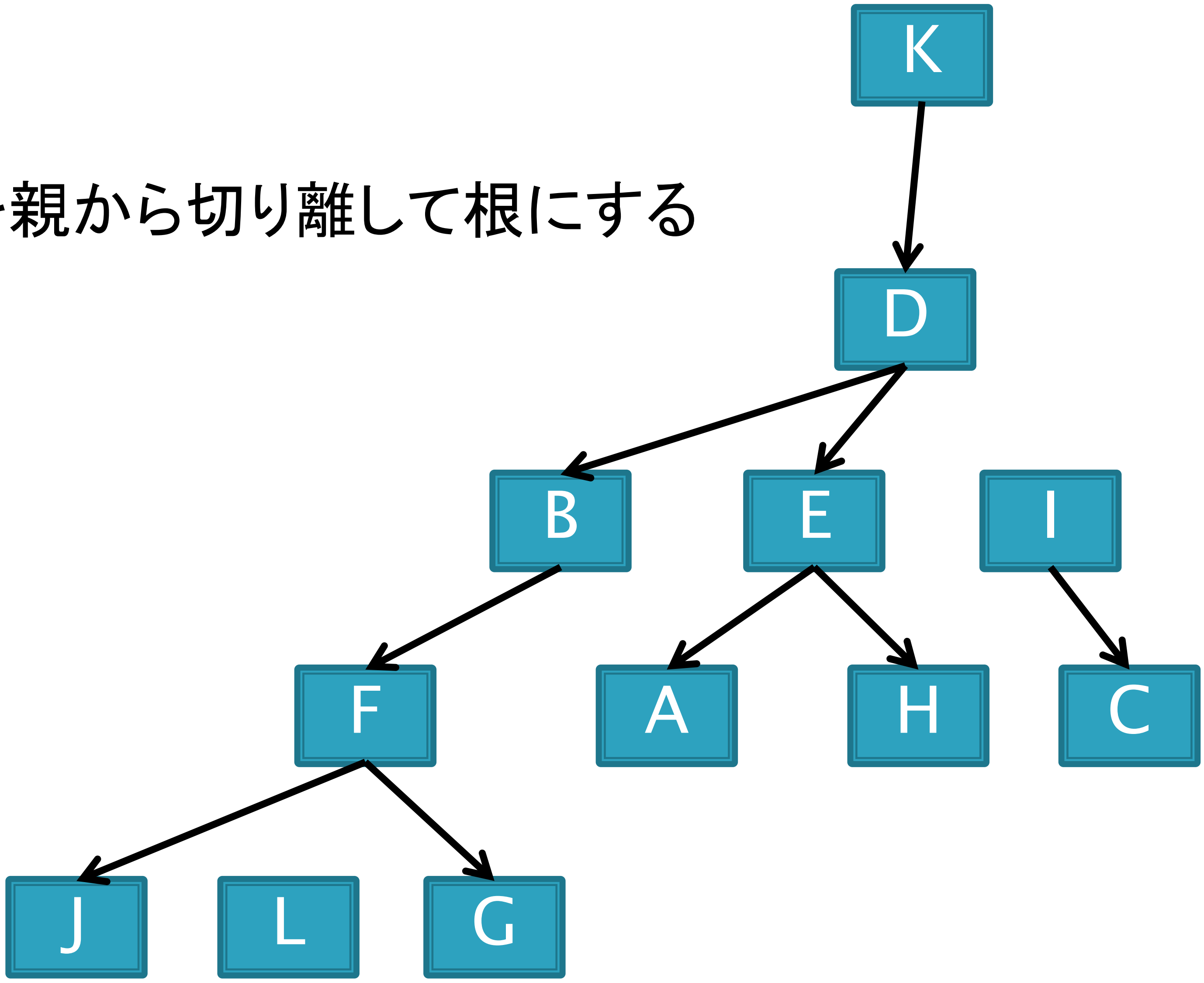
例

- 2. Lを親から切り離して根にする



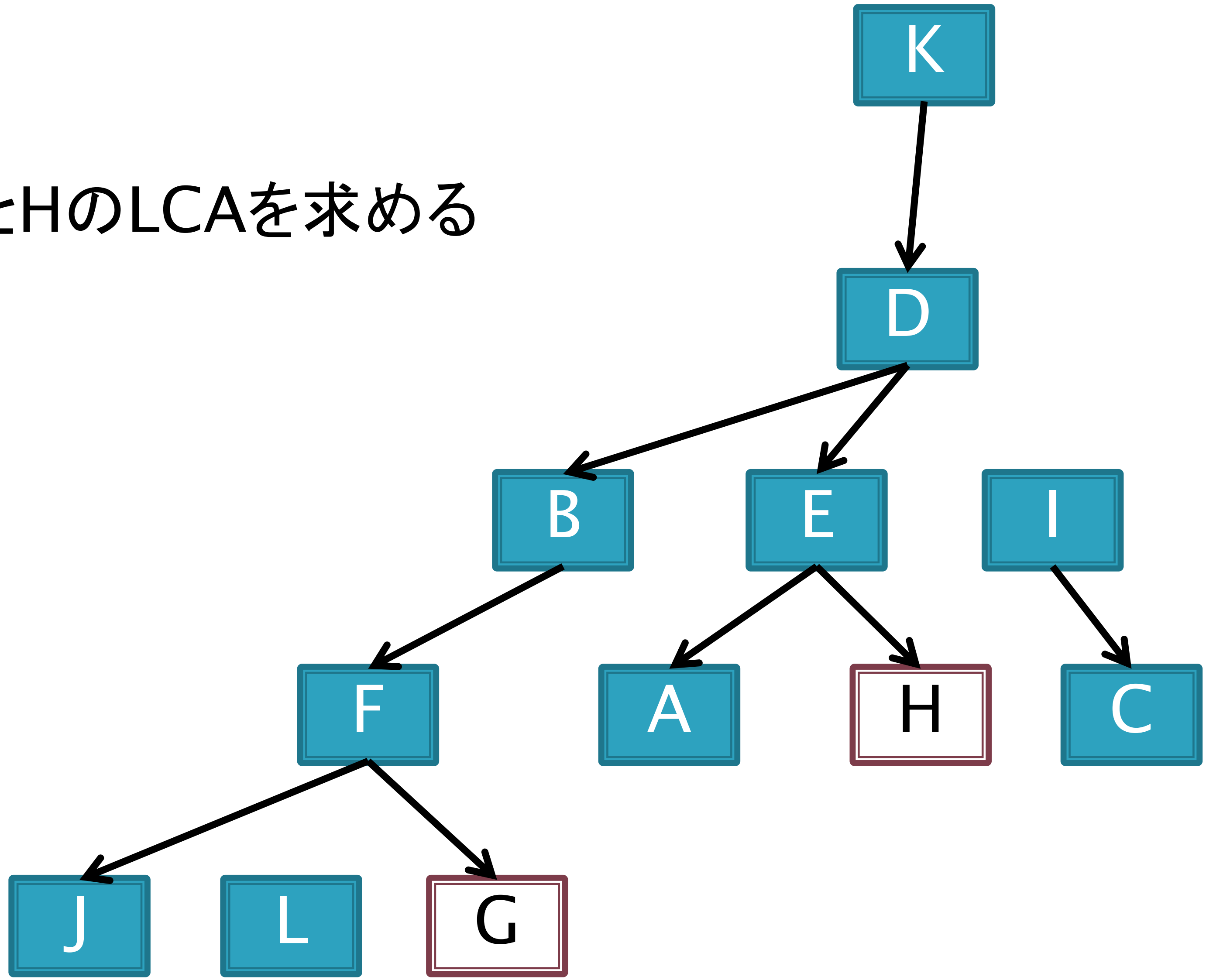
例

- 2. Lを親から切り離して根にする



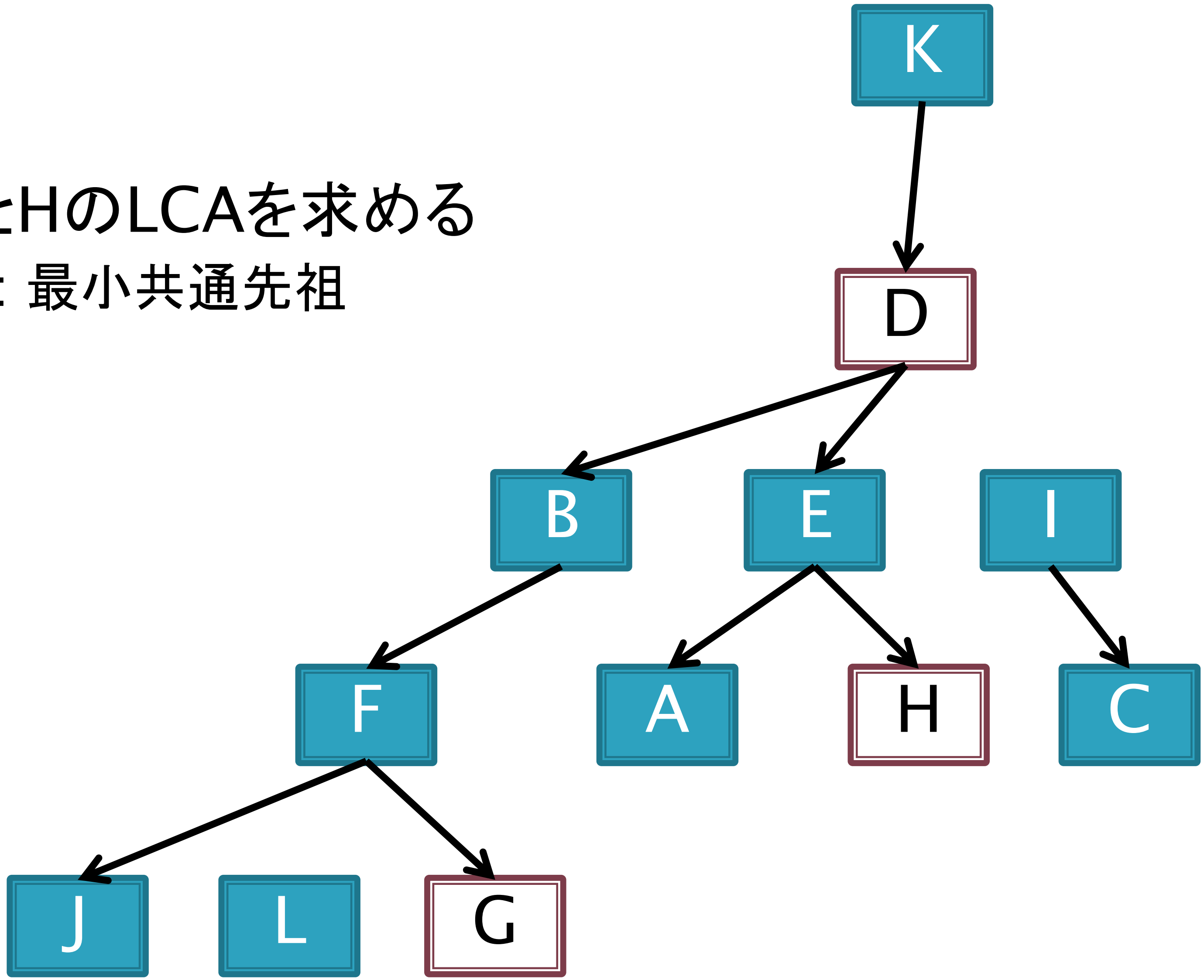
例

- 3. GとHのLCAを求める



例

- 3. GとHのLCAを求める
  - LCA: 最小共通先祖

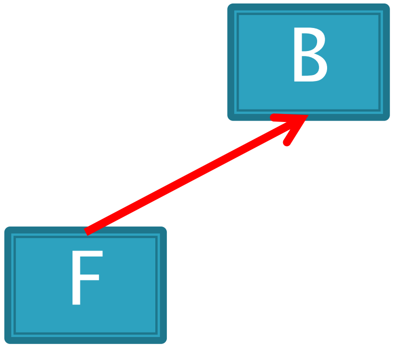


小課題1 (10点)

- 頂点数 $N \leq 5000$
- クエリ数 $Q \leq 5000$

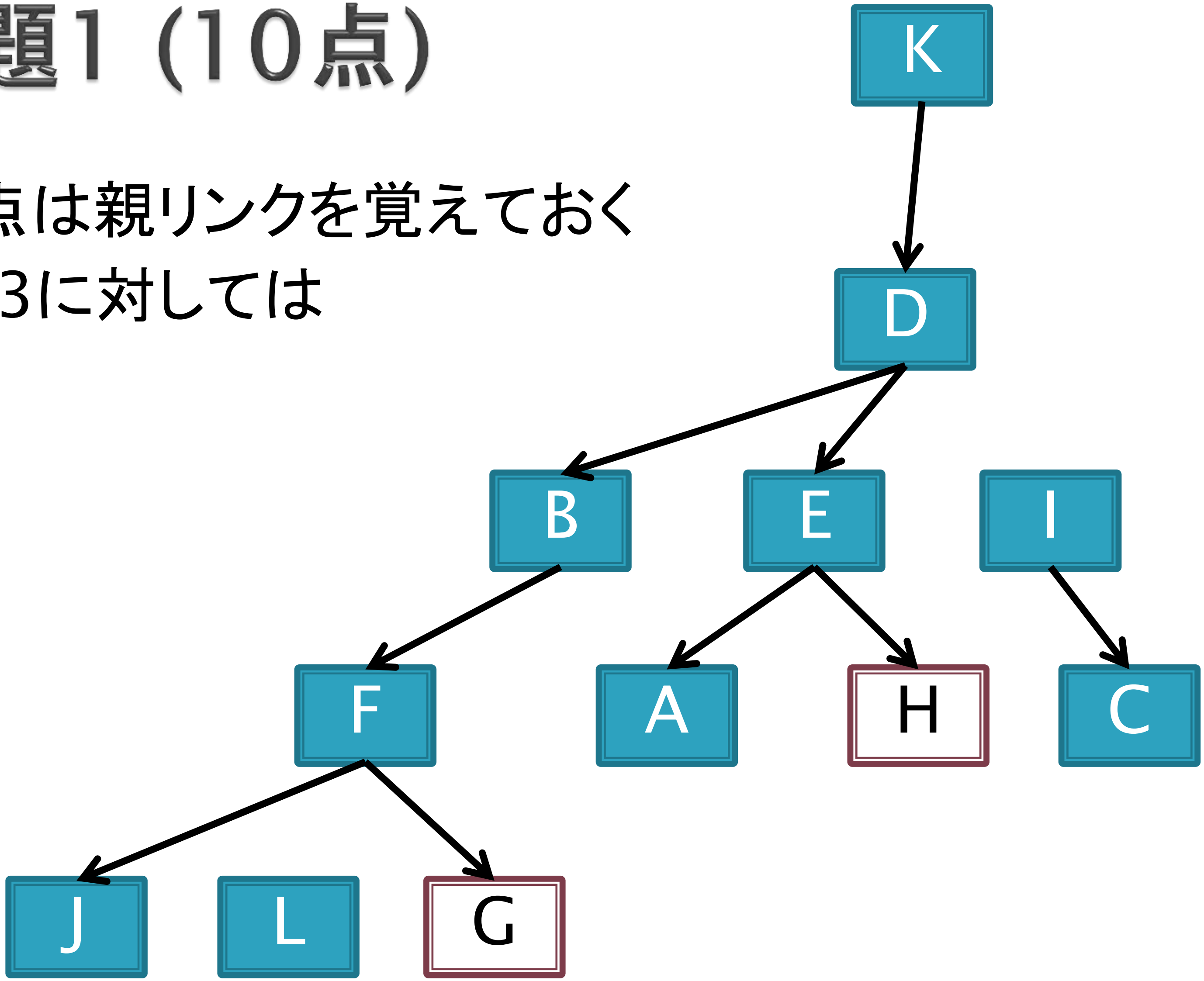
小課題1 (10点)

- 各頂点は親リンクを覚えておく
- クエリ1,2に対しては普通に答える



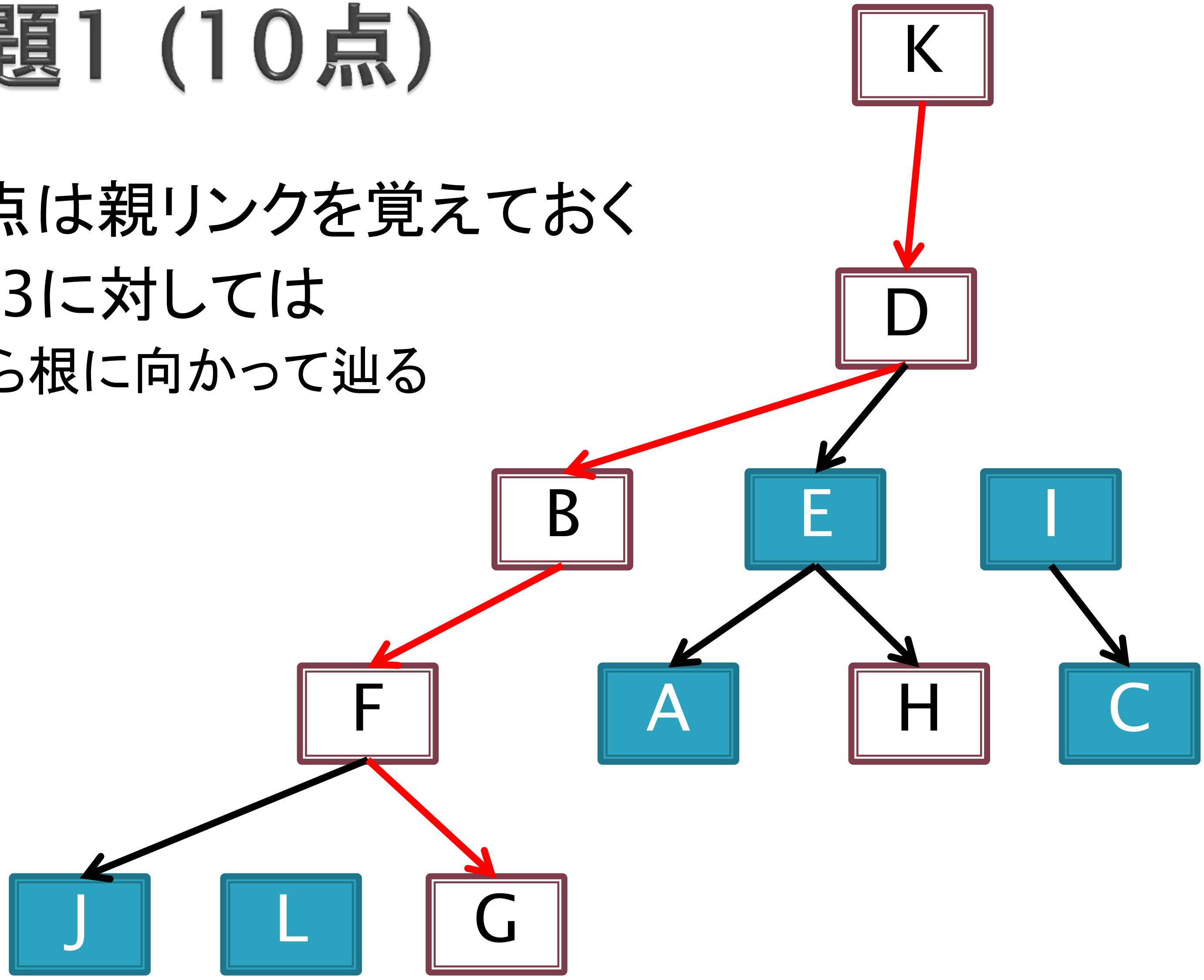
小課題1 (10点)

- 各頂点は親リンクを覚えておく
- クエリ3に対しては



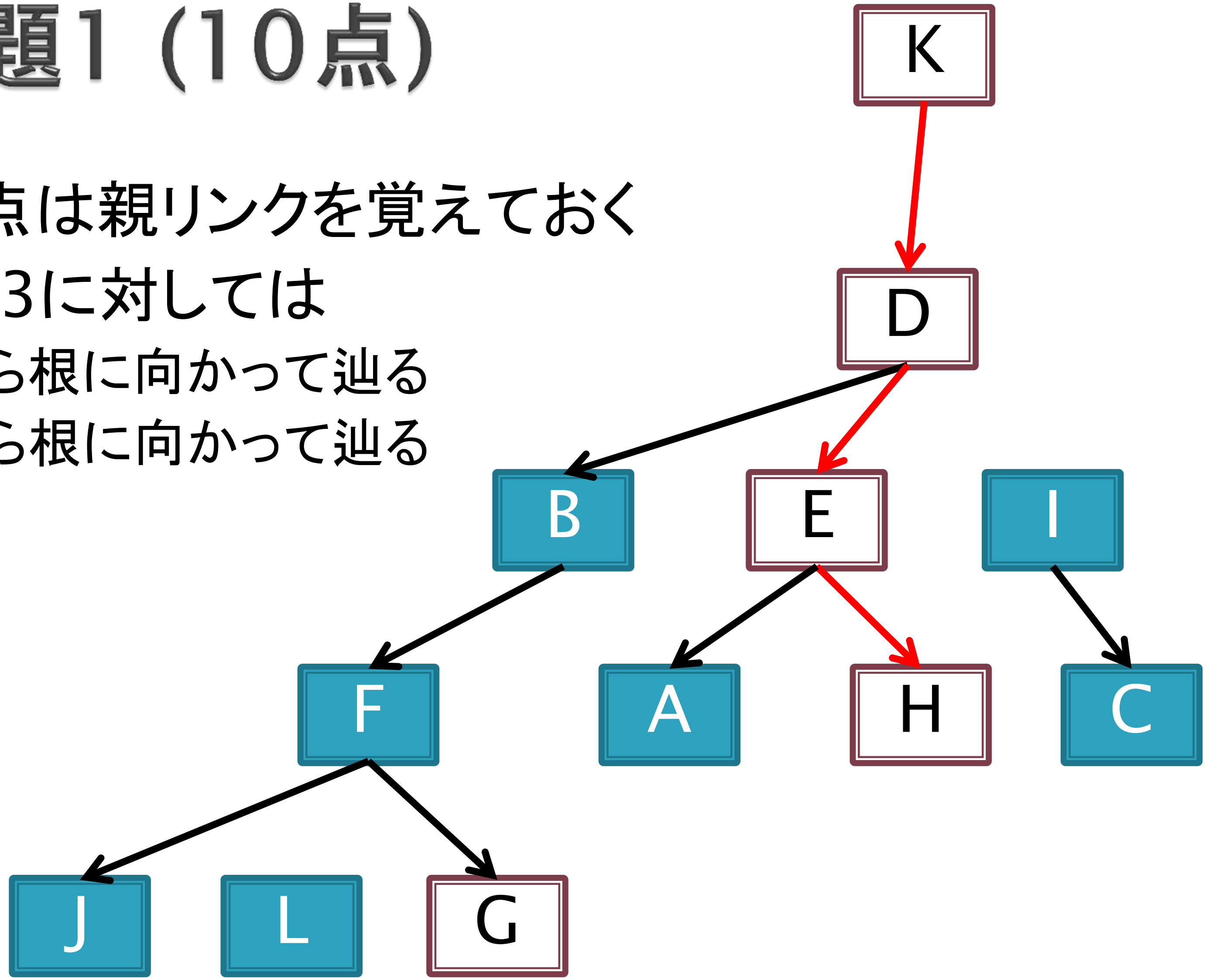
小課題1 (10点)

- 各頂点は親リンクを覚えておく
- クエリ3に対しては
  - Gから根に向かって辿る



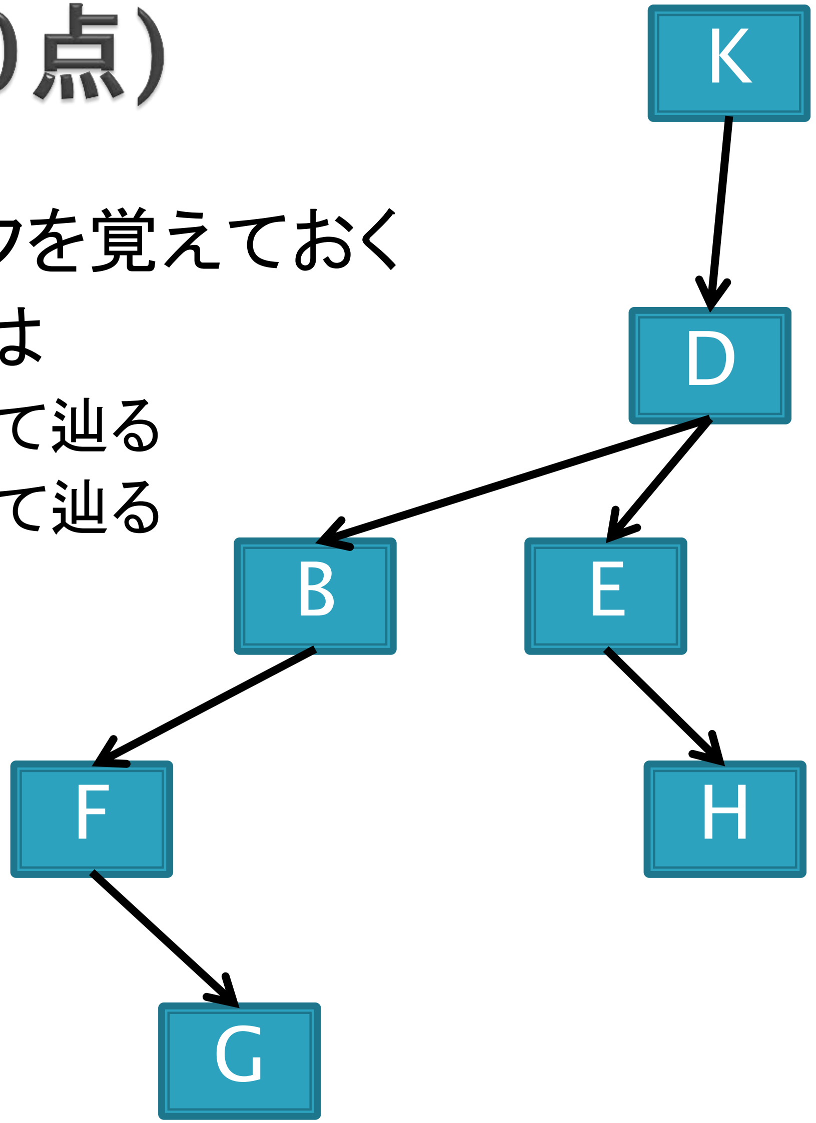
小課題1 (10点)

- 各頂点は親リンクを覚えておく
- クエリ3に対しては
  - Gから根に向かって辿る
  - Hから根に向かって辿る



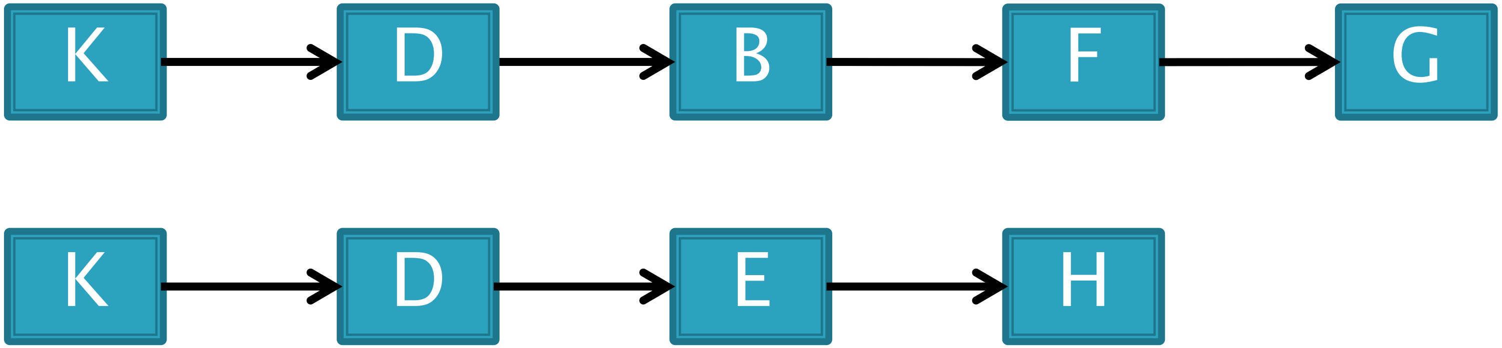
小課題1 (10点)

- 各頂点は親リンクを覚えておく
- クエリ3に対しては
  - Gから根に向かって辿る
  - Hから根に向かって辿る
  - 並べる



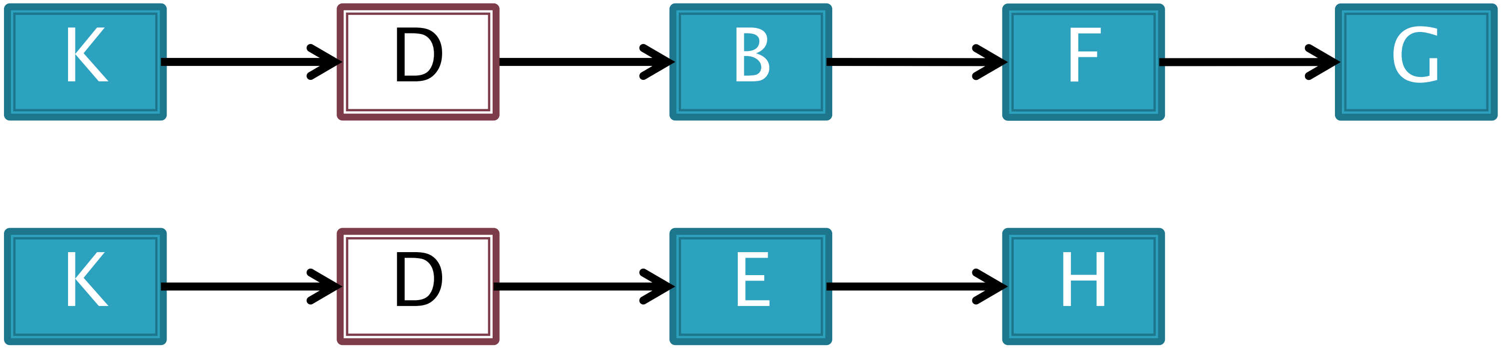
小課題1 (10点)

- 各頂点は親リンクを覚えておく
- クエリ3に対しては
  - Gから根に向かって辿る
  - Hから根に向かって辿る
  - 並べる



小課題1 (10点)

- 各頂点は親リンクを覚えておく
- クエリ3に対しては
  - Gから根に向かって辿る
  - Hから根に向かって辿る
  - 根からの順番で並べる
  - 一致する中で最も後ろのものを選ぶ



小課題1 (10点)

- 各頂点は親リンクを覚えておく
- クエリ3に対しては
  - Gから根に向かって辿る
  - Hから根に向かって辿る
  - 根からの順番で並べる
  - 根が一致しないときは–1

小課題1 (10点)

- 頂点数 $N \leq 5000$
- クエリ数 $Q \leq 5000$

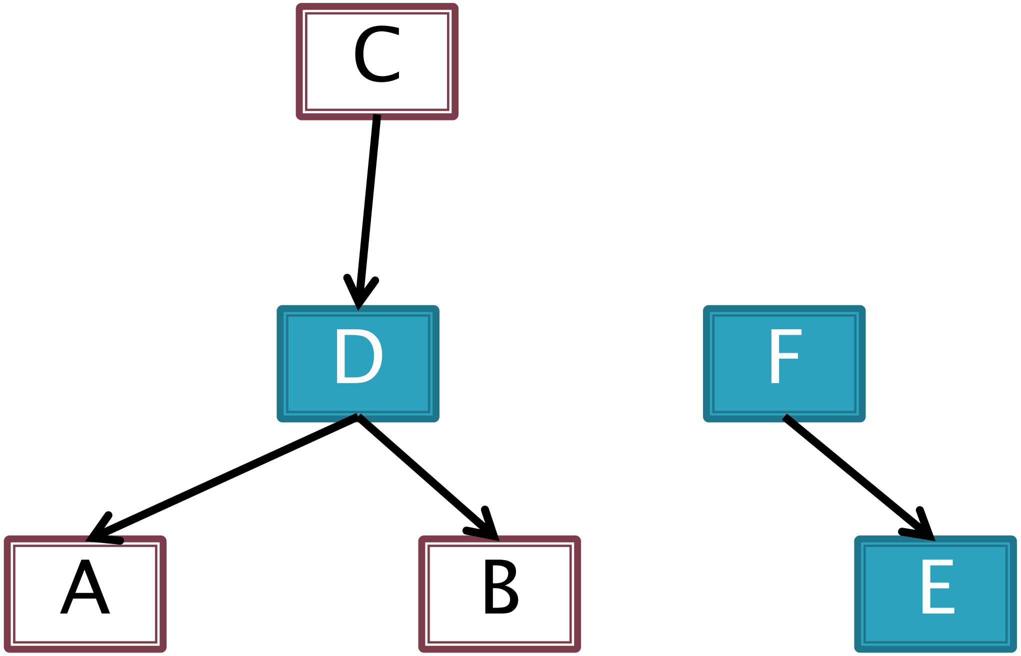
- 計算量は$O(NQ)$なので間に合う

小課題2 (30点)

