講義「アルゴリズムとデータ構造」

第７回　探索のためのデータ構造(２)

大学院情報科学研究院 情報理工学部門
情報知識ネットワーク研究室
喜田拓也

今日の内容

木を探索するアルゴリズム（木の巡回）
- 幅優先探索（breadth-first-search; BFS）
- 深さ優先探索（depth-first-search; DFS）
  - 行きがけ順（preorder traversal；前順）
  - 通りがけ順（inorder traversal；中順）
  - 帰りがけ順（postorder traversal;後順）

ポイント：
- 再帰呼び出しによるDFSの実現
- スタックによる実現＝DFS，キューによる実現＝BFS

木の巡回とは

与えられた根付き木Tのすべての頂点をちょうど一度ずつ訪問（処理）すること

あるいは，木の各頂点がちょうど一度ずつ現れるような，Tの頂点の並びのこと

様々な応用がある！

- 人工知能分野における最適解の探索
- 数式の処理
- 言語解析や意味理解
- HTMLやXMLデータ等の処理

二分探索木にあるすべての要素を小さい順に列挙するとか

木の巡回の種類

巡回の考え方：
　基本は，根から出発して下方へ全ての頂点を訪問する

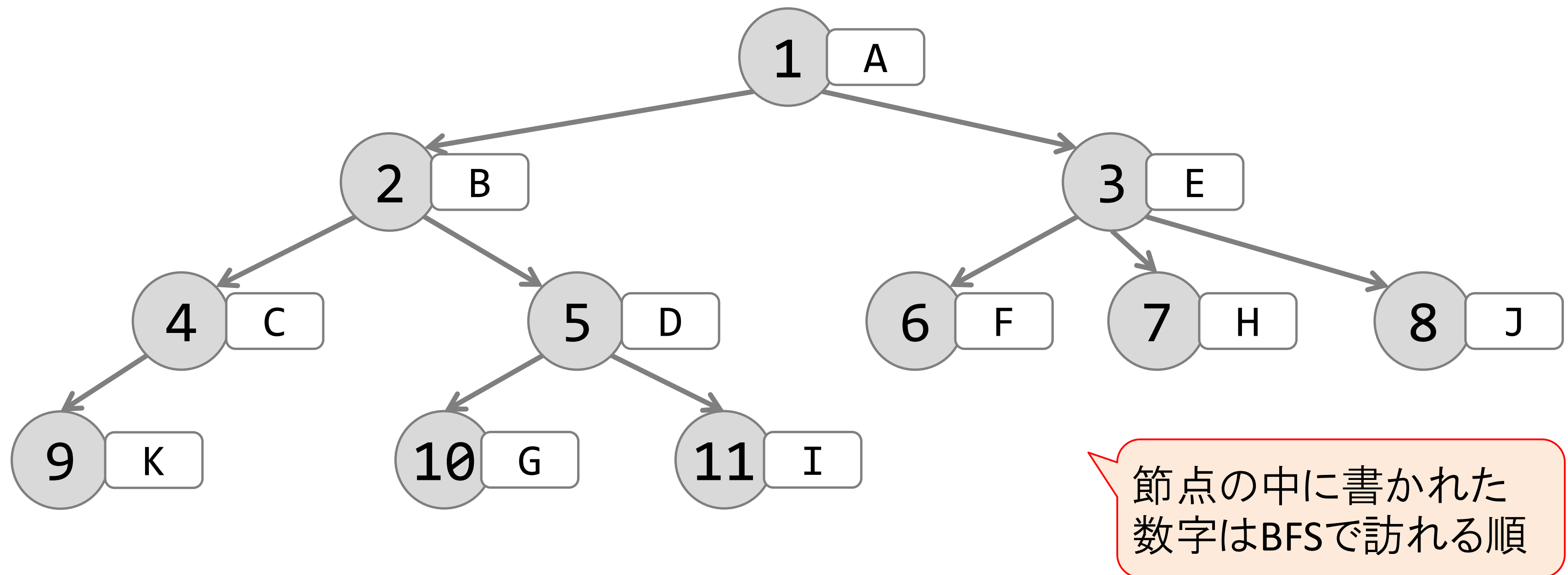
