離散最適化基礎論 第 8 回
計算量 (1)：近傍探索の高速化

岡本 吉央
okamotoy@uec.ac.jp

電気通信大学

2013 年 12 月 13 日

最終更新：2013 年 12 月 17 日　22:36

目次

局所探索法を評価する観点 (再掲)

局所探索法 (local search)：一般的枠組み

0. 許容解 $x \in S$ を適当に見つける (この x を初期解と呼ぶ)
1. 以下を繰り返し
   1. ある $x' \in N(x)$ が存在して $f(x') < f(x)$ ならば $x \leftarrow x'$
   2. そうでなければ繰り返しを終了
2. x を出力 (この x は N に関する局所最適解 (定理 1.1))

局所探索法の出力は N に関する局所最適解

局所探索法を評価する観点

- **性能保証** (performance guarantee)
  局所最適解が最適解に (目的関数値で) どれだけ近いか？
- **計算量** (computational complexity)
  局所探索法がどれだけ速く局所最適解に収束するか？

局所探索法を評価する観点：計算量の評価

局所探索法 (local search)：一般的枠組み

0. 許容解 $x \in S$ を適当に見つける (この x を初期解と呼ぶ) ←時間 T_A
1. 以下を繰り返し ←反復回数 k
   1. ある $x' \in N(x)$ が存在して $f(x') < f(x)$ ならば ←時間 T_C
      $x \leftarrow x'$
   2. そうでなければ繰り返しを終了
2. x を出力 (この x は N に関する局所最適解 (定理 1.1))

全体の計算時間

およそ $T_A + k \cdot T_C$

よって，局所探索法が多項式時間アルゴリズムになるには，次が必要

- T_A が多項式
- $k \cdot T_C$ が多項式

局所探索法を評価する観点：計算量の評価

局所探索法 (local search)：一般的枠組み

0. 許容解 $x \in S$ を適当に見つける (この x を初期解と呼ぶ) ←時間 T_A
1. 以下を繰り返し ←反復回数 k
   1. ある $x' \in N(x)$ が存在して $f(x') < f(x)$ ならば ←時間 T_C
      $x \leftarrow x'$
   2. そうでなければ繰り返しを終了
2. x を出力 (この x は N に関する局所最適解 (定理 1.1))

全体の計算時間

およそ $T_A + k \cdot T_C$

よって，局所探索法が最悪の場合指数時間かかるには，次が十分

- T_A が指数関数
- k が指数関数
- T_C が指数関数

局所探索法の計算量評価

局所探索法 (local search)：一般的枠組み

0. 許容解 $x \in S$ を適当に見つける (この x を初期解と呼ぶ) ←時間 T_A
1. 以下を繰り返し ←反復回数 k
   1. ある $x' \in N(x)$ が存在して $f(x') < f(x)$ ならば ←時間 T_C
      $x \leftarrow x'$
   2. そうでなければ繰り返しを終了
2. x を出力 (この x は N に関する局所最適解 (定理 1.1))

ポイント 1

反復回数の評価

ポイント 2

反復継続条件の確認にかかる時間の評価
(近傍探索時間の評価)

今後の予告

近傍探索時間の評価

反復回数の評価

近似局所探索による反復回数の削減

局所探索法にこだわらない局所最適化の難しさ

今日の内容

いままでの感覚

許容解 x の近傍 $N(x)$ の中で，x よりもよい許容解を見つけるためには，$|N(x)|$ に比例する時間は必要

今日のテーマ

実はそうではなく，
$|N(x)|$ よりも小さな時間で見つけられる (または，存在しないことが分かる)

2 つの例

- 最終完了時刻最小化スケジューリングの移動近傍
- 巡回セールスマン問題の 2opt 近傍

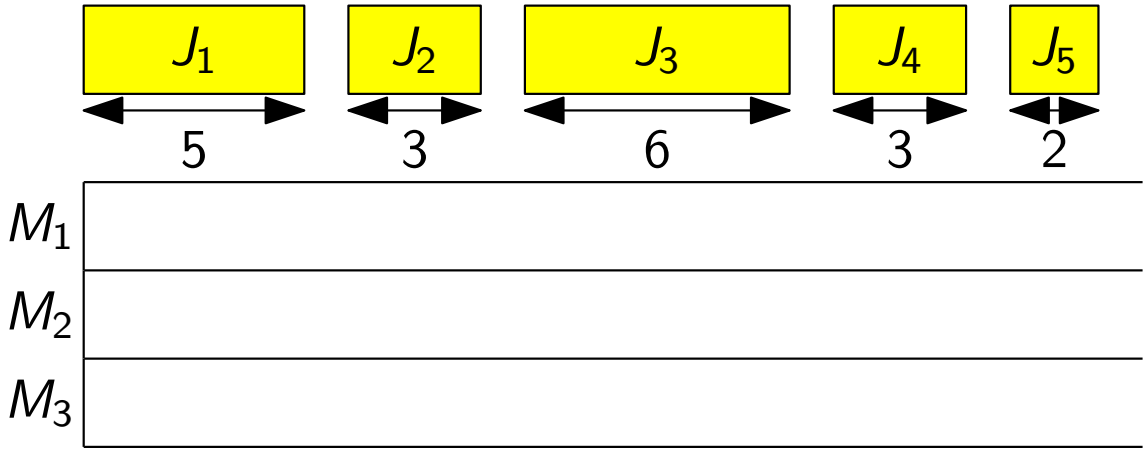
鍵となる概念

- データ構造

目次

最終完了時刻最小化スケジューリング問題

m 個の同一な機械 $M_1, \ldots, M_m$，n 個のジョブ $J_1, \ldots, J_n$，
各ジョブ J_i の処理時間 p_i



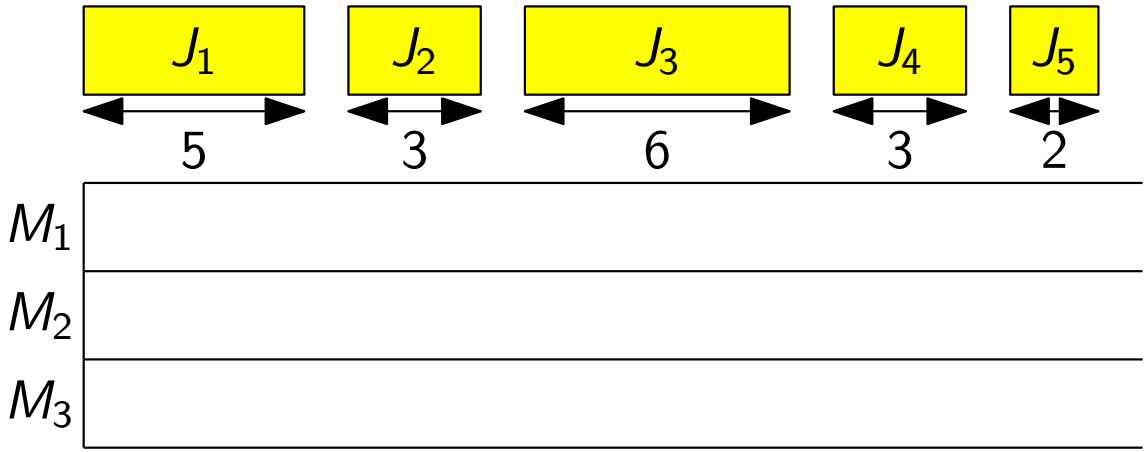
行うこと：すべてのジョブを機械で処理する

$$p_1 = 5,\ p_2 = 3,\ p_3 = 6,\ p_4 = 3,\ p_5 = 2$$

最終完了時刻最小化スケジューリング問題

決めること (スケジューリング，scheduling)

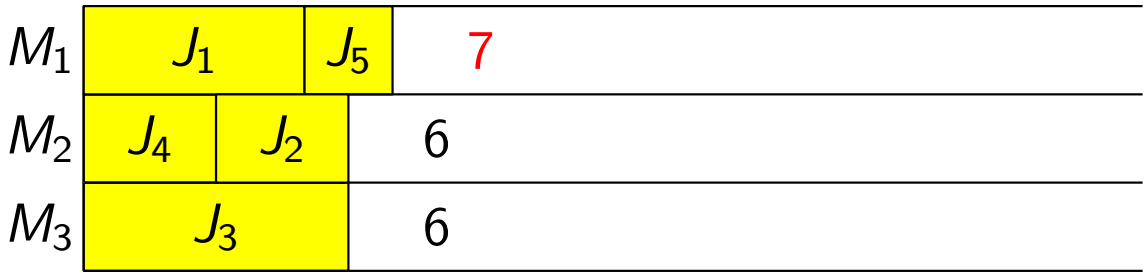
- **各ジョブをどの機械に割り当てるか？**
- **各機械で，割り当てられたジョブをどの順に処理するか？**



ジョブの中断はないものとする (non-preemptive scheduling)

最終完了時刻最小化スケジューリング問題

目的：最終完了時刻の最小化



このスケジュールにおける最終完了時刻 $= 7$

最終完了時刻最小化スケジューリング問題

最終完了時刻最小化スケジューリング問題

入力

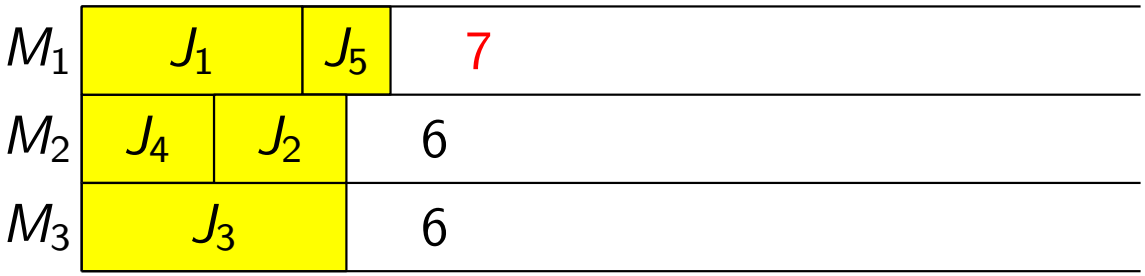
- m 個の同一な機械 $M_1, \ldots, M_m$
- n 個のジョブ $J_1, \ldots, J_n$
- 各ジョブ J_i の処理時間 p_i

許容解

- スケジュール (機械へのジョブの割当，処理順)

目的

- 最終完了時刻の最小化



最終完了時刻最小化スケジューリング問題に対する移動操作

$m =$ 機械の台数，$n =$ ジョブの個数

最終完了時刻最小化スケジューリング問題に対する移動操作

$(A_1, A_2, \ldots, A_m)$：現在保持しているジョブの割当

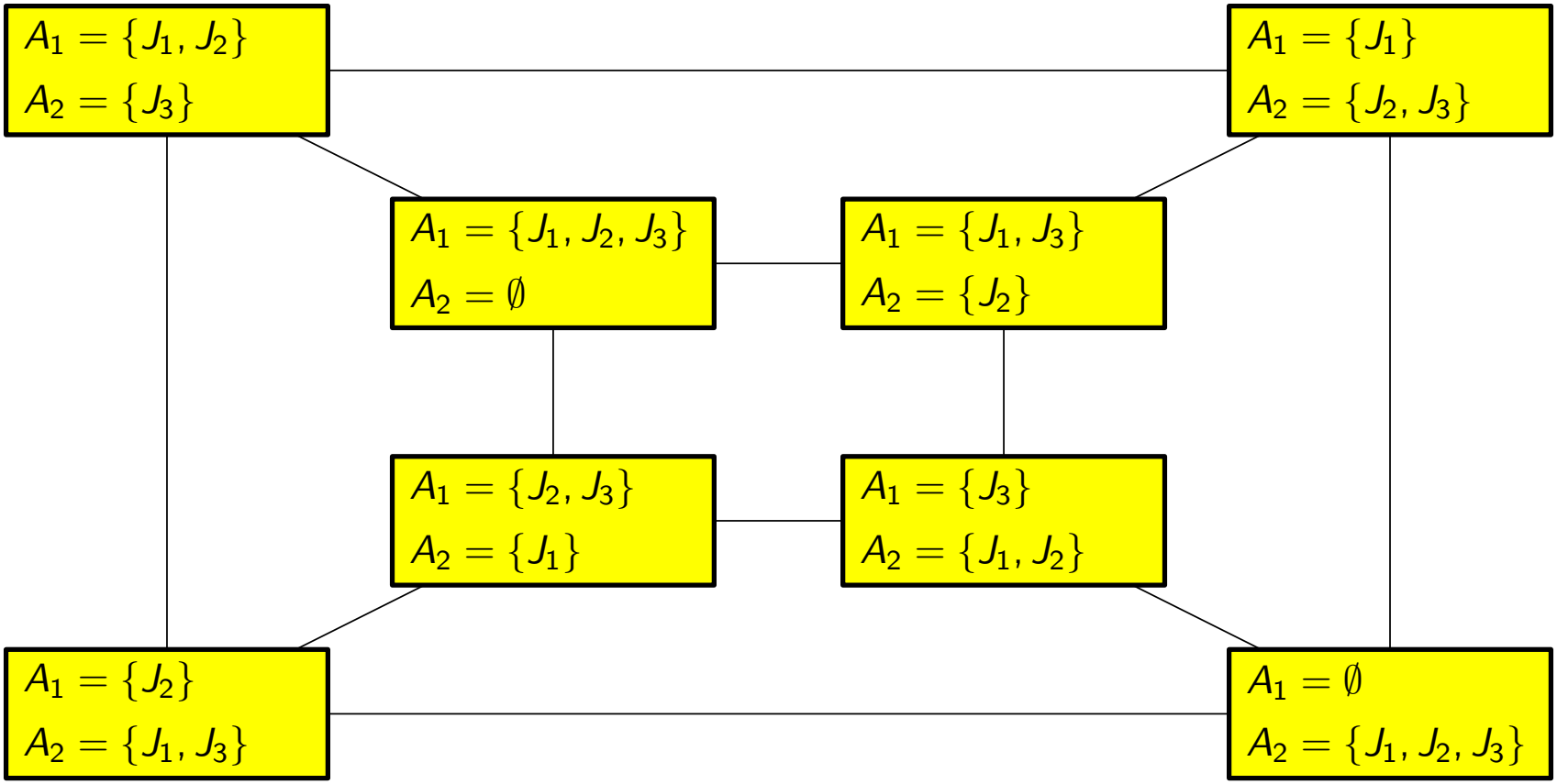
($A_i =$ 機械 i に割り当てられたジョブの集合)

1. $i \in \{1, \ldots, m\}$ を 1 つ選び，$J \in A_i$ を 1 つ選ぶ
2. $i' \in \{1, \ldots, m\} - \{i\}$ を 1 つ選び，J を A_i から $A_{i'}$ に動かす

$$\boxed{\begin{aligned} A_1 &= \{J_1, J_2, J_4\} \\ A_2 &= \{J_3, J_6\} \\ A_3 &= \{J_5, J_7\} \end{aligned}} \longrightarrow \boxed{\begin{aligned} A_1 &= \{J_1, J_4\} \\ A_2 &= \{J_3, J_6\} \\ A_3 &= \{J_2, J_5, J_7\} \end{aligned}}$$

移動操作が導く近傍を移動近傍と呼ぶ (これは対称な近傍)

移動近傍に関する近傍グラフ



移動近傍の探索を効率的に行う

移動近傍の大きさ

だいたい mn (つまり，$\Theta(mn)$)

$\therefore$ **「素朴」な近傍探索 1 回にかかる時間 $= O(mn)$**

$$\boxed{\begin{aligned} A_1 &= \{J_1, J_2, J_4\} \\ A_2 &= \{J_3, J_6\} \\ A_3 &= \{J_5, J_7\} \end{aligned}} \longrightarrow \boxed{\begin{aligned} A_1 &= \{J_1, J_4\} \\ A_2 &= \{J_3, J_6\} \\ A_3 &= \{J_2, J_5, J_7\} \end{aligned}}$$

今から示すこと

次のようなデータ構造の構築

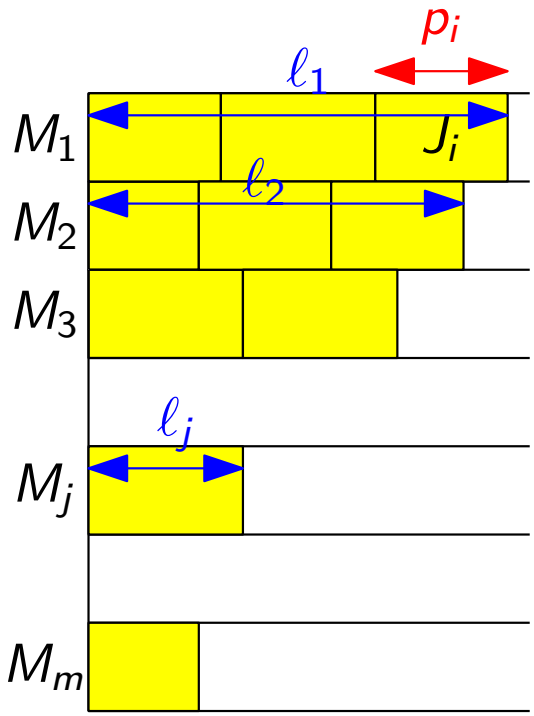
- **構築：$O(n)$ 時間** (← **初期解に対して行う**)
- **近傍探索 1 回：$O(\log n)$ 時間**
- **使用メモリ：$O(n)$**

ここで，$m \leq n$ と仮定してよい (**演習問題**)

移動操作によって最終完了時刻を小さくする方法は何か？ (1)

各機械の完了時刻が $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_m$ を満たすとする

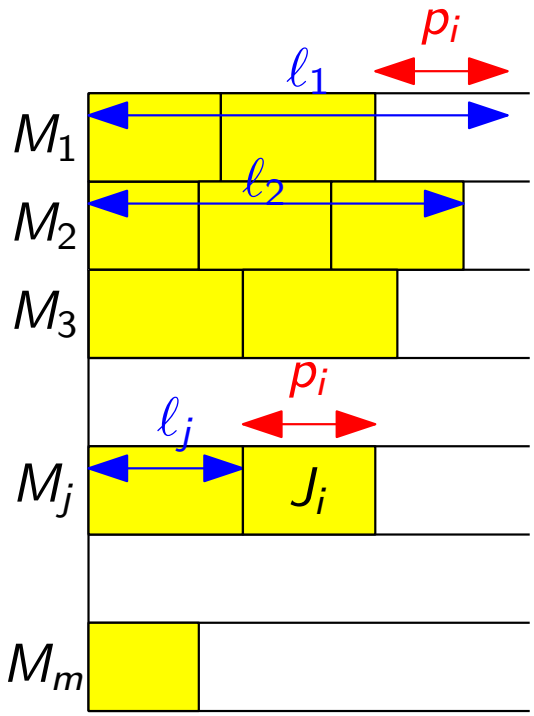
- 完了時刻が最も大きい機械 M_1 に割り当てられたジョブを移動させる必要がある
- 移動させた後に，完了時刻が最も大きい機械となりうるのは？
  - M_1
  - M_2
  - ジョブの移動先である機械 M_j



移動操作によって最終完了時刻を小さくする方法は何か？ (1)

各機械の完了時刻が $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_m$ を満たすとする

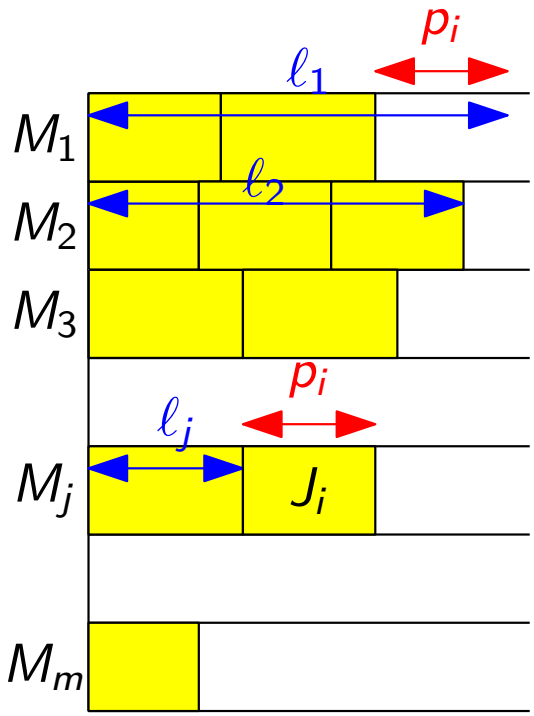
- 完了時刻が最も大きい機械 M_1 に割り当てられたジョブを移動させる必要がある
- 移動させた後に，完了時刻が最も大きい機械となりうるのは？
  - M_1
  - M_2
  - ジョブの移動先である機械 M_j



移動操作によって最終完了時刻を小さくする方法は何か？ (1)

各機械の完了時刻が $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_m$ を満たすとする

- 完了時刻が最も大きい機械 M_1 に割り当てられたジョブを移動させる必要がある
- 移動させた後に，完了時刻が最も大きい機械となりうるのは？
  - M_1
  - M_2
  - ジョブの移動先である機械 M_j
- つまり，次が必要十分条件



移動操作によって最終完了時刻を小さくできる $\Leftrightarrow$

M_1 に割り当てられたあるジョブ J_i とある機械 M_j が存在して

$$\max\{\ell_2, \ell_j + p_i\} < \ell_1$$

移動操作によって最終完了時刻を小さくする方法は何か？ (2)

M_1 に割り当てられたあるジョブ J_i とある機械 M_j が存在して

$$\max\{\ell_2, \ell_j + p_i\} < \ell_1$$

移動操作によって最終完了時刻を小さくする方法は何か？ (2)

M_1 に割り当てられたあるジョブ J_i とある機械 M_j が存在して

$$\max\{\ell_2, \ell_j + p_i\} < \ell_1$$

$\Leftrightarrow$ M_1 に割り当てられたあるジョブ J_i が存在して

$$\max\{\ell_2, \ell_m + p_i\} < \ell_1$$

移動操作によって最終完了時刻を小さくする方法は何か？ (2)

M_1 に割り当てられたあるジョブ J_i とある機械 M_j が存在して

$$\max\{\ell_2, \ell_j + p_i\} < \ell_1$$

$\Leftrightarrow$ M_1 に割り当てられたあるジョブ J_i が存在して

$$\max\{\ell_2, \ell_m + p_i\} < \ell_1$$

$\Leftrightarrow$ M_1 に割り当てられたジョブの中で**処理時間最小**のものを J_i とすると

$$\max\{\ell_2, \ell_m + p_i\} < \ell_1$$

移動操作によって最終完了時刻を小さくする方法は何か？ (2)

M_1 に割り当てられたあるジョブ J_i とある機械 M_j が存在して

$$\max\{\ell_2, \ell_j + p_i\} < \ell_1$$

$\Leftrightarrow$ M_1 に割り当てられたあるジョブ J_i が存在して

$$\max\{\ell_2, \ell_m + p_i\} < \ell_1$$

$\Leftrightarrow$ M_1 に割り当てられたジョブの中で**処理時間最小**のものを J_i とすると

$$\max\{\ell_2, \ell_m + p_i\} < \ell_1$$

つまり，分かればよいものは

- 完了時刻の最大値 (ℓ_1)
- 完了時刻の第 2 最大値 (ℓ_2)
- 完了時刻の最小値 (ℓ_m)
- 完了時刻最大の機械に割り当てられたジョブにおける処理時間最小値 (p_i)

いまから行うこと

1. この条件を簡単に確認するために作成するデータ構造の概略
2. そのようなデータ構造があるとき，
   どのように条件を確認するのか，また，
   どのようにデータ構造を更新するのか
3. データ構造の実現

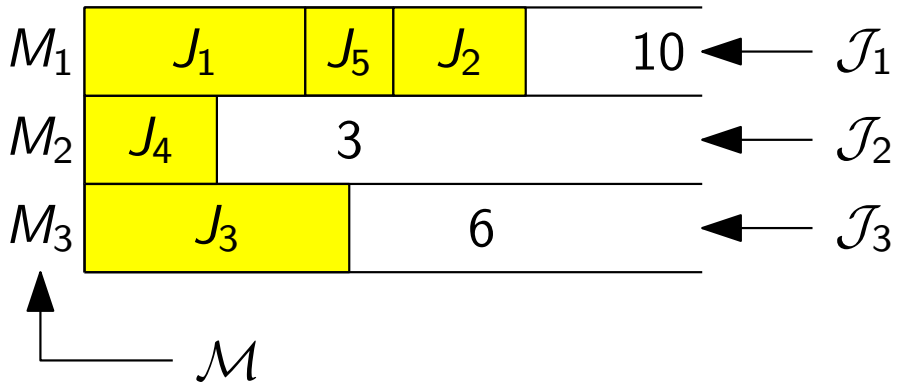
作成するデータ構造

機械を保持するデータ構造 $\mathcal{M}$

- $\langle$ 不変条件 $\rangle$：完了時刻最大の機械を取りだす操作 $O(1)$ 時間
- $\langle$ 不変条件 $\rangle$：完了時刻第 2 最大の機械を取りだす操作 $O(1)$ 時間
- $\langle$ 不変条件 $\rangle$：完了時刻最小の機械を取りだす操作 $O(1)$ 時間
- $\langle$ 不変条件 $\rangle$：指定した機械の完了時刻を変更する操作 $O(\log m)$ 時間

