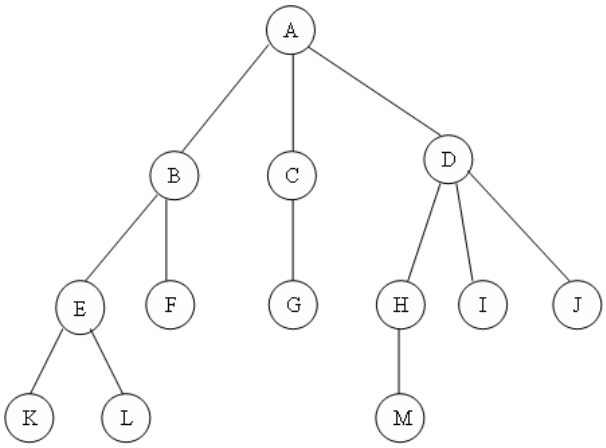
الفصل السابع : الأشجار
Trees

7-1 مفهوم الشجرة : هي مجموعة منتهية مكونة من عقدة أو أكثر بحيث تحقق :

1. يوجد عقدة مصممة بشكل خاص تدعى جذر الشجرة (root)
2. العقد المتبقية تجزأ إلى($n \geq 0$) من المجموعات المنفصلة $T_1, T_2, ..., T_n$ حيث كل مجموعة من هذه المجموعات تشكل شجرة . تدعى المجموعات $T_1, T_2, ..., T_n$ أشجاراً فرعية (sub-trees) من عقدة الجذر.

الشكل(7-1) يمثل شجرة تشير فيها العقد إلى معلومات و تربط الاتصالات بين العقد :



الشكل(7- 1) : شجرة

7-2 تعاريف

بفرض لدينا شجرة ما T

تعريف 1 :

- درجة عقدة (degree of node) هي عدد الأشجار الفرعية لهذه العقدة . فمثلا درجة العقدة A تساوي 3 و درجة العقدة C تساوي 1 و درجة العقدة F صفر .
- درجة الشجرة degree(T) هو العدد الأعظمي للأشجار الفرعية لعقدة أي أن :

degree (T) = max{degree(x) | x is a node of T} . درجة الشجرة T تساوي 3

تعريف 2 :

- تسمى كل عقدة لها شجرة فرعية واحدة على الأقل عقدة داخلية (internal node) مثل العقد : B, C, D, E, H
- تسمى كل عقدة ليس لها أي شجرة فرعية عقدة ورقية (leaf node) أو عقدة خارجية (external node) مثل العقد K, L, F, G, M, I, J

تعريف 3 :

– يمكن أن توجد بين العقد العلاقات التالية : أب (parent) – ابن(child) – أخ (sibling) – سلف (ancestor) – حفيد (grandchild). نقول عن عقدتين إنهما أختان (siblings) إذا كان لهما الأب نفسه . مثلا : العقدة B أب للعقد E, F ، وبالعكس العقدتان E, F أولاد للعقدة B . كما أن العقد H, I, J أخوات .

– نقول عن العقدة x إنها سلف لعقدة y ، إذا وفقط إذا كانت x أباً لـــ y ، أو كانت x سلفاً لأب y . مثلا : كل من العقد H, D, A يكون سلفاً للعقدة M

– نقول عن العقدة x إنها حفيد لعقدة y ، إذا وفقط إذا كانت x ابناً لـــ y ، أو كانت x حفيداً لابن y. العقد K, E, F أحفاد للعقدة B .

تعريف 4 :

نسمي مساراً (path) ضمن شجرة كل متتالية من العقد ، و نسمي فرعاً (branch) في الشجرة كل مسار يصل بين عقدة الجذر و إحدى العقد الورقية . مثلا :A, D, H, M فرع في الشجرة T

تعريف 5 :

- ارتفاع (مستوي) عقدة height(x) أو(level(x)) هو عدد الاتصالات في المسار الواصل بين هذه العقدة و عقدة الجذر و تعرف بالشكل العودي التالي :

$$height(x)=\begin{cases}0 & x\ is\ a\ root\\ 1+height(y) & y\ is\ a\ parent\ of\ x\end{cases}$$

- ارتفاع شجرة T هو أطول ارتفاع عقدة في الشجرة .

$$height\,(T)=\max\{\,height\,(x):x\ is\ a\ \ node\ \ in\ T\}$$

تعريف 6 :

حجم الشجرة Vol(T) هو عدد عقدها ويمكن تعريفه بالشكل التالي :

$$Vol(T)=1+\sum_{i=1}^{n}Vol(T_i)$$

حيث T_i الأشجار الفرعية لعقدة الجذر .

7-3 تمثيل الأشجار Representation of Trees

7-3-1 التمثيل باستخدام القوائم List Representation

7-3-1-1 القوائم الخطية :

توجد عدة طرق لرسم الأشجار ، إضافة للشكل (7-1) . على سبيل المثال يمكننا كتابة الشجرة في الشكل(7-1) على شكل قائمة خطية التي فيها كل شجرة فرعية تكتب بشكل قائمة خطية أيضا بالشكل التالي : (A(B(E(K,L),F),C(G),D(H(M),I,J)))

7-3-1-2 القوائم المترابطة : يتم تمثيل الشجرة بقائمة مترابطة من العقد ، حيث تتضمن كل عقدة حقلاً للبيانات و عدداً من حقول الربط . يعتمد هذا العدد على عدد الأشجار الفرعية لكل عقدة . و بالتالي يمكن أن يكون عدد حقول الربط مختلفاً من عقدة إلى أخرى (أي أن حجم العقدة مختلف من عقدة لأخرى).

تكون بنية العقدة في هذا التمثيل بالشكل التالي حيث كل حقل ربط يمثل ولداً للعقدة :

data	Link1	Link2	……..	link n

7-3-2 التمثيل باستخدام الابن اليساري –الأخ اليميني Left Child-Right Sibling

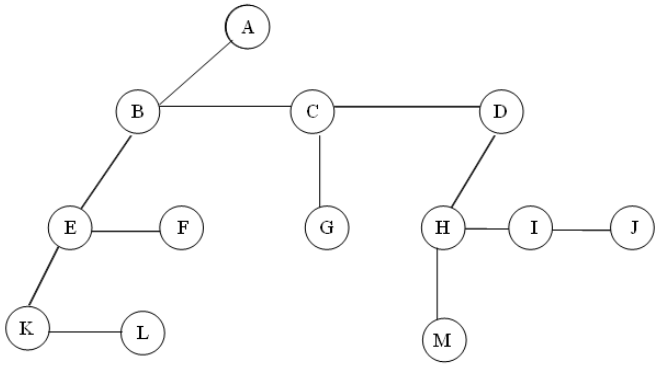
غالبا ما يكون التعامل مع العقد ذات الحجم الثابت أسهل ، لذلك لابد من اللجوء إلى طريقة تمثيل يكون فيها عدد الحقول في كل عقدة من عقد الشجرة ثابتاً .

في هذا التمثيل ، تتكون بنية العقدة من ثلاثة حقول : حقل للبيانات ، حقل ربط للابن اليساري ، و حقل ربط للأخ اليميني كما في الشكل التالي :

data	
Left child	Right sibling

لتحويل الشجرة من الشكل (7-1) إلى هذا التمثيل ، نلاحظ أن كل عقدة لها ابن واحد فقط يقع في أقصى اليسار (leftmost) و أخ يميني واحد الأقرب (closest right sibling) .

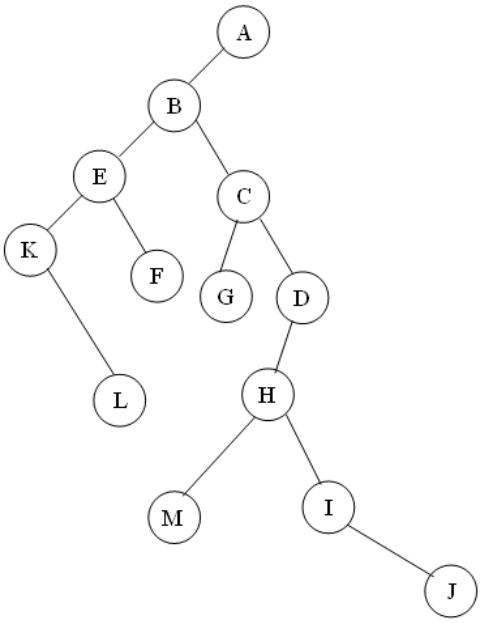
مثلا الابن في أقصى اليسار للعقدة B هو العقدة E و الأخ اليميني الأقرب لها هو العقدة C . و بما أنه لا أهمية لترتيب الأولاد في الشجرة ، و كون أي من أولاد العقدة يمكن أن يكون ابناً في أقصى اليسار، و أي من أخوات العقدة يمكن أن يكون الأخ اليميني الأقرب ؛ لذلك يمكننا إعادة رسم الشجرة في الشكل (7–1) باستخدام التمثيل الابن اليساري – الأخ اليميني بالشكل (7–2) :



الشكل (7–2) : التمثيل باستخدام الابن اليساري –الأخ اليميني

7–3–3 التمثيل باستخدام الابن اليساري –الابن اليميني Left child – Right child

بتدوير الشجرة في الشكل (7–2) بزاوية دوران 45 درجة باتجاه دوران عقارب الساعة ، نحصل على الشجرة كما في الشكل (7–3) فيها درجة كل عقدة على الأكثر 2 .



الشكل (7-3) : التمثيل باستخدام الابن اليساري –الابن اليميني

نلاحظ أنه توجد عقد في الشجرة المبينة في الشكل (7-3) تملك ابنين ، ندعوهما ابناً يسارياً و ابناً يمينياً .كما نسمي الأشجار الممثلة بهذه الطريقة أشجاراً ثنائية (Binary Trees) .

7-4 الأشجار الثنائية Binary Trees

تعريف : الشجرة الثنائية هي مجموعة منتهية من العقد ، إما أن تكون خالية أو مكونة من عقدة الجذر و شجرتين ثنائيتين منفصلتين : الشجرة الفرعية اليسارية و الشجرة الفرعية اليمينية . أي :

نسمي البنية B شجرة ثنائية إذا تحقق أحد الشرطين :

- B= ϕ (الشجرة فارغة)
- B = <O, B1, B2> حيث O عقدة الجذر . B1, B2 شجرتان ثنائيتان منفصلتان.
  نسمي B1 الشجرة الفرعية اليسارية وB2 الشجرة الفرعية اليمينية .

ملاحظة: يوجد فروق بين الأشجار الثنائية و الأشجار المعممة: 1- لا توجد شجرة معممة خالية (أي لا تتضمن أي عقدة) ، بينما توجد شجرة ثنائية خالية . 2- يوجد تمييز بين أبناء العقدة في الأشجار الثنائية ، بينما لا يوجد في الأشجار المعممة هذا التمييز .

7-4-1 خواص الأشجار الثنائية :

خاصة -1- : العدد الأعظمي للعقد في المستوي i من شجرة ثنائية يكون 2^i , $i \geq 0$

البرهان: لنثبت ذلك باستخدام الاستقراء الرياضي على i

-*أساس الاستقراء* : من أجل $i=0$ ، يوجد عقدة واحدة (عقدة الجذر) في المستوي 0 و بالتالي العدد الأعظمي للعقد في المستوي $i=0$ يكون $2^i = 2^0 = 1$.

- *الفرض الجدلي للاستقراء:* نفرض جدلاً أن العدد الأعظمي للعقد في المستوي j يكون 2^j .حيث $0 \leq j < i$

- *خطوة الاستقراء:* من الفرض الجدلي للاستقراء ، لدينا العدد الأعظمي للعقد في المستوي i-1 يساوي 2^{i-1} . و بما أن كل عقدة في الشجرة الثنائية لها ولدان على الأكثر و بالتالي العدد الأعظمي للعقد في المستوي i يساوي ضعفي العدد الأعظمي للعقد في المستوي i-1 ، أي يساوي 2^i .

خاصة -2- : العدد الأعظمي للعقد في شجرة ثنائية من الارتفاع h يكون $2^{h+1}-1$

البرهان: لدينا : $\sum_{k=0}^{k=h} 2^k = \frac{1-2^{h+1}}{1-2} = 2^{h+1}-1$

خاصة -3- : لتكن B شجرة ثنائية غير خالية و ليكن n_0 عدد العقد الورقية و n_2 عدد العقد التي من الدرجة 2 ، عندئذ يكون $n_0 = n_2+1$

البرهان: لتكن n حجم الشجرة B ، و لتكن n_1 عدد العقد من الدرجة 1 ، عندئذ يكون :

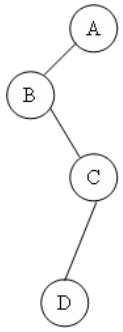
$$n = n_0 + n_1 + n_2 \qquad (1)$$

لتكن b عدد الاتصالات في الشجرة B . لدينا $n = 1+b$ ، و يوجد اتصالان لكل عقدة من الدرجة 2 و اتصال واحد لكل عقدة من الدرجة 1 لذلك يكون $b = n_1+2n_2$. و بالتالي نحصل على : $n = 1 + n_1 + 2n_2$ (2) .