木の探索には大きく分けて2種類ある

- 幅優先探索（breadth-first-search; BFS）
  探索の深さ(レベル)$d = 0, 1, 2, \ldots$を大きくしながら，その深さに属する節点を処理する
- 深さ優先探索（depth-first-search; DFS）
  いま処理している節点から，できるだけ深い方へ処理をすすめていき，葉に到達したら手前に戻ることを繰り返す

幅優先探索(BFS)

BFSは，根から近い順番(深さが浅い順)に木を探索する方法

探索が解に到達したとき，その経路は最短なものとなる



BFSで訪れた順にノードのデータを書き出すと次のようになる

A B E C D F H J K G I

キューによるBFSの実現

【BFSのアルゴリズム】

空のキュー S を用意する

キュー S に木 T の根を登録(enqueue)する

キュー S が空でない間，以下の処理を繰り返す

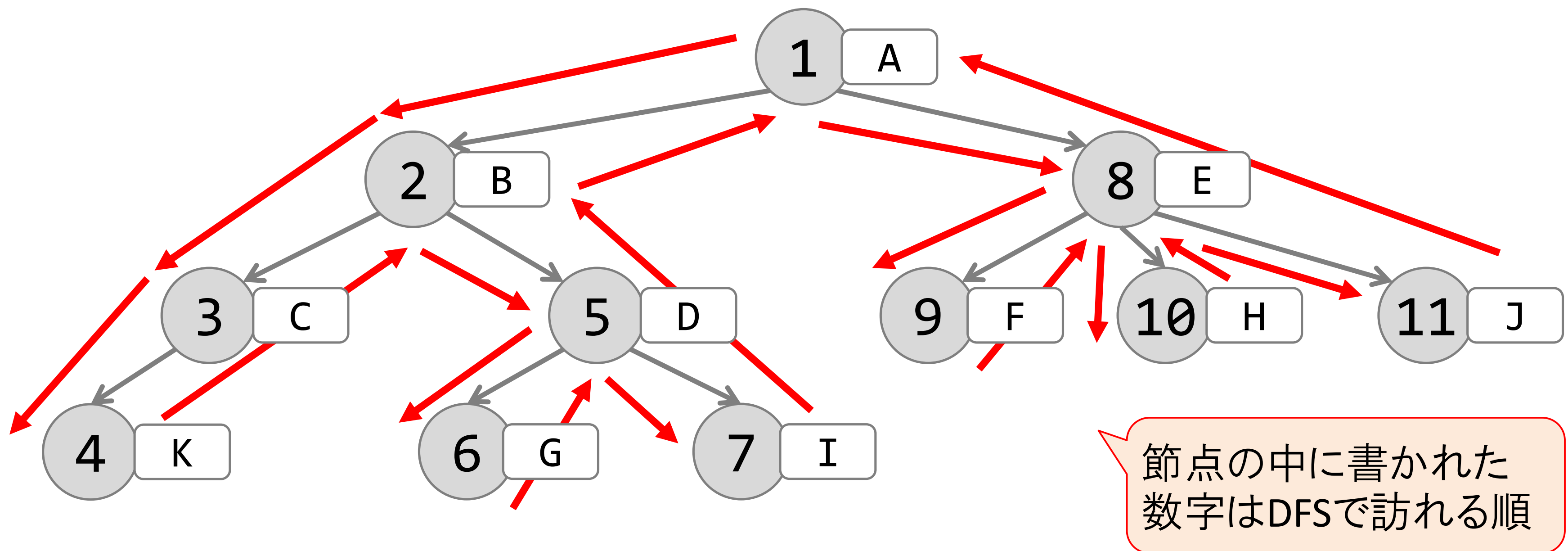
1. キュー S から要素(節点) v を取り出す(dequeue)
2. 節点 v を処理(出力)する
3. 節点 v の左の子があればキューに登録する
4. 節点 v の右の子があればキューに登録する

