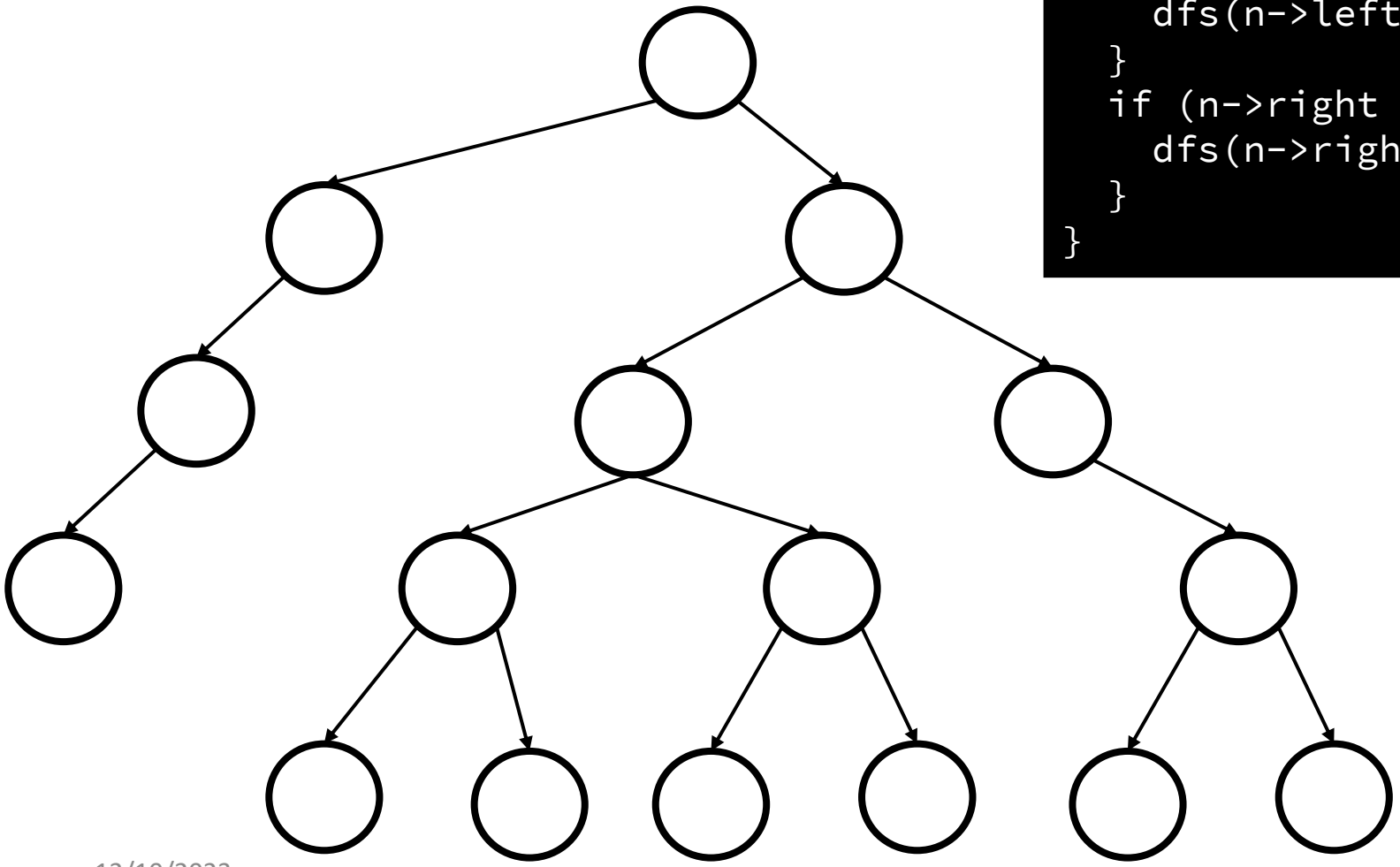
応用アルゴリズム演習

Depth First Search　/　深さ優先探索

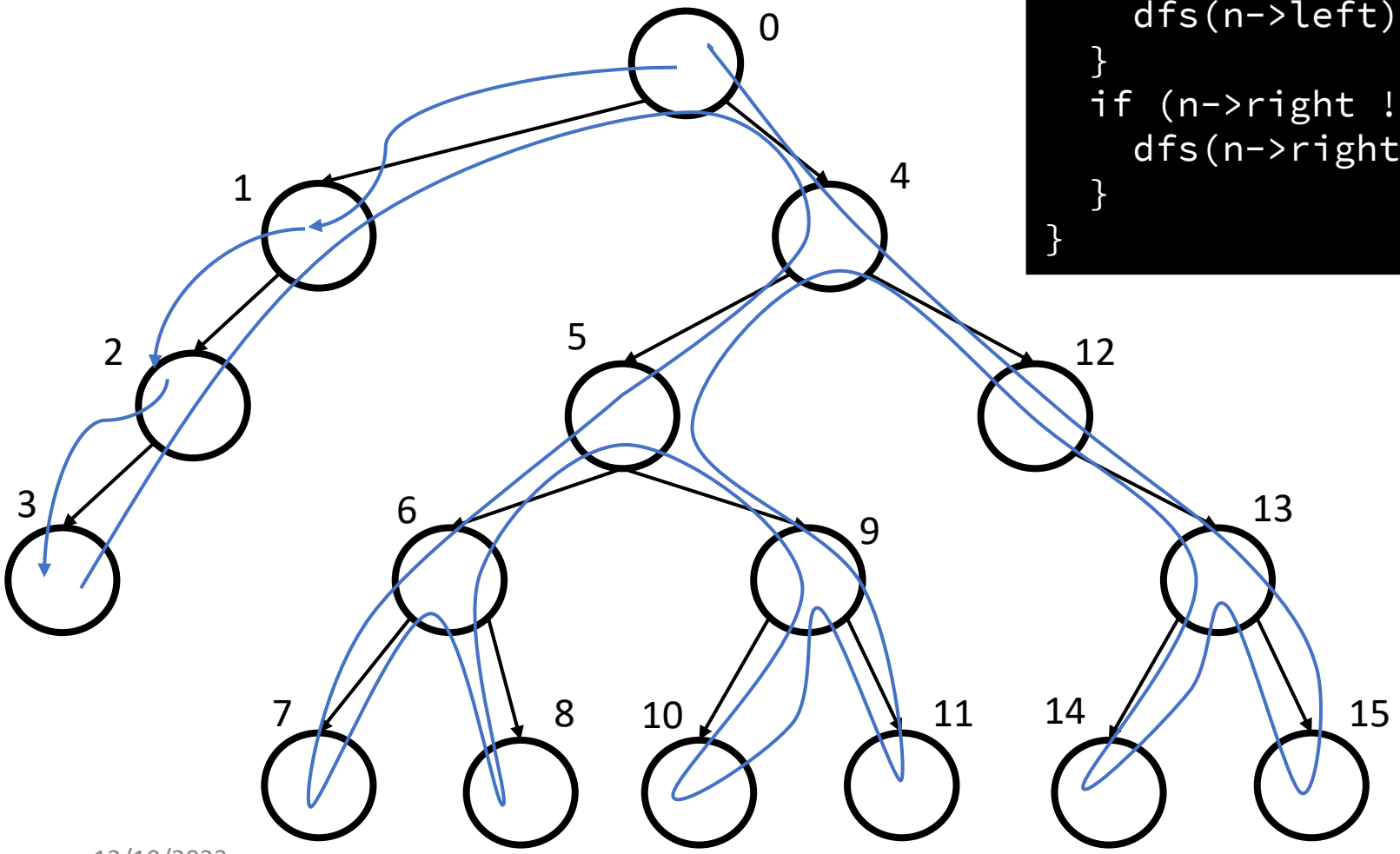
深さ優先探索
Depth First Search

```
typedef struct node {
  struct node * left;
  struct node * right;
} node_t;

void dfs(node_t * n) {
  if (n->left != NULL) {
    dfs(n->left);
  }
  if (n->right != NULL) {
    dfs(n->right);
  }
}
```