بطرح المعادلة (2) من المعادلة (1) نحصل : $n_0 = n_2+1$

7-4-2 أنواع الأشجار الثنائية الخاصة

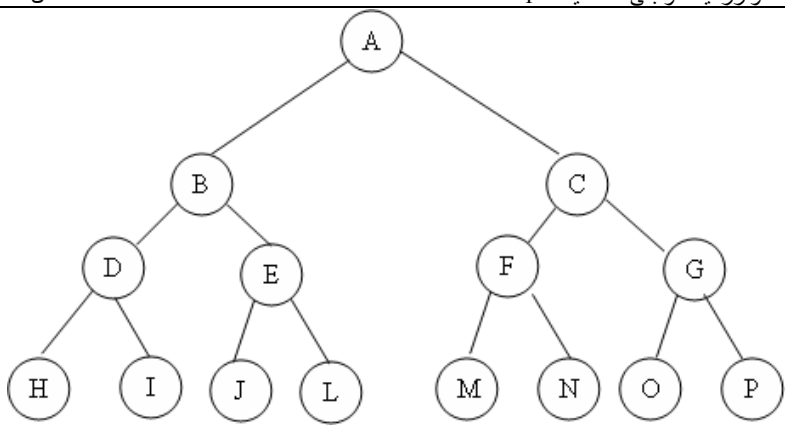
الشجرة الثنائية الخطية Linear Binary tree : هي شجرة ثنائية يكون فيها لكل عقدة ولد واحد على الأكثر، كما في الشكل (7-4)



الشكل (7-4): شجرة ثنائية خطية

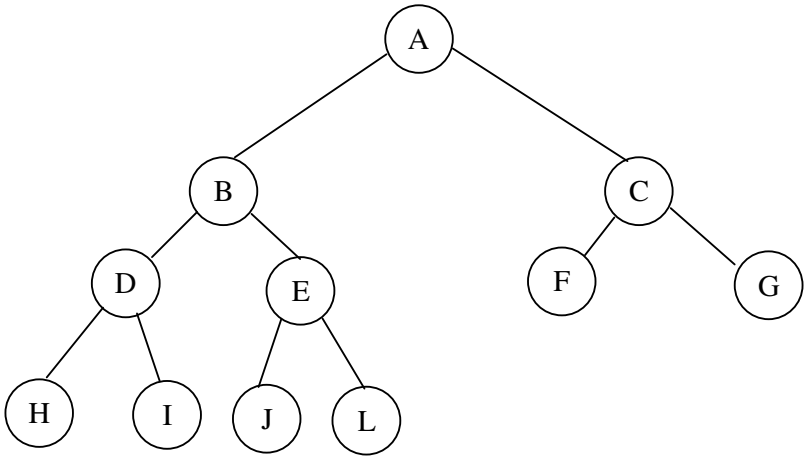
الشجرة الثنائية الممتلئة Full Binary Tree: هي شجرة تحوي عقدة واحدة في المستوي 0 و عقدتين في المستوي 1 ، و أربع عقد في المستوي 2 ،.....، 2^k في المستوي k ، أي تكون جميع مستوياتها ممتلئة بالعقد ، كما يوضح الشكل (7-5). و يكون عدد العقد في شجرة ممتلئة ارتفاعها h.

$$1 + 2 + 4 + ... + 2^h = \sum_{k=0}^{k=h} 2^k = \frac{1 - 2^{h+1}}{1 - 2} = 2^{h+1} - 1$$



الشكل(7-5) : شجرة ثنائية ممتلئة

الشجرة الثنائية الكاملة Complete Binary Tree : هي شجرة ثنائية جميع مستوياتها ممتلئة بالعقد ماعدا المستوي الأخير الذي يمكن أن يحوي فراغات . و في هذه الحالة تكون الفراغات من جهة اليمين كما يوضح الشكل (7-6).



الشكل (7-6) : شجرة ثنائية كاملة

7-4-3 دراسة نظرية لمحددات الأشجار الثنائية

نستعرض في هذه الفقرة بعض النظريات الخاصة بالأشجار الثنائية ، بغية تعّرف منهجية التعامل مع الأشجار .

نظرية 1

لتكن B شجرة ثنائية حجمها n و ارتفاعها h ، عندئذ يكون :

$$\lfloor \log_2 n \rfloor \le h \le n-1$$

البرهان

في حال ارتفاع معين h ، فإن الشجرة الثنائية B التي تحوي أقل عدد من العقد هي شجرة خطية ارتفاعها h . و يكون h=n-1 . و بالتالي من أجل أي شجرة ثنائية يكون $h \le n-1$.

أما الشجرة الثنائية B التي تحوي أكبر عدد من العقد فهي الشجرة الممتلئة من العقد ، و يكون حجمها $2^{h+1}-1$. و التالي من أي شجرة ثنائية يكون:

$$n < 2^{h+1} \Rightarrow \log_2 n < h+1 \Rightarrow \lfloor \log_2 n \rfloor \le h$$

نظرية 2

لتكن B شجرة ثنائية غير فارغة مكونة من n عقدة ، عندئذ عدد العقد من الدرجة الثانية n_2 يحقق : $n_2 \le \frac{n-1}{2}$

البرهان

سنبرهن بالاستقراء الرياضي على n

-*أساس الاستقراء* : من أجل n = 1 يكون $n_2=0$ و هذا محقق لأنه لا يوجد سوى عقدة الجذر

- *الفرض الجدلي للاستقراء:* نفرض جدلاً أن المتراجحة محققة من كل شجرة ثنائية حجمها أقل من n

- *خطوة الاستقراء:* لتكن B=<O, B1, B2> شجرة ثنائية حجمها n و ليكن n^1 حجم الشجرة الفرعية اليسارية B1 و n^2 حجم الشجرة الفرعية اليمينية B2 ، و $n=1+n^1+n^2$ يمكننا أن نكتب :

$$n_2 = 1 + n_2^1 + n_2^2 \le 1 + \frac{n^1 - 1}{2} + \frac{n^2 - 1}{2} = \frac{n^1 + n^2}{2} = \frac{n^1 + n^2 + 1 - 1}{2} = \frac{n - 1}{2}$$

نظرية 3

لتكن B شجرة ثنائية غير فارغة مكونة من n عقدة و عدد عقدها الورقية f ،عندئذ يكون ارتفاعها h يحقق المتراجحة : $h \ge \lceil \log_2 f \rceil$

البرهان

من النظرية 2 والخاصة 3 يكون $f \le \frac{n+1}{2}$ و كون التابع log متزايداً على مجموعة الأعداد الحقيقة الموجبة ، يكون :

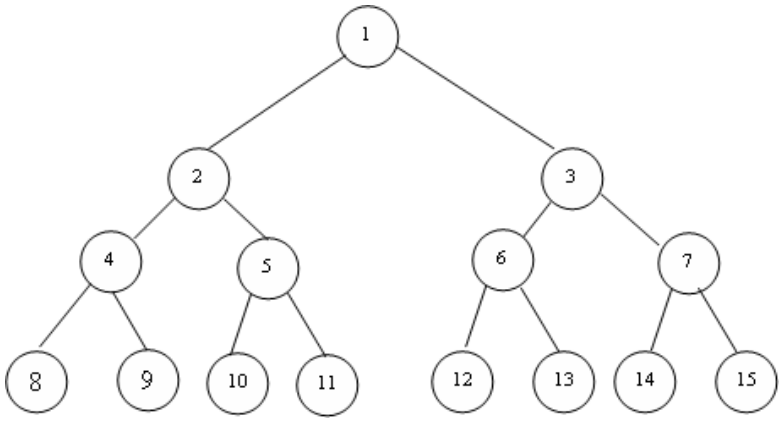
$$\log_2 f \le \log_2(\frac{n+1}{2}) \Rightarrow$$
$$\log_2 f \le \log_2(n+1) - 1 \Rightarrow$$
$$\log_2 f + 1 \le \log_2(n+1) \Rightarrow$$
$$\lceil \log_2 f \rceil + 1 \le \lceil \log_2(n+1) \rceil = \lfloor \log_2 n \rfloor + 1 \Rightarrow$$
$$\lceil \log_2 f \rceil \le h$$

7-4-4 تمثيل الأشجار الثنائية Binary Trees Representation

يمكن تمثيل الأشجار الثنائية بعدة طرق . لكل طريقة مزاياها التي تتعلق بالعمليات المطلوب القيام بها على الأشجار .

تمثيل الأشجار الثنائية باستخدام المتجهات Array Representation

بترقيم عقد الشجرة في الشكل (7-5) من 1 و حتى n , عندئذ نحصل على الشجرة ذات العقد المرقمة ، كما يظهر في الشكل (7-7)



الشكل (7-7) : شجرة ثنائية كاملة – مرقمة العقد –

نعرف متجهة ببعدين واحد لتخزين العقد (لا نستخدم الموقع 0 في المتجهة) ، و الثاني لتخزين موقع العقدة . باستخدام النظرية التالية يمكننا بسهولة تحديد مواقع : parent, left child, right child لأي عقدة i في الشجرة الثنائية .

نظرية

لتكن B شجرة ثنائية ممتلئة بالعقد و غير فارغة مكونة من n عقدة ، عندئذ يكون من اجل كل عقدة بدليل i حيث $1 \le i \le n$ ، يكون :

1. $$parent(i) = \begin{cases} \left\lfloor \frac{i}{2} \right\rfloor & if \;\; i \neq 1 \\ i \, has \, no \, parent & if \; i = 1 \; (i \; is \; the \; root) \end{cases}$$

2. $$left\_child(i) = \begin{cases} 2i & if \;\; 2i \le n \\ i \;\; has \, no \;\; left \;\; child & if \; 2i > n \end{cases}$$

3. $$right\_child(i) = \begin{cases} 2i+1 & if \;\; 2i+1 \le n \\ i \; has \, no \, right \, child & if \; 2i+1 > n \end{cases}$$

البرهان : ينتج البرهان مباشرة من الشجرة الثنائية في الشكل (7–7) .

يمكننا تمثيل الشجرة في الشكل (7–6) باستخدام المتجهات بالشكل التالي :

[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I

أما تمثيل الشجرة الخطية في الشكل (7–4) فيكون له الشكل :

[1]	A
[2]	B
[3]	----
[4]	----
[5]	C
[6]	----
[7]	----
[8]	----
[9]	----
[10]	D

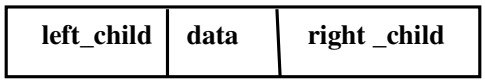
تمثيل الأشجار الثنائية باستخدام القوائم المترابطة Linked Representation

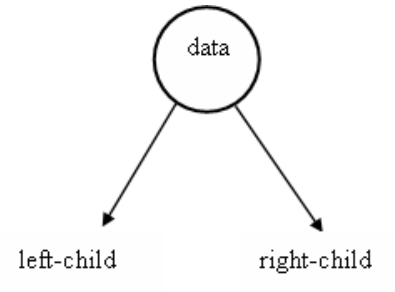
التمثيل باستخدام المتجهات يكون مقبولاً للأشجار الثنائية الممتلئة بالعقد ، ولكنه يبدد جزءاً من الذاكرة لتمثيل الأنواع الأخرى من الأشجار الثنائية . إضافة إلى ذلك ، إن حذف أو إضافة عقد من/إلى العقد الداخلية في الشجرة يتطلب تحريك عدة عقد مما يؤدي إلى تغيير

العدد الدال إلى مستوى العقد المتبقية . يمكننا التغلب على هذه المشكلات باستخدام التمثيل المترابط للشجرة . كل عقدة تملك ثلاثة حقول : left_child, data, right_child ، و تعرف في لغة C بالشكل :

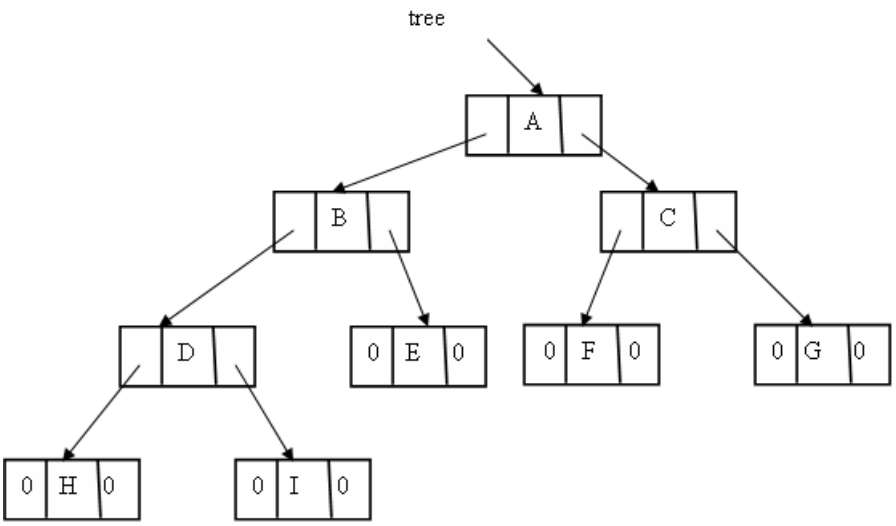
```
typedef struct node  *tree_pointer;
typedef struct node {
        int data;
        tree_pointer left_child, right_child;
         };
```

يمكننا رسم العقدة بالشكل :





الشكل (7-8) يظهر التمثيل للشجرة في الشكل(7-6) باستخدام القوائم المترابطة :



الشكل (7-9) : التمثيل المترابط للشجرة الثنائية في الشكل (7-6)

7-5 أشجار البحث الثنائية Binary Search Trees

تعريف شجرة البحث الثنائية:

نقول عن شجرة ثنائية B إنها شجرة بحث ثنائية إذا تحقق أحد الشرطين :

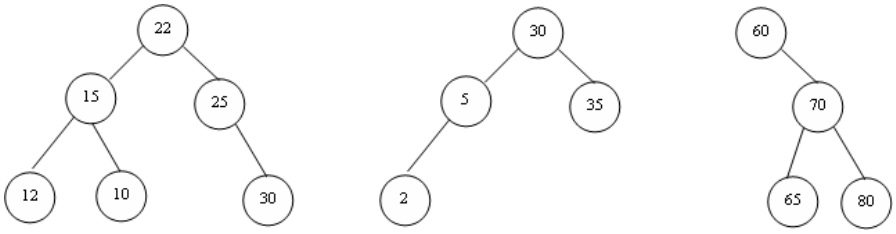
- B فارغة
- إذا كانت B غير فارغة ، يجب تحقق الشروط :

1. لكل عنصر(عقدة) في الشجرة له key ، و لا يوجد عنصران لهما نفس الـــ key (أي جميع الـــ keys مختلفة) .
2. كل الـــ keys في الشجرة الفرعية اليسارية أصغر من key الجذر.
3. كل الـــ keys في الشجرة الفرعية اليمينية أكبر من key الجذر.
4. كل من الشجرة الفرعية اليسارية و اليمينية هي شجرة بحث ثنائية أيضاً

ملاحظة: أشجار البحث الثنائية تدعم عمليات البحث عن عنصر ، إضافة عنصر ، و حذف عنصر .