キューは，入れた順番に取り出されることを思い出そう！

深さ優先探索(DFS)

DFSは，根から順に遷移できなくなるまで(葉まで)進み，
遷移できなくなったら手前(親)に戻ることを繰り返す

バックトラックはDFSを利用した解の探索手法(うまくやると速い)



DFSで訪れた順にノードのデータを書き出すと次のようになる

A B C K D G I E F H J

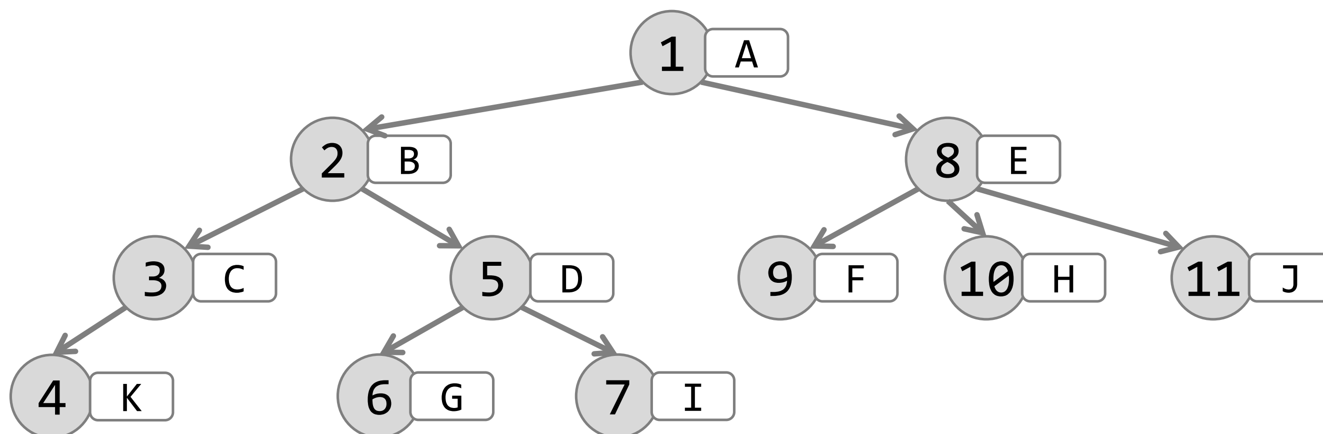
DFSの際の節点の処理順について

DFSにおいて，巡回の処理順には次の三つの方法がある

行きがけ順（前置順，前順，preorder）

通りがけ順（中置順，中順，inorder）