各機械 M_j に割り当てられたジョブを保持するデータ構造 $\mathcal{J}_j$

- $\langle$ 不変条件 $\rangle$：処理時間最小のジョブを取りだす操作 $O(1)$ 時間
- $\langle$ 不変条件 $\rangle$：指定したジョブを削除する操作 $O(\log n)$ 時間
- $\langle$ 不変条件 $\rangle$：指定したジョブを挿入する操作 $O(\log n)$ 時間



データ構造を使用した近傍探索の効率化：条件の確認

確認したい条件

M_1 に割り当てられたジョブの中で**処理時間最小**のものを J_i とすると

$$\max\{\ell_2, \ell_m + p_i\} < \ell_1$$

そのための手続き

1. $M_{j_1} \leftarrow$ 完了時刻最大の機械
2. $M_{j_2} \leftarrow$ 完了時刻第 2 最大の機械
3. $M_{j_m} \leftarrow$ 完了時刻最小の機械
4. $J_i \leftarrow M_{j_1}$ の処理時間最小ジョブ
5. $\max\{\ell_{j_2}, \ell_{j_m} + p_i\} < \ell_{j_1}$ ならば，
   J_i を M_{j_1} から M_{j_m} に移動させて，次の許容解を得る
   そうでなければ，終了

データ構造を使用した近傍探索の効率化：条件の確認 (計算量)

確認したい条件

M_1 に割り当てられたジョブの中で**処理時間最小**のものを J_i とすると

$$\max\{\ell_2, \ell_m + p_i\} < \ell_1$$

そのための手続き

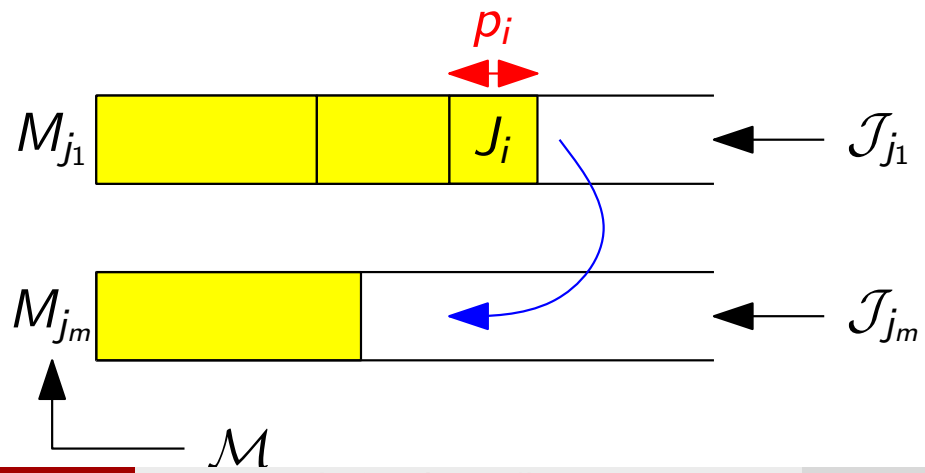
1. $M_{j_1} \leftarrow$ 完了時刻最大の機械 $\leftarrow O(1)$ 時間 ($\mathcal{M}$ への問合せ)
2. $M_{j_2} \leftarrow$ 完了時刻第 2 最大の機械 $\leftarrow O(1)$ 時間 ($\mathcal{M}$ への問合せ)
3. $M_{j_m} \leftarrow$ 完了時刻最小の機械 $\leftarrow O(1)$ 時間 ($\mathcal{M}$ への問合せ)
4. $J_i \leftarrow M_{j_1}$ の処理時間最小ジョブ $\leftarrow O(1)$ 時間 ($\mathcal{J}_{j_1}$ への問合せ)
5. $\max\{\ell_{j_2}, \ell_{j_m} + p_i\} < \ell_{j_1}$ ならば，
   J_i を M_{j_1} から M_{j_m} に移動させて，次の許容解を得る
   そうでなければ，終了

合計 $O(1)$ 時間

データ構造を使用した近傍探索の効率化：データ構造の更新

行う変更

- $\mathcal{J}_{j_1}$ から J_i を削除
- $\mathcal{J}_{j_m}$ に J_i を追加
- $\mathcal{M}$ において M_{j_1} の完了時刻を p_i だけ小さくする
- $\mathcal{M}$ において M_{j_m} の完了時刻を p_i だけ大きくする

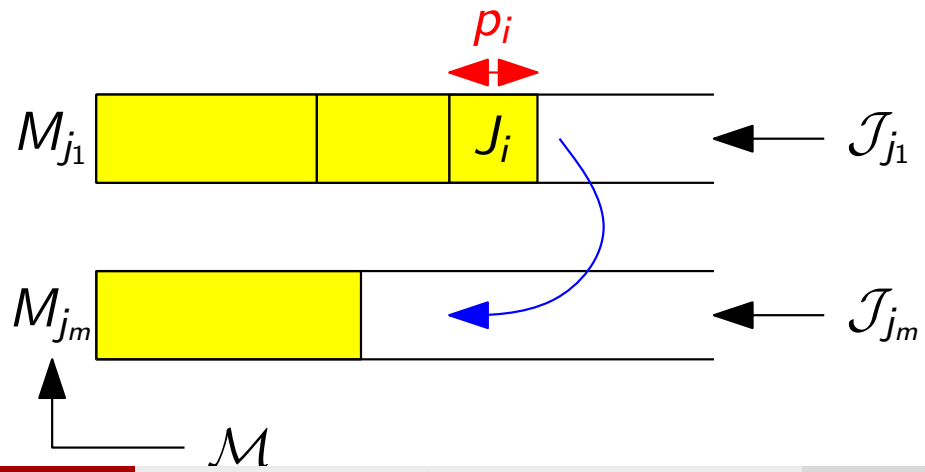


データ構造を使用した近傍探索の効率化：データ構造の更新 (計算量)

行う変更

- $\mathcal{J}_{j_1}$ から J_i を削除
  ← $O(\log n)$ 時間 ($\mathcal{J}_{j_1}$ における操作)
- $\mathcal{J}_{j_m}$ に J_i を追加
  ← $O(\log n)$ 時間 ($\mathcal{J}_{j_m}$ における操作)
- $\mathcal{M}$ において M_{j_1} の完了時刻を p_i だけ小さくする
  ← $O(\log m)$ 時間 ($\mathcal{M}$ における操作)
- $\mathcal{M}$ において M_{j_m} の完了時刻を p_i だけ大きくする
  ← $O(\log m)$ 時間 ($\mathcal{M}$ における操作)

全体で $O(\log n)$ 時間



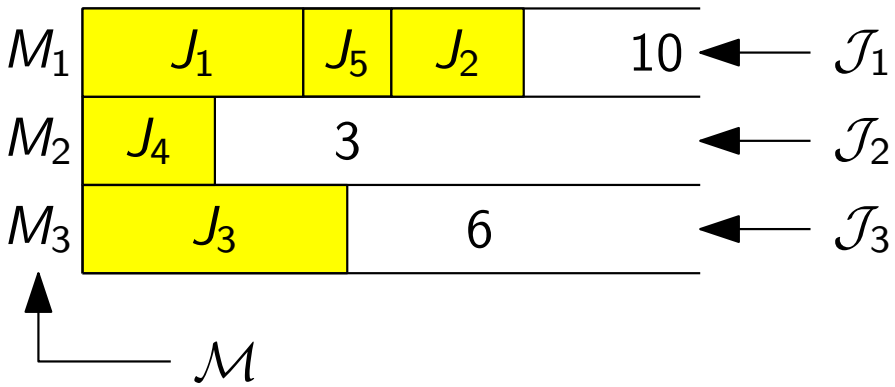
作成するデータ構造の実現

機械を保持するデータ構造 $\mathcal{M}$ ←実現：2 つの二分ヒープ

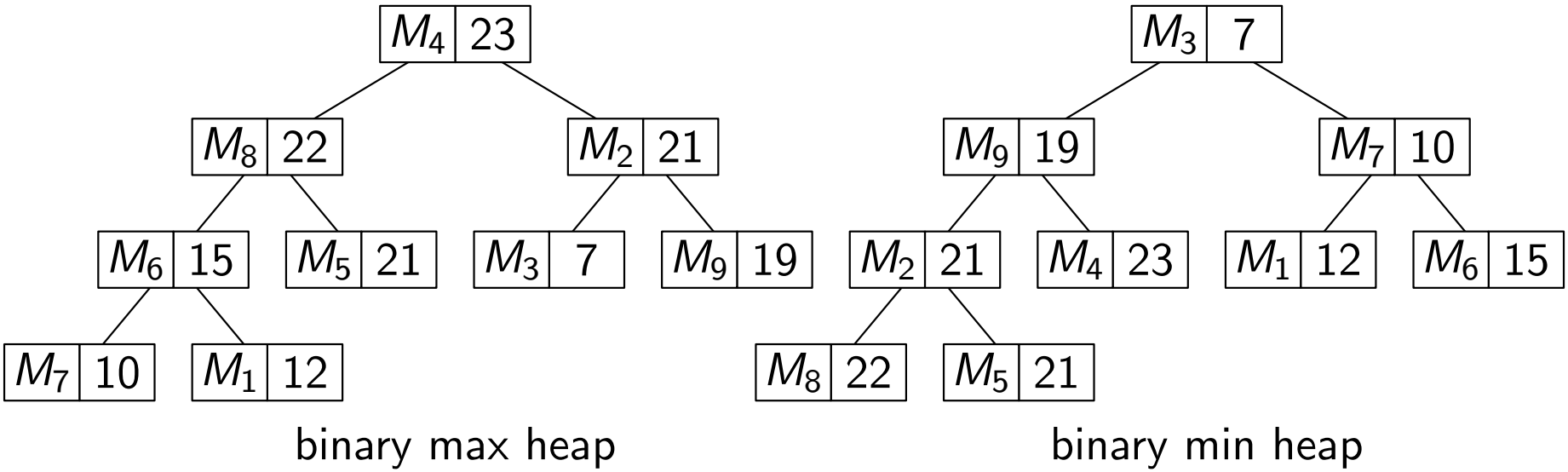
- ⟨ 不変条件 ⟩：完了時刻最大の機械を取りだす操作 $O(1)$ 時間
- ⟨ 不変条件 ⟩：完了時刻第 2 最大の機械を取りだす操作 $O(1)$ 時間
- ⟨ 不変条件 ⟩：完了時刻最小の機械を取りだす操作 $O(1)$ 時間
- ⟨ 不変条件 ⟩：指定した機械の完了時刻を変更する操作 $O(\log m)$ 時間

各機械 M_j に割り当てられたジョブを保持するデータ構造 $\mathcal{J}_j$ ←実現：二分ヒープ

- ⟨ 不変条件 ⟩：処理時間最小のジョブを取りだす操作 $O(1)$ 時間
- ⟨ 不変条件 ⟩：指定したジョブを削除する操作 $O(\log n)$ 時間
- ⟨ 不変条件 ⟩：指定したジョブを挿入する操作 $O(\log n)$ 時間



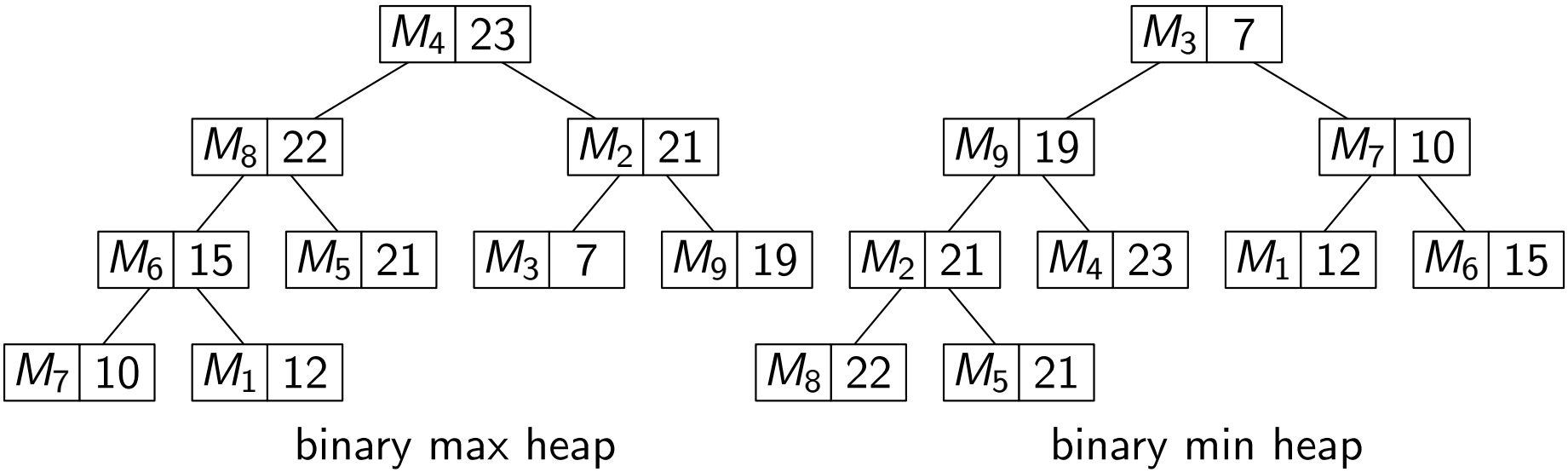
$\mathcal{M}$：2 つの二分ヒープ (1)



ヒープ条件

- **ノードの値 $\geq$ その子ノードの値** **(最大ヒープの場合)**
- **ノードの値 $\leq$ その子ノードの値** **(最小ヒープの場合)**

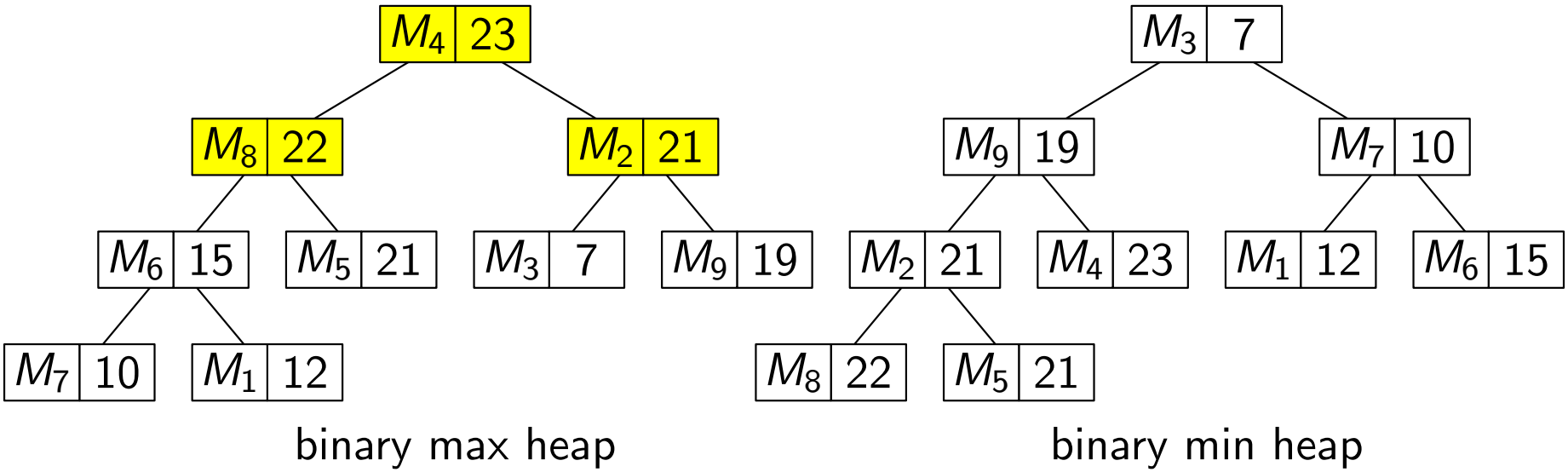
$\mathcal{M}$：2 つの二分ヒープ (2)



N 個の要素を持つ二分ヒープにおいて

- **ヒープの構築：$O(N)$ 時間**
- **最大/最小要素の発見：$O(1)$ 時間 (最大/最小ヒープにおいて)**
- **要素の挿入：$O(\log N)$ 時間**
- **要素の削除：$O(\log N)$ 時間**
- **要素の値の変更：$O(\log N)$ 時間 (←削除＋挿入)**

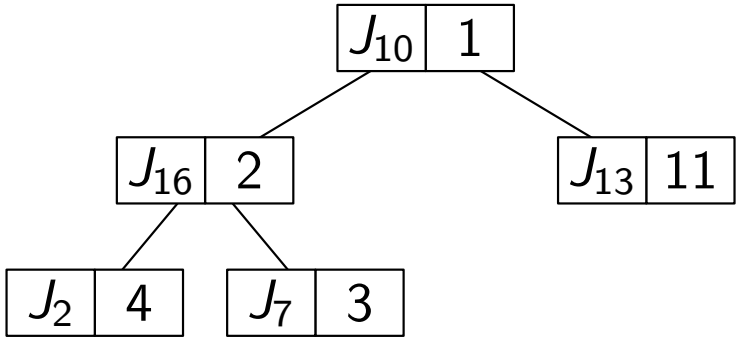
$\mathcal{M}$：2 つの二分ヒープ (3)



第 2 最大要素をどのように見つけるか？

- **ヒープ条件から，第 2 最大要素は根ノードの子ノード**
- **$\therefore$ 3 つのノードを見れば，第 2 最大要素が分かる**
- **時間：$O(1)$**

$\mathcal{J}_j$：二分ヒープ (1)

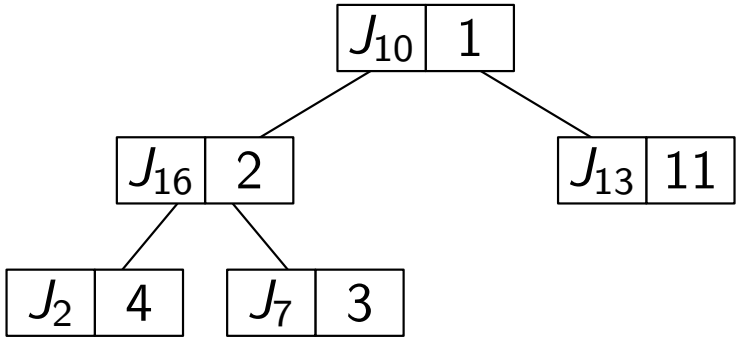


binary min heap

N 個の要素を持つ二分最小ヒープにおいて

- **ヒープの構築：$O(N)$ 時間**
- **最小要素の発見：$O(1)$ 時間**
- **要素の挿入：$O(\log N)$ 時間**
- **要素の削除：$O(\log N)$ 時間**
- **要素の値の変更：$O(\log N)$ 時間 (←削除＋挿入)**

$\mathcal{J}_j$：二分ヒープ (2)



binary min heap

$\mathcal{J}_j$ に要素が n_j 個蓄えられているとすると，

- **$\mathcal{J}_i$ の構築：$O(n_j)$ 時間**

よって，$\mathcal{J}_1, \ldots, \mathcal{J}_m$ すべてを構築するのにかかる時間は

$$O(n_1) + \cdots + O(n_m) = O(n_1 + \cdots + n_m) = O(n)$$

注意：$n_1 + \cdots + n_m = n$

移動近傍の探索を効率的に行う：まとめ

移動近傍の大きさ

だいたい mn (つまり，$\Theta(mn)$)

$\therefore$ **「素朴」な近傍探索 1 回にかかる時間 $= O(mn)$**

$$\boxed{\begin{aligned} A_1 &= \{J_1, J_2, J_4\} \\ A_2 &= \{J_3, J_6\} \\ A_3 &= \{J_5, J_7\} \end{aligned}} \longrightarrow \boxed{\begin{aligned} A_1 &= \{J_1, J_4\} \\ A_2 &= \{J_3, J_6\} \\ A_3 &= \{J_2, J_5, J_7\} \end{aligned}}$$

示したこと

次のようなデータ構造の構築

- **構築：$O(n)$ 時間** **(← 初期解に対して行う)**
- **近傍探索 1 回：$O(\log n)$ 時間**
- **使用メモリ：$O(n)$**

ここで，$m \leq n$ と仮定してよい

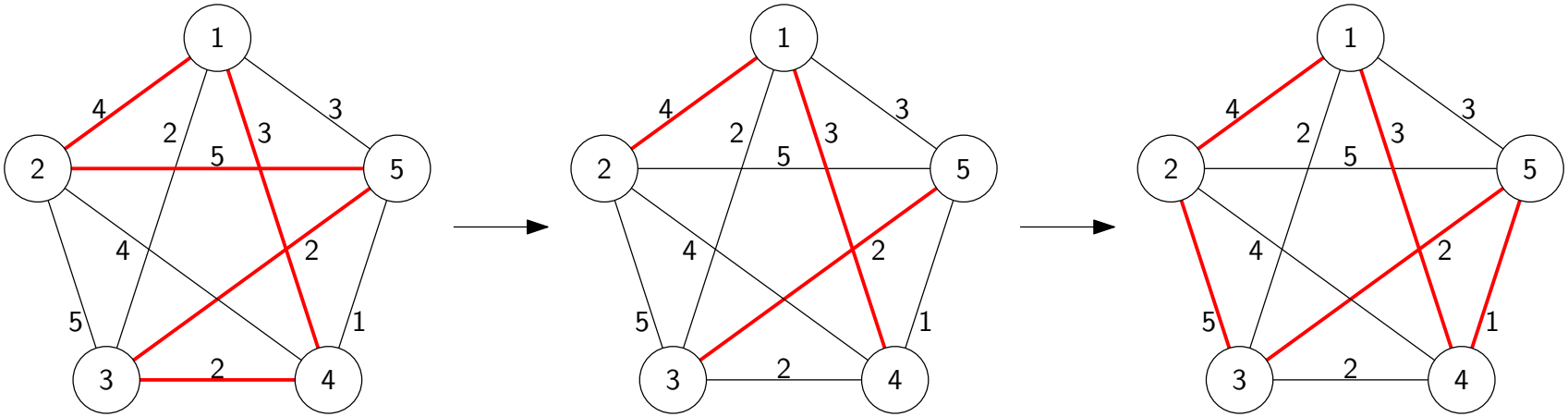
目次

巡回セールスマン問題に対する 2opt 操作

巡回セールスマン問題に対する 2opt 操作

τ：現在保持している巡回路

1. τ が使っている辺を 2 つ取り除く
2. τ が使っていなかった辺を 2 つ追加して，新たな巡回路を得る



2opt 操作が誘導する近傍を 2opt 近傍と呼ぶ

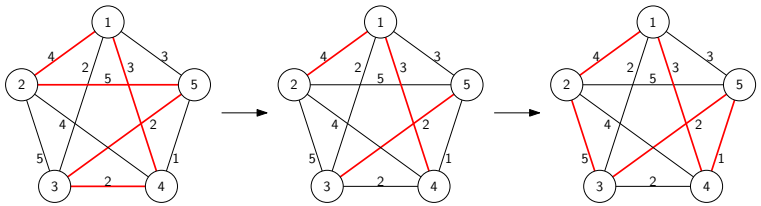
2opt 近傍の探索を効率的に行う

$n =$ 都市の数

2opt 近傍の大きさ

だいたい n^2 (つまり，$\Theta(n^2)$)

