# Splay Trees

**Disadvantage of balanced search trees:**

- worst case; no advantage for easy inputs
- additional memory required
- complicated implementation

**Splay Trees:**

- after access, an element is moved to the root; splay$(x)$
- repeated accesses are faster
- only amortized guarantee
- read-operations change the tree

# Splay Trees

**Disadvantage of balanced search trees:**

- worst case; no advantage for easy inputs
- additional memory required
- complicated implementation

**Splay Trees:**

# Splay Trees

**Disadvantage of balanced search trees:**

- worst case; no advantage for easy inputs
- additional memory required
- complicated implementation

**Splay Trees:**

- after access, an element is moved to the root; splay(x)
- repeated accesses are faster
- only amortized guarantee
- read-operations change the tree

# Splay Trees

**Disadvantage of balanced search trees:**

- worst case; no advantage for easy inputs
- additional memory required
- complicated implementation

# Splay Trees

**Disadvantage of balanced search trees:**

- worst case; no advantage for easy inputs
- additional memory required
- complicated implementation

**Splay Trees:**

+ after access, an element is moved to the root; splay($x$)
  repeated accesses are faster
- only amortized guarantee
- read-operations change the tree

# Splay Trees

**Disadvantage of balanced search trees:**

- worst case; no advantage for easy inputs
- additional memory required
- complicated implementation

**Splay Trees:**

+ after access, an element is moved to the root; splay($x$)
  repeated accesses are faster
- only amortized guarantee
- read-operations change the tree

# Splay Trees

**Disadvantage of balanced search trees:**

- worst case; no advantage for easy inputs
- additional memory required
- complicated implementation

**Splay Trees:**

+ after access, an element is moved to the root; splay$(x)$
  repeated accesses are faster
- only amortized guarantee
- read-operations change the tree

# Splay Trees

**Disadvantage of balanced search trees:**

- worst case; no advantage for easy inputs
- additional memory required
- complicated implementation

**Splay Trees:**

+ after access, an element is moved to the root; splay$(x)$
  repeated accesses are faster
- only amortized guarantee
- read-operations change the tree

# Splay Trees

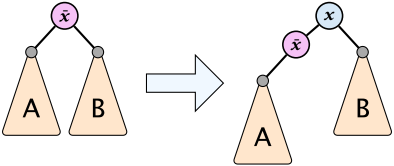
**find($x$)**

- search for $x$ according to a search tree
- let $\bar{x}$ be last element on search-path
- splay($\bar{x}$)

# Splay Trees

**insert($x$)**

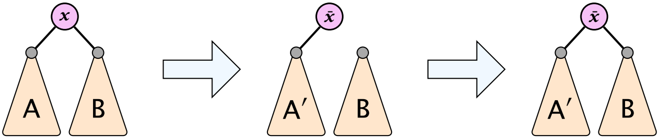
- search for $x$; $\bar{x}$ is last visited element during search (successer or predecessor of $x$)
- splay($\bar{x}$) moves $\bar{x}$ to the root
- insert $x$ as new root



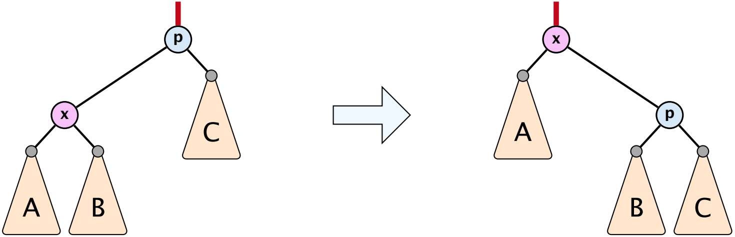
# Splay Trees

**delete($x$)**

- search for $x$; splay($x$); remove $x$
- search largest element $\bar{x}$ in $A$
- splay($\bar{x}$) (on subtree $A$)
- connect root of $B$ as right child of $\bar{x}$



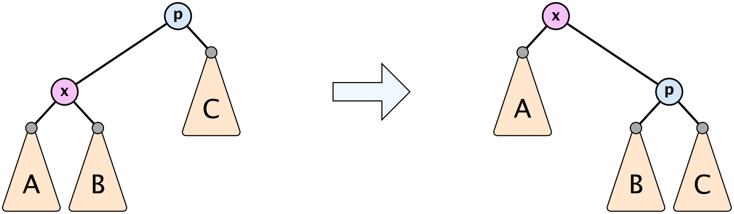
# Move to Root




**How to bring element to root?**

- one (bad) option: moveToRoot($x$)
- iteratively do rotation around parent of $x$ until $x$ is root
- if $x$ is left child do right rotation otw. left rotation

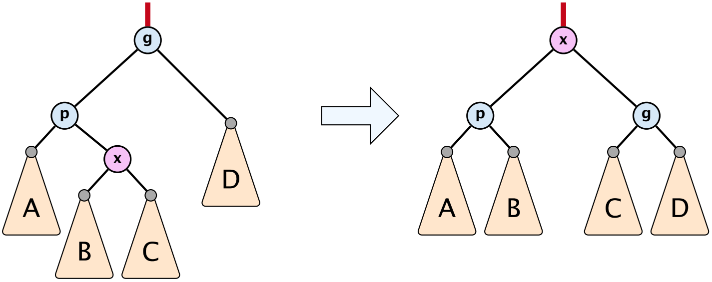
# Splay: Zig Case




**better option splay($x$):**

- zig case: if $x$ is child of root do left rotation or right rotation around parent

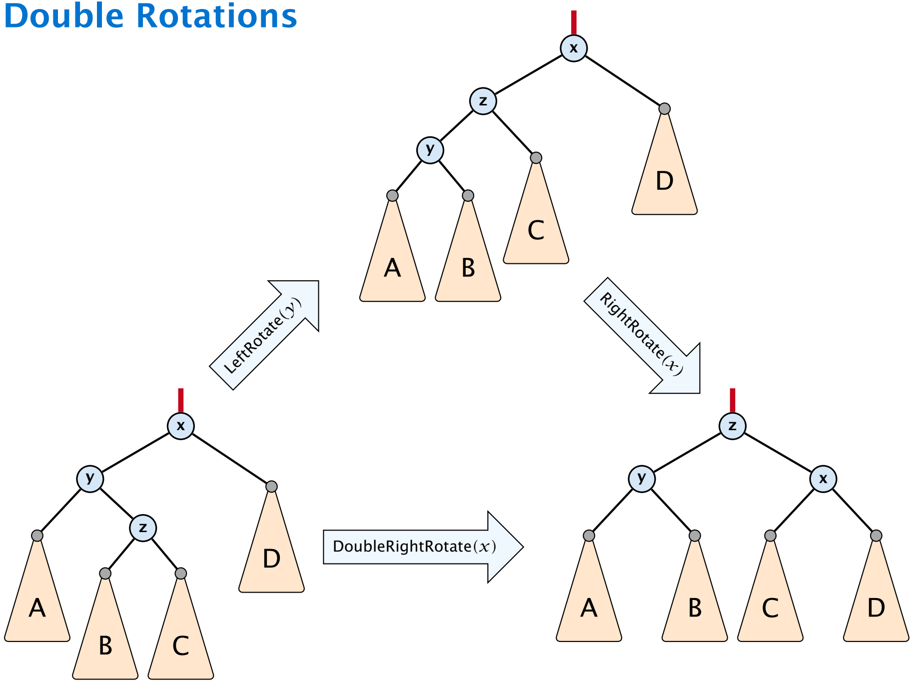
# Splay: Zigzag Case



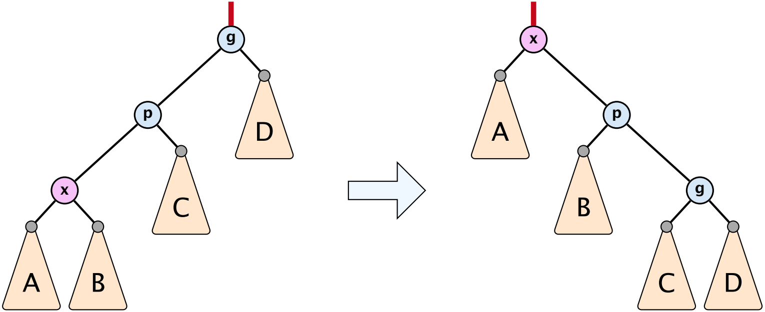

**better option splay($x$):**

- zigzag case: if $x$ is right child and parent of $x$ is left child (or $x$ left child parent of $x$ right child)
- do double right rotation around grand-parent (resp. double left rotation)

