

Optimisation changes for tree.h

```
--- tree.h.orig 2008-03-21 14:07:15.000000000 +0100
+++ tree.h 2010-05-03 19:35:24.000000000 +0200
@@ -324,15 +324,11 @@
     RB_COLOR(elm, field) = RB_RED;
 } while (/*CONSTCOND*/ 0)
```

Removed RB_SET_BLACKRED macro. It makes code difficult to read, and after the changes there is no need to colour black and red in one statement.

```

-#define RB_SET_BLACKRED(black, red, field) do {
-    RB_COLOR(black, field) = RB_BLACK;
-    RB_COLOR(red, field) = RB_RED;
-} while (/*CONSTCOND*/ 0)
-
#ifdef RB_AUGMENT
#define RB_AUGMENT(x) (void)(x)
#endif
```

Rotations

This comment is important. Other optimisations assume that, after the rotation, **tmp** points to the topmost element.

```

/* After rotation, tmp will be the new top element (new elm's parent) */
#define RB_ROTATE_LEFT(head, elm, tmp, field) do {
    (tmp) = RB_RIGHT(elm, field);
    if ((RB_RIGHT(elm, field) = RB_LEFT(tmp, field)) != NULL) {
```

This is a cosmetic change. Use RB_ROOT macro, as other macros are used as well.

```
@@ -345,7 +341,7 @@
    else
        RB_RIGHT(RB_PARENT(elm, field), field) = (tmp);
    } else
        (head)->rbh_root = (tmp);
+    RB_ROOT(head) = (tmp);
    RB_LEFT(tmp, field) = (elm);
    RB_PARENT(elm, field) = (tmp);
    RB_AUGMENT(tmp);
@@ -365,7 +361,7 @@
    else
        RB_RIGHT(RB_PARENT(elm, field), field) = (tmp);
    } else
        (head)->rbh_root = (tmp);
+    RB_ROOT(head) = (tmp);
    RB_RIGHT(tmp, field) = (elm);
    RB_PARENT(elm, field) = (tmp);
    RB_AUGMENT(tmp);
```

Element insertion

For all the cases, grand-parent element has always been coloured red. Now colour it red beforehand.

```
@@ -405,41 +401,40 @@
    while ((parent = RB_PARENT(elm, field)) != NULL &&
        RB_COLOR(parent, field) == RB_RED) {
        gparent = RB_PARENT(parent, field);
+    RB_COLOR(gparent, field) = RB_RED;
        if (parent == RB_LEFT(gparent, field)) {
            tmp = RB_RIGHT(gparent, field);
            if (tmp && RB_COLOR(tmp, field) == RB_RED) {
                RB_COLOR(tmp, field) = RB_BLACK;
-    RB_SET_BLACKRED(parent, gparent, field);
```

```

+         RB_COLOR(parent, field) = RB_BLACK; \
         elm = gparent; \
         continue; \
     } \

```

Utilise the fact, that `RB_ROTATE_LEFT` sets `tmp` to the topmost element, in this case `elm`'s new parent.

```

         if (RB_RIGHT(parent, field) == elm) { \
             RB_ROTATE_LEFT(head, parent, tmp, field);\
-         tmp = parent; \
-         parent = elm; \
-         elm = tmp; \
+         elm = parent; \
+         parent = tmp; \
        } \
-         RB_SET_BLACKRED(parent, gparent, field); \
+         RB_COLOR(parent, field) = RB_BLACK; \
         RB_ROTATE_RIGHT(head, gparent, tmp, field); \
    } else { \
        tmp = RB_LEFT(gparent, field); \
        if (tmp && RB_COLOR(tmp, field) == RB_RED) { \
            RB_COLOR(tmp, field) = RB_BLACK; \
-            RB_SET_BLACKRED(parent, gparent, field);\
+            RB_COLOR(parent, field) = RB_BLACK; \
            elm = gparent; \
            continue; \
        } \
        if (RB_LEFT(parent, field) == elm) { \
            RB_ROTATE_RIGHT(head, parent, tmp, field);\
-         tmp = parent; \
-         parent = elm; \
-         elm = tmp; \
+         elm = parent; \
+         parent = tmp; \
        } \
-         RB_SET_BLACKRED(parent, gparent, field); \
+         RB_COLOR(parent, field) = RB_BLACK; \
         RB_ROTATE_LEFT(head, gparent, tmp, field); \
    } \
} \
- RB_COLOR(head->rbh_root, field) = RB_BLACK; \
+ RB_COLOR(RB_ROOT(head), field) = RB_BLACK; \
} \
\
\
attr void \

