به نام خدا

ساختمان داده ها

جلسه پانزدهم

دانشگاه صنعتی همدان

گروه مهندسی کامپیوتر

نیم سال دوم 98-1397

درختها

Trees

مطالب

- مقدمه و تعاریف
- نمایش درختان
- درختان دودویی
  - ADT درختان دودویی
  - خواص درختان دودیی
  - نمایش درختان دودویی
  - نمایش درختان با لیست ها
  - پیمایش درختان دودویی
  - اعمال روی درختان دودیی
  - درختان دودویی نخ کشی شده
- هرمها (Heaps)
- درختان جستجو
- درختان انتخاب
- جنگلها

چند مثال درباره اعمال روی درختها

- کپی کردن یک درخت در درخت دیگر

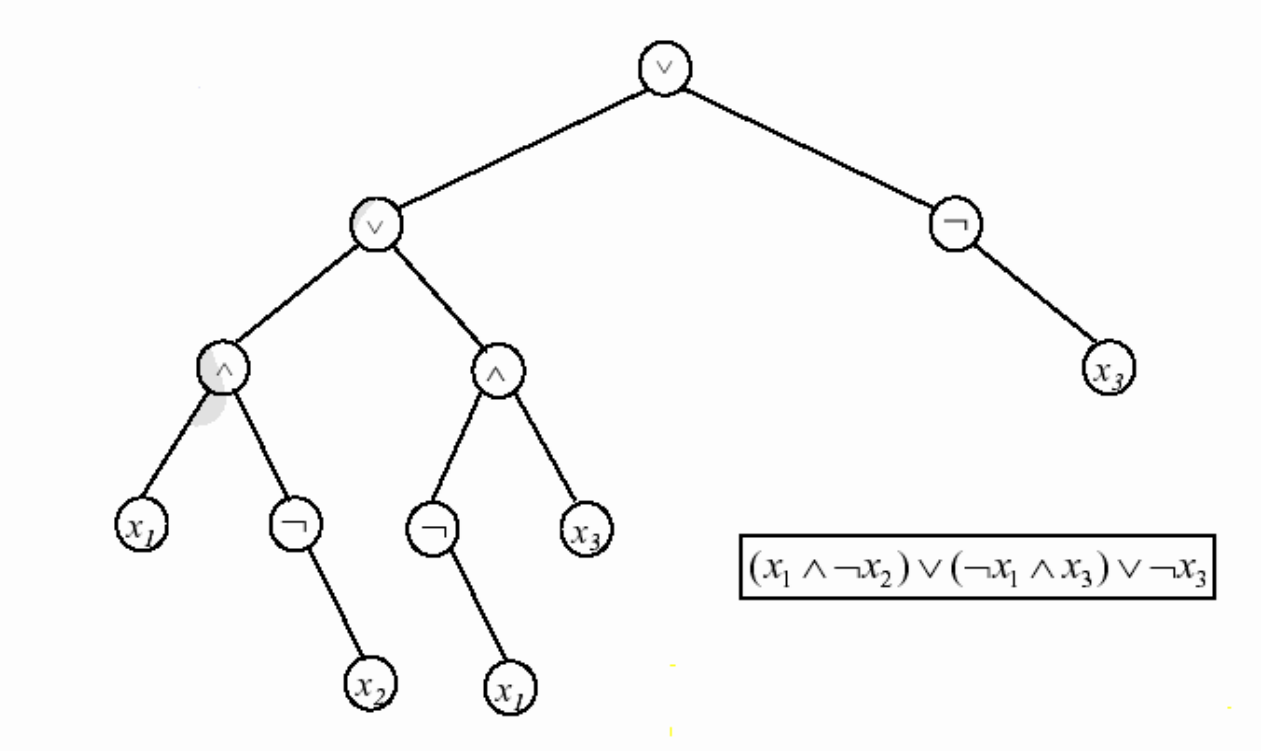
```
Tree::Tree(const Tree& s) {
  root = copy(s.root);
}

TreeNodes* Tree::copy(TreeNode *orig) {
  if(orig) {
    TreeNode *tmp = new TreeNode;
    tmp->data = orig->data;
    tmp->LeftChild = copy(orig->LeftChild);
    tmp->RightChild = copy(orig->RightChild);
    return tmp;
  }
  return 0; // an empty binary tree
}
```

