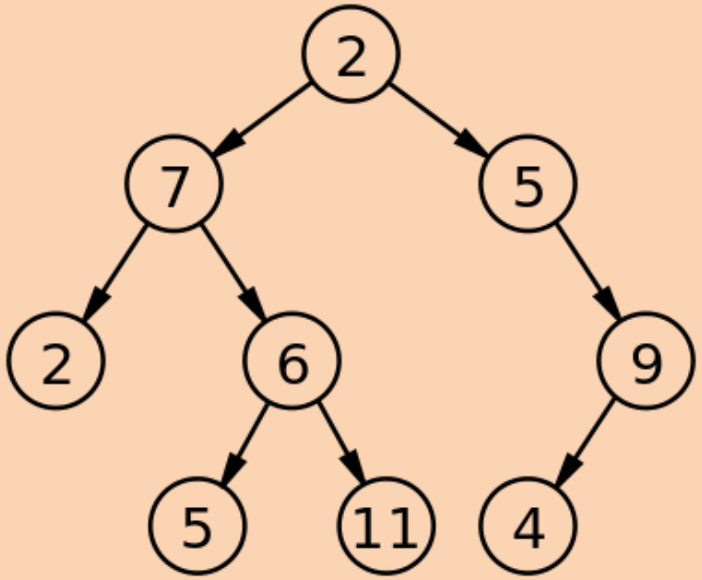
ساختمان‌های داده

درخت‌های قرمز-سیاه تکیه بر چپ

Left-leaning red-black trees



مدرس:

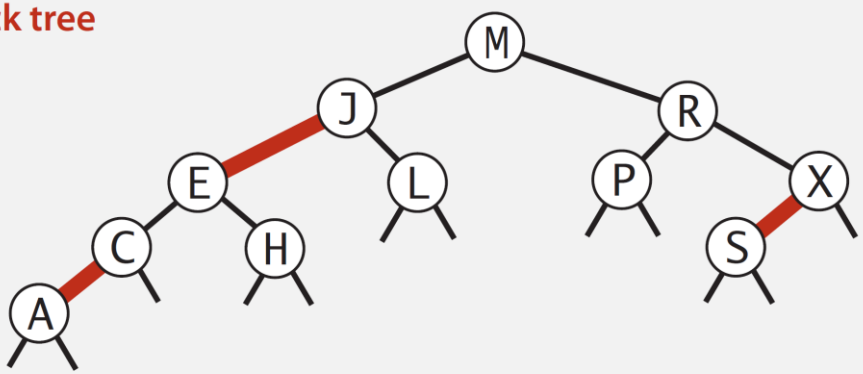
سید کمال الدین غیاثی شیرازی

تعریف درخت‌های قرمز–سیاه تکیه برچپ

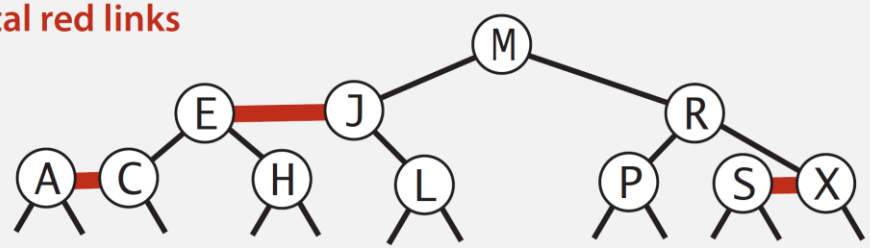
- یک درخت قرمز-سیاه تکیه بر چپ یک درخت قرمز-سیاه است که در آن همه‌ی گره‌های قرمز فرزندان چپ هستند.
- هر درخت قرمزسیاه تکیه بر چپ دقیقا با یک درخت ۲-۳ معادل است.

