アルゴリズムとデータ構造

- 第6回講義トピック：木
  - 木(tree)の定義
  - 木の表現
  - 2分木
  - 解析木
  - 木の走査

Algorithms and data structures

- Topic of lecture no. 6: tree
  - definition of tree
  - representations of tree
  - binary tree
  - parse tree
  - traverse of tree

木 (tree)

- １次元のデータ構造：
  - 連結リスト、スタック、キュー
  - 特徴：項目が順番に１列に並んでいる
- ２次元のリンクによるデータ構造：
  - 木：１つのノードが２つ以上の要素へのリンクを持つ
- 木の例：
  - 家系図
  - スポーツのトーナメント
  - 会社の組織図
  - UNIX階層ファイルシステム
  - …

What is a tree?

- One-dimensional data structures:
  e.g., array, linked list, stack, queue
- Two-dimensional data structures:
  a node of a tree has more than two links
- Some examples of tree data structure
  - family trees
  - tournament matches
  - organization of companies
  - computer file systems
  - …

木に関連する基本用語(1)

- **節点(node)**:木や連結リストにおけるデータを格納する要素である。グラフのイメージから頂点(vertex)とも呼ばれる。各節点はデータ以外に他の節点への連結情報が含まれる。
- 辺(edge):節点から節点へのリンクを指す。
- **パス/道 (path)**:辺で結ばれている節点の並びである。
- **ルート/根 (root)**:特別に指定された1つの節点で、木のトップに当たる(木の根っこみたいで、そこから節点が広がる)。
- **木 (tree)**:ルートを含む節点(node)と辺(edge)の集合からなるもので、以下の性質を持つ。
  - 任意の節点はルートと唯一のパスで結ばれる。
    - パスが唯一でない場合はグラフ (graph) と呼ばれる。
  - どことも結ばれていない節点がない。

Terminology (1)

- node: an element of a tree with data part and link part.
- edge: a link connecting two different
- path: a sequence of nodes connected with edges
- root: the top node of a tree
- tree: a collection of nodes and edges with the following properties
  - Every node is connected with the root by only a single unique path.
    - Otherwise is called a graph.
  - There is no node without connected with any others.

木に関連する基本用語(2)

- **親** (parent)／**子** (child): 自分のすぐ上にある節を親、逆にすぐ下にある節を子と呼ぶ。
- **兄弟**(sibling) : 同じ親を持つ節を兄弟と呼ぶ。
- **葉** (leaf): 子のない節点を葉という。終端点、**外部節点**(external node)とも言う。
- 子のある節点は非終端点あるいは**内部節点** (internal node)という。
- **節点のレベル** (level): 木のルートからの節点までの道にある節点の数のこと
- **高さ** (height): ルートからもっとも遠い節点までの距離
- **道長** (path length) : 木の節点のレベルの総和
  ＝ルートから全ての節点へのパスの長さをすべて加えたもの
- **部分木** (subtree): ある任意の節以下の節からなる節の集合を木全体に対して部分木と呼ぶ。この場合、一番上にある節を部分木のルートと呼ぶ。どの節点も自身より下にある節点からなる部分木のルートになる。
- **森** (forest): 独立の木、あるいは部分木からなる集合を森と呼ぶ。

Terminology (2)

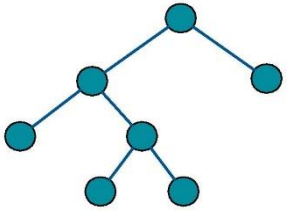
- parent/child: a node is a parent when it is connected with other nodes (children) in the path away from the root.
- siblings: nodes with the a parent
- leaf (external node): a node with no children
- internal node (non-terminal node): a node with more than one children
- level (depth) of node: the number of edges in the path from the root to the node
- height of node: the number of edges of the longest path from the node to a leaf
- height of tree: the height of the root node
- path length of tree: the sum of all the distances between nodes and the root
- subtree: every node together with its children and descendants is a subtree. The node is then the root of the subtree.
- forest: a set of disjoint trees