$\therefore$ 「素朴」な近傍探索 1 回にかかる時間 $= O(n^2)$



今から示すこと

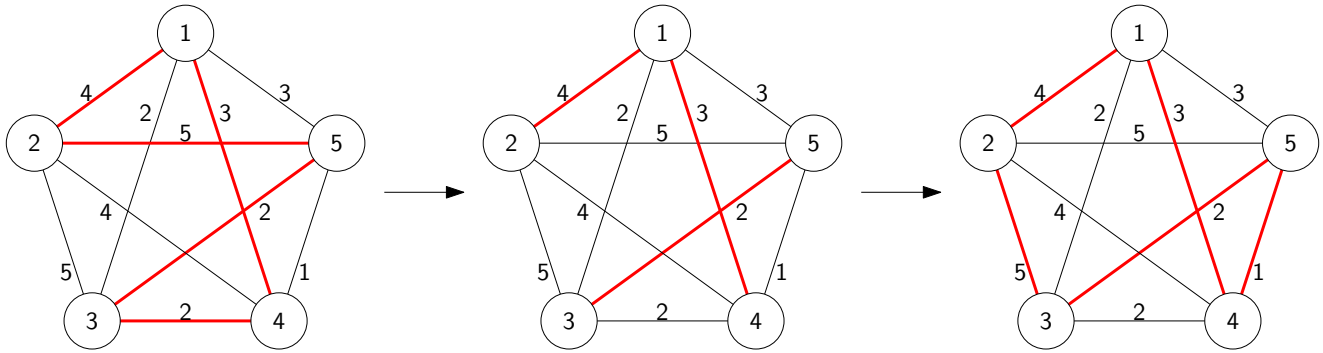
次のようなデータ構造の構築

- 構築：$O(n^2)$ 時間　　(← 初期解に対して行う)
- 近傍探索 1 回：$O(n \log n)$ 時間
- 使用メモリ：$O(n^2)$

基本的アイディア

基本的アイディア

取り除く辺を 2 つ決めたときの巡回路長の変化を記録する



- **辺** $\{2,5\}$ **と** $\{3,4\}$ **を取り除いたら，巡回路の長さの変化は**

$$-c(2,5) - c(3,4) + c(2,3) + c(4,5) = -5 - 2 + 5 + 1 = -1$$

- **辺** $\{2,5\}$ **と** $\{1,4\}$ **を取り除いたら，巡回路の長さの変化は**

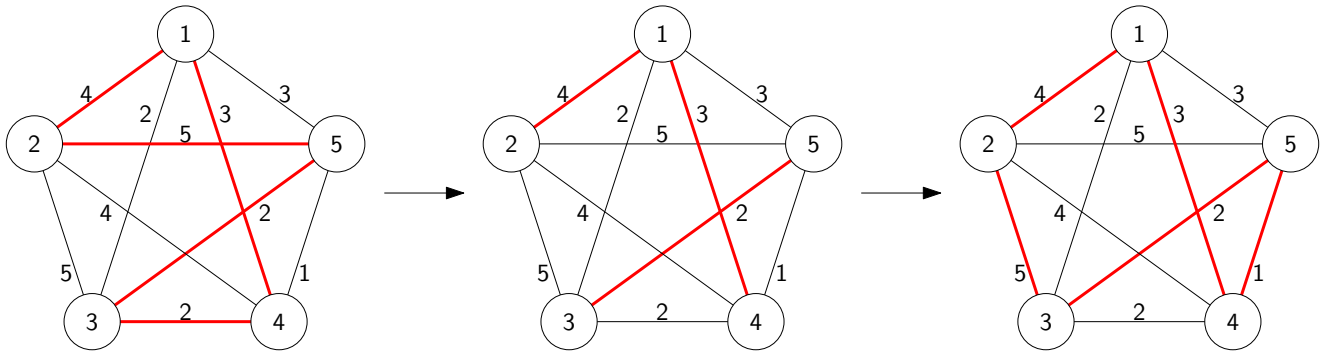
$$-c(2,5) - c(1,4) + c(2,4) + c(1,5) = -5 - 2 + 4 + 3 = 0$$

この変化は，2 辺を決めたら，$O(1)$ 時間で計算できる

基本的アイディア (続き)

基本的アイディア

取り除く辺を 2 つ決めたときの巡回路長の変化を記録する



巡回路の長さの変化が最も小さい 2 辺の組が見つけられればよい

- **その最小変化** $< 0 \Rightarrow$ 2opt **操作により，許容解が改善される**
- **その最小変化** $\geq 0 \Rightarrow$ 2opt **操作により，許容解は改善されない**

行いたいこと

行いたいこと

取り除く 2 辺の候補の数 $= \Theta(n^2)$ であるが，
その中で，変化が最小のものを簡単に見つけたい

$\rightsquigarrow$ **最小ヒープが使えそう (要素数 $= \Theta(n^2)$)**

作成するデータ構造

2opt 操作による巡回路長の変化を保持するデータ構造 $\mathcal{H}$

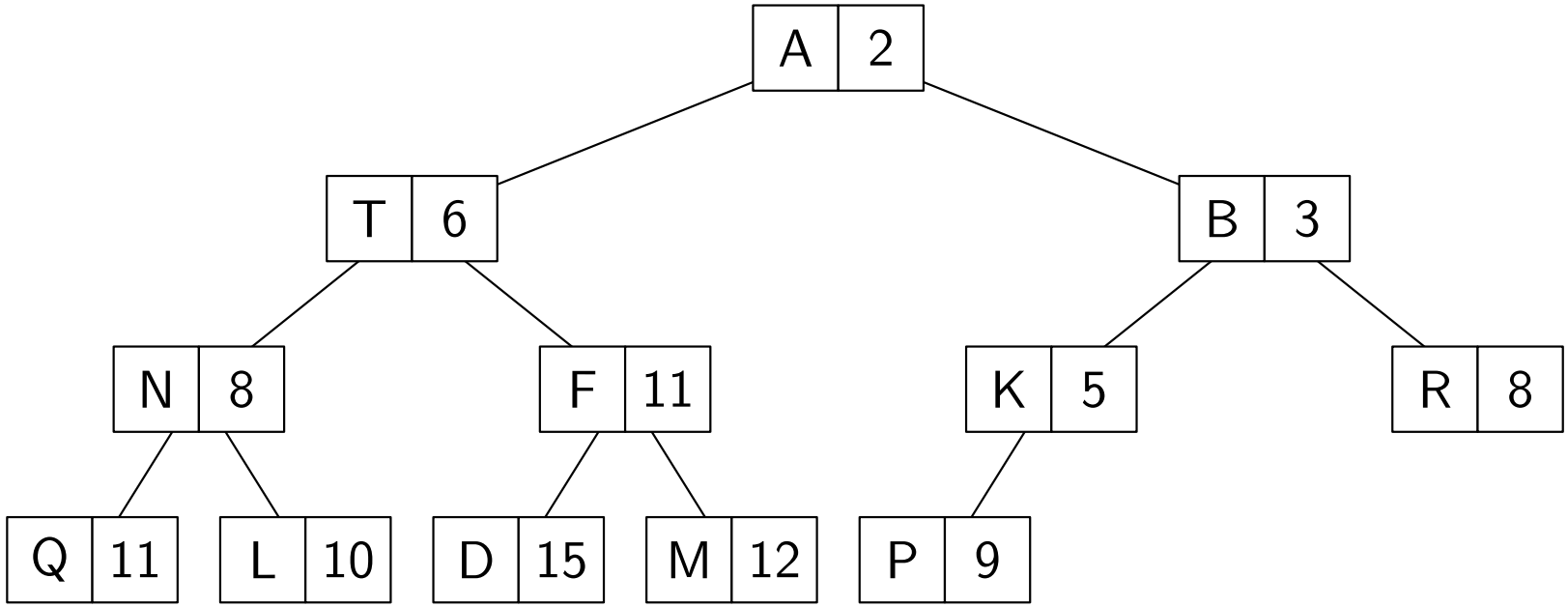
- $\langle$ 不変条件 $\rangle$：変化が最小の 2 辺組を取りだす操作　　$O(1)$ 時間
- $\langle$ 不変条件 $\rangle$：指定した 2 辺組に対する要素を削除する操作
  $O(\log n)$ 時間
- $\langle$ 不変条件 $\rangle$：指定した 2 辺組に対する要素を挿入する操作
  $O(\log n)$ 時間

$\rightsquigarrow$ 単純に二分ヒープを用いればよい (詳細は演習問題)

目次

二分ヒープの定義 (1)

二分ヒープは完全二分木で，一番下の層は左詰め



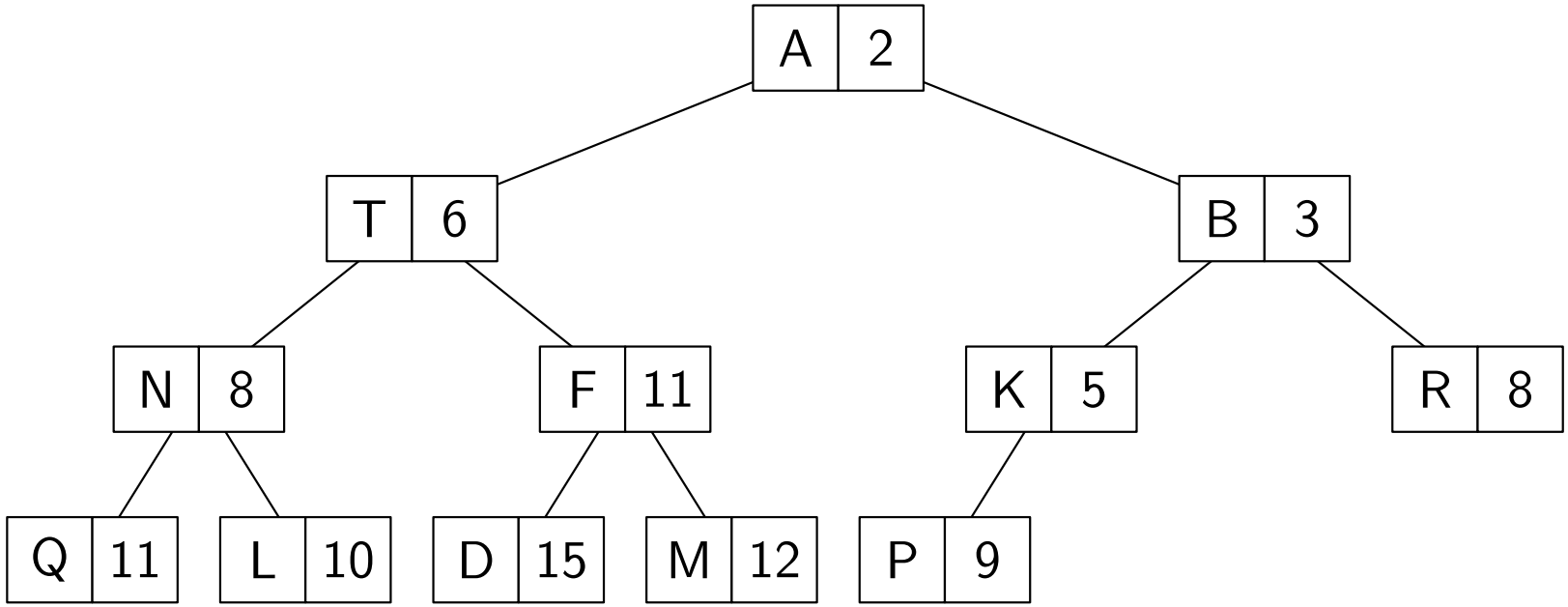
各ノードには | キー | 値 | **が蓄えられる (キーはすべて異なると仮定)**

完全二分木の重要な性質

ノード数 $= n \Rightarrow$ **高さ** $= \lfloor \log_2 n \rfloor = O(\log n)$

二分ヒープの定義 (2)

ノードの値は次のヒープ条件を満たす



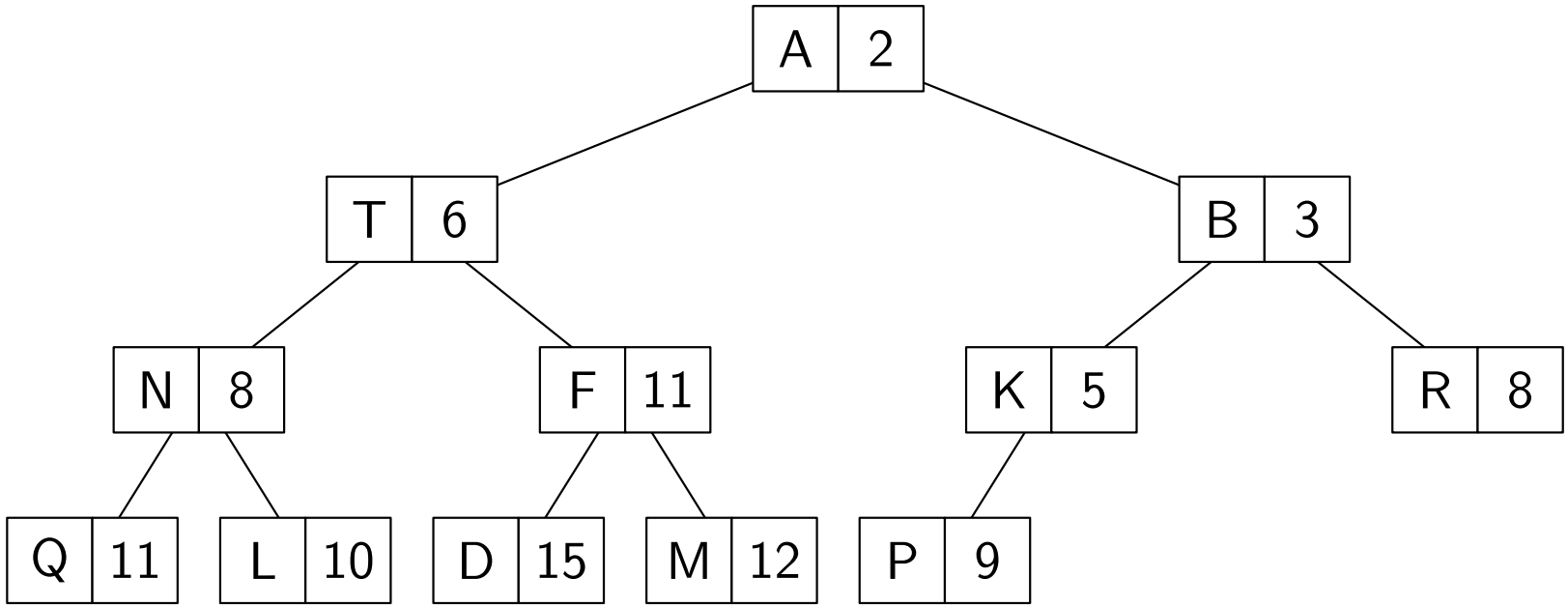
ヒープ条件

各ノードに対して

$$\text{そのノードの値} \leq \text{そのノードの子ノードの値}$$

二分ヒープの定義 (3)：アクセス

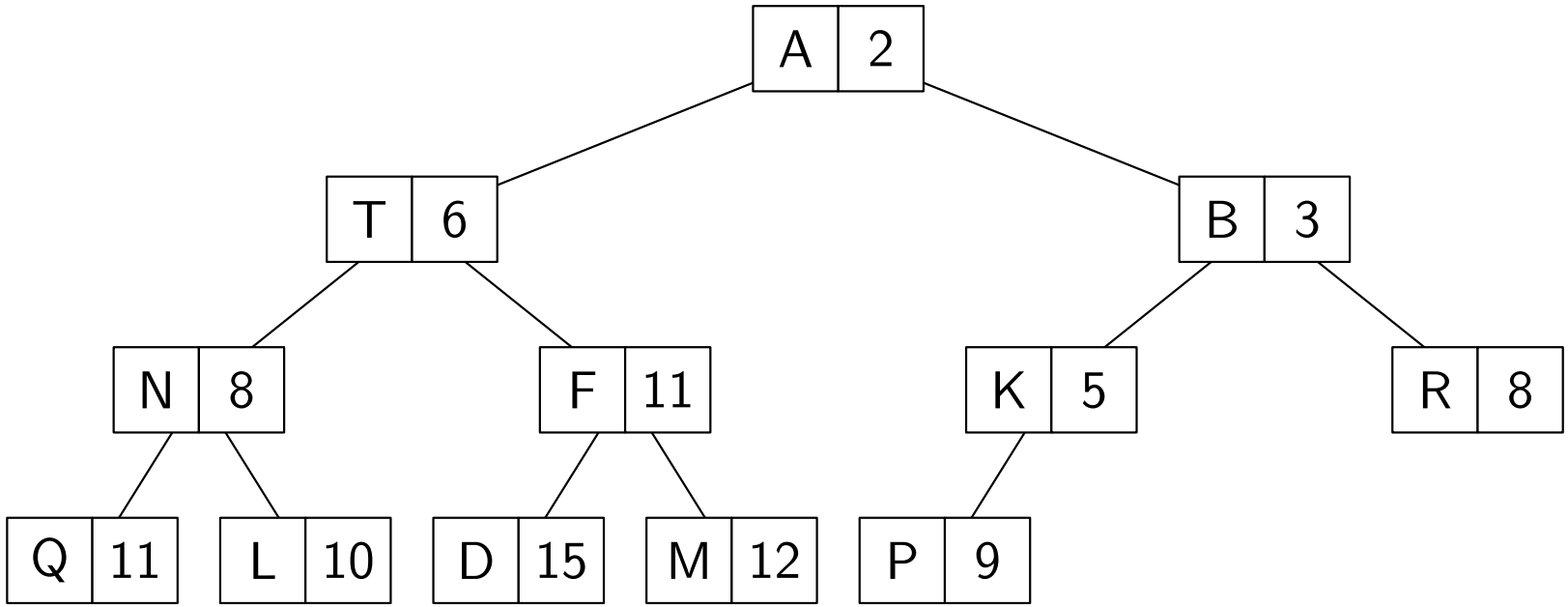
根ノードには $O(1)$ 時間でアクセスできる



キーを指定したとき，そのキーを持つノードに $O(1)$ 時間でアクセス可

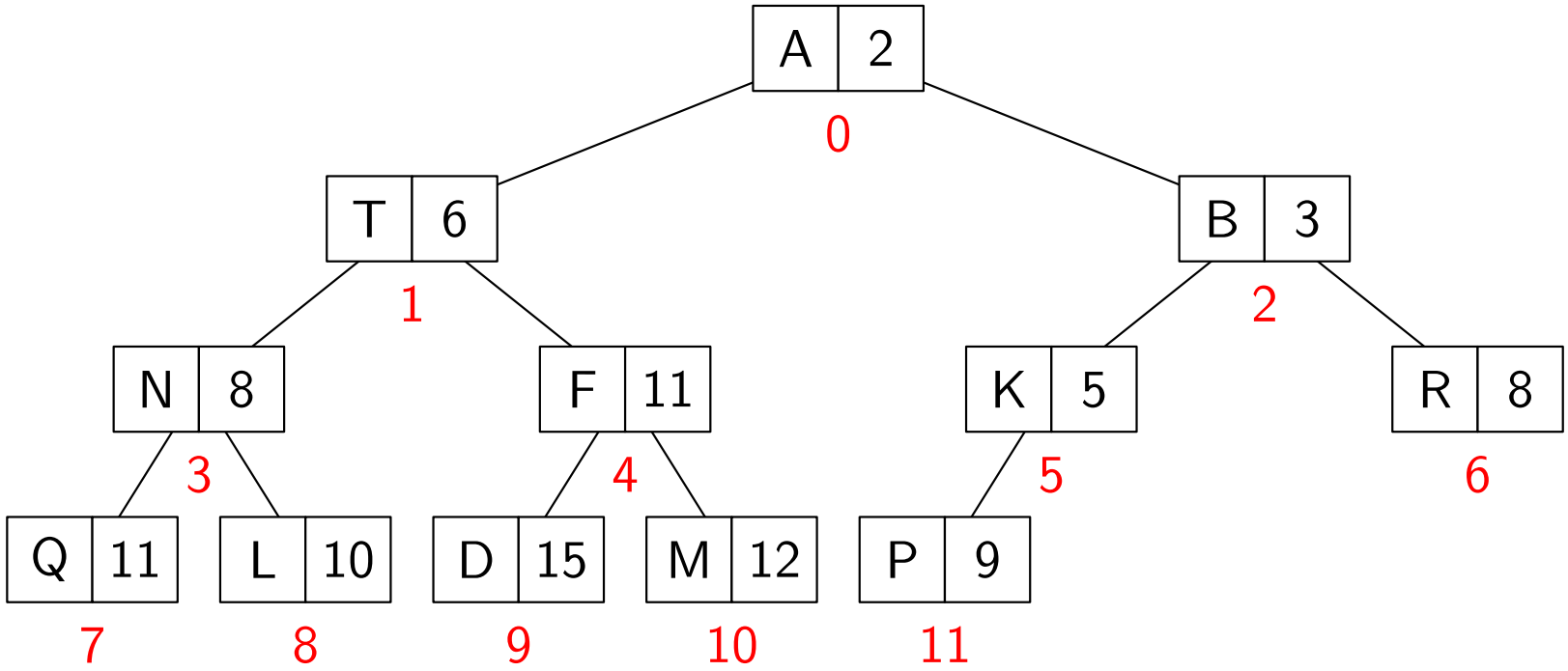
二分ヒープの定義 (4)：探索

各辺は $O(1)$ 時間で (どちらの向きにも) たどれる



二分ヒープの実装

二分ヒープは配列として実装されることが多い

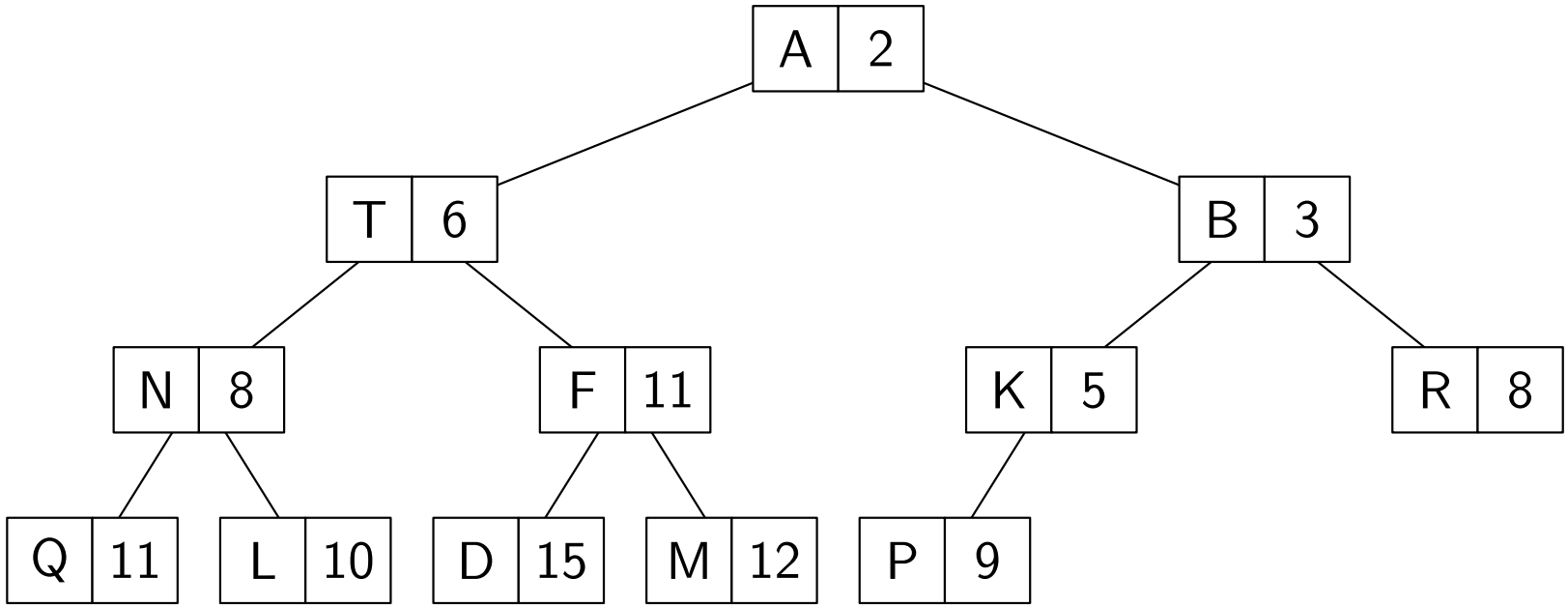


このとき，添字が i であるノードに対して，

- その親ノードの添字は $\lfloor (i-1)/2 \rfloor$
- その左子ノードの添字は $\lfloor 2i+1 \rfloor$
- その右子ノードの添字は $\lfloor 2i+2 \rfloor$

