جَامعة المَنارة
MANARA UNIVERSITY

كلية الهندسة قسم المعلوماتية

بنى معطيات 1

Data Structure 1

ا. د. علي عمران سليمان

محاضرات الأسبوع الثا منه

الأشجار1

Tree 1

الفصل الثاني 2023-2024

محتوى الفصل

الأشجار 1 + 2	Trees 1 +2
1- مقدمة	
2- الأشجار العامة.	2- General Trees.
3- خوارزميات التجول عبر الشجرة.	
4- الأشجار الثنائية.	
5- تعريف شجرة البحث الثنائية كقاموس بيانات.	
6- تحقيق خوارزميات البحث.	
7- حساب أداء خوارزميات البحث.	
8 - تحقيق طرائق التبديل على شجرة بحث ثنائية.	
9 - تحقيق شجرة البحث الثنائية بلغة C++	

References

- Deitel & Deitel, Java How to Program, Pearson; 10th Ed(2015)

- د.علي سليمان، بنى معطيات بلغة JAVA، بنى معطيات بلغة C++، بنى معطيات بلغة Pascal جامعة تشرين 2014، 2007، 1998

1- تعريف الشجرة وخصائصها 1

- يقول خبراء الإنتاجية إن التطور الحاسم في لغات البرمجة تأتى من خلال التفكير اللاخطي nonlinearly.
- تعتبر الأشجار trees من أهم بنى المعطيات اللاخطية في مجال الحوسبة،
- وهى أحد الفتوحات الكبيرة في مجال تنظيم البيانات، فهي تتيح لنا تحقيق عدة خوارزميات ذات أداء أسرع بكثير من حالة استخدام بنى معطيات خطية، مثل اللائحة.
- تقدم الأشجار أيضاً تنظيماً طبيعياً للبيانات، فأصبحت بالتالي بنى موجودة في كل مكان في نظام الملفات، واجهة المستخدم الرسومية، قواعد البيانات، مو اقع الويب، وباقي الأنظمة الحاسوبية.
- عندما نقول إن الأشجار لاخطية، فإننا نشير إلى علاقة تنظيمية أكثر غنى من علاقات " قبل before" و "بعد after" بين الأغراض في بنى المتتاليات.
- إن العلاقات في الأشجار متر اتبة هرمياً hierarchical، حيث تكون بعض الأغراض "فوق above" وبعضها "تحت below".
- إن المصطلحات الرئيسية لبنية المعطيات الشجرة، مشتقة من شجرة العائلة، مع استخدام مصطلحات مثل "أب parent" و "إبن child"، "سلف ancestor" و "سليل descendent" كأكثر المصطلحات شيوعاً لوصف العلاقات.

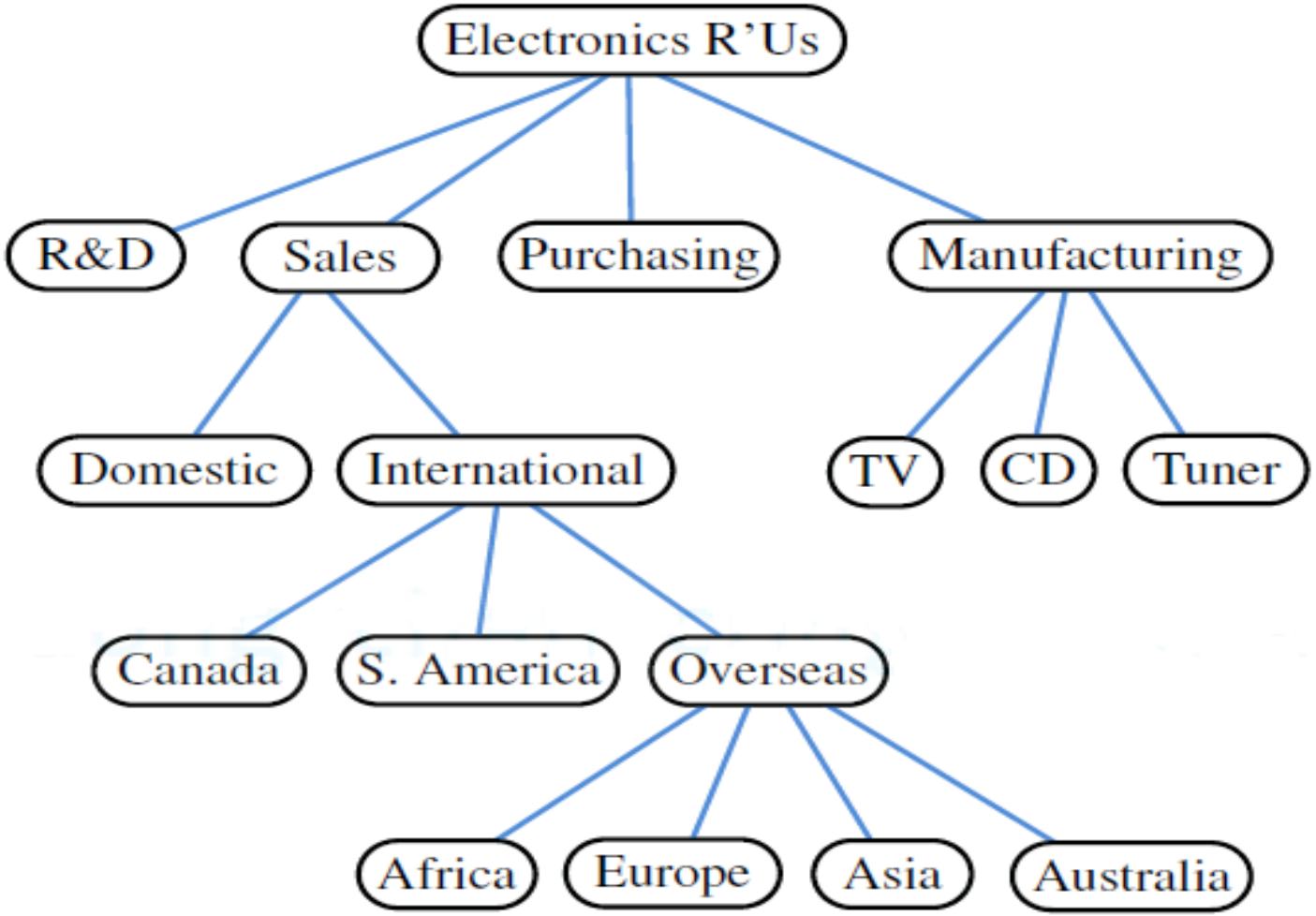
1- تعريف الشجرة وخصائصها 2

- إن الشجرة treeهي نمط بيانات مجرد يخزن العناصر هرمياً. مع استثناء العنصر الأعلى، فإن كل عنصر له عنصر أب parent، وقد يكون له عنصر إبن child أو أكثر.

