アルゴリズムとデータ構造
演習第７回
ツリー２（二分探索木）

データを二分木に入れて、それを基に探索を行う方法 （アルゴリズムＣ 第２巻 p.13）についてです。

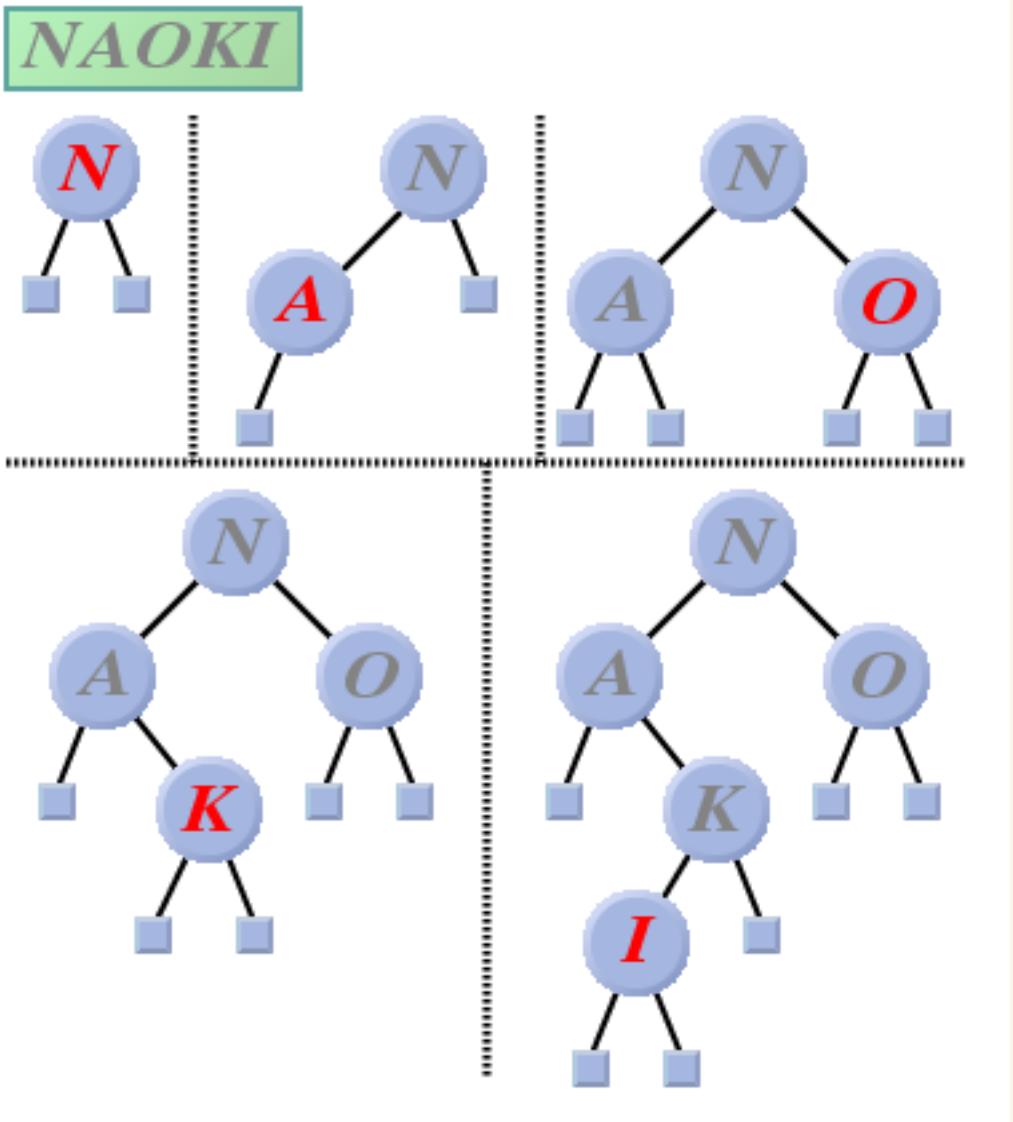
問題１ ［印刷用 PostScript］

(1) 次のデータを順番に空の木に入れていった場合に、作られる木を書きなさい。 ただし、途中の過程も書くこと。

```
O L D F A S H I O N
```