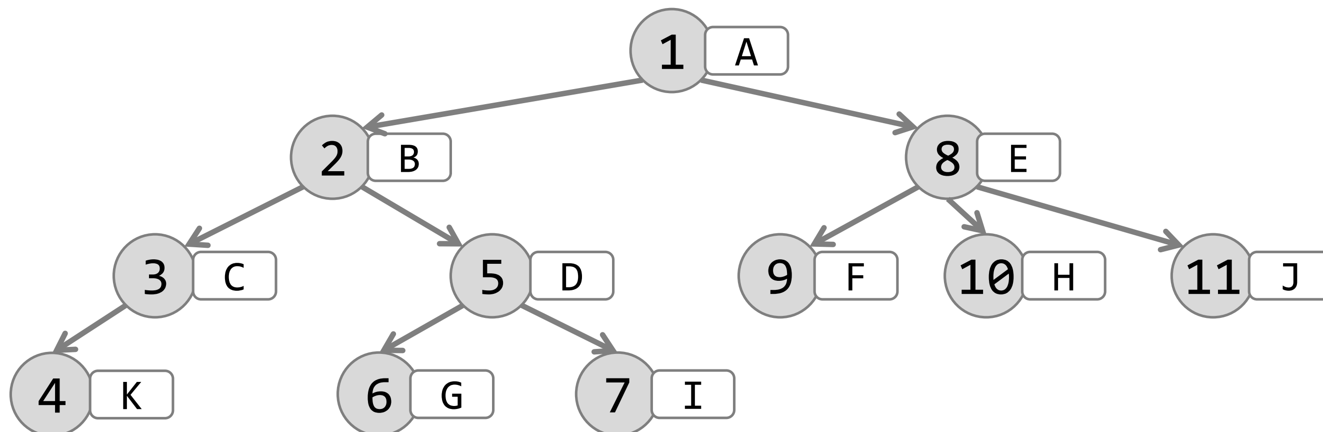
帰りがけ順（後置順，後順，postorder）



さっきの「訪れた順」（A B C K D G I E F H J）は行きがけ順

行きがけ順（preorder）

各節点 v は，最初に訪問したときに処理される
内部節点は上から来たときに処理される
葉は訪問したときに処理される
結果として，木の節点は上から下向きに処理される



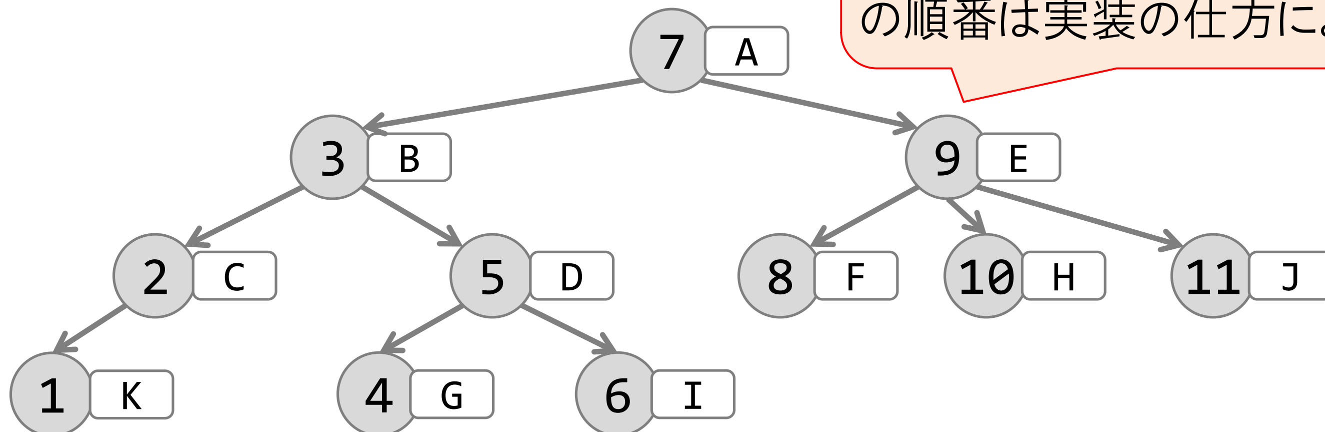
行きがけ順での節点の処理順： A B C K D G I E F H J

通りがけ順(inorder)