- يتم تمثيل الشجرة عادة من خلال وضع العناصر في أشكال مستطيلة أو بيضوية، ورسم وصلات بين الأب والأبناء بواسطة خطوط مستقيمة (كما يبين الشكل 5-2)، نسمي العنصر الأعلى في الشجرة باسم جذر الشجرة root، ولكنه يرسم كأعلى عنصر، وتكون باقي العناصر متصلة إلى الأسفل.

- يبين الشكل 5-2 شجرة ذات 17 عقدة تمثل شركة وهمية، حيث إن الجذر هو Electronics R'Us و أبناء الجذر هم R&D، sales، Purchasing، Manufacturing، .... إلخ. Back

1- تعريف الشجرة وخصائصها 3



الشكل 5-2- شجرة تمثل شركة افتراضية. العمق الارتفاع NLR

2- التعريف الرسمي للشجرة

Formal Tree Definition

رسمياً، نعرف الشجرة T بأنها مجموعة من العقد تخزن عناصر بحيث إن جميع العقد تملك العلاقة "أب-إبن parent-child" التي تحقق الصفات التالية:

- إذا كانت T غير فارغة، فإنها تحوي عقدة خاصة تدعى جذر الشجرة T أو root، وهذه العقدة ليس لها أب.
- كل عقدة v في T (ماعدا الجذر)، لها عقدة أب parent واحدة w، وكل عقدة يكون أبوها w، تدعى "إبن child" w.

لاحظ أنه وفقاً لتعريفنا، يمكن للشجرة أن تكون فارغة، وهذا يعني أنها لاتحوي أي عقدة.

يتيح لنا هذا الاصطلاح أن نعرف الشجرة عودياً بحيث إن الشجرة T هي إما فارغة أو تحوي عقدة r تسمى جذر T و مجموعة (قد تكون فارغة) من الأشجار التي تكون جذورها هي أبناء لـ r.

ندعو عقدتين أبناء لنفس الأب بالاسم بأنهن أخوة siblings. ونقول عن عقدة v أنها خارجية external إذا لم يكن لها أبناء، وندعو عقدة بأنها داخلية internal إذا كان لها إبن أو أكثر. تدعى العقد الخارجية أحياناً باسم الأوراق leaves.

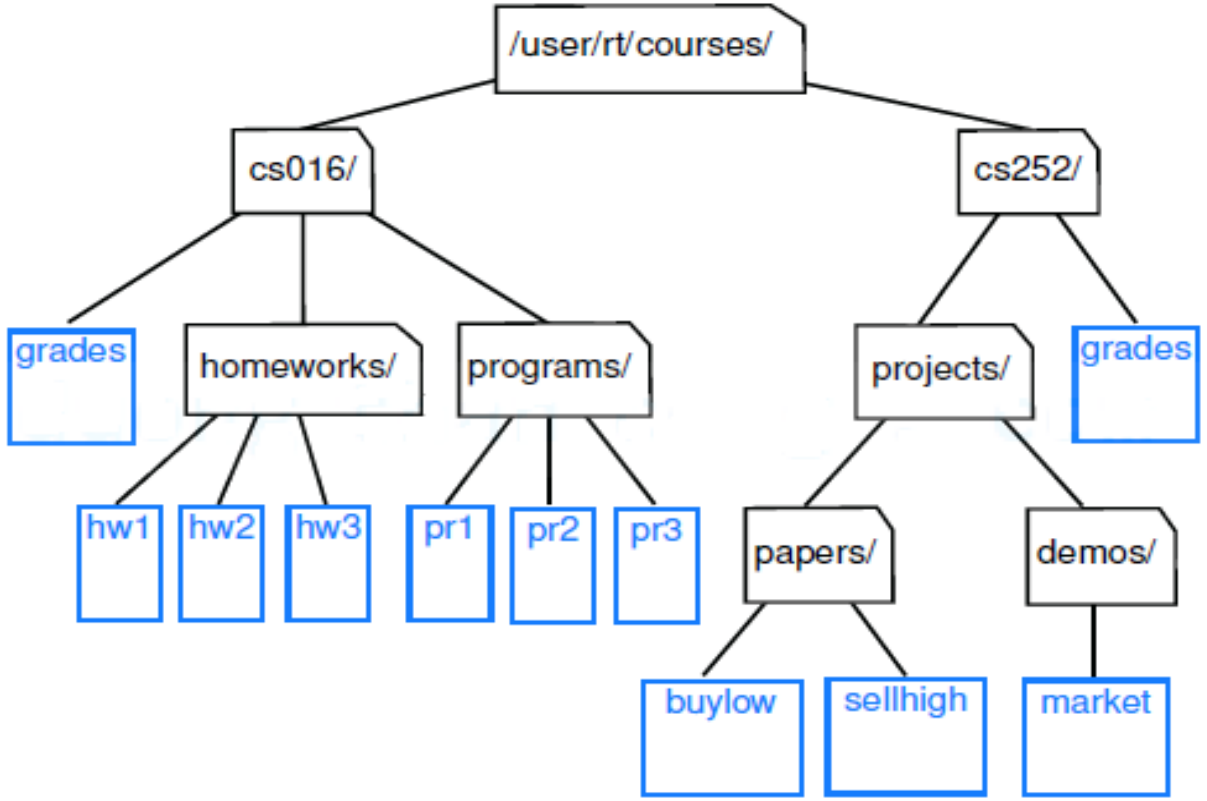
مثال 1- تنظم الملفات في أغلب نظم التشغيل هرمياً في مجلدات متداخلة، حيث يتم عرضها للمستخدم على شكل شجرة (الشكل 5-3).

وبتخصيص أكبر، تربط العقد الداخلية للشجرة مع مجلدات، وتربط العقد الخارجية مع ملفات. يدعى جذر الشجرة في نظم التشغيل Linux أو Unix بالاسم المجلد الجذر root directory ويتم ترميزه بالرمز /.

