追補（演習3）：順序木の列挙（木マイニング）

有村博紀
北海道大学大学院情報科学研究科

演習3: 順序木の列挙（木マイニング）

順序木に関する次の問題に答えよ．．

1. サイズがn = 4以下の順序木を全て書き出せ．それらの総数は何個か？
2. オプション問題）正整数 n を入力としてもらい，サイズがn以下の順序木をすべて書きだすプログラムを書け．プログラミング言語は何を使ってもよい．日本語や英語の疑似コードでもかまわない．

演習: 集合の列挙（頻出集合マイニング）

Let S = {1, 2, 3, 4} be a finite set of four elements.

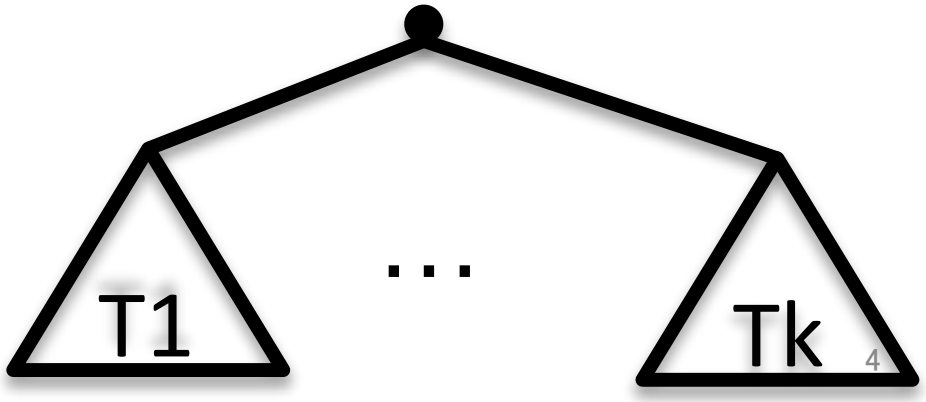
1. Write all ordered trees of size up to n = 4.
2. How many are they?
3. Optional Problem) Write a computer program (algorithm) that receives a positive integer *n*, and prints all ordered trees of size *n*. You can use any programing languages as well as pseudo code written in Japanese/English.

順序木

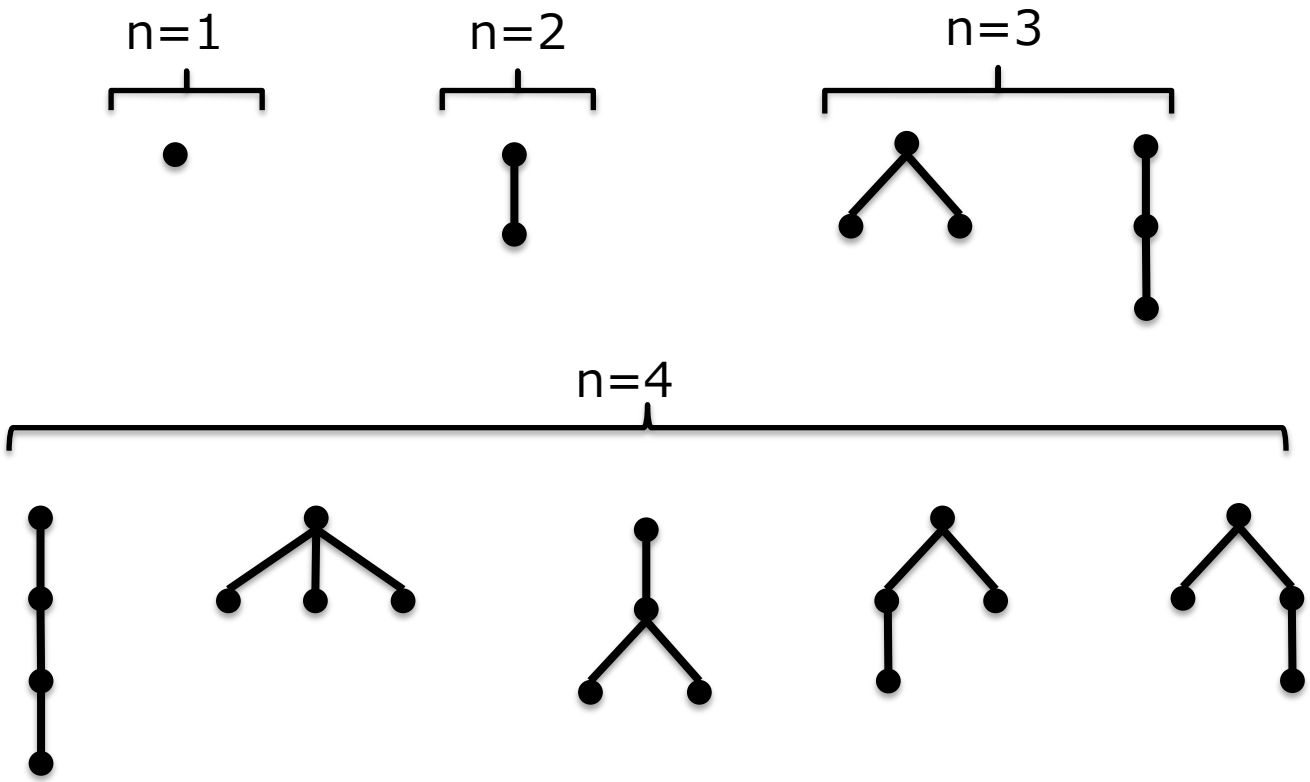
- 順序木は，子供の間に順序がある根付き木.
- サイズ1の順序木は一つの頂点からなる.
- サイズn≥2の順序木は，次のように作られる：
  サイズ $n_1, \cdots, n_k \geq 1$のk個の順序木があったら，
  新しい頂点の下にそれらを並べると，サイズ $n = 1 + n_1 + \cdots + n_k$ の順序木が得られる.