各頂点 v は，途中で訪問する時(左から右の子へ移る時)に処理される

結果として，木の節点は，左端から右端へ並んだ順で出力される

二分木でない場合(3つ以上の子がある場合)，間にある子と親の順番は実装の仕方による



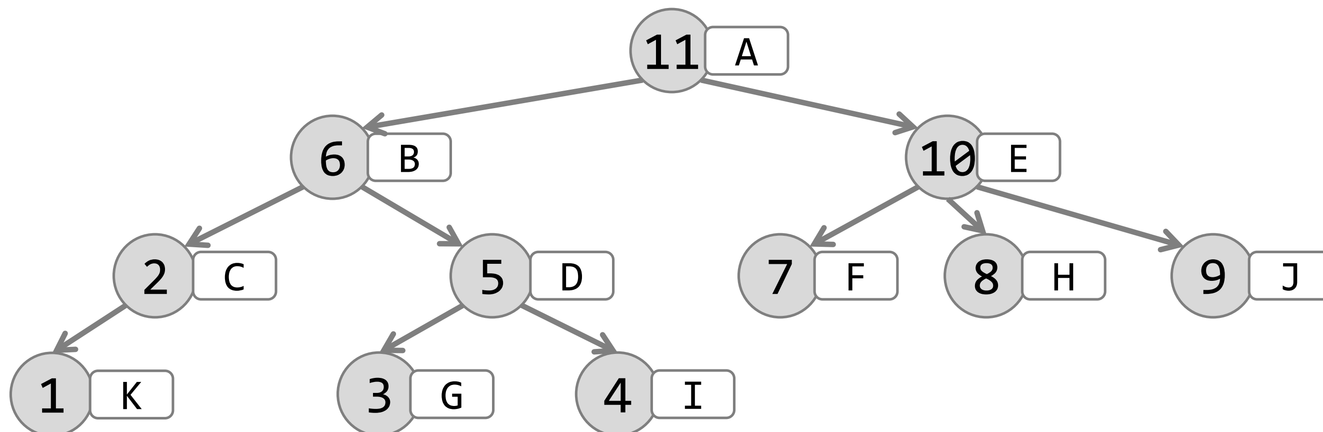
通りがけ順での節点の処理順： K C B G D I A F E H J

帰りがけ順（postorder）

各頂点 v は，最後に訪問する時（下から上へ戻る時）に処理される

結果として，木の節点は，下から上向きに処理される

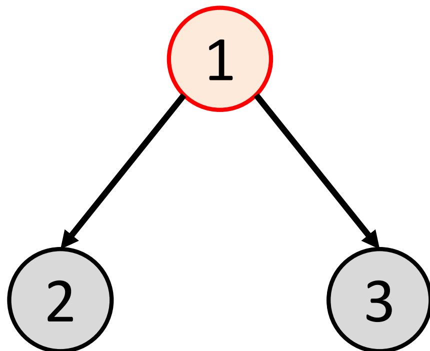
葉のデータを集約しながら全体のデータを得るときに使う



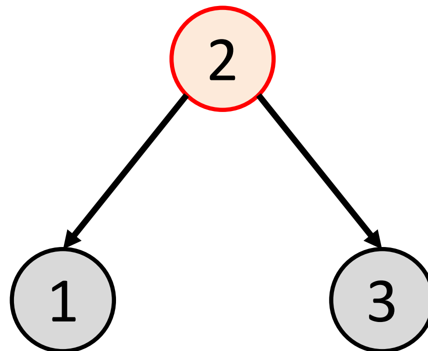
帰りがけ順での節点の処理順： K C G I D B F H J E A

先生，憶えるのが大変です・・・

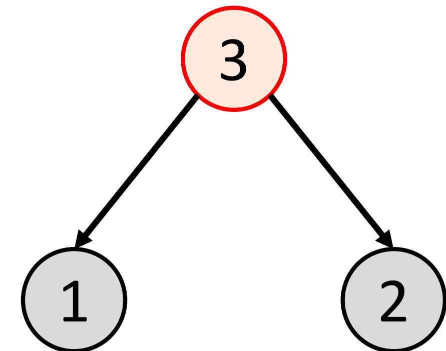
親と二つの子からなる木を考えたとき，
親の順番に注目すると憶えやすいです



行きがけ順
（preorder）



通りがけ順
（inorder）



帰りがけ順
（postorder）

スタックによるDFSの実現

【DFSのアルゴリズム（行きがけ順）】

空のスタック S を用意する

スタック S に木 T の根を登録（push）する

スタック S が空でない間，以下の処理を繰り返す

1．スタック S から要素（節点）v を取り出す（pop）
2．節点 v を処理（出力）する
3．節点 v の右の子があればスタック S に登録する
4．節点 v の左の子があればスタック S に登録する