- 頂点数 $N \leq 10^6$
- クエリ数 $Q \leq 10^6$
- 辺の削除は行われない

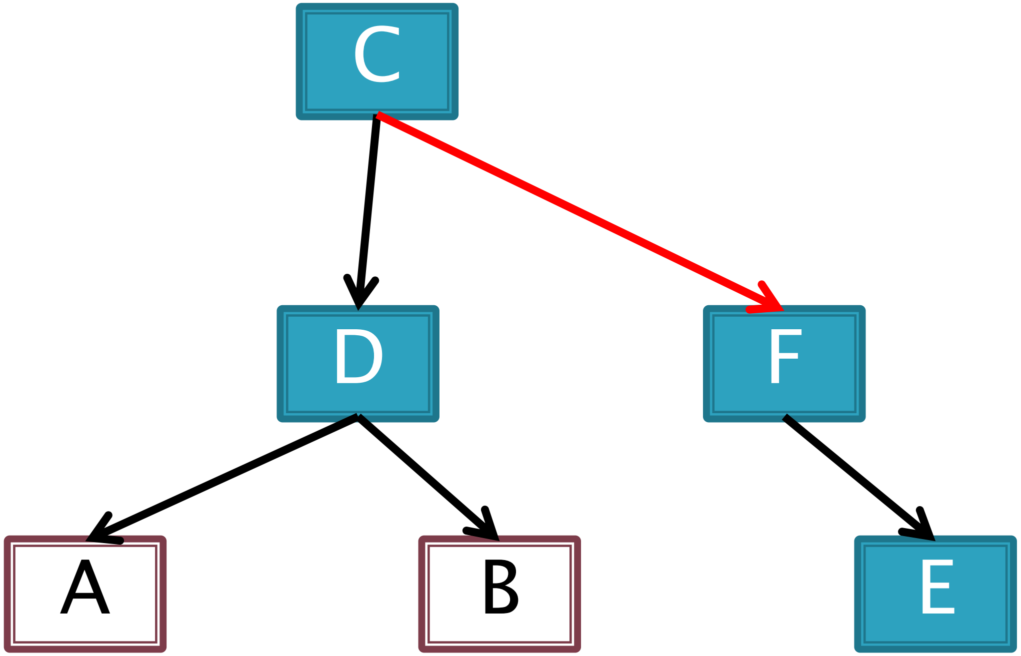
小課題2 (30点) – LCAならLCA

- CがAとBのLCA



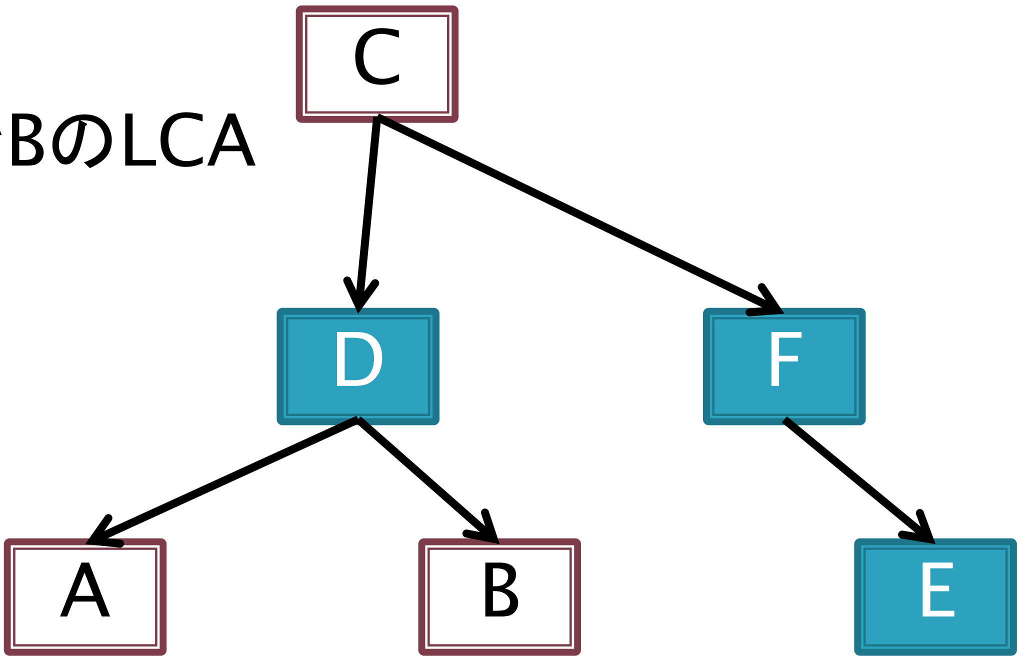
小課題2 (30点) – LCAならLCA

- CがAとBのLCA
  ↓辺を追加



小課題2 (30点) – LCAならLCA

- CがAとBのLCA
  ↓辺を追加
- Cは依然としてAとBのLCA



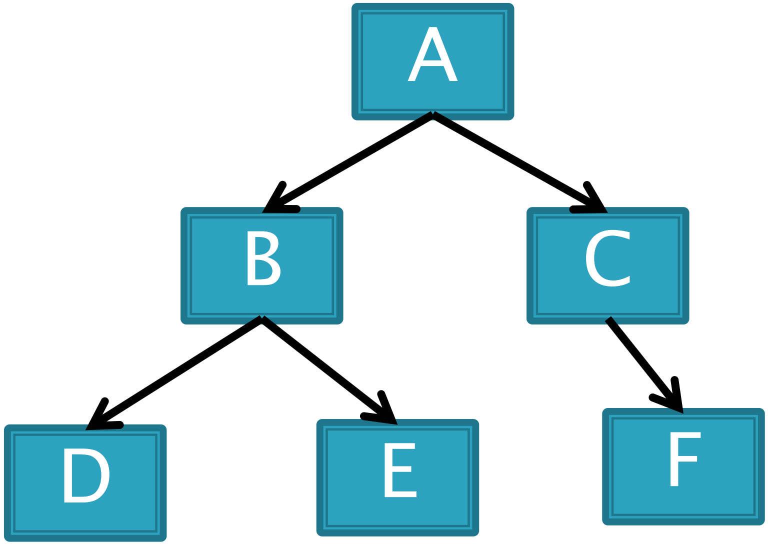
小課題2 (30点) – LCAならLCA

- CがAとBのLCA
  ↓辺を追加
- Cは依然としてAとBのLCA

- 最終的にできる木の上でLCAを計算できればよい

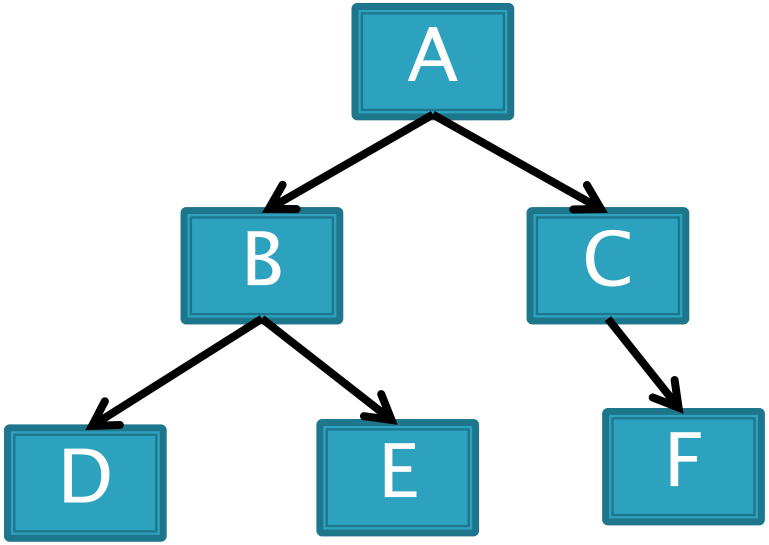
小課題2 (30点) – 1. LCAの計算

- 木の嬉しい順序(DFS順序)
  - Preorder –頂点に入るときに記録する順序
- A, B, D, E, C, F



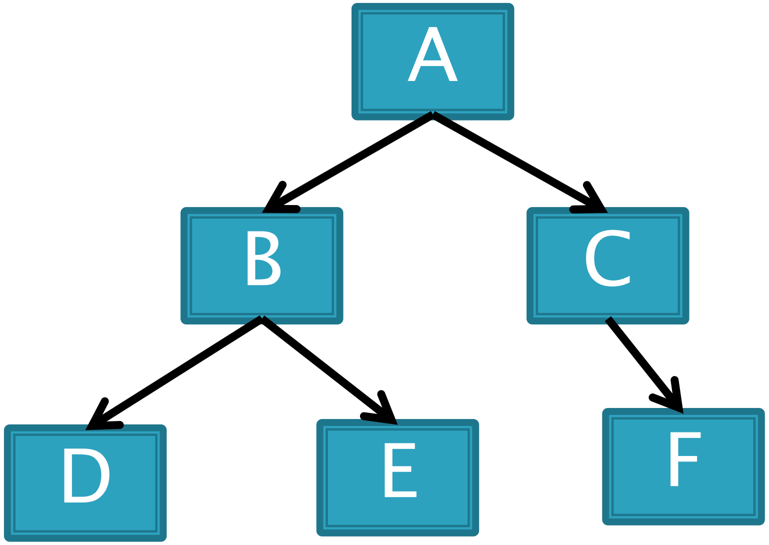
小課題2 (30点) – 1. LCAの計算

- 木の嬉しい順序(DFS順序)
  - Postorder – 頂点から出るときに記録する順序
- D, E, B, F, C, A



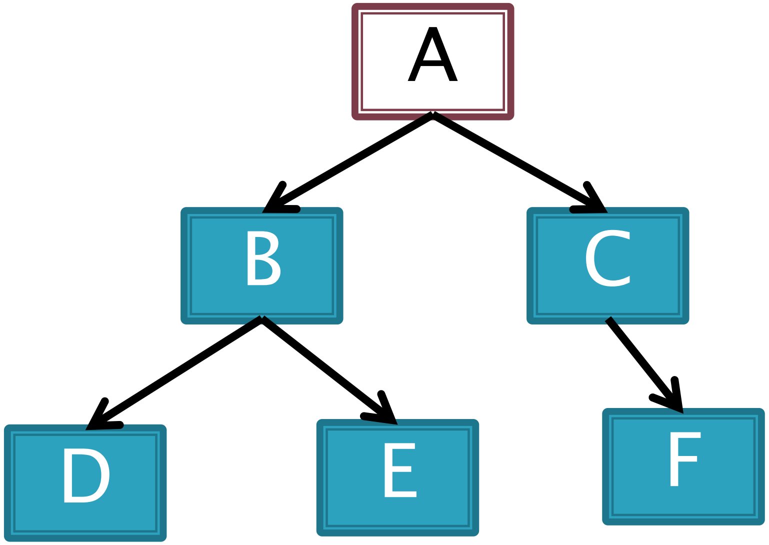
小課題2 (30点) – 1. LCAの計算

- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



小課題2 (30点) – 1. LCAの計算

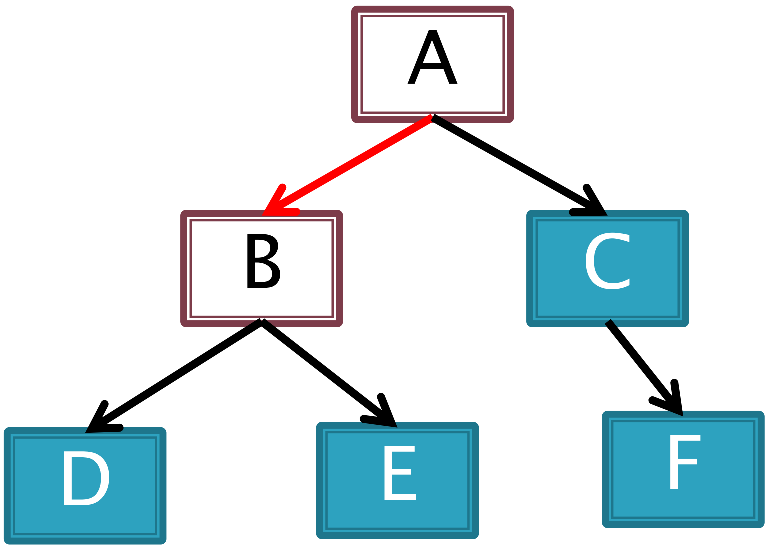
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A,

小課題2 (30点) – 1. LCAの計算

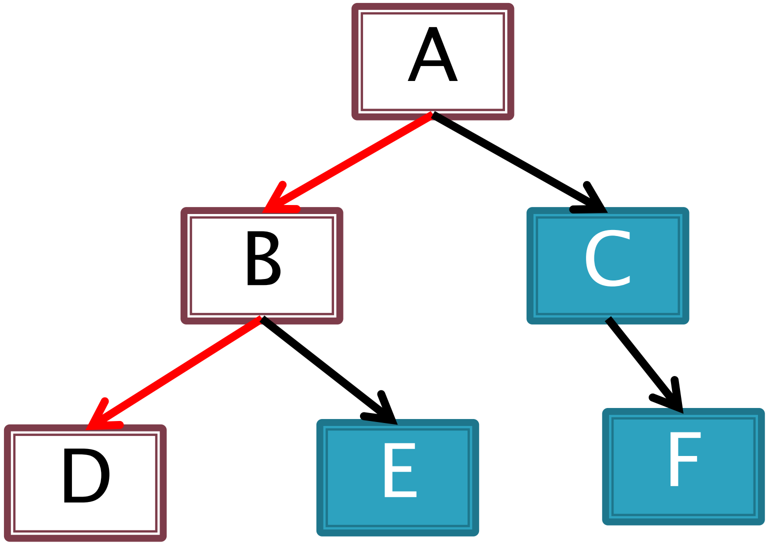
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A, B,

小課題2 (30点) – 1. LCAの計算

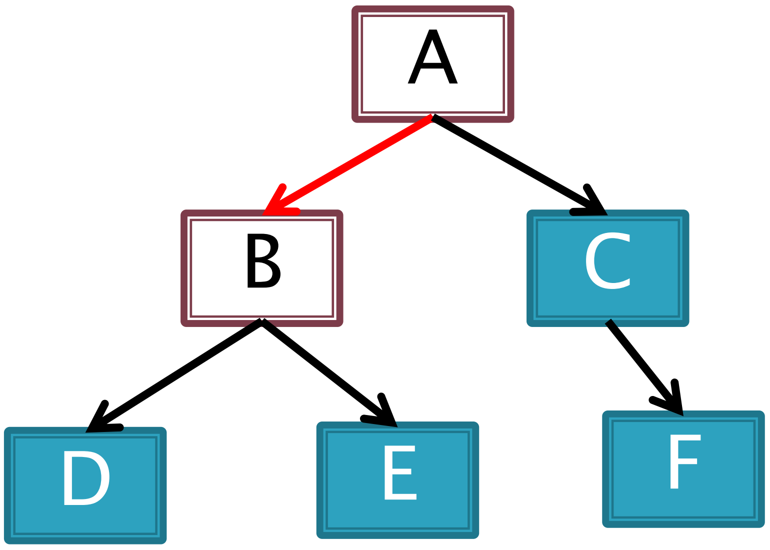
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A, B, D,

小課題2 (30点) – 1. LCAの計算

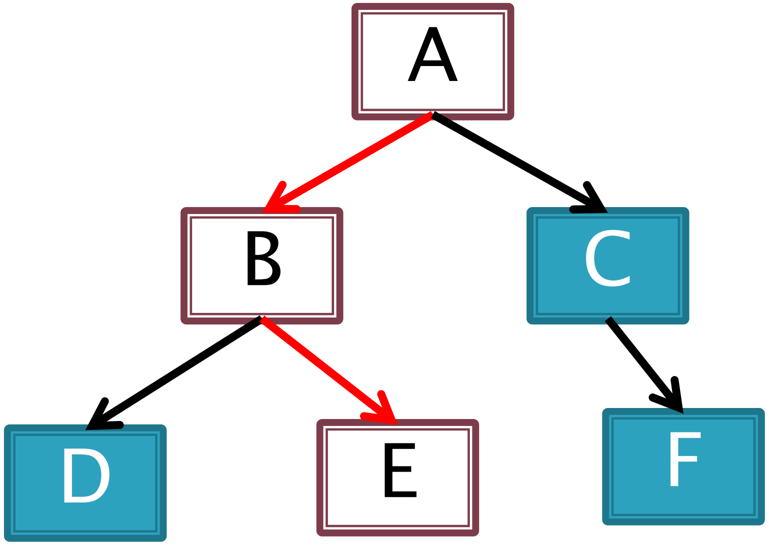
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A, B, D, B,

小課題2 (30点) – 1. LCAの計算

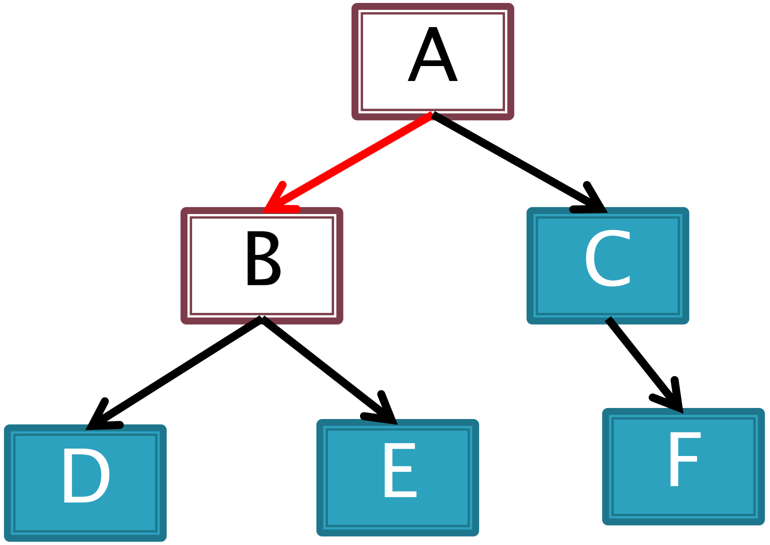
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A, B, D, B, E

小課題2 (30点) – 1. LCAの計算

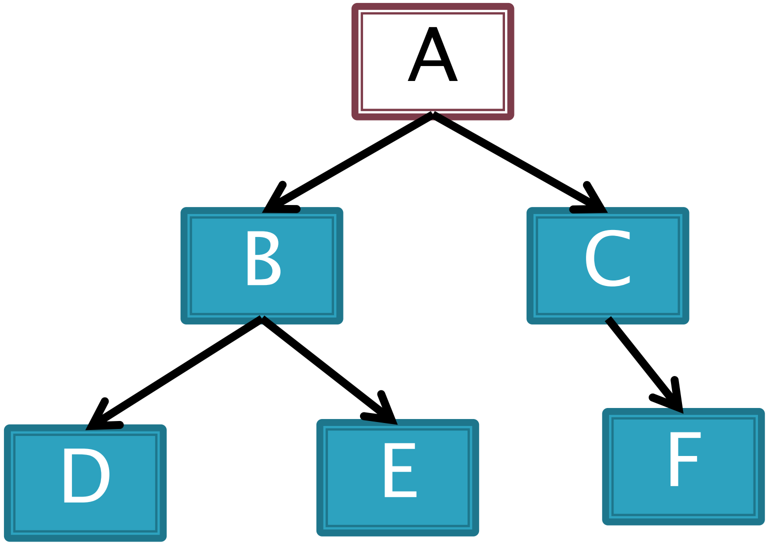
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A, B, D, B, E, B,

小課題2 (30点) – 1. LCAの計算

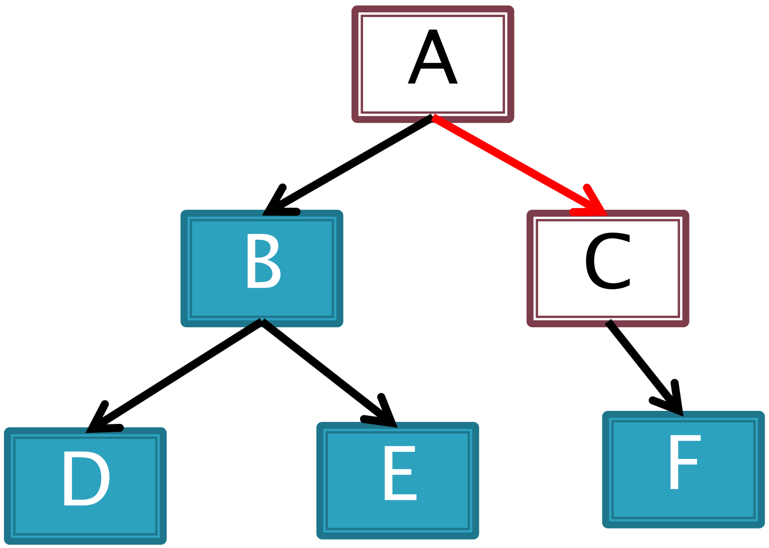
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A, B, D, B, E, B, A,

小課題2 (30点) – 1. LCAの計算

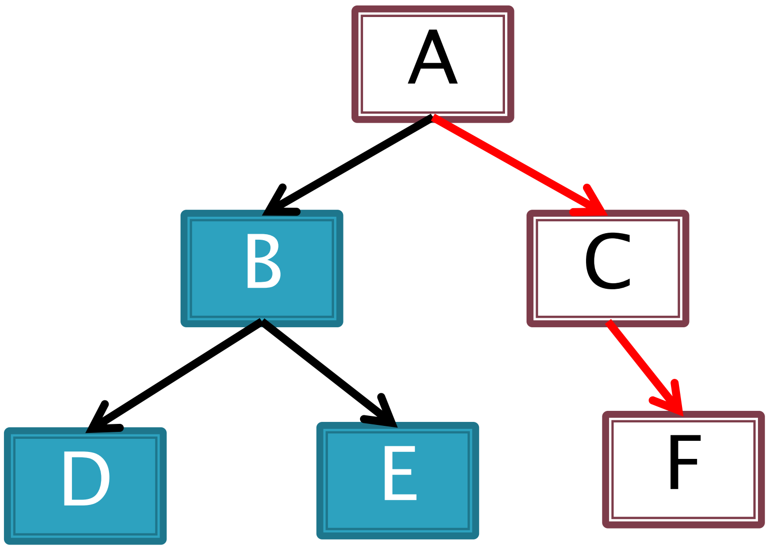
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A, B, D, B, E, B, A,
C,

小課題2 (30点) – 1. LCAの計算

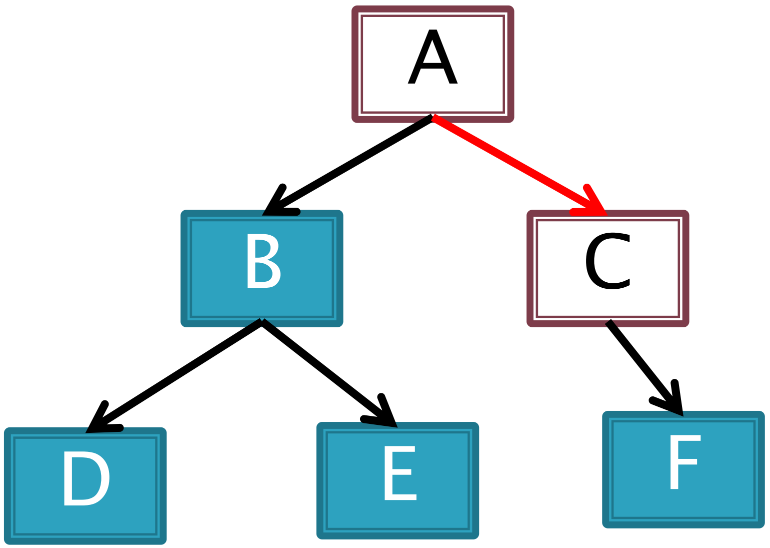
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A, B, D, B, E, B, A,
C, F,

小課題2 (30点) – 1. LCAの計算

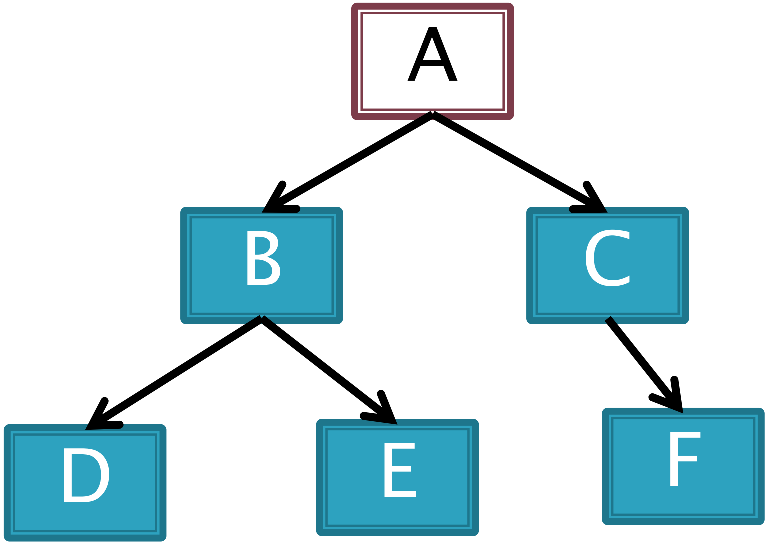
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A, B, D, B, E, B, A,
C, F, C,

小課題2 (30点) – 1. LCAの計算

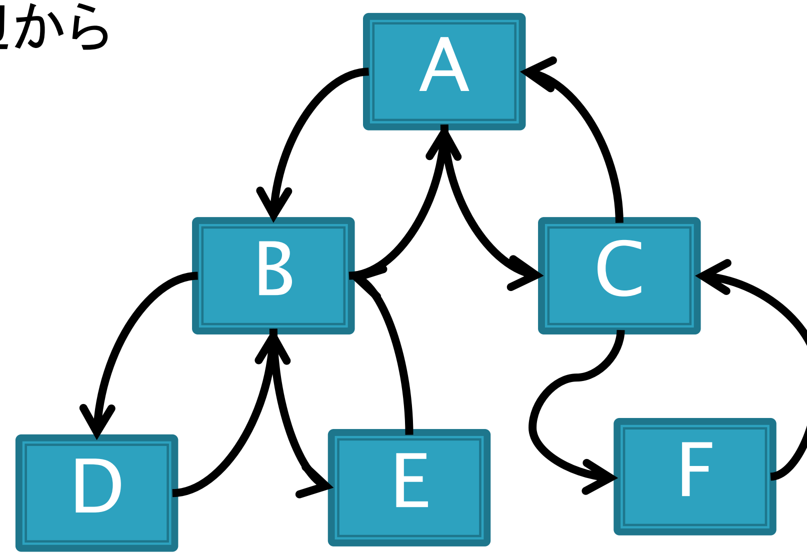
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 頂点から入るときに記録し、
    - 頂点から出るときにも
      自分の親を記録する順序



A, B, D, B, E, B, A,
C, F, C, A

小課題2 (30点) – 1. LCAの計算

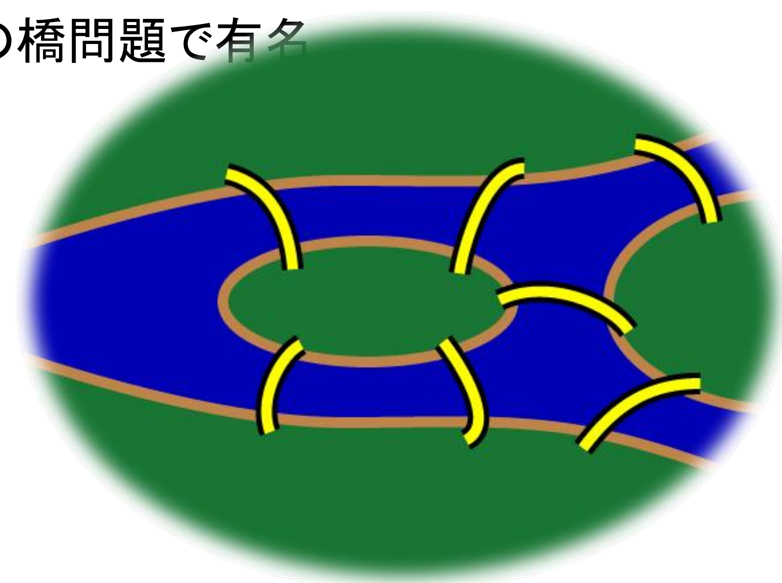
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 木の辺を、行きと帰りの2つの辺からなるとみなすときのオイラー閉路に対応する



A, B, D, B, E, B, A, C, F, C, A

小課題2 (30点) – 1. LCAの計算

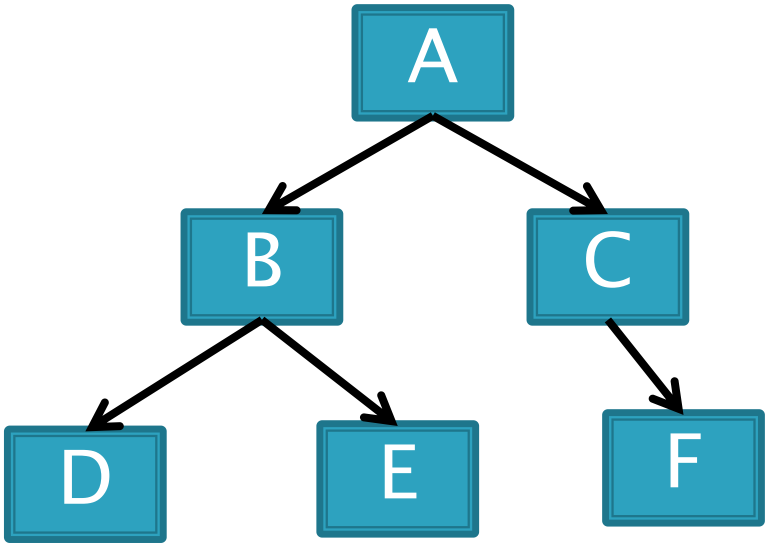
- 木の嬉しい順序(DFS順序)
  - Euler Tour
    - 木の辺を、行きと帰りの2つの辺から
      なるとみなすときの
      オイラー閉路に対応する
    - オイラー閉路 (Euler Tour) : 全ての辺を1度ずつ通る閉路
      - オイラー路はケーニヒスベルクの橋問題で有名



Wikipediaより。CC3.0-BY-SA

小課題2 (30点) – 1. LCAの計算

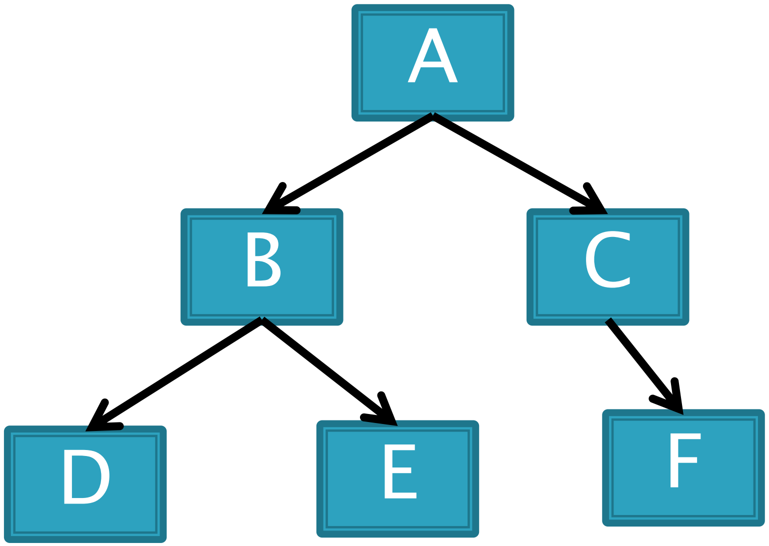
- Euler TourによるLCAの計算



小課題2 (30点) – 1. LCAの計算

- Euler TourによるLCAの計算

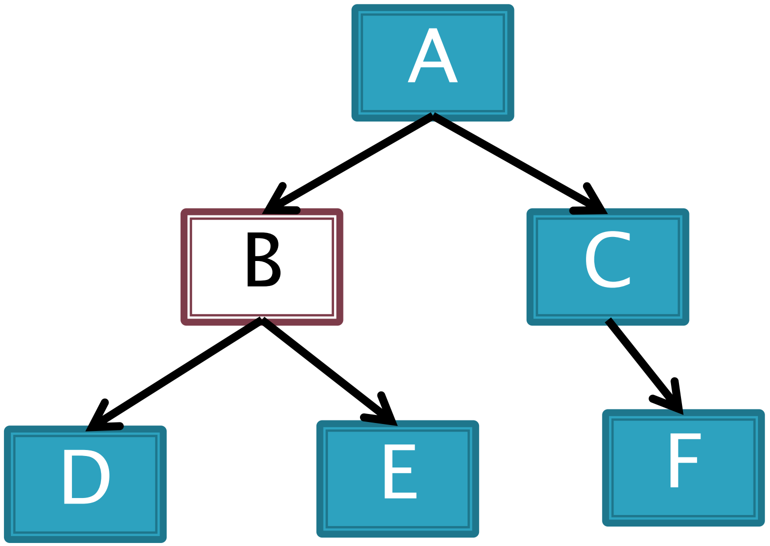
	A	B	D	B	E	B	A	C	F	C	A
	1	2	3	2	3	2	1	2	3	2	1