# Double Rotations



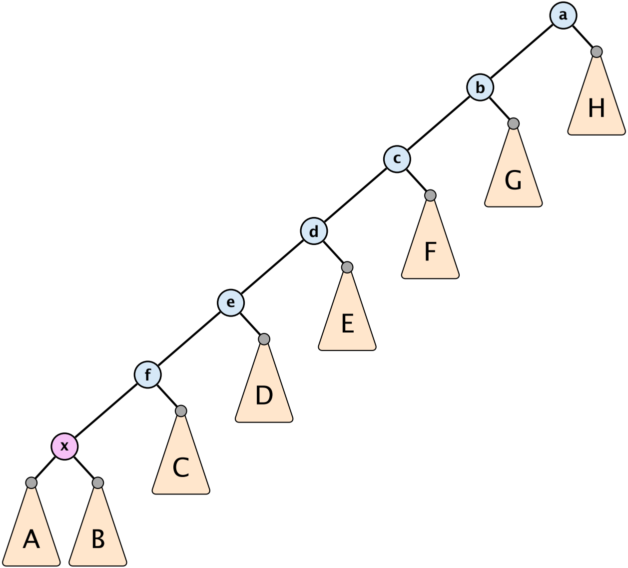
# Splay: Zigzig Case



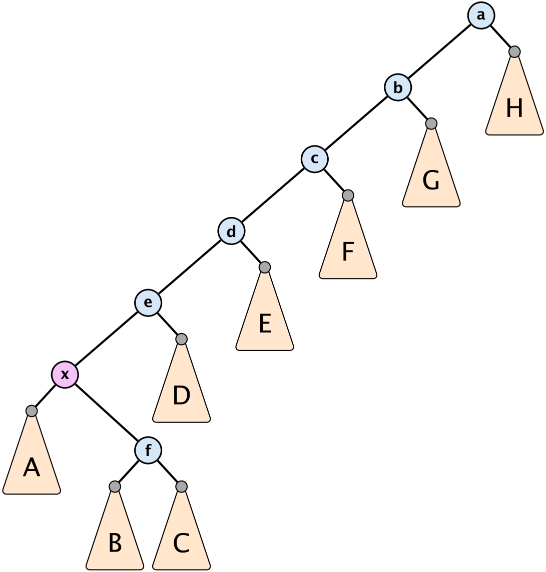

**better option splay($x$):**

- zigzig case: if $x$ is left child and parent of $x$ is left child (or $x$ right child, parent of $x$ right child)
- do right roation around grand-parent followed by right rotation around parent (resp. left rotations)