مثال : في الشكل (7-10) ، أن كل من الشجرتين (c) ,(b) شجرة بحث ثنائية أما الشجرة (a) فهي ليست شجرة بحث ثنائية : لأنها لم تحقق الشرط الرابع .

<u>ملاحظة:</u> بما أن شجرة البحث الثنائية هي شكل خاص من الشجرة الثنائية و بالتالي الإعلان عن شجرة بحث ثنائية لا يختلف عن الإعلان المستخدم لإنشاء شجرة ثنائية .



(a) (b) (c)

الشكل (7-10) : أشجار ثنائية

7-5-1 البحث في شجرة البحث الثنائية Searching a Binary Search Tree

لنفرض أننا نرغب بالبحث عن عنصر بـــ key يساوي x في شجرة بحث ثنائية . نبدأ أولاً بالجذر . إذا كان الجذر يساوي NULL ، أي شجرة البحث لا تضمن أي عنصر ، و بالتالي البحث ينتهي بفشل . و إلا ، نقارن x مع key الجذر ، فإذا كانا متساويين ، عندئذ البحث ينتهي بنجاح . أما إذا كان x أصغر من key الجذر فعندئذ يتم البحث في الشجرة الفرعية اليسارية للجذر ، لأنه لا يوجد عنصر في الشجرة الفرعية اليمينية له key يساوي x. و إذا كان x أكبر من key الجذر عندئذ يتم البحث في الشجرة الفرعية اليمينية للجذر.

الدالة الإجرائية العودية المسؤولة عن تنفيذ عملية البحث هي :

```
tree_pointer search(tree_pointer root, int x)
 {
 /* return a pointer to the node that contains x. If there is no such node,
return null */
        if (root == null) return null;
        if ( x == root->data) return root;
        if (x < root->data) return search(root->left_child, x);
           return search (root -> right_child, x);
         }
```

الدالة الإجرائية التكرارية المسؤولة عن تنفيذ عملية البحث هي :

```
tree_pointer iter_search(tree_pointer tree, int x)
 {
 /* return a pointer to the node that contains x. If there is no such node,
return null */
      while (tree !=null) {
        if (x == tree->data) return tree;
        if ( x < tree ->data) tree = tree->left_child;
   else
         tree = tree->right_child;
                    }
         return null;
             }
```

تحليل دالة البحث : ليكن h ارتفاع شجرة البحث الثنائية ، عندئذ يتم إنجاز البحث باستخدام الدالة search() أو الدالة iter_search() بزمن قدره O(h) .

البحث عن أصغر key و أكبر key في شجرة البحث الثنائية:

بفرض لدينا شجرة بحث ثنائية tree . و لنرغب بالبحث عن عقدة تملك أصغر key (minimum key) و عقدة تملك أكبر key (maximum key) .

أولا : البحث عن أصغر key : يبدأ بالذهاب إلى الابن اليساري لعقدة الجذر ، ومن ثم إلى ابنه اليساري ، و هكذا حتى يتم الوصول إلى عقدة لا تتضمن ابناً يسارياً . عندئذ تكون هذه العقدة تملك أصغر key في الشجرة.

- الدالة الإجرائية المسؤولة عن ذلك :

```
tree_pointer minimum (tree_pointer node )
        {
          tree_pointer ptr;
          while(node !=null){
                ptr = node;
                node= node ->left_child;
                    }
          return ptr;
          }
```

ثانيا : البحث عن أكبر key : يأخذ هذا البحث المسار المعاكس حيث تتكرر العملية نفسها و لكن على الابن اليميني للعقدة ، و يستمر حتى يتم العثور على عقدة لا تتضمن ابناً يمينياً . عندئذ تكون هذه العقدة تملك أكبر key في الشجرة .

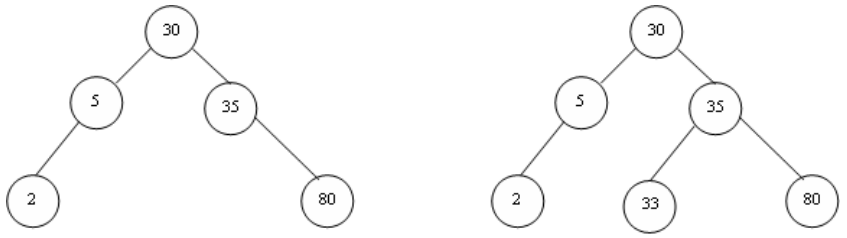
- الدالة الإجرائية المسؤولة عن ذلك :

```
tree_pointer maximum (tree_pointer node )
        {
          tree_pointer ptr;
          while(node !=null){
                ptr = node;
                node= node ->right_child;
                    }
          return ptr;
          }
```

7-5-2 الإضافة في شجرة البحث الثنائية Inserting a Binary Search Tree

لإضافة عنصر جديد بـــ key يساوي x ، يجب أن نتأكد من أن x مختلف عن كل key في الشجرة و ذلك باستخدام دالة البحث search . فإذا كان البحث فاشلاً ، عندئذ نضيف العنصر في الشجرة عند النقطة التي انتهى فيها البحث . على سبيل المثال ، لإضافة عنصر بـــ key = 80 إلى شجرة البحث (b) في الشكل (7-10) ، نلاحظ أولاً أن البحث ينتهي بفشل ، و العقدة ذات القيمة 40 هي آخر عقدة تم فحصها . عندئذ نضيف العقدة الجديدة كابن

يميني لهذه العقدة . الشكل (7−11) يظهر إضافة العقدة ذات key =80 و كذلك إضافة العقدة ذات key =33 .



إضافة 80 **إضافة 33**

الشكل (7−11) : الإضافة في شجرة بحث ثنائية

هذه الإستراتيجية تنفذ باستخدام دالة إجرائية insert_node الموصوفة بالشكل التالي :

```
void insert_node (tree_pointer *node, int num)

/* if num is in the tree pointed at by node do nothing ; otherwise add a
new node with data = num*/
      {
        tree_pointer ptr , temp = modified_search (*node, num);
        if (temp != null || (*node) == null){
          /* num is not in the tree */
          ptr = (tree_pointer )malloc(sizeof (node));
          if (ptr = = null ){
               printf("error : the memory is full");
             exit(1) ;
               }
           ptr ->data = num;
             ptr->left_child = ptr->right_child = null ;
         if (*node != null)   /* insert as child of temp */
             if (num < temp-> data) temp->left_child = ptr;
             else temp -> right_child = ptr;
          else *node = ptr;
         }
       }
```

حيث الدالة modified_search() نسخة معدلة عن الدالة iter_search() . هـذه الدالـة تبحث في شجرة البحث الثنائية tree* node عـن عقـدة بـــــ key = num . الدالـة modified_search() تسترجع null إذا كانت الشجرة فارغة أو إذا كانت الشجرة تتضمن العقدة ذات key = num . و غير ذلك ، تسترجع مؤشراً إلى آخر عقدة وصل إليها البحث . و معرفة بالإجرائية التالية:

```
tree_pointer modified_search(tree_pointer tree, int x)
 {
  tree_pointer ptr;
      if(tree == null) return null;
     while (tree !=null) {
         ptr = tree;
       if (x == tree->data) return null;
       if ( x < tree ->data) tree = tree->left_child;
  else
         tree = tree->right_child;
                    }
         return ptr ;
              }
```

تحليل الدالة insert_node()

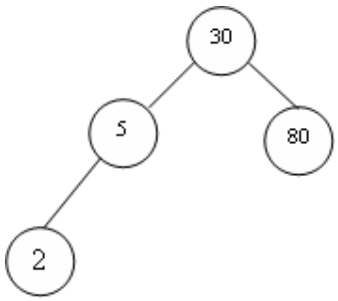
ليكن h ارتفاع الشجرة . تحتاج الدالة insert_node إلى زمن حساب قدره O(h) لتنفيذ البحث على الشجرة ، و كذلك تحتاج إلى Θ(1) من الزمن لإضافة عقدة في الموقع المناسب من الشجرة . و بالتالي الزمن الكلي للدالة الذكورة O(h) .

3-5-7 الحذف من شجرة البحث الثنائية Deletion from a Binary Search Tree

لحذف عقدة من شجرة بحث ثنائية نميز ثلاث حالات :

1. إذا كانت العقدة المراد حذفها عقدة ورقية . نجعل حقل الربط للابن (اليساري أو اليميني : حسب وضع العقدة الورقية) لوالد العقدة يساوي null ثم نحذف هذه العقدة .

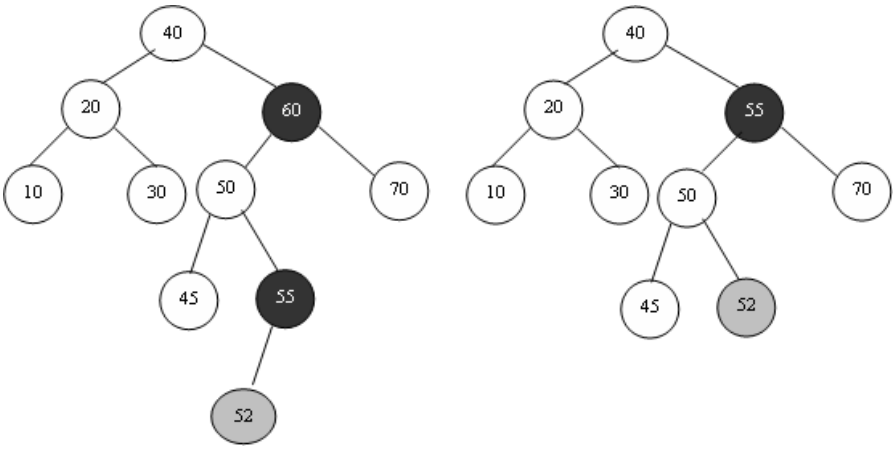
2. إذا كانت العقدة المراد حذفها عقدة داخلية لها ابن واحد . نستبدل هذه العقدة بعقدة الابن . على سبيل المثال ، بحذف العقدة 35 من الشجرة (a) من الشكل(7−11) نحصل على الشجرة التالية :



الشكل (7−12) : الحذف من شجرة البحث الثنائية

3. إذا كانت العقدة لها ابنان . عندئذ نستبدل هذه العقدة إما بأكبر عقدة في شجرتها الفرعية اليسارية ، أو بأصغر عقدة في شجرتها الفرعية اليمينية . و من ثم نباشر بحذف العقدة المستبدلة من الشجرة الفرعية . على سبيل المثال : لنفترض أننا نرغب بحذف العقدة 60 في الشجرة (a) من الشكل(7−13) ، عندئذ يمكننا تبديل العقدة 60 إما بالعقدة 70 أو بالعقدة 55 ولنفترض أننا سنبدل بالعقدة 55 في الشجرة الفرعية اليسارية . نحرك العقدة 55 إلى جذر الشجرة الفرعية اليمينية . ثم نجعل الابن اليساري للعقدة المتضمنة سابقاً القيمة 55 ، ابناً يمينياً للعقدة التي تتضمن القيمة 50 ، و من ثم نحذف العقدة القديمة التي كانت تتضمن القيمة 55 . الشجرة (b) من الشكل (7−13) تبين النتيجة الأخيرة لعملية الحذف هذه .

<u>ملاحظة:</u> من خلال الأمثلة لعملية حذف عقدة من شجرة بحث ثنائية ، نجد أن زمن الحساب لعملية حذف عقدة من شجرة بحث ثنائية بارتفاع h ينجز في O(h) .