小課題2 (30点) – 1. LCAの計算

- Euler TourによるLCAの計算

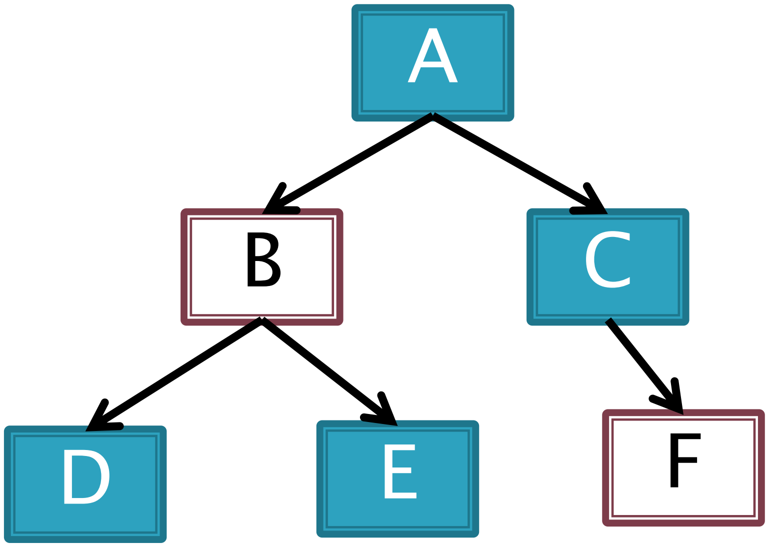
	A	B	D	B	E	B	A	C	F	C	A
	1	2	3	2	3	2	1	2	3	2	1



小課題2 (30点) – 1. LCAの計算

- Euler TourによるLCAの計算

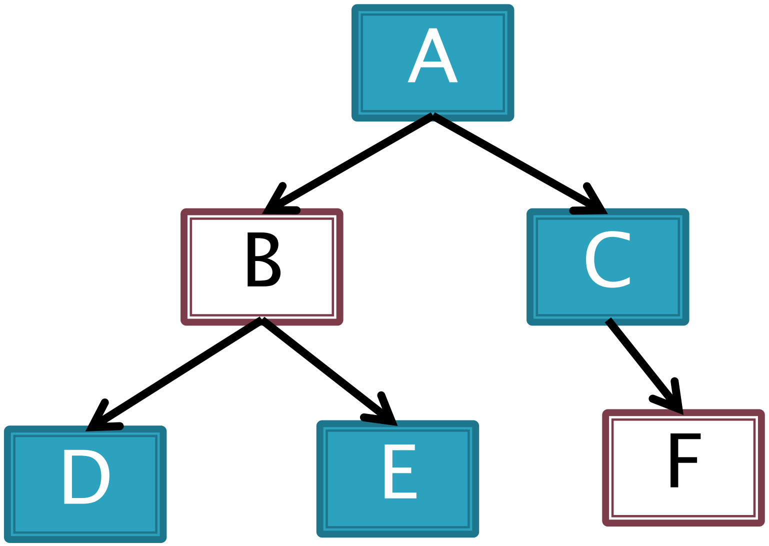
	A	B	D	B	E	B	A	C	F	C	A
	1	2	3	2	3	2	1	2	3	2	1



小課題2 (30点) – 1. LCAの計算

- Euler TourによるLCAの計算

	A	B	D	B	E	B	A	C	F	C	A
	1	2	3	2	3	2	1	2	3	2	1

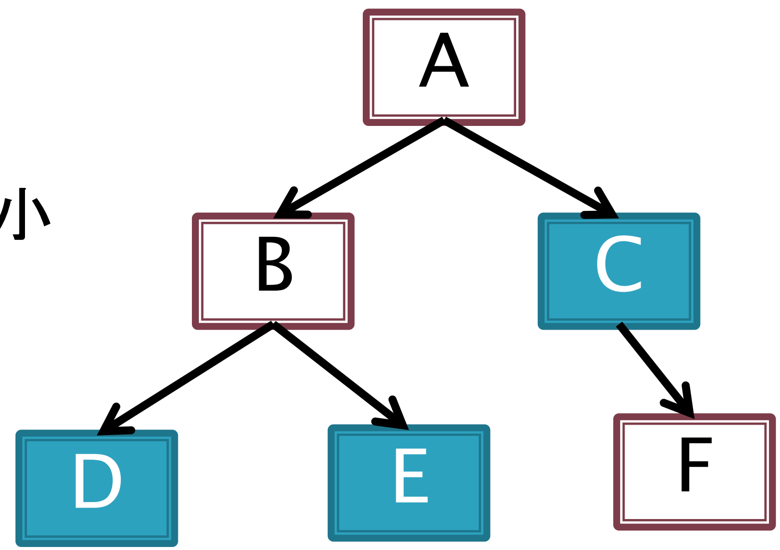


小課題2 (30点) – 1. LCAの計算

- Euler TourによるLCAの計算

	A	B	D	B	E	B	A	C	F	C	A
	1	2	3	2	3	2	1	2	3	2	1

深さ最小



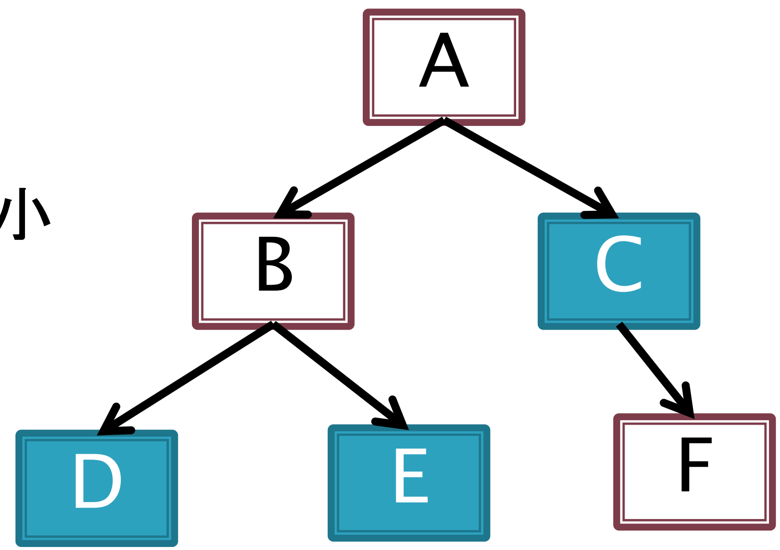
小課題2 (30点) – 1. LCAの計算

- Euler TourによるLCAの計算

	A	B	D	B	E	B	A	C	F	C	A
	1	2	3	2	3	2	1	2	3	2	1

深さ最小

- RMQを利用



小課題2 (30点) – 1. LCAの計算

- 正当性

小課題2 (30点) – 1. LCAの計算

- 正当性: CがAとBのLCAのとき
  - Euler Tour上でCは[A,B]に含まれる
  - Euler Tour上で[A,B]に含まれるのはCの部分木
- を言えばよい

小課題2 (30点) – 1. LCAの計算

- 正当性(1): Euler Tour上でCは[A,B]に含まれる
  - AからBに行くにはCを経由しないといけない(LCAの性質より)
  - ので当たり前

小課題2 (30点) – 1. LCAの計算

- 正当性(2): Euler Tour上で[A,B]に含まれるのはCの部分木
  - Euler Tourにおいて部分木は連続した部分列として現れる
  - ので当たり前

小課題2 (30点)

- LCAを求めて終わり？

小課題2 (30点)

- LCAを求めて終わり？
  - あと少しだけやることがあります

小課題2 (30点) – 2. 連結性

- 「削除がない場合の利点」の考察を思い出す

小課題2 (30点) – 2. 連結性

- 「削除がない場合の利点」の考察を思い出す
- LCAが存在するなら、最終的な木の上で計算すればよい

小課題2 (30点) – 2. 連結性

- 「削除がない場合の利点」の考察を思い出す
- LCAが存在するなら、最終的な木の上で計算すればよい

小課題2 (30点) – 2. 連結性

- A, Bが同じ木上にあるかどうかの判定が必要

小課題2 (30点) – 2. 連結性

- A, Bが同じ木上にあるかどうかの判定が必要

  ですが

小課題2 (30点) – 2. 連結性

- A, Bが同じ木上にあるかどうかの判定が必要

  ですが

  辺の追加クエリしかないのでUnionFindでよい
  ということはすぐにわかると思います

小課題2 (30点)

- 頂点数 $N \leq 10^6$
- クエリ数 $Q \leq 10^6$
- 辺の削除は行われない

- $O(N + Q \log N)$ なので間に合う

小課題1, 2

- ここまでの両方を実装すれば40点

小課題1, 2

- ここまでの両方を実装すれば40点

- 複数のアルゴリズムを条件によって使い分けるテクはさすがに使っていると思います

```
if(N <= 5000) {
  solve1(N, Q, T, A, B);
} else {
  solve2(N, Q, T, A, B);
}
```

小課題3 (60点)

- 頂点数 $N \leq 10^6$
- クエリ数 $Q \leq 10^6$

小課題3 (60点)

- 頂点数 $N \leq 10^6$
- クエリ数 $Q \leq 10^6$
- 削除クエリもある

小課題3 (60点)

- 追加も削除もある場合の頻出テク

小課題3 (60点)

- 追加も削除もある場合の頻出テク
  - クエリの(平方)分割
  - がんばって動的になんとかする

小課題3 (60点)

- 追加も削除もある場合の頻出テク
  - <u>がんばって動的になんとかする</u>

小課題3 (60点)

- 追加も削除もある場合の頻出テク
  - <u>**がんばって**動的になんとかする</u>

小課題3 (60点) – 動的Euler Tour

- 木が静的な場合のLCA (復習)

小課題3 (60点) – 動的Euler Tour

- 木が静的な場合のLCA (復習)
  - Euler Tour上で必要とされるクエリは以下の通り

小課題3 (60点) – 動的Euler Tour

- 木が静的な場合のLCA (復習)
  - Euler Tour上で必要とされるクエリは以下の通り

1. 区間の最小値をとる

小課題3 (60点) – 動的Euler Tour

- 木が静的な場合のLCA (復習)
  - Euler Tour上で必要とされるクエリは以下の通り
  1. 区間の最小値をとる
  - RMQで実現可能

小課題3 (60点) – 動的Euler Tour

- 木が動的な場合のLCA (絶望)
  - Euler Tour上で必要とされるクエリは以下の通り
  1. 区間の最小値をとる
  2.
  - RMQで実現可能な気がする

小課題3 (60点) – 動的Euler Tour

- 木が動的な場合のLCA (絶望)
  - Euler Tour上で必要とされるクエリは以下の通り
  1. 区間の最小値をとる
  2. 列の連結をする
  3.
  - RMQで実現可能な気がする

小課題3 (60点) – 動的Euler Tour

- 木が動的な場合のLCA (絶望)
  - Euler Tour上で必要とされるクエリは以下の通り
  1. 区間の最小値をとる
  2. 列の連結をする
  3. 列の分割をする
  4.
  - RMQで実現可能な気がする

小課題3 (60点) – 動的Euler Tour

- 木が動的な場合のLCA (絶望)
  - Euler Tour上で必要とされるクエリは以下の通り

1. 区間の最小値をとる
2. 列の連結をする
3. 列の分割をする
4. それだけ？

小課題3 (60点) – 動的Euler Tour

- 木が動的な場合のLCA (絶望)
  - Euler Tour上で必要とされるクエリは以下の通り

1. 区間の最小値をとる
2. 列の連結をする
3. 列の分割をする
4. 区間に値を足す

小課題3 (60点) – 動的Euler Tour

- 木が動的な場合のLCA (絶望)
  - Euler Tour上で必要とされるクエリは以下の通り
  1. 区間の最小値をとる
  2. 列の連結をする
  3. 列の分割をする
  4. 区間に値を足す
  - この業界では「Starry Sky木」として知られているもの

小課題3 (60点) – 動的Euler Tour

- 木が動的な場合のLCA (絶望)
  - Euler Tour上で必要とされるクエリは以下の通り
  1. 区間の最小値をとる
  2. 列の連結をする
  3. 列の分割をする
  4. 区間に値を足す
  - この業界では「Starry Sky木」として知られているものを**平衡二分木**として実装する必要がある (絶望)

小課題3 (60点) – 動的Euler Tour

- 木が動的な場合のLCA (絶望)
  - Euler Tour上で必要とされるクエリは以下の通り
  1. 区間の最小値をとる
  2. 列の連結をする
  3. 列の分割をする
  4. 区間に値を足す
  - この業界では「Starry Sky木」として知られているものを**平衡二分木**として実装する必要がある (絶望)
    - しかもmerge/splitベースで

小課題3 (60点) – 動的Euler Tour

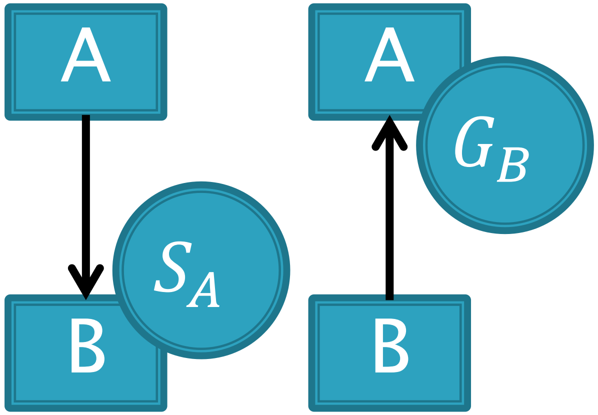
- 平衡二分木の中身は後回し

小課題3 (60点) – 動的Euler Tour

- 平衡二分木の中身は後回し
- 平衡二分木を使った具体的な実装方法

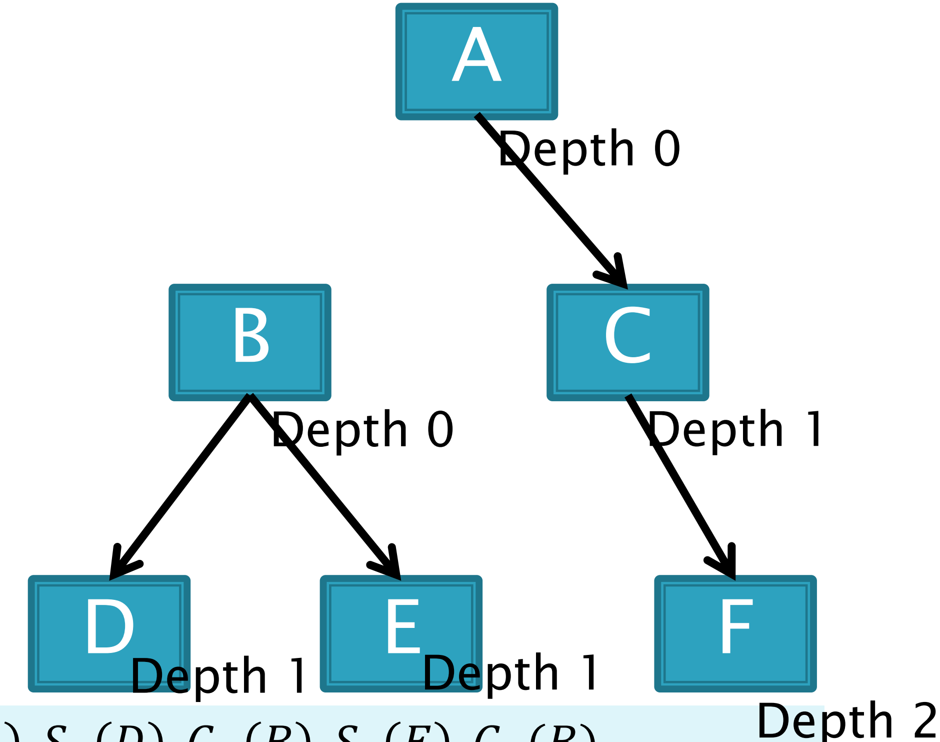
小課題3 (60点) – 動的Euler Tour

- 木の各頂点ごとに、Euler Tourのためのノードを2つ用意する(S_A, G_A)
  - S_A上には頂点Aの番号と、その深さが記録されている
  - G_A上には頂点Aの親Pの番号と、その深さが記録されている
  - Aが根のときはS_Aのみ使う



小課題3 (60点) – 動的Euler Tour

- 木の連結

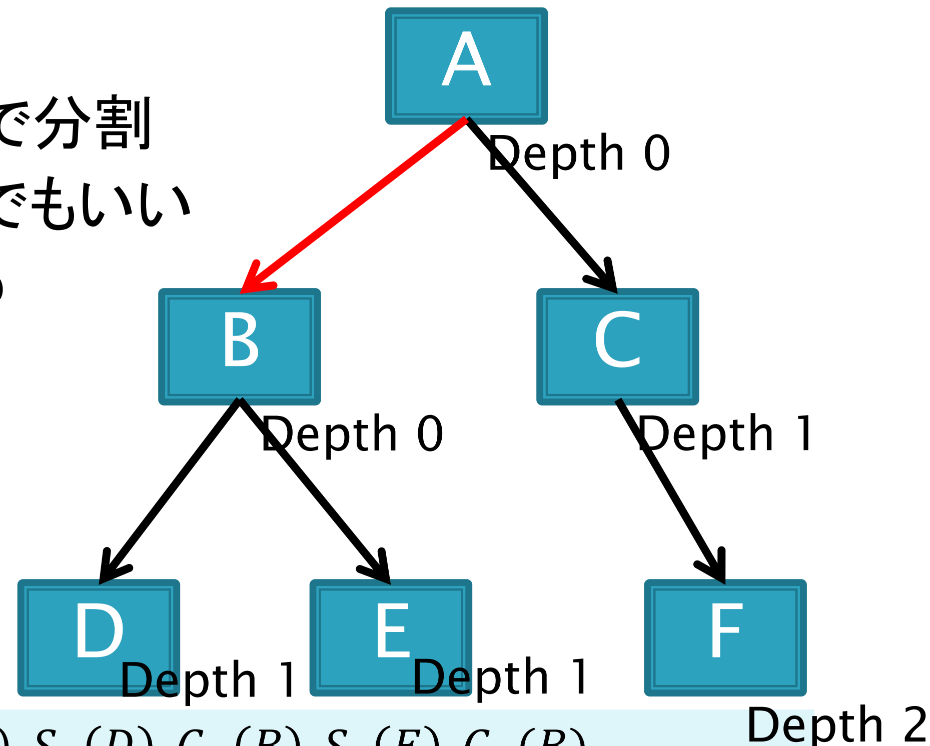


$S_B(B), S_D(D), G_D(B), S_E(E), G_E(B)$

$S_A(A), S_C(C), S_F(F), G_F(C), G_F(A)$

小課題3 (60点) – 動的Euler Tour

- 木の連結
  1. Euler TourをA点で分割
     - A直下ならどの位置でもいい
       - S_Aの直後がおすすめ

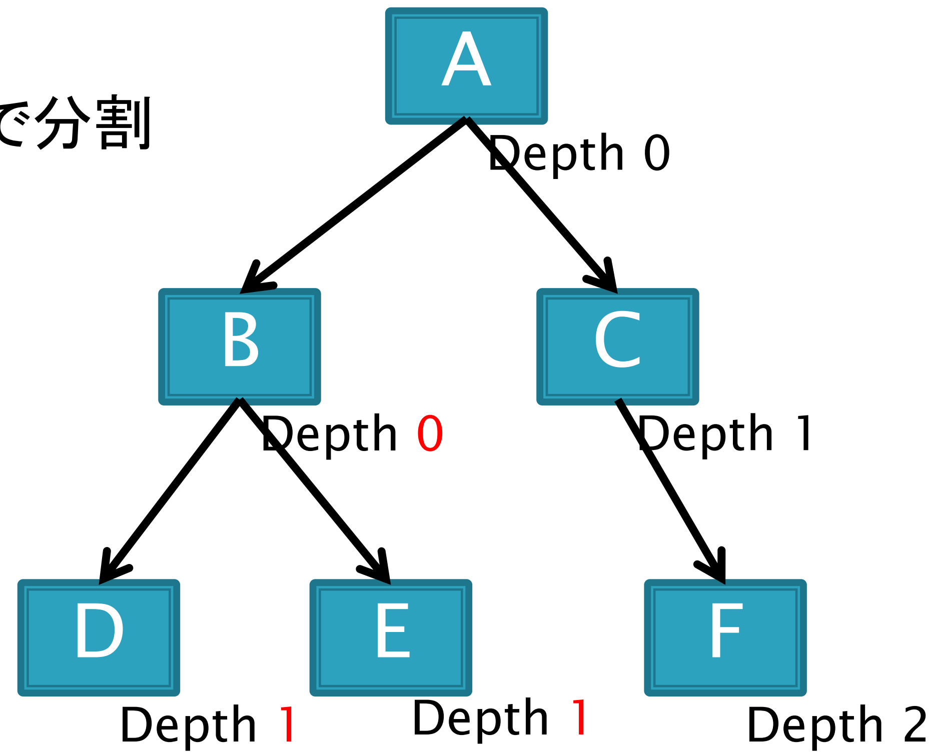


$S_B(B), S_D(D), G_D(B), S_E(E), G_E(B)$

$S_A(A),$ $S_C(C), S_F(F), G_F(C), G_F(A)$

小課題3 (60点) – 動的Euler Tour

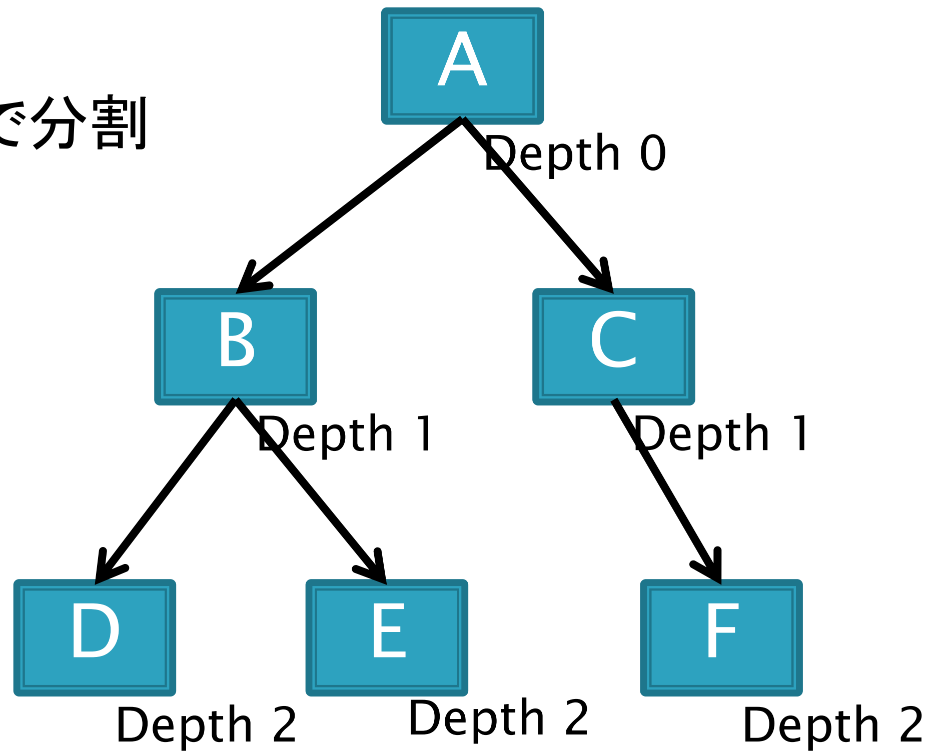
- 木の連結
  1. Euler TourをA点で分割
  2. Euler Tourを挿入



$$S_A(A), S_B(B), S_D(D), G_D(B), S_E(E), G_E(B), G_B(E), S_C(C), S_F(F), G_F(C), G_F(A)$$

小課題3 (60点) – 動的Euler Tour

- 木の連結
  1. Euler TourをA点で分割
  2. Euler Tourを挿入
  3. 深さを調整



$S_A(A), S_B(B), S_D(D), G_D(B), S_E(E), G_E(B), G_B(E), S_C(C), S_F(F), G_F(C), G_F(A)$

小課題3 (60点) – 動的Euler Tour

- 木の削除: 追加のときと逆操作

小課題3 (60点) – 動的Euler Tour

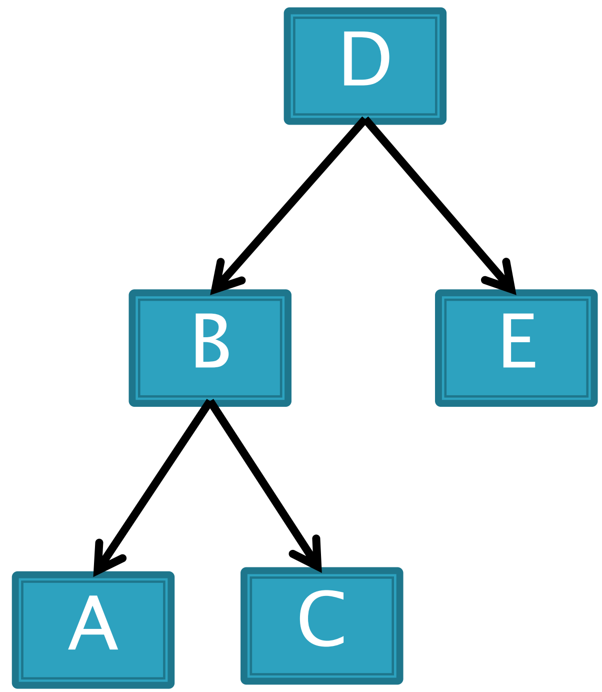
- 平衡Starry Sky Treeがあればよいことがわかった

小課題3 (60点) – SSTの回転

- 平衡Starry Sky Treeがあればよいことがわかった
- ではどのように実装するか？(実装例)

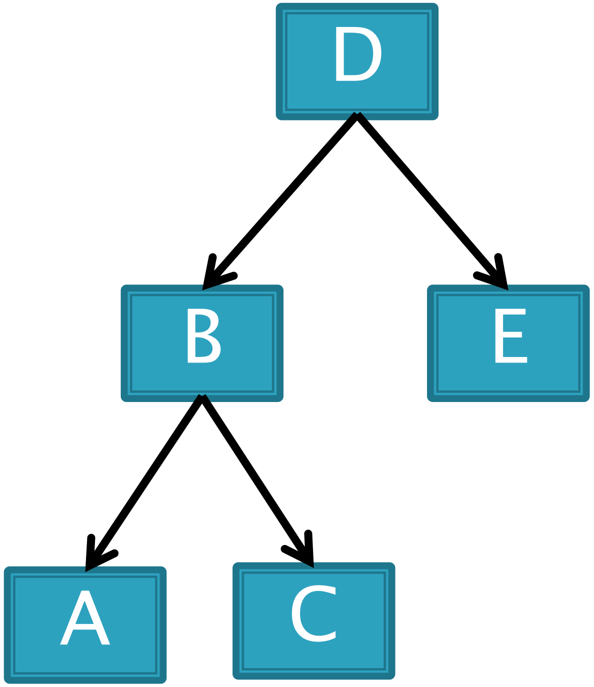
小課題3 (60点) – SSTの回転

- 今回は、葉ノードと内部ノードの区別をしない
  - A,B,C,D,Eはどれも列上の項目とする



小課題3 (60点) – SSTの回転

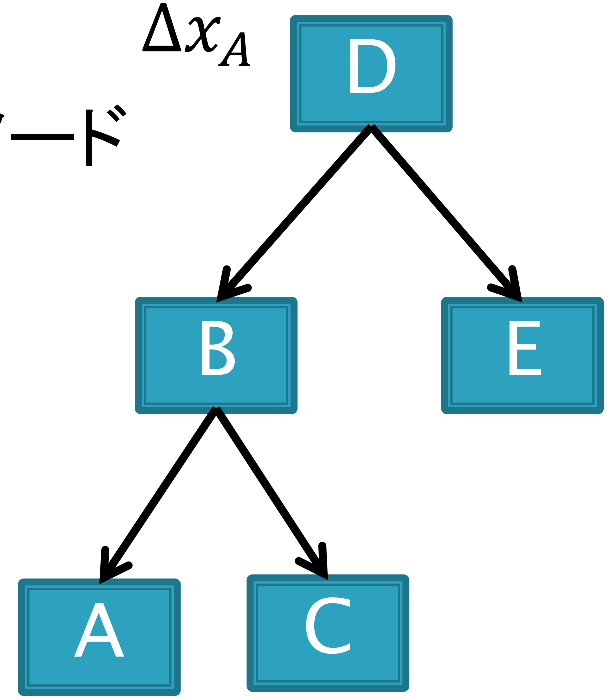
- 各ノードは、Δx_Aというフィールドを持つ



小課題3 (60点) – SSTの回転

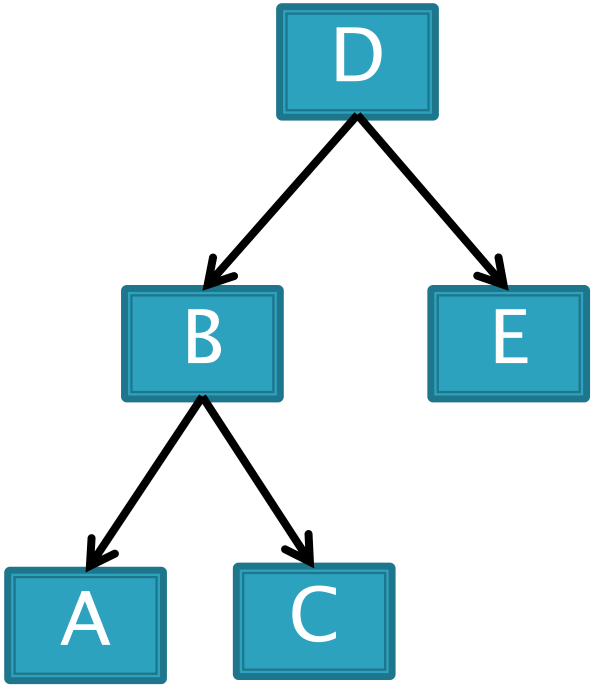
- 各ノードは、Δx_Aというフィールドを持つ
- 各ノードに定めたい値x_Aは、

$$x_A = \sum_{A\text{から根までのパス上のノード}} \Delta x_A$$



小課題3 (60点) – SSTの回転

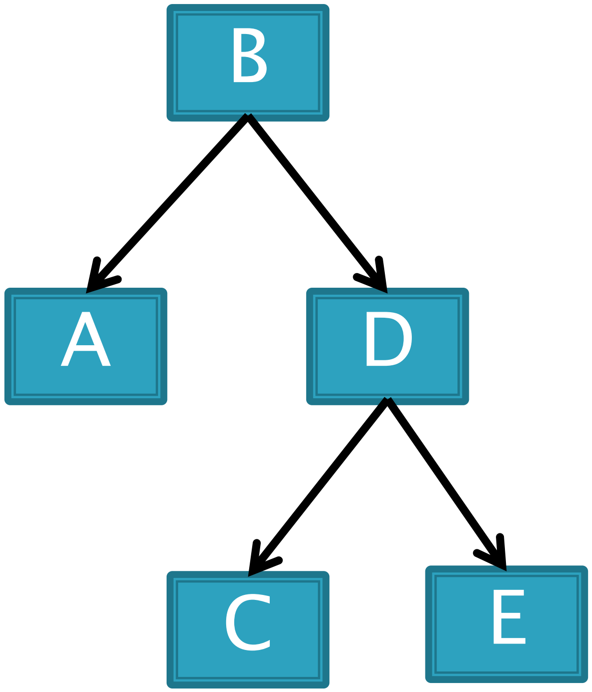
- $x_A = \Delta x_D + \Delta x_B + \Delta x_A$
- $x_B = \Delta x_D + \Delta x_B$
- $x_C = \Delta x_D + \Delta x_B + \Delta x_C$
- $x_D = \Delta x_D$
- $x_E = \Delta x_D + \Delta x_E$