スタックは，最後に入れたものが先に取り出されることを思い出そう！

通りがけ順，帰りがけ順にするにはどうすればいいでしょうか？

再帰処理を用いたDFS(行きがけ順)

```
void preorder(node *v) {
    if (v == null) return;   //再帰の終端条件
    vを出力する;
    preorder(v->left);   //左の子を処理
    preorder(v->right); //右の子を処理
}

void main() {
    node *root = Tの根へのポインタ;
    preorder(root);
}
```

根からスタート
頂点vに来たら，まず自分自身を出力し，
次に左と右の部分木を順に処理する

再帰処理を用いたDFS（通りがけ順）

```
void preorder(node *v) {
    if (v == null) return;   //再帰の終端条件
    preorder(v->left);   //左の子を処理
    vを出力する;
    preorder(v->right); //右の子を処理
}

void main() {
    node *root = Tの根へのポインタ;
    preorder(root);
}
```

根からスタート
頂点vに来たら，まず左の部分木を処理し，次に自分自身を出力してから，右の部分木を処理する

再帰処理を用いたDFS(帰りがけ順)

```
void preorder(node *v) {
    if (v == null) return;   //再帰の終端条件
    preorder(v->left);   //左の子を処理
    preorder(v->right); //右の子を処理
    vを出力する;
}

void main() {
    node *root = Tの根へのポインタ;
    preorder(root);
}
```

根からスタート
頂点vに来たら，まず左と右の部分木を処理し，最後に自分自身を出力する

これら三つのアルゴリズムを見て何か気づきませんか？

まとめ

木を探索するアルゴリズム（木の巡回）

幅優先探索（breadth-first-search; BFS）

⇒ キューを使って実現できる

深さ優先探索（depth-first-search; DFS）

⇒ スタックを使って実現できる

⇒ 再帰呼び出しを使うと簡単に記述できる！

行きがけ順（preorder traversal；前順）

通りがけ順（inorder traversal；中順）

帰りがけ順（postorder traversal;後順）

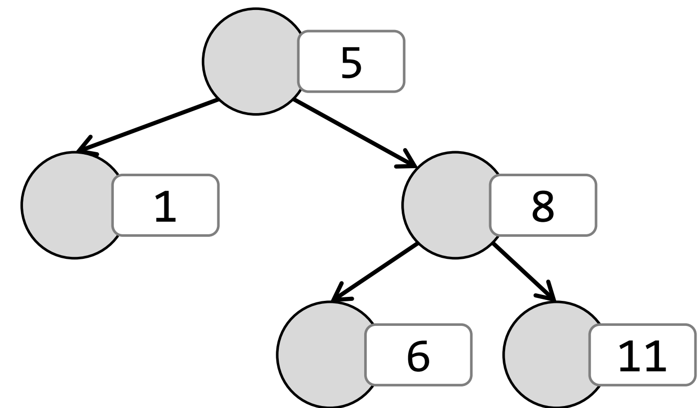
おまけ：二分探索木の実装例

bst.h

```
typedef struct BinarySearchTreeNode {
  int data;
  struct BinarySearchTreeNode *left;
  struct BinarySearchTreeNode *right;
} bst_node;

bst_node *insert(bst_node *root, int data) {
  if(root==NULL) {
    root = (bst_node *)malloc(sizeof(bst_node));
    if(root==NULL) {
      printf("memory allocation error¥n");
      exit(EXIT_FAILURE);
    } else {
      root->data = data;
      root->left = root->right = NULL;
    }
  } else {
    if(data < root->data)
      root->left  = insert(root->left, data);
    else if (data > root->data)
      root->right = insert(root->right, data);
  }
  return root;
}
```