サイズ1の順序木

サイズ$n = 1 + n_1 + \ldots + n_k$の順序木 T



問題1の回答
サイズ n = 4までのすべての順序木を示す.
総数は9個.



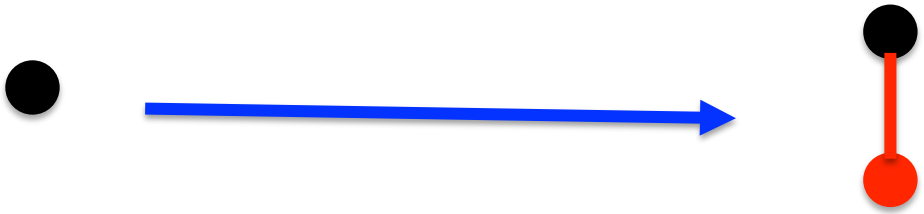
順序木の列挙(生成)

- 深さ優先探索で，サイズnまでの順序木を列挙(もれなく，重複なく生成)することができる
- 方針
  - サイズ1の木からはじめて，新しい頂点を一つずつ加えていく
- 最右拡張法(rightmost expansion)
  - サイズn-1の現在の順序木を受け取る.
  - 最右枝上の頂点に，その最も右の子として，新しい頂点を追加して，サイズnの順序木を生成する.
  - 最右枝上:木の根から木の一番右下の頂点(＝前順巡回番号が最大)までの枝
  - 再帰呼び出しを使って，すべての可能な位置に上記の追加を系統的に試す.
  - それ以上追加できなくなったら探索の親へ戻る.