小課題3 (60点) – SSTの回転

- 木の回転: x_Aが保存されるように行う

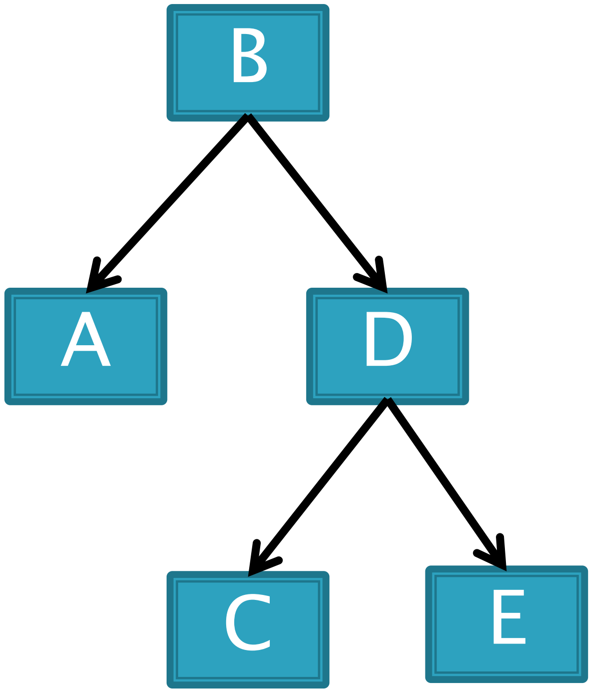
- $\Delta x'_A = x_A - x_B$
- $\Delta x'_B = x_B$
- $\Delta x'_C = x_C - x_D$
- $\Delta x'_D = x_D - x_B$
- $\Delta x'_E = x_E - x_D$



小課題3 (60点) – SSTの回転

- 木の回転: x_Aが保存されるように行う

- $\Delta x'_A = \Delta x_A$
- $\Delta x'_B = \Delta x_D + \Delta x_B$
- $\Delta x'_C = \Delta x_B + \Delta x_C$
- $\Delta x'_D = -\Delta x_B$
- $\Delta x'_E = \Delta x_E$



小課題3 (60点) – SSTの回転

- 最後に、ノードに値y_Aを
- $y_A = \min_{A\text{の全ての子孫}B} x_B - x_A$
- となるように計算して保持しておく

小課題3 (60点) – SSTの回転

- 最後に、ノードに値y_Aを
- $y_A = \min_{Aの全ての子孫B} x_B - x_A$
- となるように計算して保持しておく

- これでStarry Sky Tree相当の計算を行えるようになる

小課題3 (60点) – 平衡二分木

- 平衡二分木の基本

小課題3 (60点) – 平衡二分木

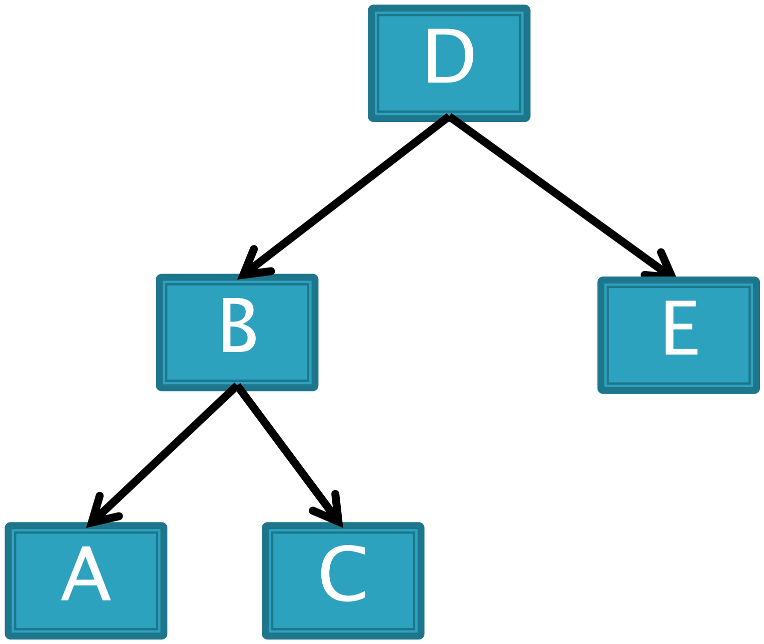
- 平衡二分木の基本:回転操作

小課題3 (60点) – 平衡二分木

- 回転操作:　順序を保存したまま木構造を変形

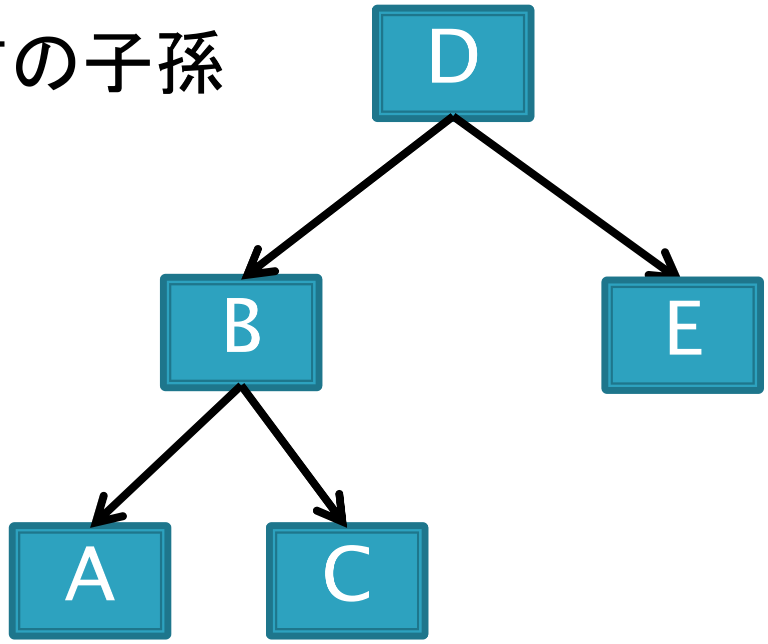
小課題3 (60点) – 平衡二分木

- 回転操作：　順序を保存したまま木構造を変形
- 次のような二分木を考える



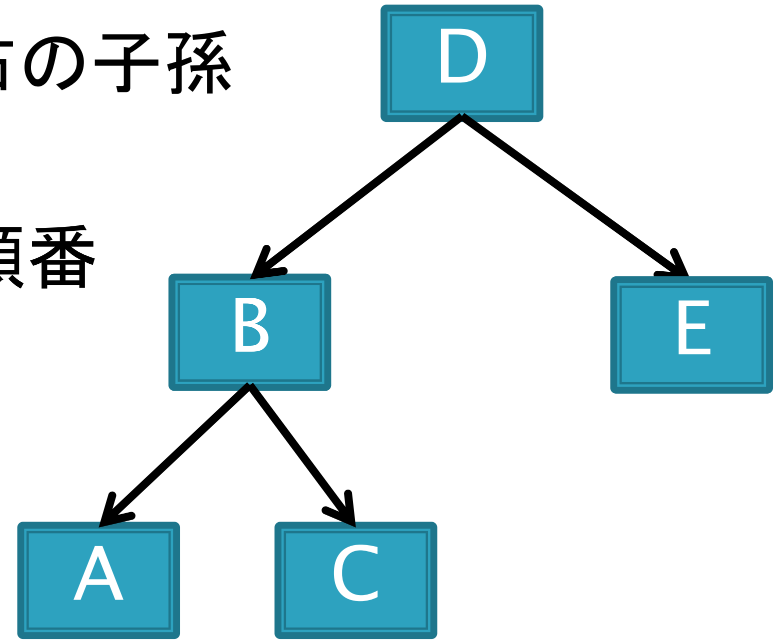
小課題3 (60点) – 平衡二分木

- 回転操作：　順序を保存したまま木構造を変形
- 次のような二分木を考える
- 順序：　左の子孫→自分→右の子孫



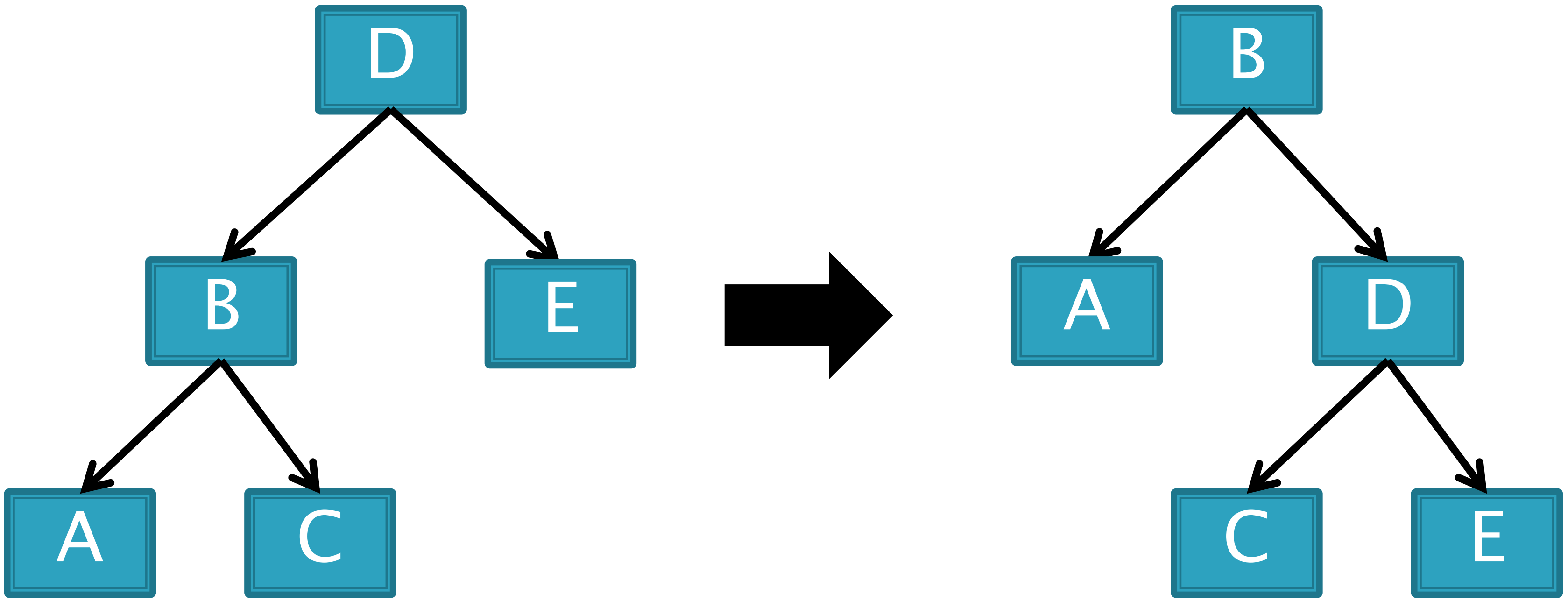
小課題3 (60点) – 平衡二分木

- 回転操作：　順序を保存したまま木構造を変形
- 次のような二分木を考える
- 順序：　左の子孫→自分→右の子孫
- この場合は A, B, C, D, E の順番



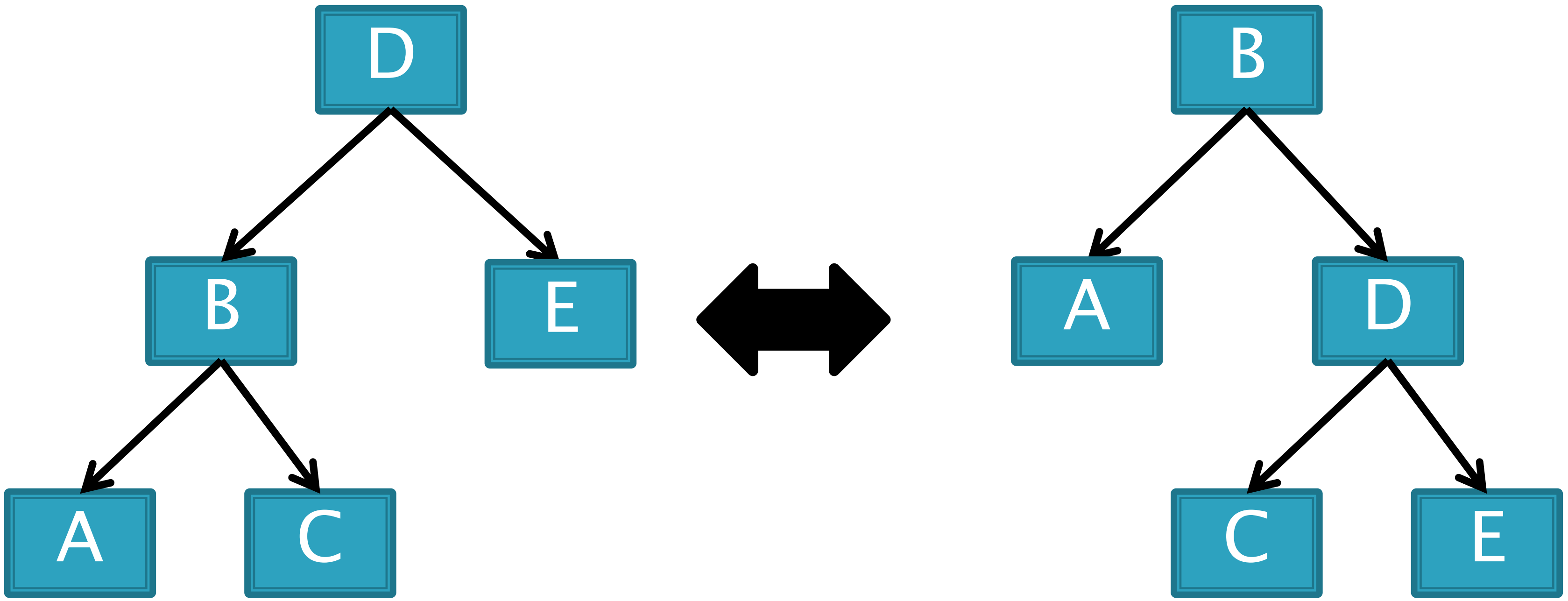
小課題3 (60点) – 平衡二分木

- 次のように変形しても順番はA,B,C,D,E



小課題3 (60点) – 平衡二分木

- 次のように変形しても順番はA,B,C,D,E
- これを「木の回転」と言う



小課題3 (60点) – 平衡二分木

- 次のように変形しても順番はA,B,C,D,E
- これを「木の回転」と言う

- うまく回転をすることで、偏りが起きないようにする二分木を「平衡二分木」と言う
  - 回転以外の方法で平衡を保つものもある

小課題3 (60点) – 平衡二分木

- 平衡二分木の実装方法

小課題3 (60点) – 平衡二分木

- 平衡二分木の実装方法

- 今回は何でもOK!
  - 赤黒木
  - RBST
  - Treap
  - Splay木
  - などなど…

小課題3 (60点) – Splay木

- この解説ではSplay木の説明をします

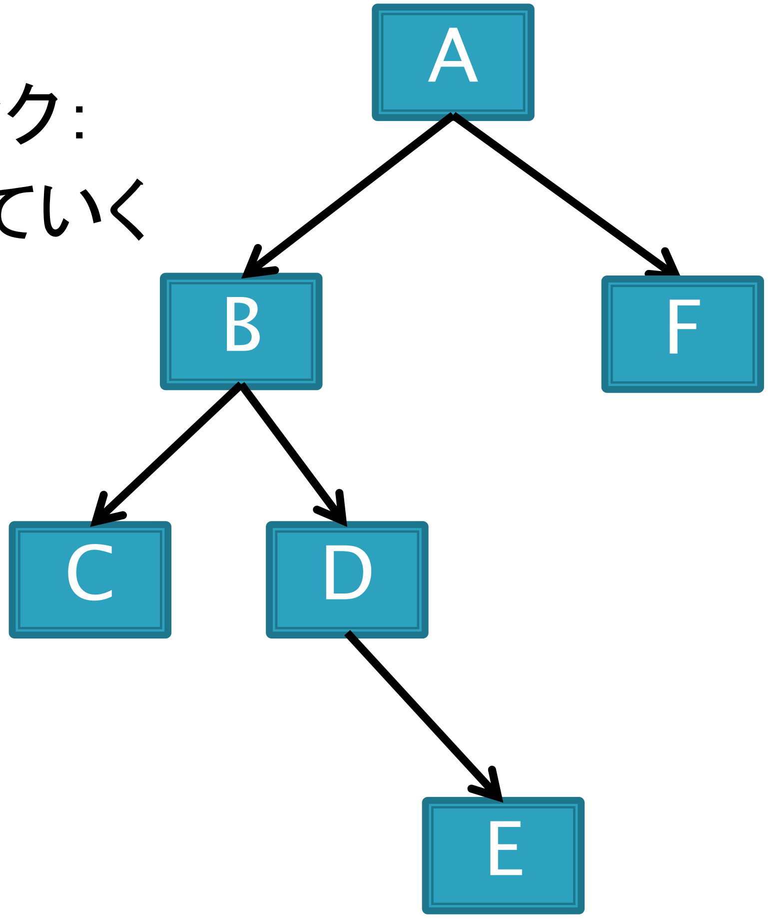
小課題3 (60点) – Splay木

- 突然ですが、Union Findの復習をします

小課題3 (60点) – Splay木

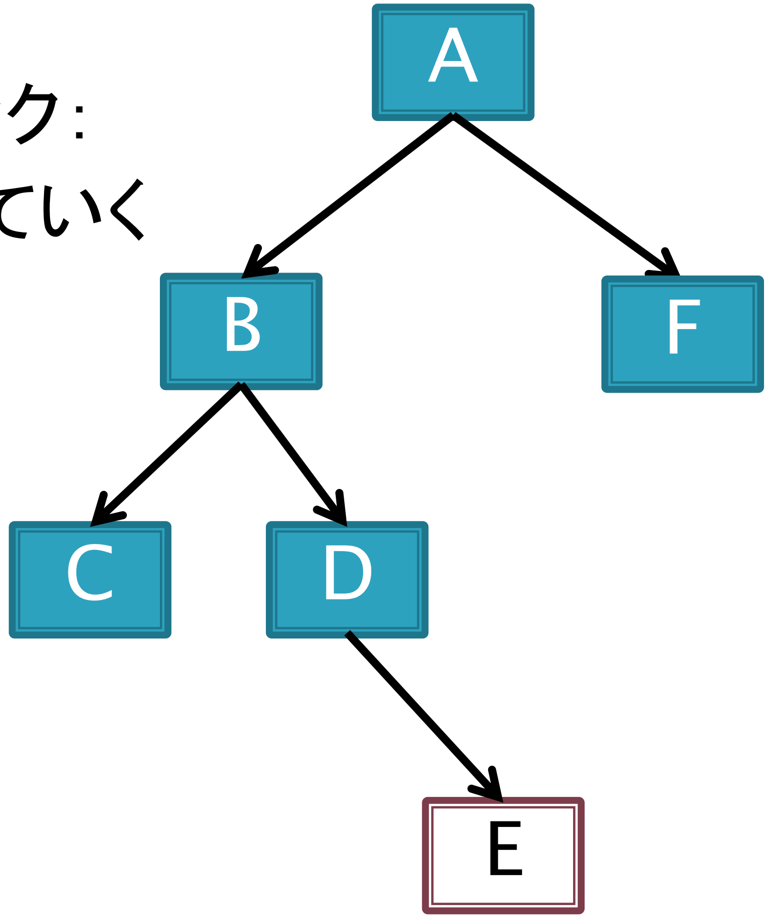
- 突然ですが、Union Findの復習をします

- Union Find の効率化テクニック:
- アクセスした頂点を根へ持っていく



小課題3 (60点) – Splay木

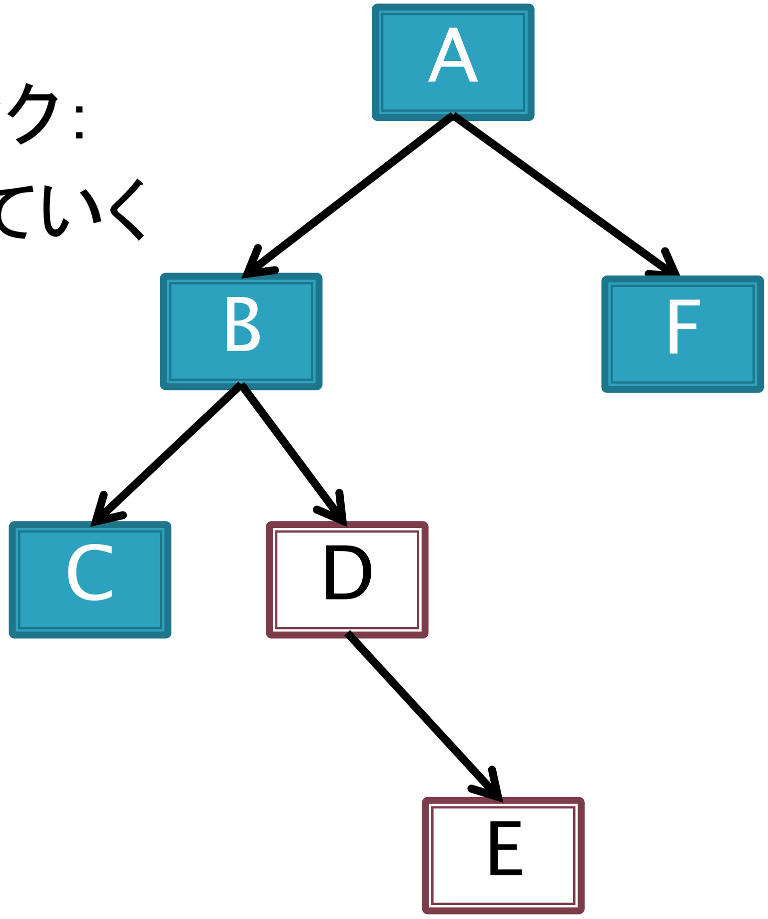
- 突然ですが、Union Findの復習をします
- Union Find の効率化テクニック:
- アクセスした頂点を根へ持っていく



小課題3 (60点) – Splay木

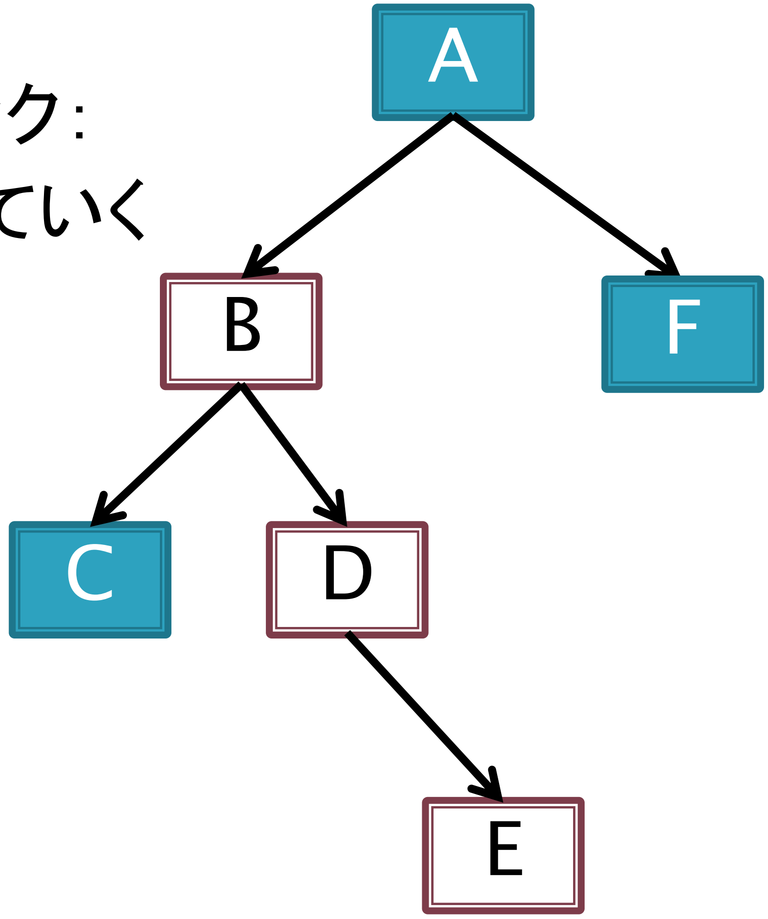
- 突然ですが、Union Findの復習をします

- Union Find の効率化テクニック:
- アクセスした頂点を根へ持っていく



小課題3 (60点) – Splay木

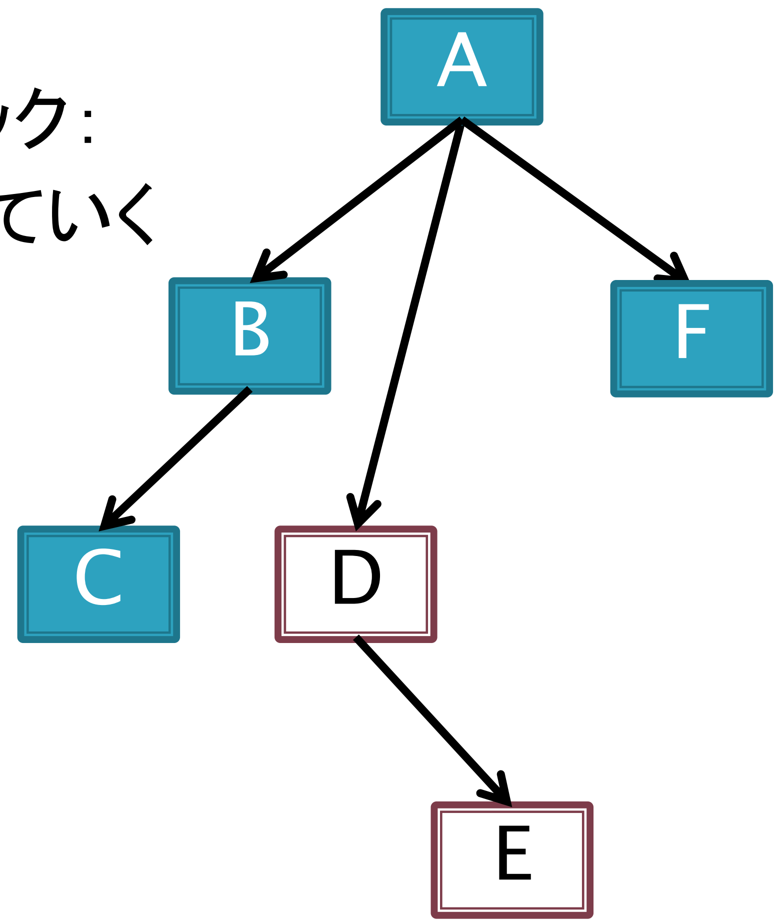
- 突然ですが、Union Findの復習をします
- Union Find の効率化テクニック:
- アクセスした頂点を根へ持っていく



小課題3 (60点) – Splay木

- 突然ですが、Union Findの復習をします

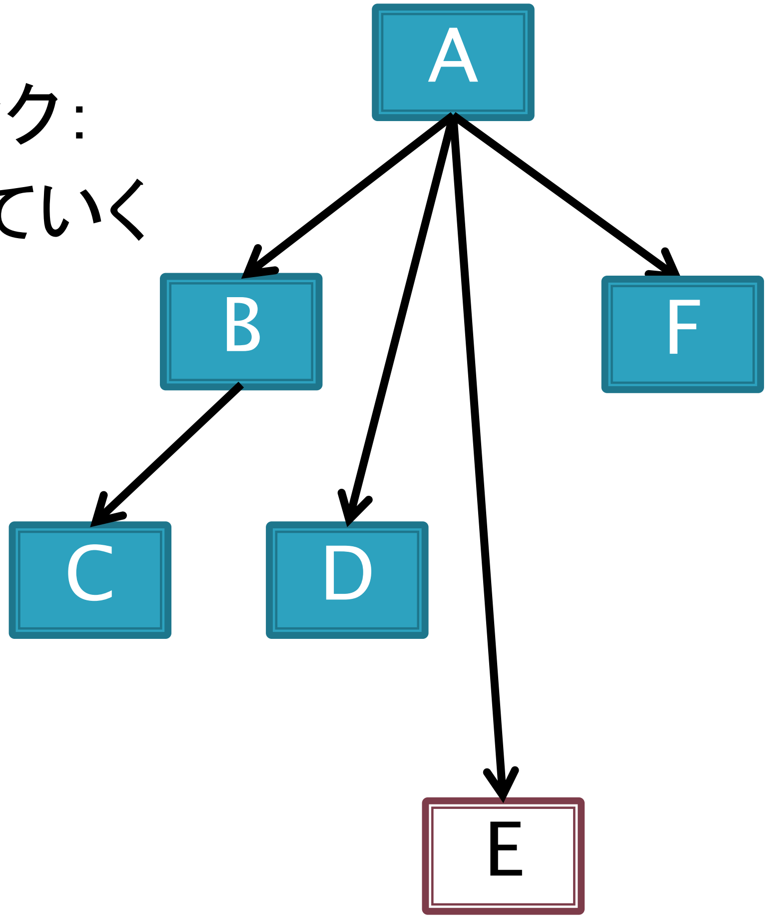
- Union Find の効率化テクニック:
- アクセスした頂点を根へ持っていく



小課題3 (60点) – Splay木

- 突然ですが、Union Findの復習をします

- Union Find の効率化テクニック:
- アクセスした頂点を根へ持っていく



小課題3 (60点) – Splay木

- 同じようなことを、二分探索木でもできないか？

小課題3 (60点) – Splay木

- 同じようなことを、二分探索木でもできないか？
  - →Move-to-root heuristic

小課題3 (60点) – Splay木

- Move-to-root heuristic
  - 頂点にアクセスしたら、それが根に行くまで繰り返し回転する

小課題3 (60点) – Splay木

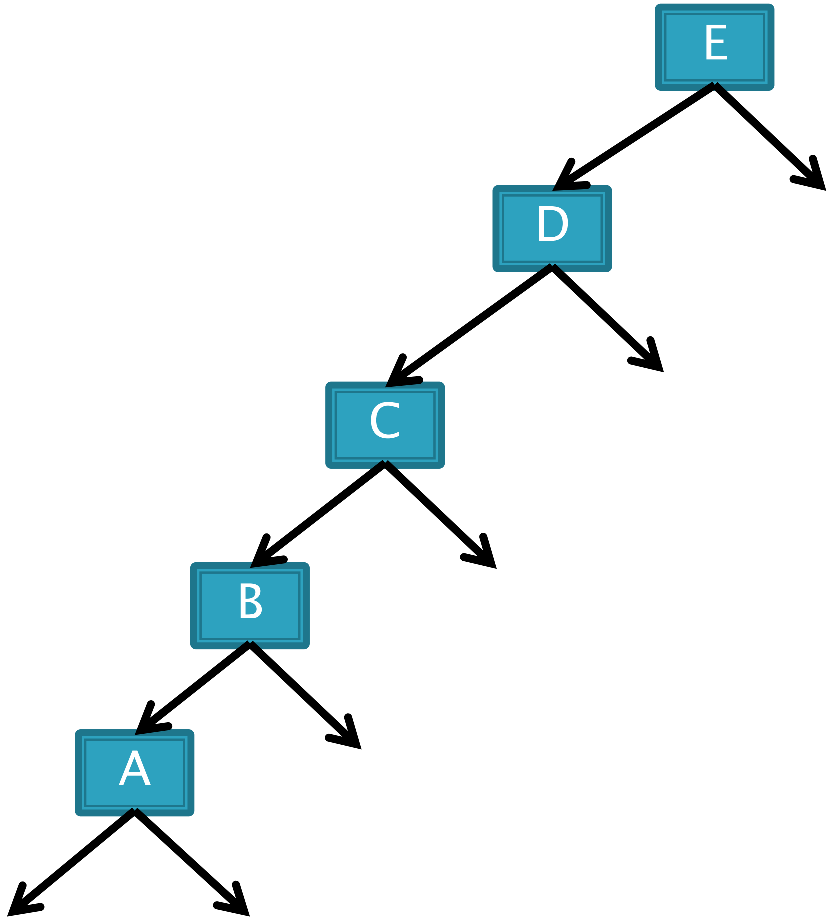
- Move-to-root heuristic
  - 頂点にアクセスしたら、それが根に行くまで繰り返し回転する
  - そんなので上手くいくわけないだろ！！