Formal Tree Definition

2- التعريف الرسمي للشجرة

تكون عقدة u سلف ancestor لعقدة v، إذا كانت u=v أو كانت u سلف لأب العقدة v، وباتجاه معاكس، نقول أن عقدة v هي سليلة descendent لعقدة u، إذا كانت u سلف v. cs252/ is an ancestor of papers/, and pr3 is a descendant of cs016/



على سبيل المثال، في الشكل 5-3، المجاور

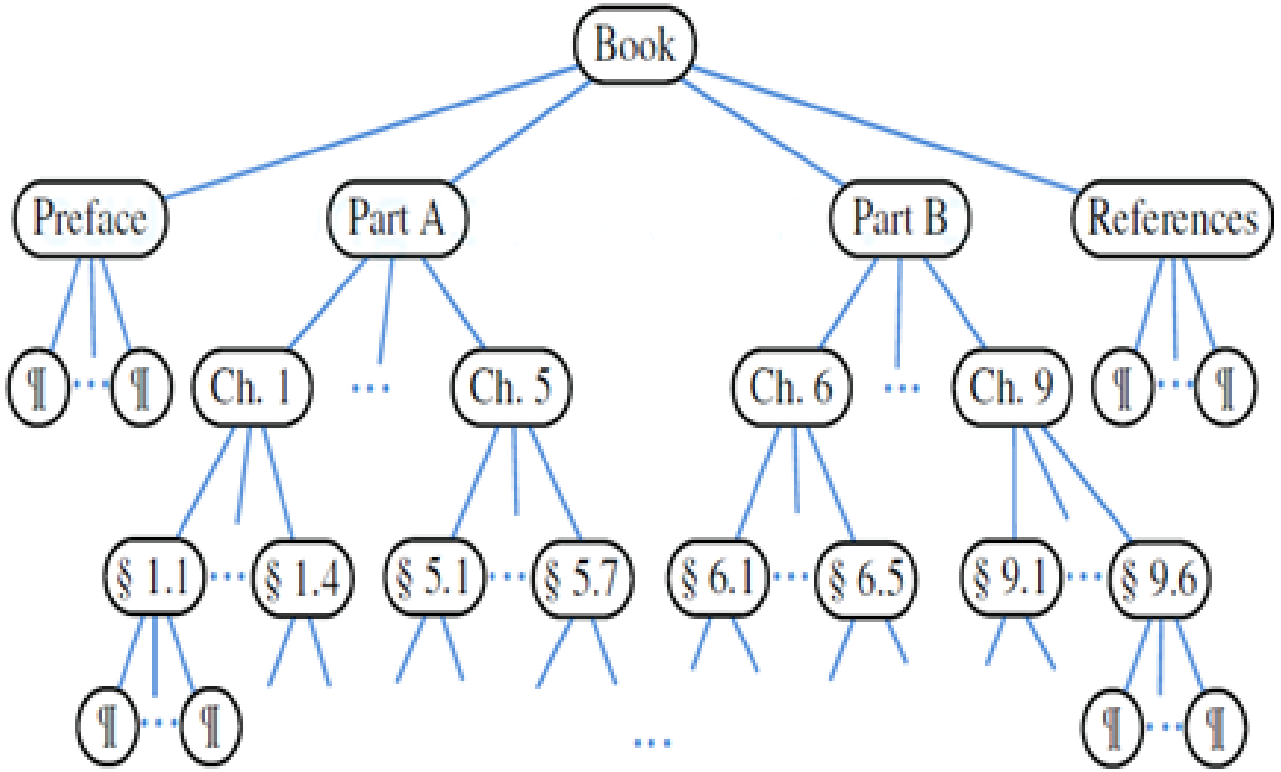
- إن /cs252 هي سلف /papers،
- pr3 هي سليلة /cs016.
- الشجرة الجزئية subtree من الشجرة T ذات الجذر v هي شجرة تحوي v وجميع نسل v في T.
- على سبيل المثال، الشجرة الجزئية ذات الجذر /cs016 تحوي العقد /cs016، /grades، /homeworks، /programs، hw1، hw2، hw3، pr1، pr2 و pr3.

Edges and Paths in Trees

الحدود والمسارات في الأشجار

- حد edge الشجرة T هو الوصلة مابين زوج من العقد (u,v) بحيث إن u هي أب لـ v، أو بالعكس.

- أما مسار path الشجرة T هو تتالٍ من العقد مع حدودها على سبيل المثال، تحوي الشجرة في الشكل 3-5 المسار:
- (cs252, projects/, demos/, market)
- مثال 3- إن مكونات مستند هيكلي، مثل كتاب على سبيل المثال، منظمة هرمياً على شكل شجرة تكون عقدها الداخلية هي الأجزاء، الفصول، والفقرات. وتكون عقدها الخارجية هي المقاطع، الجداول، الأشكال ... وهكذا.



الشكل 5-4- شجرة مرتبة تعبرعن كتاب.

الشجرة كنمط بيانات مجرد

تخزن الشجرة كنمط بيانات مجرد عناصرها في مو اقع positions.

المو اقع positions في شجرة هي عقدها nodes، حيث تحقق المو اقع المجاورة العلاقات أب-إبن parent-child والتي تعرف شجرة صالحة.

وبالتالي، فإن المصطلحان موقع position وعقدة node مترادفان بالنسبة للأشجار.

وكما في موقع اللائحة، فإن الغرض position بالنسبة للشجرة يدعم الطريقة التالية:

- element(): يعيد الغرض المخزن في ذلك الموقع.

إن قوة مو اقع العقد في الشجرة تتأتى من طر ائق الوصول accessory methods للشجرة والتي تعيد وتقبل المو اقع، كما في الطر ائق التالية:

- تعيد جذر الشجرة، يحدث خطأ إذا كانت الشجرة فارغة.
- parent(v): تعيد أب v، يحدث خطأ إذا كانت v هي الجذر.
- children(v): تعيد مجموعة تتضمن أبناء العقدة v.

إذا كانت الشجرة T مرتبة، فإن المجموعة children(v) تخزن أبناء v بشكل مرتب. وإذا كانت v عقدة خارجية، عندئذ children(v) مجموعة فارغة.

بالإضافة إلى طرائق الوصول الأساسية السابقة، نقوم أيضاً بتضمين طرائق الاستعلام query methods التالية:

- isInternal(v): تختبر فيما إذا كانت العقدة v داخلية.
- isExternal(v): تختبر فيما إذا كانت العقدة v خارجية.
- isRoot(v): تختبر فيما إذا كانت العقدة v جذراً.

إن هذه الطرائق تجعل البرمجة باستخدام الأشجار أسهل وأكثر قابلية للقراءة، وذلك لأننا نستخدمها في الأوامر الشرطية if والحلقات while.

هناك أيضاً عدد من الطرائق العامة generic methods يجب أن تدعمها الأشجار رغم أنها ليست بالضرورة مرتبطة بالبنية الشجرية، من بينها:

- size(): تعيد عدد العقد في الشجرة.
- isEmpty(): تختبر فيما إذا كانت الشجرة فارغة.
- iterator(): تعيد المكرر iterator لجميع العناصر المخزنة في عقد الشجرة.
- positions(): تعيد مجموعة قابلة للتجول التكراري من جميع العقد في الشجرة.
- replace(v,e): تعيد العنصر المخزن في العقدة v وتستبدله بـ e.