(a) الشجرة قبل حذف 60 **(b) الشجرة بعد حذف 60**

الشكل (7-13) : حذف عقدة داخلية لها ولدين

7-5-4 ارتفاع شجرة البحث الثنائية Height of a Binary Search Tree

بفرض لدينا الـــــ keys 1, 2, 3, .. ,n ، يتم بناء شجرة بحث ثنائية للقيم المعطاة باستخدام الدالة insert_node لإضافة القيم السابقة إلى شجرة بحث ثنائية فارغة . إذا تم إضافة القيم المعطاة بنفس الترتيب نحصل على شجرة بحث ثنائية (خطية) يكون ارتفاعها أكبر ما يمكن و يساوي n-1 أما إذا تم أخذ القيم السابقة بشكل عشوائي ، فيكون ارتفاع شجرة البحث الثنائية الناتجة $O(\log_2 n)$ في الحالة الوسطية .

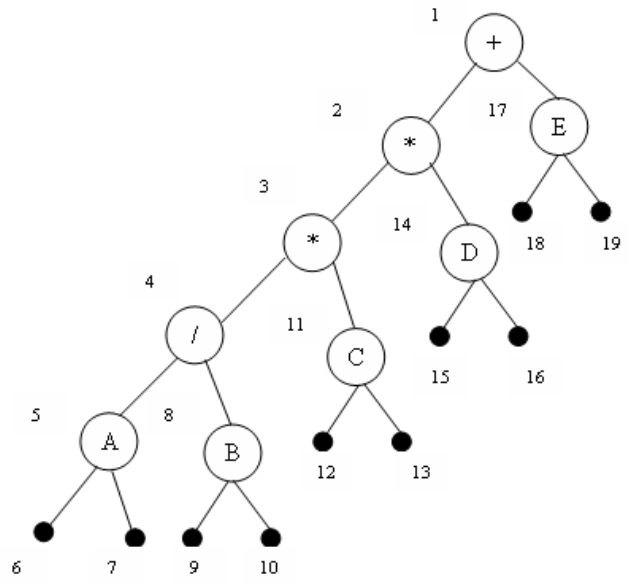
7-6 أساليب التنقل عبر الشجار الثنائية Binary Trees Traveling

يقصد بعملية التنقل المرور بكل عقدة (زيارة كل عقدة) من عقد الشجرة مرة واحدة حسب ترتيب معين بهدف إجراء معالجة معينة عليها (طباعة حقل البيانات في العقدة مثلا). التنقل على كامل الشجرة سينتج ترتيباً خطياً للمعلومات في الشجرة .

يرمز للتحرك يساراً بـــــ L ، و للتحرك يميناً بـــــ R ، و للمرور بالعقدة (زيارة العقدة) بـــــ V . و بالتالي يوجد ستة تراكيب هي : LVR, LRV, VLR و RVL, RLV, VRL . و لنفرض أنه يجب التحرك يساراً قبل التحرك يميناً ، عندئذ يبقى فقط ثلاثة

تراكيب : LVR, LRV, VLR . تدعى هذه الأساليب بـــــ : in-order, post-order, pre-order و ذلك اعتماداً على موضع V بالنسبة لـــــ L و R . مثلاً : في أسلوب التنقل post-order يتم المرور على عقدة بعد التنقل شجرتها الفرعية اليسارية و اليمينية ، بينما في أسلوب Pre-order فيتم زيارة عقدة قبل التنقل على أشجارها الفرعية .

يوجد تقابل بين أساليب التنقل هذه و إنتاج تعابير حسابية ممثلة بأنماط infix , postfix, prefix ، لتوضيح ذلك ، الشجرة في الشكل (7-14) تمثل التعبير الحسابي A/B*C*D+E



الشكل (7-14) : شجرة ثنائية تمثل تعبيراً حسابياً

7-6-1 التنقل بأسلوب In-order

التنقل يبدأ بالتحرك نزولا إلى أسفل الشجرة باتجاه اليسار حتى يصل إلى عقدة فارغة (null node) ، عندئذ يتم زيارة والد هذه العقدة ، و من ثم يتم التحرك باتجاه اليمين عقدة واحدة و نبدأ بتحرك جديد من هذه العقدة . إذا لم يكن مسموحاً التحرك نحو اليمين ، فيتم زيارة آخر عقدة غير مزاره في المستوي التالي الأعلى من الشجرة . و بعد ذلك يبدأ التحرك باتجاه اليمين عقدة واحدة و نبدأ بتحرك جديد من هذه العقدة

يمكننا استخدام مفهوم العودية في كتابة الدالة الإجرائية التي تنفذ هذا الأسلوب من التنقل ، و ستؤدي هذه الدالة ثلاث مهام .

1) تستدعي نفسها مرة أخرى للتنقل عبر الشجرة الفرعية اليسارية للعقدة node
2) تقوم بزيارة العقدة node
3) تستدعي نفسها مرة أخرى للتنقل عبر الشجرة الفرعية اليمينية للعقدة node

تكتب الدالة العودية التي تنفذ أسلوب التنقل in-order بالشكل التالي :

```
void in-order(tree_pointer  ptr)
   {
    if (ptr!=null){
      in-order(ptr->left_child);
      printf("%d", ptr ->data);
      in-order(ptr ->right_child);
        }
      }
```

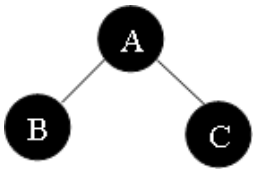
الجدول(7-1) يمثل تتبعاً لخطوات الدالة in-order باستخدام الشجرة في الشكل(7 -14) باعتبار أن جذر الشجرة هو دخل الدالة . حيث كل خطوة من هذا التتبع يمثل استدعاء للدالة in-order ، قيمة الجذر ، و عمل الدالة printf .

الجدول (7-1) : يمثل تتبعاً للدالة in-order()

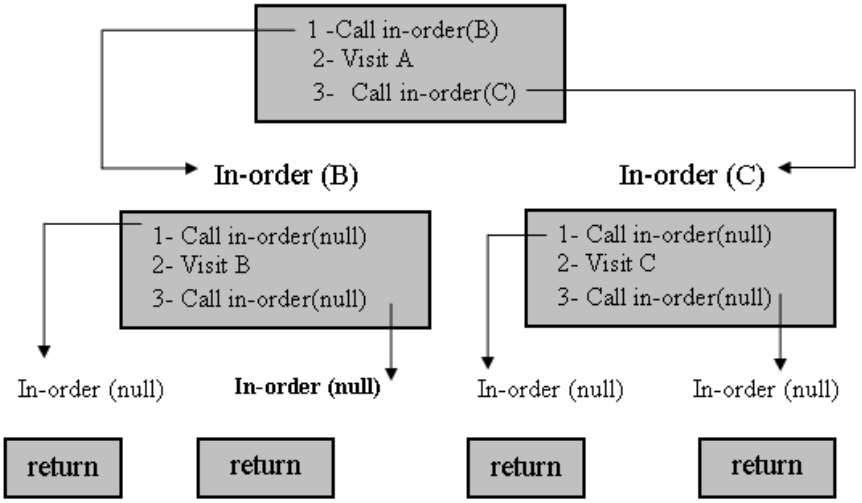
Call of In-order	Value of root	Action	Call of In-order	Value of root	Action
1	+	------	11	C	
2	*	------	12	null	
3	*	------	11	C	printf
4	/	------	13	null	
5	A	------	2	*	printf
6	null	------	14	D	
5	A	printf	15	null	
7	null	------	14	D	printf
4	/	printf	16	null	
8	B	------	1	+	printf
9	null	------	17	E	
8	B	printf	18	null	
10	null	------	17	E	printf
3	*	printf	19	null	

مثال: التنقل عبر شجرة مكونة من ثلاث عقد

لنعطي مثالاً يوضح فيه آلية العمل للدالة العودية in-order() من خلال شجرة مكونة من ثلاث عقد فقط : root(A), left child(B), right child(C) و الموضحة في الشكل (7-15) .



In-order (A)



الشكل (7-15) : التنقل عبر شجرة مكونة من ثلاث عقد

و تتلخص خطوات تنفيذ عملية التنقل بهذا الشكل في النقاط التالية :

1) استدعاء الدالة in-order() بوضع القيمة في ptr كعنصر دخل فيها . و يطلق على الاستدعاء in-order (A)
2) تستدعى الدالة in-order(A) التي تحتوي على left child(B) كعنصر دخل . و يطلق على هذا الاستدعاء الثاني اسم in-order(B)

3) تستدعى الدالة in-order(B) مرة أخرى مع وجود left child كعنصر دخل فيها . و بما أنها لا تحوي على left child ، فإن عنصر الدخل فيها سيكون (null) . و يطلق على هذا الاستدعاء اسم in-order(null)

4) سيكون هناك ثلاث نسخ من in-order() :

In-order(A), in-order(B), in-order(null),

و ينتهي in-order(null) بالعثور على القيمة الموضوعة فيها كعنصر دخل .

5) تبدأ in-order(B) في زيارة B ، باعتبار أن هدف الزيارة هو عرض محتويات العقدة B .

6) تواصل in-order(B) استدعاءها لـــــ in-order() مرة أخرى ، و لكن مع اختلاف عنصر الإدخال في هذه المرة ليكون القيمة right child ، و عندما تصل القيمة إلى Null مرة ثانية ، تنتهي الدالة بـــ in-order(null) .

7) يتم إنهاء المهام الثلاث السابق ذكرها في in-order(B)

8) يتم زيارة العقدة A ثم تستدعى الدالة in-order() مرة أخرى مع وجود العقدة C كعنصر إدخال فيها مكونة in-order(C) . و مثل in-order(B) ، فإن in-order(C) لا تحتوي على أبناء . لذا ، فإن الاستدعاء الأول ينتهي دون تنفيذ أية مهمة . و في الاستدعاء الثاني ، يتم زيارة C ، و تعود في الاستدعاء الثالث بلا أي تأثير .

9) تنتقل النتيجة في in-order(B) بعد ذلك إلى in-order(A)

10) بما أن in-order(A) قد تم تنفيذها بالفعل ، فإن الاستدعاء كله ينتهي ليكون بذلك قد تم انجاز عملية التنقل كاملة خلال الشجرة .

باستخدام مفهوم المكدس ، يمكننا كتابة الدالة in-order بالصيغة التكرارية:

```
void iter-in-order(tree_pointer node)
    {
      if (node == NULL) return;
       int top = -1;  /* initialize stack */
        tree_pointer stack[MAX_STACK_SIZE];
        for( ;  ; ) {
            for(  ; node !=null ; node = node ->left − child);
                add (&top, node);    /* add to stack */
            node = delete(&top);   /* delete from stack */
```

```
        if (node == null) break;  /* empty stack*/
        printf ("%d", node ->data);
        node = node ->right_child;
            }
        }
```

تحليل الدالة iter_in_order()

ليكن n عدد العقد في الشجرة ، نلاحظ أن كل عقدة من الشجرة توضع في المكدس ثم تحذف منه ، لمرة واحدة ، و التالي التعقيد الزمني للدالة : $\Theta(n)$.

7-6-2 التنقل بأسلوب Pre-order

يبدأ هذا التنقل بالعقدة الأولى ، و بعد ذلك يتم إتباع الفرع الأيسر بزيارة كل العقد الواقعة عليه حتى نصل إلى العقدة الفارغة (Null) ، و بعد ذلك يتم العودة إلى أقرب سلف لهذه العقدة له ابن يميني ، و بعد ذلك يستمر التنقل بهذا الابن . باستخدام هذا التنقل على الشجرة في الشكل 7-13 : ينتج التعبير الرياضي : +**/ABCDE

تعطى الدالة المسؤولة عن تنفيذ هذا التنقل بالصيغة العودية بالشكل التالي :

```
void pre-order(tree_pointer  node)
    {
    if (node !=null){
    printf("%d", node->data);
    pre-order(node->left_child);
    pre-order(node->right_child);
     }
       }
```

أما الصيغة التكرارية لدالة pre-order فتعطى بالشكل :

```
void  iter-pre-order(tree_pointer  node)
      {
        if( node == null) return;
        int top = -1;  /* initialize stack */
```

```
        tree_pointer stack[MAX_STACK_SIZE];
        add (&top, node);    /* add to stack */
      for( ;  ; ) {
          node = delete(&top);   /* delete from stack */
          if (node ==null) break;  /* empty stack*/
         printf ("%d", node ->data);
        if (node ->right_child != null)
           add(&top, node ->right_child);
        if (node ->left_child != null)
           add(&top, node ->left_child);
                }
            }
```

7-6-3 التنقل بأسلوب post-order

يبدأ هذا التنقل بزيارة أبناء العقدة قبل زيارة العقدة نفسها . باستخدام هذا التنقل على الشجرة في الشكل 6-13 : ينتج التعبير الرياضي : AB/C*D*E+

تعطى الدالة المسؤولة عن تنفيذ هذا التنقل بالصيغة العودية بالشكل التالي :

```
void post-order(tree_pointer  node)
    {
    if (node !=null){
     post-order(node->left_child);
     post-order(node->right_child);
     printf("%d", pter->data);
         }
     }
```

أما الصيغة التكرارية لدالة post-order فتعطى بالشكل :

```
void  iter-post-order(tree_pointer  node)
       {
         if( node== null) return;
         int top = -1;  /* initialize stack */
          tree_pointer stack[MAX_STACK_SIZE];
         add (&top, node);    /* add to stack */
         for( ;  ; ) {
              node = delete(&top);   /* delete from stack */
              if (node ==null) break;  /* empty stack*/
             printf ("%d", node ->data);
           if ( node ->left_child != null)
              add(&top, node ->left_child);
```

```
        if (node ->right_child != null)
            add(&top, node ->right_child);
                }
            }
```

ملاحظة: التعبير الحسابي الناتج من تطبيق الدالة ()iter-post-order يخزن في مكدس آخر، و من ثم يتم تفريغ هذا المكدس ، عندئذ نحصل على تعبير حسابي ممثل بنمط postfix .