小課題3 (60点) – Splay木

- 実際ダメ

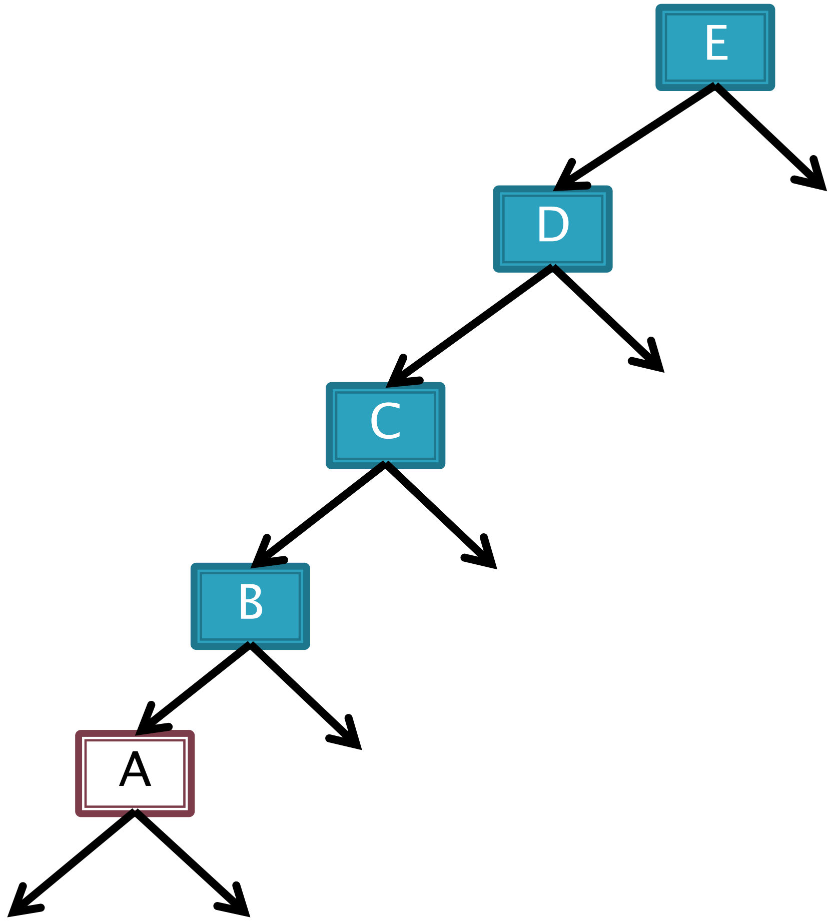
小課題3 (60点) – Splay木

- 実際ダメ



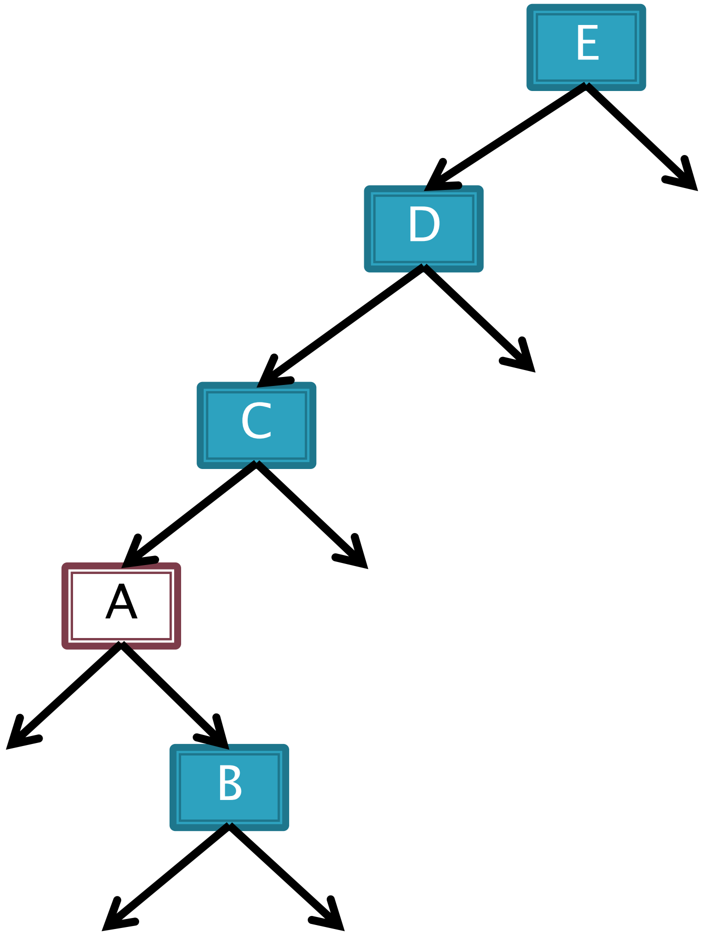
小課題3 (60点) – Splay木

- 実際ダメ



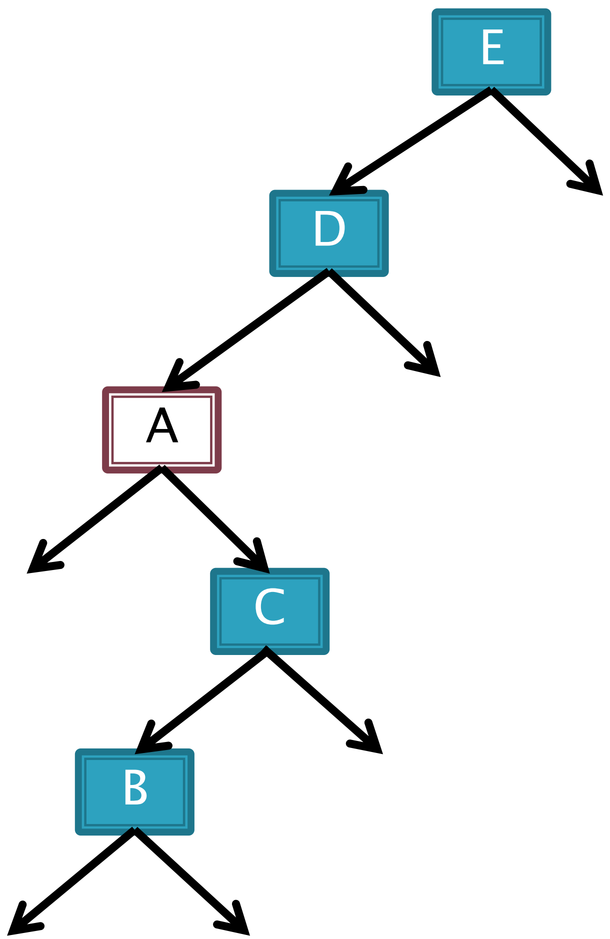
小課題3 (60点) – Splay木

- 実際ダメ



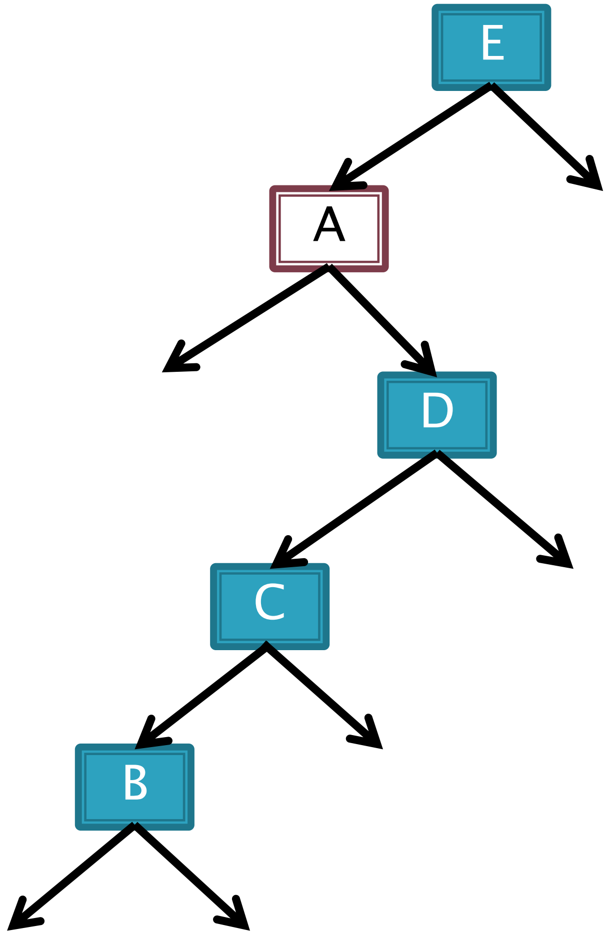
小課題3 (60点) – Splay木

- 実際ダメ



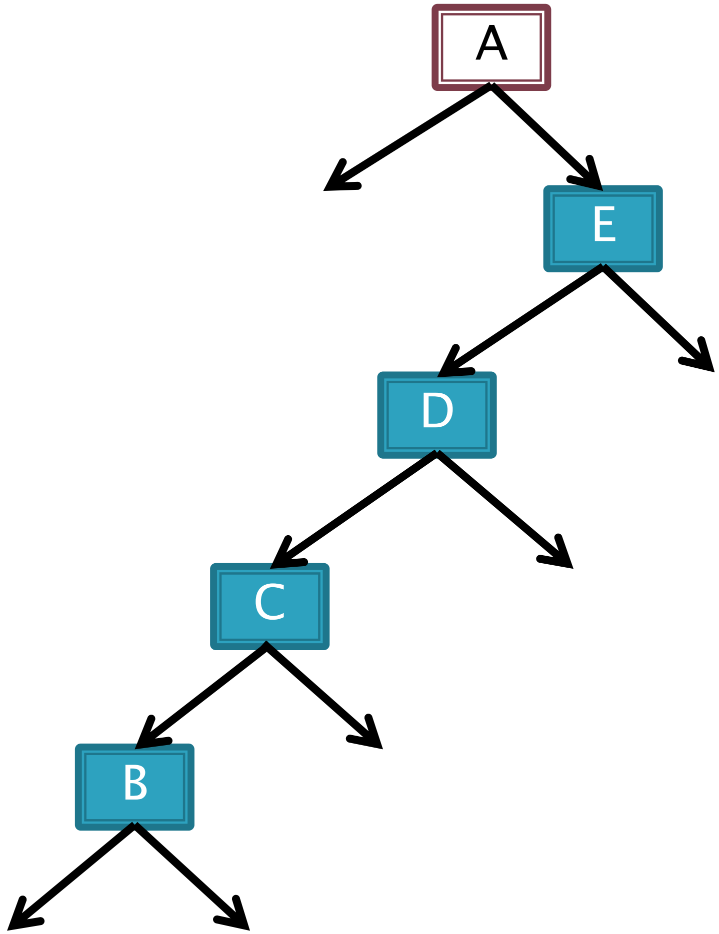
小課題3 (60点) – Splay木

- 実際ダメ



小課題3 (60点) – Splay木

- 実際ダメ



小課題3 (60点) – Splay木

- 実際ダメ
- この後A, B, C, D, Eの順にアクセスしたら
$$n + (n-1) + \ \dots + 1 = O(n^2)$$
- のコストがかかってしまう

小課題3 (60点) – Splay木

- 実際ダメ
- この後A, B, C, D, Eの順にアクセスしたら
  $$n + (n-1) + \ \ldots + 1 = O(n^2)$$
- のコストがかかってしまう

- どうする？

小課題3 (60点) – Splay木

- 解決策: 木の回転を3つに分ける

小課題3 (60点) – Splay木

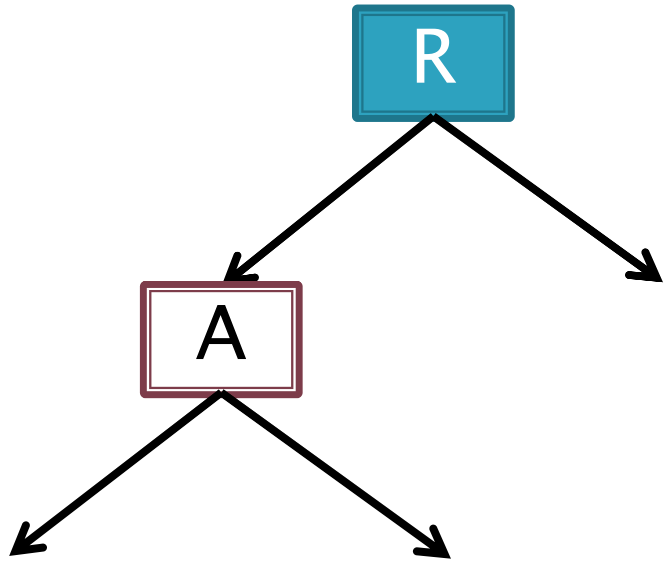
- 解決策: 木の回転を3つに分ける
  - “zig” step
  - “zig-zag” step
  - “zig-zig” step

小課題3 (60点) – Splay木

- (1) “zig”-step

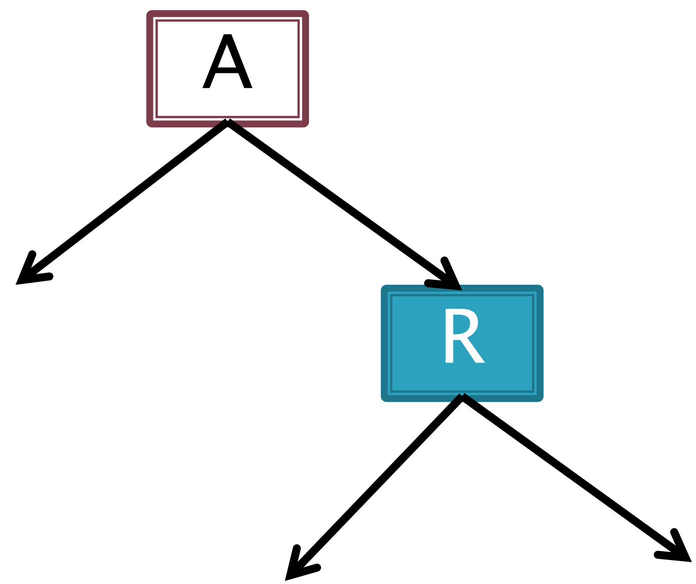
小課題3 (60点) – Splay木

- (1) "zig"-step
- すぐ上が根の場合



小課題3 (60点) – Splay木

- (1) “zig”-step
- すぐ上が根の場合
- 普通に回転する

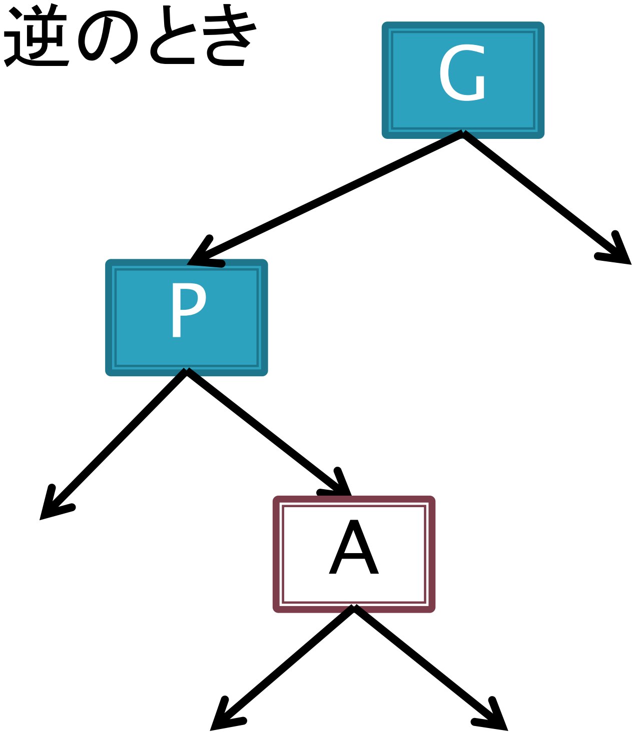


小課題3 (60点) – Splay木

- (2) “zig-zag”-step

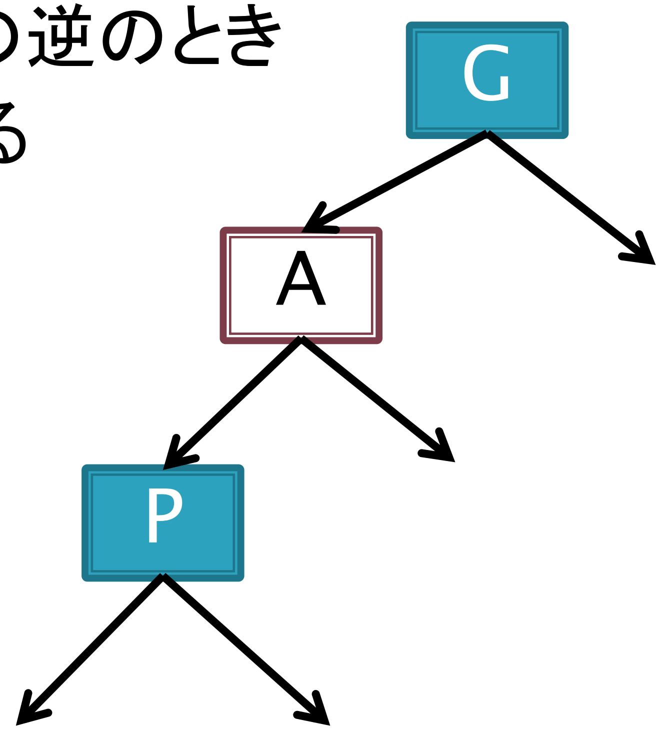
小課題3 (60点) – Splay木

- (2) “zig-zag”-step
- 左→右、またはその逆のとき



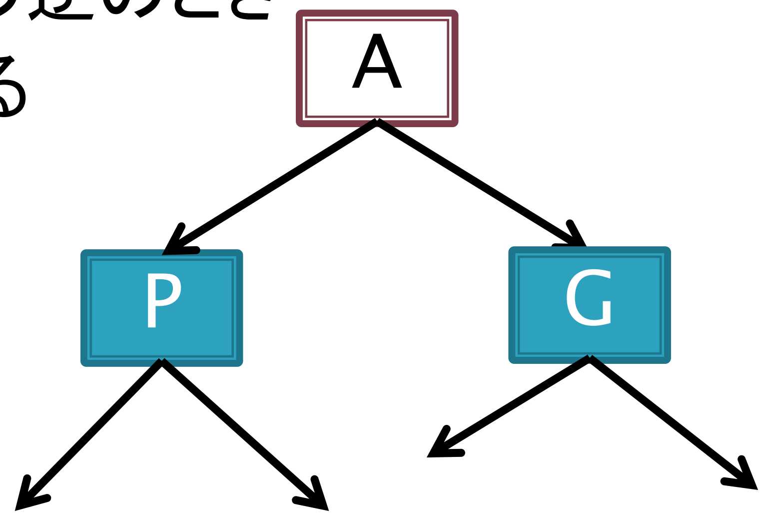
小課題3 (60点) – Splay木

- (2) “zig-zag”-step
- 左→右、またはその逆のとき
- 普通に2回回転する



小課題3 (60点) – Splay木

- (2) “zig-zag”-step
- 左→右、またはその逆のとき
- 普通に2回回転する



小課題3 (60点) – Splay木

- (2) “zig-zag”-step
- 左→右、またはその逆のとき
- 普通に2回回転する

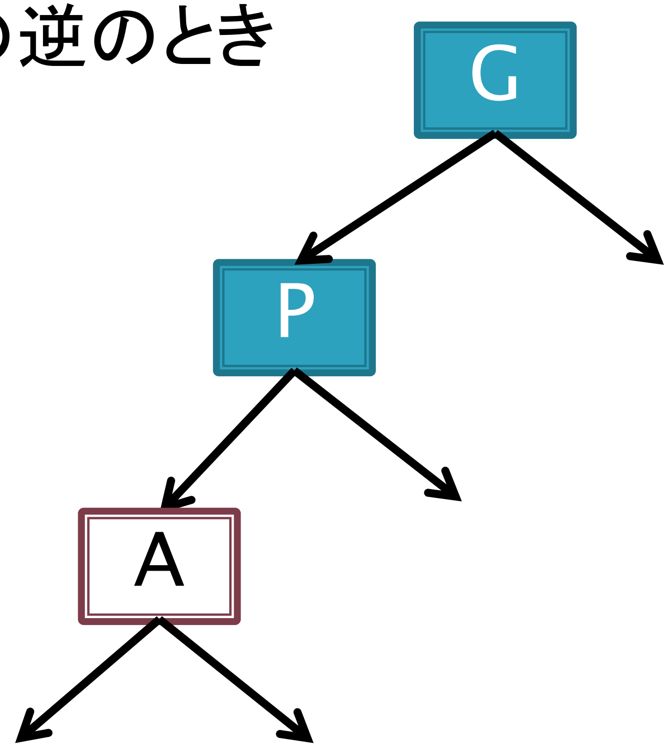
- ここまでは先ほどと同じ

小課題3 (60点) – Splay木

- (3) “zig-zig”-step

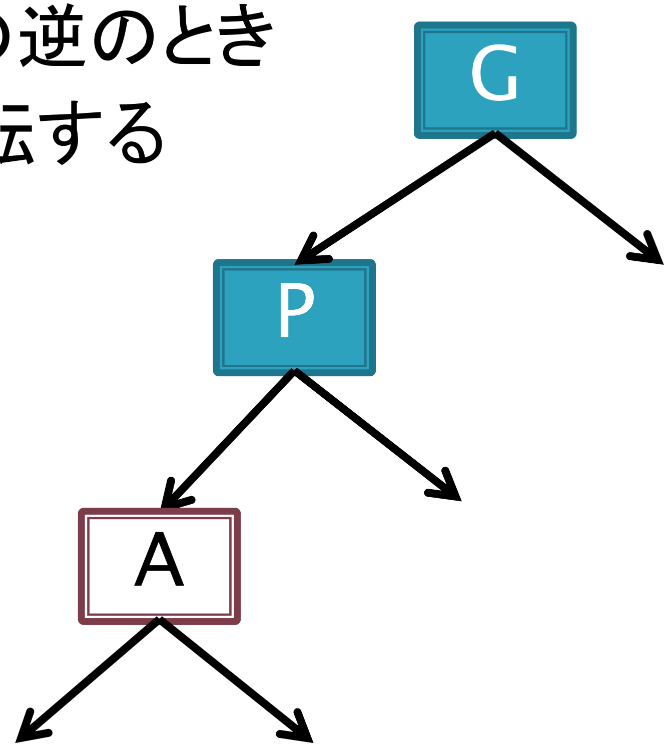
小課題3 (60点) – Splay木

- (3) “zig-zig”-step
- 左→左、またはその逆のとき



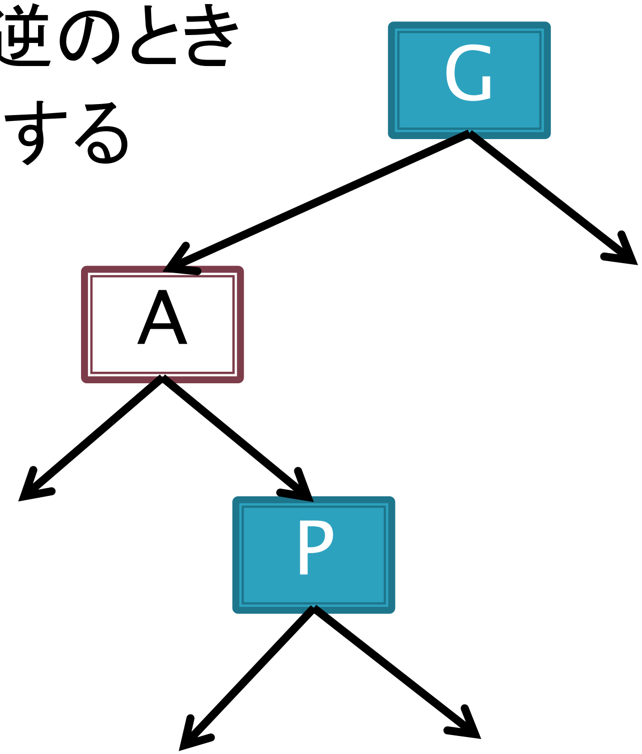
小課題3 (60点) – Splay木

- (3) “zig-zig”-step
- 左→左、またはその逆のとき
- 2回ではなく3回回転する



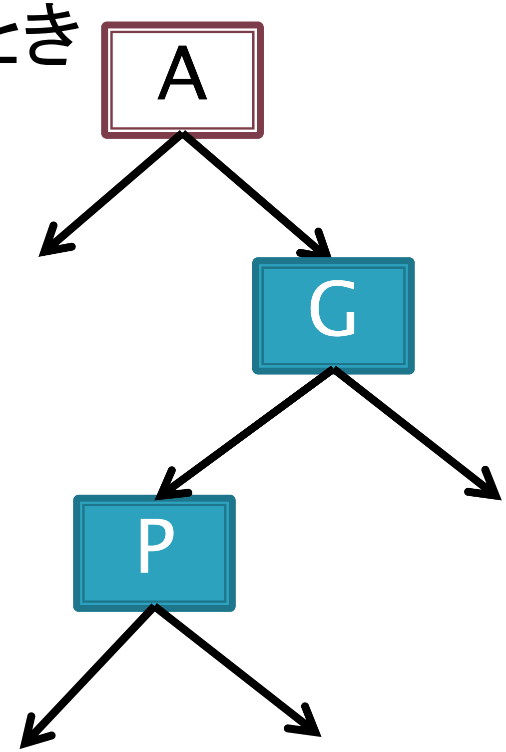
小課題3 (60点) – Splay木

- (3) “zig-zig”-step
- 左→左、またはその逆のとき
- 2回ではなく3回回転する



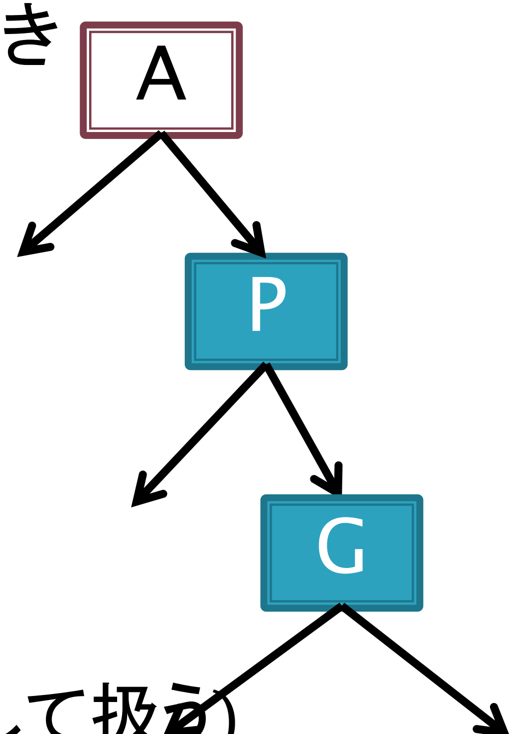
小課題3 (60点) – Splay木

- (3) “zig-zig”-step
- 左→左、またはその逆のとき
- 2回ではなく3回回転する



小課題3 (60点) – Splay木

- (3) “zig-zig”-step
- 左→左、またはその逆のとき
- 2回ではなく3回回転する



- (ただし、後でこれを2回として扱う)

小課題3 (60点) – Splay木

- Splaying operation
  - 偏った位置にあるときだけ余計に回転する

小課題3 (60点) – Splay木

- Splaying operation
  - 偏った位置にあるときだけ余計に回転する
  - そんなので上手くいくわけないだろ！！

小課題3 (60点) – Splay木

- 実は上手くいく

小課題3 (60点) – Splay木

- 実は上手くいく
- 具体的には: $O(\log N)$ amortized

小課題3 (60点) – Splay木

- 実は上手くいく
- 具体的には: $O(\log N)$ amortized

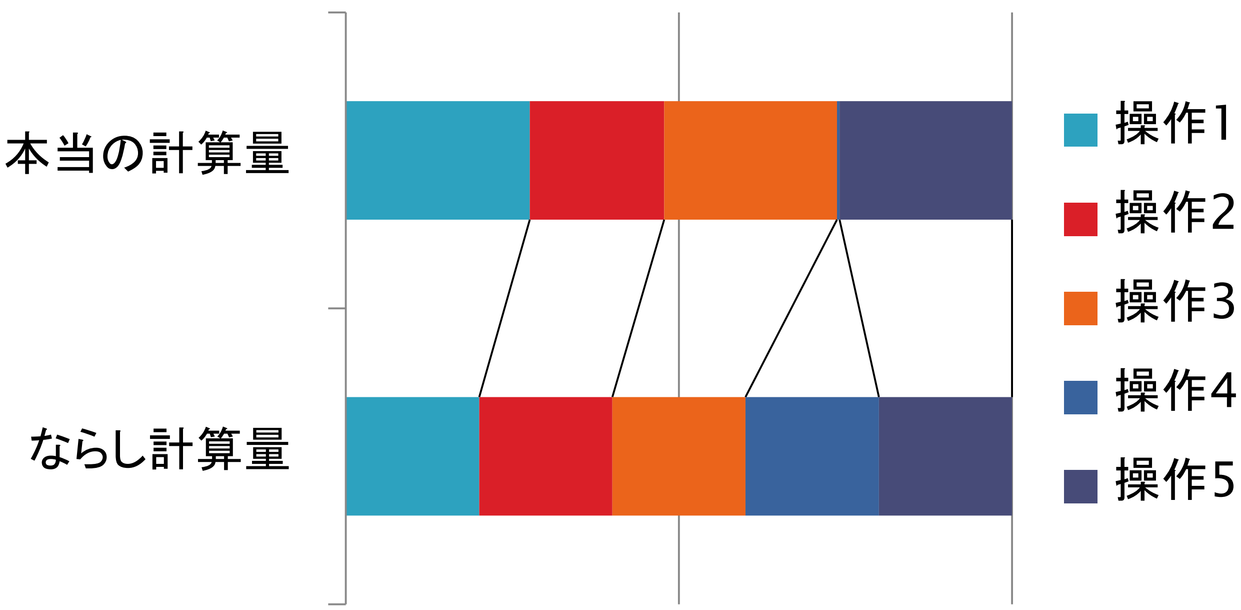
小課題3 (60点) – ならし計算量

- ならし計算量 (amortized time complexity)

- N個の一連の操作が$O(f(N))$で行えるとする
- 1つ1つの操作は、本当は$O(\frac{f(N)}{N})$とは限らない
- これを$O(\frac{f(N)}{N})$として扱うのが、ならし計算量

小課題3 (60点) – ならし計算量

- ならし計算量のイメージ



小課題3 (60点) – ならし計算量

- ならし計算量の向き/不向き

小課題3 (60点) – ならし計算量

- ならし計算量の向き/不向き
- 向いているもの
  - 全体での処理効率が重視されるバッチ型の処理
  - 例: プログラミングコンテスト
- 向いていないもの
  - リアルタイム性能が重視される処理
  - 例: 信号処理

小課題3 (60点) – ならし計算量

- ならし計算量と平均計算量

小課題3 (60点) – ならし計算量

- ならし計算量と平均計算量
  - この2つは別物！！
  - ならし計算量 : 時系列上での平均
  - 平均計算量 : 確率変数上での平均

小課題3 (60点) – ならし計算量

- ならし計算量と平均計算量
  - この2つは別物！！
  - ならし計算量 : 時系列上での平均
  - 平均計算量 : 確率変数上での平均

  - ならし計算量 : 不確定要素は無い！

小課題3 (60点) – Splay木の解析

- Splayのならし計算量の評価

小課題3 (60点) – Splay木の解析

- Splayのならし計算量の評価
- 「ポテンシャル関数」の概念を導入

小課題3 (60点) – Splay木の解析

- Splayのならし計算量の評価
- 「ポテンシャル関数」の概念を導入
  - 借金みたいなもの

小課題3 (60点) – Splay木の解析

- ポテンシャル関数を用いた計算量の均し(ならし)
- $a_j = t_j + \Phi_{j+1} - \Phi_{\mathrm{j}}$
  - t_j : その操作の実際の計算量
  - $\Phi_{j+1} - \Phi_{\mathrm{j}}$: ポテンシャルの増加量
  - a_j : その操作のならし計算量