二分探索木は、データの集合に対して、効率よい検索を提供するデータ構造

各ノードは高々2個の子供を持ち、自身が保持する値は、左の子孫より大きく、右の子孫より小さい



EXIT_FAILUREは<stdlib.h>で定義されているexit関数の戻り値

おまけ：二分探索木の実装例(つづき)

bst.h(つづき)

```
bst_node *find(bst_node *root, int data)
{
  if(root==NULL) return NULL;
  else if(data < root->data)
    return find(root->left, data);
  else if(data > root->data)
    return find(root->right, data);
  else // case that root->data == data
    return root;
}

bst_node *find_min(bst_node *root)
{
  if(root==NULL) return NULL;
  else if(root->left==NULL) return root;
  else return find_min(root->left);
}

bst_node *find_max(bst_node *root)
{
  if(root==NULL) return NULL;
  else if(root->right==NULL) return root;
  else return find_max(root->right);
}
```

```
void inorder_search(bst_node *root)
{
  if(root) {
    inorder_search(root->left);
    printf("%d¥n", root->data);
    inorder_search(root->right);
  }
}
```

inorder_searchは、木を深さ優先探索しながら中間順(通りがけ順)でノードを処理する
ここでは、単にノードの値を出力しているが、木に格納した整数のソートを行っているのと同じ結果になる

一番左にあるノードの値が最小値

一番右にあるノードの値が最大値

おまけ：二分探索木の使用例

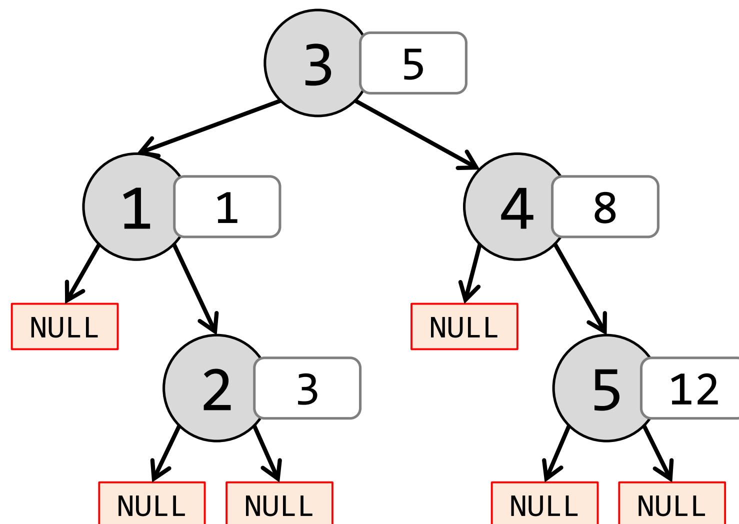
test_bst.c

```
#include <stdio.h>
#include <stdlib.h>
#include "bst.h"

int main(void)
{
  int d = 3;
  bst_node *tmp=NULL;
  bst_node *root=NULL;

  root=insert(root,5);
  root=insert(root,8);
  root=insert(root,1);
  root=insert(root,12);
  root=insert(root,3);
  printf("min=%d¥n", find_min(root)->data);
  printf("max=%d¥n", find_max(root)->data);
  if((tmp=find(root, d)) != NULL)
    printf("%d was found¥n", tmp->data);
  else
    printf("%d was not found¥n", d);
  inorder_search(root);
  return 0;
}
```

insertの戻り値を必ずrootにセットする

実行結果

```
$ ./test_bst
min=1
max=12
3 was found
1
3
5
8
12
```



プログラミング問題（迷路の最短路）

　俺は現在，相棒と共にとあるダンジョン（迷宮）の奥深くに居る．ダンジョンの最奥に隠されていた宝箱を開けた際，運悪く毒針の罠にかかってしまった．解毒するには，地上で待っている仲間のもとまでたどり着く必要があった．

「・・・アスナ・・・すまない・・・」
「しっかりして！大丈夫よ．私このフロアの地図を持っているわ．最短の経路で地上へ出るから！」

　ダンジョンは大きさが$N \times M$（$N, M < 100$）マスの迷路になっている．迷路は通路（.）と壁（#）でできており，現在位置からは隣接する上下左右の4方向の通路へと移動できる．
　便宜的に，1マスの移動に1分かかるとする．また，毒が回りきる時間T（分）が与えられる．スタート(S)からゴール(G)にたどり着くまでの時間がT分を超える場合は”dead”と表示し，そうでなければゴールまでの最短時間を表示するプログラムを書け．