ALinked Structure for General Trees

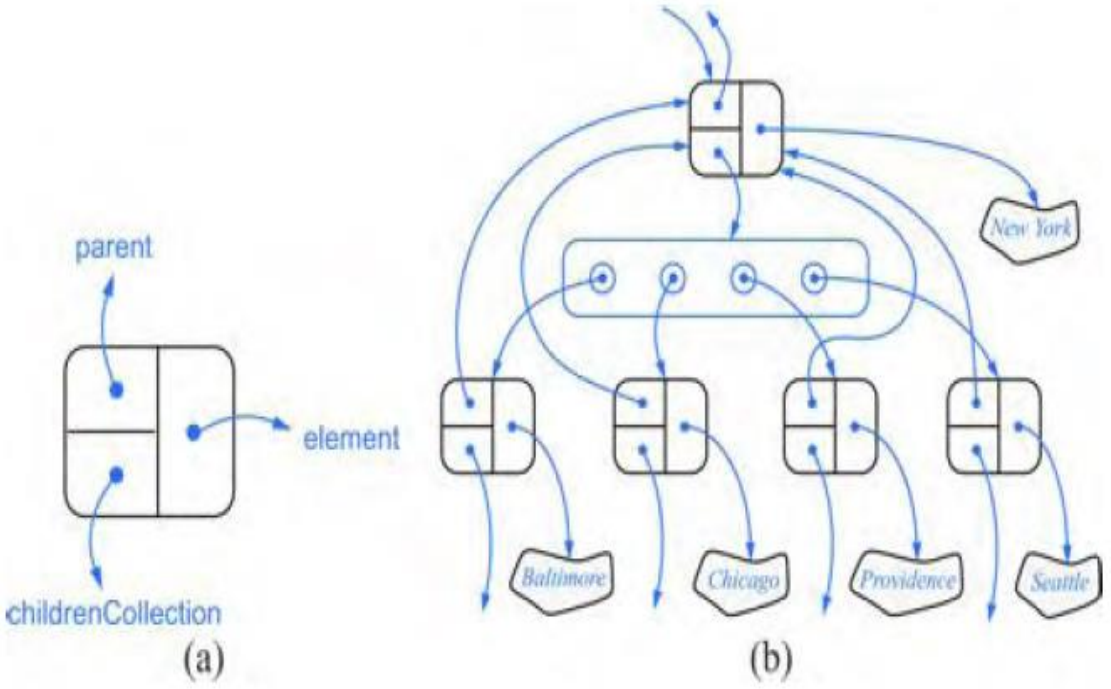
البنية المترابطة للأشجار العامة

من الطرق الطبيعية لتحقيق شجرة T هي استخدام البنية المترابطة، حيث نمثل كل عقدة v من T من خلال الغرض position (الشكل 5-5-a) بحيث يحوي الحقول التالية:

1- مرجع إلى العنصر المخزن في v. 2- ارتباط link إلى أب العقدة v. 3- مجموعة من نوع ما (لائحة أو مصفوفة) لتخزين روابط إلى أبناء v.

إذا كانت v هي جذر T، عندئذ فإن حقل الأب لـ v هو null. كما أننا نخزن مرجعاً إلى جذر T وأرقام العقد في T في متحولات داخلية.

يمكن تمثيل هذه البنية تخطيطياً في الشكل 5-5-b.



الشكل 5-5- البنية المترابطة لشجرة عامة.

Operation	Time
size , isEmpty	O(1)
positions	O(n)
replace	O(1)
root, parent	O(1)
isInternal, isExternal, isRoot	O(1))

جدول يمثل درجة التعقيد للعمليات المدرجه.

خوارزميات التجول عبر الشجرة 1

Tree Traversal Algorithms

خوارزميات إنجاز عمليات التجول على شجرة من خلال الوصول إليها من خلال طرائق الشجرة كنمط بيانات مجرد.

العمق والارتفاع Depth and Height: لتكن v عقدة في الشجرة T، عمق depth العقدة v هو عدد الآباء لـ v، باستثناء v بحد ذاتها. على سبيل المثال، في الشجرة المبينة في الشكل 5-2، عمق العقدة التي تخزن International هو 2، لاحظ أن هذا التعريف يعني أن عمق جذر الشجرة T هو 0.

يمكن تعريف عمق عقدة v عودياً كما يلي:

- إذا كانت v هي الجذر، عندئذ فإن العمق هو 0.
- وإلا، عمق v هو عمق العقدة الأب لـ v مضافاً إليه واحداً.

بناء على هذا التعريف، تمثل الخوارزمية العودية التالية حساب عمق عقدة v في شجرة T.

Algorithm depth(T,v)
  if v is the root of T **then** **return** 0
  else **return** 1+depth(T,w) , where w is the parent of v in T

المقطع 5-2- خوارزمية حساب العمق.

خوارزميات التجول عبر الشجرة 2

إن زمن التنفيذ للخوارزمية depth(T,v) هو $O(d_v)$ حيث تمثل d_v عمق العقدة v في الشجرة T. وذلك لأن الخوارزمية تنفذ خطوة عودية ذات زمن ثابت من أجل كل أب لـ v. وبالتالي، فإن الخوارزمية depth(T,v) تنفذ في زمن O(n) في الحالة الأسوأ، حيث n هو العدد الكلي للعقد في T، وذلك بما أن عقدة ما في الشجرة T يمكن أن تكون ذات عمق n-1 في الحالة الأسوأ.

الارتفاع Height: يعرف ارتفاع height عقدة v في شجرة T أيضاً بشكل عودي على النحو:

- إذا كانت v عقدة خارجية، عندئذ فإن ارتفاعها هو 0.
- وإلا، إرتفاع v هو الارتفاع الأعظمي لإبن العقدة v مضافاً إليه واحداً.

إن ارتفاع شجرة غير فارغة T هو ارتفاع جذر الشجرة، على سبيل المثال، الشجرة في الشكل 5-2 ارتفاعها هو 4.

يمكن أيضا النظر إلى الارتفاع height لشجرة غير فارغة T على أنه العمق depth الأعظمي لعقدة خارجية من T.

نبين فيما يلي خوارزمية حساب ارتفاع شجرة.

Algorithm height1(T)

h←0

for each vertex v in T **do** **if** v is an external node in T **then** h←max(h,depth(T,v)

return h

المقطع 5-4- خوارزمية حساب الارتفاع.