تابعی برای بررسی مساوی بودن دو درخت

```
bool operator==(const Tree& s, const Tree& t)
{  return equal(s.root, t.root);  }

// assume the below function is a friend of class TreeNode
bool equal(TreeNode *a, TreeNode *b) {
  if((! a) && (! b)) return 1;  // both a and b are 0

  if(a && b // both a and b are non-0
       && (a->data == b->data) // data is the same
       && equal(a->LeftChild, b->LeftChild) // same left
       && equal(a->RightChild, b->RightChild)) // same right
         return true;
  return false;
}
```

مثال ارزیابی عبارتهای منطقی

- برای ارزیابی عبارتهای منطقی که با یک درخت پیاده شده اند می توان از پیمایش پس ترتیب استفاده کرد. در این روش یک فیلد value به گره ها اضافه می شود.



تابع ارزیابی

```
enum OpType { Not, And, Or, True, False };

class SatTree;  // forward declaration
class SatNode {
  friend class SatTree;
  SatNode *LeftChild;
  OpType data;
  bool value;
  SatNode *RightChild;
}
class SatTree {
  SatNode *root;
  void PostOrderEval(SatNode *);
public:
  PostOrderEval();
  void rootvalue() { cout << root->value; }
};
```

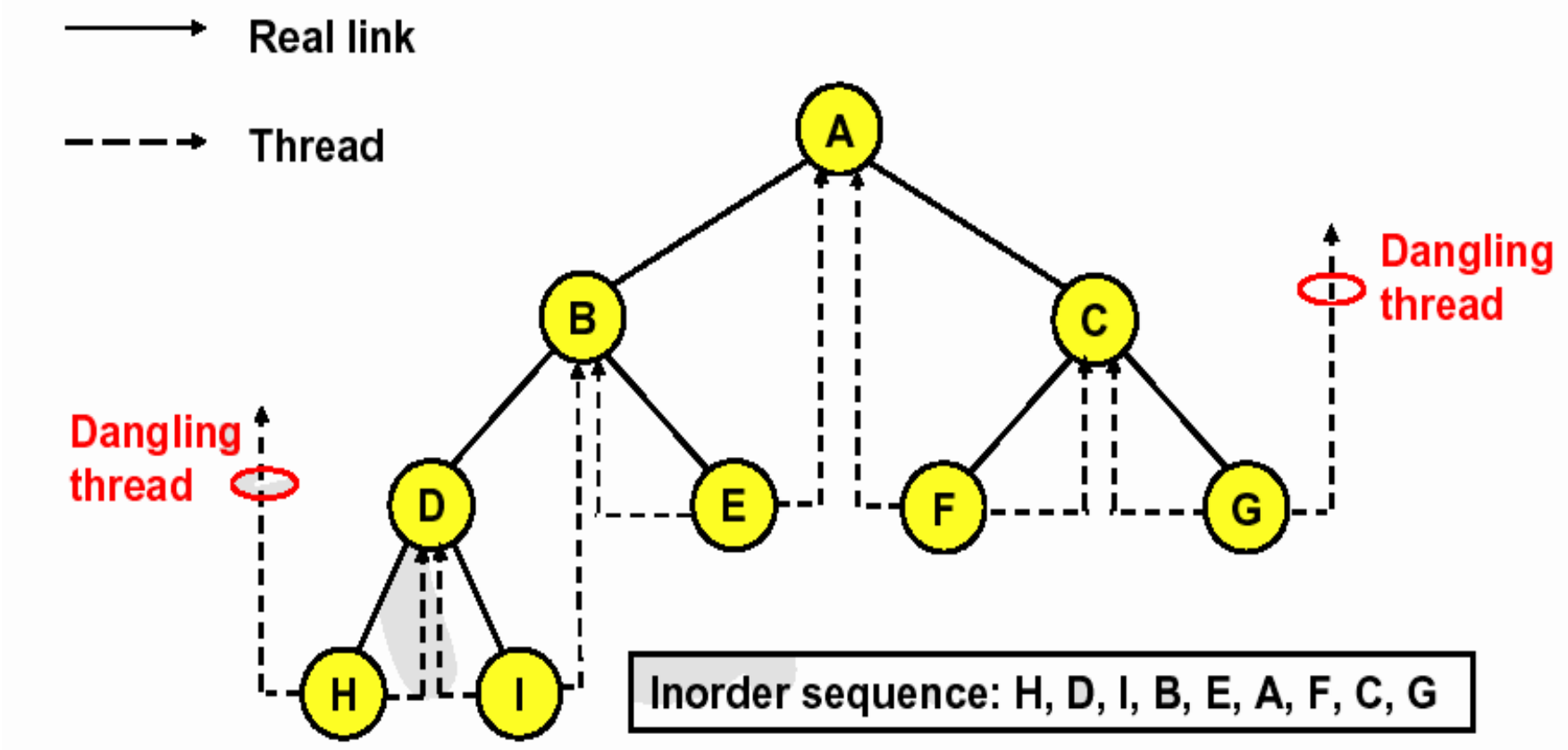
```
void SatTree::PostOrderEval() { PostOrderEval(root); }

void SatTree::PostOrderEval(SatNode *s) {
  if(s) { // not null
    PostOrderEval(s->LeftChild);
    PostOrderEval(s->RightChild);
    switch(s->data) {
      case Not  : s->value = ! s->RightChild->value; break;
      case And  : s->value = s->LeftChild->value &&
                             s->RightChild->value; break;
      case Or   : s->value = s->LeftChild->value ||
                             s->RightChild->value; break;
      case True : s->value = true; break; // terminal node
      case False: s->value = false; // terminal node
    }
  }
}
```