小課題3 (60点) – Splay木の解析

- ポテンシャル関数を用いた計算量の均し(ならし)
- $a_j = t_j + \Phi_{j+1} - \Phi_{\mathrm{j}}$
  - t_j : その操作の実際の計算量
  - $\Phi_{j+1} - \Phi_{\mathrm{j}}$: ポテンシャルの増加量
  - a_j : その操作のならし計算量
- ポテンシャルの意味
  - より大きい: 木はより偏っている
  - より小さい: 木はより平坦になっている

小課題3 (60点) – Splay木の解析

- ならし計算量の総和をとる
- $\sum_j a_j = \sum_j t_j + \Phi_m - \Phi_0$
  - $\sum_j t_j$: 実際の計算量の総和
  - $\Phi_m - \Phi_0$: ポテンシャルの総変化量
  - $\sum_j a_j$: ならし計算量の総和

小課題3 (60点) – Splay木の解析

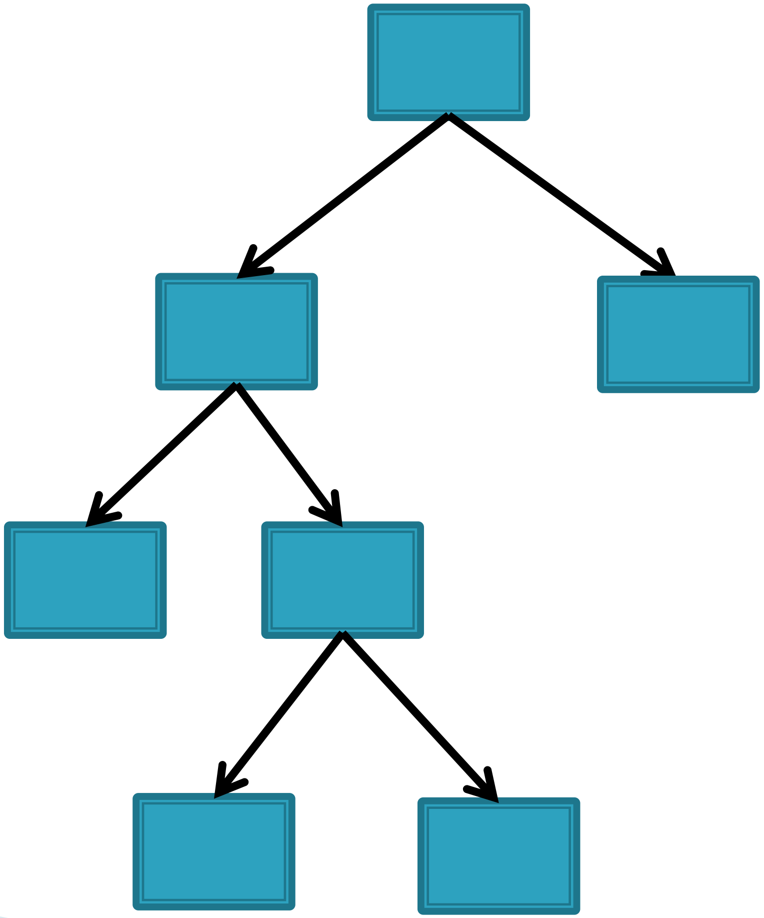
- ならし計算量の総和をとる
- $\sum_j a_j = \sum_j t_j + \Phi_m - \Phi_0$
  - $\sum_j t_j$: 実際の計算量の総和
  - $\Phi_m - \Phi_0$: ポテンシャルの総変化量
  - $\sum_j a_j$: ならし計算量の総和
- ポテンシャルの総変化量が小さければうまく評価できる

小課題3 (60点) – Splay木の解析

- Splay木のポテンシャル
  - Splay木の各頂点の重さを$w(x)$とする
    - 計算量の見積もり方にあわせて自由に決めてよい
  - Splay木の頂点のサイズ $s(x) = \sum_{x\text{の全ての子孫}y} w(y)$
  - Splay木の頂点のランク $r(x) = \log_2 s(x)$
  - Splay木のポテンシャル $\Phi = \sum_{\text{全ての頂点}x} r(x)$

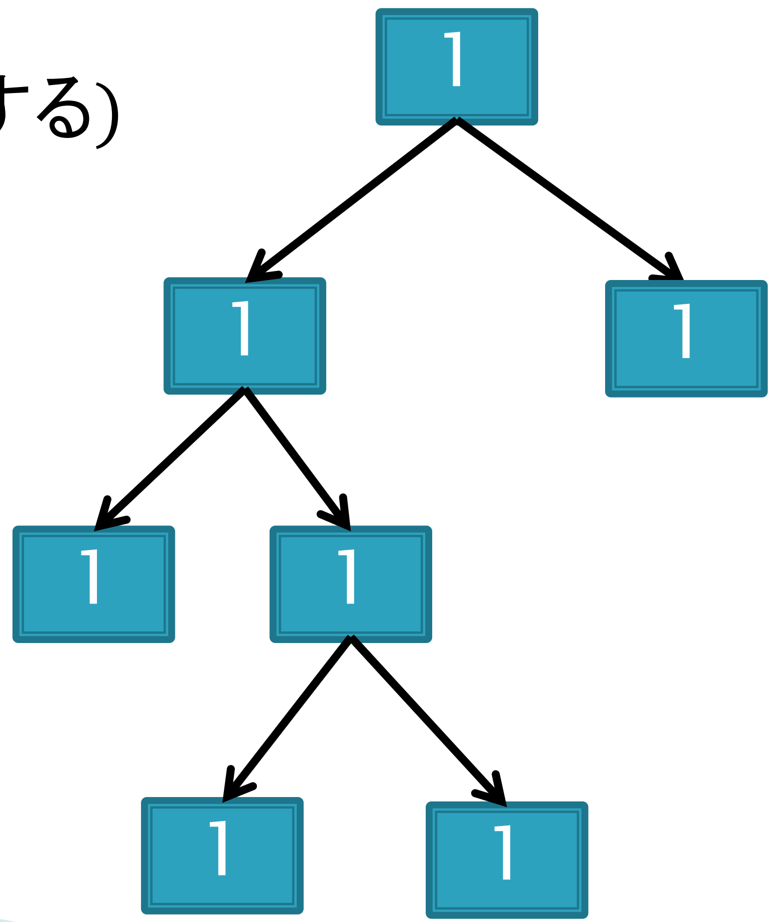
小課題3 (60点) – Splay木の解析

- Splay木のポテンシャル: 例



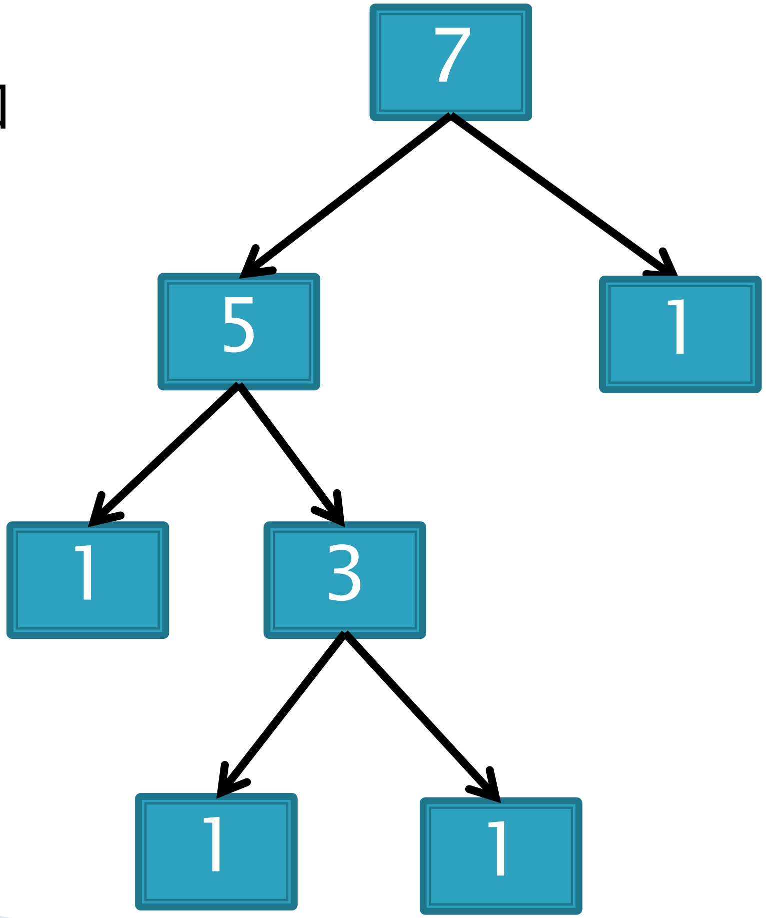
小課題3 (60点) – Splay木の解析

- Splay木のポテンシャル: 例
- 重さ $w(x)$
  (今回は全て1とする)



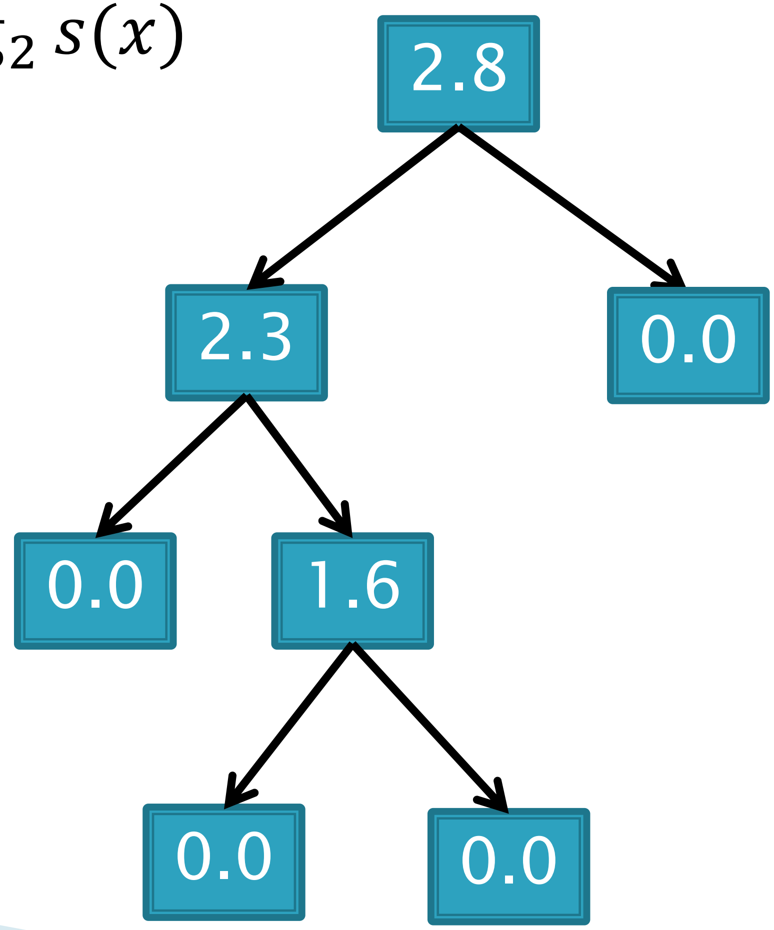
小課題3 (60点) – Splay木の解析

- Splay木のポテンシャル: 例
- サイズ $s(x)$
  - 部分木の重さの和



小課題3 (60点) – Splay木の解析

- Splay木のポテンシャル: 例
- ランク $r(x) = \log_2 s(x)$



小課題3 (60点) – Splay木の解析

- Splay木のポテンシャル: 例
- ポテンシャル: ランクの総和
-

$$\Phi = 2.8 + 2.3 + 1.6 = 6.7$$

小課題3 (60点) – Splay木の解析

- Splay木のポテンシャルの良い性質

小課題3 (60点) – Splay木の解析

- Splay木のポテンシャルの良い性質 …
- 回転の影響を受ける頂点が少ない
  - 解析が簡単になる

小課題3 (60点) – Splay木の解析

- アクセス補題 (Access Lemma)

小課題3 (60点) – Splay木の解析

- アクセス補題 (Access Lemma)

- x : 木のノード
- t : 木の根　とするとき

- 木をsplayする操作一回にかかる時間(回転の回数)は、ならし計算量で
$$3r(t) - 3r(x) + 1$$
- 以下である。

小課題3 (60点) – Splay木の解析

- アクセス補題の証明

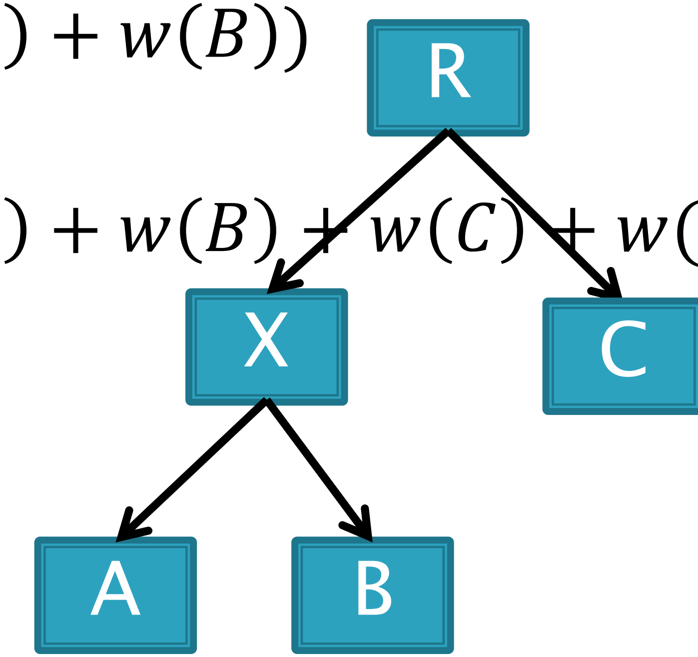
小課題3 (60点) – Splay木の解析

- アクセス補題の証明

- 各回転ステップのならし計算量が
  1. “zig”-stepでは $3r'(x) - 3r(x) + 1$ 以下
  2. それ以外では $3r'(x) - 3r(x)$ 以下
- (ただし、$r'(x)$: 操作後のランク)
- であることを示す。
- そうすると、1のケースに登場する$r'(x)$は初期の$r(t)$と等しい(木全体のサイズの対数)ので、合計すると$3r(t) - 3r(x) + 1$になる。

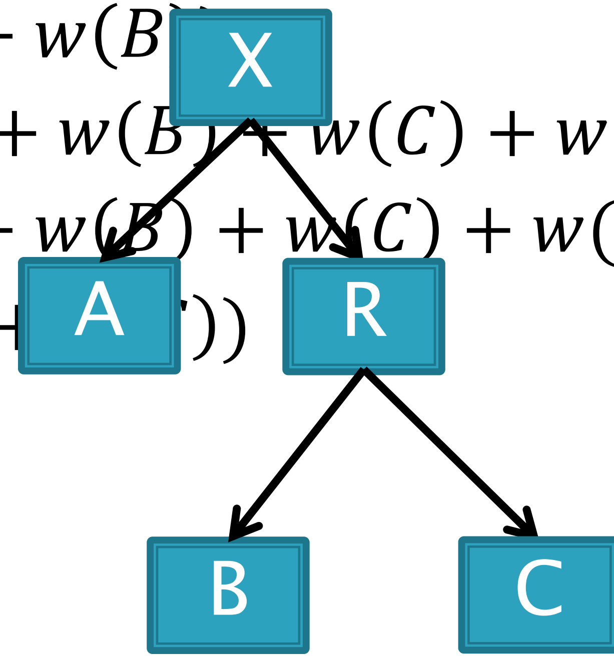
小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (1) “zig”-step の場合
- $r(X) = \log_2(w(X) + w(A) + w(B))$
- $r(R) = \log_2(w(X) + w(A) + w(B) + w(C) + w(R))$



小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (1) “zig”-step の場合
- $r(X) = \log_2(w(X) + w(A) + w(B))$
- $r'(X) = \log_2(w(X) + w(A) + w(B) + w(C) + w(R))$
- $r(R) = \log_2(w(X) + w(A) + w(B) + w(C) + w(R))$
- $r'(R) = \log_2(w(R) + w(B) + w(C))$



小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (1) “zig”-step の場合
- $r(X) = \log_2(w(X) + w(A) + w(B))$
- $r'(X) = \log_2(w(X) + w(A) + w(B) + w(C) + w(R))$
- $r(R) = \log_2(w(X) + w(A) + w(B) + w(C) + w(R))$
- $r'(R) = \log_2(w(R) + w(B) + w(C))$

小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (1) “zig”-step の場合
- $r(X) = \log_2(w(X) + w(A) + w(B))$
- $r'(X) = \log_2(w(X) + w(A) + w(B) + w(C) + w(R))$
- $r(R) = \log_2(w(X) + w(A) + w(B) + w(C) + w(R))$
- $r'(R) = \log_2(w(R) + w(B) + w(C))$

- $r(X) \leq r'(X), r'(R) \leq r(R)$

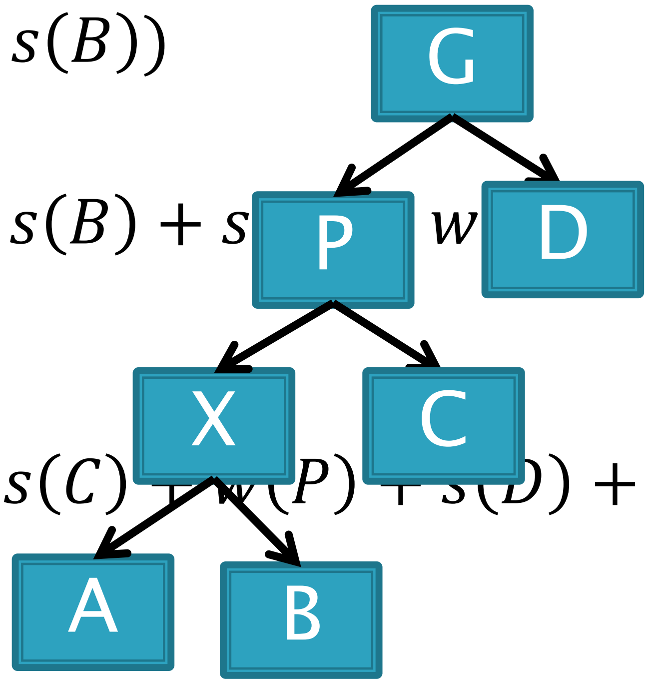
小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (1) “zig”-step の場合
- $r(X) \leq r'(X), r'(R) \leq r(R)$
- ならし計算量

$$
\begin{aligned}
a &= t + \Phi' - \Phi \\
&= 1 + r'(X) + r'(R) - r(X) - r(R) \\
&\leq 1 + r'(X) - r(X) \\
&\leq 1 + 3r'(X) - 3r(X)
\end{aligned}
$$

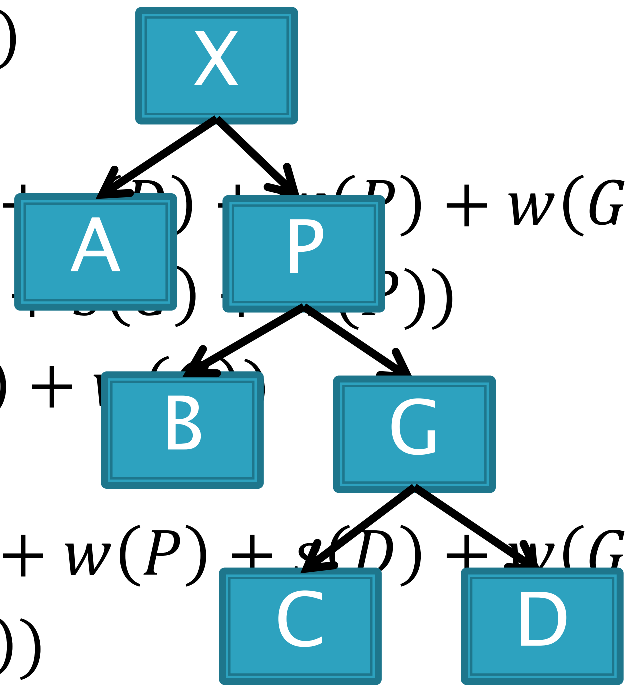
小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (2) “zigzig”-step の場合
- $r(X) = \log_2(w(X) + s(A) + s(B))$
- $r(P) = \log_2(w(X) + s(A) + s(B) + s$ w
- $r(G) =$ $\log_2(w(X) + s(A) + s(B) + s(C) + w(P) + s(D) + w(G))$



小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (2) “zigzig”-step の場合
- $r(X) = \log_2(w(X) + s(A) + s(B))$
- $r'(X) = \log_2(w(X) + s(A) + s(B) + s(C) + s(D) + w(P) + w(G))$
- $r(P) = \log_2(w(X) + s(A) + s(B) + s(C) + w(P))$
- $r'(P) = \log_2(s(B) + s(C) + w(P) + w(G))$
- $r(G) = \log_2(w(X) + s(A) + s(B) + s(C) + w(P) + s(D) + w(G))$
- $r'(G) = \log_2(s(C) + s(D) + w(G))$



小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (2) “zigzig”-step の場合
- $r(X) = \log_2(w(X) + s(A) + s(B))$
- $r'(X) = \log_2(w(X) + s(A) + s(B) + s(C) + s(D) + w(P) + w(G))$
- $r(P) = \log_2(w(X) + s(A) + s(B) + s(C) + w(P))$
- $r'(P) = \log_2(s(B) + s(C) + w(P) + w(G))$
- $r(G) = \log_2(w(X) + s(A) + s(B) + s(C) + w(P) + s(D) + w(G))$
- $r'(G) = \log_2(s(C) + s(D) + w(G))$

小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (2) “zigzig”-step の場合
- $r'(X) = r(G), r'(X) \leq r'(P), r(P) \leq r(X)$

小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (2) “zigzig”-step の場合
- $r'(X) = r(G), r'(P) \le r'(X), r(X) \le r(P)$

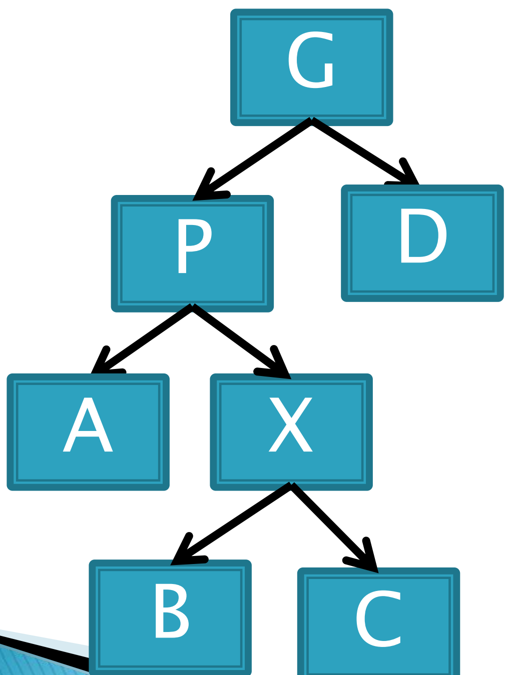
$$a = t + \Phi' - \Phi$$
$$= 2 + r'(X) + r'(P) + r'(G) - r(X) - r(P)$$
$$- r(G)$$
$$\le 2 + r'(X) + r'(G) - 2r(X)$$

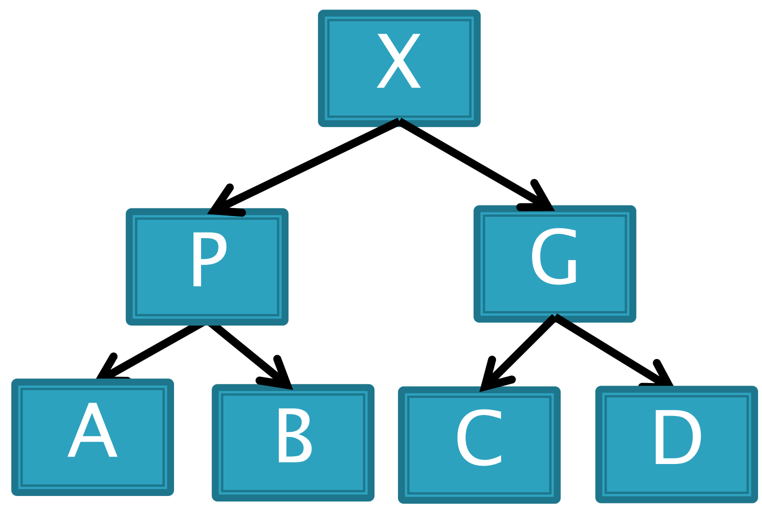
小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (2) “zigzig”-step の場合
- $2 + r'(X) + r'(G) - 2r(X) \leq 3r'(X) - 3r(X)$
- 理由: $r'(G) + r(X) - 2r'(X) \leq -2$ を示したい。
- ところで左辺は$\log_2\left(\frac{s'(G)}{s'(X)}\right) + \log_2\left(\frac{s(X)}{s'(X)}\right)$ であり、
- $\log_2 x$が上に凸で、$s'(G) + s(X) \leq s'(X)$なのでこの値は高々-2
- よって不等式は示された。

小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (3) “zigzag”-step の場合
- $r'(X) = r(G), r(X) \leq r(P)$
- $s'(P) + s'(G) \leq s'(X)$





小課題3 (60点) – Splay木の解析

- アクセス補題の証明 (3) “zigzag”-step の場合
- 以下(2)と同様

小課題3 (60点) – Splay木の解析

- 以上より、Splay操作がならし計算量で$O(\log N)$であることがわかった。
- ところで、Splay操作のポテンシャルは高々$O(N \log N)$なので、全体で$O((Q+N)\log N)$でクエリを処理できることがわかった。

小課題3 (60点) – Splay木の解析

- 以上より、Splay操作がならし計算量で$O(\log N)$であることがわかった。
- ところで、Splay操作のポテンシャルは高々$O(N \log N)$なので、全体で$O((Q+N)\log N)$でクエリを処理できることがわかった。

- 以上、満点解法その1

小課題3 (60点) – L/C木

- 満点解法その2 - Link/Cut木

小課題3 (60点) – L/C木

- 満点解法その2 - Link/Cut木

- Link/Cut木
  - 元々、フローアルゴリズムの高速化のためにSleatorとTarjanが考案したもの

小課題3 (60点) – L/C木

- 満点解法その2 - Link/Cut木

- Link/Cut木
  - 元々、フローアルゴリズムの高速化のためにSleatorとTarjanが考案したもの
  - この問題のために必要な実装は、それよりもはるかに容易

小課題3 (60点) – L/C木

- 満点解法その2 - Link/Cut木

- Link/Cut木
  - 元々、フローアルゴリズムの高速化のためにSleatorとTarjanが考案したもの
  - この問題のために必要な実装は、それよりもはるかに容易→Link/Cut木の練習としても適している

小課題3 (60点) – L/C木

- 満点解法その2 - Link/Cut木

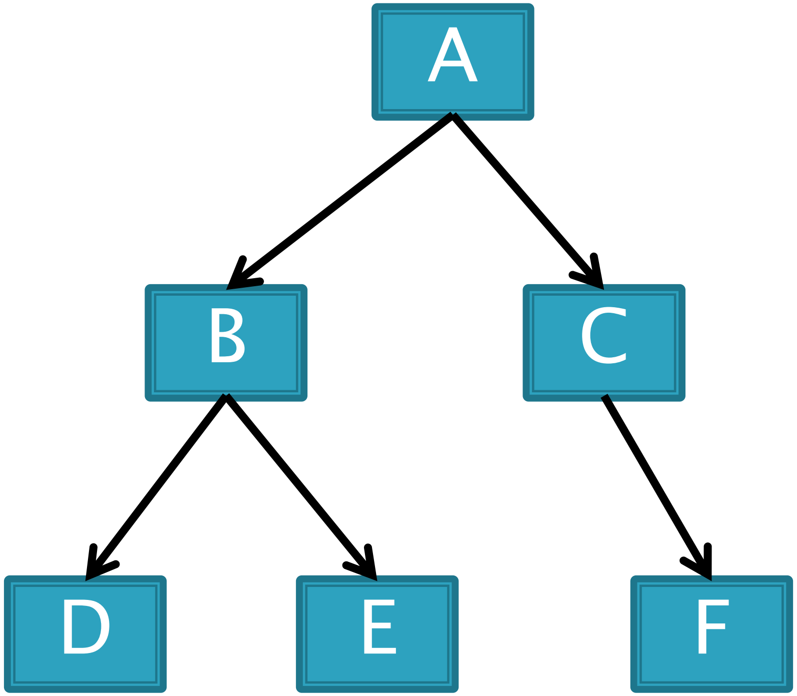
- Link/Cut木
  - 元々、フローアルゴリズムの高速化のためにSleatorとTarjanが考案したもの
  - この問題のために必要な実装は、それよりもはるかに容易 →Link/Cut木の練習としても適している
  - いろいろなバージョンがあるが、Splay木によるものが使いやすい