6-6-4 التنقل بأسلوب level-order

يبدأ هذا التنقل بزيارة عقدة الجذر ، ثم زيارة الابن اليساري للجذر و بعد ذلك الابن اليميني للجذر، يستمر التنقل بهذا النمط ، عن طريق زيارة العقد في كل مستوٍ جديد ابتداءً من العقدة الواقعة في أقصى اليسار إلى العقدة الواقعة في في أقصى اليمين .

الدالة المسؤولة عن تنفيذ هذا التنقل level-order ، و التي تبدأ بإضافة عقدة الجذر إلى رتل ، ثم يتم حذف العقدة من مقدمة الرتل ، و طباعة حقل بيانات هذه العقدة ، و بعد ذلك يتم إضافة الابن اليساري ثم الابن اليميني لهذه العقدة إلى الرتل . و بما أن أبناء عقدة تقع في المستوي الأدنى التالي ، و يتم إضافة الابن اليساري للعقدة قبل الابن اليميني و بالتالي الدالة تطبع العقد للشجرة في الشكل 6-13 : +*E*D/CAB

```
void  level-order(tree_pointer  node)
      {
         int front=rear = -1;  /* initialize queue */
         tree_pointer queue[MAX_QUEUE_SIZE];
        if(node == null) return /*empty tree*/
        addq(&rear, node);    /* add to queue */
       for( ;  ; ) {
            node = deleteq(&front, rear);   /* delete from queue */
            if (node ==null) break;  /* empty queue*/
           printf ("%d", node ->data);
        if (node ->left_child != null)
            addq(&rear, node ->left_child);
        if (node ->right_child != null)
            addq(&rear, node ->right_child);
                }
            }
```

7-7 بعض العمليات الإضافية على الأشجار الثنائية Additional Binary trees operations

7-7-1 نسخ الأشجار الثنائية Copping Binary Trees

باستخدام مفهوم الأشجار الثنائية و أساليب التنقل Pre-order, In-order, Post-order عبر الأشجار الثنائية يمكننا نسخ شجرة ثنائية بالشكل التالي :

```
tree_pointer copy (tree_pointer original )
        {
  if (original !=null){
  tree_pointer temp = (tree_pointer)malloc(sizeof(node));
  if (temp == null){
    printf("error: the memory is full");
    exit(1);
          }
 temp->left_child = copy(original->left_child);
 temp->right_child = copy(original->right_child);
temp->data = orginal->data;
return temp
     }
return null;
    }
```

نلاحظ أن الدالة copy هي نسخة معدلة من أسلوب التنقل Post-order

7-7-2 اختبار تطابق الأشجار الثنائية Testing for Equality of Binary Trees

نقول عن شجرتين ثنائيتين إنهما متطابقتان إذا كانتا فارغتين معاً ، أو إذا كانتا غير فارغتين : تملكان نفس المعلومات في جذريهما و الشجرتان الفرعيتان اليساريتان من كلتيهما متطابقتان ، و الشجرتان الفرعيتان اليمينيتان من كلتيهما متطابقتان.

الدالة المسؤولة عن تنفيذ هذا التطابق هي :

```
int equal (tree_pointer first , tree_pointer second)
    {
   /*function returns FALSE if the binary trees first and second are not
equal, otherwise it returns TRUE */
   return(first == null && second == null) ||
          (first !=null && second !=null) &&
          (first -> data == second -> data) &&
          equal(first->left_child, second->left_child)&&
          equal(first->right_child, second->right_child))
           }
```

نلاحظ أن الدالة equal هي نسخة معدلة من أسلوب التنقل pre-order

7-7-3 مسألة التقييم (التحقيق) The Satisfiability Problem

تستخدم الأشجار الثنائية لتمثيل التعابير الحسابية و القضايا المنطقية المركبة ، يمكننا تقييم تعبير(حسابي أو منطقي) ممثل بشجرة ثنائية ، و ذلك باستخدام أسلوب التنقل post-order . حيث يتم تقيم كل تعبير جزئي الممثل بشجرة فرعية حتى يختصر التعبير الكلي في قيمة واحدة ، تكون نتيجة التقييم مخزنة في عقدة الجذر.

بنية العقدة للشجرة الثنائية التي تستخدم لتمثيل و تقييم التعابير مكونة من أربعة حقول : 1- حقل ربط للابن اليساري . 2- حقل للبيانات يمثل المتغير أو المؤثر . 3- حقل القيمة يمثل إما قيمة المتغير أو نتيجة تقييم الشجرة الفرعية 4- حقل ربط للابن اليميني . الشكل التالي يمثل بنية العقدة .

Left_child	Data	Value	Right- child

كما يتم التصريح عن بنية العقدة هذه بالصيغة التالية :

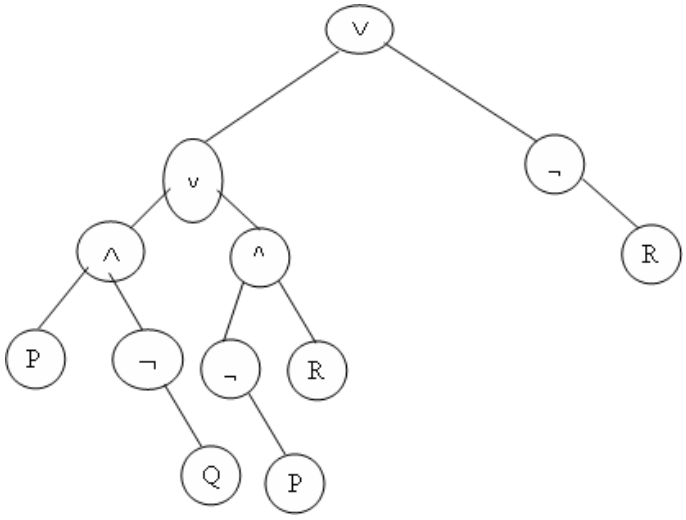
```
typedef struct node *tree_pointer ;
typedef struct node {
                      tree_pointer left_child;
                       type data ;
                       int value ;
```

```
        tree_pointer right_child ;
         };
```

مثال : ليكن التعبير المنطقي التالي

$$(P \wedge \neg Q) \vee (\neg P \wedge R) \vee \neg R$$

لتقييم التعبير المنطقي المعطى يتم أولا تمثيله باستخدام شجرة ثنائية :



الشكل (7-16) : التعبير المنطقي في الشجرة الثنائية

يعبر عن بنية العقدة في لغة C بالشكل التالي :

```
typedef enum {not, and, or, true, false} logical;
typedef struct node *tree_pointer ;
typedef struct node {
                    tree_pointer left_child;
                     logical data ;
                     short int value ;
                     tree_pointer right_child ;
                      };
```

يتم الحصول على الدالة الإجرائية التي تقيم التعبير المنطقي الممثل بالشجرة (7−16) بإجراء تعديل بسيط على الدالة post-order بالشكل التالي :

```
  void post-order-eval(tree_pointer node)
     {
     /* modified post order traversal to evaluate a
       propositional calculus tree */
if(node !=null){
     post-order-eval(node->left_child);
     post-order-eval(node->righr_child);
     switch (node->data) {
        case not : node->value=!node->right_child->value ;
               break;
        case and : node-> value =
                    node->right_chid->value &&
                    node -> left_child->value;
                     break;
                    case or :  node-> value =
                    node->right_chid->value ||
                    node -> left_child->value;
                     break;
                    case true :  node-> value =TRUE
                          break
                    case false :  node-> value =FALSE
                          break;
                          }
                     }
                    }
```

8−7 الأشجار الثنائية الخيطية Threaded Binary Tress

و جدنا من طريقة المؤشرات لتمثيل الأشجار الثنائية بأنه يوجد n+1 روابط صفرية من أصل 2n العدد الكلي للروابط أي أن عدد الروابط الصفرية أكثر من عدد المؤشرات الحقيقية .

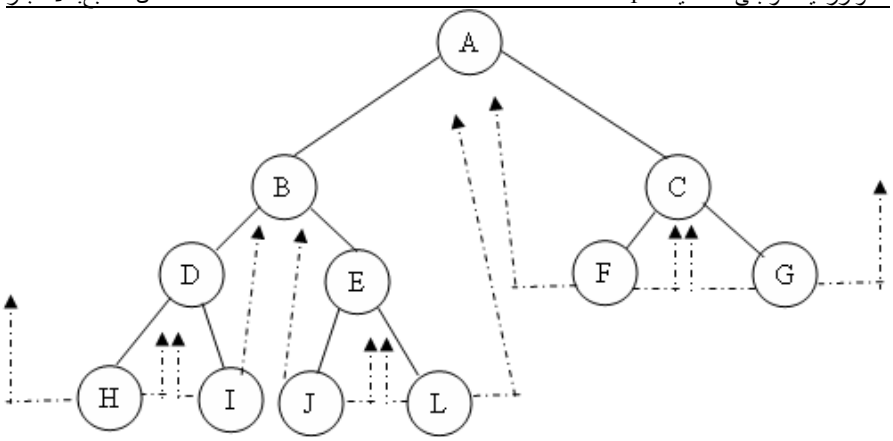
A. J. Perlis and C. Thornton قدموا طريقة لاستخدام الروابط الصفرية . استبدلوا الروابط الصفرية بمؤشرات إلى عقد أخرى من الشجرة و سموها خيوط (Treads) ، و لبناء هذه الخيوط نتبع القواعد التالية (بفرض أن ptr يمثل عقدة) :

1) إذا كان الرابط ptr -> left_child صفرياً ، عندئذ بدل هذا الرابط بمؤشر إلى العقدة التي تزار قبل العقدة ptr مباشرة بأسلوب التنقل in-order .
2) إذا كان الرابط ptr -> right_child صفرياً ، عندئذ بدل هذا الرابط بمؤشر إلى العقدة التي تزار بعد العقدة ptr مباشرة بأسلوب التنقل in-order .

يظهر الشكل (7–17) شجرة ثنائية بخيوط مرسومة بخطوط منقطة ، حيث تملك هذه الشجرة 11 عقدة و 12 رابطاً صفرياً ، حيث تم استبدالهم بخيوط . إذا تنقلنا عبر هذه الشجرة بأسلوب in-order ، عندئذ يتم زيارة العقد في الترتيب التالي : H, D, I, B, J, E, L, A, F, C, G . لنوضح كيف تم إنشاء الخيوط للعقدة L مثلاً . كون الابن اليساري للعقدة L رابطاً صفرياً ، نبدل هذا الرابط بمؤشر إلى عقدة تأتي قبل L و التي هي العقدة E . و كذلك كون الابن اليميني للعقدة L رابطاً صفرياً ، نستبدل هذا الرابط بمؤشر إلى عقدة تأتي بعد L و التي هي العقدة A .

7–8–1 تمثيل الأشجار الثنائية الخيطية

لتمثيل هذا النوع من الأشجار يجب أن نميز بين الخيوط و المؤشرات الطبيعية ، و يتم ذلك بإضافة حقليين جديدين إلى بنية العقدة : left-thread, right- thread . و لنفرض من أجل عقدة ما ptr في الشجرة الخيطية . إذا كان ptr -> left-thread = TRUE عندئذ : ptr -> left-thread يتضمن خيطاً ؛ و غير ذلك يتضمن مؤشر إلى الابن اليساري . و بشكل مشابه نجد إذا كان ptr -> right-thread = TRUE عندئذ : ptr -> right-thread يتضمن خيطاً ؛ و غير ذلك يتضمن مؤشراً إلى الابن اليميني .