```

Element removal

Converting `RB_SET_BLACKRED` to `RB_COLOR`.

```

@@ -451,7 +446,8 @@
         if (RB_LEFT(parent, field) == elm) { \
             tmp = RB_RIGHT(parent, field); \
             if (RB_COLOR(tmp, field) == RB_RED) { \
-                 RB_SET_BLACKRED(tmp, parent, field); \
+                 RB_COLOR(tmp, field) = RB_BLACK; \
+                 RB_COLOR(parent, field) = RB_RED; \
                 RB_ROTATE_LEFT(head, parent, tmp, field);\
                 tmp = RB_RIGHT(parent, field); \
             } \

```

In this case the left child of `tmp` always exists and is red (otherwise we would end up in other cases). So, there is no need to explicitly check of its existence. Also, get rid of `oleft` – use `elm` instead, as it is not used (until the end of the condition).

```

@@ -465,18 +461,12 @@
        } else {
            if (RB_RIGHT(tmp, field) == NULL || \
                RB_COLOR(RB_RIGHT(tmp, field), field) == RB_BLACK) {\
-                struct type *oleft;
-                if ((oleft = RB_LEFT(tmp, field)) \
-                    != NULL)
-                    RB_COLOR(oleft, field) = RB_BLACK;\

```

Colouring **tmp** red is useless – later it is coloured black.

```

-                RB_COLOR(tmp, field) = RB_RED; \
-                RB_ROTATE_RIGHT(head, tmp, oleft, field);\

```

Again, make use of the fact, that `RB_ROTATE_RIGHT` sets its third argument to the new topmost element. In this case, after the rotation, **parent**'s right child is **tmp**'s parent, which is stored in **elm**.

```

-                tmp = RB_RIGHT(parent, field); \
+                RB_ROTATE_RIGHT(head, tmp, elm, field);\
+                tmp = elm;
        }

```

The right child of **tmp** must exist, otherwise it is the condition above.

```

        RB_COLOR(tmp, field) = RB_COLOR(parent, field);\
        RB_COLOR(parent, field) = RB_BLACK; \
-        if (RB_RIGHT(tmp, field))
-            RB_COLOR(RB_RIGHT(tmp, field), field) = RB_BLACK;\
+        RB_COLOR(RB_RIGHT(tmp, field), field) = RB_BLACK;\
        RB_ROTATE_LEFT(head, parent, tmp, field);\
        elm = RB_ROOT(head);
        break;

```

Below is the symmetric case.

```

@@ -484,7 +474,8 @@
        } else {
            tmp = RB_LEFT(parent, field);
            if (RB_COLOR(tmp, field) == RB_RED) {
-                RB_SET_BLACKRED(tmp, parent, field);\
+                RB_COLOR(tmp, field) = RB_BLACK; \
+                RB_COLOR(parent, field) = RB_RED; \
                RB_ROTATE_RIGHT(head, parent, tmp, field);\
                tmp = RB_LEFT(parent, field);
            }
@@ -498,18 +489,12 @@
        } else {
            if (RB_LEFT(tmp, field) == NULL || \
                RB_COLOR(RB_LEFT(tmp, field), field) == RB_BLACK) {\
-                struct type *oright;
-                if ((oright = RB_RIGHT(tmp, field)) \
-                    != NULL)
-                    RB_COLOR(oright, field) = RB_BLACK;\
-                RB_COLOR(tmp, field) = RB_RED; \
-                RB_ROTATE_LEFT(head, tmp, oright, field);\
-                tmp = RB_LEFT(parent, field); \
+                RB_ROTATE_LEFT(head, tmp, elm, field);\
+                tmp = elm;
            }
            RB_COLOR(tmp, field) = RB_COLOR(parent, field);\
            RB_COLOR(parent, field) = RB_BLACK; \
-            if (RB_LEFT(tmp, field))
-                RB_COLOR(RB_LEFT(tmp, field), field) = RB_BLACK;\
+            RB_COLOR(RB_LEFT(tmp, field), field) = RB_BLACK;\
            RB_ROTATE_RIGHT(head, parent, tmp, field);\
            elm = RB_ROOT(head);

```

break;

\