تناظر یك به یك درخت‌های قرمز–سیاه تکیه برچپ با درخت‌های 2-3

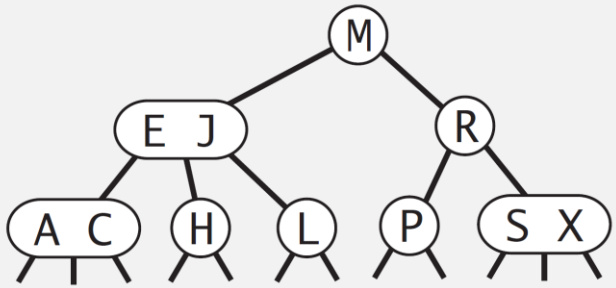
red–black tree



horizontal red links



2-3 tree

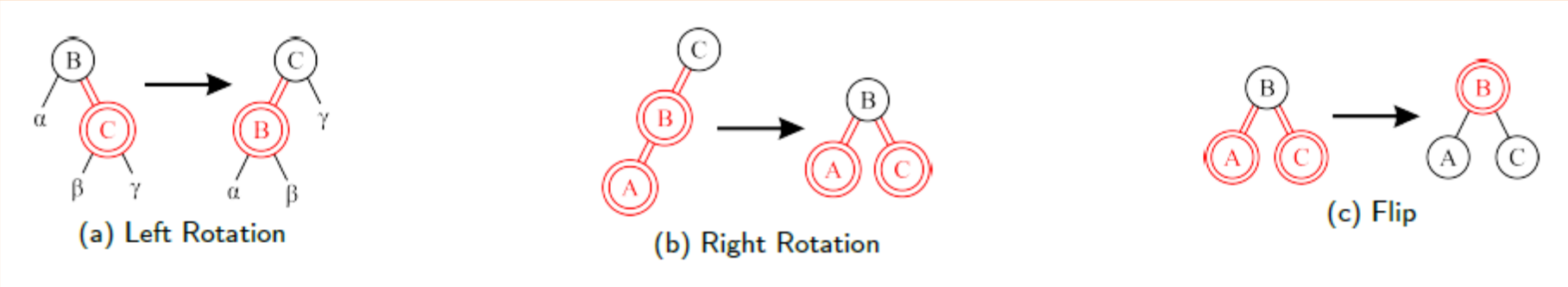


الگوریتم درج

- محل گره جدید را طبق قانون درخت جستجوی دودویی به دست آور.
- گره جدید را با رنگ قرمز درج کن.
- با شروع از گره جاری تا رسیدن به ریشه قوانین صفحه‌ی بعد را اجرا کن.

- **توجه:** عملیات درج و حذف در LLRB به صورت بازگشتی پیاده‌سازی می‌شود.
  - درج عنصر در درخت در مرحله‌ی بالا به پایین انجام می‌شود.
  - عمل FixUp و رفع مشکلات درخت LLRB هنگام بازگشت از توابع (پایین به بالا) انجام می‌شود.

قوانین اصلاح درخت پس از عمل درج



- عملیات به ترتیب (a) سپس (b) و در نهایت (c) اجرا می‌شوند.

کد درج در LLRB

```
inline Node *put(Node *h, Key key, Value value) {
    if (h == nil)
        return new Node(key, value, nil);

    if (key == h->key)
        h->value = value;
    else if (key < h->key)
        h->left = put(h->left, key, value);
    else
        h->right = put(h->right, key, value);



    if (isRed(h->right))
        h = rotateLeft(h);

    if (isRed(h->left) && isRed(h->left->left))
        h = rotateRight(h);

    if (isRed(h->left) && isRed(h->right))
        colorFlip(h);

    return h;
}
```

مشاهده‌ی عملکرد الگوریتم درج در یک مثال جامع

- فایل llrb-insert-images.pdf را ببینید.

الگوریتم حذف در LLRB

- الگوریتم کلی حذف بدین شکل است که از ریشه شروع می کنیم و گرهی را که باید حذف شود جستجو می کنیم.
- در حین پایین آمدن همواره این خاصیت را حفظ می کنیم که در سمتی که پایین می‌رویم یا فرزند و یا نوه قرمز باشند.
- حذف از گره درجه‌ی ۲ نهایتا به حذف مقدار مینیمم زیر درخت راست آن تبدیل می‌شود.
- الگوریتم تضمین می‌کند که حذف واقعی نهایتا بر روی یک گره برگ قرمز انجام می‌شود.