2分木の定義

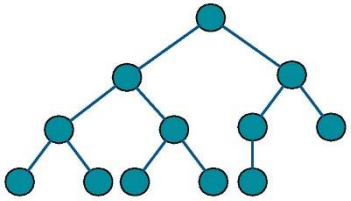
- **順序木 (ordered tree)**: 節点のすべての子に順序関係がある場合、その木を順序木と呼ぶ。節点の子に順序関係がない場合は木は複数通りに描くことができる。
- **2分木 (binary tree)**: 順序木の中で、各節点が持てる子の最大数が2の時、2分木と呼ぶ(2以上の場合は多分木という)。この時、左にある子を左の子、右にある子を右の子という。
  ＊片っ方の子しか持たない場合も左か右かを区別する。この場合、もう片っ方の子はないのではなく、空であると考えるべきである。
- **全2分木 (full binary tree)**: 終端以外のどの節点も2つの子をもつ2分木を全2分木と呼ぶ。全2分木は外部節点に情報を持たせないで単にダミーの節点とみなすことがよくある。どの2分木もダミーの外部節点を加えることで全2分木にすることができる。
- **完全2分木のもう1つの定義(complete binary tree)**: 2分木で、全ての外部節点が最大レベルの内部節点と同じか高いレベルにあり、その最大レベルの内部節点が全て外部節点よりも左にある時、完全2分木という。(ヒープツリー)
  ＊この定義では完全2分木は全2分木である必要がない。
- **完全2分木 (perfect binary tree)**: 最大レベルを除いて、どのレベルも内部節点で完全に詰まっている状態の全2分木をいう。

Binary tree

- ordered tree: the children of each node is ordered
- binary tree: an ordered tree, each node has at most two children (left child and right child)
- full binary tree (proper binary tree, strictly binary tree): a binary tree with all non-terminal nodes have two children. Any tree can form full binary tree by adding dummy terminal nodes.



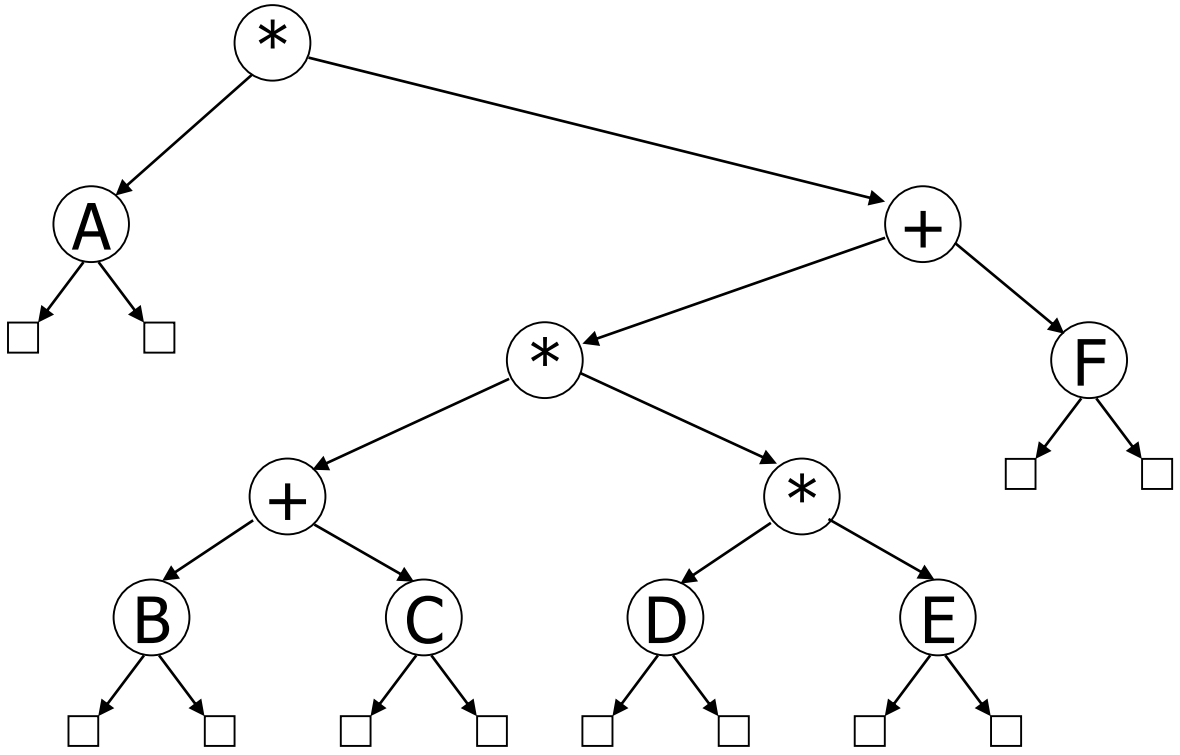
- complete binary tree (heap tree): a binary tree, all of the terminal nodes are same or higher than any non-terminal nodes, and the highest non-terminal nodes are in the left than terminal nodes.
  A complete binary tree no need to be a full binary tree.