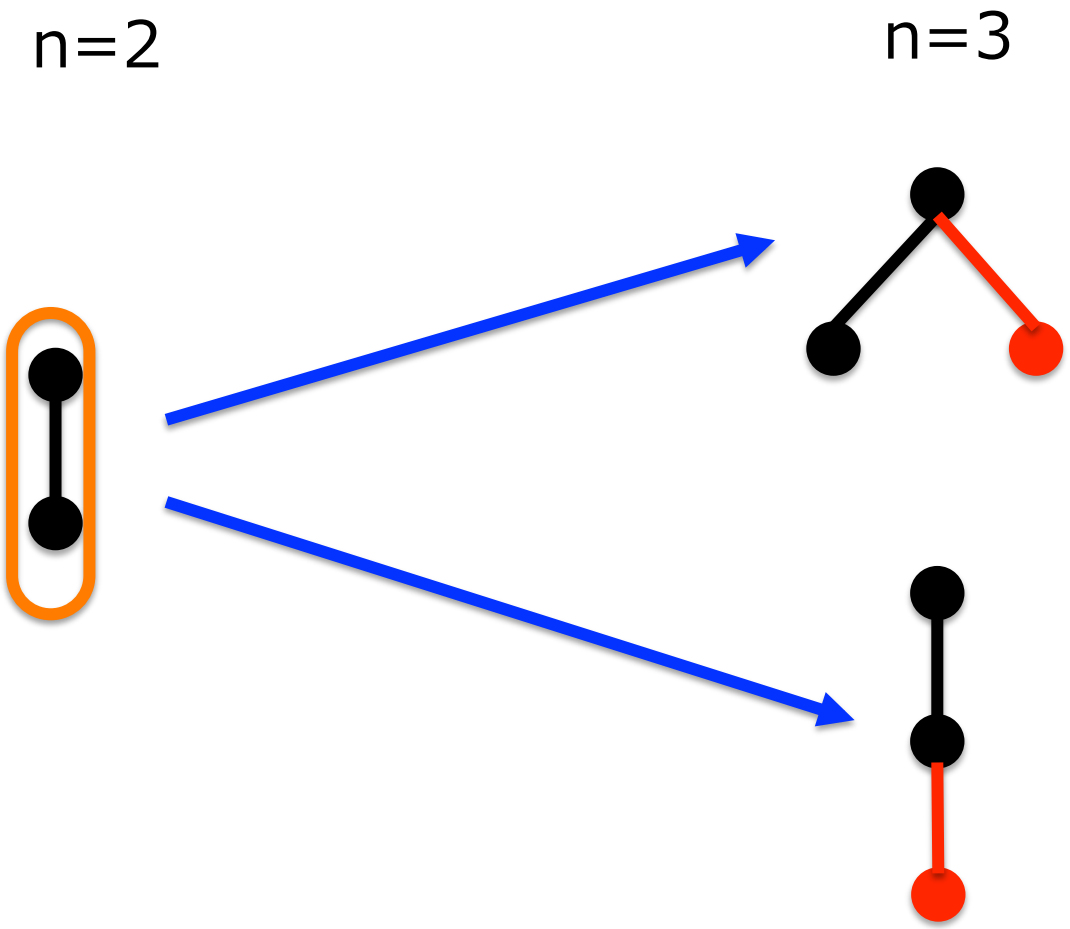
例：順序木の生成

n=1

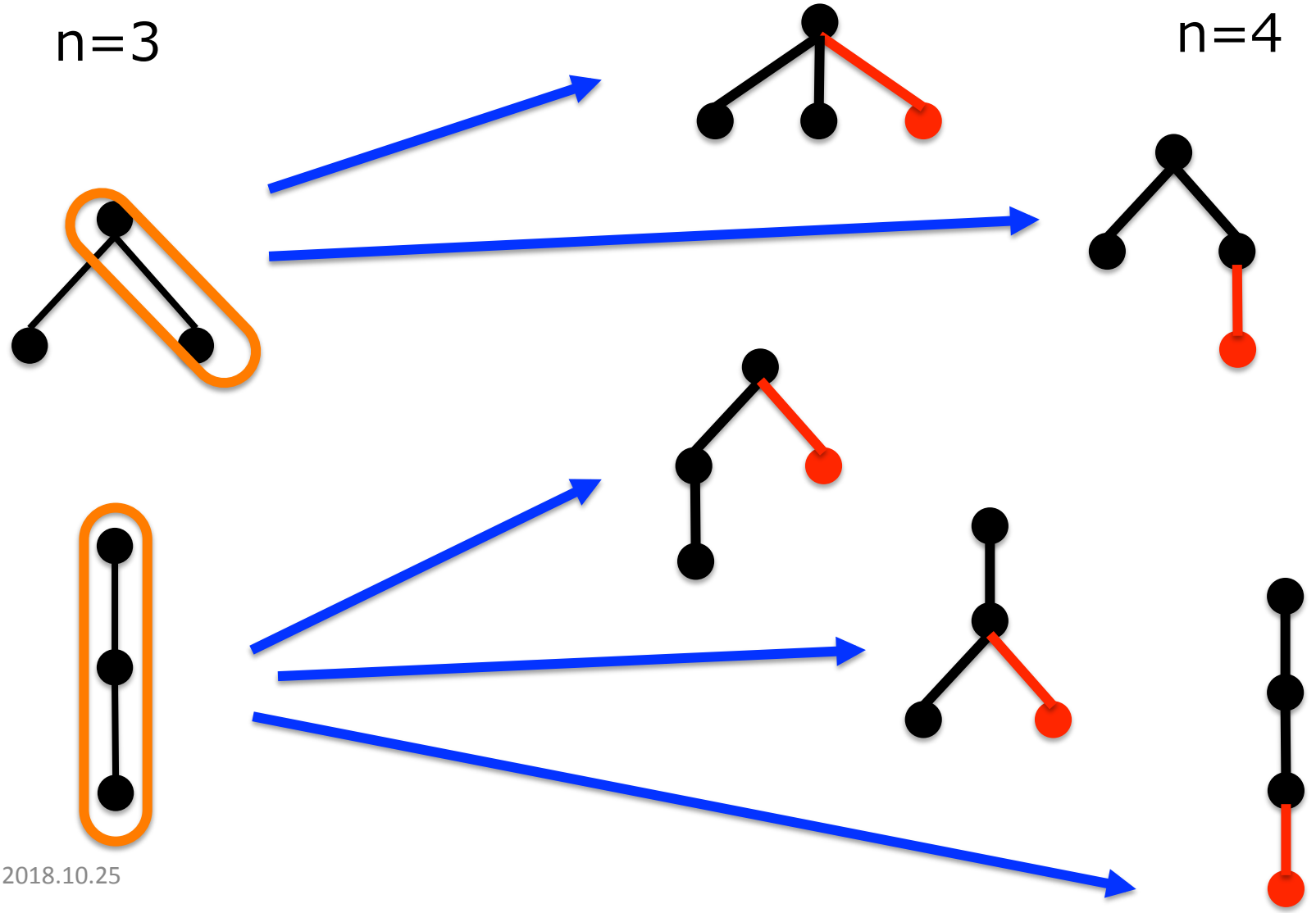
n=2



例：順序木の生成

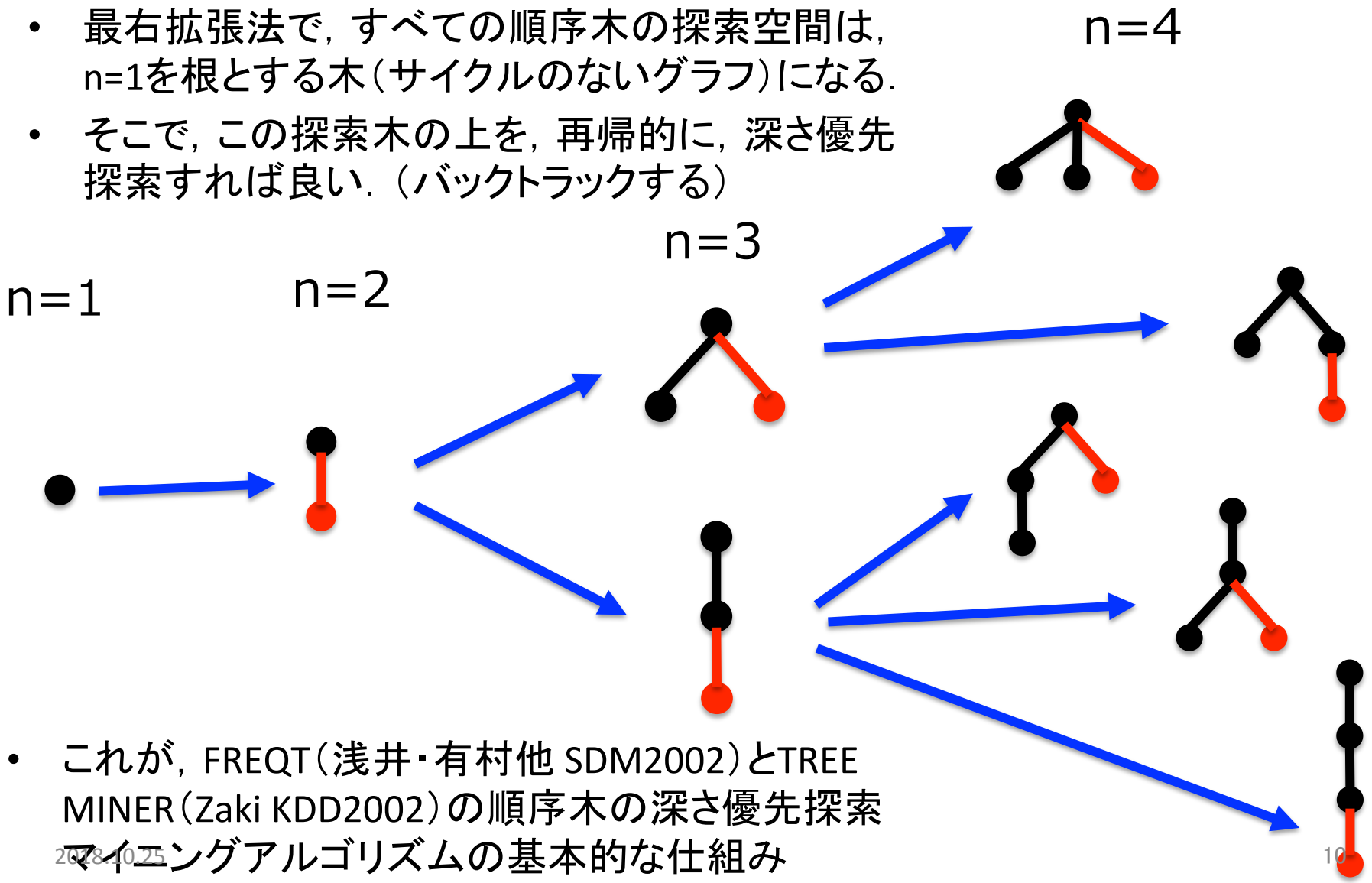


例:順序木の生成



順序木の列挙アルゴリズム

- 最右拡張法で，すべての順序木の探索空間は，n=1を根とする木（サイクルのないグラフ）になる．
- そこで，この探索木の上を，再帰的に，深さ優先探索すれば良い．（バックトラックする）



- これが，FREQT（浅井・有村他 SDM2002）とTREE MINER（Zaki KDD2002）の順序木の深さ優先探索マイニングアルゴリズムの基本的な仕組み