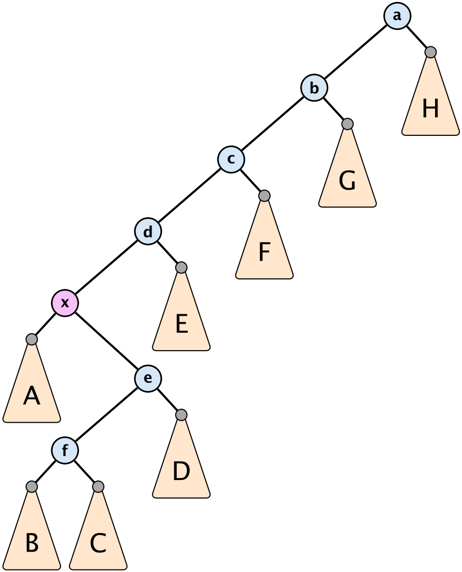
# Splay vs. Move to Root



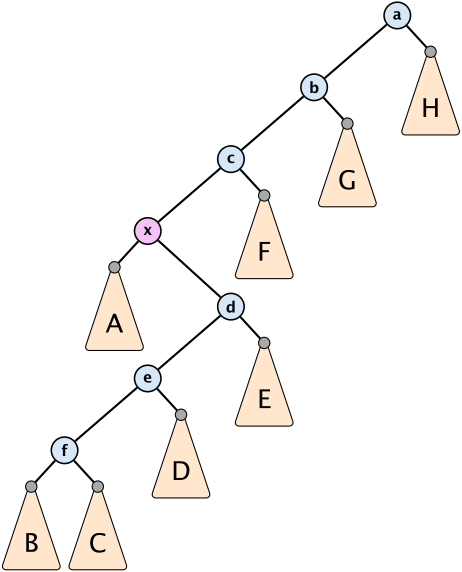
# Splay vs. Move to Root



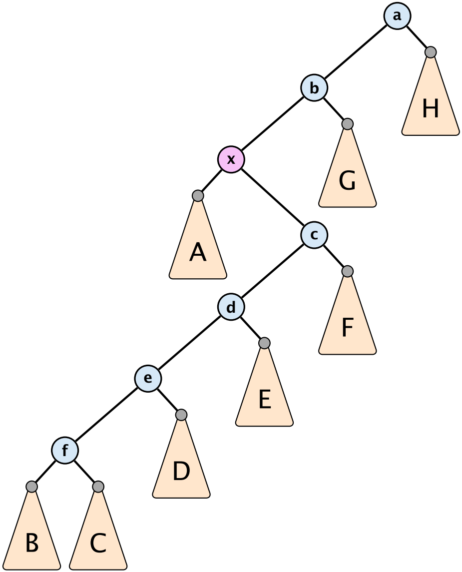
# Splay vs. Move to Root



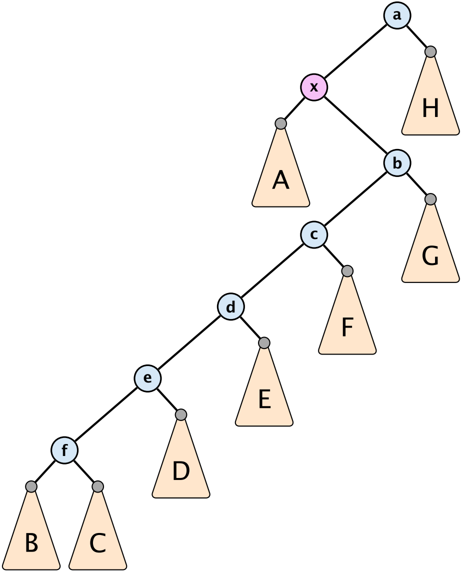
# Splay vs. Move to Root



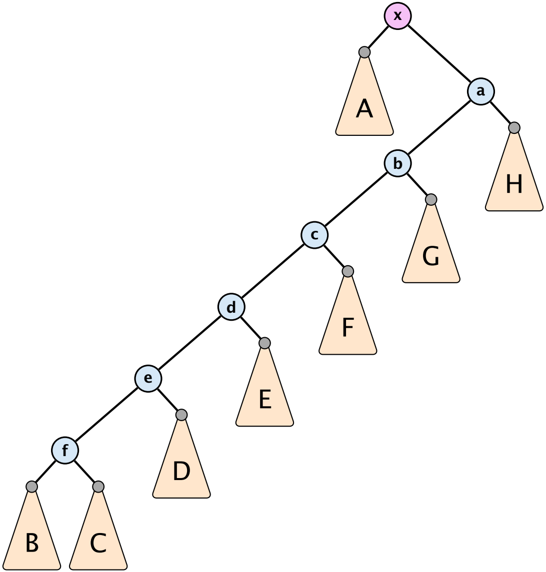
# Splay vs. Move to Root



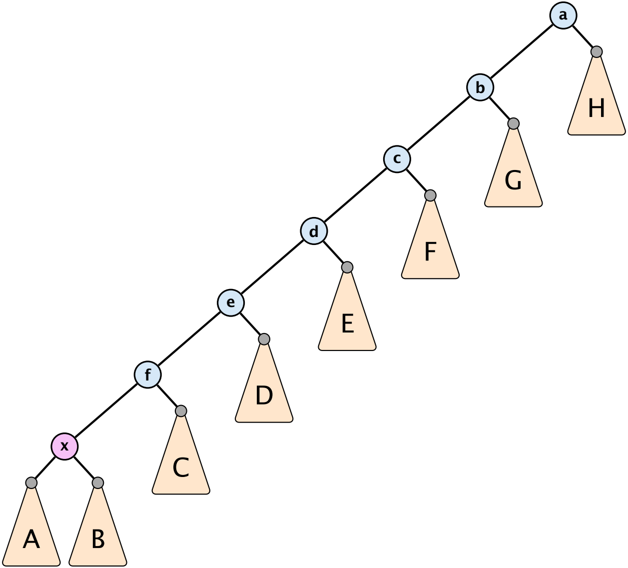
# Splay vs. Move to Root



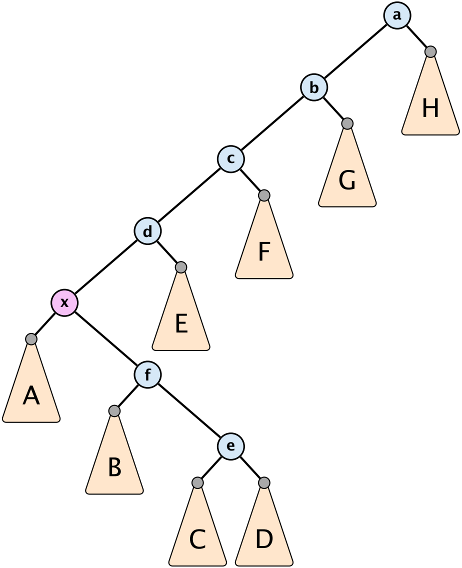
# Splay vs. Move to Root



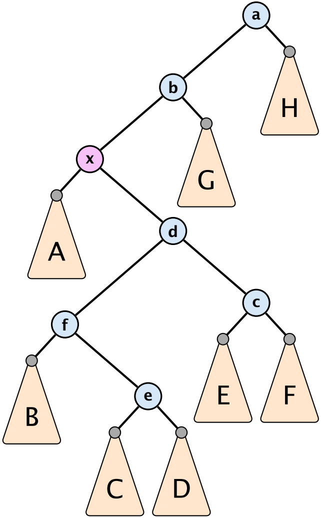
# Splay vs. Move to Root



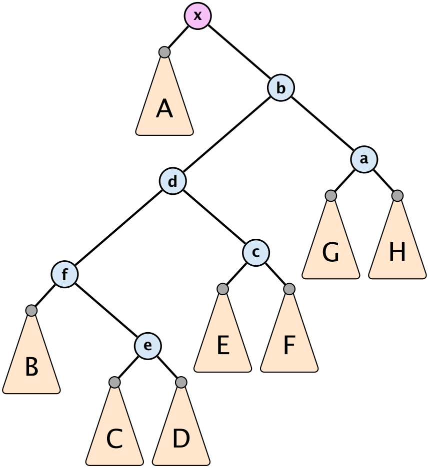
# Splay vs. Move to Root



# Splay vs. Move to Root



# Splay vs. Move to Root



# Static Optimality

Suppose we have a sequence of $m$ find-operations. find$(x)$ appears $h_x$ times in this sequence.

The cost of a static search tree $T$ is:

$$\mathrm{cost}(T) = m + \sum_x h_x \,\mathrm{depth}_T(x)$$

The total cost for processing the sequence on a splay-tree is $\mathcal{O}(\mathrm{cost}(T_{\min}))$, where $T_{\min}$ is an optimal static search tree.

# Dynamic Optimality

Let $S$ be a sequence with $m$ find-operations.

Let $A$ be a data-structure based on a search tree:

- the cost for accessing element $x$ is $1 + \mathrm{depth}(x)$;
- after accessing $x$ the tree may be re-arranged through rotations;

**Conjecture:**
A splay tree that only contains elements from $S$ has cost $\mathcal{O}(\mathrm{cost}(A, S))$, for processing $S$.

**Lemma 1**

*Splay Trees have an amortized running time of $\mathcal{O}(\log n)$ for all operations.*

# Amortized Analysis

**Definition 2**
A data structure with operations $\mathrm{op}_1(), \dots, \mathrm{op}_k()$ has amortized running times $t_1, \dots, t_k$ for these operations if the following holds.

Suppose you are given a sequence of operations (starting with an empty data-structure) that operate on at most $n$ elements, and let $k_i$ denote the number of occurences of $\mathrm{op}_i()$ within this sequence. Then the actual running time must be at most $\sum_i k_i \cdot t_i(n)$.

# Potential Method

**Introduce a potential for the data structure.**

## Potential Method

**Introduce a potential for the data structure.**

- $\Phi(D_i)$ is the potential after the $i$-th operation.

# Potential Method

**Introduce a potential for the data structure.**

- $\Phi(D_i)$ is the potential after the $i$-th operation.
- Amortized cost of the $i$-th operation is

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \ .$$

# Potential Method

**Introduce a potential for the data structure.**

- $\Phi(D_i)$ is the potential after the $i$-th operation.
- Amortized cost of the $i$-th operation is

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \ .$$

- Show that $\Phi(D_i) \geq \Phi(D_0)$.

# Potential Method

**Introduce a potential for the data structure.**

- $\Phi(D_i)$ is the potential after the $i$-th operation.
- Amortized cost of the $i$-th operation is

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \ .$$

- Show that $\Phi(D_i) \geq \Phi(D_0)$.

Then

$$\sum_{i=1}^{k} c_i$$

# Potential Method

**Introduce a potential for the data structure.**

- $\Phi(D_i)$ is the potential after the $i$-th operation.
- Amortized cost of the $i$-th operation is

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \ .$$

- Show that $\Phi(D_i) \geq \Phi(D_0)$.

Then

$$\sum_{i=1}^{k} c_i \leq \sum_{i=1}^{k} c_i + \Phi(D_k) - \Phi(D_0)$$

# Potential Method

**Introduce a potential for the data structure.**

- $\Phi(D_i)$ is the potential after the $i$-th operation.
- Amortized cost of the $i$-th operation is

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \ .$$

- Show that $\Phi(D_i) \geq \Phi(D_0)$.

Then

$$\sum_{i=1}^{k} c_i \leq \sum_{i=1}^{k} c_i + \Phi(D_k) - \Phi(D_0) = \sum_{i=1}^{k} \hat{c}_i$$

This means the amortized costs can be used to derive a bound on the total cost.

# Example: Stack

**Stack**

- $S.\mathbf{push}()$
- $S.\mathbf{pop}()$
- $S.\mathbf{multipop}(k)$: removes $k$ items from the stack. If the stack currently contains less than $k$ items it empties the stack.
- The user has to ensure that pop and multipop do not generate an underflow.

**Actual cost:**

- $S.\mathbf{push}()$: cost $1$.
- $S.\mathbf{pop}()$: cost $1$.
- $S.\mathbf{multipop}(k)$: cost $\min\{\text{size}, k\} = k$.

# Example: Stack

**Stack**

- $S.\textbf{push}()$
- $S.\textbf{pop}()$
- $S.\textbf{multipop}(k)$: removes $k$ items from the stack. If the stack currently contains less than $k$ items it empties the stack.
- The user has to ensure that pop and multipop do not generate an underflow.

**Actual cost:**

- $S.\textbf{push}()$: cost $1$.
- $S.\textbf{pop}()$: cost $1$.
- $S.\textbf{multipop}(k)$: cost $\min\{\text{size}, k\} = k$.

# Example: Stack

Use potential function $\Phi(S) =$ number of elements on the stack.

**Amortized cost:**

- S. push(): cost

$$\hat{C}_{push} = C_{push} + \Delta\Phi = 1 + 1 \le 2 .$$

- S. pop(): cost

$$\hat{C}_{pop} = C_{pop} + \Delta\Phi = 1 - 1 \le 0 .$$

- S. multipop(k): cost

$$\hat{C}_{mp} = C_{mp} + \Delta\Phi = \min(size, k) - \min(size, k) \le 0 .$$

# Example: Stack

Use potential function $\Phi(S)$ = number of elements on the stack.

**Amortized cost:**

- $S.\textbf{push}()$: cost

$$\hat{C}_{\text{push}} = C_{\text{push}} + \Delta\Phi = 1 + 1 \le 2 \ .$$

- $S.\textbf{pop}()$: cost

$$\hat{C}_{\text{pop}} = C_{\text{pop}} + \Delta\Phi = 1 - 1 \le 0 \ .$$

- $S.\textbf{multipop}(k)$: cost

$$\hat{C}_{\text{mp}} = C_{\text{mp}} + \Delta\Phi = \min\{\text{size}, k\} - \min\{\text{size}, k\} \le 0 \ .$$

# Example: Stack

Use potential function $\Phi(S)$ = number of elements on the stack.

**Amortized cost:**

- $\boldsymbol{S.\,\mathrm{push}()}$: cost

$$\hat{C}_{\text{push}} = C_{\text{push}} + \Delta\Phi = 1 + 1 \leq 2 \enspace .$$

- $\boldsymbol{S.\,\mathrm{pop}()}$: cost

$$\hat{C}_{\text{pop}} = C_{\text{pop}} + \Delta\Phi = 1 - 1 \leq 0 \enspace .$$

- $\boldsymbol{S.\,\mathrm{multipop}(k)}$: cost

$$\hat{C}_{\text{mp}} = C_{\text{mp}} + \Delta\Phi = \min\{\text{size}, k\} - \min\{\text{size}, k\} \leq 0 \enspace .$$

## Example: Stack

Use potential function $\Phi(S)$ = number of elements on the stack.