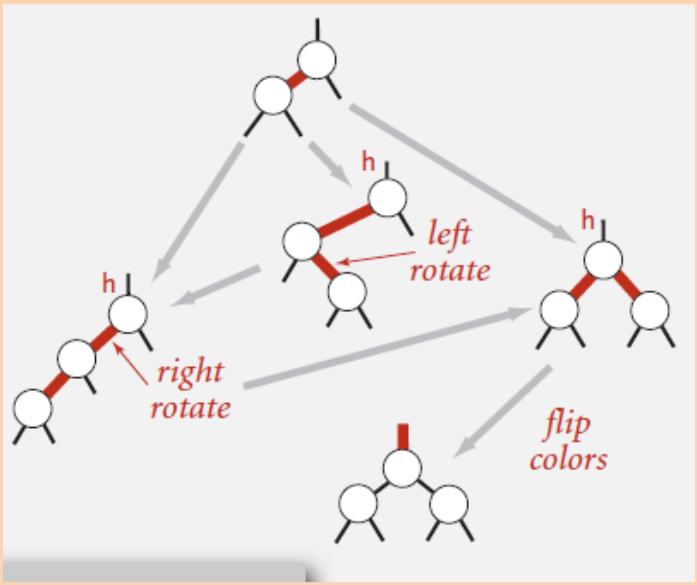
fixUp

```
Node *fixUp(Node *h) {
    if (isRed(h->right))
        h = rotateLeft(h);

    if (isRed(h->left) && isRed(h->left->left))
        h = rotateRight(h);

    if (isRed(h->left) && isRed(h->right))
        colorFlip(h);

    return h;
}
```



MoveRedLeft & MoveRedRight

```
Node *moveRedLeft(Node *h) {
    // Assuming that h is red and both h.left and h.left.left
    // are black, make h.left or one of its children red.
    colorFlip(h);
    if (h->right != nil && isRed(h->right->left)) {
        h->right = rotateRight(h->right);
        h = rotateLeft(h);
        colorFlip(h);
    }
    return h;
}

Node *moveRedRight(Node *h) {
    // Assuming that h is red and both h.right and h.right.left
    // are black, make h.right or one of its children red.
    colorFlip(h);

    if (h->left != nil && isRed(h->left->left)) {
        h = rotateRight(h);
        colorFlip(h);
    }
    return h;
}
```

deleteMin

```
Node *deleteMin(Node *h) {
    if (h->left == nil) {
        delete h;
        return nil;
    }
    if (!isRed(h->left) && !isRed(h->left->left))
        h = moveRedLeft(h);

    h->left = deleteMin(h->left);

    return fixUp(h);
}
```

Remove

```
Node *remove(Node *h, Key key) {
    if (h != nil) {
        if (key < h->key) {
            if (!isRed(h->left) && h->left != nil && !isRed(h->left->left))
                h = moveRedLeft(h);
            h->left = remove(h->left, key);
        } else {
            if (isRed(h->left))
                h = rotateRight(h);
            if ((key == h->key) && (h->right == nil)) {
                delete h;
                return nil;
            }
            if (!isRed(h->right) && h->right != nil && !isRed(h->right->left))
                h = moveRedRight(h);
            if (key == h->key) {
                h->value = get(h->right, min(h->right));
                h->key = min(h->right);
                h->right = deleteMin(h->right);
            } else
                h->right = remove(h->right, key);
        }

        return fixUp(h);
    }
    return nil;
}
```

مشاهده‌ی عملکرد الگوریتم حذف در یك مثال جامع

- فایل llrb-delete-images.pdf را ببینید.

پیاده‌سازی

- پیاده‌سازی ما از این الگوریتم به زبان C++ را می‌توانید اینجا مشاهده کنید.

- https://github.com/k-ghiasi/RedBlackTrees/blob/main/LeftLeaningRedBlackBST.h

- این کد از روی کد اصلی که به زبان جاوا است نوشته شده است:

- https://algs4.cs.princeton.edu/33balanced/RedBlackBST.java.html