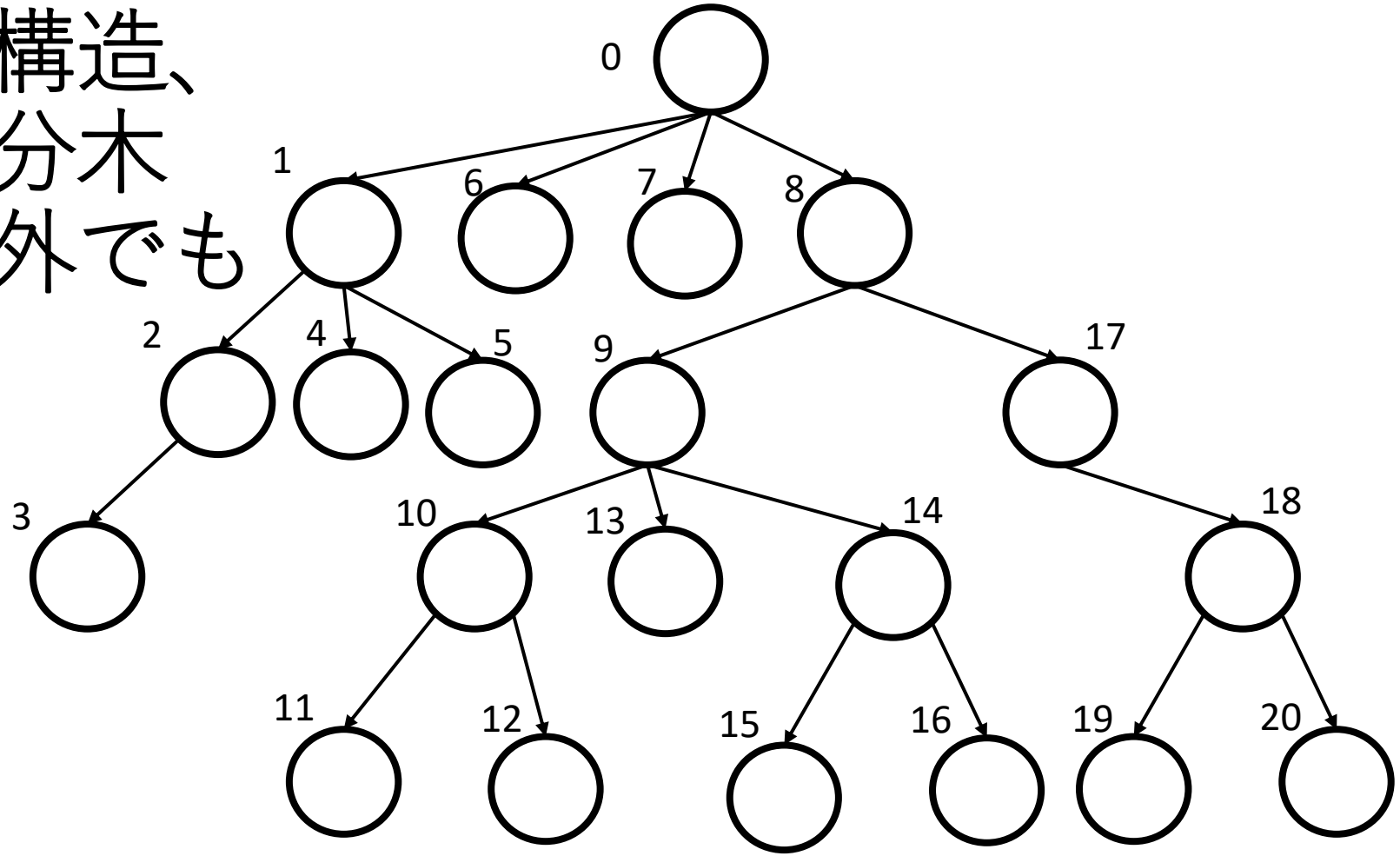


木構造の場合

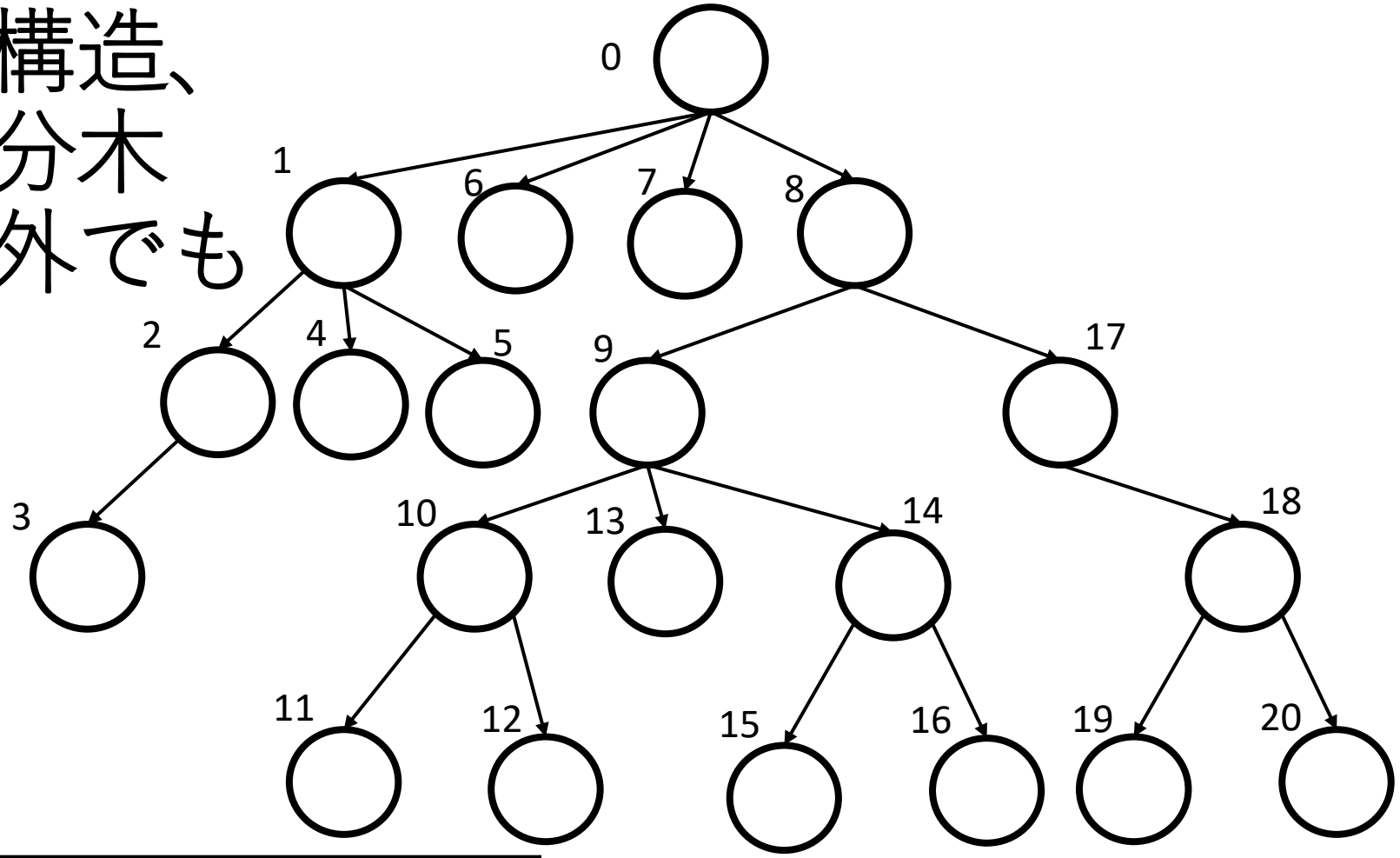
```
typedef struct node {
  struct node * left;
  struct node * right;
} node_t;

void dfs(node_t * n) {
  if (n->left != NULL) {
    dfs(n->left);
  }
  if (n->right != NULL) {
    dfs(n->right);
  }
}
```