درختان نخ کشی شده (Threaded binary trees)

- در درختان دودویی تعداد لینکهای نول از تعداد لینکهای غیر نول بیشتر است. همیشه در یک درخت دودویی با n گره که دارای 2n لینک می باشد n+1 لینک نول است.
- می توان از این لینکهای نول برای بهبود الگوریتمها استفاده کرد.
- اگر از این لینکها برای اتصال به دیگر گره ها استفاده شود به آنها نخ می گوییم. درخت دودویی حاصل را درخت نخی یا نخ کشی شده می گویند.

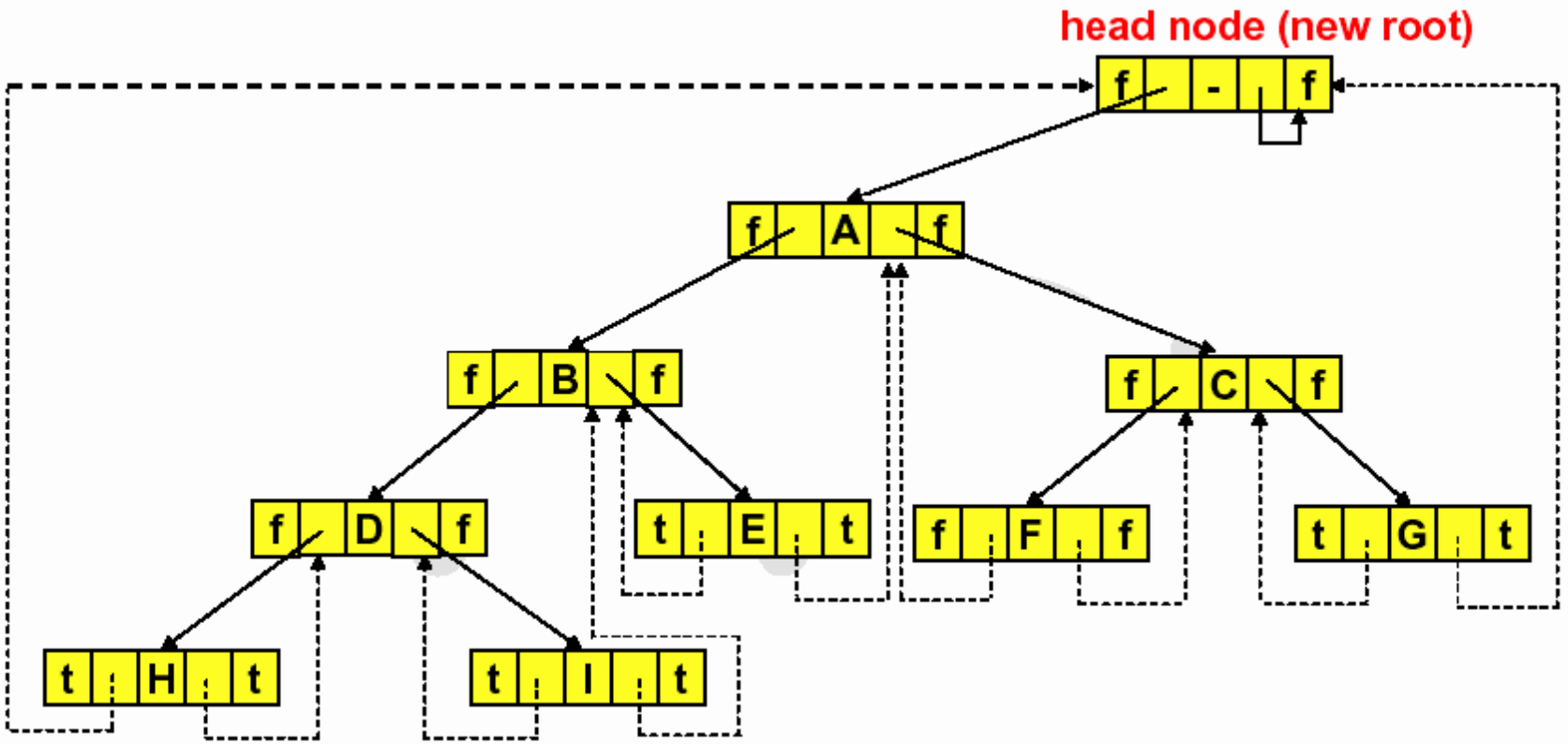
در نخ کشی درختها به صورت زیر عمل می کنیم:

- یک لینک تهی فرزند راست از گره p با آدرس گره بعدی در پیمایش میان ترتیب پر می شود
- یک لینک تهی فرزند چپ گره p با آدرس گره ما قبل p در پیمایش میان ترتیب پر می شود.


Real link
Thread
A
B
C
D
E
F
G
H
I
Dangling thread
Dangling thread
Inorder sequence: H, D, I, B, E, A, F, C, G

- برای تشخیص تفاوت بین اشاره گر واقعی و اشاره گر نخی از دو فیلد اضافی Boolean برای هر اشاره گر استفاده می کنیم. اگر اشاره گر نخی باشد فیلد مربوطه true است وگرنه false می باشد. بنابر این ساختار گره های یک درخت نخی به صورت زیر می باشد:

