MA346m Quiz 02 Friday 20/10/17

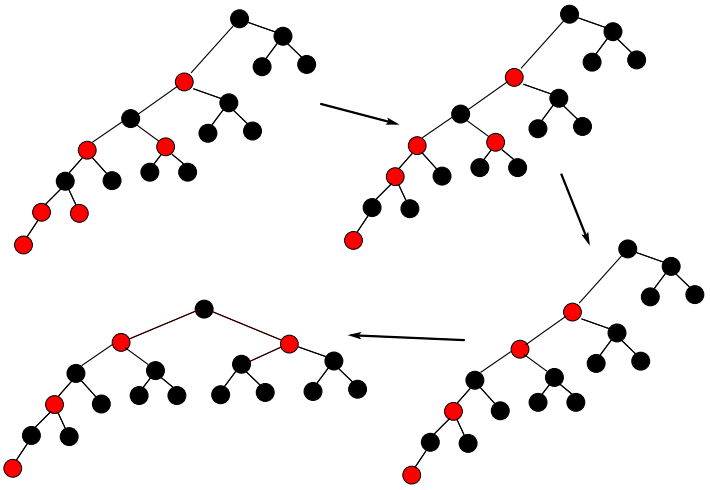
(1) A complete binary tree is one where no node has just one child. Binary operators provide an example. It is possible to construct a complete binary tree with n nodes from a single array with n 1-bit entries, indicating which nodes are leaves and which are not, assuming that the order of array entries follows *preorder*. Write a recursive routine

```
BST_NODE * build ( int * pre, int is_leaf[] )
called as
pre = 0;
p = build ( &pre, is_leaf )
```

which builds a tree (with p pointing to its root). `BST_NODE` should have left and right pointers and a `pre_rank` entry. Ignore the parent links.

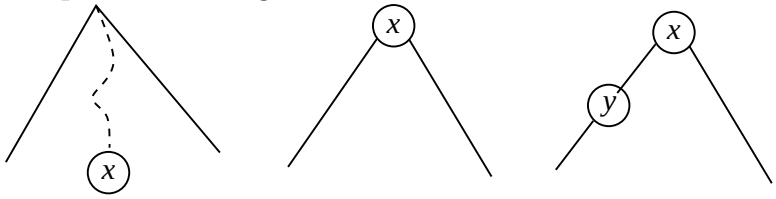
```
Answer
BST_NODE * build ( int * pre, int is_leaf [] )
{
  BST_NODE * new = (BST_NODE *) calloc ( 1, sizeof(BST_NODE));
  new->pre_rank = * pre;
  ++ * pre;
  if ( ! is_leaf [ new->pre_rank ] )
  {
    new->left = build ( pre, is_leaf );
    new->right = build ( pre, is_leaf );
  }
  return new;
}
```

(2). Apply fix double red to this tree.



(3). Splay trees achieve good amortised times for various operations. The idea is when performing a certain operation, bring a suitable node to the root by splaying. One must make allowances for increased potential when trees are joined, nodes are inserted, etcetera.

Design an 'insert' operation with good amortised time.



Answer. Suppose that searching for a key y fell off the 'left' of a node x. Splay x to root at cost $\leq 1 + 3\log_2 n$. Then inserting x raises the total weight of all descendants of y from $1/n$ to at most $1 - 1/n$ and for x it increases by $1/n$. Let r be the rank of x after the splay and before the insertion. So the potential increase is at most

$$\log_2(1 - 1/n) - \log_2(1/n) + \log_2(r + 1/n) - \log_2(r) \leq \log_2 n + \log_2(1 + 1/nr) \leq \log_2 n + 1.$$

(4). Write a recursive routine `install_ranks`:

```
typedef struct BST_NODE
{
  struct BST_NODE * left, * right;
  int pre_rank, in_rank, post_rank;
} BST_NODE;

Answer
void install_ranks ( int * pre, int * in, int * post,
        BST_NODE * p )
{
  if ( p != NULL )
  {
    p->pre_rank = * pre; ++ * pre;

    install_ranks ( p->left );
    p->inorder_rank = *in; ++ * in;
    install_ranks ( p->right );

    p->post_rank = * post; ++ *post;
  }
}
```