**Amortized cost:**

- $\boldsymbol{S.\,\mathrm{push}()}$: cost

$$\hat{C}_{\mathrm{push}} = C_{\mathrm{push}} + \Delta\Phi = 1 + 1 \leq 2 \ .$$

- $\boldsymbol{S.\,\mathrm{pop}()}$: cost

$$\hat{C}_{\mathrm{pop}} = C_{\mathrm{pop}} + \Delta\Phi = 1 - 1 \leq 0 \ .$$

- $\boldsymbol{S.\,\mathrm{multipop}(k)}$: cost

$$\hat{C}_{\mathrm{mp}} = C_{\mathrm{mp}} + \Delta\Phi = \min\{\mathrm{size}, k\} - \min\{\mathrm{size}, k\} \leq 0 \ .$$

# Example: Binary Counter

**Incrementing a binary counter:**
Consider a computational model where each bit-operation costs one time-unit.

Incrementing an $n$-bit binary counter may require to examine $n$-bits, and maybe change them.

**Actual cost:**

- Changing bit from $0$ to $1$: cost $1$.
- Changing bit from $1$ to $0$: cost $1$.
- Increment: cost is $k+1$, where $k$ is the number of consecutive ones in the least significant bit-positions (e.g, 001101 has $k=1$).

# Example: Binary Counter

**Incrementing a binary counter:**
Consider a computational model where each bit-operation costs one time-unit.

Incrementing an $n$-bit binary counter may require to examine $n$-bits, and maybe change them.

**Actual cost:**

- Changing bit from 0 to 1: cost 1.
- Changing bit from 1 to 0: cost 1.
- Increment: cost is $k+1$, where $k$ is the number of consecutive ones in the least significant bit-positions (e.g, 001101 has $k=1$).

# Example: Binary Counter

**Incrementing a binary counter:**
Consider a computational model where each bit-operation costs one time-unit.

Incrementing an $n$-bit binary counter may require to examine $n$-bits, and maybe change them.

**Actual cost:**

- Changing bit from $0$ to $1$: cost $1$.
- Changing bit from $1$ to $0$: cost $1$.
- Increment: cost is $k+1$, where $k$ is the number of consecutive ones in the least significant bit-positions (e.g, 001101 has $k=1$).

# Example: Binary Counter

Choose potential function $\Phi(x) = k$, where $k$ denotes the number of ones in the binary representation of $x$.

**Amortized cost:**

$0 \to 1$:

$$\hat{C}_{0\to1} = C_{0\to1} + \Delta\Phi = 1 + 1 \le 2 .$$

$1 \to 0$:

$$\hat{C}_{1\to0} = C_{1\to0} + \Delta\Phi = 1 - 1 \le 0 .$$

Let $k$ denotes the number of consecutive ones in the least significant bit-positions. An increment involves $k$ $(1 \to 0)$-operations, and one $(0 \to 1)$-operation.

Hence, the amortized cost is $k\hat{C}_{1\to0} + \hat{C}_{0\to1} \le 2$.

# Example: Binary Counter

Choose potential function $\Phi(x) = k$, where $k$ denotes the number of ones in the binary representation of $x$.

**Amortized cost:**

- Changing bit from 0 to 1:

$$\hat{C}_{0\to1} = C_{0\to1} + \Delta\Phi = 1 + 1 \le 2 \ .$$

- Changing bit from 1 to 0:

$$\hat{C}_{1\to0} = C_{1\to0} + \Delta\Phi = 1 - 1 \le 0 \ .$$

- Increment: Let $k$ denotes the number of consecutive ones in the least significant bit-positions. An increment involves $k$ $(1 \to 0)$-operations, and one $(0 \to 1)$-operation.

  Hence, the amortized cost is $k\hat{C}_{1\to0} + \hat{C}_{0\to1} \le 2$.

# Example: Binary Counter

Choose potential function $\Phi(x) = k$, where $k$ denotes the number of ones in the binary representation of $x$.

**Amortized cost:**

- Changing bit from $0$ to $1$:

$$\hat{C}_{0\to1} = C_{0\to1} + \Delta\Phi = 1 + 1 \le 2 \ .$$

- Changing bit from $1$ to $0$:

$$\hat{C}_{1\to0} = C_{1\to0} + \Delta\Phi = 1 - 1 \le 0 \ .$$

- Increment: Let $k$ denotes the number of consecutive ones in the least significant bit-positions. An increment involves $k$ $(1 \to 0)$-operations, and one $(0 \to 1)$-operation.

  Hence, the amortized cost is $k\hat{C}_{1\to0} + \hat{C}_{0\to1} \le 2$.

# Example: Binary Counter

Choose potential function $\Phi(x) = k$, where $k$ denotes the number of ones in the binary representation of $x$.

**Amortized cost:**

- Changing bit from $0$ to $1$:

$$\hat{C}_{0\to 1} = C_{0\to 1} + \Delta\Phi = 1 + 1 \le 2 \ .$$

- Changing bit from $1$ to $0$:

$$\hat{C}_{1\to 0} = C_{1\to 0} + \Delta\Phi = 1 - 1 \le 0 \ .$$

- Increment: Let $k$ denotes the number of consecutive ones in the least significant bit-positions. An increment involves $k$ $(1 \to 0)$-operations, and one $(0 \to 1)$-operation.

  Hence, the amortized cost is $k\hat{C}_{1\to 0} + \hat{C}_{0\to 1} \le 2$.

## Example: Binary Counter

Choose potential function $\Phi(x) = k$, where $k$ denotes the number of ones in the binary representation of $x$.

**Amortized cost:**

- Changing bit from $0$ to $1$:

$$\hat{C}_{0\to 1} = C_{0\to 1} + \Delta\Phi = 1 + 1 \leq 2 \ .$$

- Changing bit from $1$ to $0$:

$$\hat{C}_{1\to 0} = C_{1\to 0} + \Delta\Phi = 1 - 1 \leq 0 \ .$$

- Increment: Let $k$ denotes the number of consecutive ones in the least significant bit-positions. An increment involves $k$ $(1 \to 0)$-operations, and one $(0 \to 1)$-operation.

  Hence, the amortized cost is $k\hat{C}_{1\to 0} + \hat{C}_{0\to 1} \leq 2$.

# Splay Trees

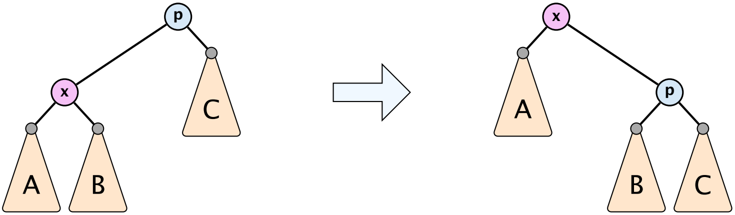
**potential function for splay trees:**

- size $\mathrm{s}(x) = |T_x|$
- rank $\mathrm{r}(x) = \log_2(s(x))$
- $\Phi(T) = \sum_{v \in T} r(v)$

amortized cost = real cost + potential change

The cost is essentially the cost of the splay-operation, which is 1 plus the number of rotations.

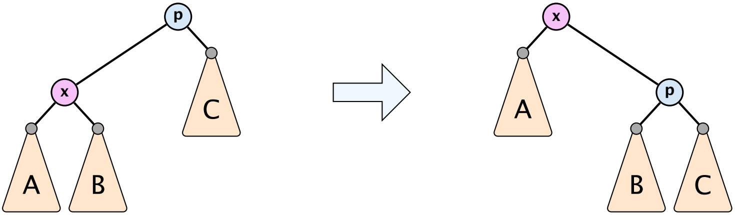
# Splay: Zig Case




$$\Delta\Phi = r'(x) + r'(p) - r(x) - r(p)$$
$$= r'(p) - r(x)$$
$$\leq r'(x) - r(x)$$

$$\mathrm{cost}_{\mathrm{zig}} \leq 1 + 3(r'(x) - r(x))$$

# Splay: Zig Case



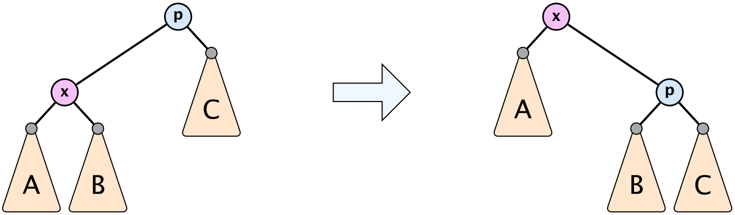

$$\Delta\Phi = r'(x) + r'(p) - r(x) - r(p)$$

$$= r'(p) - r(x)$$

$$\leq r'(x) - r(x)$$

$$\mathrm{cost}_{\mathrm{zig}} \leq 1 + 3(r'(x) - r(x))$$

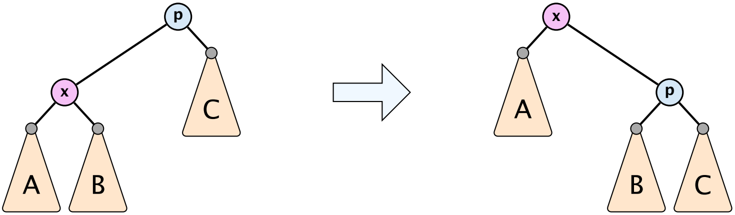
# Splay: Zig Case




$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) - r(x) - r(p) \\
&= r'(p) - r(x) \\
&\leq r'(x) - r(x)
\end{aligned}
$$