小課題3 (60点) – H/L分解

- 予備知識 – Heavy/Light decomposition

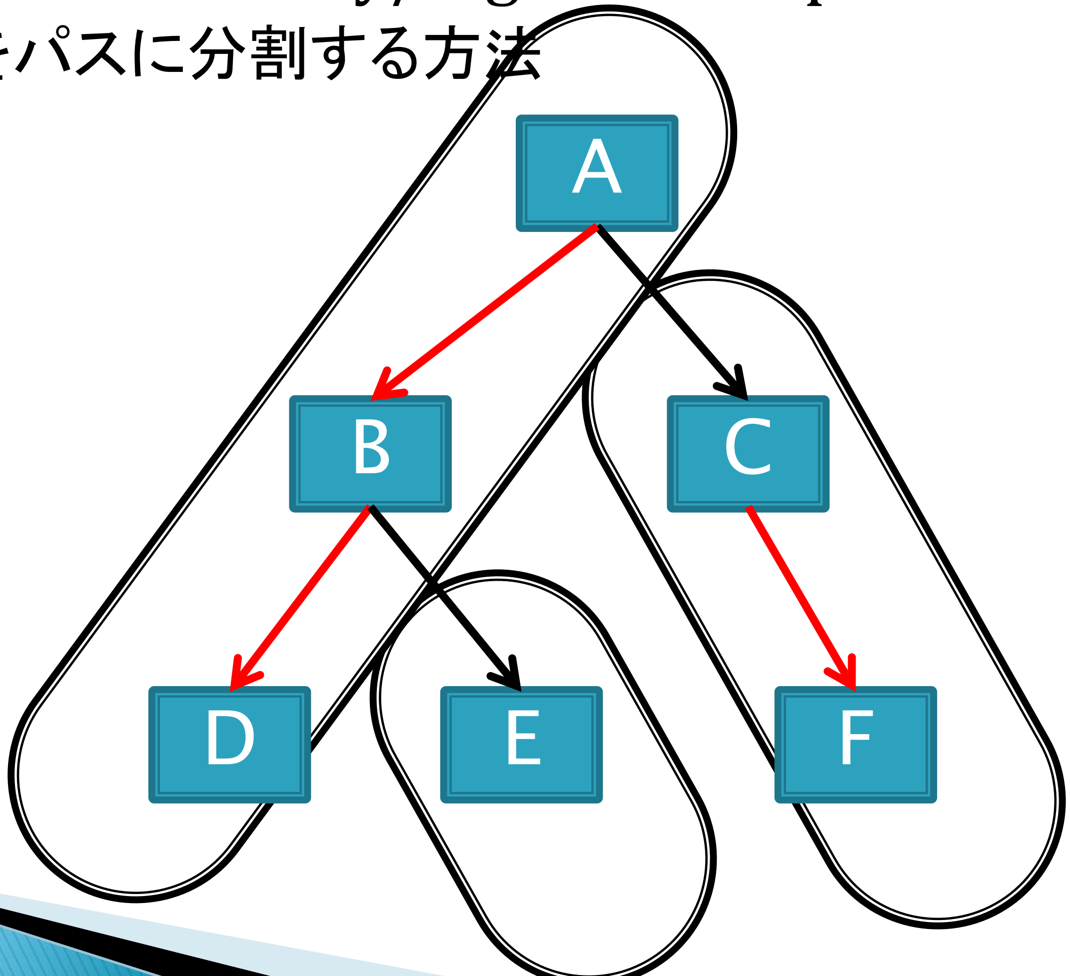
小課題3 (60点) – H/L分解

- 予備知識 – Heavy/Light decomposition
  - 木をパスに分割する方法



小課題3 (60点) – H/L分解

- 予備知識 – Heavy/Light decomposition
  - 木をパスに分割する方法



小課題3 (60点) – H/L分解

- 予備知識 – Heavy/Light decomposition
  - 木をパスに分割する方法

- 変な形の木でも、「パスの木」の形に潰すと安定する

小課題3 (60点) – L/C木

- Splay Treeの世界

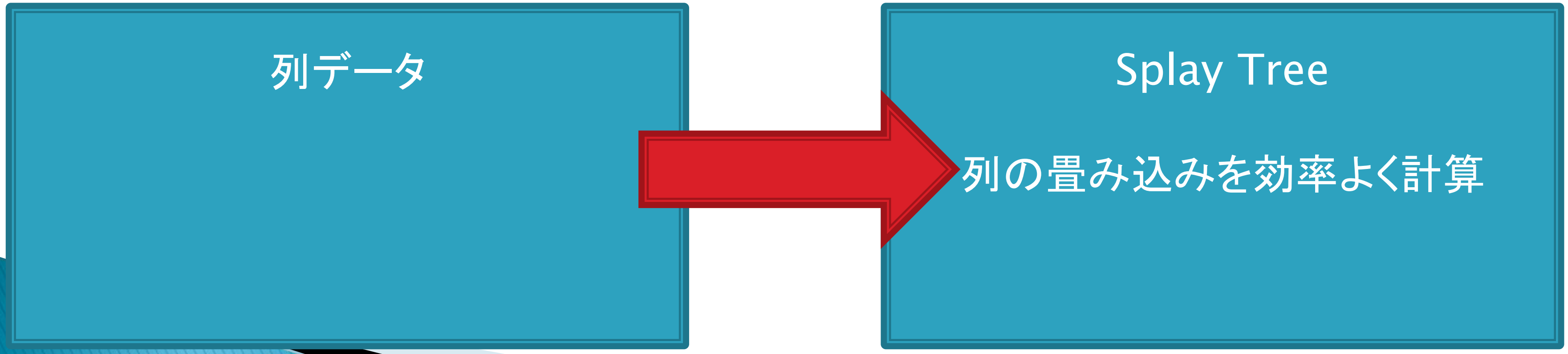
列データ

Splay Tree

列の畳み込みを効率よく計算

小課題3 (60点) – L/C木

- Splay Treeの世界



小課題3 (60点) – L/C木

- Heavy/Light decompositionの世界

有向木

列データ

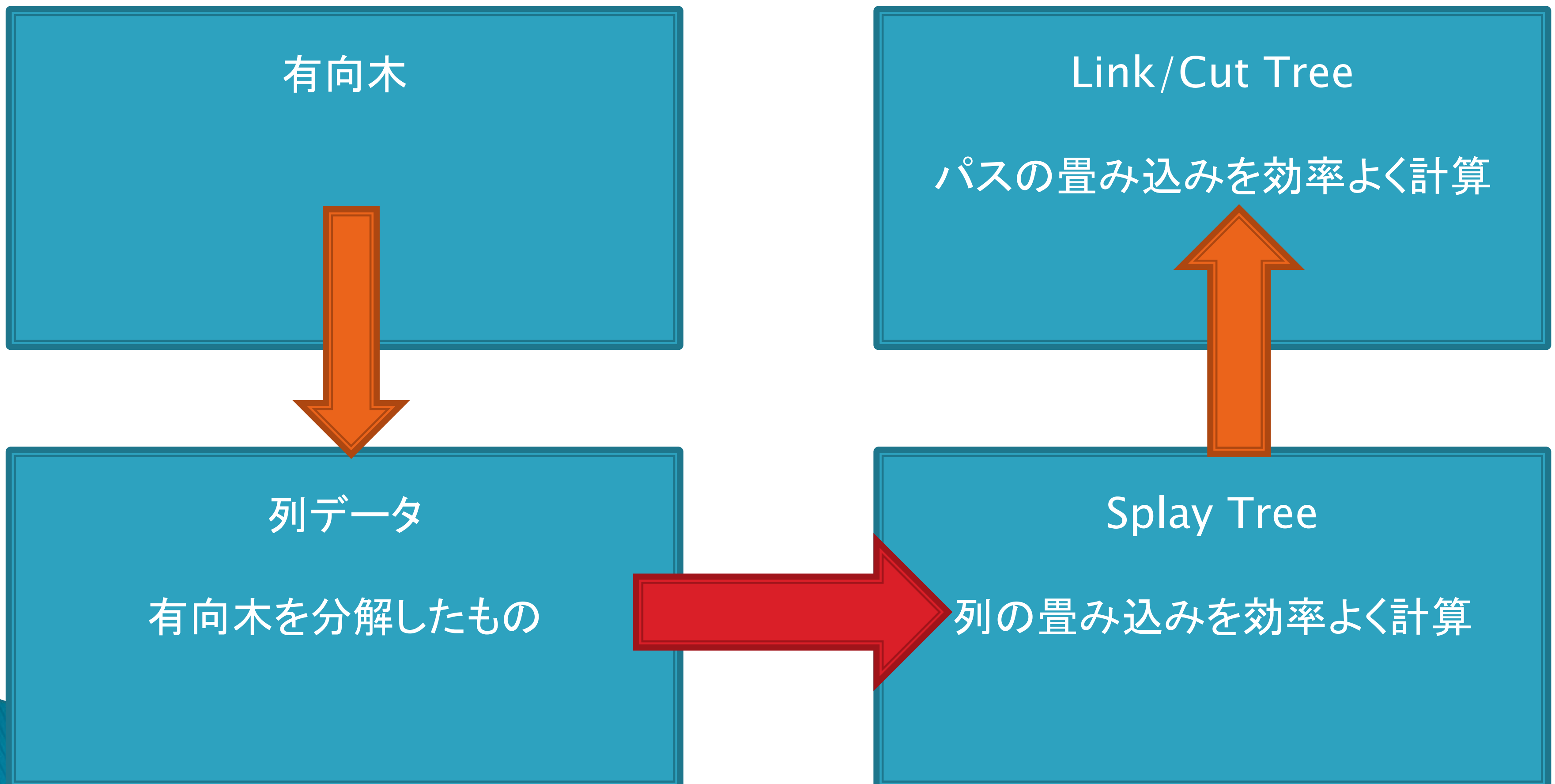
有向木を分解したもの

Splay Tree

列の畳み込みを効率よく計算

小課題3 (60点) – L/C木

- Link/Cut Treeの世界



小課題3 (60点) – L/C木

- Link/Cut Treeの世界
- H/L分解 = パスからなる木

小課題3 (60点) – L/C木

- Link/Cut Treeの世界

- H/L分解 = パスからなる木
- Link/Cut Tree = Splay木からなる木

小課題3 (60点) – L/C木の実装

- Link/Cut Tree の辺は二種類ある
  - Solid(Heavy) edge
  - Dashed(Light) edge

小課題3 (60点) – L/C木の実装

- Link/Cut Tree の辺は二種類ある

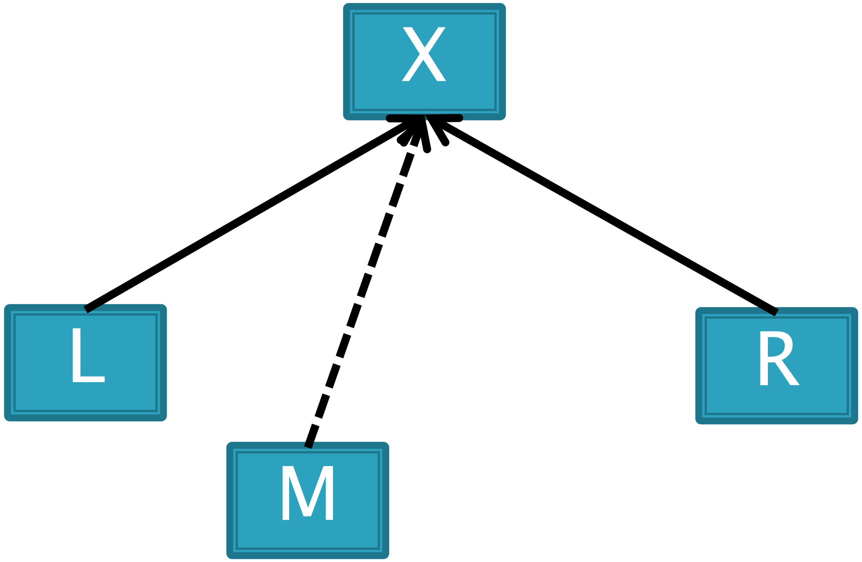
	Solid(Heavy)	Dashed(Light)
所属	Splay Tree	H/L分解の木
分類	二分木	多分木
左右の区別	左右の区別あり	なし
親	本当は祖先か子孫	本当の親
子供	本当は祖先か子孫	本当は子孫

小課題3 (60点) – L/C木の実装

- Solid, Dashedの区別

小課題3 (60点) – L/C木の実装

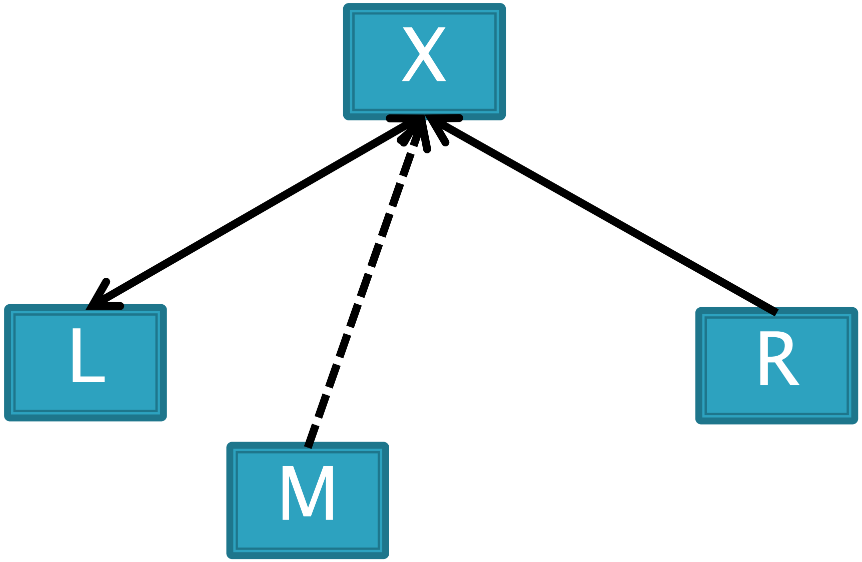
- Solid, Dashedの区別
- いずれも、親方向リンクを持つ



小課題3 (60点) – L/C木の実装

- Solid, Dashedの区別

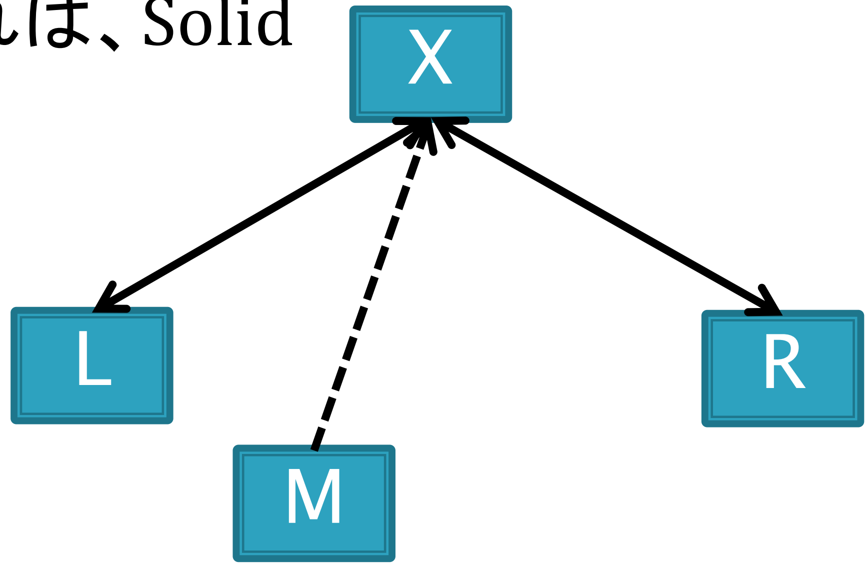
- いずれも、親方向リンクを持つ
- 親から左方向リンクがあれば、Solid



小課題3 (60点) – L/C木の実装

- Solid, Dashedの区別

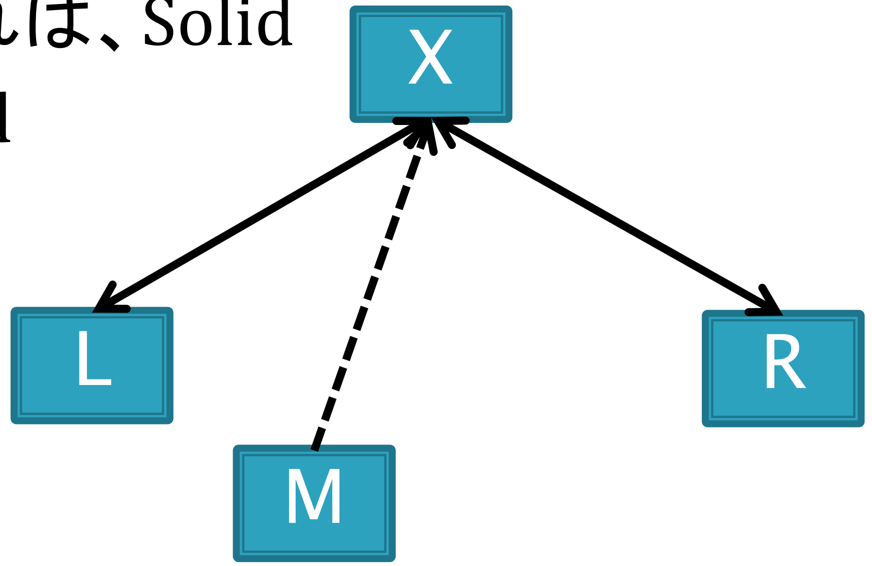
- いずれも、親方向リンクを持つ
- 親から左方向リンクがあれば、Solid
- 親から右方向リンクがあれば、Solid



小課題3 (60点) – L/C木の実装

- Solid, Dashedの区別

- いずれも、親方向リンクを持つ
- 親から左方向リンクがあれば、Solid
- 親から右方向リンクがあれば、Solid
- どちらもなければ、Dashed



小課題3 (60点) – L/C木の実装

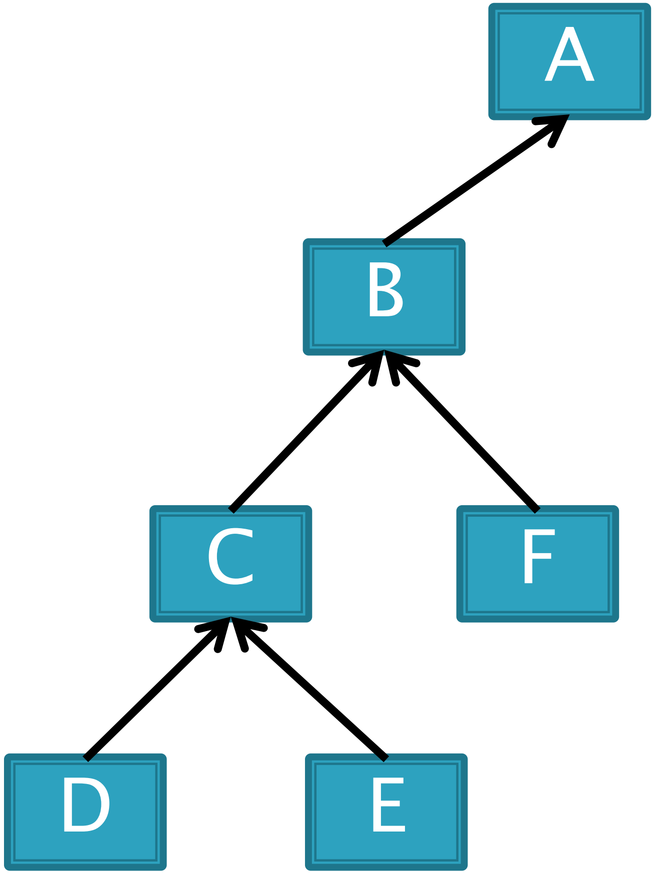
- Solid, Dashedの区別

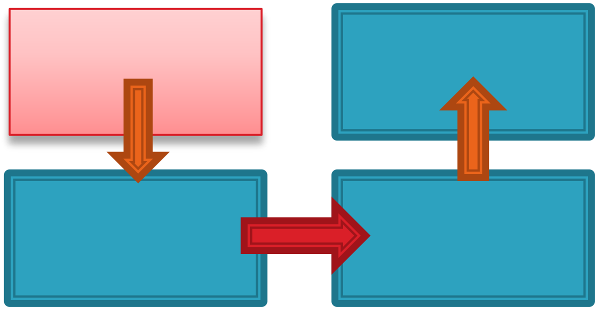
- いずれも、親方向リンクを持つ
- 親から左方向リンクがあれば、Solid
- 親から右方向リンクがあれば、Solid
- どちらもなければ、Dashed

- 「右, 左, 親」の3つのリンクだけで構造を保持できる！

小課題3 (60点) – L/C木の実装

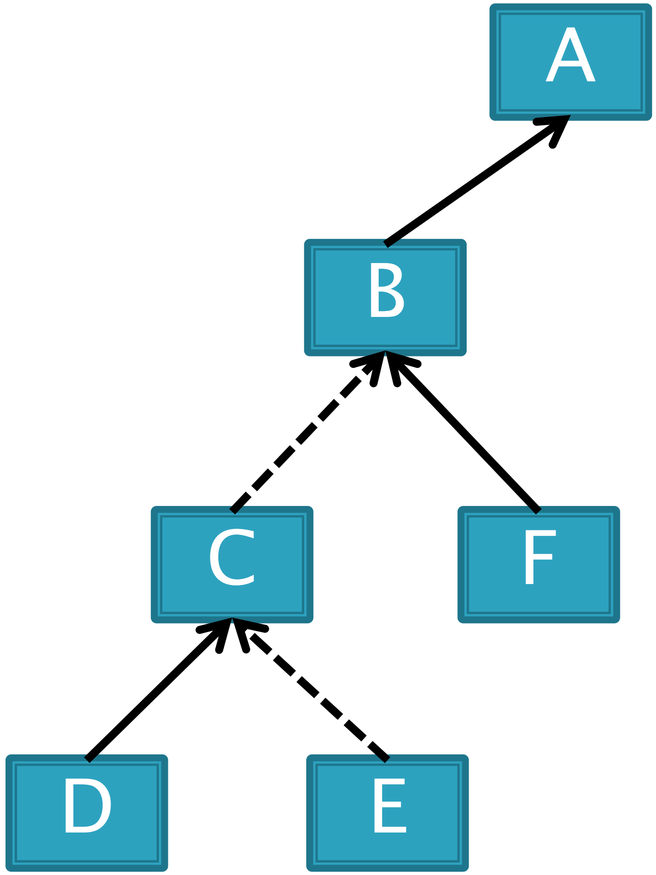
- 小さな例

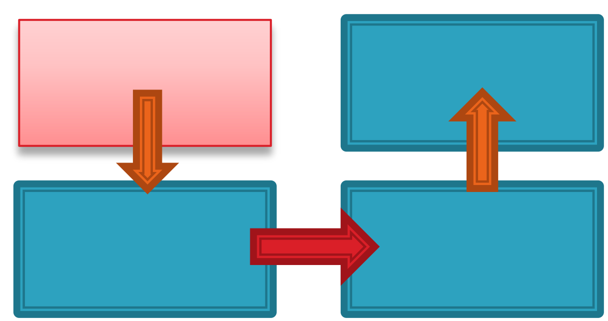




小課題3 (60点) – L/C木の実装

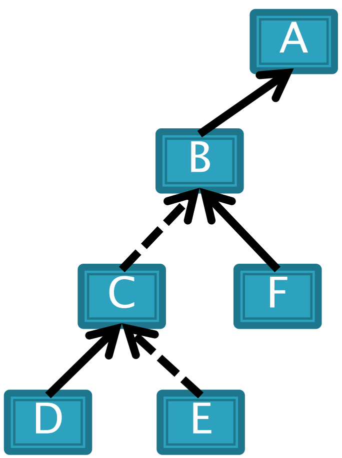
- 小さな例

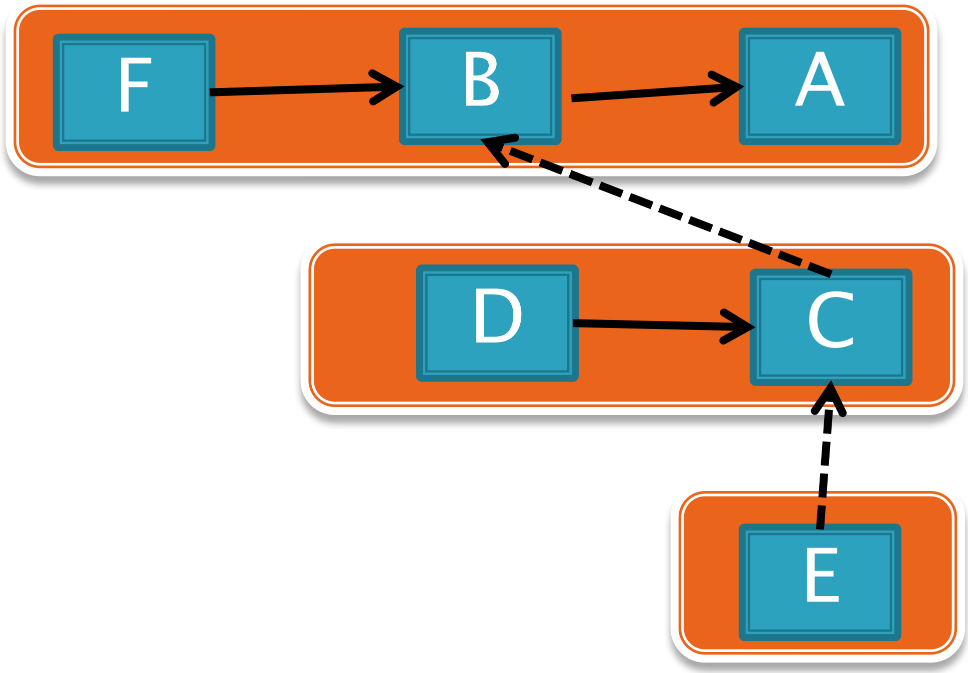


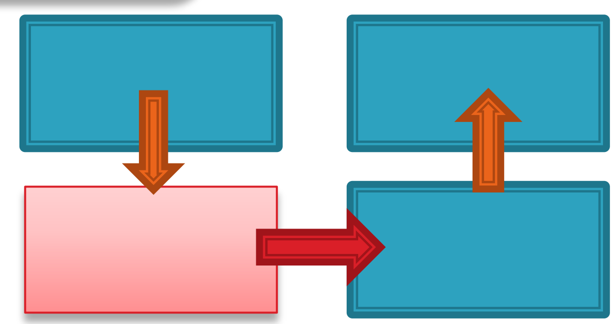


小課題3 (60点) – L/C木の実装

- 小さな例

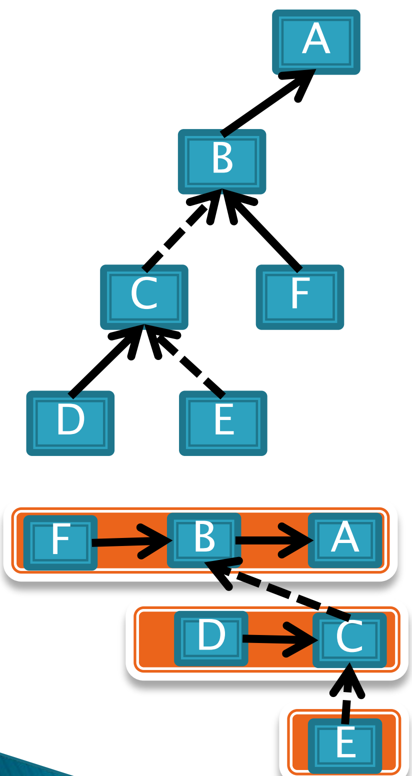


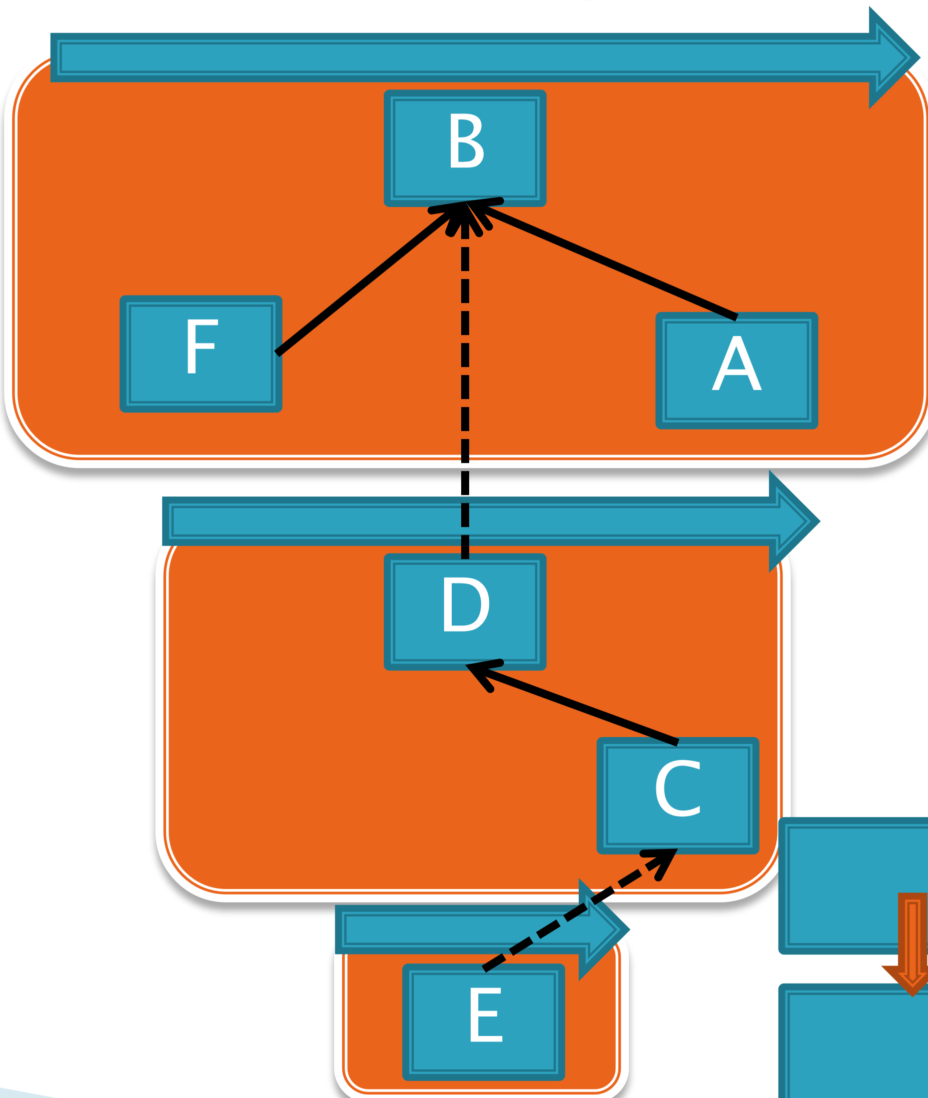


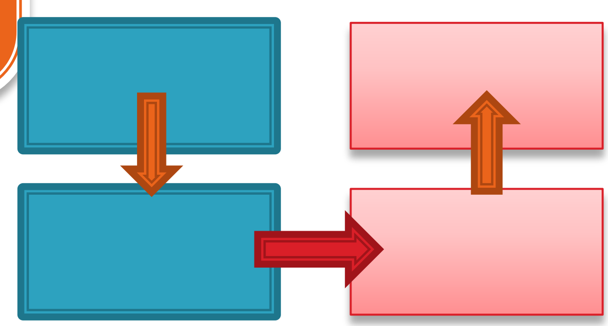


小課題3 (60点) – L/C木の実装

- 小さな例







小課題3 (60点) – L/C木の実装

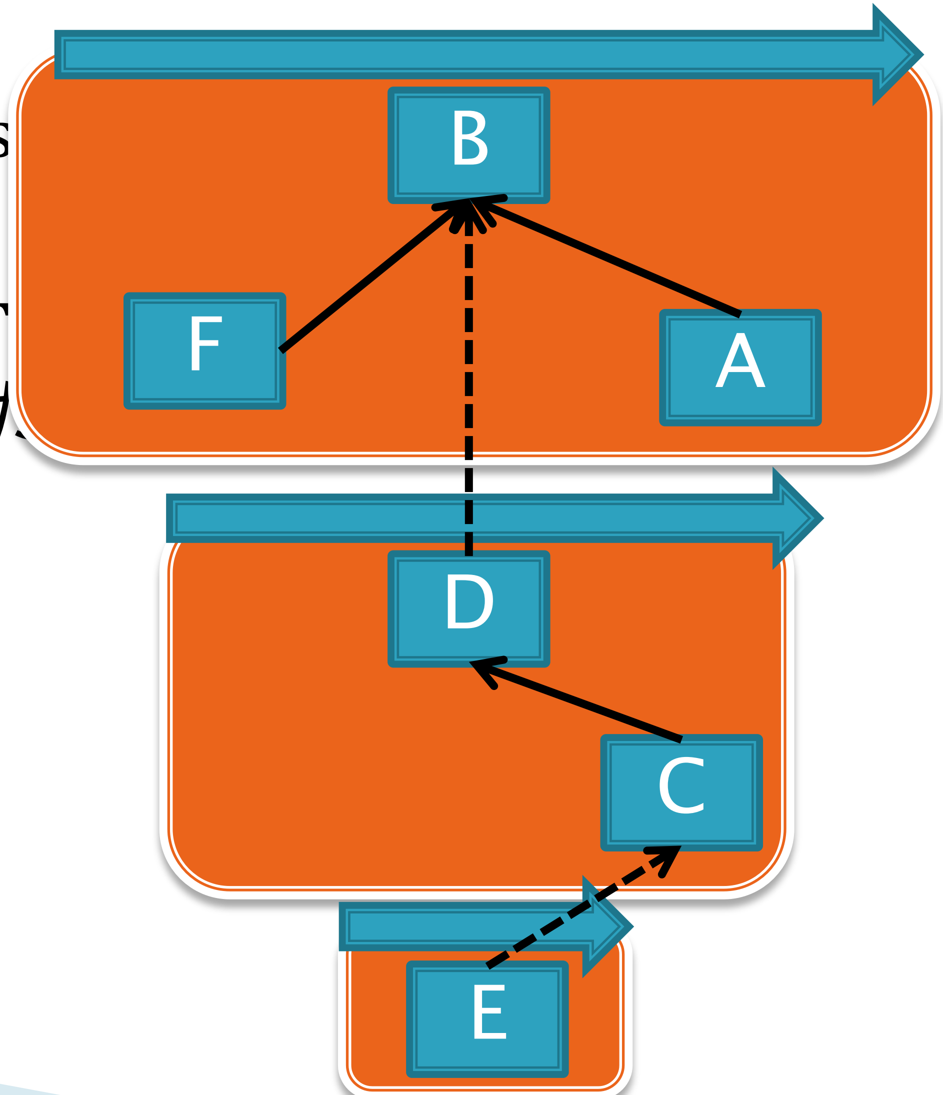
- Link/Cut Treeの操作: splayLC()
- Link/Cut Tree上では、任意の頂点Xを根に持っていくことができる
  - 元の木の構造は変化しないことに注意