二分ヒープにおける操作 (1)：最小値発見

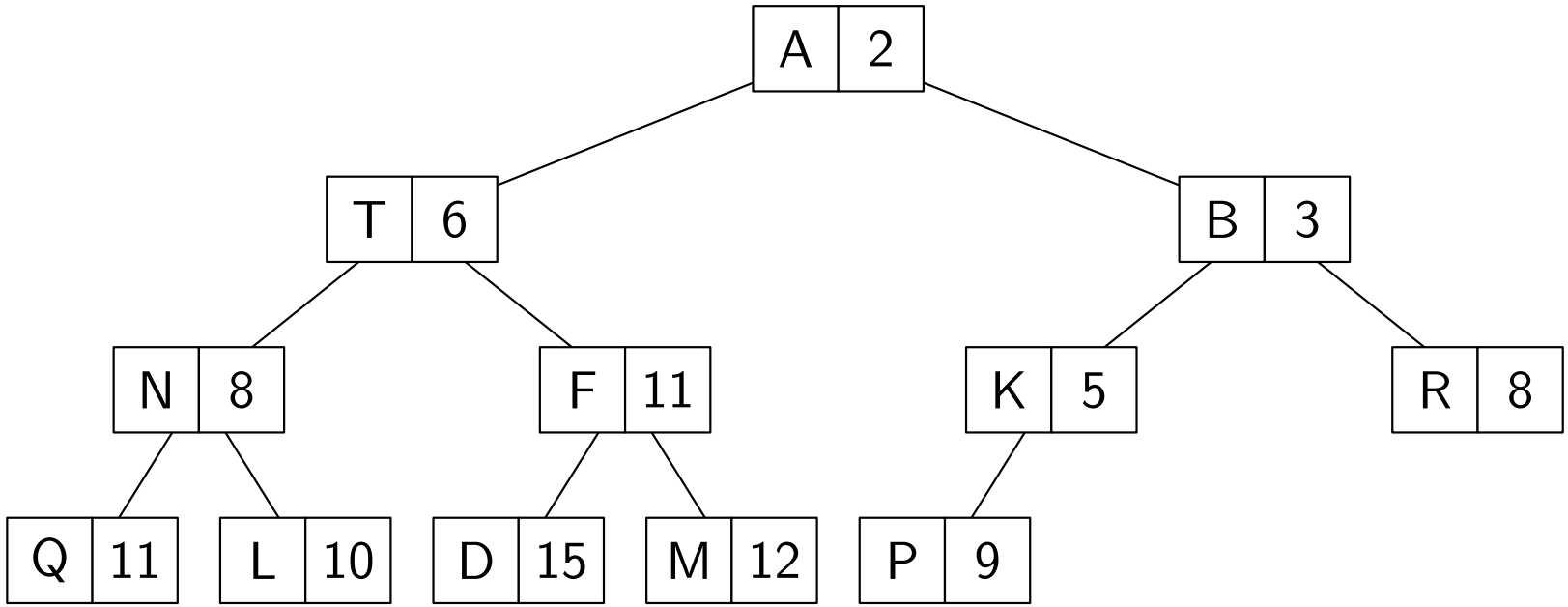
値が最小であるノードは $O(1)$ 時間で発見できる



それは根ノード

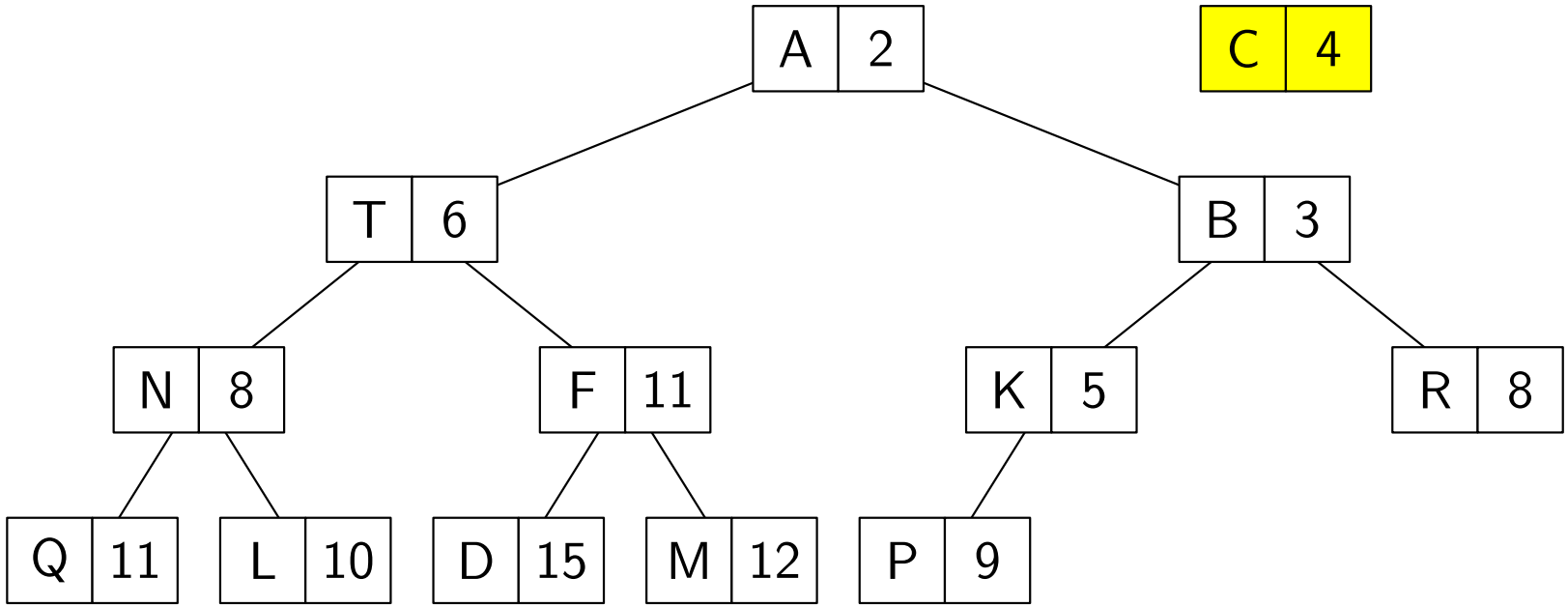
二分ヒープにおける操作 (2)：挿入

新しいノードの挿入は $O(\log n)$ 時間でできる



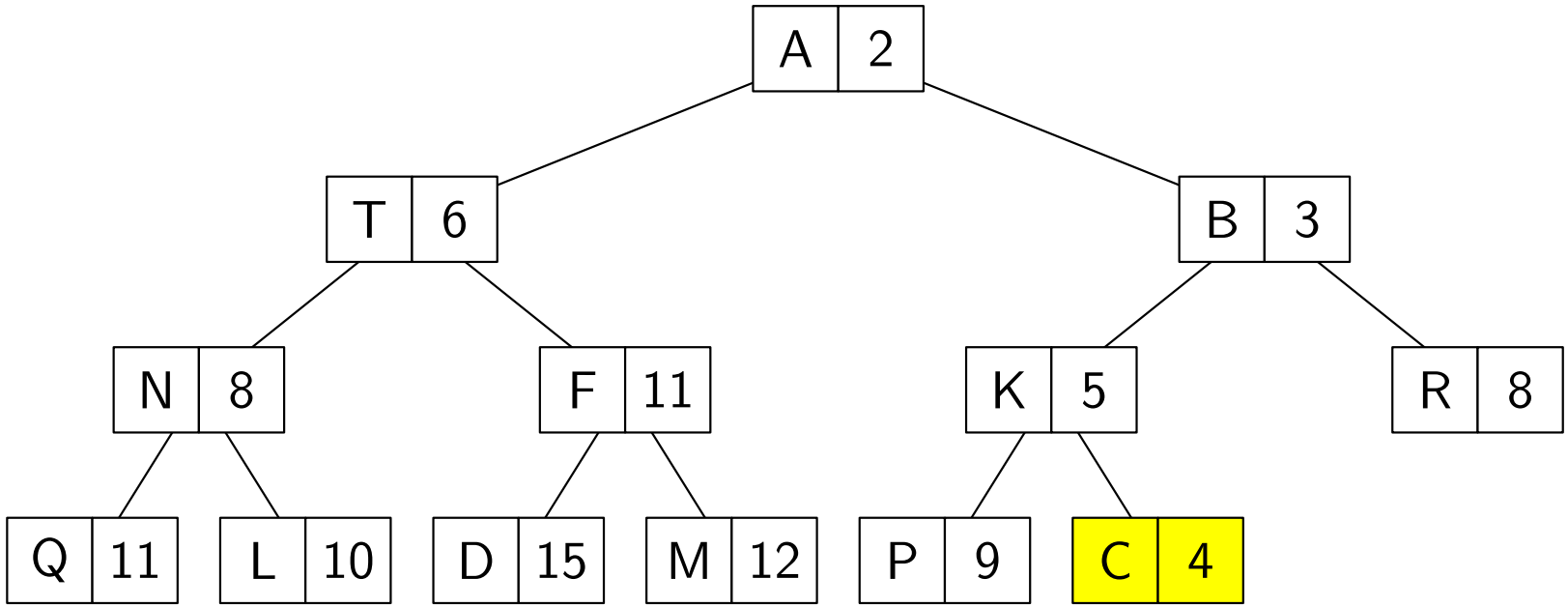
二分ヒープにおける操作 (2)：挿入

新しいノードの挿入は $O(\log n)$ 時間でできる



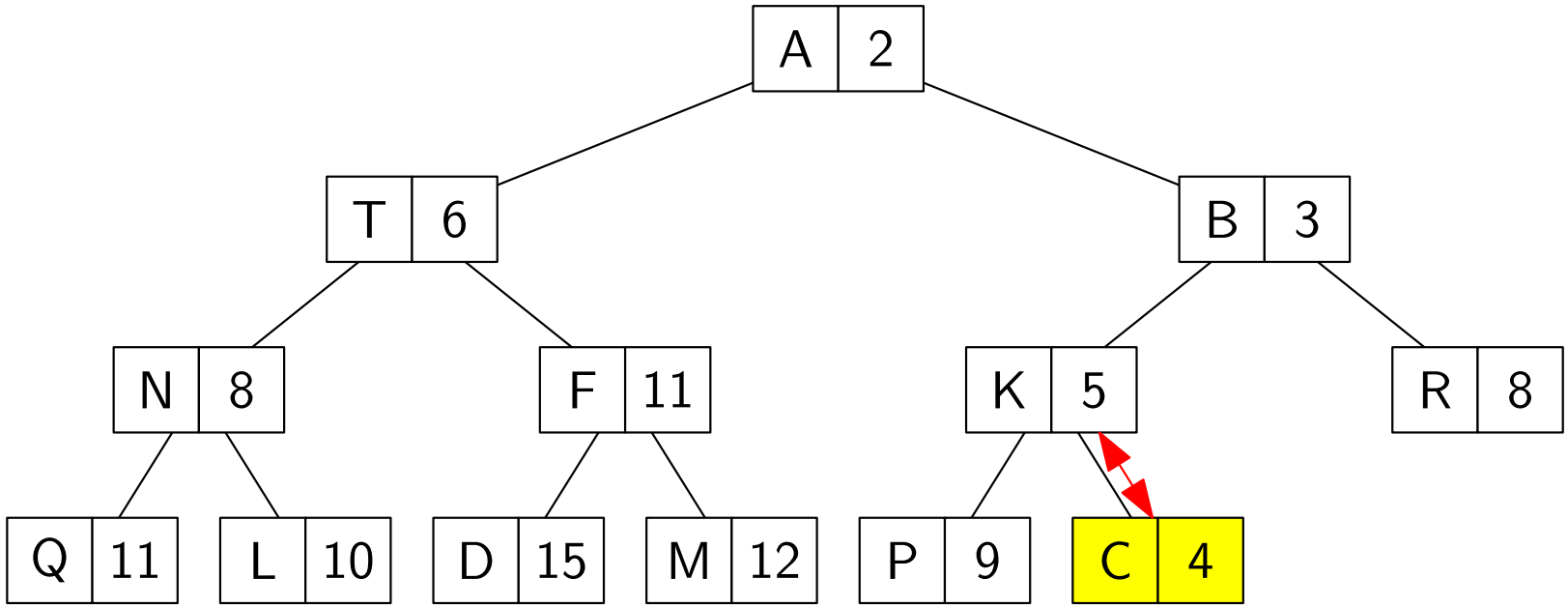
二分ヒープにおける操作 (2)：挿入

新しいノードの挿入は $O(\log n)$ 時間でできる



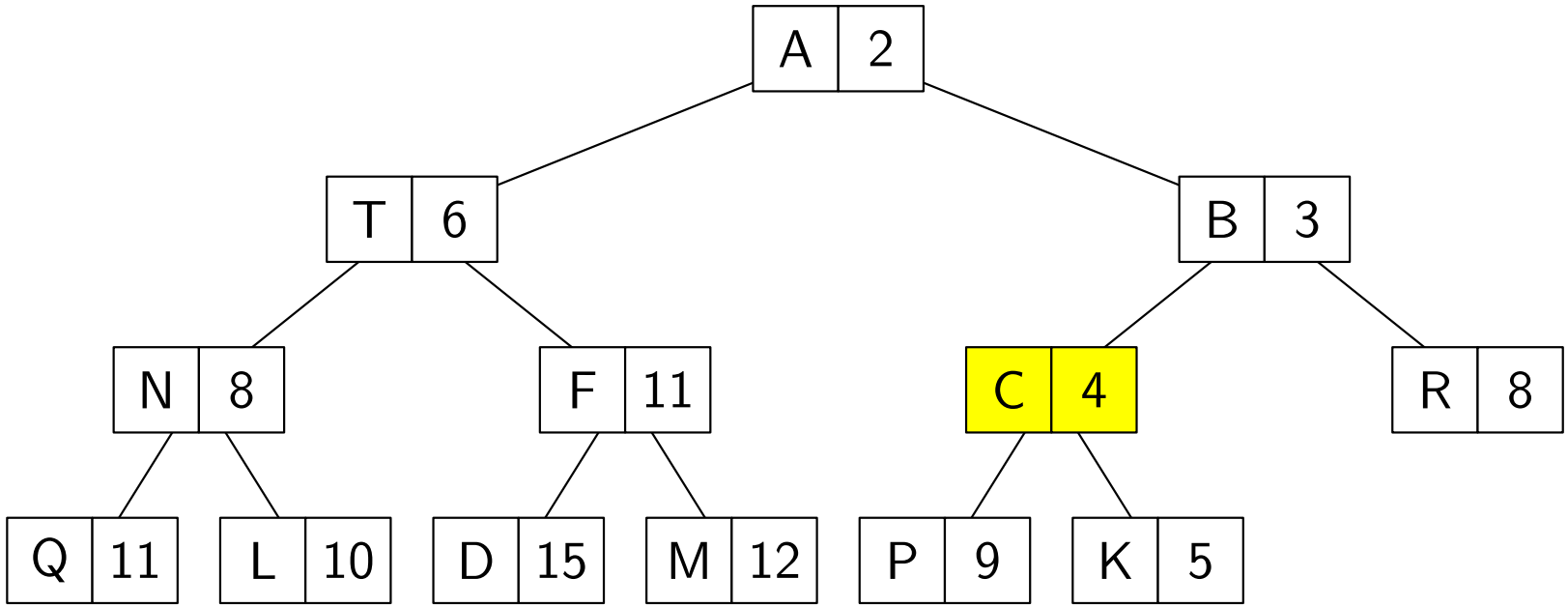
二分ヒープにおける操作 (2)：挿入

新しいノードの挿入は $O(\log n)$ 時間でできる



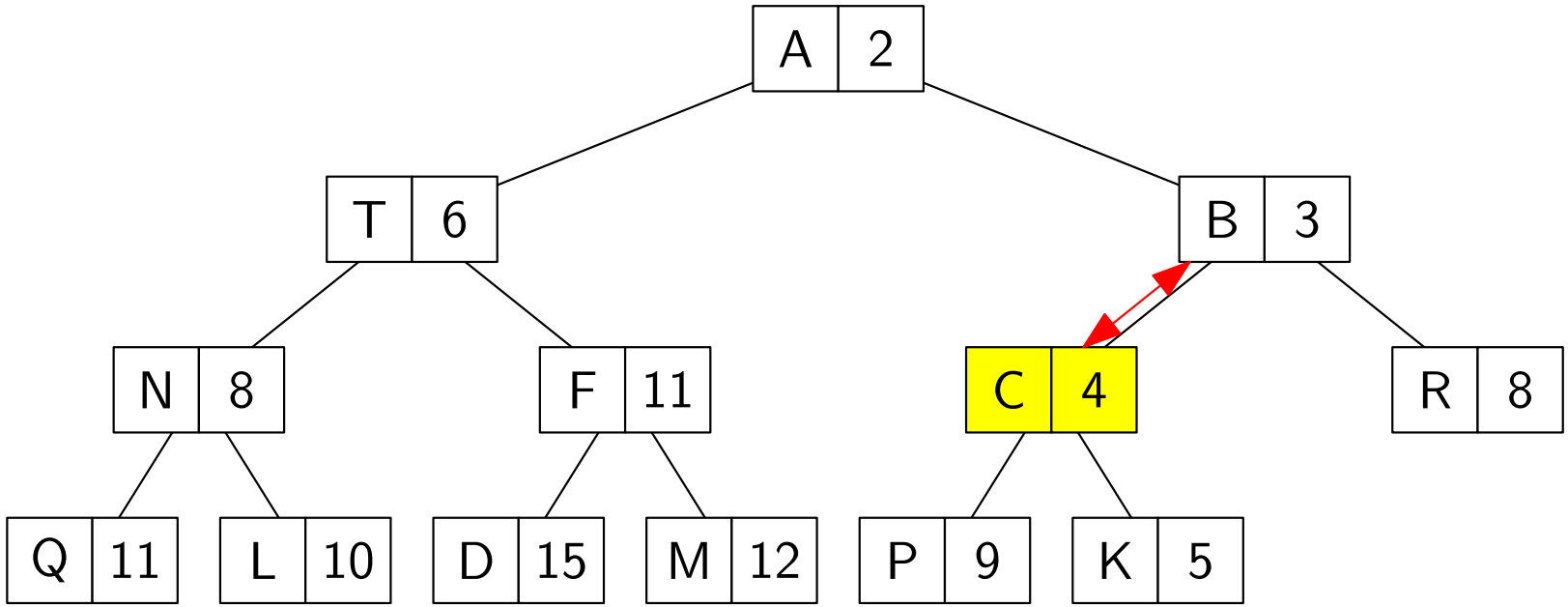
二分ヒープにおける操作 (2)：挿入

新しいノードの挿入は $O(\log n)$ 時間でできる



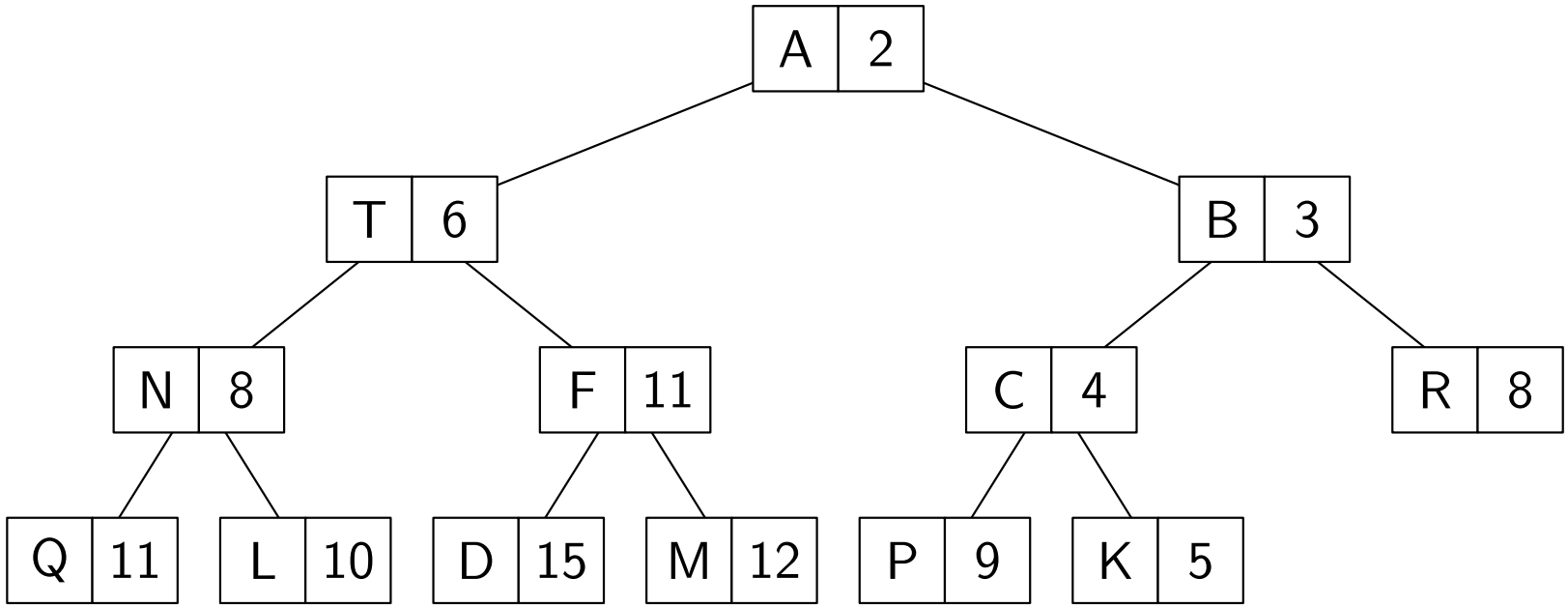
二分ヒープにおける操作 (2)：挿入

新しいノードの挿入は $O(\log n)$ 時間でできる



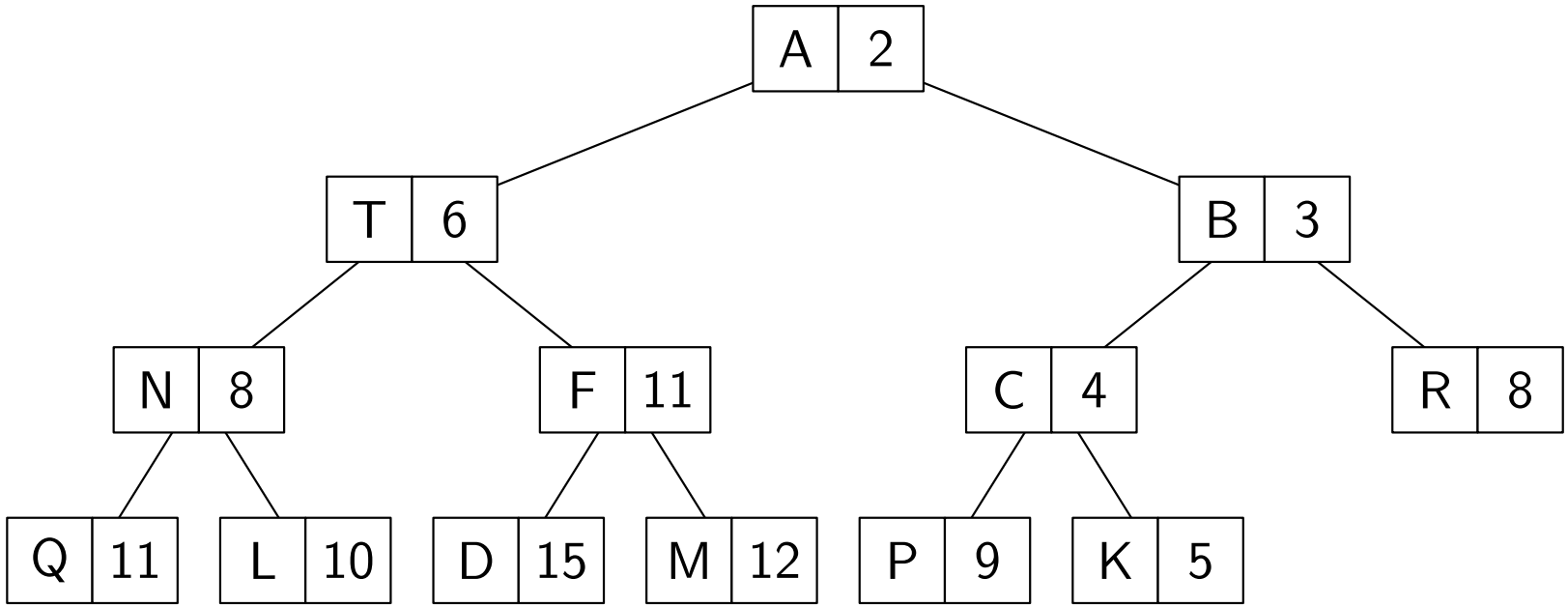
二分ヒープにおける操作 (2)：挿入

新しいノードの挿入は $O(\log n)$ 時間でできる



二分ヒープにおける操作 (2)：挿入

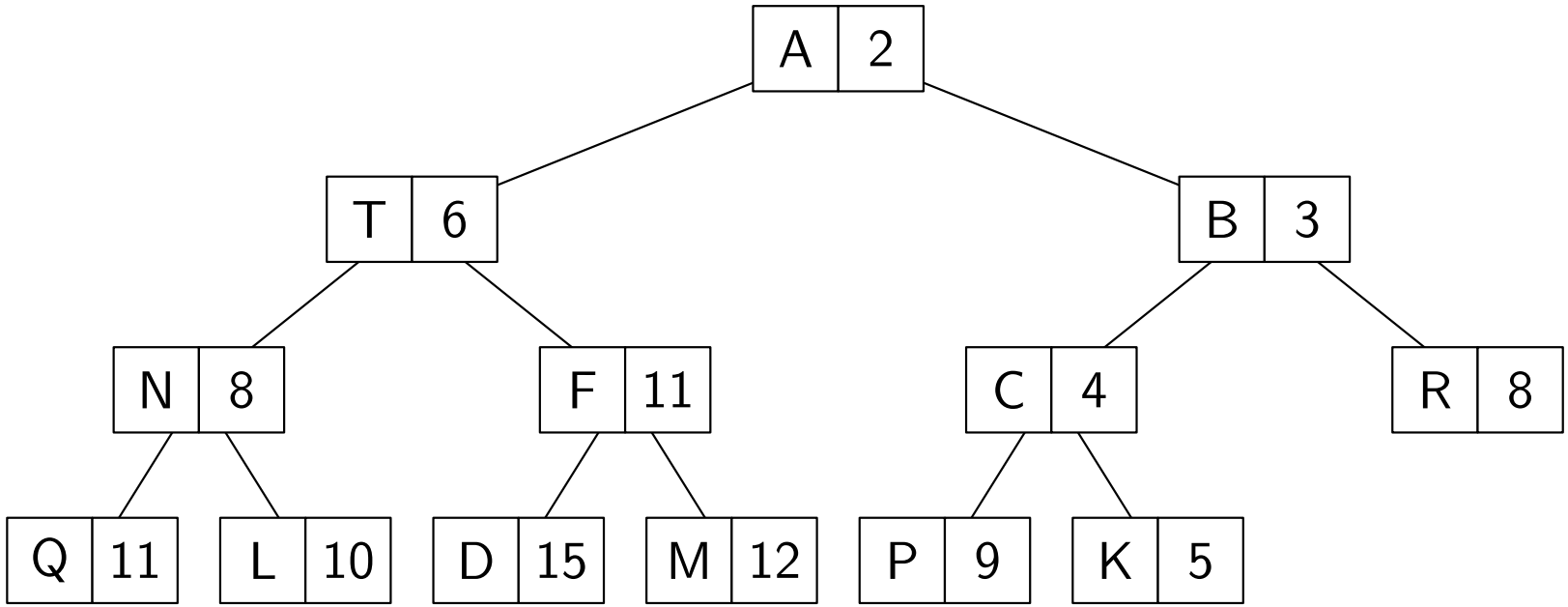