- perfect binary tree: a full binary tree with all leaf nodes at the same depth.

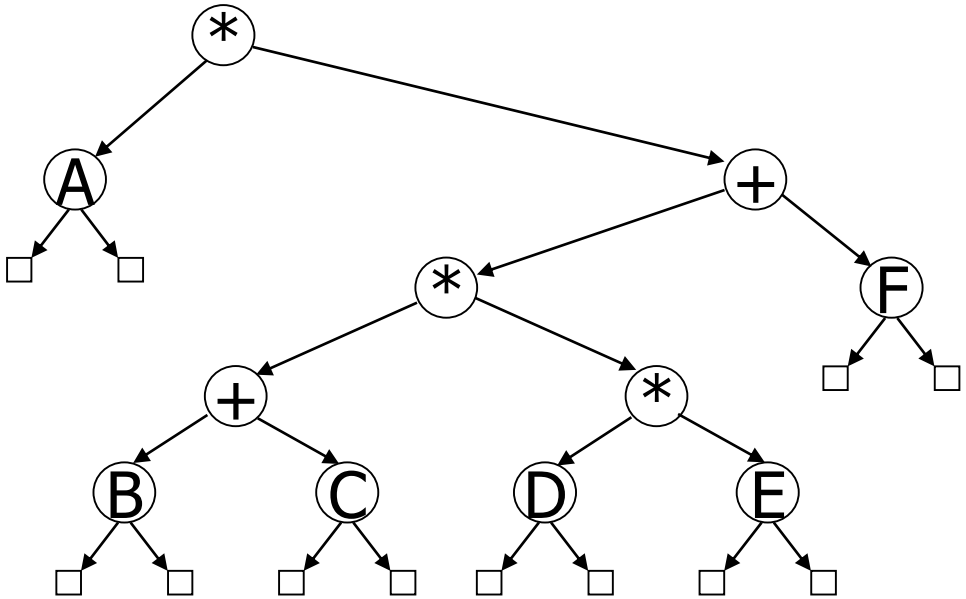
2分木の例

以下に2分木の例を示す（算術式の解析木）。この例では、○は内部節点、□は情報を持たない外部節点（外部節点はダミーであるため、実装において1つのノードzとしてまとめることができる）。



算術式の解析木 (parse tree)

- **算術式の解析木**：算術式の構文を表す。再帰的に、それぞれの部分木においてルートには演算子をおき、最初の被演算子である式を左側の木とし、２番目の被演算子である式を右側の木とする。
  この例は、式A*(((B+C)*(D*E))+F) （逆ポーランド記法で ABC+DE**F+* ）の２分木による表現を示す。



解析木の生成

- 後置記法の式から2分木を生成する

```
struct node {
  char info;
  struct node *l, *r;
};
struct node *x, *z;
char c;
for (stackinit(); scanf(“%1s”, &c)!=EOF; ){
  x=(struct node *)malloc(sizeof *x);
  x->info = c; x->l = z; x->r = z;
  if(c==‘+’ || c== ‘*’){
    x->r = pop();
    x->l = pop();
  }
  push(x)
}
```

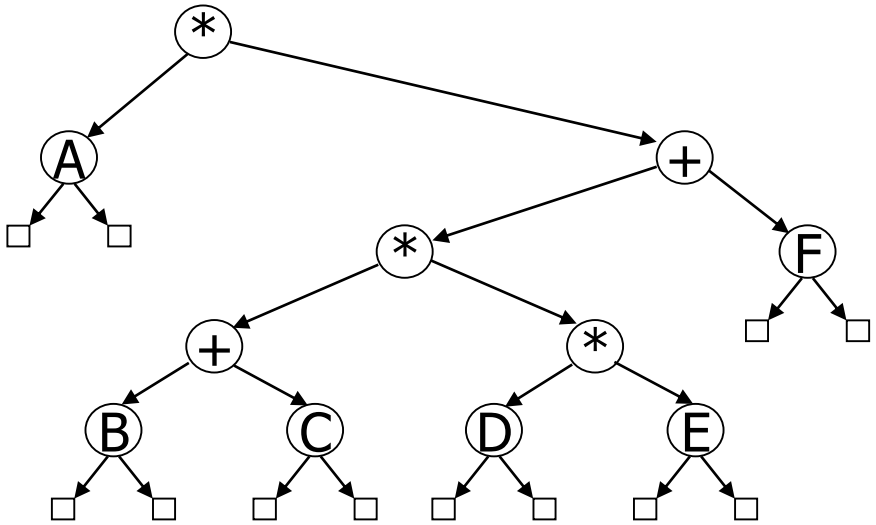
木の走査 (traverse)