木構造、２分木以外でも



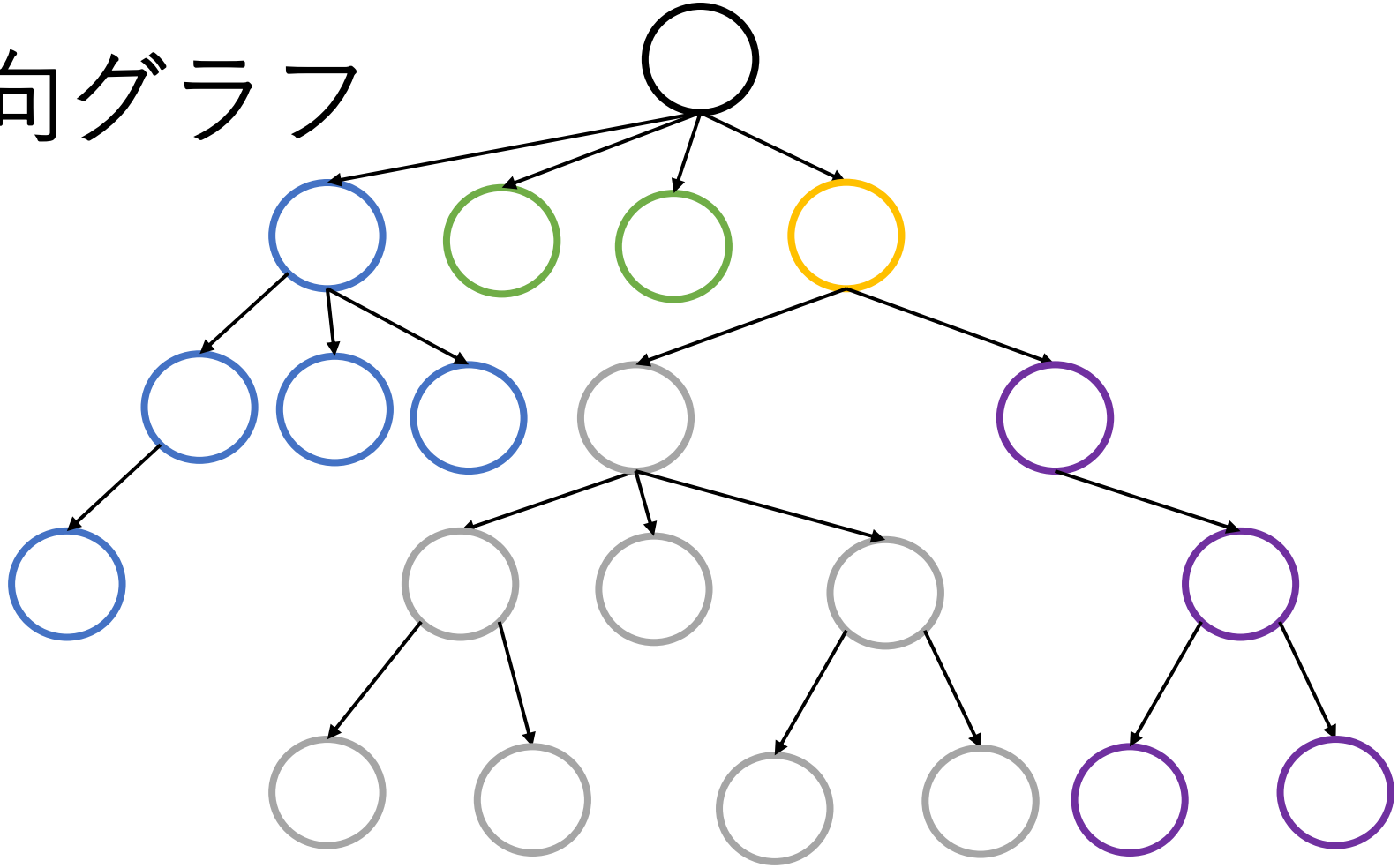
木構造、２分木以外でも



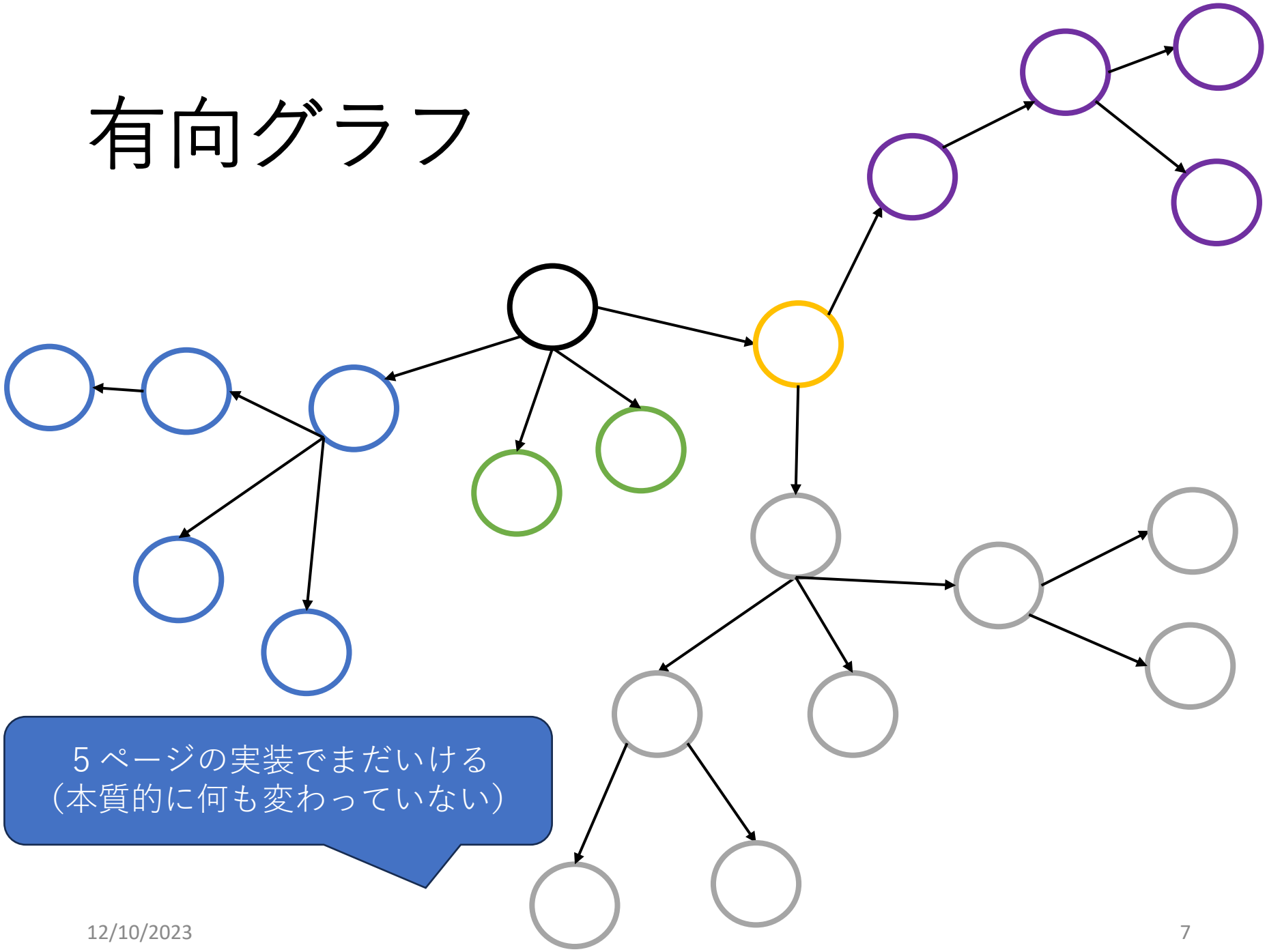

```
typedef struct node {
  struct node * neighbors [MAX_NEIGHBORS];
  int nbNeighbors;
} node_t;

void dfs(node_t * n) {
  for(int i=0; i < n->nbNeighbors; i++) {
    dfs(n->neighbors[i]);
  }
}
```