新しいノードの挿入は $O(\log n)$ 時間でできる



かかる時間は，高々高さにしか比例しないので $O(\log n)$

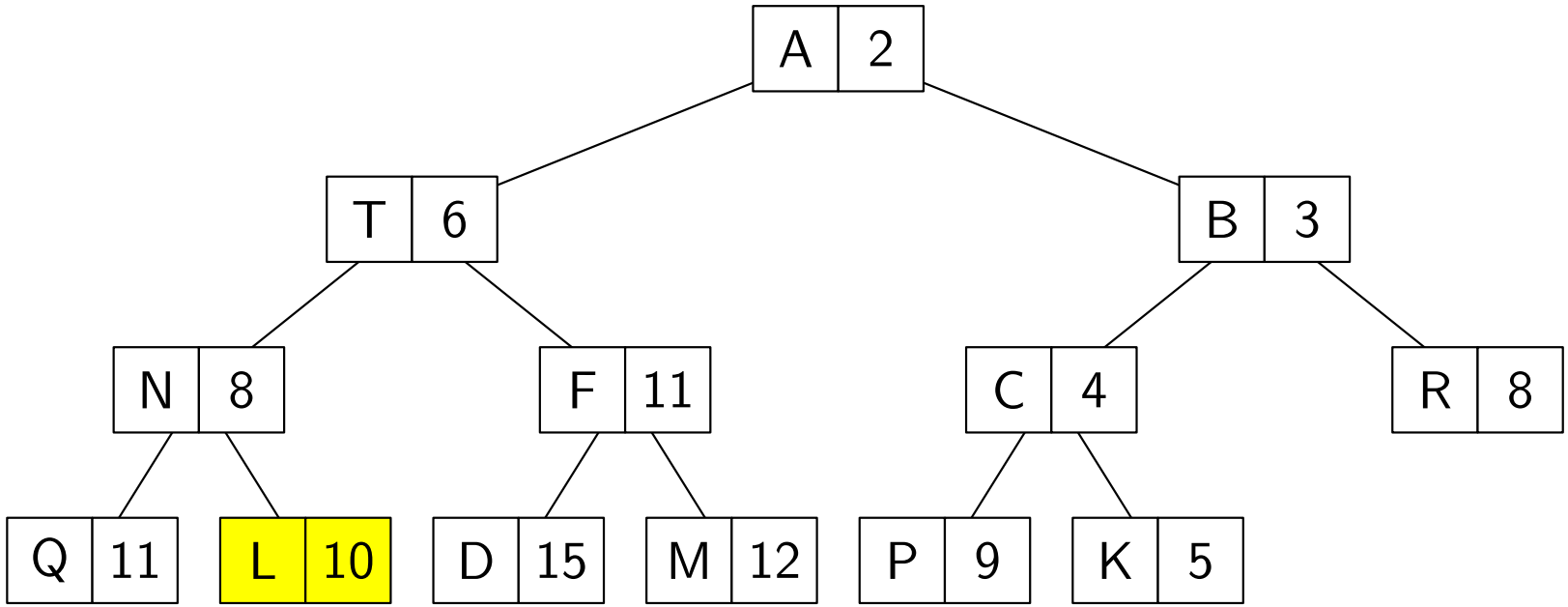
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



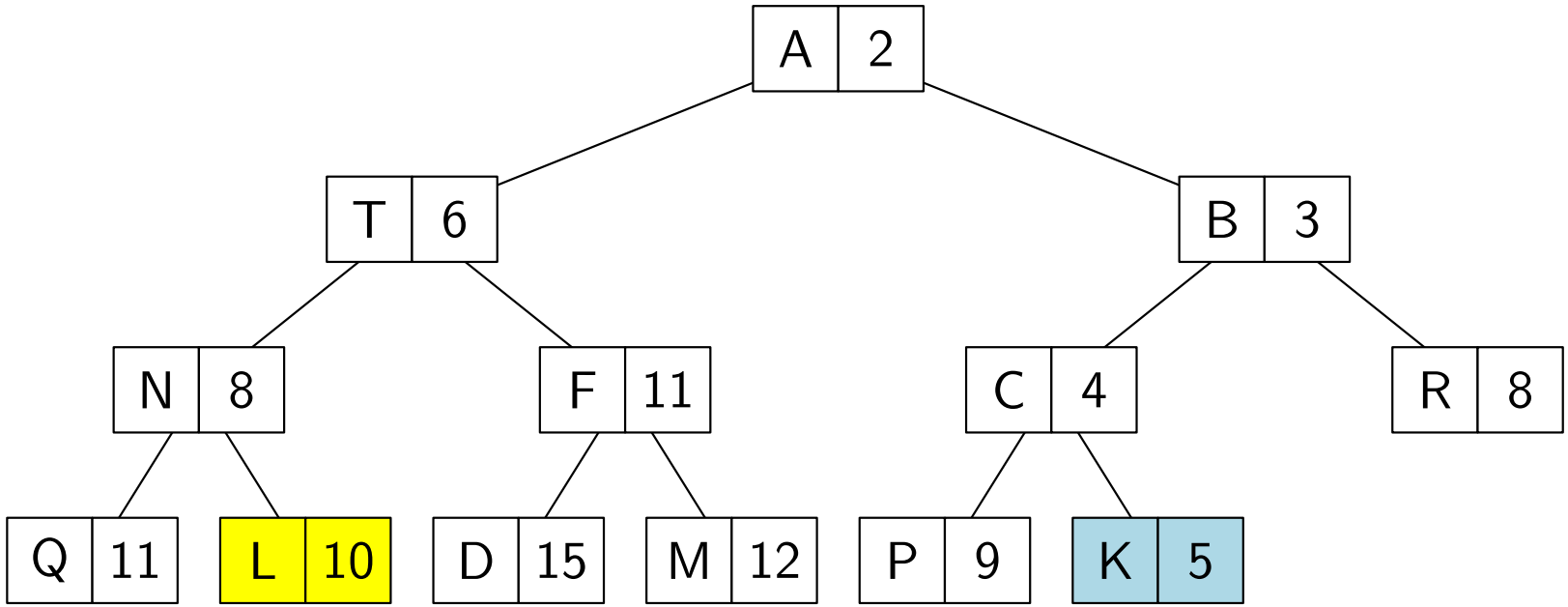
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



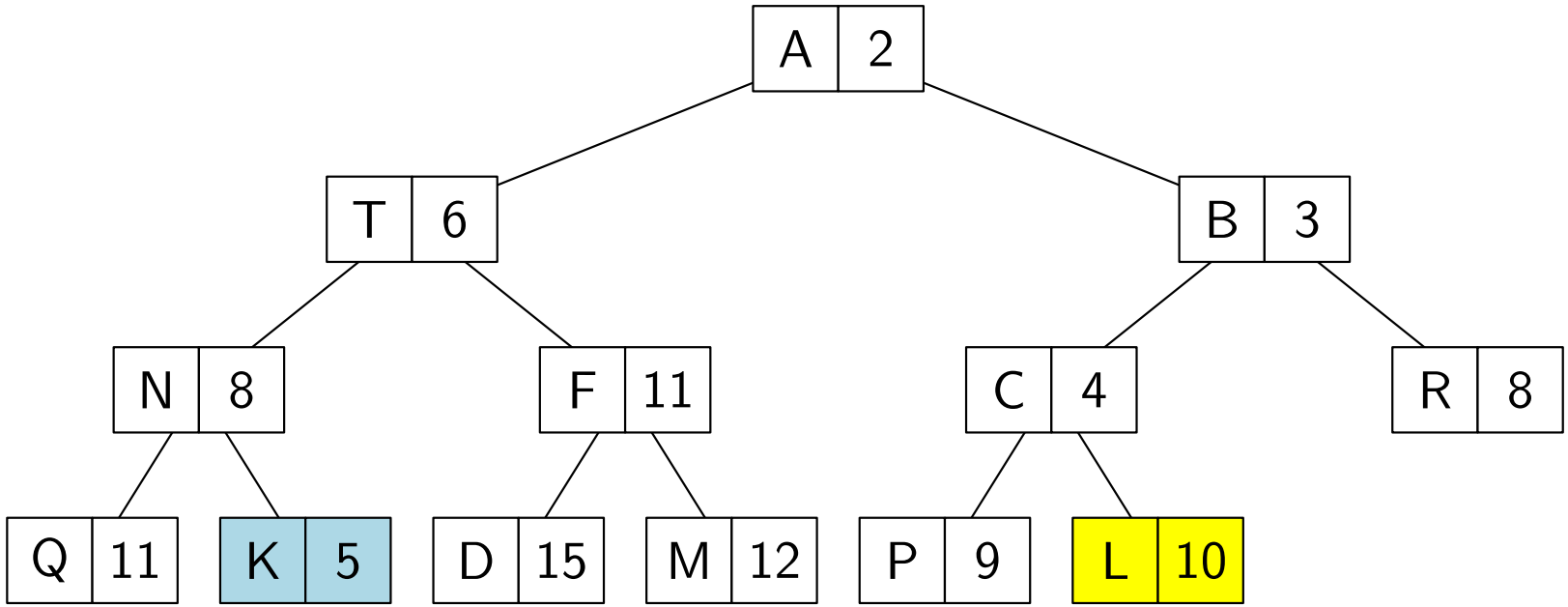
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



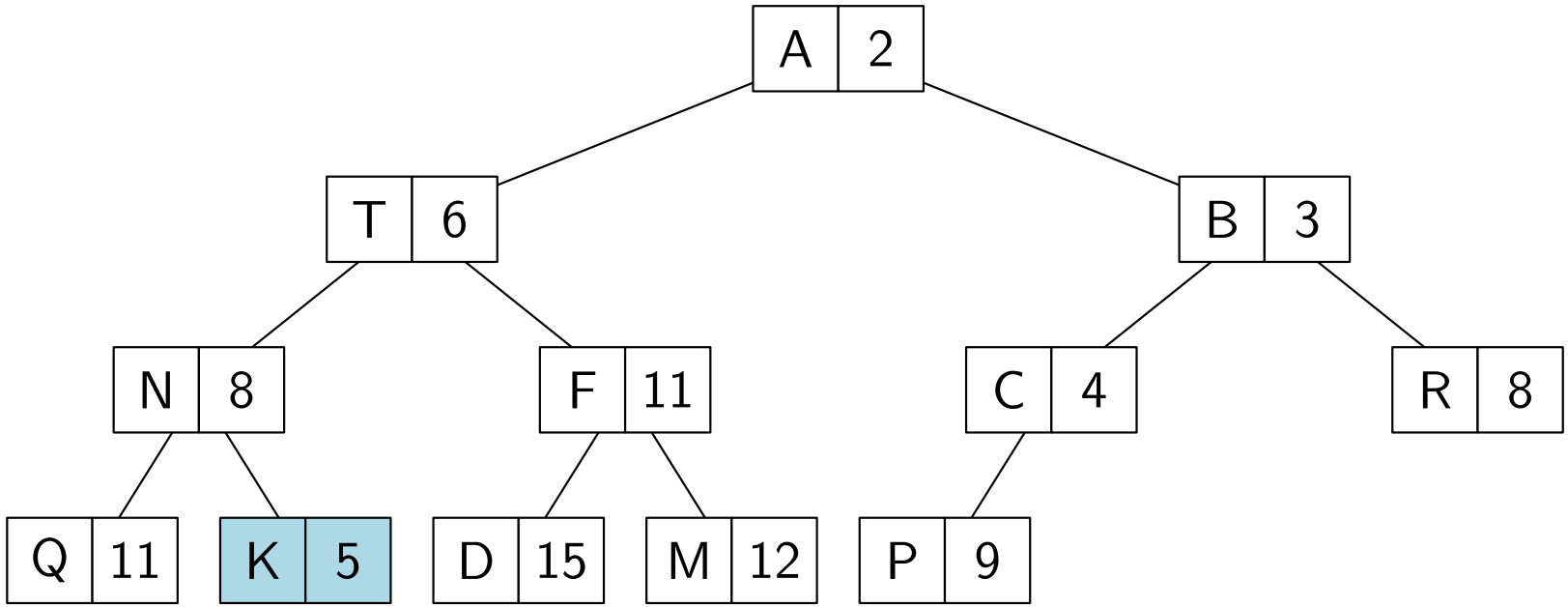
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



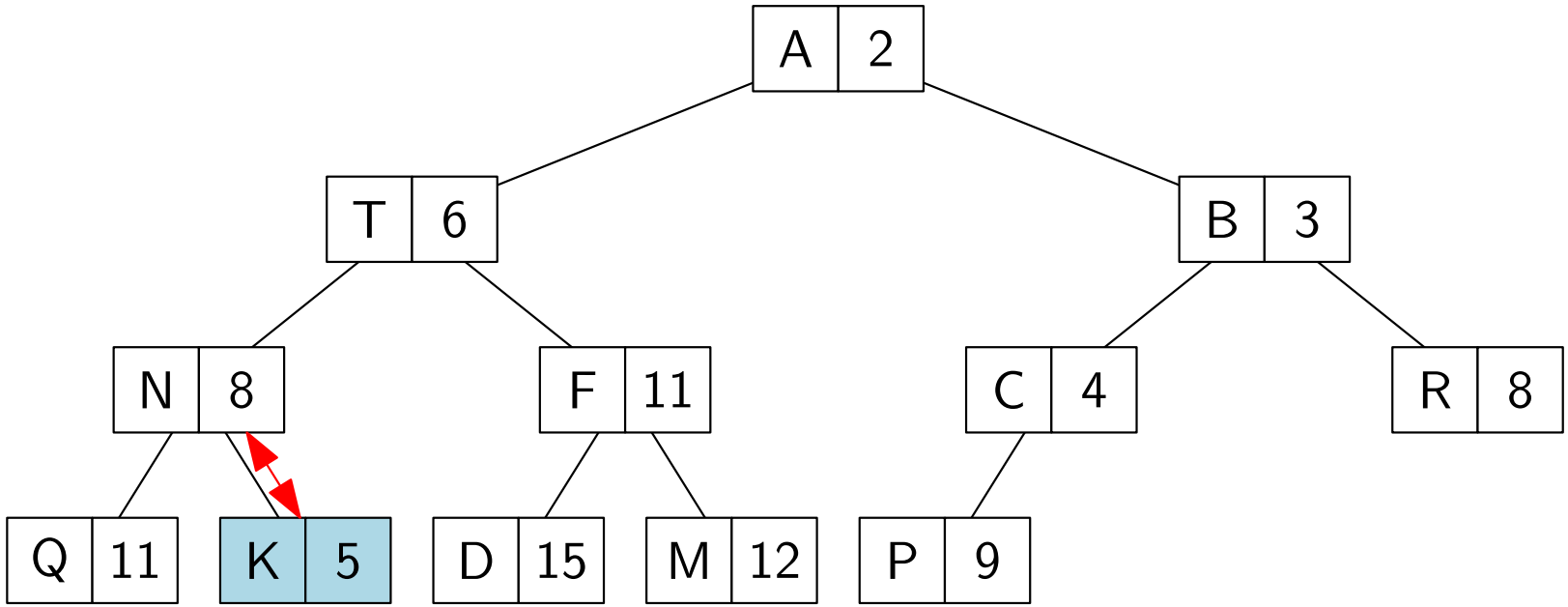
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



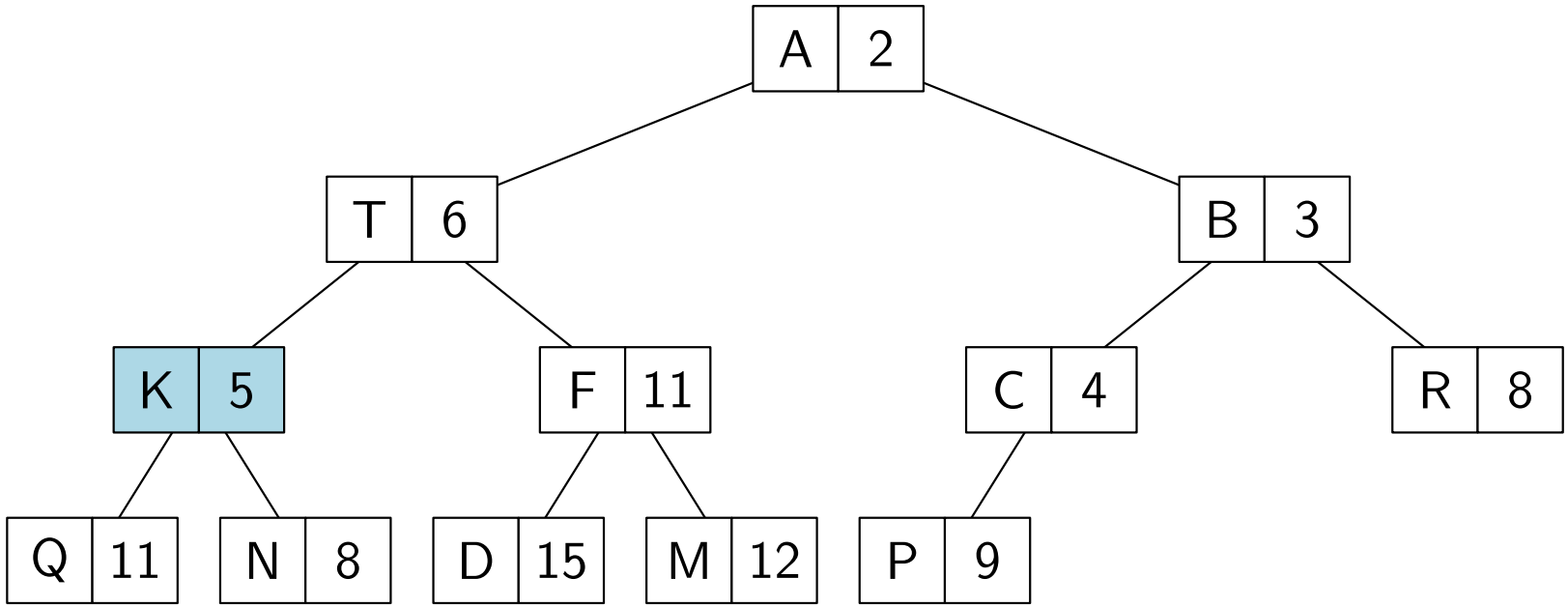
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



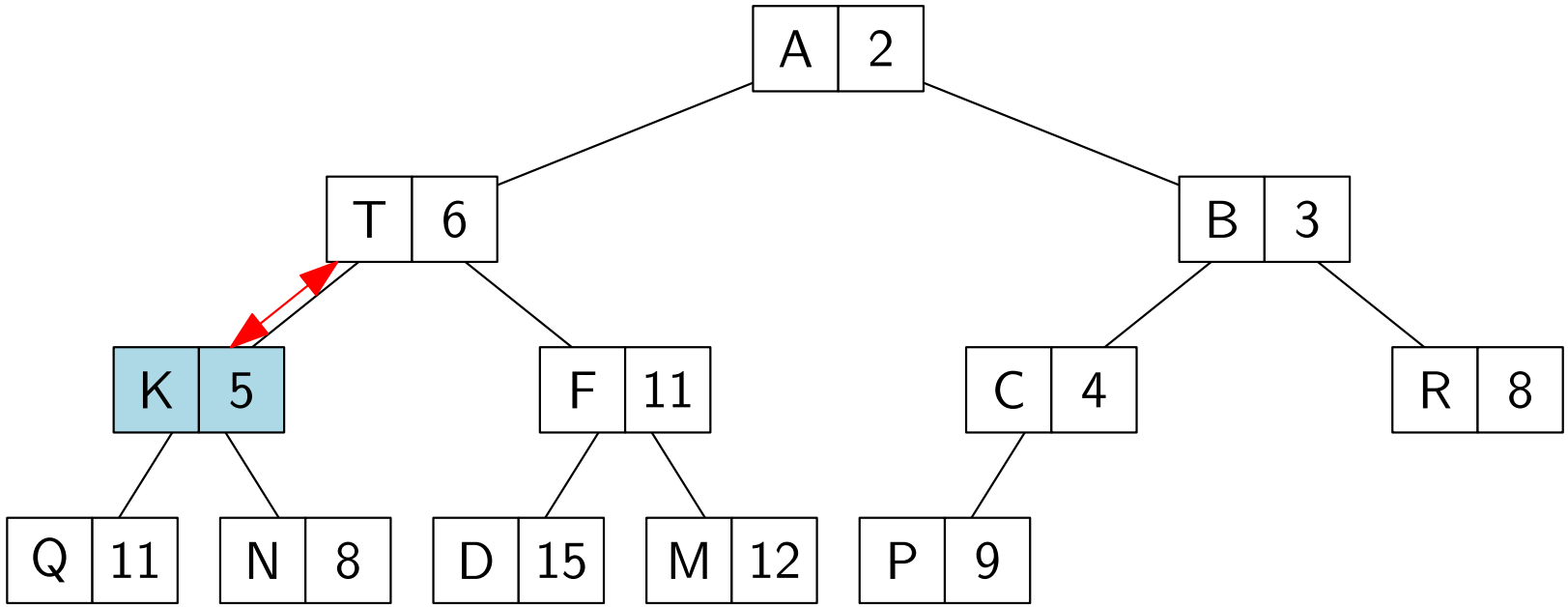
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



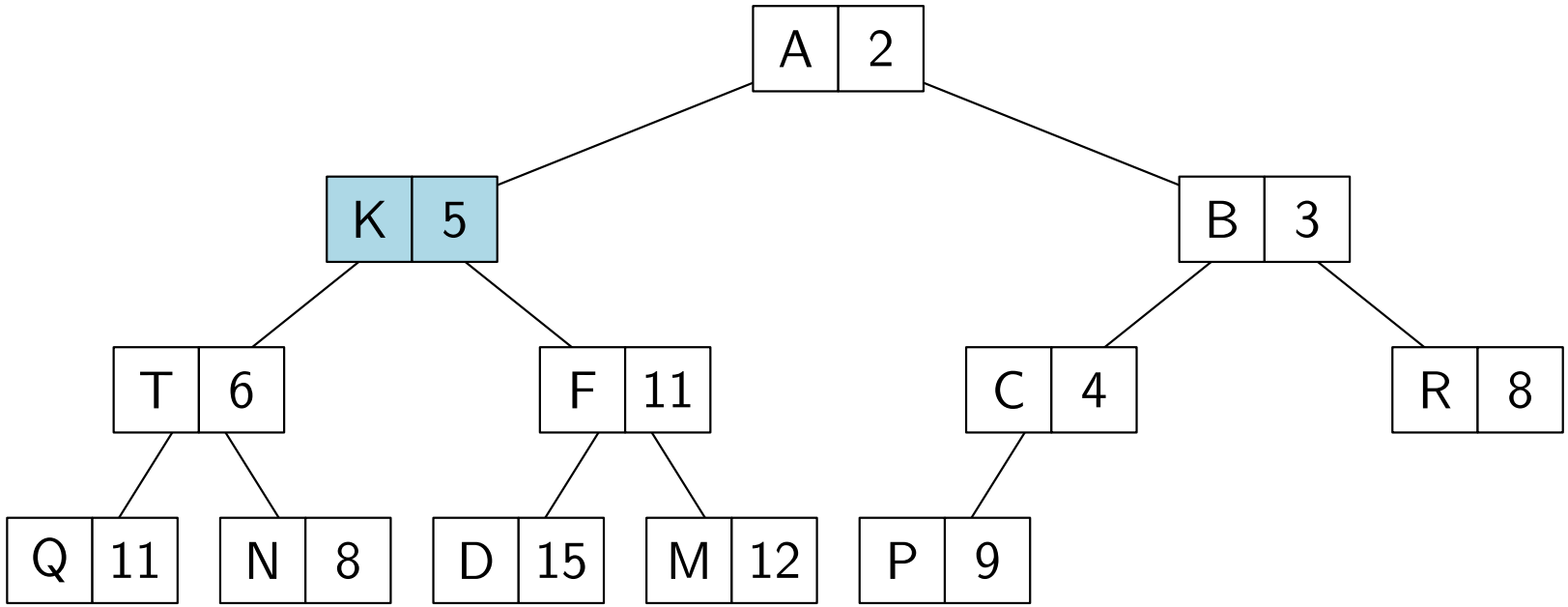
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