- **走査**：決められたルールに従って、すべての節点を1回だけ訪れることを指す。
- 木では、以下の走査の仕方が考えられる。
  - 先行順 (preorder)：子よりも先にルートを訪れる。
  - 中央順 (inorder)：左の子、ルート、右の子の順に訪れる。
  - 後行順 (postorder)：左の子、右の子、ルートの順に訪れる。
  - レベル順 (level-order)：レベルの低い節点から順番に訪れる。

先行順走査

- 再帰による定義：
  ルートを訪れ、次に左の部分木を先行順に走査し、そして右の部分木を先行順に走査する。

```
traverse(struct node *t){
  if (t != z) {
    visit(t);
    traverse(t->l);
    traverse(t->r);
  }
}
```

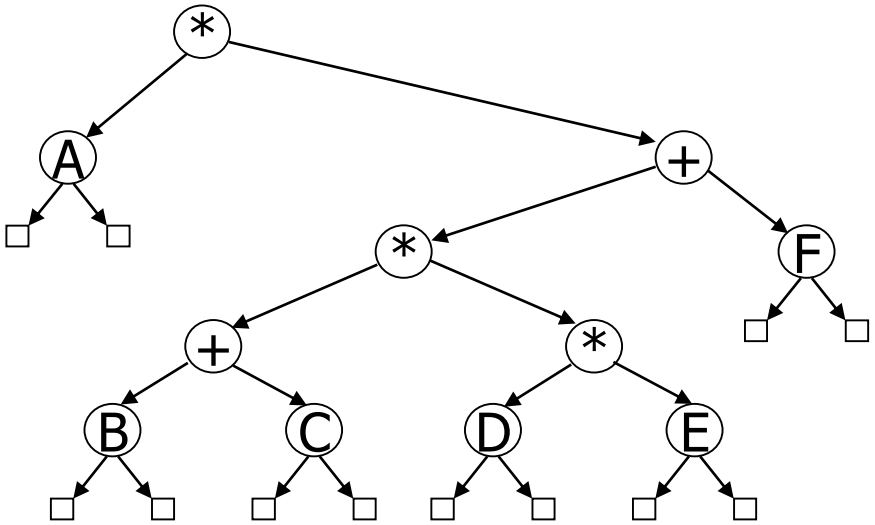


訪れる順番：*A+*+BC*DEF

中央順(対称順)走査

- 再帰による定義:
  左の部分木を中央順に走査し、次にルートを訪れ、そして右の部分木を中央順に走査する。

```
traverse(struct node *t){
  if (t != z) {
    traverse(t->l);
    visit(t);
    traverse(t->r);
  }
}
```

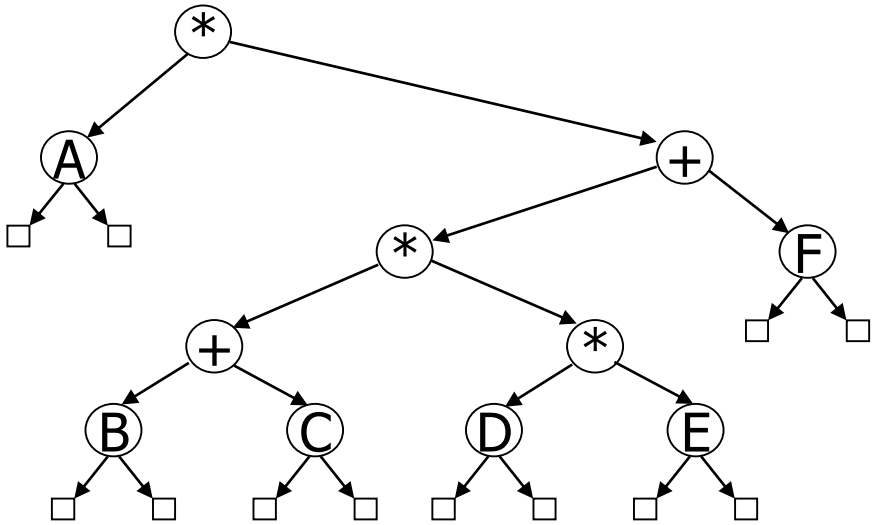


訪れる順番:A*B+C*D*E+F

後行順走査

- 再帰による定義：
  左の部分木を後行順に走査し、次に右の部分木を後行順に走査し、そしてルートを訪れる。

```
traverse(struct node *t){
  if (t != z) {
    traverse(t->l);
    traverse(t->r);
    visit(t); }
}
```