التجول السابق للترتيب

Preorder Traversal PLR

إن التجول عبر شجرة هو طريقة نظامية لصعود أو لزيارة visiting جميع العقد في شجرة T.

نعرض في هذه الفقرة، أسلوب التجول الأساسي عبر الأشجار، ويدعى التجول السابق للترتيب NLR، وفي الفقرة القادمة، سندرس أسلوب تجول أساسي آخر يدعى التجول اللاحق للترتيب LRN ولاحقاً سندرس التجوال وفق الترتيب LNR .

في التجول السابق للترتيب preorder traversal لشجرة T، يتم زيارة جذر الشجرة T أولاً ومن ثم يتم التجول عودياً عبر الأشجار الفرعية والتي جذورها هي أبناء جزر الشجرة T. إذا كانت الشجرة مرتبة ordered عندئذ فإن الأشجار الفرعية يتم التجول عليها وفقاً لترتيب الأبناء.

إن الفعل المحدد المرافق لعملية زيارة عقدة ما v يتوقف على التطبيق الخاص بالتجول، وقد يتضمن أي شيء بدءاً من مزايدة عداد وانتهاء بالقيام ببعض العمليات الحسابية المعقدة على v.

يبين المقطع التالي شبه الشيفرة الخاصة بخوارزمية التجول السابق للترتيب لشجرة جذرها العقدة v.

Algorithm preorder(T,v):

perform the "visit" action for node v

for each child w of v in T **do** preorder(T,w) {recursively traverse the subtree rooted at w}

المقطع 5-8- خوارزمية التجول preorder.

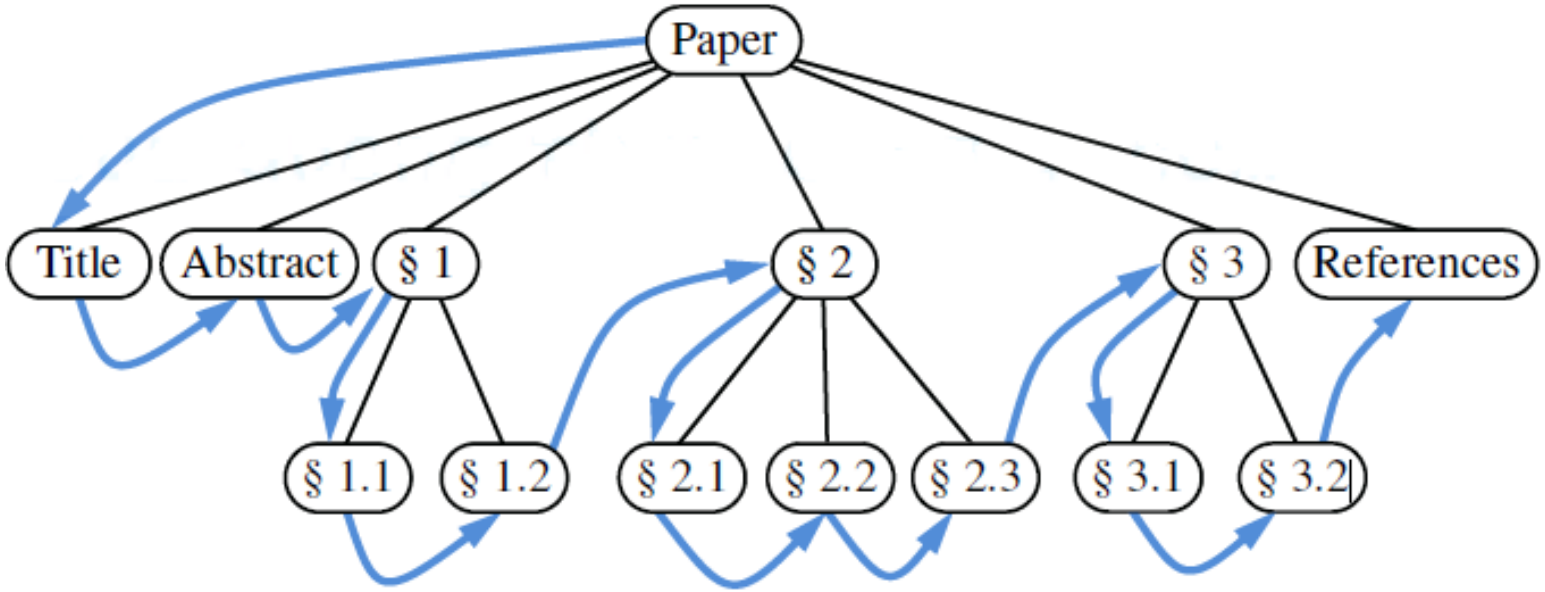
التجول السابق للترتيب2

Preorder Traversal PLR

إن خوارزمية التجول السابق للترتيب مفيدة من أجل إنتاج ترتيب خطي للعقد شجرة بحيث إن الآباء يجب أن يأتو قبل أبنائهم في الترتيب. إن مثل هذا الترتيب له العديد من التطبيقات المختلفة، نبين في المثال التالي أحدها.

مثال

إن التجول السابق للترتيب للشجرة المرفقة بمستند (كما في مثال سابق) تختبر كامل المستد بشكل متتالي، منذ البداية وحتى النهاية. إذا تمت إزالة العقد الخارجية قبل التجول، عندئذ فإن التجول يختبر جدول المحتويات للمستند (كما يبين الشكل التالي).



الشكل 5-6- التجول السابق للترتيب لشجرة مرتبة PLR

التجول السابق للترتيب3

هناك تطبيق هام لخوارزمية التجول السابق للترتيب، وهو يولد تمثيلاً لشجرة كاملة على شكل سلسلة محرفية.

لنفترض مجدداً، من أجل كل عنصر e مخزن في شجرة T، أن استدعاء e.print() يعيد سلسلة محرفية مرتبطة بـ e. فإن التمثيل الأبوي على شكل سلسلة محرفية parenthetic string representation يمكن ان يعرف عودياً كمايلي:

- إذا كانت الشجرة T مؤلفة من عقدة واحدة ν فإن $P(T) = \nu.element().print()$
- وإلا فإن $P(T) = \nu.element().print()+"("+P(T_1)+","+P(T_2)+","+....+P(T_k)+")"$

حيث إن ν هي جذر T و $T_1,T_2,....,T_k$ هي الأشجار الفرعية التي جذورها هي أبناء ν.

بناء عليه فإن التمثيل الأبوي على شكل سلسلة محرفية للشجرة المبينة في الشكل 5-2 هو على النحو التالي:

Electronics R'Us (R&D Sales (Domestic International (Canada S. America
Overseas (Africa Europe Asia Australia))
Purchasing
Manufacturing (TV CD Tuner))

الشكل 5-7- التمثيل الأبوي على شكل سلسلة محرفية.