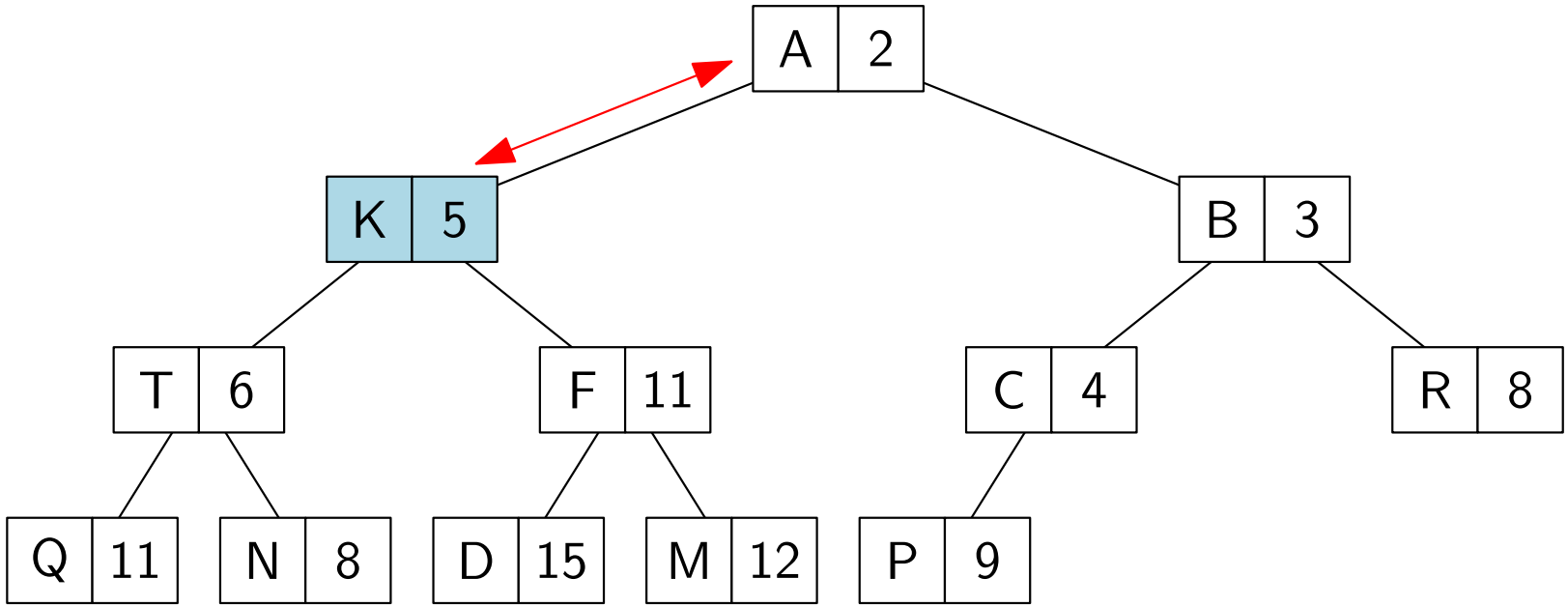
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



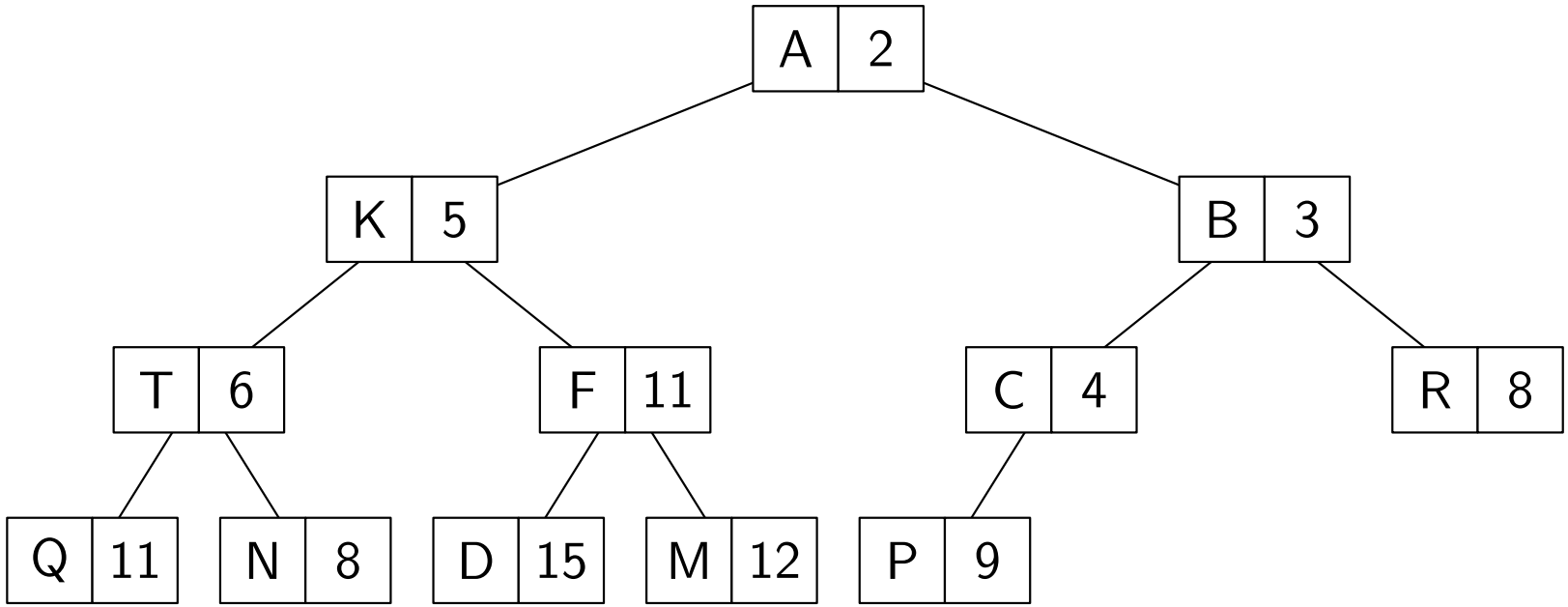
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



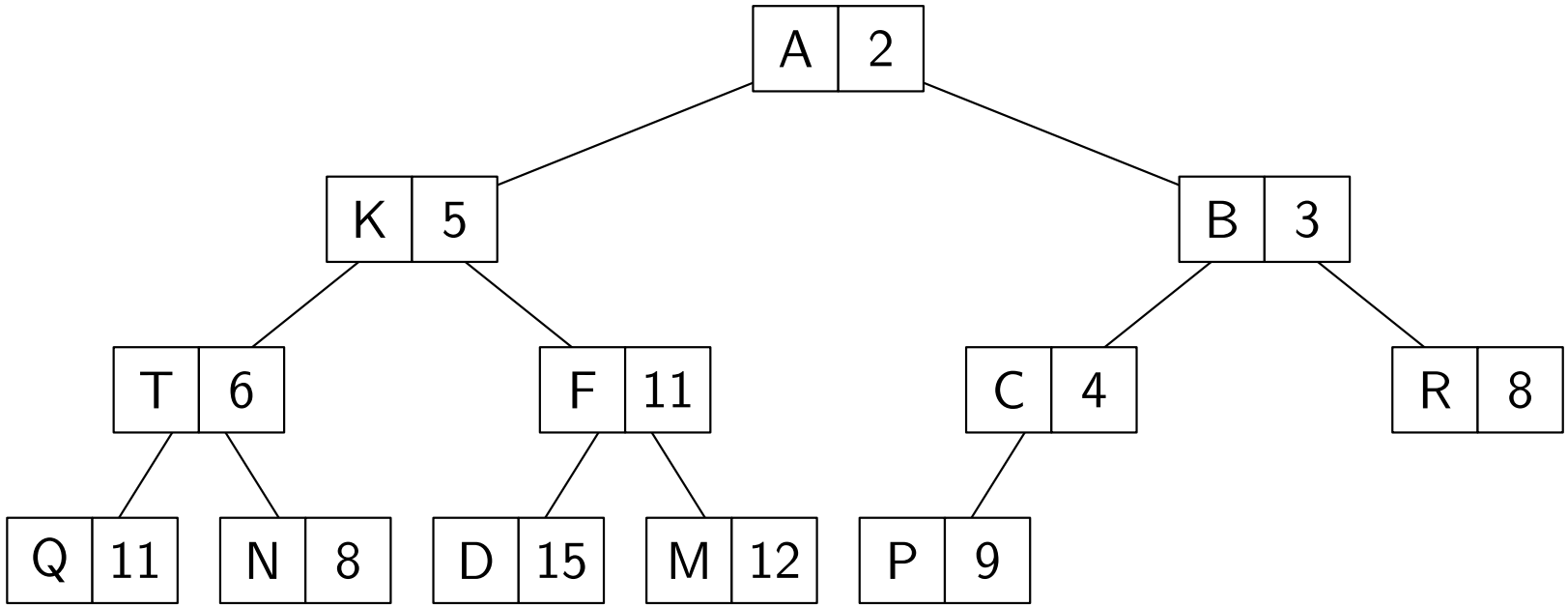
二分ヒープにおける操作 (3)：削除

新しいノードの削除は $O(\log n)$ 時間でできる



二分ヒープにおける操作 (3)：削除

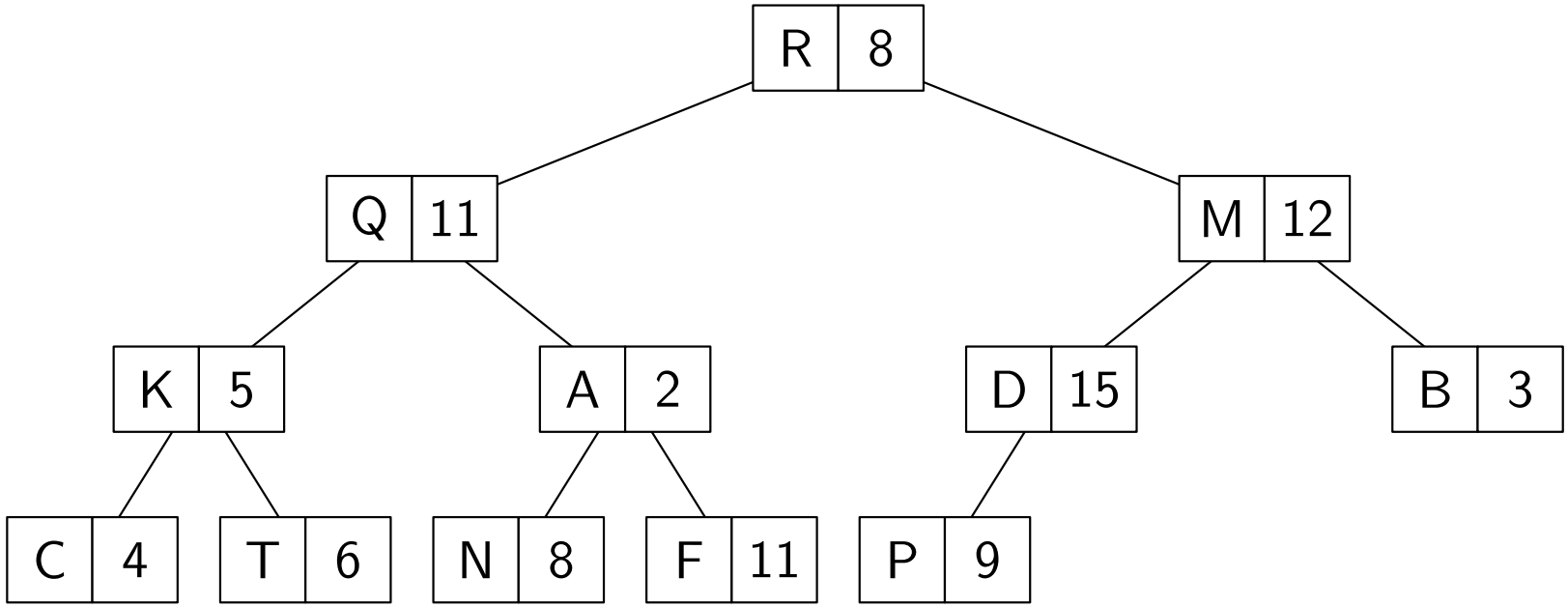
新しいノードの削除は $O(\log n)$ 時間でできる



かかる時間は，高々高さにしか比例しないので $O(\log n)$

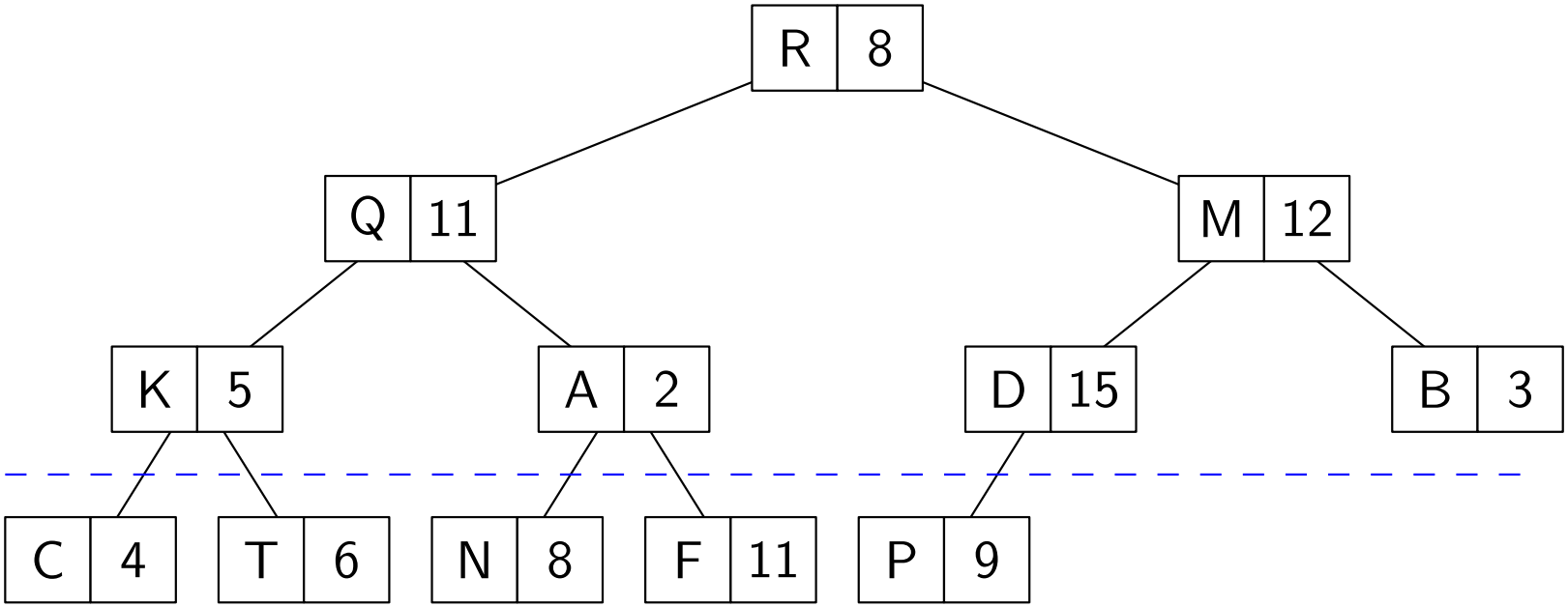
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



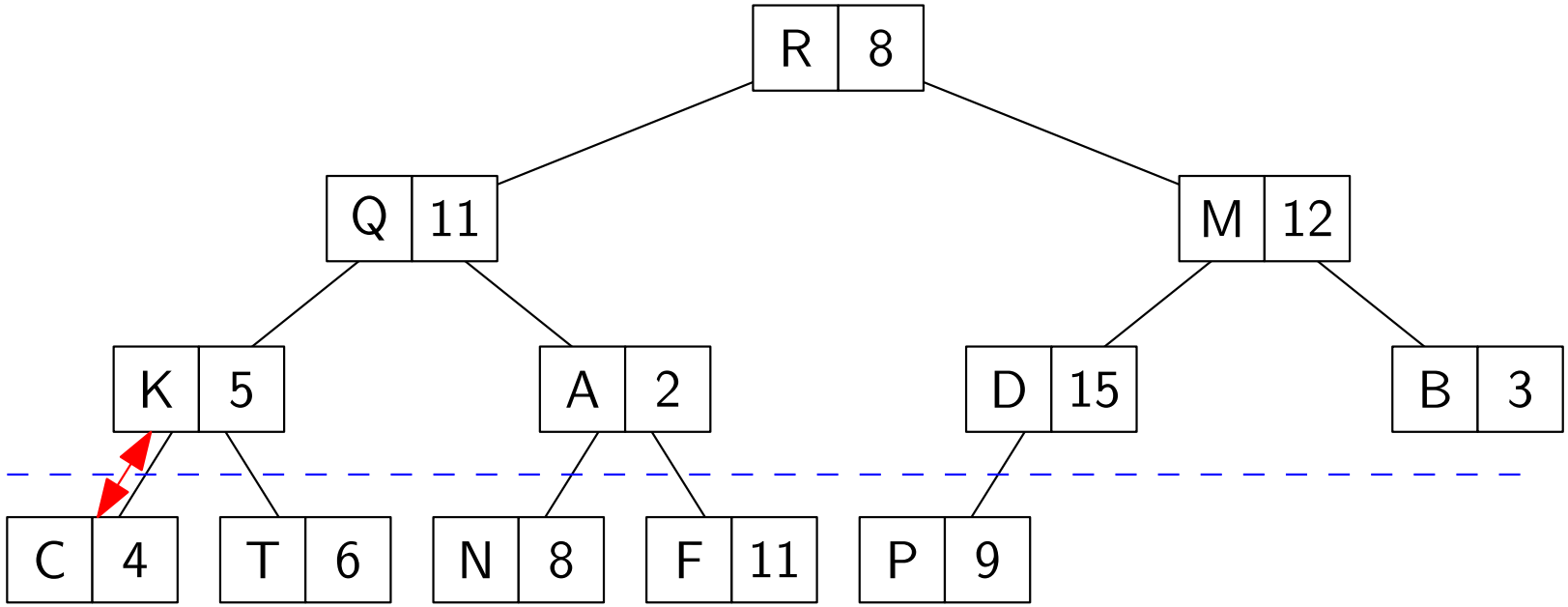
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



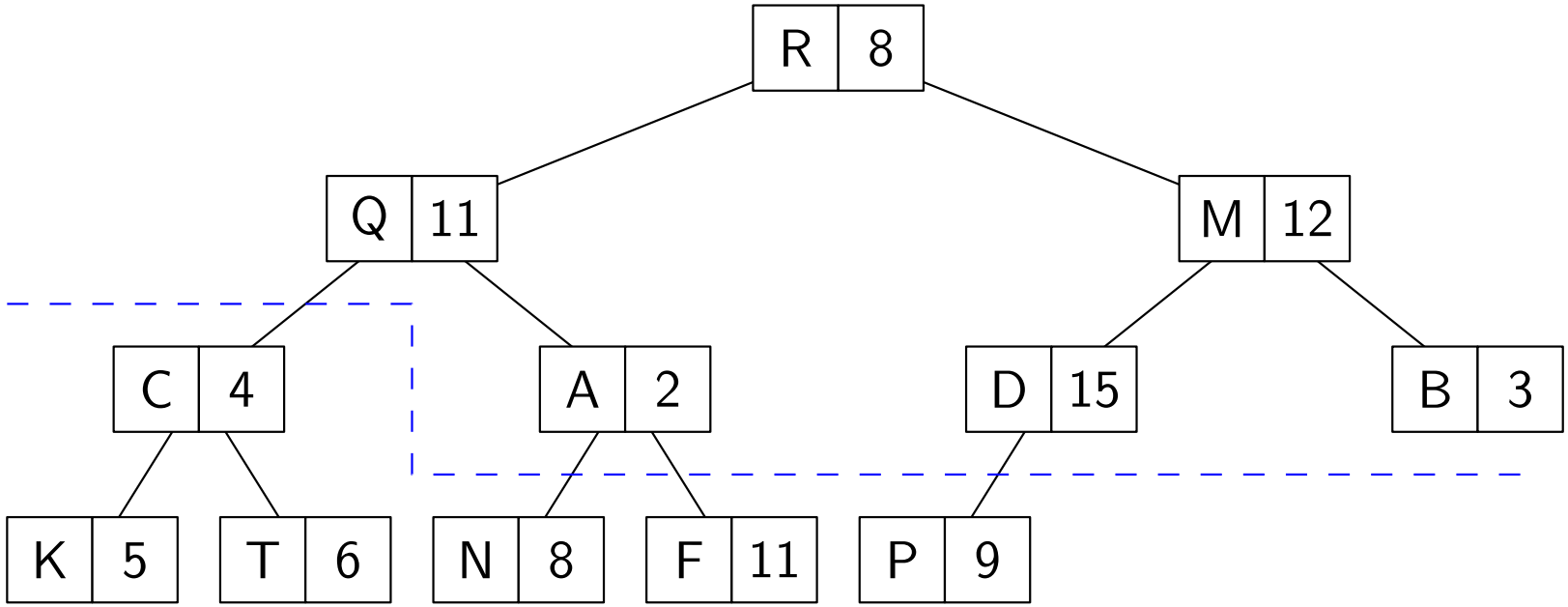
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



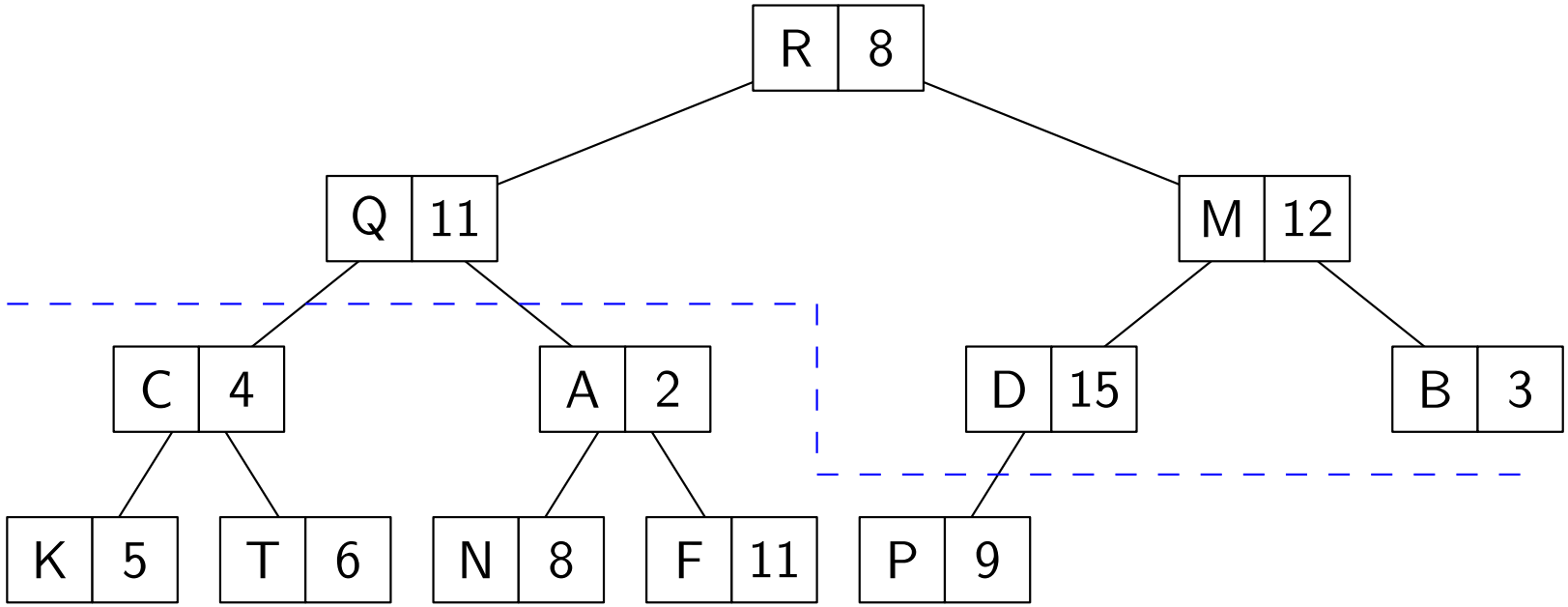
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