問題2の回答．入力として正整数nを受け取り，サイズn以下の全ての順序木を出力するプログラム（ruby言語）．

```
def enum(t, k, n)
  ##output a tree t
   print "size: #{k}, tree: #{t.str()}¥n"
   if k >= n then
      return
   end
  x = t  ## root of t
   while (x != nil)
      if x.sons.empty? then
         next_x = nil
      else
            #現在頂点xの末っ子を得る
         next_x = x.sons[x.sons.size - 1]
      end
      x.sons.push(X.new())      #末っ子の追加
      enum(t, k+1, n)           #再帰呼び出し
      x.sons.pop()     #末っ子の追加を取りけす
    x = next_x
   end
end

enum(Node.new(), 1, n)  #メインの呼び出し
#EOF
```

#プログラムの後半

- 木tに頂点を追加して，サイズn以下の順序木をすべて出力する
- Nodeオブジェクトは，連番（id）と子供のリスト（sons）をもつ.
- 再帰手続き enum(t, k, n)は，木tと，そのサイズk，最大サイズnを受け取り，tに新しい頂点を追加して得られるサイズk以下の順序木を全て探索し，出力する.
- 追加では，木tの最右枝上のノードを選び，その末っ子として新しい頂点を追加する.
- サイズが1の木から開始する.