التجول اللاحق للترتيب 1

Postorder Traversal LRP

خوارزمية هامة أخرى للتجول عبر الشجرة هي خوارزمية التجول اللاحق للترتيب **postorder traversal**. يمكن النظر إلى هذه الخوارزمية على أنها معاكسة لخوارزمية التجول السابق للترتيب، وذلك لأنها تتجول عودياً على الأشجار الفرعية التي جذورها أبناء لجذر الشجرة أولاً ومن ثم تزور الجذر.

إنها شبيهة بخوارزمية التجول السابق للترتيب في أننا نستخدمها لحل مسائل خاصة من خلال تخصيص الفعل المرافق لعملية الزيارة للعقدة v.

وأيضاً، وكما في حالة التجول السابق للترتيب، إذا كانت الشجرة مرتبة، فإننا نجري استدعاءات عودية لأبناء العقدة v بحسب ترتيبها المحدد.

المقطع التالي يوضح شبه الشيفرة لخوارزمية التجول اللاحق للترتيب:

```
Algorithm postorder(T,v):
  for each child w of v in T do
    postorder(T,w) {recursively traverse the subtree rooted at w}
perform the "visit" action for node v
```

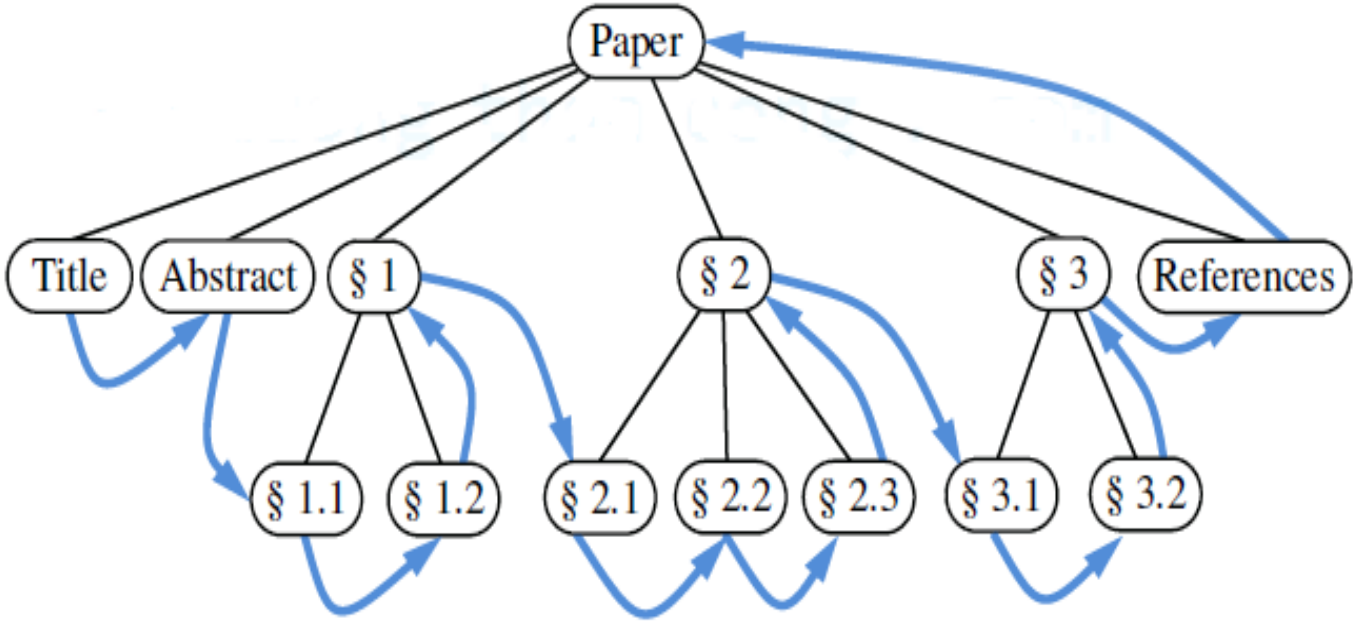
المقطع 5-11- خوارزمية التجول postorder.

التجول اللاحق للترتيب 1

Postorder Traversal LRP

إن أصل التسمية "التجول اللاحق للترتيب postorder traversal" يأتي من حقيقة أن طريقة التجول ستقوم بزيارة v بعد أن تكون قد قامت بزيارة جميع العقد في الشجرة الفرعية التي جذرها v.

يبين الشكل 5-8 هذا التجول المماثل للشكل 5-6:



الشكل 5-8- التجول اللاحق للترتيب عبر شجرة مرتبة LRP.

فإن التجول اللاحق للترتيب لشجرة T ذات n عقدة يستغرق زمناً O(n) وذلك بفرض أن زيارة كل عقدة تتطلب زمناً O(1). وبالتالي، فإن التجول اللاحق للترتيب تنفذ في زمن خطي.

التجول اللاحق للترتيب 2

إن طريقة التجول اللاحق للترتيب مفيدة في حل المسائل التي نرغب فيها بحساب صفة ما لكل عقدة في شجرة، إلا أن حساب تلك الصفة لـ v يتطلب أن نكون قد حسبنا نفس الصفة لأبناء v.