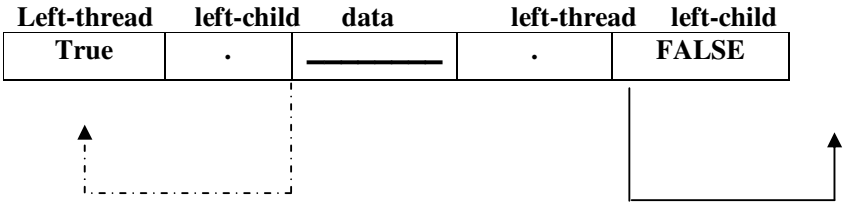
الشكل(7-17): شجرة ثنائية خيطية موافقة للشجرة في الشكل(7-6)

يتم الإعلان عن بنية عقدة في شجرة ثنائية خيطية بالشكل :

```
typedef struct threaded-tree  *threaded-pointer ;
typedef struct threaded-tree {
        short int left-treaded;
        threaded-pointer left-child;
        char data;
        short int right-treaded;
        threaded-pointer right-child;
          };
```

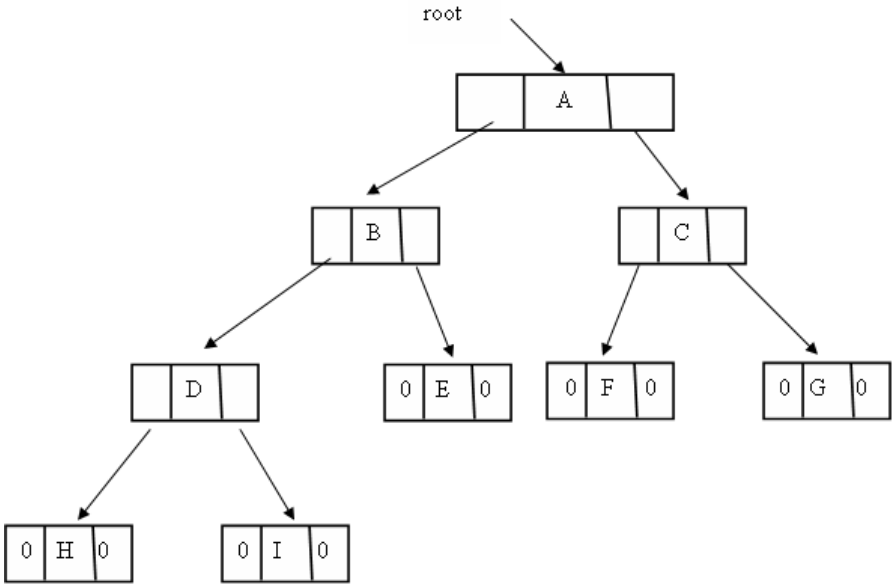
ملاحظة

يوجد في الشجرة من الشكل(7-17) خيطيان : الابن اليساري لـــــ H ، و الابن اليميني لـــــ G . في حالة H ، لا يمكننا تبديل رابط الصفري للابن اليساري لـــــ H بخيط إلى العقدة التي تزار قبل العقدة H مباشرة كون H أول عقدة تزار في هذا الأسلوب . و بشكل مشابه نجد أنه لا يمكننا تبديل رابط الصفري للابن اليميني لـــــ G بخيط إلى العقدة التي تزار بعد العقدة G مباشرة كون G آخر عقدة تزار في هذا الأسلوب . لذلك نفترض أن كل الشجار الثنائية الخيطية تملك عقدة رأسية (head node) و بالتالي أن كل شجرة ثنائية خيطية فارغة تتكون من عقدة واحدة ممثلة بالشكل(7-18) .



الشكل(7-18) : شجرة ثنائية خيطية فارغة

الشكل (7-19) يظهر التمثيل الكامل للشجرة في الشكل باستخدام المؤشرات .



الشكل (7-19) : التمثيل المترابط للشجرة الثنائية في الشكل (7-17)

7-8-2 التنقل بأسلوب In-Order عبر الشجرة الثنائية الخيطية:

باستخدام الخيوط يمكننا تبسيط خوارزمية التنقل بأسلوب in-order عبر الشجرة الثنائية الخيطية.فمن أجل أي عقدة ptr من شجرة خيطية نجد : إذا كان ptr->right-thread=TRUE عندئذ العقدة التي تلي العقدة ptr بنمط in-order هي ptr->right-child . و غير ذلك نحصل على العقدة التالية لـــــــ ptr بتعقب مسار من روابط الابن اليساري للابن اليميني للعقدة ptr حتى نصل إلى عقدة بـــــ left-thread=TRUE . الدالة الإجرائية insucc() توجد العقدة التالية لأي عقدة بأسلوب in-order .

```
threaded-pointer insucc(threaded-pointer tree)
        {
/* find the in-order successor of tree in a threaded binary tree */
        threaded-pointer temp;
        temp = tree -> right-child;
        if (tree->right-threaded != TRUE)
        while (temp-> left-threaded !=TRUE);
            temp =temp -> left-child;
        return temp;
        }
```

الدالة الإجرائية التي تنفذ عملية التنقل بأسلوب in-order عبر الشجرة الخيطية ، tin-order ، تستدعي الدالة insucc() لعدة مرات .

```
void tin-order(threaded-pointer tree)
        {
/* traverse the threaded binary tree in-order */
    threaded-pointer temp = tree;
    for( ; ; ) {
    temp = insucc(temp);
    if(temp = tree) break;
    printf("%c", temp->data);
        }
  }
```

تحليل الدالة ()tin-order

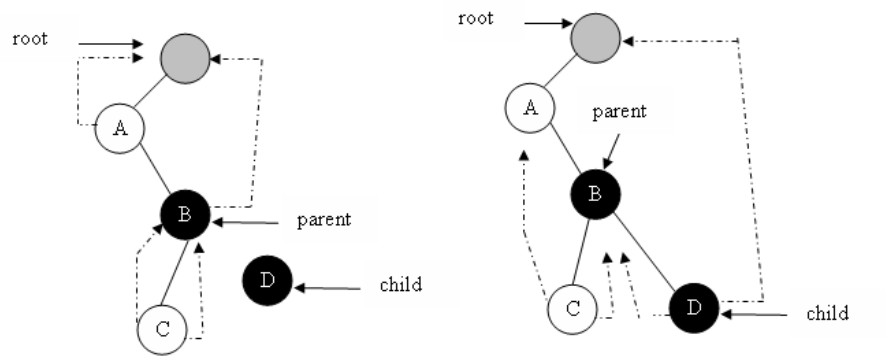
لنفرض أن الشجرة الخيطية مكونة من n عقدة . زمن حساب هذه الدالة O(n) ، و هو أقل بمقدار ثابت من زمن حساب الدالة iter-in-order .

7-8-3 إضافة عقدة إلى شجرة خيطية Inserting a node into threaded binary tree

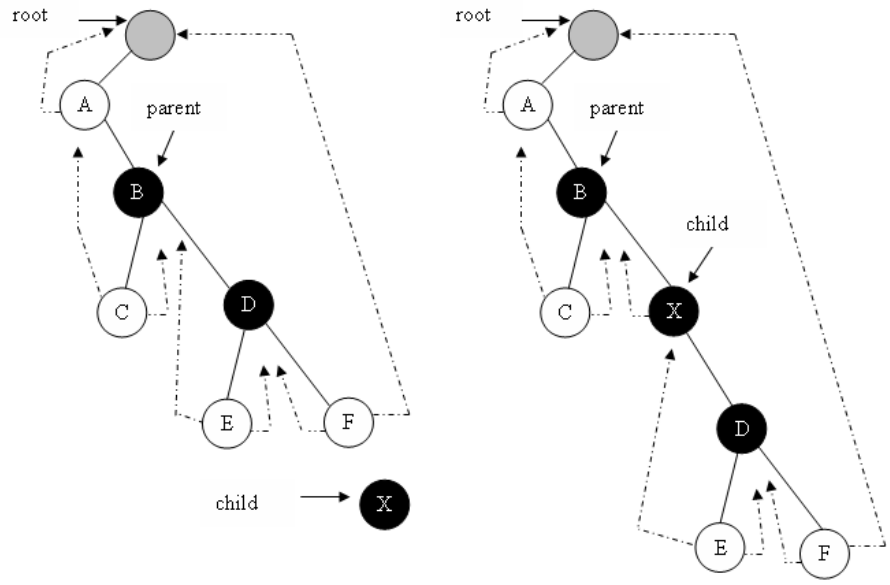
لتكن لدينا شجرة ثنائية خيطية tree . لتكن parent عقدة ما من الشجرة لها شجرة فرعية يمينية فارغة . و لنفرض أننا نرغب بإضافة العقدة child كابن يميني للعقدة parent . لإضافة العقدة child نتبع الخطوات التالية :

1) Change parent -> right-threaded to FALSE
2) Set child -> left-threaded and child -> right-threaded to TRUE
3) Set child -> left-child to point to parent
4) Set child -> right-child to parent -> right-child
5) change parent -> right-child to point to child

يوضح الشكل(7-20) (a) عملية إضافة العقدة D كابن يميني للعقدة B .
أما إذا كان للعقدة parent شجرة فرعية يمينية غير فارغة ، عندئذ تكون عملية الإضافة هذه أكثر تعقيداً ، لأن الشجرة الفرعية اليمينية للعقدة parent تصبح شجرة فرعية يمينية للعقدة child بعد الإضافة . الشكل(7-20) (b) يوضح عملية إضافة العقدة X بين العقدتين B, D . الدالة الإجرائية insert-right تنفذ عمليتي الإضافة السابقتين .



(a)



before **after**

(b)

الشكل(7–20) : إضافة ابن يميني لعقدة ما في شجرة خيطية ثنائية

```
void insert-right (threaded-pointer parent, threaded-pointer child)
{
/* insert child as the right child of parent in a threaded binary tree */
  threaded-pointer temp;
 child -> right-child = parent -> right-threaded;
 child -> left-child = parent;
 child -> left-thread = TRUE;
 parent -> right-child = child;
 parent -> right-thread = FALSE ;
 if ( child -> right-threaded == FALSE) {
    temp = insucc(child);
    temp -> left-child = child ;
            }
       }
```

7-9 الغابة Forest

تعريف: الغابة هي مجموعة مكونة من n $(n \geq 0)$ أشجاراً منفصلة . يعتبر مفهوم الغابة شبيهاً جداً بالشجرة ، لأنه إذا حذفنا جذر شجرة حصلنا على غابة ، على سبيل المثال ، حذف جذر أي شجرة ثنائية ينتج غابة مكونة من شجرتين .

7-9-1 العمليات على الغابة Forest operations

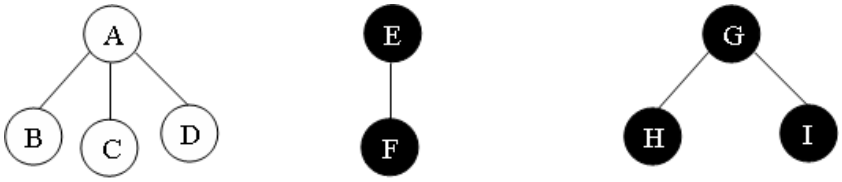
تحويل الغابة إلى شجرة ثنائية Transforming a Forest Into a Binary Tree

ليكن لدينا غابة مكونة من ثلاث أشجار الموضحة في الشكل (6-20). لتحويل هذه الغابة إلى شجرة ثنائية تتبع الخطوات التالية :

1. نمثل كل شجرة من الأشجار بطريقة left child-right sibling للحصول على أشجار ثنائية .
2. نربط تلك الأشجار بعضها ببعض من خلال حقل ربط الأخ (sibling) لعقدة الجذر.
3. نطبق التحويل التالي إلى الغابة في الشكل(7−21) نحصل على الشجرة الثنائية المعطاة في الشكل 7−22 .

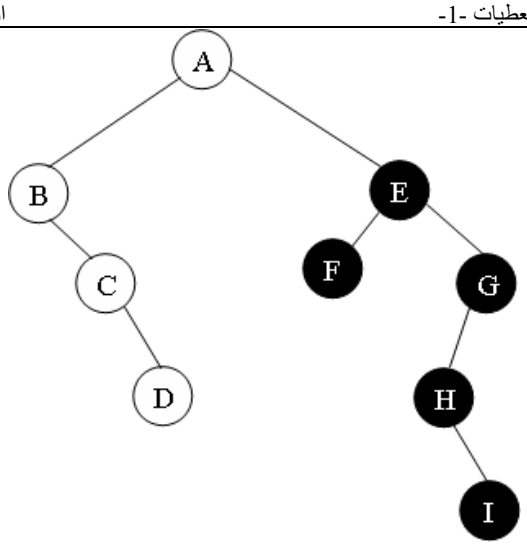
يعرف هذا التحويل بالصيغة التالية :

إذا كان $T_1, T_2, ..., T_n$ غابة مكونة من n شجرة ، عندئذ يرمز للشجرة الثنائية الموافقة لهذه الغابة بـــ $B(T_1, T_2, ..., T_n)$:



الشكل (7− 21) : غابة من ثلاث أشجار

- تكون $B(T_1, T_2, ..., T_n)$ خالية ، إذا كان n = 0
- عقدة الجذر لـــــلشجرة $B(T_1, T_2, ..., T_n)$ هي جذر الشجرة T_1 ؛ شجرتها الفرعية اليسارية $B(T_{11}, T_{12}, ..., T_{1m})$ ، حيث $T_{11}, T_{12}, ..., T_{1m}$ أشجار فرعية من عقدة الجذر للشجرة T_1 ؛ و شجرتها الفرعية اليمينية $B(T_2, T_3, ..., T_n)$.



الشكل (7-22): الشجرة الثنائية الناتجة من الغابة

7-9-2 أساليب التنقل عبر الغابة Forest Traversals

أساليب التنقل Pre-order, In-order, Post-order المطبقة عبر شجرة ثنائية T ناتجة من غابة ما F ، تطبق أيضاً على الغابة F .