(2) (1)で作られた木を inorder でトラバースした結果を書きなさい。

(1) 木にデータを入れていく （アルゴリズムＣ 第２巻 p.18） ときは、まず木を辿って空いている所（葉）を探し、 見つかったらそこにデータを入れます。 木を辿るときに、左右どちらに行くかは、入れたいデータがそのノードの値よりも 大きいか小さいかで決めます。以下の図を参考にして書いてみてください。



(2) 演習第 6 回でやった inorder で トラバースしてみてください。正しく木が作れていて、 正しくトラバースできれば、アルファベット順に並んでいるはずです。 上の例では `A I K N O` になります。

問題２

再帰を使って、入力されたデータから木を作成する関数

```
void treeinsert_r(char c, NodePointer p);
```

を作成しなさい。引数の `c` は入力されたデータ、 `p` は現在いるノードを表す。 また、結果は inorder でトラバースしたものを 表示。ファイルは ex07-2-skel.c を使用すること。

実行例：

```
% ./a.out
INPUT DATA:  NAOKI
inorder:  A I K N O
```

main で入力を `getchar` 関数などで一文字ずつ読み込んで、 それを順に木に繋いでいきます。繋ぐときには、main で `treeinsert_r(c, head);` というように呼出します。 読み込んだ文字 `c` が入る場所を、head から辿りながら探して 入れる、という意味です。

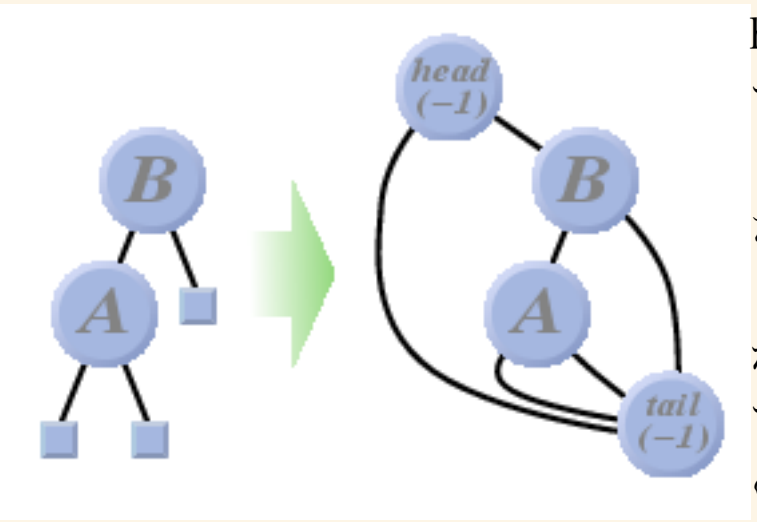
treeinsert_r 関数は再帰で作ります。流れはこのようになります。

1. 現在いるノード p の key と木に繋ぐデータ c を比べて、 左右どちらに辿るかを判定する
2. 辿る先が tail だったら、そこにノードを作って繋ぐ
3. 辿る先が tail ではなかったら、辿った先で 同様の処理を繰り返す（再帰する）

アルゴリズムC 第２巻 p.17 に非再帰版の treeinsert が載っているので参考にしてください。

注意：木のデータ構造

ある木の実際のデータ構造は図のようになっています （アルゴリズムC 第２巻 p.15） 。



head の右のノードが実際の木の根になっていて、 head の左や何も繋っていないところはすべて tail に繋がっています。 また head の key は -1 になっているため、どのような文字データと比べても -1 の方が小さくなります。このような構造になっているため、 上記の流れを head から始めることによって うまく処理することができます（head の左には繋がらないし、最初のデータを空の木に入れる場合に別の処理にしなくても良い）。

問題３

木からデータを削除する関数 delete() を書きなさい。ただし、 削除するときに木を繋ぎ換えるのではなく、 ノードに *deletion* という情報を付加し、 この値によって削除されているかどうかを判定するようにしなさい。 （ex07-3-skel.c）

```
% ./a.out
INPUT DATA:  NAOKI
inorder:  A I K N O

DELETE:  N
inorder:  A I K O
```

データを木から削除するときに、木を繋ぎ換える方法もありますが、 繋ぎ換えはとても複雑になります （アルゴリズムC 第２巻 p.20） 。

ここでは、構造体 *node* に *deletion* というメンバを 増やし、このメンバが 0 ならば削除されていないデータ、 1 ならば削除されたデータとして扱います。あとは削除されたデータは表示しない という形で構いません。

Written by わかまつなおき