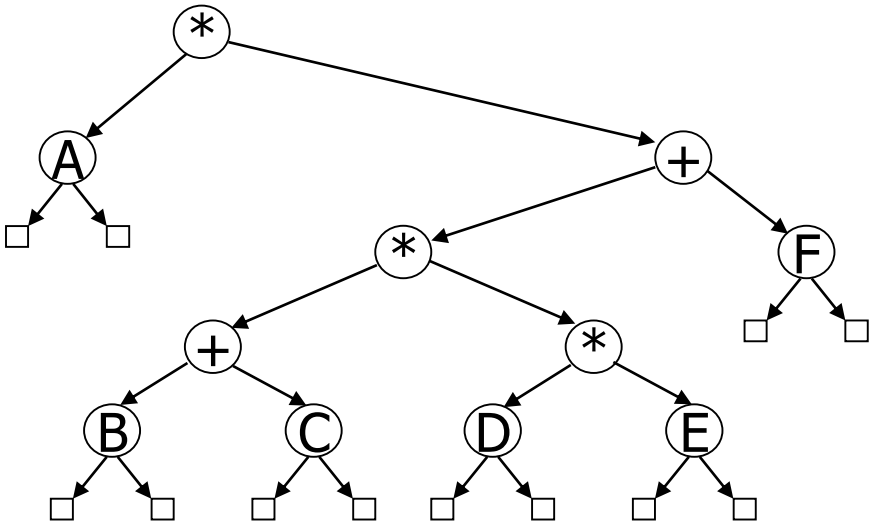


訪れる順番：ABC+DE**F+*

スタックによる先行順走査

- スタックによる先行順走査

```
traverse(struct node *t){
   push(t);
   while (!stackempty()){
      t = pop();
      visit(t);
      if (t->r != z)
        push(t->r);
      if (t->l != z)
        push(t->l);
    }
}
```



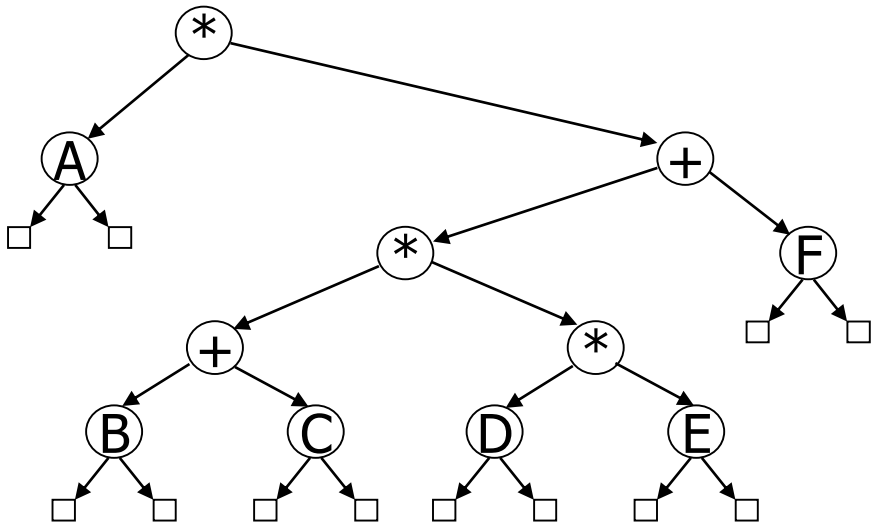
訪れる順番：*A+*+BC*DEF

キューによるレベル順走査

- 上から下へ、左から右へ、レベル毎に節点を全部を訪れる。

```
traverse(struct node *t){
   enqueue(t);
   while(!queueempty()){
      t = dequeue();
      visit(t);
      if (t->l != z)
        enqueue(t->l);
      if (t->r != z)
        enqueue(t->r);
     }
   }
```

＊レベル順は再帰的に表せない



訪れる順番：*A+*F+*BCDE