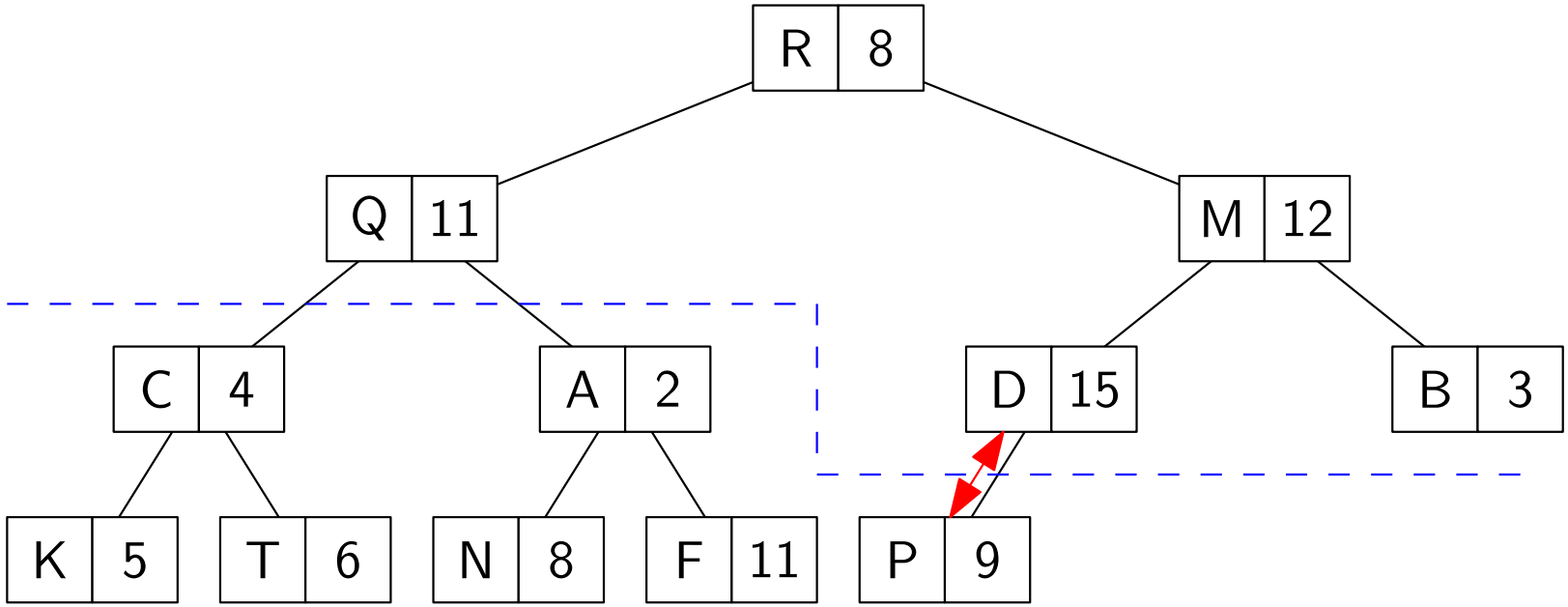
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



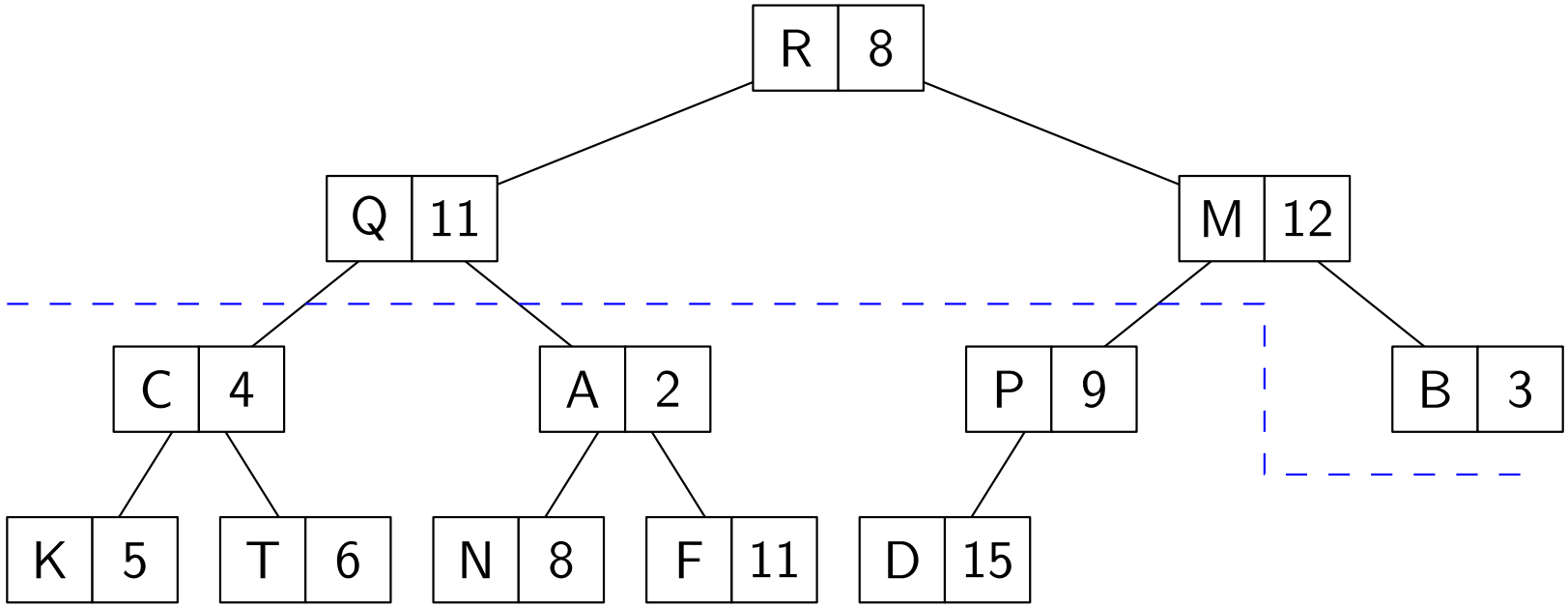
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



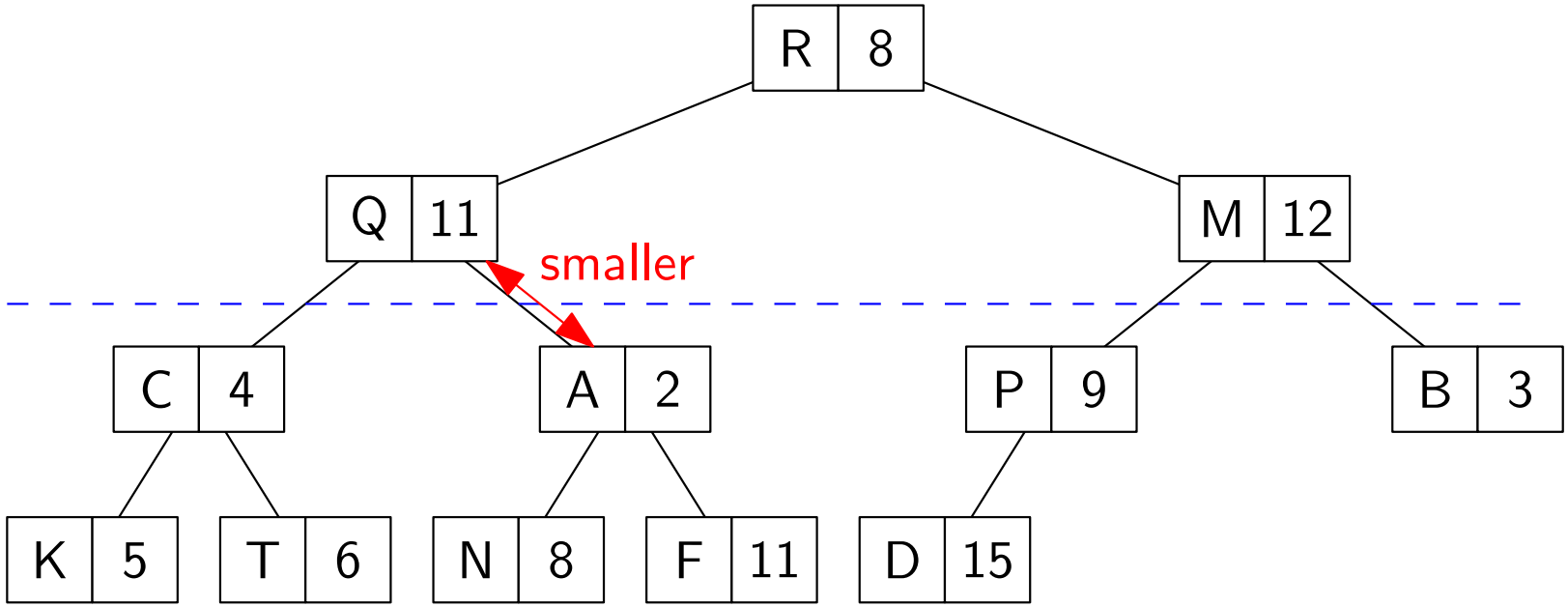
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



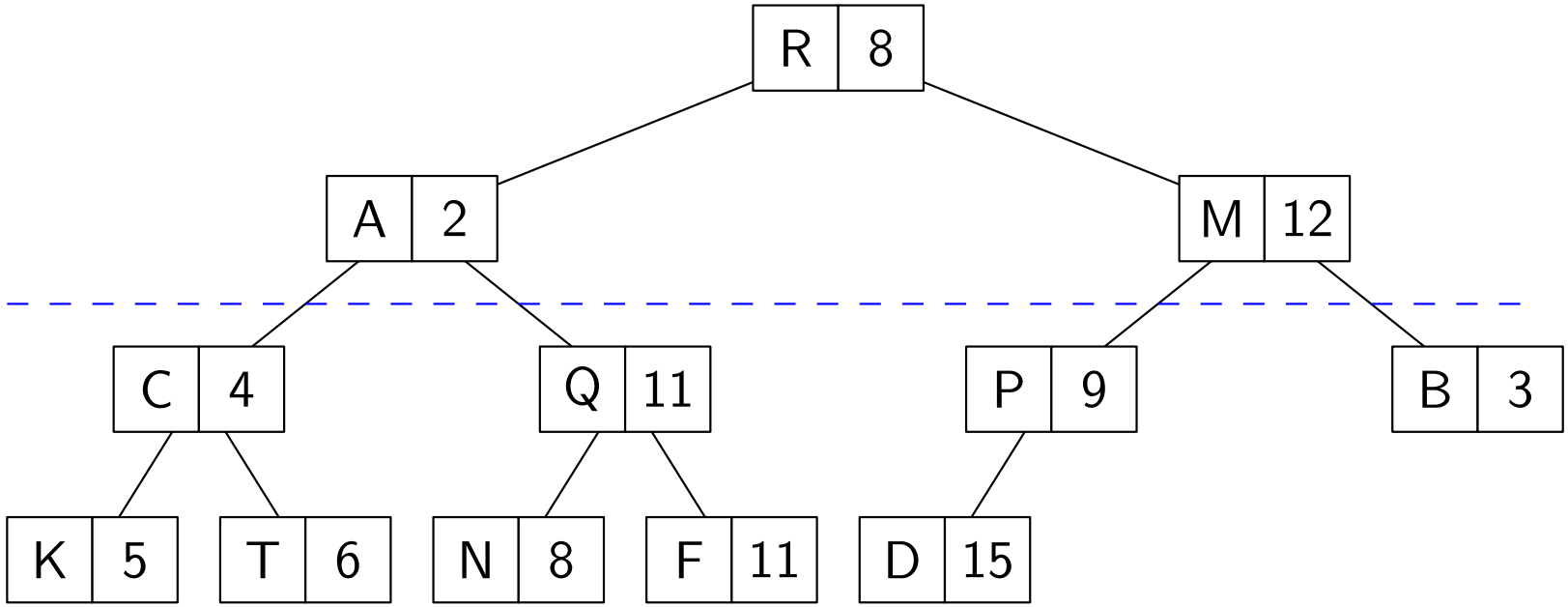
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



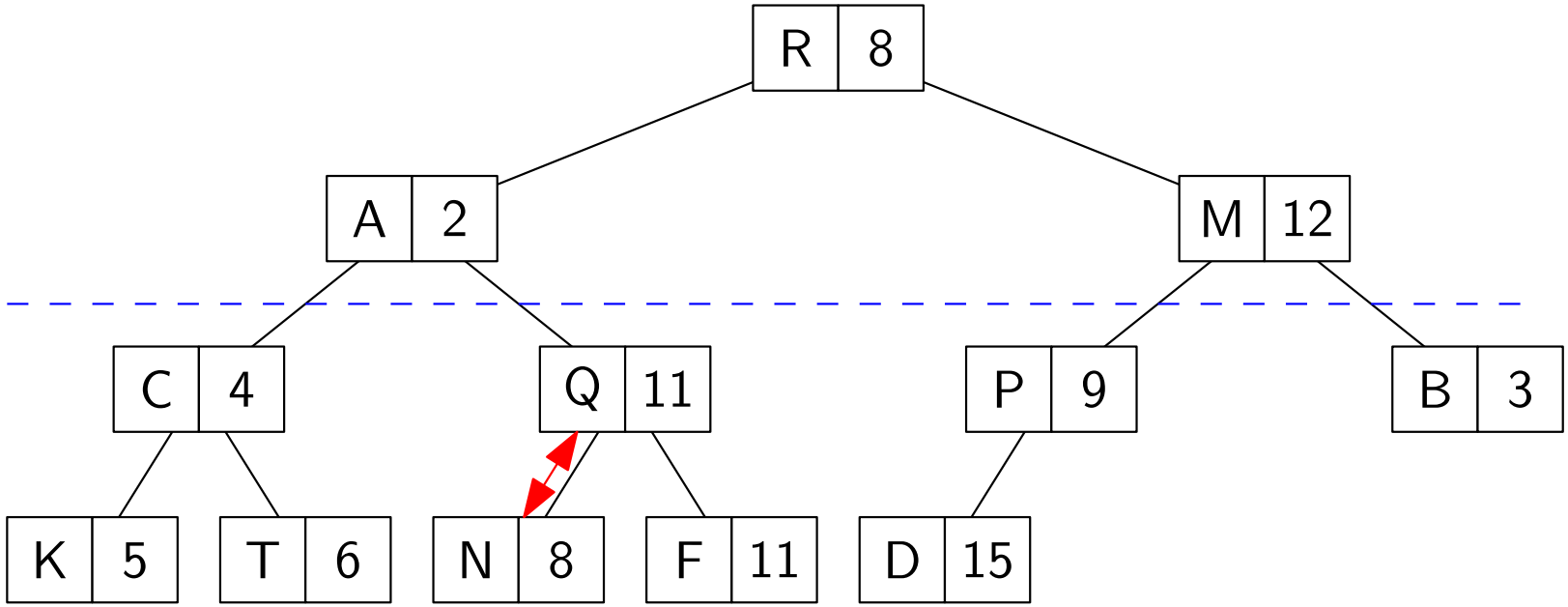
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



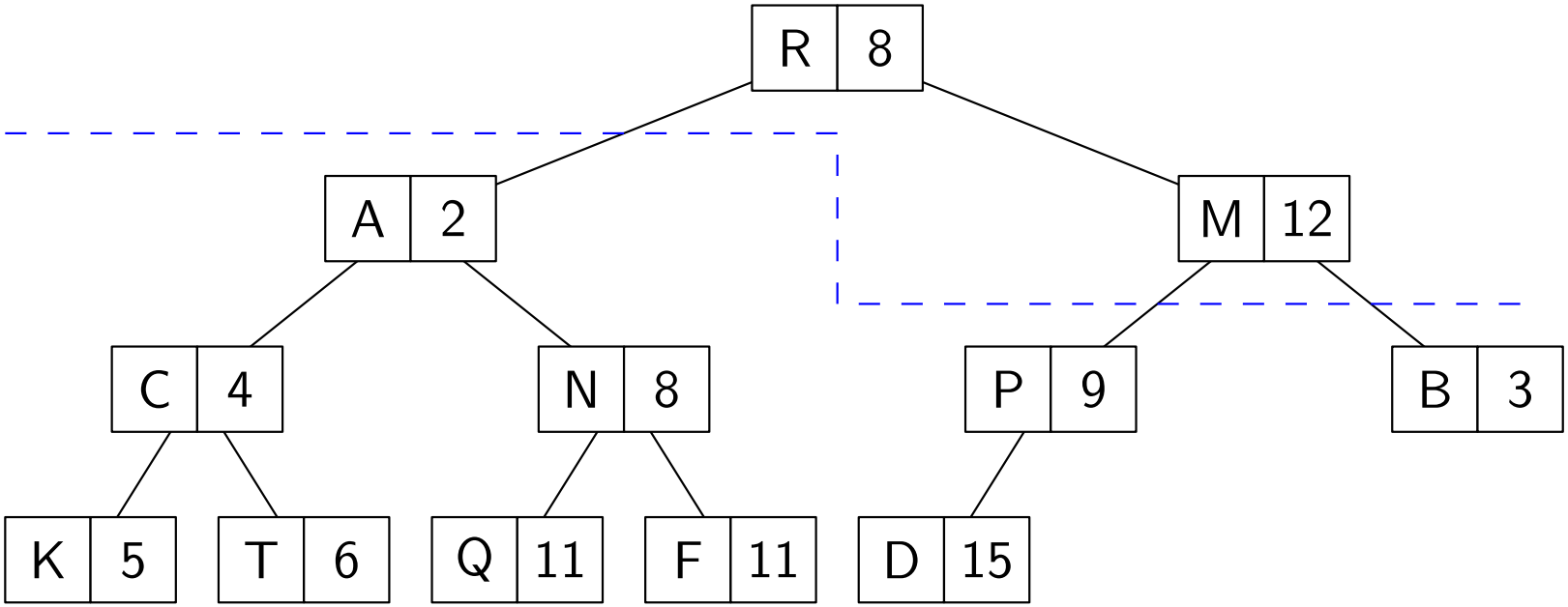
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



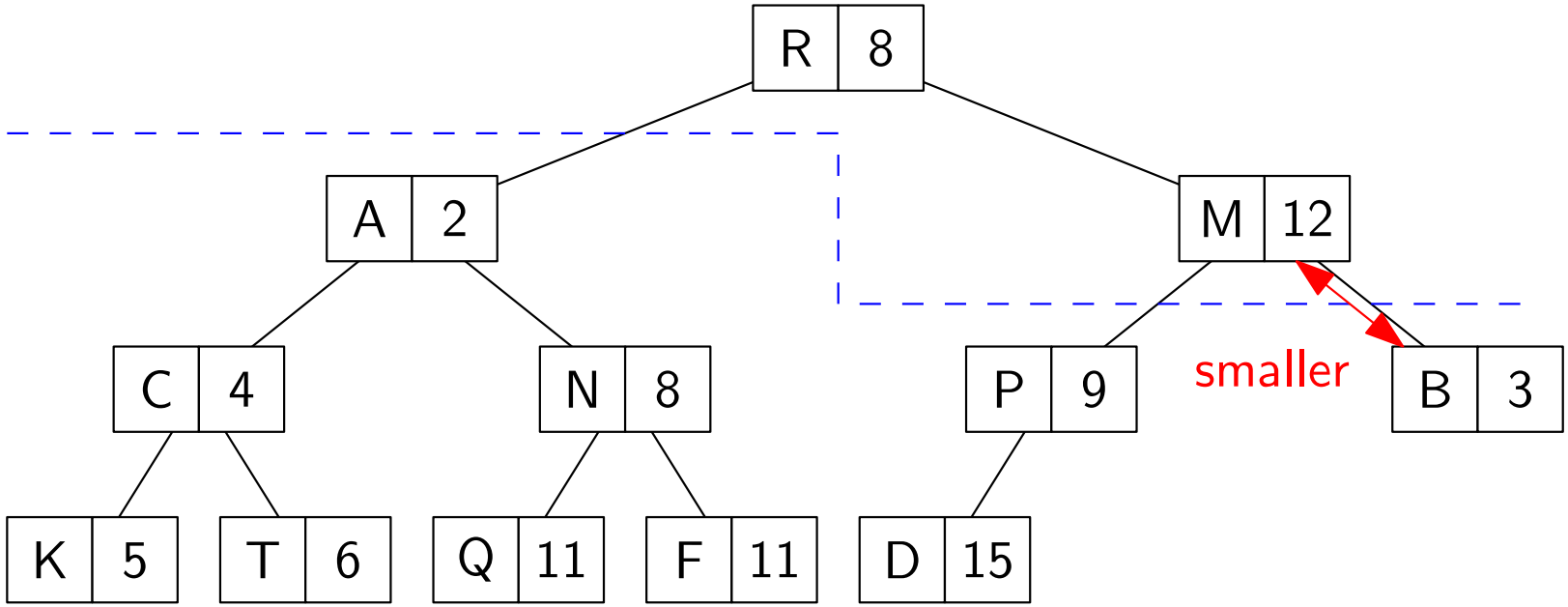
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



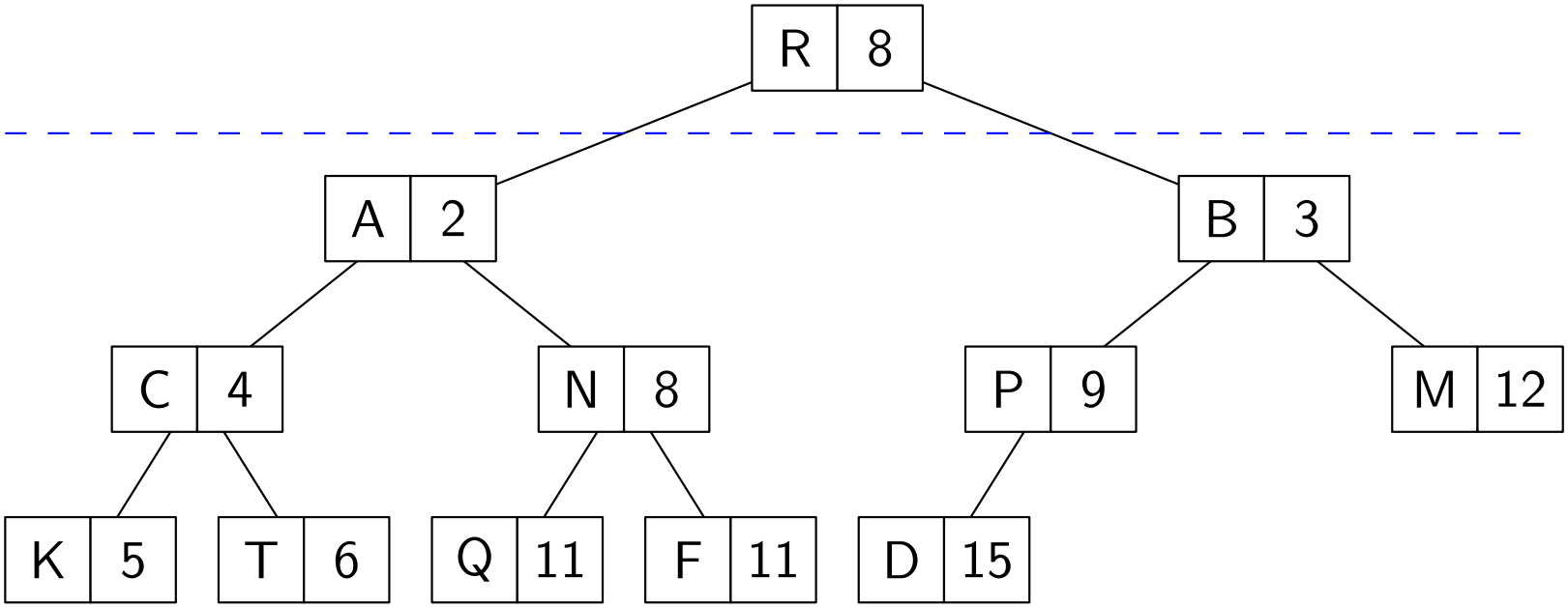
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