التنقل بأسلوب Pre-order

في هذا الأسلوب ، يتم زيارة العقد في الغابة F بأسلوب pre-order المطبق عبر الأشجار الثنائية . الخطوات التالية توضح عملية التنقل بهذا الأسلوب عبر غابة :

1. If F is empty , then return
2. Visit the root of the first tree of F
3. Traverse the sub-trees of the first tree in tree preorder
4. Traverse the remaining trees of F in preorder

التنقل بأسلوب In-order

في هذا الأسلوب ، يتم زيارة العقد في الغابة F بأسلوب in-order المطبق عبر الأشجار الثنائية. الخطوات التالية توضح عملية التنقل بهذا الأسلوب عبر غابة :

1) If F is empty , then return
2) Traverse the sub-trees of the first tree in tree inorder
3) Visit the root of the first tree of F
4) Traverse the remaining trees of F in inorder

التنقل بأسلوب Post-order

لا يوجد تشابه في أسلوب هذا التنقل عبر الغابة مع أسلوب هذا التنقل عبر الشجرة الثنائية الناتجة عن غابة . الخطوات التالية توضح عملية التنقل بهذا الأسلوب عبر غابة :

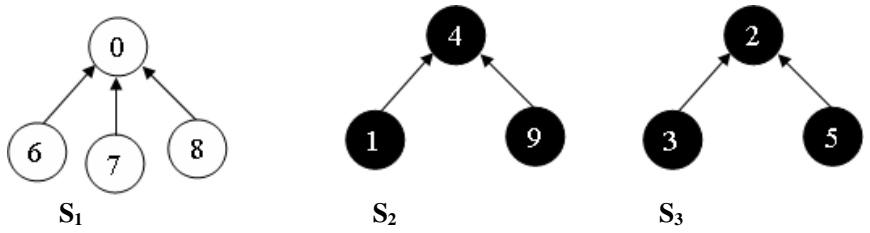
1- If F is empty , then return
2- Traverse the sub-trees of the first tree of F in tree post-order
3- Traverse the remaining trees of F in post-order
4- Visit the root of the first tree of F

7-10 تمثيل المجموعة Set Representation

في هذا المقطع ، ندرس استخدام الأشجار في تمثيل المجموعات . لنفترض أن عناصر المجموعات عبارة عن أعداد 0, 1, 2, ….., n-1 التي يمكن أن تمثل أدلة في جدول رموز (symbol table) الذي يخزن الأسماء الحقيقية للعناصر . و كذلك نفترض أن المجموعات الممثلة باستخدام الأشجار منفصلة . على سبيل المثال ، إذا كان لدينا 10 عناصر مرقمة من 0 إلى 9 ، نجزئ تلك العناصر إلى ثلاث مجموعات منفصلة :

$S_1 = \{0, 6, 7, 8\}$, $S_2 = \{1, 4, 9\}$, $S_3 = \{2, 3, 5\}$

يوضح الشكل (7-23) إحدى الطرق الممكنة لتمثيل هذه المجموعات ، حيث نلاحظ أن روابط العقد من الأبناء إلى الآباء .

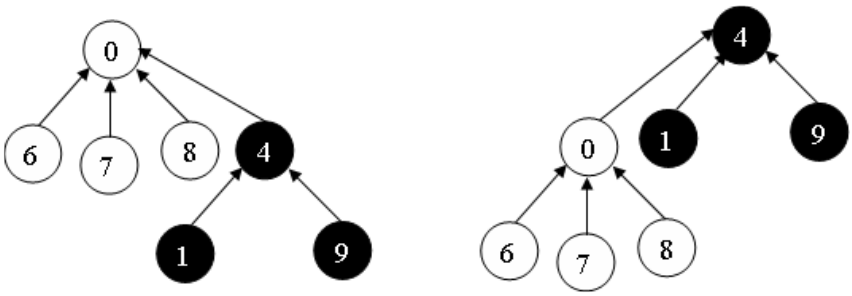


الشكل (7-23) غابة تمثل مجموعات

7-10-1 بعض العمليات على المجموعات

7-10-1-1 الاجتماع Union

يتم الحصول على الاجتماع $S_1 \cup S_2$ بجعل إحدى الشجرتين في الشكل (7-23) شجرة فرعية من الأخرى . $S_1 \cup S_2$ يمكن أن يمثل واحدة من الشجرتين في الشكل (7-24)



$S_1 \cup S_2$ = $S_2 \cup S_1$

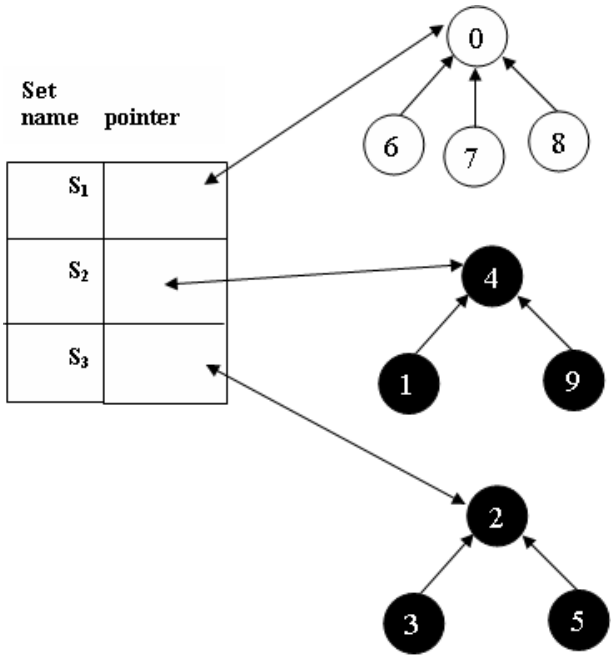
الشكل(7-24): تمثيل للمجموعة $S_1 \cup S_2$

7-10-1-2 تنفيذ دالة Union

لتنفيذ عملية الاجتماع هذه ، نجعل لكل جذر شجرة مؤشراً إلى اسم المجموعة الممثلة بهذه الشجرة كما هو موضح في الشكل (7-25). لتبسيط دراسة خوارزميات الإيجاد (Find) و الاجتماع ، نجعل أسماء المجموعات جذور الأشجار التي تمثلها . فمثلاً يشار إلى اسم المجموعة S_1 بـــ 0 . يكون الانتقال إلى اسم المجموعة سهلاً ، و ذلك بافتراض جدول []name ، يتضمن أسماء المجموعات . إذا كان العنصر i في الشجرة ذات الجذر j يملك مؤشر إلى المدخل k في جدول الأسماء ، عندئذ يكون اسم المجموعة name[k] .

بما أن العقد في الأشجار مرقمة من 0 إلى n-1 و بالتالي يمكننا استخدام هذه الأرقام كأدلة – أي أن كل عقدة تحتاج إلى حقل واحد فقط ، و الذي يمثل دليل والد هذه العدة . و يتم التعبير عن ذلك ببنى المعطيات : int parent [MAX-ELEMENTS]; حيث MAX-ELEMENTS يمثل العدد الأعظمي للعناصر كما نفترض أن :

parent[root] = -1. الشكل (7–26) يوضح هذا التمثيل للمجموعات S_1, S_2, S_3 .



الشكل(7–25): تمثيل البيانات للمجموعات S_1, S_2, S_3

i	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
parent	-1	4	-1	2	-1	2	0	0	0	4

الشكل(7–26) : تمثيل المجموعات S_1, S_2, S_3 باستخدام المتجهة

7-10-1-3 تنفيذ دالة Find()

يتم تنفيذ الدالة find(i) بتتبع الأدلة التي تبدأ من الدليل i و نستمر حتى نصل إلى دليل والد سالب . مثلا ، find(5) ، نبدأ من الدليل 5 و نتحرك إلى والد 5 و الذي هو 2 . و بما أن للعقدة 2 دليلاً سالباً و بذلك نكون قد وصلنا إلى الجذر . أما في الدالة union(i , j) ، فيتم تمرير جذور شجرتين i , j . كما نصطلح أن جذر الشجرة الأولى يصبح ابناً لجذر الشجرة الثانية ، أي أن العبارة parent[i] =j تنجز عملية الاجتماع . الدوال التالية تنجز عمليتي find , union .

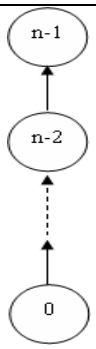
```
int find1 (int i)
    {
    for( ; parent [i] >=0 ; i = parent[i])
          ;
        return i ;
        }


void  union1 (int i, int j)
     {
   parent[i] = j;
     }
```

تحليل الدوال union1(), find1()

على الرغم من سهولة تنفيذ دالتي find1, union1 ، إلا أن مميزات إنجازهما ليست جيدة جداً . على سبيل المثال ، إذا بدأنا بـــ p عنصراً في كل مجموعة $S_i = \{i\},\ 0 \le i < p$ عندئذ يكون الشكل الابتدائي غابة مكونة من p عقدة و $parent[i] = -1,\ 0 \le i < p$. متتالية المؤثرات union-find التالية:

```
union(0,1) , find(0)
union(1,2) , find(0)
        .
        .
union(n-2,n-1) , find(0)
```

تنتج شجرة خطية موضحة في الشكل(7-27) التالي :



الشكل (7-27) : شجرة خطية

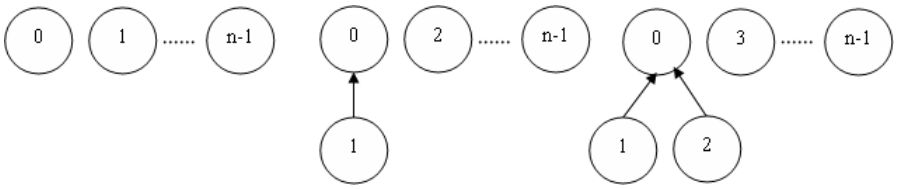
نلاحظ أن زمن حساب دالة ()union ثابت و بالتالي نعالج1-n استدعاء لهذه الدالة بزمن O(n). إضافة لذلك ، من أجل كل find نتبع سلسلة من روابط الوالد (parent links) من 0 إلى الجذر . إذا كان العنصر في المستوي i فإن الزمن المطلوب لإيجاد جذره O(i) . و بالتالي يكون الزمن الكلي لمعالجة n-1 finds :

$$\sum_{i=2}^{n} i = O(n)$$

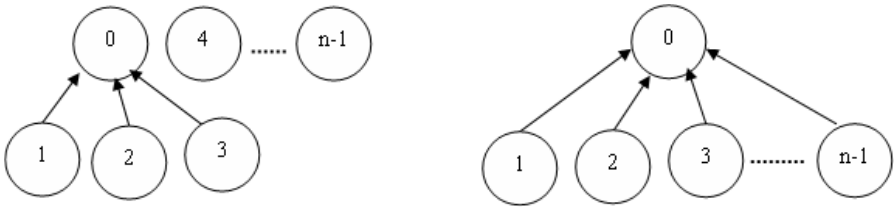
لتجنب إنشاء شجرة خطية وإيجاد تنفيذ أفضل لمؤثرات union, find نتبع قاعدة الوزن (weighting rule) للدالة union (i , j) والتي تعرف بالشكل التالي :

تعريف : قاعدة الوزن للدالة union (i , j) : إذا كان عدد العقد في الشجرة i أقل من عدد العقد في الشجرة j عندئذ اجعل العقدة j والداً للعقدة i ؛ و غير ذلك اجعل i والداً للعقدة j .

بتطبيق هذه القاعدة على متتالية المؤثرات union-find الموصوفة أعلاه، نحصل على الأشجار الموضحة بالشكل (7-28):



initial **union (0,1)** **union(0, 2)**
0 = find(0) **0 = find(0)**



union(0, 3) **union(o, n-1)**
0 = find(0)

الشكل(7-28): الأشجار المبنية بقاعدة الوزن

لتنفيذ قاعدة الوزن ، نحتاج لمعرفة عدد العقد في كل شجرة ، و يتم ذلك باستخدام حقل count لجذر كل شجرة ، أي إذا كان i عقدة جذر فإن count[i] يساوي عدد العقد في الشجرة . بما أن حقل الوالد لكل عقد جذر يساوي عدداً سالباً ، فيمكننا جعل القيمة parent[i] تساوي -count[i] . و بالتالي باستخدام قاعدة الوزن تعدل الدالة union() لتصبح بالشكل التالي :

```
  void union2( int i, int j)
        {
/* union the sets with roots i and j , i !=j , using the weighting rule .
parent [i] = -count[i]  and parent [j] = -count[j] */
          int temp = parent[i] + parent[j];
          if (parent [i] > parent[j]) {
               parent [i] = j;    /* make j the new root */
              parent [j] = temp;
```

```
        }
    else
      {
parent [j] = i;   /* make i the new root */
parent [i] = temp;
        }
      }
```

نظرية : لتكن T شجرة مكونة من n عقدة تم إنشاؤها باستخدام الدالة union2 . عندئذ $level(T) \le \lfloor \log_2 n \rfloor$.

البرهان: لنستخدم الاستقراء الرياضي على n في إثبات المتراجحة $level(T) \le \lfloor \log_2 n \rfloor$.

المتراجحة واضحة و صحيحة من أجل n = 1 .

نفرض أن المتراجحة صحيحة على كل الأشجار المكونة من i عقدة و $i \le n-1$ و لنبرهن على صحتها من أجل i = n .