問題の入力例

例1（$N = 17, M = 11, T = 60$）

<table>
<tr><td>#</td><td>#</td><td>S</td><td>#</td><td>#</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td></tr>
<tr><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>#</td><td>#</td><td>.</td></tr>
<tr><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td><td>.</td><td>#</td></tr>
<tr><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>.</td><td>#</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td></tr>
<tr><td>#</td><td>#</td><td>#</td><td>.</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td></tr>
<tr><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>#</td><td>#</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td><td>.</td><td>.</td><td>.</td><td>#</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td></tr>
<tr><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td></tr>
<tr><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>#</td></tr>
<tr><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>.</td><td>G</td><td>#</td></tr>
</table>

例2（$N = 17, M = 14, T = 60$）

<table>
<tr><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>#</td><td>#</td><td>#</td><td>.</td><td>#</td><td>#</td><td>.</td><td>#</td><td>#</td><td>#</td><td>.</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td></tr>
<tr><td>#</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td></tr>
<tr><td>.</td><td>.</td><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td></tr>
<tr><td>.</td><td>#</td><td>#</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td></tr>
<tr><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>S</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>#</td><td>#</td></tr>
<tr><td>#</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td><td>G</td></tr>
<tr><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td><td>#</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>#</td><td>.</td><td>.</td><td>#</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td></tr>
<tr><td>.</td><td>#</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>.</td></tr>
<tr><td>.</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>#</td><td>.</td></tr>
<tr><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td></tr>
</table>

プログラミング問題（シミの数を数える）

　特に女性にとって，顔にできたシミは非常に疎ましい存在である．お肌のシミが気になり始めた梅子のもとに，とあるセールスマンがやってきた．

「できたシミを，レーザーで治療できるのをご存知ですか？」

「えっ！本当に？」

「ええ．いまならシミの大きさに関わらず，一つあたり1，000円で治療がうけられます．いまだけの特別ご奉仕価格です」

　――これで鞠絵のような綺麗な頬になれるかも．でもいったい幾らかかるのかしら――

　梅子は美貌の友人を思い浮かべながら，さっそく手鏡に映った頬のシミを数え始めたのであった．

　便宜的に，手鏡の大きさは$N \times M$（$N, M \leq 50$）で，シミ（S）は8近傍で隣接しているとつながっているとします．この時，シミの数を数えるプログラムを書きなさい．

Sの8近傍とは
下の＊の部分

＊	＊	＊
＊	S	＊
＊	＊	＊

問題の入力例

例1（$N = 8, M = 8$）

S	S	S	.	.	.	.	.
.	S	.	.	.	S	S	S
.	.	.	.	S	S	S	.
.	.	.	.	.	.	.	.
.	S	.	.	.	.	.	.
S	S	S	.	.	S	.	S
S	S	.	.	.	.	S	.
.	S	.	.	.	S	.	S

Sはシミを表し，ピリオド「．」はきれいな肌を表す

例1の場合，シミの数は4

例2（$N = 15, M = 12$）

.	.	.	.	.	.	.	.	S	.	.	.
S	S	S	S	S	.	S	S	S	S	S	.
.	.	.	.	S	.	.	.	S	.	S	.
.	.	S	S	.	.	.	.	S	.	S	.
.	.	S	.	.	.	.	S	.	.	S	.
.	.	.	.	.	.	.	S	.	.	.	.
S	.	S	.	S	.	S	S	S	S	S	.
S	.	S	.	S	.	.	.	S	.	.	.
.	.	.	.	S	.	S	S	S	S	S	.
.	.	.	S	.	.	.	.	.	S	.	.
.	.	.	.	.	.	.	.	.	.	.	.
S	.	.	S	.	.	S	.	S	S	S	S
S	.	S	.	S	.	S	.	S	S	S	S
S	.	S	.	S	.	S	.	S	S	.	.
S	S	.	S	.	.	.	S	.	S	S	S