$$
\mathrm{cost}_{\mathrm{zig}} \leq 1 + 3(r'(x) - r(x))
$$

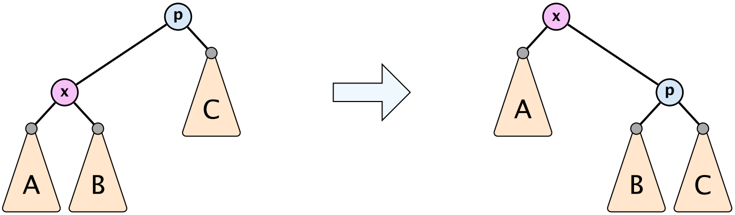
# Splay: Zig Case




$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) - r(x) - r(p) \\
&= r'(p) - r(x) \\
&\leq r'(x) - r(x)
\end{aligned}
$$

$$\text{cost}_{\text{zig}} \leq 1 + 3(r'(x) - r(x))$$

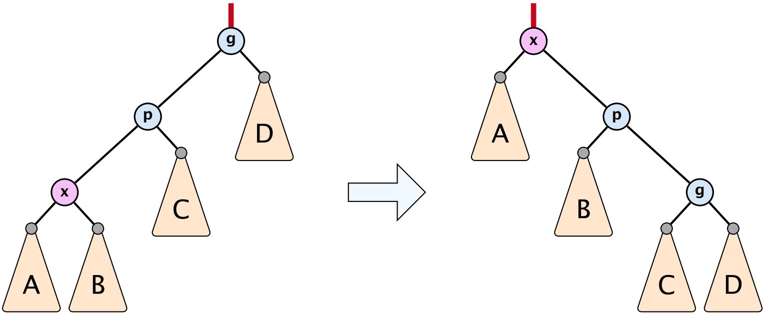
# Splay: Zig Case




$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) - r(x) - r(p) \\
&= r'(p) - r(x) \\
&\le r'(x) - r(x)
\end{aligned}
$$

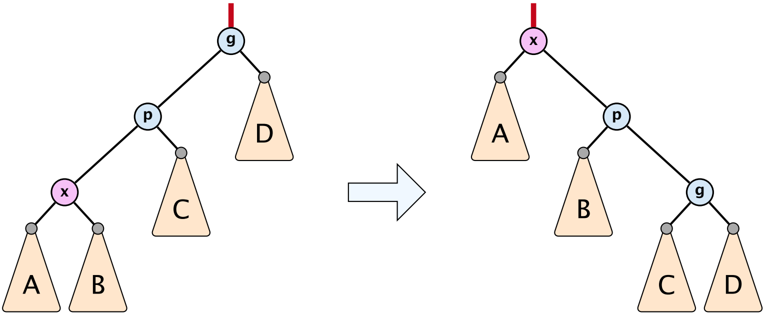
$$
\text{cost}_{\text{zig}} \le 1 + 3(r'(x) - r(x))
$$

# Splay: Zigzig Case



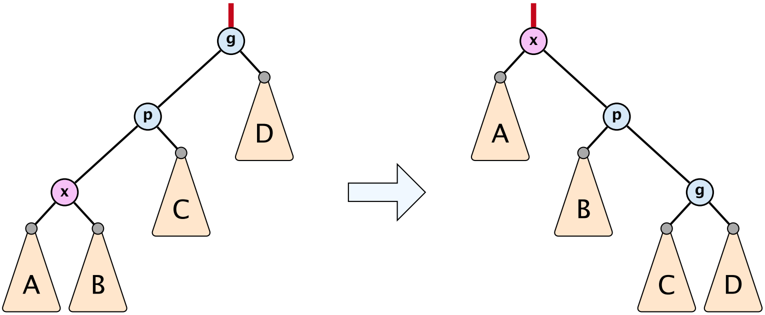

$$\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\le r'(x) + r'(g) - r(x) - r(x) \\
&= r'(x) + r'(g) + r(x) - 3r'(x) + 3r'(x) - r(x) - 2r(x) \\
&= -2r'(x) + r'(g) + r(x) + 3(r'(x) - r(x)) \\
&\le -2 + 3(r'(x) - r(x)) \quad \Rightarrow \mathrm{cost}_{\mathrm{zigzig}} \le 3(r'(x) - r(x))
\end{aligned}$$

# Splay: Zigzig Case



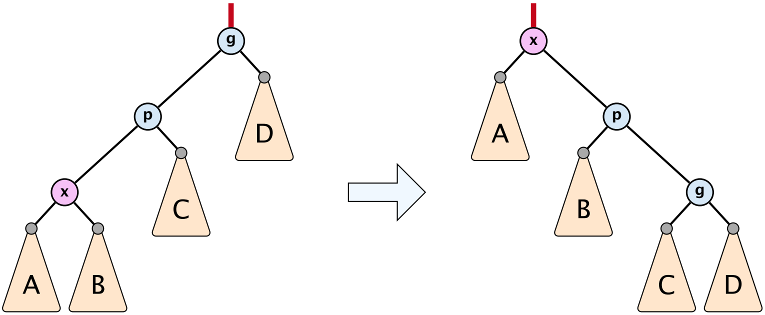

$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\le r'(x) + r'(g) - r(x) - r(x) \\
&= r'(x) + r'(g) + r(x) - 3r'(x) + 3r'(x) - r(x) - 2r(x) \\
&= -2r'(x) + r'(g) + r(x) + 3(r'(x) - r(x)) \\
&\le -2 + 3(r'(x) - r(x)) \quad \Rightarrow \mathrm{cost}_{\mathrm{zigzig}} \le 3(r'(x) - r(x))
\end{aligned}
$$

# Splay: Zigzig Case



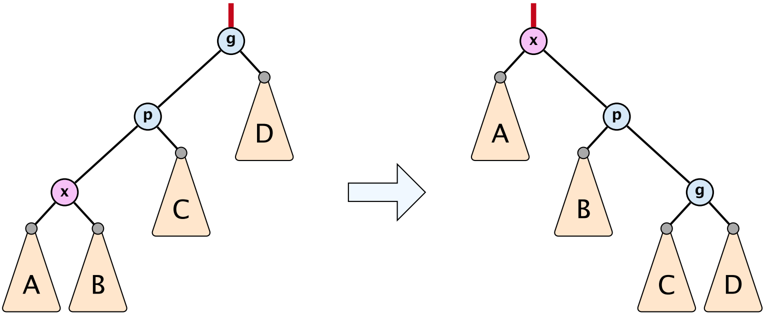

$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\leq r'(x) + r'(g) - r(x) - r(x) \\
&= r'(x) + r'(g) + r(x) - 3r'(x) + 3r'(x) - r(x) - 2r(x) \\
&= -2r'(x) + r'(g) + r(x) + 3(r'(x) - r(x)) \\
&\leq -2 + 3(r'(x) - r(x)) \quad \Rightarrow \mathrm{cost}_{\mathrm{zigzig}} \leq 3(r'(x) - r(x))
\end{aligned}
$$

# Splay: Zigzig Case



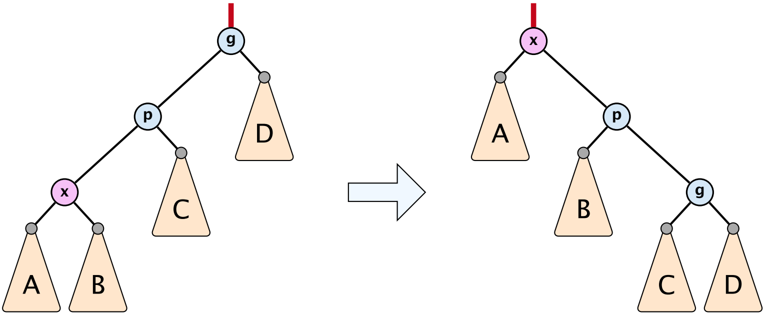

$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g)\\
&= r'(p) + r'(g) - r(x) - r(p)\\
&\le r'(x) + r'(g) - r(x) - r(x)\\
&= r'(x) + r'(g) + r(x) - 3r'(x) + 3r'(x) - r(x) - 2r(x)\\
&= -2r'(x) + r'(g) + r(x) + 3(r'(x) - r(x))\\
&\le -2 + 3(r'(x) - r(x)) \quad \Rightarrow \mathrm{cost}_{\mathrm{zigzig}} \le 3(r'(x) - r(x))
\end{aligned}
$$