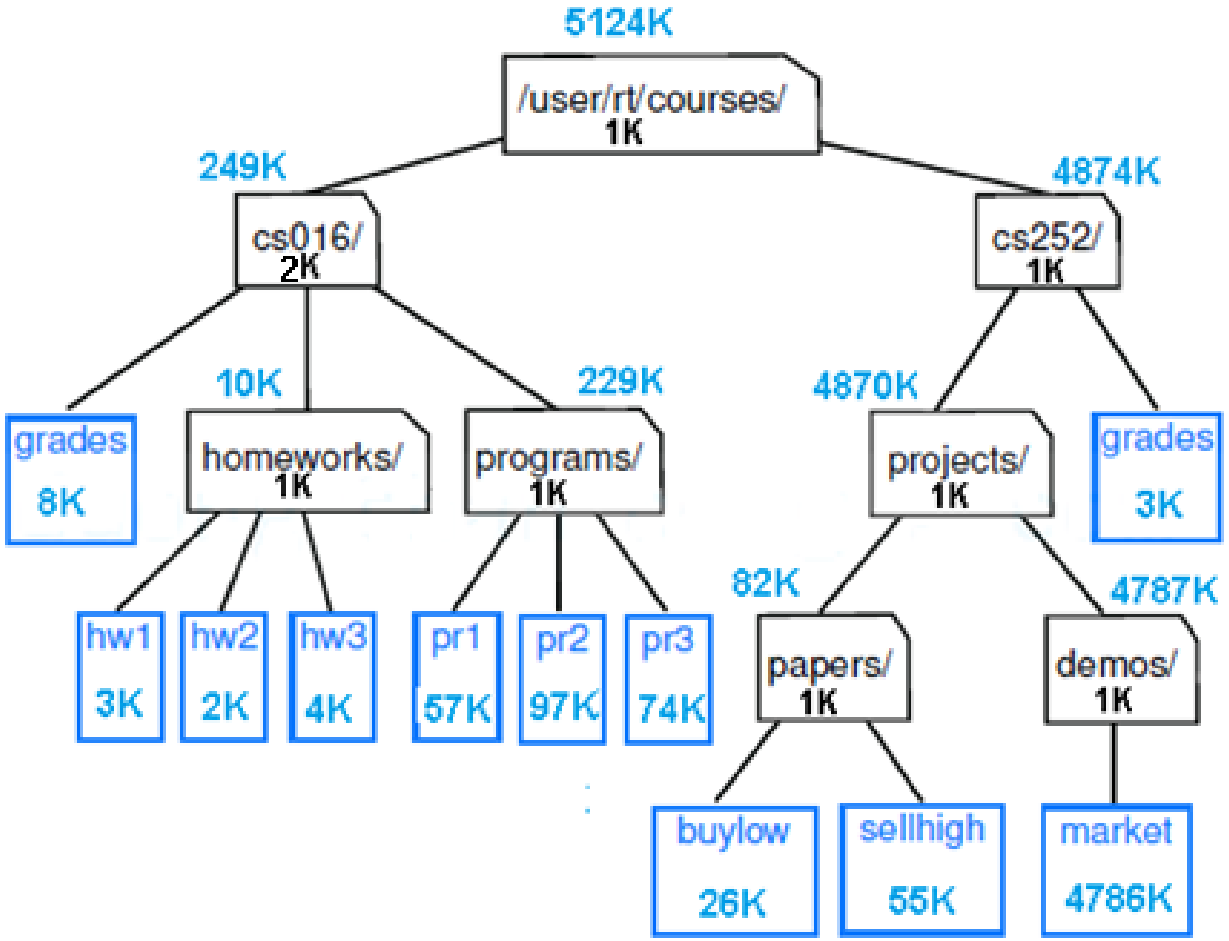
كمثال على هذه الحالة، ليكن لدينا شجرة نظام الملفات T، حيث تمثل العقد الخارجية الملفات والعقد الداخلية المجلدات. وبفرض أننا نرغب بحساب مساحة القرص التي يشغلها مجلد ما، فإن هذا الأمر يحسب عودياً على أنه مجموع:

- حجم المجلد بحد ذاته.
- حجم الملفات في المجلد.
- الحجم المشغول بالمجلدات الأبناء.

بالنظر إلى الشكل 5-9، إن هذا الحساب يمكن أن يتم تنفيذه بالتجول اللاحق للترتيب عبر الشجرة T، فبعد أن يتم التجول عبر الأشجار الفرعية لعقدة داخلية، نقوم بحساب الحجم المستخدم من قبل v من خلال جمع حجوم المجلد v بحد ذاته والملفات المحتواة في v إلى الحجم المشغول من قبل كل عقدة إبن داخلية لـ v، والتي حسبت من خلال التجوالات العودية اللاحقة للترتيب لأبناء v.

التجول اللاحق للترتيب 3

Postorder Traversal LRP



الشكل 5-9- شجرة الملفات مرفقة بحجوم الملفات.

ويمكن كتابة الطريقة diskSpace لتستحدم التجول اللاحق للترتيب لشجرة نظام الملفات T، لطباعة أسماء الملفات ومساحة القرص المستخدم من قبل المجلد المرفق بكل عقدة داخلية من T.

عندما يتم استدعاؤها على جذر الشجرة T فإنها تنفذ في زمن O(n) حيث n هو عدد العقد في T.

النوع الثالث للتجوال InOrder LNR سيتم دراسته لاحقاً

الأشجار الثنائية1

الشجرة الثنائية binary tree هي شجرة مرتبة تملك الخصائص التالية:

- كل عقدة لها على الأكثر إبنين. - كل عقدة إبن يشار إليها على أنها إما إبن يسار left child أو إبن يمين right child.
- الابن اليسار يسبق الابن اليمين في ترتيب أبناء العقدة.

تدعى الشجرة الفرعية التي جذرها عند الابن اليسار أو اليمين لعقدة داخلية v بالاسم شجرة فرعية يسارية left subtree أو شجرة فرعية يمينية right subtree على التوالي.

يقال عن شجرة ثنائية أنها تامة proper إذا كانت كل عقدة داخلية إما لاتملك أي أبناء وإما تملك إبنين. وتسمى عادة بالشجرة الثنائية الممتلئة full، وبالتالي، في الشجرة الثنائية التامة يكون لكل عقدة داخلية ابنين تماماً،

شجرة ثنائية مثالية Perfect Binary Tree: مثل السابقة وتكون جميع العقد الورقية في نفس المستوى.