لتكن T شجرة مكونة من n عقدة تم إنشاؤها باستخدام الدالة union2 . لتكن union(k, j) آخر عملية اجتماع منجزة ، ليكن m عدد العقد في الشجرة j ، ويكون عدد العقد في الشجرة jيساوي n-m . يمكننا افتراض أن $1 \le m \le \frac{n}{2}$. عندئذ المستوي الأعظمي للشجرة T إما هو مستوي الشجرة k أي :

$$level(T) \le \lfloor \log_2 (n-m) \rfloor \le \lfloor \log_2 n \rfloor$$

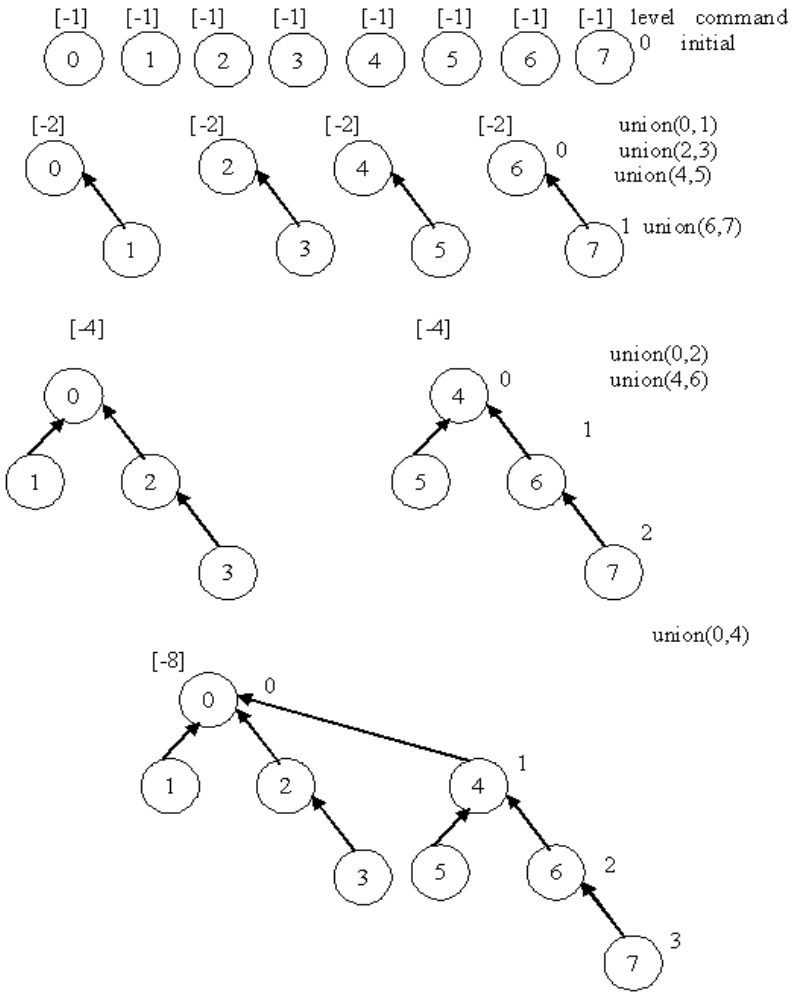
أو أكثر بواحد من مستوي الشجرة j . و بالتالي يكون :

$$level\ (T) \le \lfloor \log_2 m \rfloor + 1 \le \left\lfloor \log_2 \frac{n}{2} \right\rfloor + 1 \le \lfloor \log_2 n \rfloor$$

مثال: الشكل(7–29) نتيجة لتطبيق الدالة union2 على متتالية من الاجتماعات تبدأ من الشكل الابتدائي :

$$parent\ [i] = -count\ [i] = -1,\ 0 \le i \le n = 2^3\ .$$

union(0,1) union(2,3) union(4,5) union(6,7)
union(0,2) union(4,6) union(0,4)



الشكل(7–29): الأشجار نتيجة لتطبيق الدالة union2

الزمن اللازم لعملية find في شجرة مكونة من n عنصراً يكون $O(\log_2 n)$. أما الزمن المستهلك لمعالجة متتالية مكونة من union (n-1) و m find يكون $O(n+m\log_2 n)$.

يوجد تحسين إضافي لخوارزمية union -find و ذلك بإضافة قاعدة الطي (Collapsing rule) لعملية find .

تعريف (قاعدة الطي): إذا كانت العقدة j واقعة على المسار من العقدة i إلى عقدة الجذر عندئذ نجعل العقدة j ابناً لعقدة الجذر .
باستخدام قاعدة الطي يمكننا تعديل الدالة find() بالنسخة التالية :

```
int find2 (int i)
   {
/* Find the root of tree containing element i . Use the collapsing rule to
collapse all nodes from i to root */
   int   root, trail, lead;
   for(root = i; parent[root] >=0; root = parent[root])
        ;
   for (trail = i; trail !=root; trail = lead) {
        lead = parent [trail];
        parent [trail] = root;
          }
   return root;
  }
```

تحليل خوارزمية union-find

لتكن الدالة $\alpha(n, m)$ ذات نمو بطيء جداً و التي تعتبر معكوساً لدالة Ackermann $A(p, q)$ ذات النمو السريع جداً ، و تعرف بالصيغة الآتية :

أما دالة Ackermann . $\alpha(n,m) = \min\{z \geq 1 \mid A(z,4\left\lceil \frac{m}{n} \right\rceil) > \log_2 n\}$

المستخدمة هي :

$$A(p,q) = \begin{cases} 2q & p = 0 \\ 0 & q = 0 \ \ and \ \ p \geq 1 \\ 0 & p \geq 1 \ \ and \ \ p = 1 \\ A(p-1, q-1)) & p \geq 1 \ \ and \ \ q \geq 2 \end{cases}$$

يمكننا إثبات أن :

$$(1) \quad A(3,4) = 2^{2^{\cdot^{\cdot^{2}}}} \} \ 56 \ ,536 \quad twos$$

$$(2) \quad A(p, q+1) > A(p,q)$$

$$(3) \quad A(p+1, q) \geq A(p,q)$$

إذا افترضنا أن $m \neq 0$ ، عندئذ من (2) و (3) و تعريف $\alpha(n,m)$ يقتضي أن $\alpha(n,m) \leq 3$ و ذلك من أجل $\log_2 n < A(3,4)$. من (1) نجد أن قيمة $A(3,4)$ كبيرة جداً . من النظرية 2 ، سيكون عدد unions يساوي n ، و إذا فرضنا أن $\log_2 n < A(3,4)$ عندئذ يكون $\alpha(n,m) \leq 3$.

نظرية 2[Tarjan]: ليكن T(m, n) الزمن الأعظمي المطلوب لمعالجة متتالية متداخلة من finds $(m \geq n)$ و (n-1) unions عندئذ :

$$k_1 m\, \alpha(m,n) \leq T(m,n) \leq k_2 m\, \alpha(m,n)$$

حيث k_1 و k_2 ثوابت موجبة .

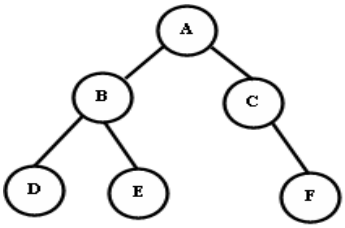
على الرغم من أن الدالة $\alpha(n,m)$ ذات نمو بطيء جداً ، فإن التعقيد الزمني لخوارزمية union-find ليس خطيا في m (عدد finds) .

7-11 تمارين الفصل السابع

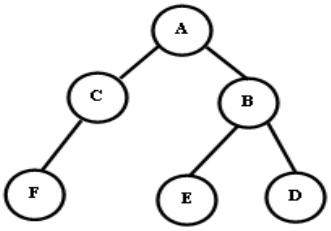
تمرين 1: بفرض لديك شجرة ثنائية غير فارغة مكونة من n عقدة و المطلوب :

1. اكتب الدوال الإجرائية التالية في هذه الشجرة الثنائية :
   - حساب عدد العقد الداخلية فيها .
   - حساب عدد الأوراق .
   - حساب ارتفاع الشجرة .
   - حساب عدد العقد الداخلية فيها .
   - التحقق من كون الشجرة الثنائية خطية .
   - التحقق من كون الشجرة الثنائية ممتلئة بالعقد
   - التحقق من كون الشجرة الثنائية كاملة .
2. أوجد التعقيد الزمني للدوال الإجرائية السابقة .

تمرين 2: نقول عن شجرتين إنهما متشابهتان بالتعاكس (Mirror Similar) إذا كانتا فارغتين معاً ، أو إذا كانتا غير فارغتين و الشجرة الجزئية اليسارية من كلتيهما متشابهة بالتعاكس مع الشجرة الجزئية اليمينية من كلتيهما (انظر الرسم التوضيحي) .



(a)



(b)

اكتب خوارزمية لتتحقق من شجرتين ثنائيتين ، متشابهتين بالتعاكس .

تمرين 3: بفرض لدينا التعبير الحسابي التالي :

$$(a+b)*(c-d)/e$$

حيث a, b, c, d, e أرقام عددية صحيحة. والمطلوب:

1. أنشئ شجرة ثنائية للتعبير الحسابي المعطى .
2. قدم بنى المعطيات اللازمة لتعريف عقدة فيها .
3. اكتب دالة إجرائية تقيم التعبير الحسابي السابق مستخدما أسلوب التنقل المناسب عبر الأشجار الثنائية .

تمرين 4: نسمي شجرة ثنائية مرتبة ، شجرة ثنائية يوجد بين عقدها علاقة ترتيب . نفترض أن كل عقدة تحوي عددا صحيحا ، و علاقة الترتيب بين العقد هي : كل ابن يساري لعقدة هو أصغر تماما من العقدة الأب ، كل ابن يميني هو أكبر تماما من العقدة الأب .

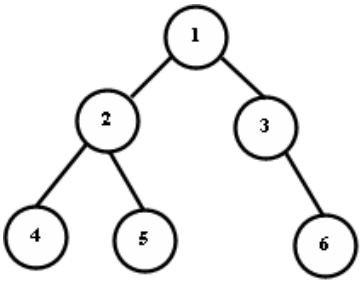
1. اكتب الدوال الإجرائية التالية في هذه الشجرة المرتبة :
   - إضافة عقدة في مكانها الصحيح .
   - حذف عقدة من مكانها الصحيح .
   - البحث عن عقدة ما .
2. أوجد التعقيد الزمني للدوال الإجرائية السابقة .

تمرين 5: بين أن عدد الأشجار الثنائية التي ارتفاعها $n \geq$ تعطى بالعلاقة العودية:

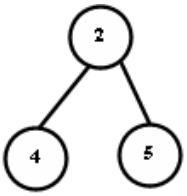
$$T(n)=\begin{cases}1 & if\ n=1\\ (T(n-1))^2+1 & if\ n>1\end{cases}$$

ثم بين أن $T(n)=\lfloor c^{2^n} \rfloor$ حيث c عدد ثابت ، ثم أوجد هذا الثابت .

تمرين 6: نقول عن شجرة T_1 إنها جزئية من شجرة ثنائية أخرى T_2 إذا كانت T_2 تحوي كامل T_1 (انظر الرسم التوضيحي) .



(T_2)



(T_1)

1- اكتب دالة إجرائية تتحقق من كون الشجرة T_1 جزئية من شجرة أخرى T_2 .

2- اكتب دالة إجرائية تقوم بحساب تكرار وجود الشجرة الجزئية T_1 في الشجرة T_2 .

تمرين 7: أثبت أن عدد الأشجار الثنائية المختلفة التي تملك ($n \geq 1$) عقدة تعطى بالعلاقة العودية : $T(n)=\sum_{i=0}^{n-1} T(i)*T(n-i-1),\ n \geq 1, T(0)=1$ ، ثم حل هذه العلاقة .

تمرين 8: قدم خوارزمية تقوم بإنشاء شجرة ثنائية تمثل تعبيراً حسابياً (أو منطقياً) ثم أوجد التعقيد الزمني لها .

تمرين 9: اكتب دالة إجرائية insert-left تقوم بإضافة عقد جديدة ، child ، كابن يساري لعقدة parent في شجرة ثنائية خيطية.

تمرين 10 : لتكن شجرة ثنائية خيطية ، tree ، و المطلوب :

1. اكتب دالة تقوم بالتنقل عبر الشجرة tree بأسلوب post-order و أوجد التعقيد الزمني لها .
2. اكتب دالة تقوم بالتنقل عبر الشجرة tree بأسلوب pre-order و أوجد التعقيد الزمني لها .

تمرين 11 : لتكن T شجرة بحث ثنائية و المطلوب :

1. اكتب النسخة العودية للدالة التكرارية insert_node() التي تضيف عقدة إلى T ثم قارنها مع الدالة insert_node() .
2. اكتب دالتين إجرائيتين تكرارية و عودية لحذف عقدة من الشجرة T . ثم اوجد التعقيد الزمني لهما .
3. بفرض أن الشجرة T ممثلة بشكل شبيه لـــ شجرة البحث الثنائية الخيطية . اكتب الدوال الإجرائية: search, insert, delete .

تمرين 12: بفرض شجرة T ممثلة باستخدام الابن اليساري– الأخ اليميني . اكتب دوال إجرائية تكرارية تنفذ أساليب التنقل :pre-order, post-order , in-order عبر الشجرة T.