# Splay: Zigzig Case



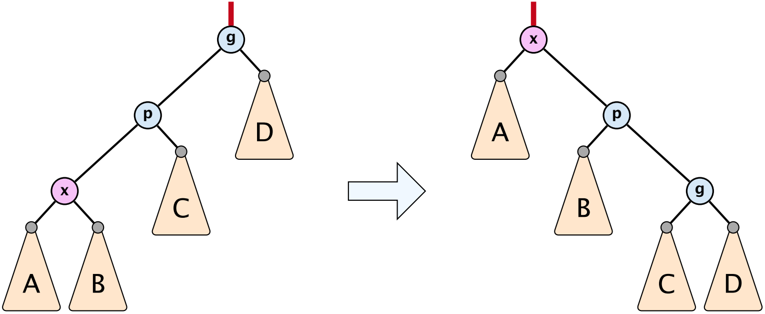

$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\le r'(x) + r'(g) - r(x) - r(x) \\
&= r'(x) + r'(g) + r(x) - 3r'(x) + 3r'(x) - r(x) - 2r(x) \\
&= -2r'(x) + r'(g) + r(x) + 3(r'(x) - r(x)) \\
&\le -2 + 3(r'(x) - r(x)) \quad \Rightarrow \mathrm{cost}_{\mathrm{zigzig}} \le 3(r'(x) - r(x))
\end{aligned}
$$

# Splay: Zigzig Case



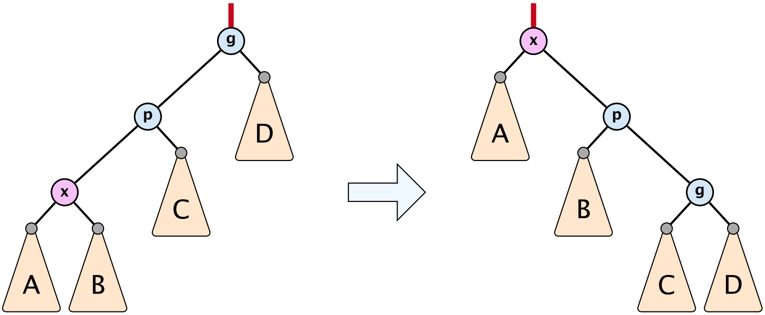

$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\leq r'(x) + r'(g) - r(x) - r(x) \\
&= r'(x) + r'(g) + r(x) - 3r'(x) + 3r'(x) - r(x) - 2r(x) \\
&= -2r'(x) + r'(g) + r(x) + 3(r'(x) - r(x)) \\
&\leq -2 + 3(r'(x) - r(x)) \quad \Rightarrow \mathrm{cost}_{\mathrm{zigzig}} \leq 3(r'(x) - r(x))
\end{aligned}
$$

# Splay: Zigzig Case



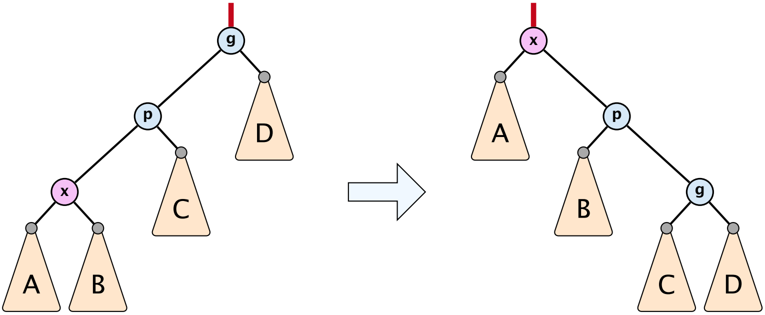

$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\leq r'(x) + r'(g) - r(x) - r(x) \\
&= r'(x) + r'(g) + r(x) - 3r'(x) + 3r'(x) - r(x) - 2r(x) \\
&= -2r'(x) + r'(g) + r(x) + 3(r'(x) - r(x)) \\
&\leq -2 + 3(r'(x) - r(x)) \quad \Rightarrow \mathrm{cost}_{\mathrm{zigzig}} \leq 3(r'(x) - r(x))
\end{aligned}
$$

# Splay: Zigzig Case




$$\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\le r'(x) + r'(g) - r(x) - r(x) \\
&= r'(x) + r'(g) + r(x) - 3r'(x) + 3r'(x) - r(x) - 2r(x) \\
&= -2r'(x) + r'(g) + r(x) + 3(r'(x) - r(x)) \\
&\le -2 + 3(r'(x) - r(x)) \quad \Rightarrow \text{cost}_{\text{zigzig}} \le 3(r'(x) - r(x))
\end{aligned}$$

# Splay: Zigzig Case



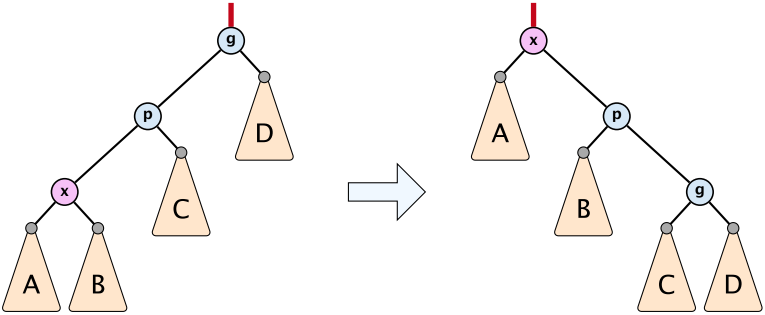

$$\frac{1}{2}\Big(r(x) + r'(g) - 2r'(x)\Big)$$

$$= \frac{1}{2}\Big(\log(s(x)) + \log(s'(g)) - 2\log(s'(x))\Big)$$

$$= \frac{1}{2}\log\Big(\frac{s(x)}{s'(x)}\Big) + \frac{1}{2}\log\Big(\frac{s'(g)}{s'(x)}\Big)$$

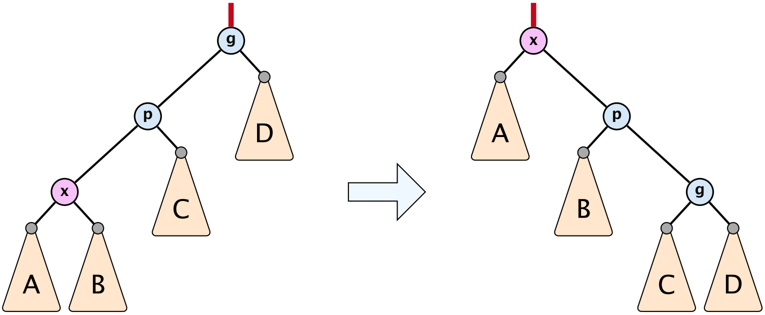
$$\leq \log\Big(\frac{1}{2}\frac{s(x)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\Big) \leq \log\Big(\frac{1}{2}\Big) = -1$$

# Splay: Zigzig Case




$$
\begin{aligned}
&\frac{1}{2}\Big(r(x) + r'(g) - 2r'(x)\Big)\\
&\qquad = \frac{1}{2}\Big(\log(s(x)) + \log(s'(g)) - 2\log(s'(x))\Big)\\
&\qquad = \frac{1}{2}\log\Big(\frac{s(x)}{s'(x)}\Big) + \frac{1}{2}\log\Big(\frac{s'(g)}{s'(x)}\Big)\\
&\qquad \le \log\Big(\frac{1}{2}\frac{s(x)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\Big) \le \log\Big(\frac{1}{2}\Big) = -1
\end{aligned}
$$

# Splay: Zigzig Case



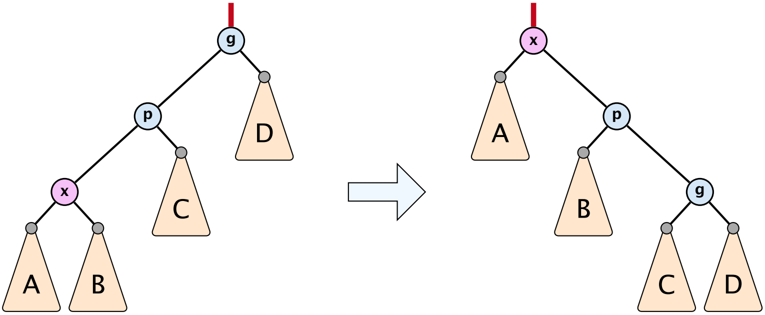

$$\frac{1}{2}\Big(r(x) + r'(g) - 2r'(x)\Big)$$

$$= \frac{1}{2}\Big(\log(s(x)) + \log(s'(g)) - 2\log(s'(x))\Big)$$

$$= \frac{1}{2}\log\left(\frac{s(x)}{s'(x)}\right) + \frac{1}{2}\log\left(\frac{s'(g)}{s'(x)}\right)$$

$$\le \log\left(\frac{1}{2}\frac{s(x)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\right) \le \log\left(\frac{1}{2}\right) = -1$$