小課題3 (60点) – L/C木の実装

- 前準備 : splay
- 各Splay Tree上でsplayをすることで、Xから今の根までを点線だけで行けるようにする

小課題3 (60点) – L/C木の実装

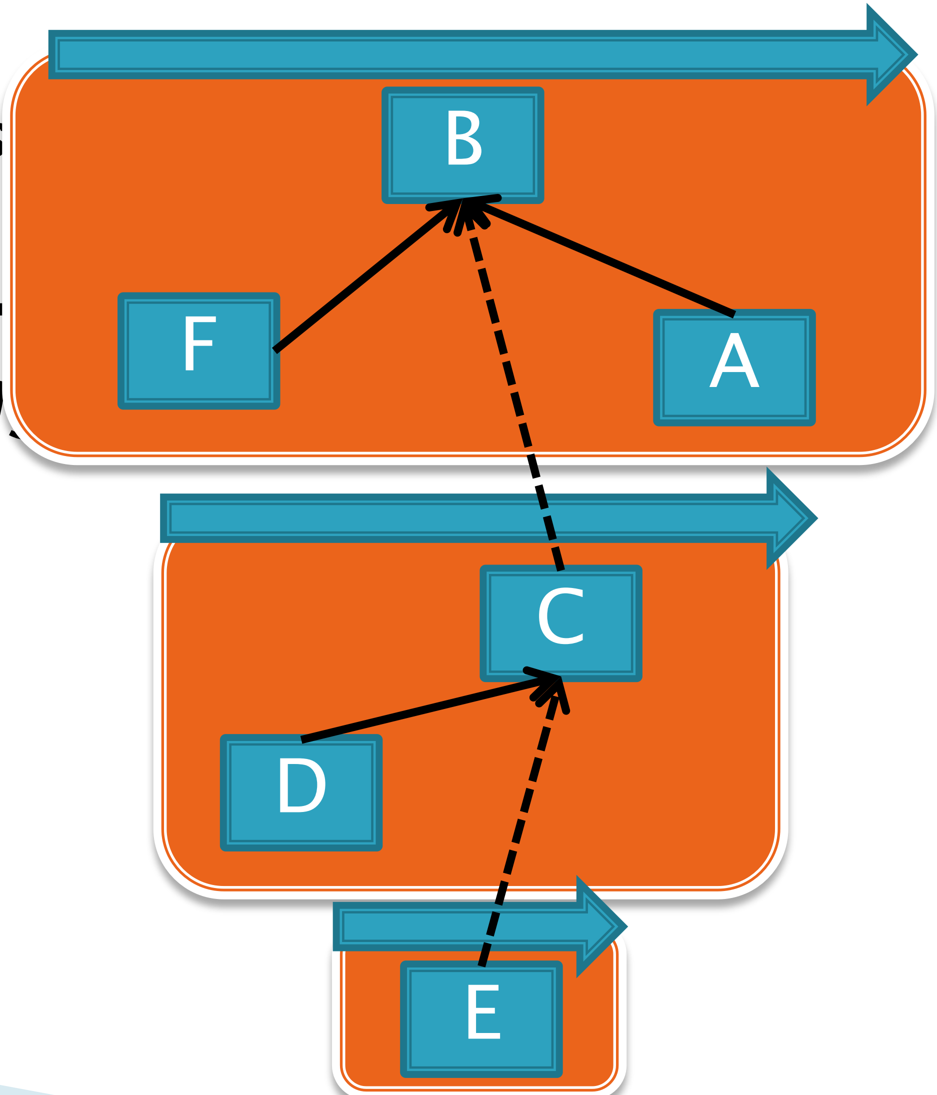
- 前準備：s
- 各Splay T　　　　　　　　　　　　　　　　　ら今の根までを点線た



小課題3 (60点) – L/C木の実装

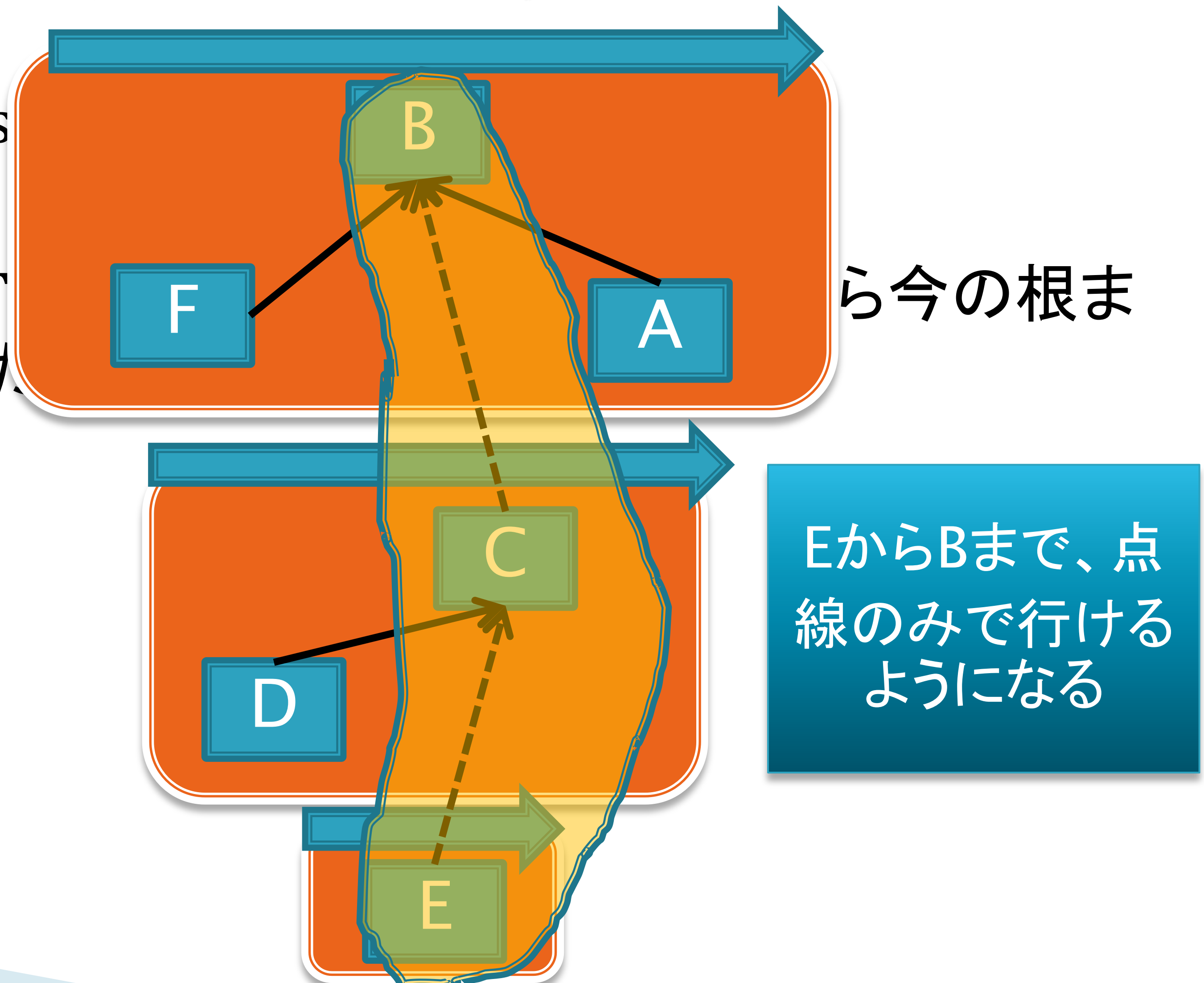
- 前準備: s

- 各Splay T　　　　　　　　　　　　　　　ら今の根までを点線た



小課題3 (60点) – L/C木の実装

- 前準備 : s
- 各Splay T　　ら今の根ま
でを点線た

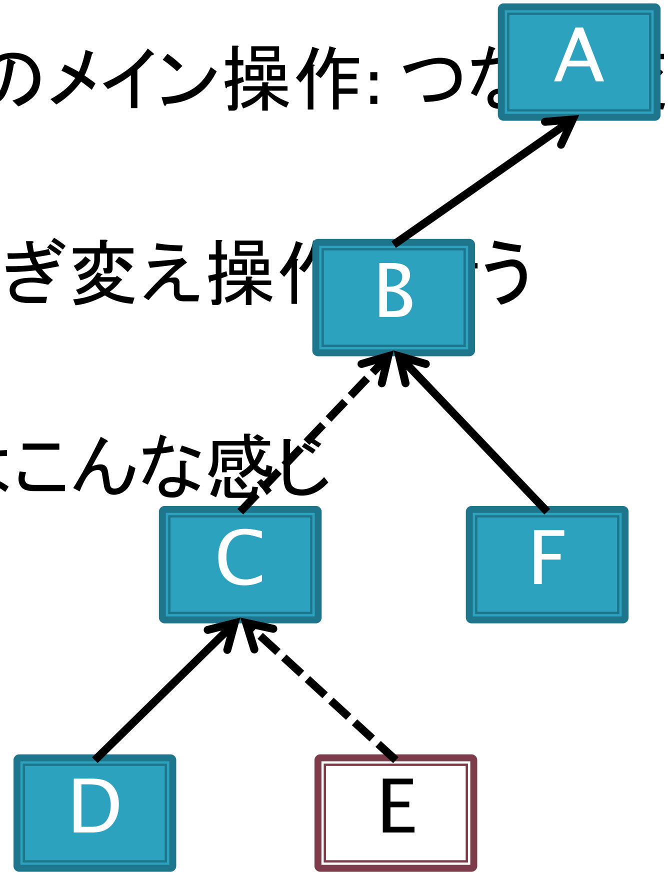


小課題3 (60点) – L/C木の実装

- splayLC() のメイン操作: つなぎ変え(expose操作)
- パスのつなぎ変え操作を行う
- 元の木ではこんな感じ

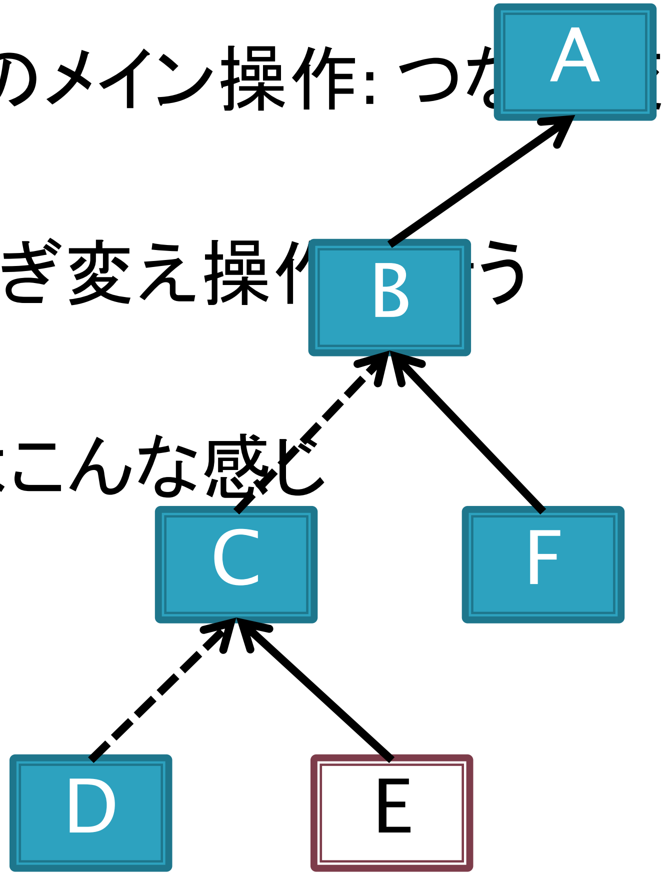
小課題3 (60点) – L/C木の実装

- splayLC() のメイン操作: つなぎ変え(expose操作)
- パスのつなぎ変え操作を行う
- 元の木ではこんな感じ



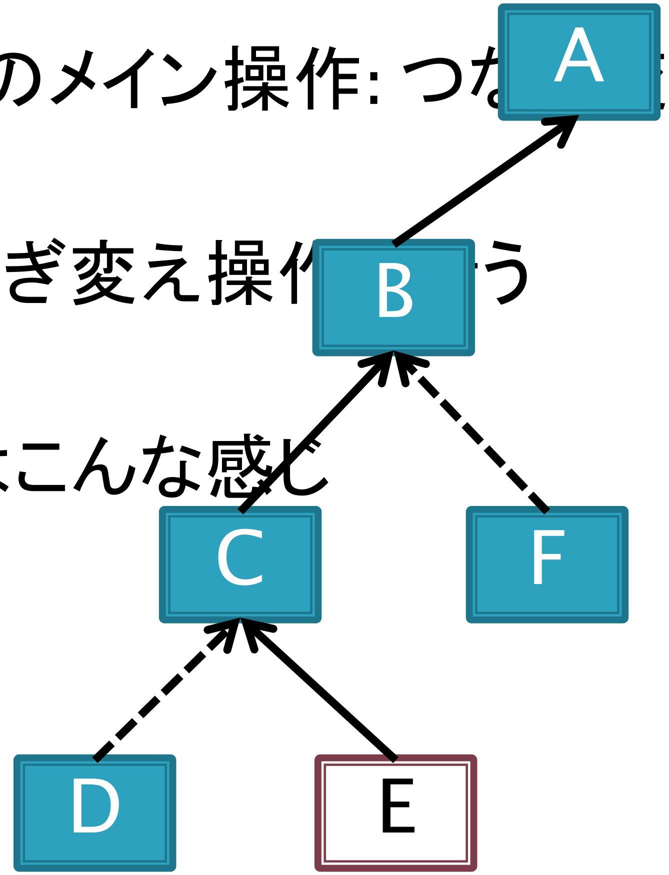
小課題3 (60点) – L/C木の実装

- splayLC() のメイン操作: つな　　え(expose操作)
- パスのつなぎ変え操作　　う
- 元の木ではこんな感じ



小課題3 (60点) – L/C木の実装

- splayLC() のメイン操作: つなぎ変え(expose操作)
- パスのつなぎ変え操作を行う
- 元の木ではこんな感じ

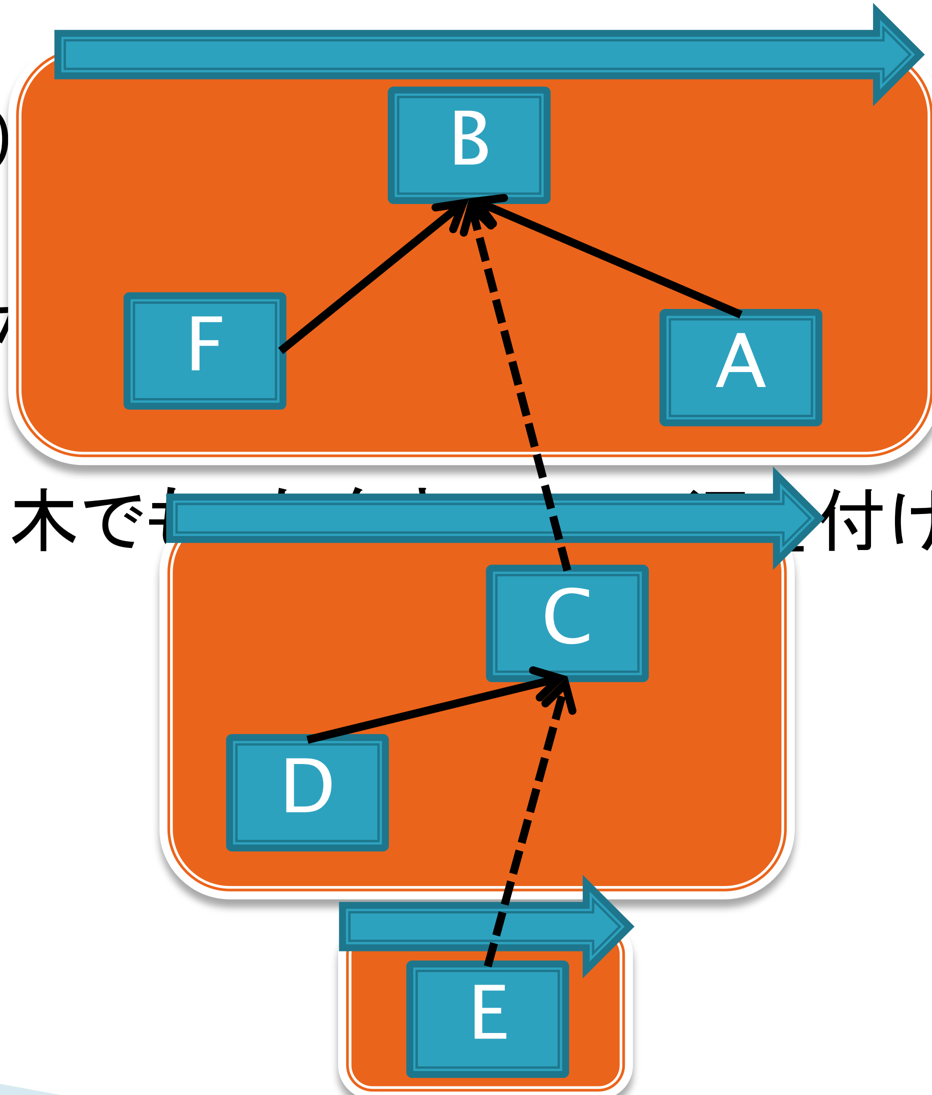


小課題3 (60点) – L/C木の実装

- splayLC() のメイン操作: つなぎ変え(expose操作)
- パスのつなぎ変え操作を行う
- Link/Cut 木でも、左向きのSolid辺を付け替えるだけ

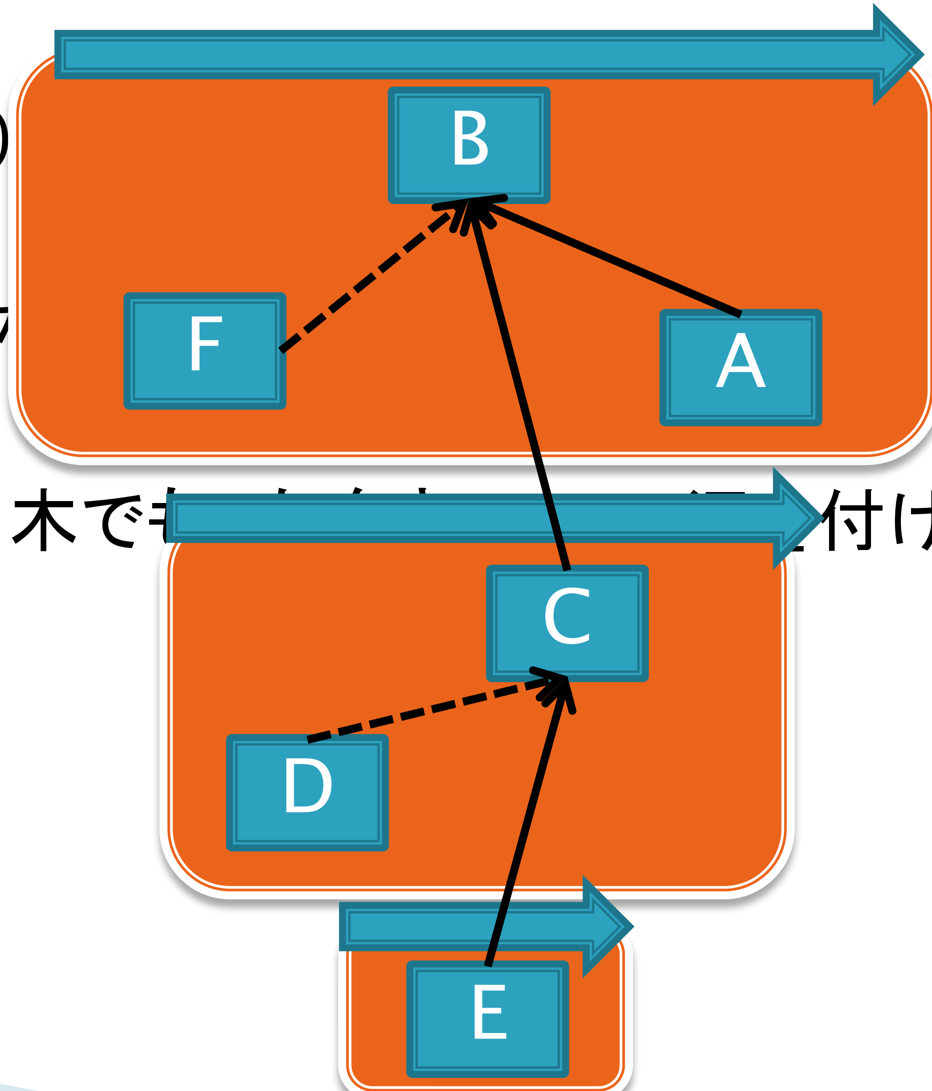
小課題3 (60点) – L/C木の実装

- splayLC() ... ose操作)
- パスのつ ...
- Link/Cut 木でも ... 付け替えるだけ



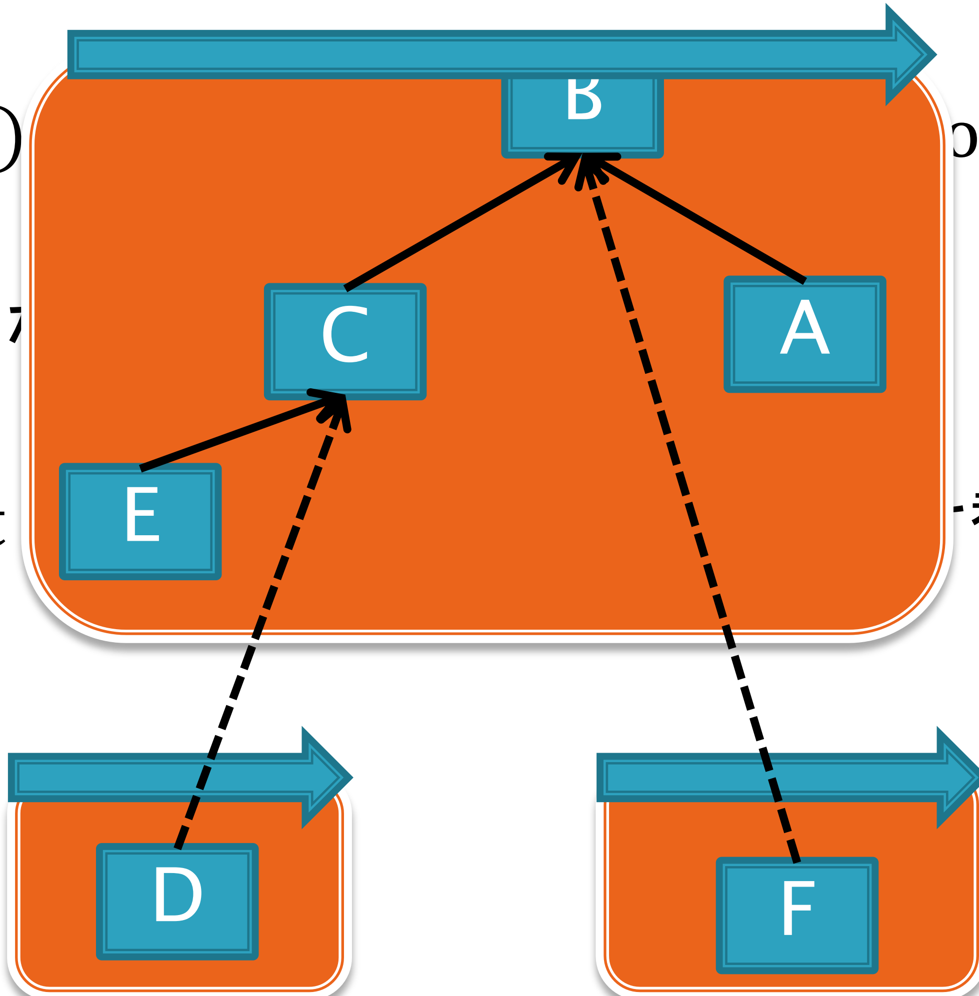
小課題3 (60点) – L/C木の実装

- splayLC() ...ose操作)
- パスのつ...
- Link/Cut 木でも...付け替えるだけ



小課題3 (60点) – L/C木の実装

- splayLC()
- パスのつ
- Link/Cut



ose操作)

替えるだけ

小課題3 (60点) – L/C木の実装

- 左向きの辺をつなぎ替えるだけで、Eが一番上の木に所属するようになった

小課題3 (60点) – L/C木の実装

- 左向きの辺をつなぎ替えるだけで、Eが一番上の木に所属するようになった
- 最後にもう1度splay()操作を行うことで、Link/Cut Treeの一番上の根にEが来る

小課題3 (60点) – L/C木の解析

- L/C木のsplayLC()はSplay Treeの解析を少し応用すると、対数時間であることが言える

小課題3 (60点) – L/C木の解析

- L/C木のsplayLC()はSplay Treeの解析を少し応用すると、対数時間であることが言える

- 1回ごとのsplay()操作が対数時間であることは既にわかっている

小課題3 (60点) – L/C木の解析

- L/C木のsplayLC()はSplay Treeの解析を少し応用すると、対数時間であることが言える

- 1回ごとのsplay()操作が対数時間であることは既にわかっている
- しかし実際にはsplay()操作がk回呼ばれている
  - kは、パス分割された木の上での深さ

小課題3 (60点) – L/C木の解析

- ポテンシャルの定義
  - サイズ = solid/dashedに関わらず、子孫になっている頂点の数
  - ランク = その対数
  - ポテンシャル = ランクの総和の2倍として定める

小課題3 (60点) – L/C木の解析

- Splay Treeのならし計算量は $1 + 3r(t) - 3r(x)$ だった
- 今回のならし計算量は$k + 6r(t) - 6r(x)$になる
  - 「Splayがk回呼ばれる」という認識を改めてみる
  - Splayは根に向かって順番に呼ばれるということを考慮すると、「Splayが1回呼ばれるが、途中でk回、強制的にzigステップを使われるかもしれない」と考えることができる
  - 係数が2倍なのはポテンシャルの定義を変えたから

小課題3 (60点) – L/C木の解析

- 余った定数項kの回収

- Expose操作のあとに1回行うsplay操作：k回の回転を行う。
- ポテンシャルの定義を2倍にしたので、splayの回転操作1回につき1の追加コストを課しても問題ない

小課題3 (60点) – L/C木の解析

- ならしコスト$6\log_2 N$のsplay操作を2回呼んでいるので、splayLC()のならし計算量は$12\log_2 N = O(\log N)$であるとわかった。

小課題3 (60点) – LCAを求める

- AとBのLCAを求める。

小課題3 (60点) – LCAを求める

- AとBのLCAを求めるには、まず
  1. Bに対してsplayLC()を行う
  2. Aに対してsplayLC()を行う
- このとき、Bは浅い位置にいる。
  - Splay Treeに対するSplay操作1回で、他の頂点の深さは高々2段しか下がらないので、この時点でBは深さ高々4程度。

小課題3 (60点) – LCAを求める

- AとBの位置関係に基いて条件分岐

小課題3 (60点) – LCAを求める

- AとBの位置関係に基いて条件分岐
- (1) BがAの左側にある場合
  - この場合は、BはAの子孫ということになるので、AとBのLCAはAになる。
- (2) BがAの右側にある場合
  - 次のページへ

小課題3 (60点) – LCAを求める

- BがAの右側にある場合の条件分岐
- (1) BがAと同じSplay Treeに属する場合
  - この場合は、AはBの子孫ということになるので、AとBのLCAはBになる。

小課題3 (60点) – LCAを求める

- BがAの右側にある場合の条件分岐
- (1) BがAと同じSplay Treeに属する場合
  - この場合は、AはBの子孫ということになるので、AとBのLCAはBになる。
- (2) BがAと異なるSplay Treeに属する場合
  - 一番一般的な場合。
  - Bから上に辿り、Aと同じSplay Treeに到達したところの頂点が、AとBのLCAになる。

小課題3 (60点) – LCAを求める

- BがAの右側にある場合の条件分岐
- (1) BがAと同じSplay Treeに属する場合
  - この場合は、AはBの子孫ということになるので、AとBのLCAはBになる。
- (2) BがAと異なるSplay Treeに属する場合
  - 一番一般的な場合。
  - Bから上に辿り、Aと同じSplay Treeに到達したところの頂点が、AとBのLCAになる。

- これでLCAは求められた。

小課題3 (60点) – 木の操作

- クエリ1,2番に対応する「接続」「切断」は、Link/Cut Treeの”link”, “cut” に対応する。

小課題3 (60点) – 木の操作

- クエリ1,2番に対応する「接続」「切断」は、Link/Cut Treeの"link", "cut" に対応する。
- (1) Link操作 – AをBの子にする
  - AとBをsplayLC()しておいてから、Aの親として(dashedで)Bを設定するだけ。
  - 計算量: AとBがLink/Cut Treeにおける根にあるので、Bのサイズが高々N増える程度。これによってポテンシャルは$O(\log N)$ しか増えない。

小課題3 (60点) – 木の操作

- クエリ1,2番に対応する「接続」「切断」は、Link/Cut Treeの”link”, “cut” に対応する。
- (2) Cut操作 – Aを親から切り離す
  - AをsplayLC()してからAの右の子を切り離す。
  - 計算量：ポテンシャルは明らかに減っている。

小課題3 (60点)

- 以上がLink/Cut Treeによる満点解法。

Euler Tour vs. Link/Cut

- Euler Tour Tree と Link/Cut Tree は動的木の筆頭

Euler Tour vs. Link/Cut

- Euler Tour Tree と Link/Cut Tree は動的木の筆頭
- 今回はどちらを選ぶべきだったか？

Euler Tour vs. Link/Cut

- Euler Tour Tree と Link/Cut Tree は動的木の筆頭
- 今回はどちらを選ぶべきだったか？

- (他の問題は解き終わっているとして)

Euler Tour vs. Link/Cut

- Euler Tour Tree
  - 知識 :
  - 実装 :

- Link/Cut Tree
  - 知識 :
  - 実装 :

Euler Tour vs. Link/Cut

- Euler Tour Tree
  - 知識: 過去にも出題済みの知識の組合せ。
  - 実装:
- Link/Cut Tree
  - 知識:
  - 実装:

Euler Tour vs. Link/Cut

- Euler Tour Tree
  - 知識: 過去にも出題済みの知識の組合せ。
  - 実装: 組み合わせてはいけないものを組み合わせてしまった感じ
- Link/Cut Tree
  - 知識:
  - 実装:

Euler Tour vs. Link/Cut

- Euler Tour Tree
  - 知識: 過去にも出題済みの知識の組合せ。
  - 実装: 組み合わせてはいけないものを組み合わせてしまった感じ
- Link/Cut Tree
  - 知識: 必須
  - 実装:

Euler Tour vs. Link/Cut

- Euler Tour Tree
  - 知識: 過去にも出題済みの知識の組合せ。
  - 実装: 組み合わせてはいけないものを組み合わせてしまった感じ
- Link/Cut Tree
  - 知識: 必須
  - 実装: 頂点にデータを持たせなくてよいなど、この問題においては極めて有利

Euler Tour vs. Link/Cut

- Euler Tour Tree
  - 知識: 過去にも出題済みの知識の組合せ。
  - 実装: 組み合わせてはいけないものを組み合わせてしまった感じ
- Link/Cut Tree
  - 知識: 必須
  - 実装: 頂点にデータを持たせなくてよいなど、この問題においては極めて有利
- 知っているならLink/Cut を書くべきだったかもしれない

Link/Cutを学ぶべきか？

- Link/Cutを学ぶべきか？

Link/Cutを学ぶべきか？

- Link/Cutを学ぶべきか？
  - Link/Cutでなければ出来ない、という問題は、恐らくない

Link/Cutを学ぶべきか？

- Link/Cutを学ぶべきか？
  - Link/Cutでなければ出来ない、という問題は、恐らくない
  - しかし、Link/Cutを使うと有利な問題は実際に存在している

Link/Cutを学ぶべきか？

- qnighyからの提案

Link/Cutを学ぶべきか？

- qnighyからの提案
  - 合宿参加者の大半にとっては、Link/Cut Treeを習得するコストが高くつく上に、他の学習をしたほうがずっと為になると思う。

Link/Cutを学ぶべきか？

- qnighyからの提案
  - 合宿参加者の大半にとっては、Link/Cut Treeを習得するコストが高くつく上に、他の学習をしたほうがずっと為になると思う。
  - より上位の人や、単純に興味があるという人に関しては、この限りではない。

Link/Cutを学ぶべきか？

- qnighyからの提案
  - 合宿参加者の大半にとっては、Link/Cut Treeを習得するコストが高くつく上に、他の学習をしたほうがずっと為になると思う。
  - より上位の人や、単純に興味があるという人に関しては、この限りではない。
  - いずれにせよ、学習するつもりなら、身に付けるために問題を解くべきだろう。

参考問題

- JOI2010春合宿 Day4 “Highway”
- JOI2012本選 問題5 “Festivals in JOI Kingdom”
- IOI2011 Day2 “Elephants”
- IJPC2012 Day3 “Animals2”

参考資料

- 完全制覇・ツリー上でのクエリ処理技法 [iwiwi]
  http://topcoder.g.hatena.ne.jp/iwiwi/20111205/1323099376
- プログラミングコンテストでのデータ構造 2 ～動的木編～ [iwiwi]
  http://www.slideshare.net/iwiwi/2-12188845
- 蟻本 [iwiwi]

参考文献

- Daniel D. Sleator and Robert E. Tarjan, A Data Structure for Dynamic Trees, Journal of Computer and System Sciences, Volume 26 Issue 3, June 1983, pp. 362 – 391
- Daniel D. Sleator and Robert E. Tarjan, Self-adjusting binary search trees, Journal of the ACM, Volume 32 Issue 3, July 1985, pp. 652 – 686

得点分布