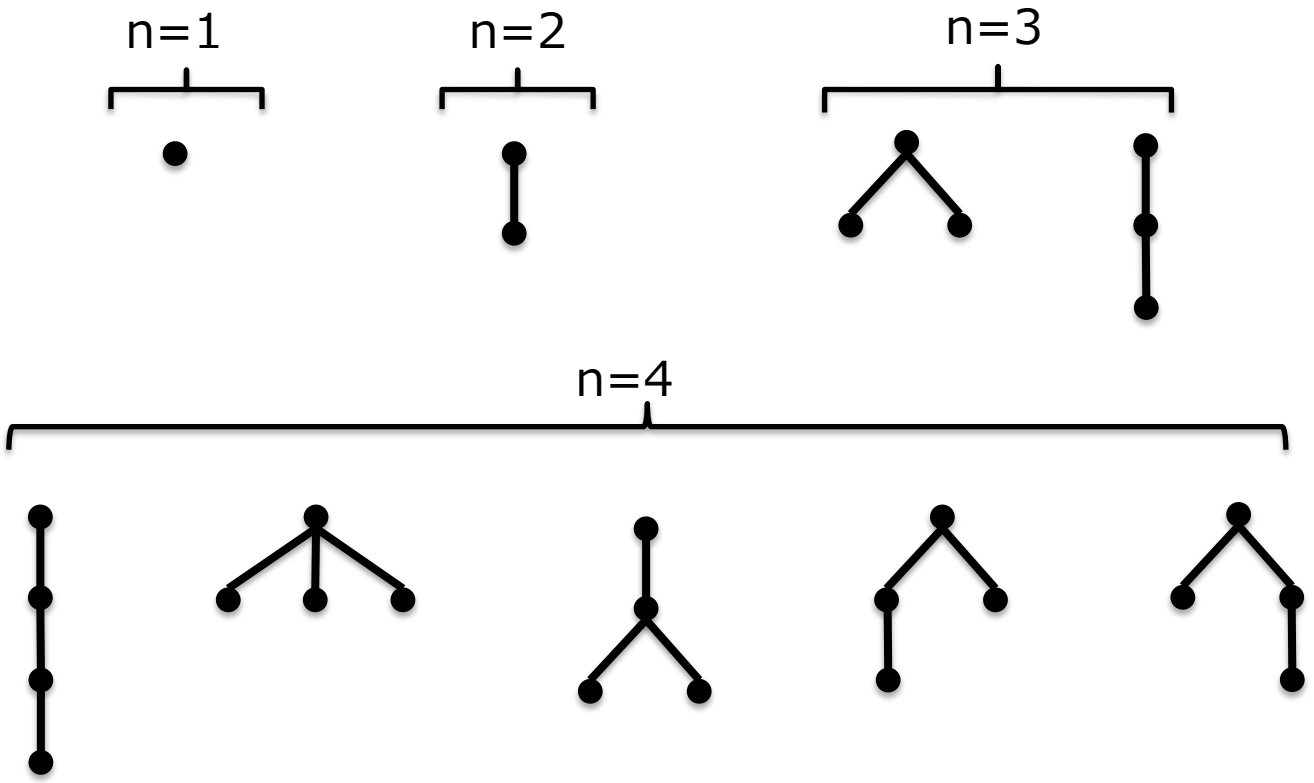
（3）プログラムの前半部分（ヘッダー部分）.

```
#coding: utf-8
if ARGV.size() < 1 then
   print "usage: #{$0} n¥n
  exit 1;
end
n = ARGV[0].to_i  ## 文字列を整数に変換
print "#{$0}: printing all ordered trees of size
#{n}...¥n"
class Node
  @@node_id = 0
   def initialize()
    @id = @@node_id
    @children = []
    @@node_id = @@node_id + 1
   end
   attr_accessor :id        ## getter/setter
   attr_accessor :children  ## getter/setter
   def str()
      自分を根とする木を印刷する.
   end
end
```

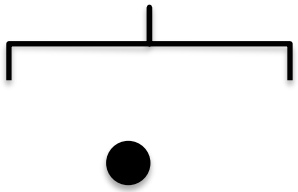
Ruby言語はやわかり (Quick tour)

- 言語一般
  - ARGV: コマンド引数の配列
  - def 名前(引数) ... end: 関数定義
  - if 条件 then ... else ... end: 条件文
  - while 条件 ... end: 繰り返し文
  - 文字列出力．#{x}: 変数xを文字列表現で展開する
  - 配列A. 空配列[], A[i] 第i番目の要素，A.push(x) 末尾へ要素追加, A.pop() 末尾要素の削除
- クラス関係
  - class 名前 ... end: クラス定義
  - クラス名.new: クラスのインスタンス（オブジェクト）生成
  - @@名前: クラス変数（クラス内のグローバル変数）
  - @名前: インスタンス変数（オブジェクトのメンバー変数）
  - attr_accessor :名前: インスタンス変数のセッター/ゲッター定義（例えばインスタンス変数@idに対して，"attr_accessor :id"としておくと，"A.id = x" や "x = A.id"と書けるようになる）

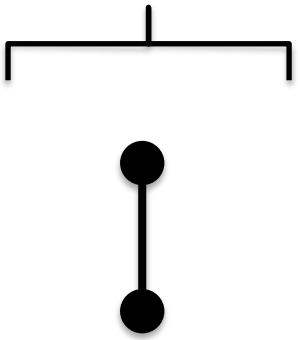
例：順序木の生成



n=1



n=2



n=3