# Splay: Zigzig Case



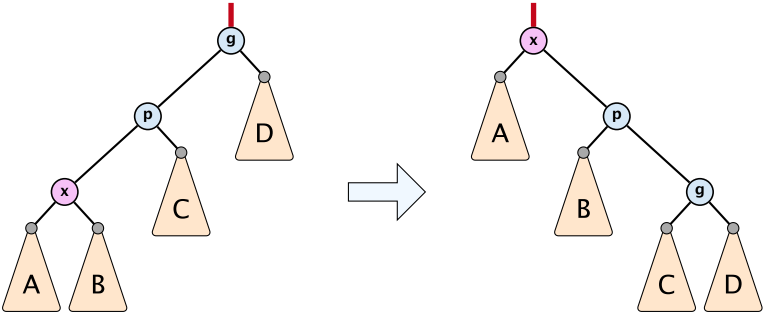

$$\frac{1}{2}\Big(r(x) + r'(g) - 2r'(x)\Big)$$

$$= \frac{1}{2}\Big(\log(s(x)) + \log(s'(g)) - 2\log(s'(x))\Big)$$

$$= \frac{1}{2}\log\Big(\frac{s(x)}{s'(x)}\Big) + \frac{1}{2}\log\Big(\frac{s'(g)}{s'(x)}\Big)$$

$$\leq \log\Big(\frac{1}{2}\frac{s(x)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\Big) \leq \log\Big(\frac{1}{2}\Big) = -1$$

# Splay: Zigzig Case



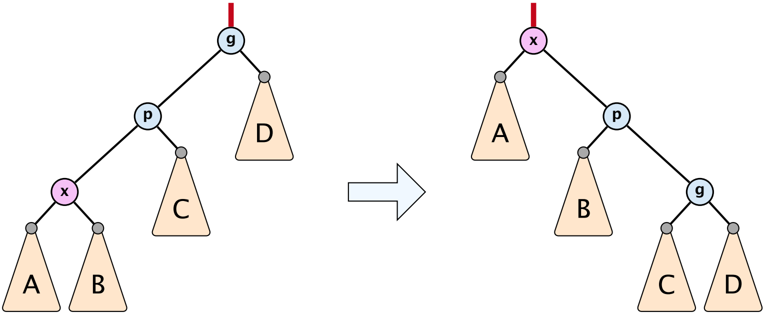

$$\frac{1}{2}\Big(r(x) + r'(g) - 2r'(x)\Big)$$

$$= \frac{1}{2}\Big(\log(s(x)) + \log(s'(g)) - 2\log(s'(x))\Big)$$

$$= \frac{1}{2}\log\Big(\frac{s(x)}{s'(x)}\Big) + \frac{1}{2}\log\Big(\frac{s'(g)}{s'(x)}\Big)$$

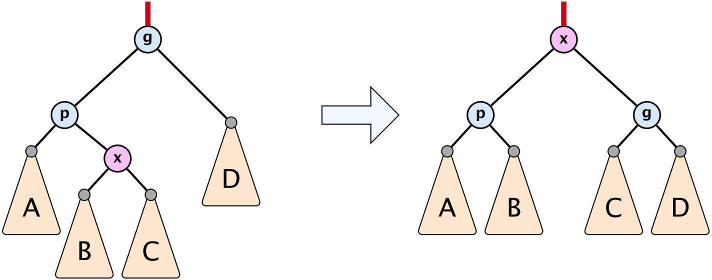
$$\le \log\Big(\frac{1}{2}\frac{s(x)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\Big) \le \log\Big(\frac{1}{2}\Big)$$

# Splay: Zigzig Case



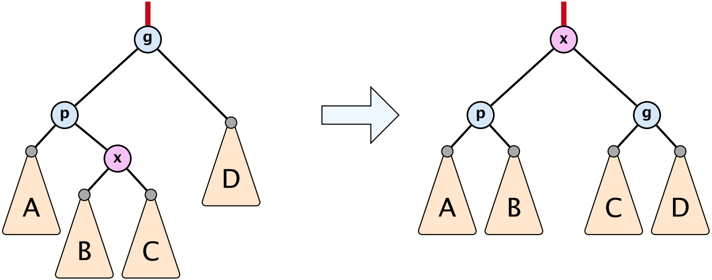

$$
\begin{aligned}
\frac{1}{2}\Big(r(x) + r'(g) - 2r'(x)\Big) \\
&= \frac{1}{2}\Big(\log(s(x)) + \log(s'(g)) - 2\log(s'(x))\Big) \\
&= \frac{1}{2}\log\Big(\frac{s(x)}{s'(x)}\Big) + \frac{1}{2}\log\Big(\frac{s'(g)}{s'(x)}\Big) \\
&\le \log\Big(\frac{1}{2}\frac{s(x)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\Big) \le \log\Big(\frac{1}{2}\Big) = -1
\end{aligned}
$$

# Splay: Zigzag Case



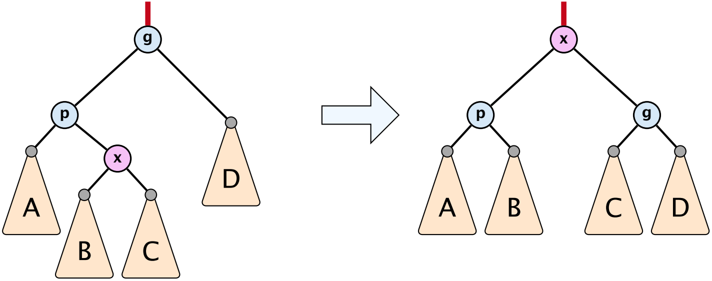

$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\leq r'(p) + r'(g) - r(x) - r(x) \\
&= r'(p) + r'(g) - 2r'(x) + 2r'(x) - 2r(x) \\
&\leq -2 + 2(r'(x) - r(x)) \quad \Rightarrow \mathrm{cost}_{\mathrm{zigzag}} \leq 3(r'(x) - r(x))
\end{aligned}
$$

# Splay: Zigzag Case



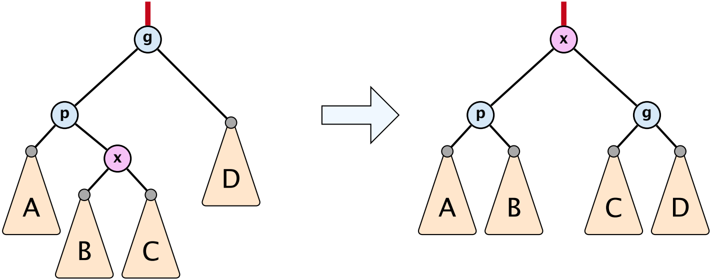

$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\le r'(p) + r'(g) - r(x) - r(x) \\
&= r'(p) + r'(g) - 2r'(x) + 2r'(x) - 2r(x) \\
&\le -2 + 2(r'(x) - r(x)) \qquad \Rightarrow \mathrm{cost}_{\mathrm{zigzag}} \le 3(r'(x) - r(x))
\end{aligned}
$$

# Splay: Zigzag Case



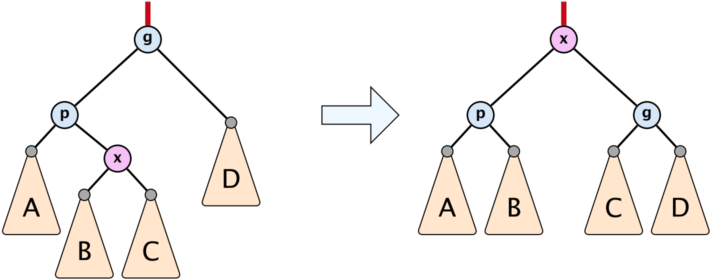

$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\le r'(p) + r'(g) - r(x) - r(x) \\
&= r'(p) + r'(g) - 2r'(x) + 2r'(x) - 2r(x) \\
&\le -2 + 2(r'(x) - r(x)) \qquad \Rightarrow \text{cost}_{\text{zigzag}} \le 3(r'(x) - r(x))
\end{aligned}
$$