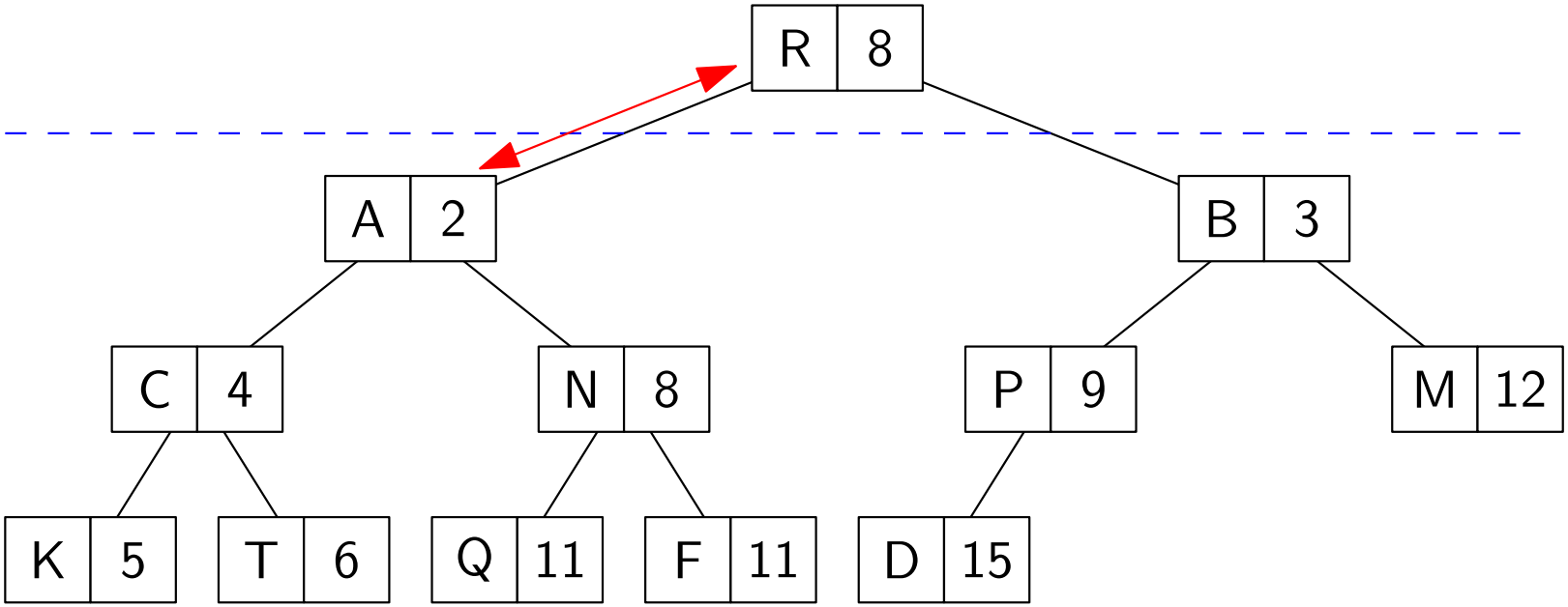
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



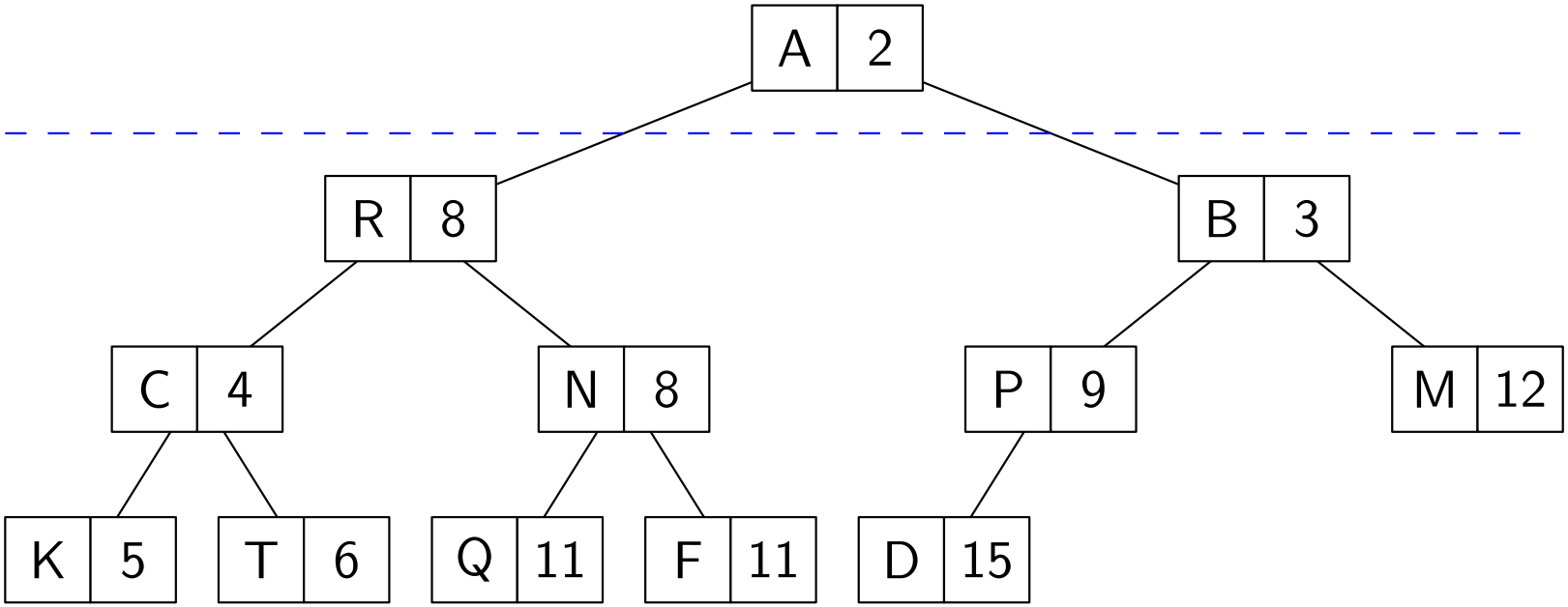
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



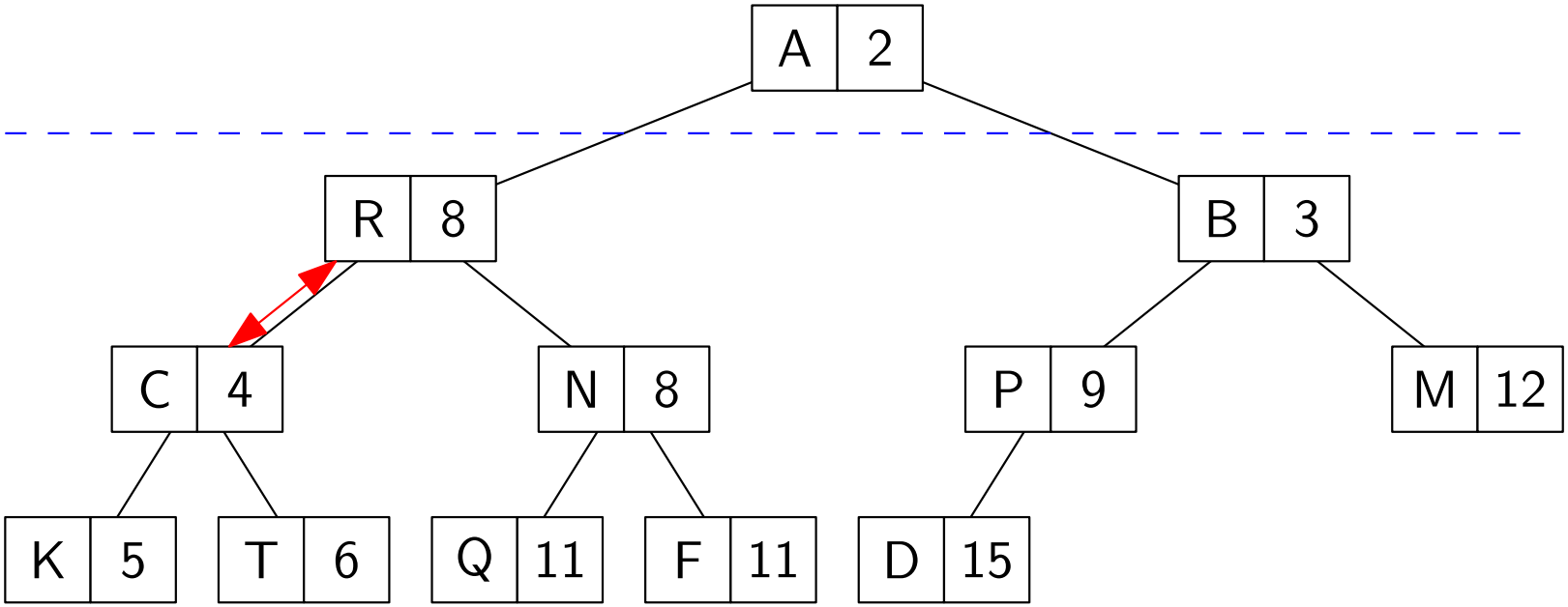
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



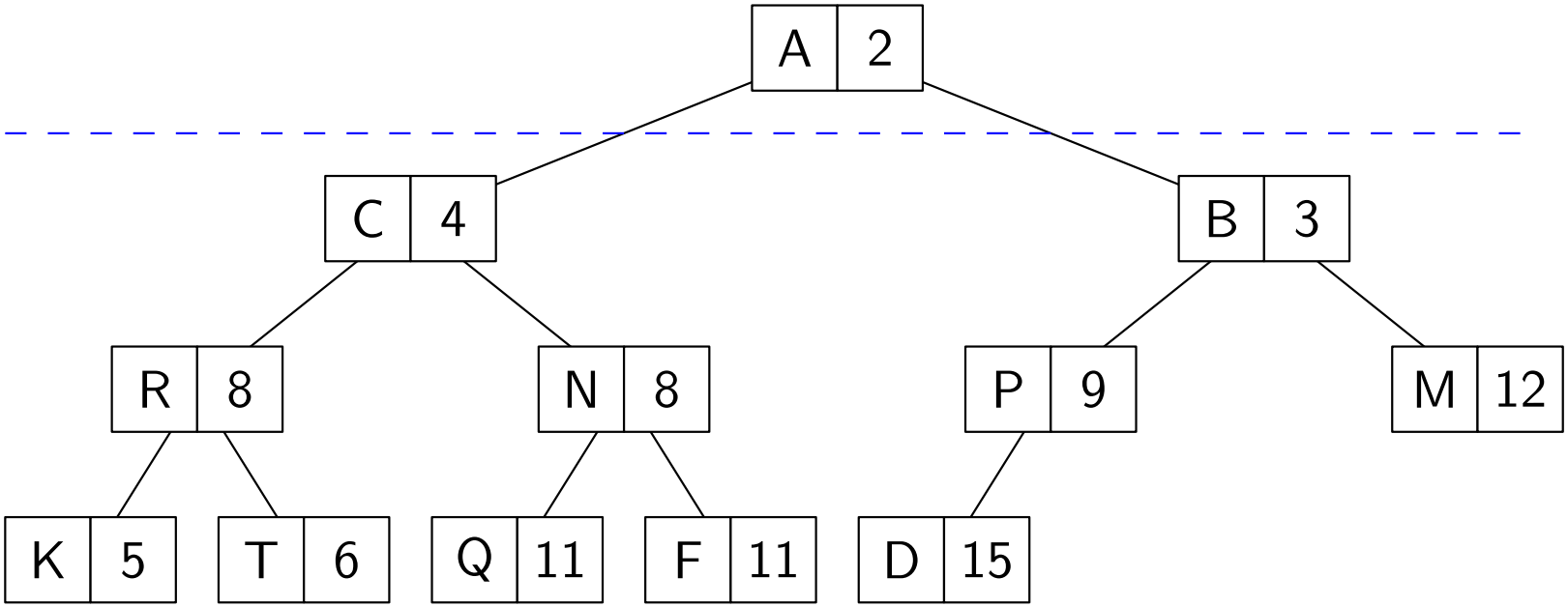
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



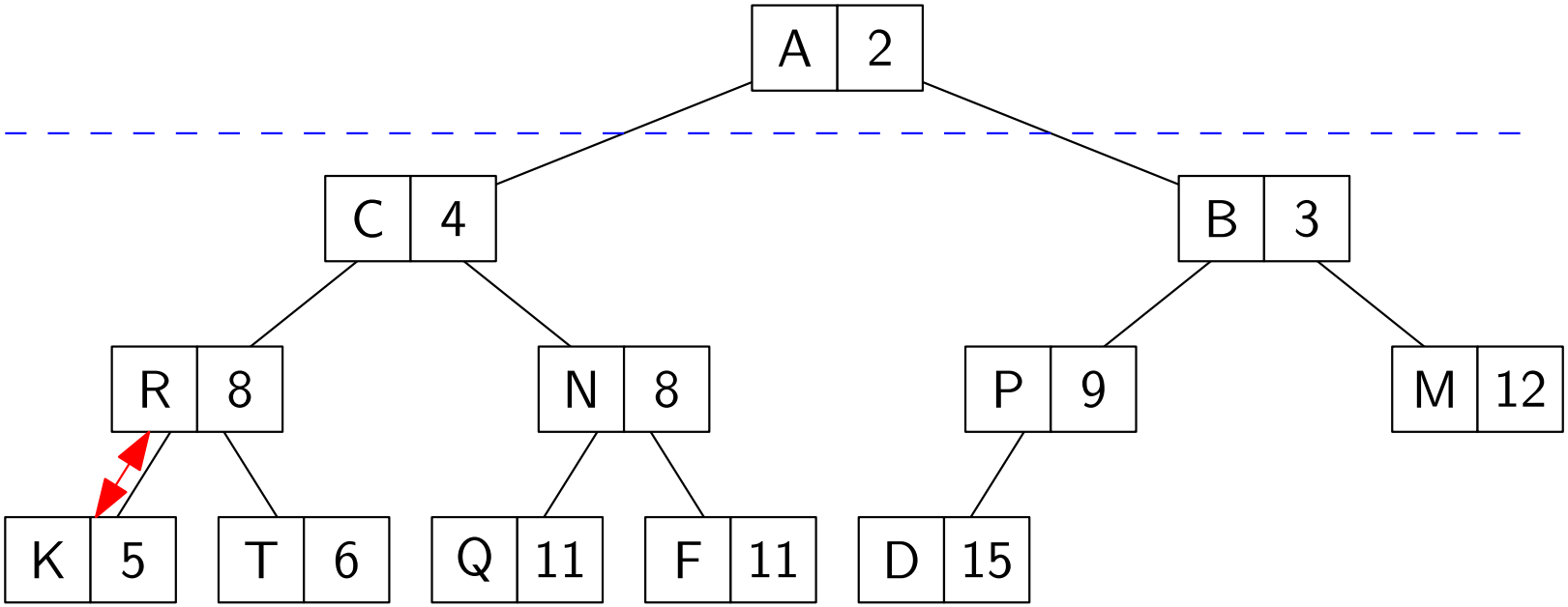
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



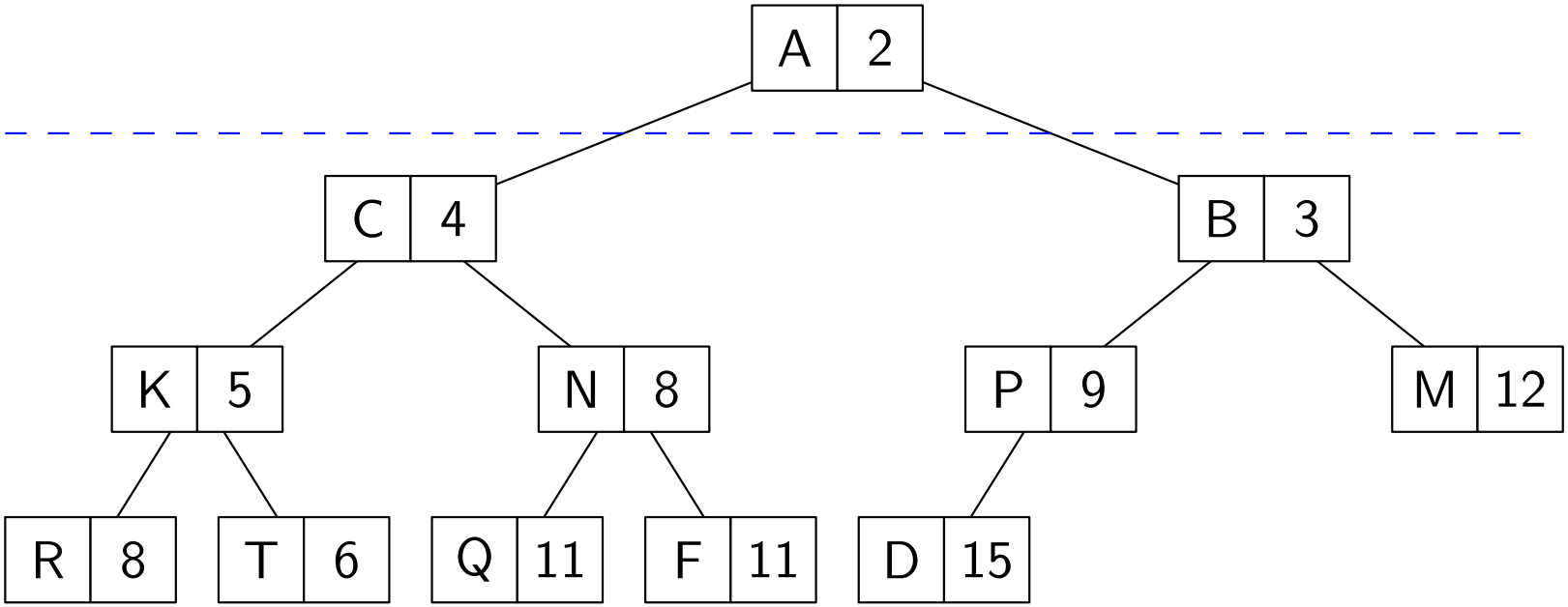
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



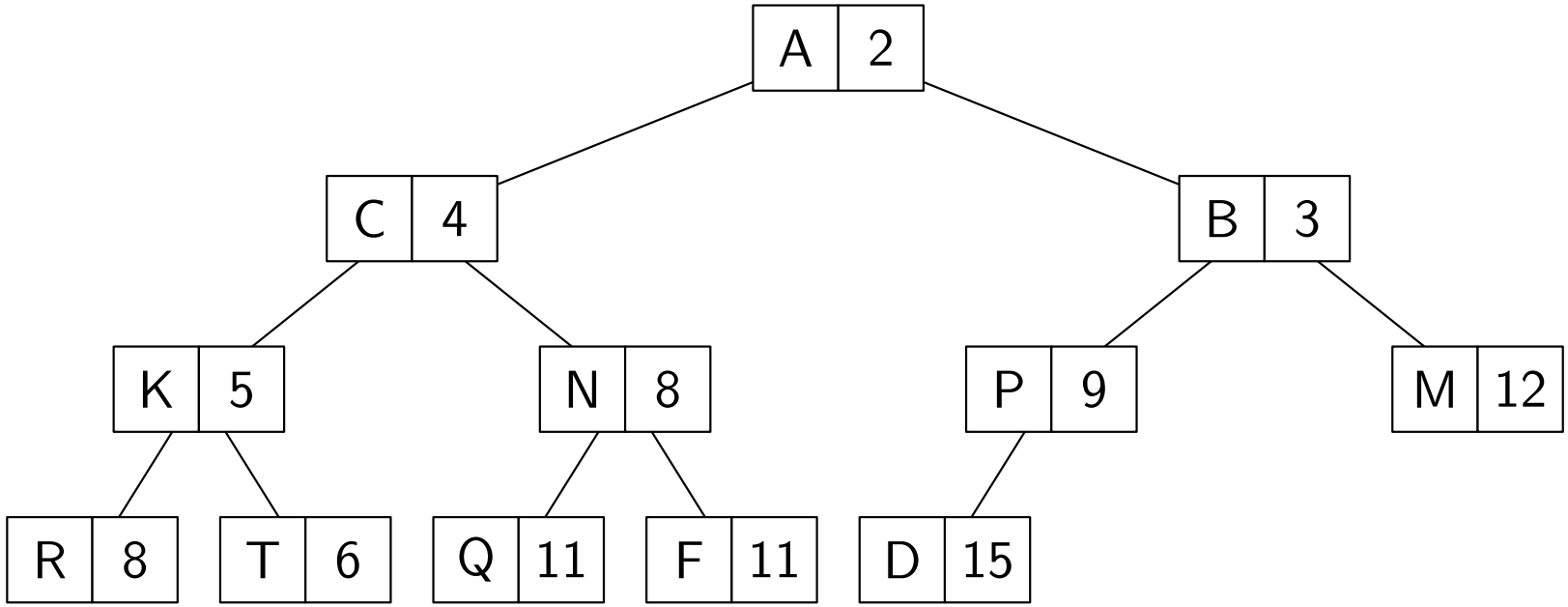
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



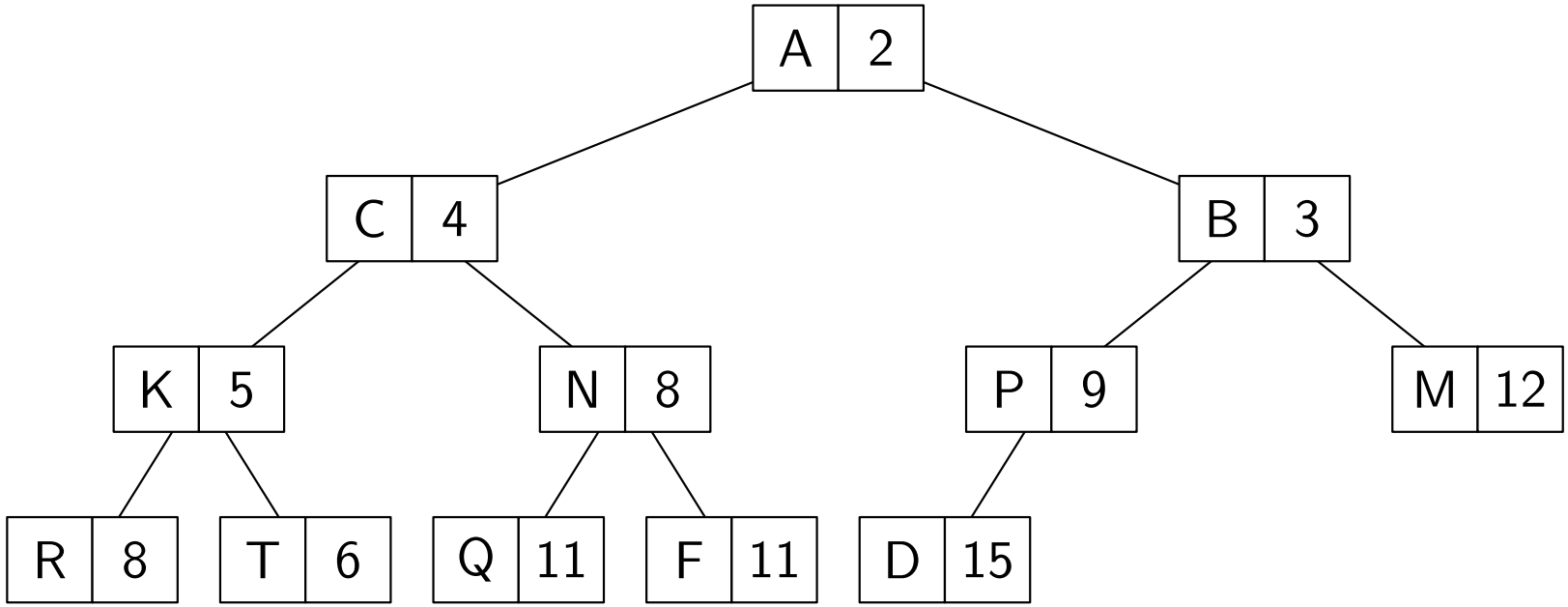
二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



二分ヒープの構成

二分ヒープをはじめに構成するときはどうするか？



処理時間は？

二分ヒープの構成：処理時間

- 下からの高さが低いノードを根とする部分木から，高いノードを根とする部分木へ順に処理が移っていく
- 高さ $h \in \{0, \ldots, \lfloor \log_2 n \rfloor\}$ のノード数 $= O\left(\frac{n}{2^h}\right)$
- 高さ h のノード 1 つにかかる処理時間 $= O(h)$

二分ヒープの構成：処理時間

- 下からの高さが低いノードを根とする部分木から，高いノードを根とする部分木へ順に処理が移っていく
- 高さ $h \in \{0, \ldots, \lfloor \log_2 n \rfloor\}$ のノード数 $= O\left(\frac{n}{2^h}\right)$
- 高さ h のノード 1 つにかかる処理時間 $= O(h)$
- したがって，全体の処理時間は高々

$$\begin{aligned}\sum_{h=0}^{\lfloor \log_2 n \rfloor} O(h) \cdot O\left(\frac{n}{2^h}\right) &= n \cdot O\left(\sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h}\right) \\ &\leq n \cdot O\left(\sum_{h=0}^{\infty} \frac{h}{2^h}\right) \\ &= n \cdot O(1) = O(n)\end{aligned}$$

注意：$\sum_{h=0}^{\infty} \frac{h}{2^h} = 2$ (演習問題)

目次

今日のまとめ

いままでの感覚

許容解 x の近傍 $N(x)$ の中で，x よりもよい許容解を見つけるためには，$|N(x)|$ に比例する時間は必要

今日のテーマ

実はそうではなく，
$|N(x)|$ よりも小さな時間で見つけられる (または，存在しないことが分かる)

2 つの例

- 最終完了時刻最小化スケジューリングの移動近傍
- 巡回セールスマン問題の 2opt 近傍

鍵となる概念

- データ構造

今回は二分ヒープのみを使ったが，他のデータ構造も多く利用される

残った時間の使い方

- 演習問題をやる
  - 相談推奨 (ひとりでやらない)
- 質問をする
  - 教員は巡回
- 退室時，小さな紙に感想など書いて提出する ← 重要
  - 内容は何でも OK
  - 匿名で OK

目次