برای حل مشکل دو اشاره گر پایانی می توانیم از یک گره سر به صورت شکل زیر استفاده کنیم.



تعریف کلاس درختان نخی

```
class ThreadedNode {
   friend class ThreadedTree;
   friend class ThreadedInorderIterator;
   bool LeftThread;
   ThreadedNode *Left;
   char data;
   ThreadedNode *Right;
   bool RightThread;
};

class ThreadedTree {
   friend class ThreadedInorderIterator;
   ThreadedNode *root;
public:
   // ...
};
```

تعریف کلاس درختان نخی

```
class ThreadedInorderIterator {
  ThreadedTree& t;
  ThreadedNode* Cur;
public:
  ThreadedInorderIterator(ThreadedTree& t)
    :t(tree), Cur(t.root) { }
  Char* Next();
};

char* ThreadedInorderIterator::Next(){
  ThreadedNode *tmp = Cur->Right;
  if(! Cur->RightThread)
    while(! tmp->LeftThread) tmp = tmp->Left;
  cur = tmp;
  if(Cur == t.root) return 0;  // traversal done
  return &Cur->data;
}
```

پیمایش میان ترتیب درخت نخی

```
Void ThreadedInorderIterator::Inorder()
{
For (char *ch=Next();ch;ch=Next())
Cout<<*ch<<endl;
}
```