# Splay: Zigzag Case



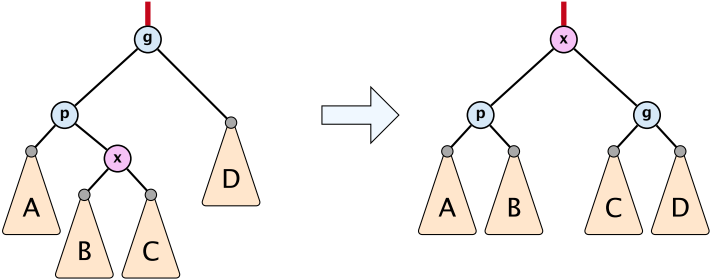

$$\Delta\Phi = r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g)$$
$$= r'(p) + r'(g) - r(x) - r(p)$$
$$\leq r'(p) + r'(g) - r(x) - r(x)$$
$$= r'(p) + r'(g) - 2r'(x) + 2r'(x) - 2r(x)$$
$$\leq -2 + 2(r'(x) - r(x)) \quad \Rightarrow \text{cost}_{\text{zigzag}} \leq 3(r'(x) - r(x))$$

# Splay: Zigzag Case



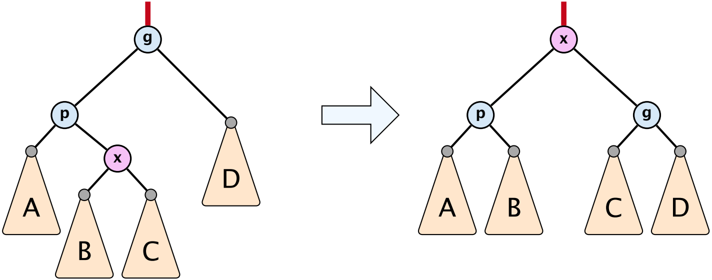

$$
\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\leq r'(p) + r'(g) - r(x) - r(x) \\
&= r'(p) + r'(g) - 2r'(x) + 2r'(x) - 2r(x) \\
&\leq -2 + 2(r'(x) - r(x)) \quad \Rightarrow \mathrm{cost}_{\mathrm{zigzag}} \leq 3(r'(x) - r(x))
\end{aligned}
$$

# Splay: Zigzag Case



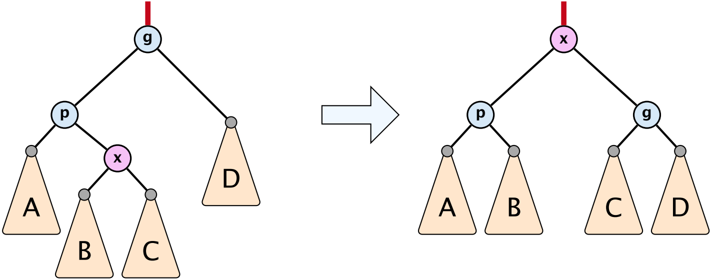

$$\begin{aligned}
\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\
&= r'(p) + r'(g) - r(x) - r(p) \\
&\le r'(p) + r'(g) - r(x) - r(x) \\
&= r'(p) + r'(g) - 2r'(x) + 2r'(x) - 2r(x) \\
&\le -2 + 2(r'(x) - r(x)) \qquad \Rightarrow \mathrm{cost}_{\mathrm{zigzag}} \le 3(r'(x) - r(x))
\end{aligned}$$

# Splay: Zigzag Case




$$\begin{aligned}\Delta\Phi &= r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g)\\ &= r'(p) + r'(g) - r(x) - r(p)\\ &\le r'(p) + r'(g) - r(x) - r(x)\\ &= r'(p) + r'(g) - 2r'(x) + 2r'(x) - 2r(x)\\ &\le -2 + 2(r'(x) - r(x)) \quad \Rightarrow \text{cost}_{\text{zigzag}} \le 3(r'(x) - r(x))\end{aligned}$$

# Splay: Zigzag Case

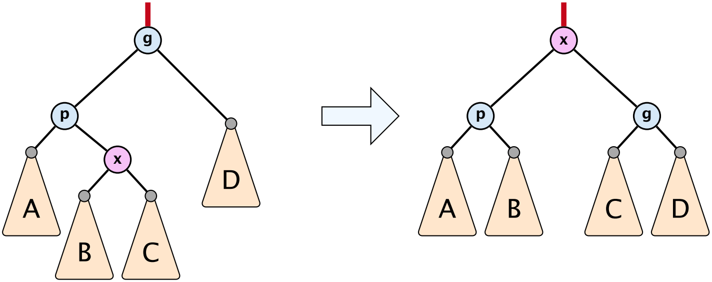



$$\frac{1}{2}\Big(r'(p) + r'(g) - 2r'(x)\Big)$$

$$= \frac{1}{2}\Big(\log(s'(p)) + \log(s'(g)) - 2\log(s'(x))\Big)$$

$$\leq \log\left(\frac{1}{2}\frac{s'(p)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\right) \leq \log\left(\frac{1}{2}\right) = -1$$

# Splay: Zigzag Case

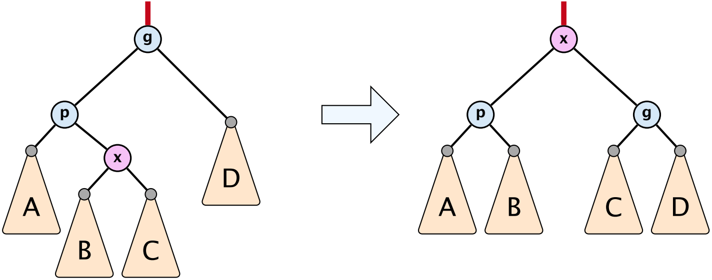



$$\frac{1}{2}\Big(r'(p) + r'(g) - 2r'(x)\Big)$$

$$= \frac{1}{2}\Big(\log(s'(p)) + \log(s'(g)) - 2\log(s'(x))\Big)$$

$$\leq \log\left(\frac{1}{2}\frac{s'(p)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\right) \leq \log\left(\frac{1}{2}\right) = -1$$

# Splay: Zigzag Case

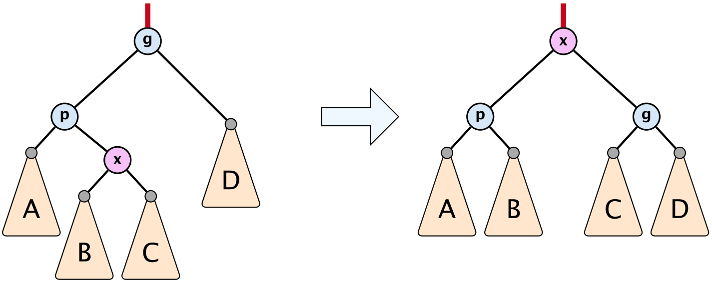



$$\frac{1}{2}\Big(r'(p) + r'(g) - 2r'(x)\Big)$$

$$= \frac{1}{2}\Big(\log(s'(p)) + \log(s'(g)) - 2\log(s'(x))\Big)$$

$$\leq \log\left(\frac{1}{2}\frac{s'(p)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\right) \leq \log\left(\frac{1}{2}\right) = -1$$

# Splay: Zigzag Case

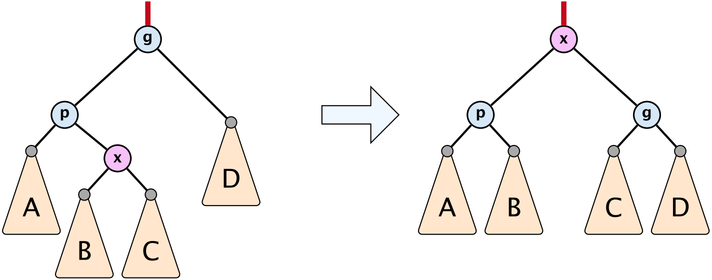



$$\frac{1}{2}\Big(r'(p) + r'(g) - 2r'(x)\Big)$$

$$= \frac{1}{2}\Big(\log(s'(p)) + \log(s'(g)) - 2\log(s'(x))\Big)$$

$$\le \log\left(\frac{1}{2}\frac{s'(p)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\right) \le \log\left(\frac{1}{2}\right) = -1$$

# Splay: Zigzag Case




$$\frac{1}{2}\Big(r'(p) + r'(g) - 2r'(x)\Big)$$

$$= \frac{1}{2}\Big(\log(s'(p)) + \log(s'(g)) - 2\log(s'(x))\Big)$$

$$\le \log\left(\frac{1}{2}\frac{s'(p)}{s'(x)} + \frac{1}{2}\frac{s'(g)}{s'(x)}\right) \le \log\left(\frac{1}{2}\right) = -1$$

Amortized cost of the whole splay operation:

$$
\begin{aligned}
&\leq 1 + 1 + \sum_{\text{steps } t} 3(r_t(x) - r_{t-1}(x)) \\
&= 2 + r(\text{root}) - r_0(x) \\
&\leq \mathcal{O}(\log n)
\end{aligned}
$$