もう少し汎用的な実装

有向グラフ

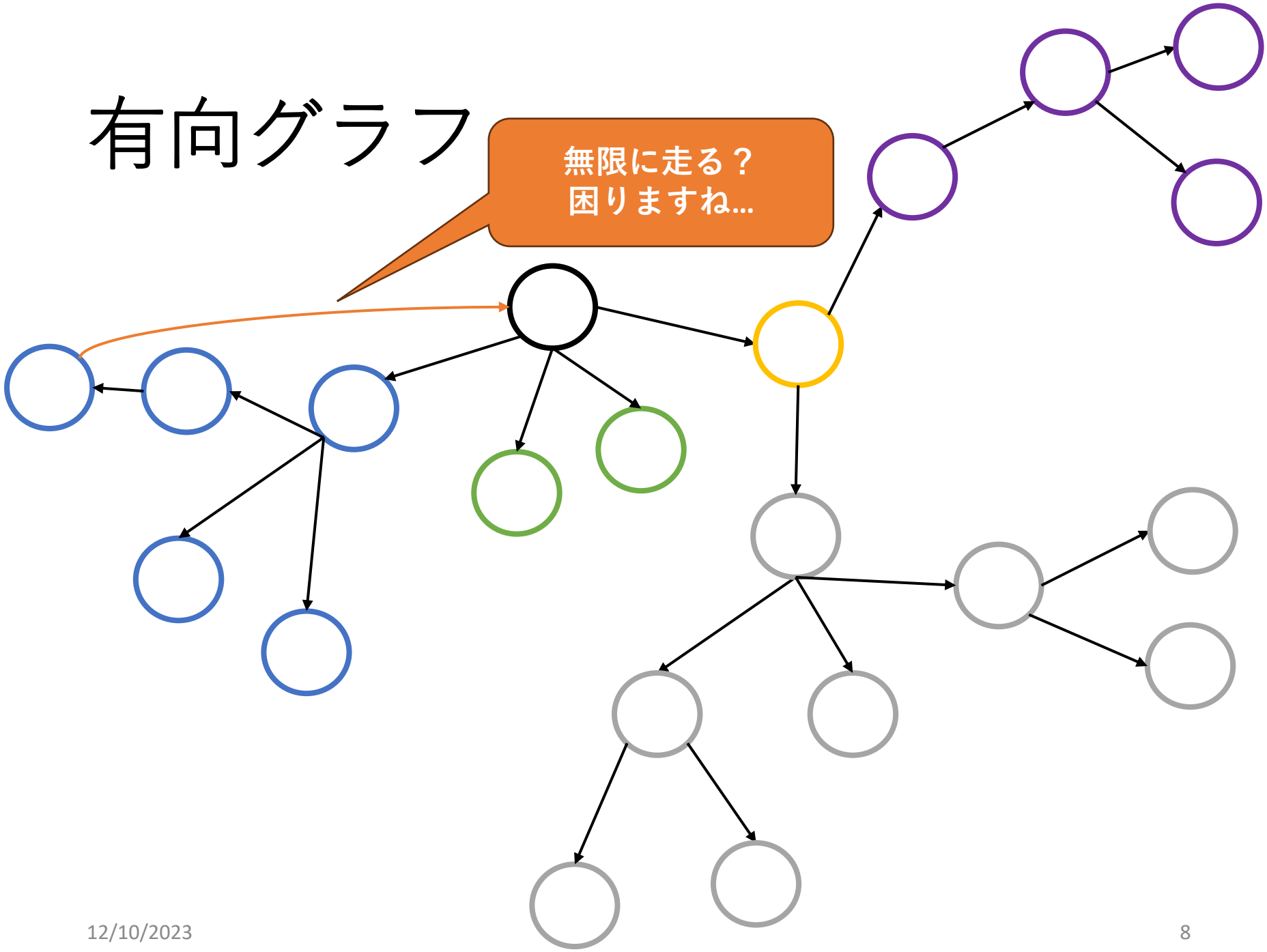


有向グラフ

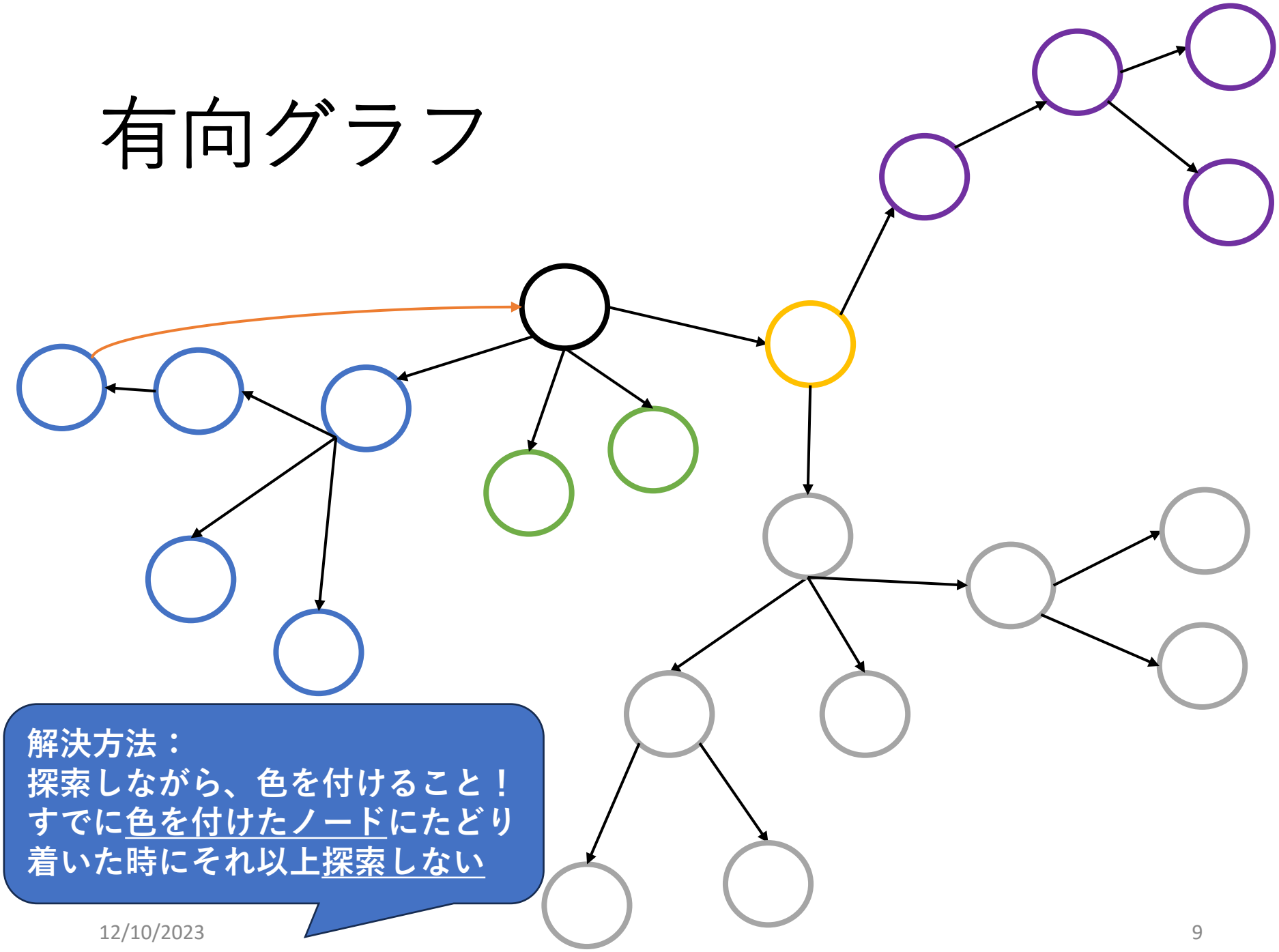


5 ページの実装でまだいける
（本質的に何も変わっていない）

有向グラフ

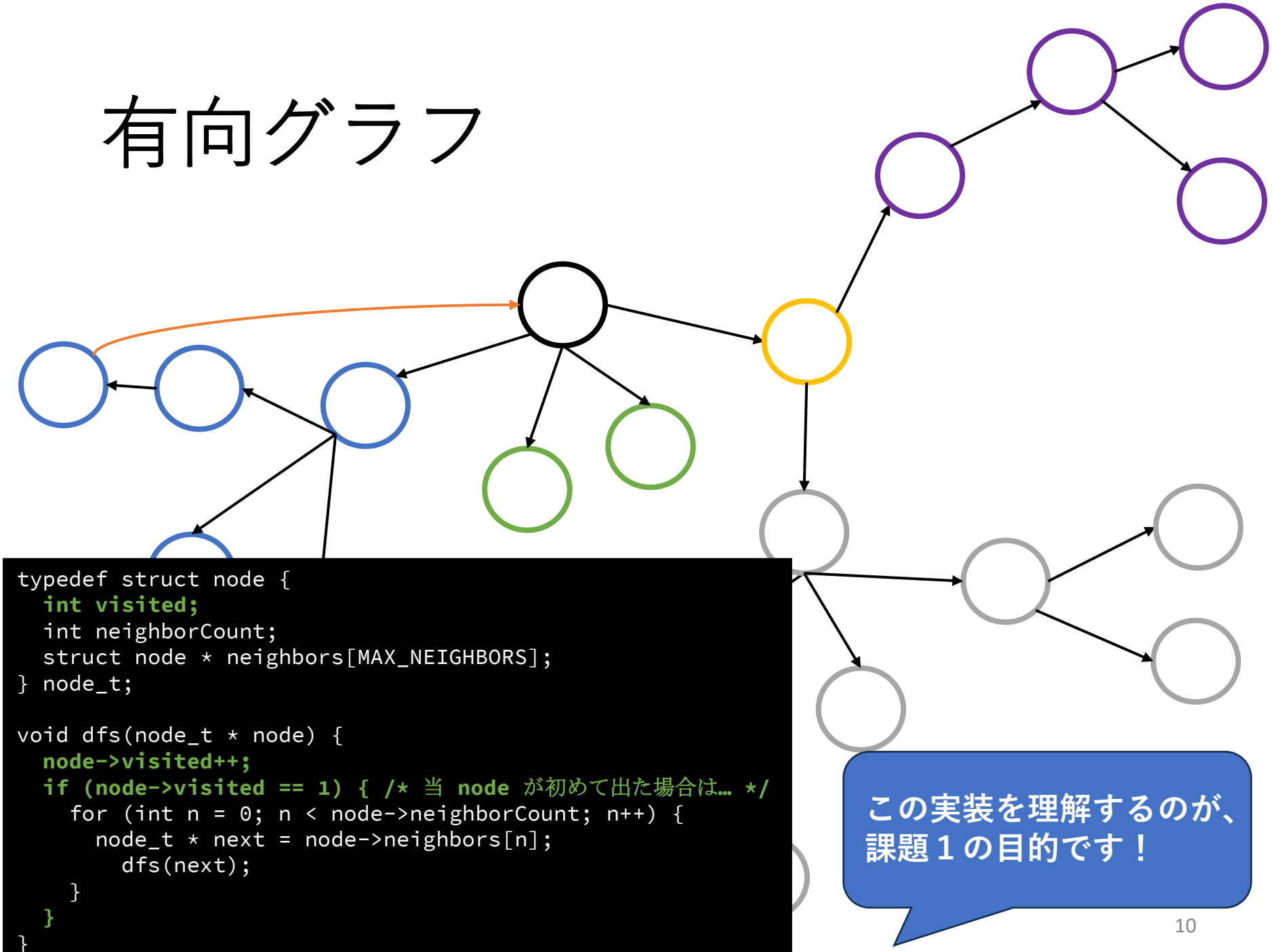


有向グラフ



解決方法：
探索しながら、色を付けること！
すでに色を付けたノードにたどり着いた時にそれ以上探索しない

有向グラフ



```c
typedef struct node {
  int visited;
  int neighborCount;
  struct node * neighbors[MAX_NEIGHBORS];
} node_t;

void dfs(node_t * node) {
  node->visited++;
  if (node->visited == 1) { /* 当 node が初めて出た場合は… */
    for (int n = 0; n < node->neighborCount; n++) {
      node_t * next = node->neighbors[n];
        dfs(next);
    }
  }
}
```

この実装を理解するのが、課題１の目的です！