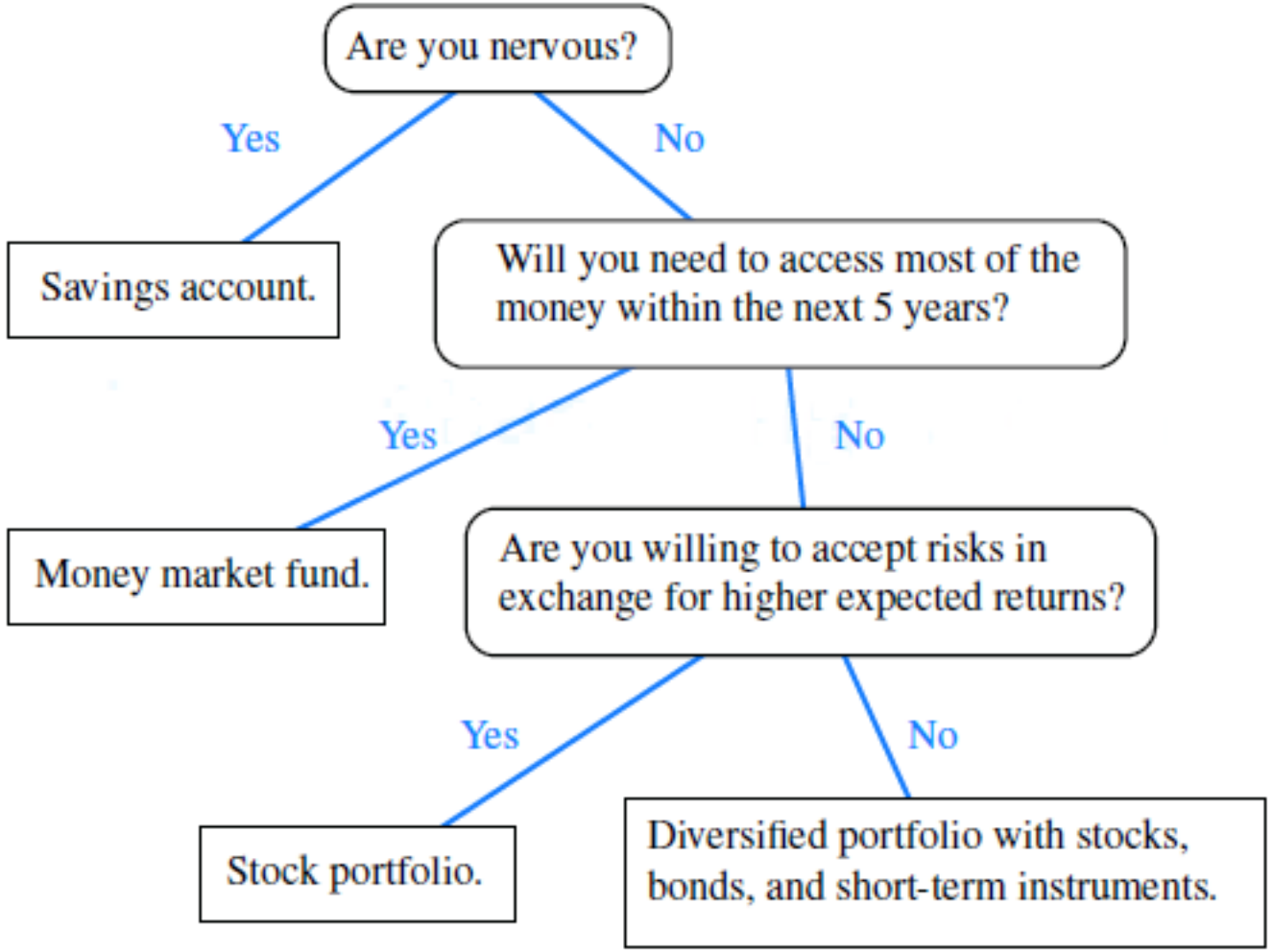
Complete Binary Tree الكاملة مثل السابقة مع السماخ أن يكون العنصر الأخير لايملك أخ يميني.

مثال 1- أحد الأصناف الهامة للأشجار الثنائية يظهر عندما نرغب بتمثيل عدد من النواتج التي يمكن أن تنتج من الإجابة عن سلسلة من الأسئلة ذات الإجابات (نعم-لا). تكون كل عقدة داخلية مرتبطة بسؤال. بدءاً من الجذر، نحن نذهب إلى اليسار أو اليمين للعقدة الحالية، بحسب فيما إذا كان الجواب نعم أم لا.

بإمكاننا تتبع حد edge من أب إلى إبن، من خلال تتبع مسار في الشجرة من الجذر إلى عقدة خارجية.

الأشجار الثنائية 2

تعرف مثل هذه الأشجار الثنائية باسم أشجار القرار decision trees، وذلك لأن كل عقدة خارجية v في مثل هذه الشجرة تمثل قراراً بما يجب فعله. يبين الشكل 5-10 شجرة قرار تعطي توصيات للمستثمرين المحتملين:

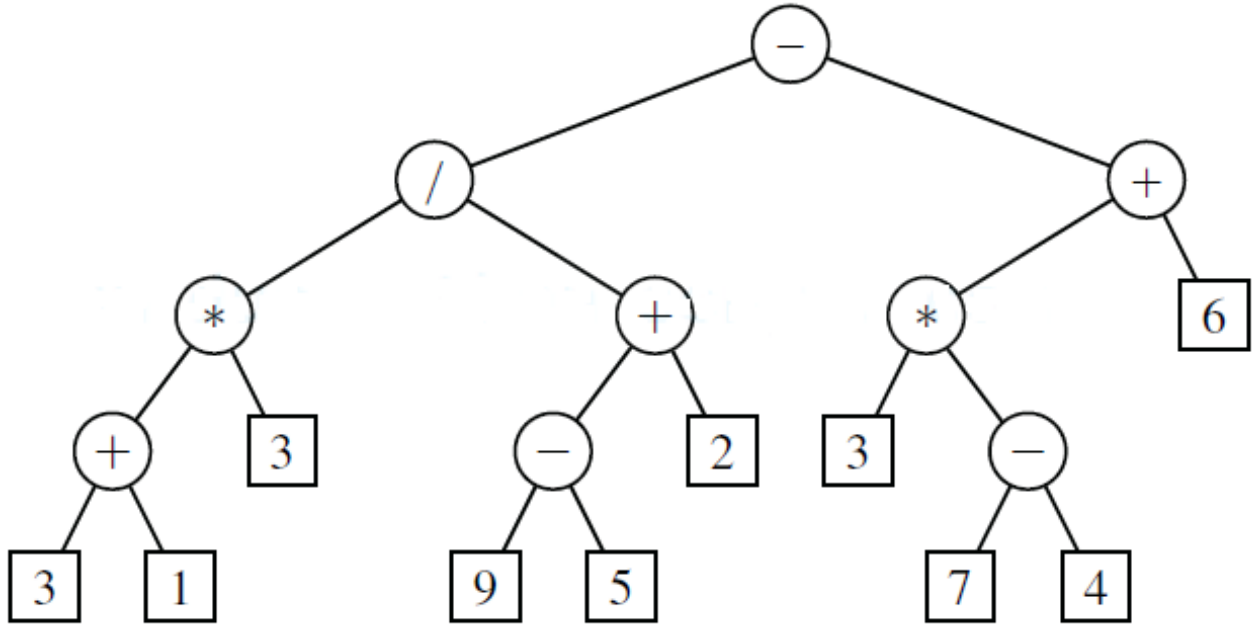


الشكل 5-10- شجرة القرار.

الأشجار الثنائية 3

مثال 2- يمكن تمثيل تعبير حسابي من خلال شجرة ثنائية تكون العقد الخارجية فيها مرتبطة بالمتحولات أو الثوابت، وتكون عقدها الداخلية مرتبطة بأحد المعاملات +، -، *، / كما يبين الشكل 5-11 الذي يمثل التعبير:

((((3+1)*3)/((9–5)+2))-((3*(7-4))+6))



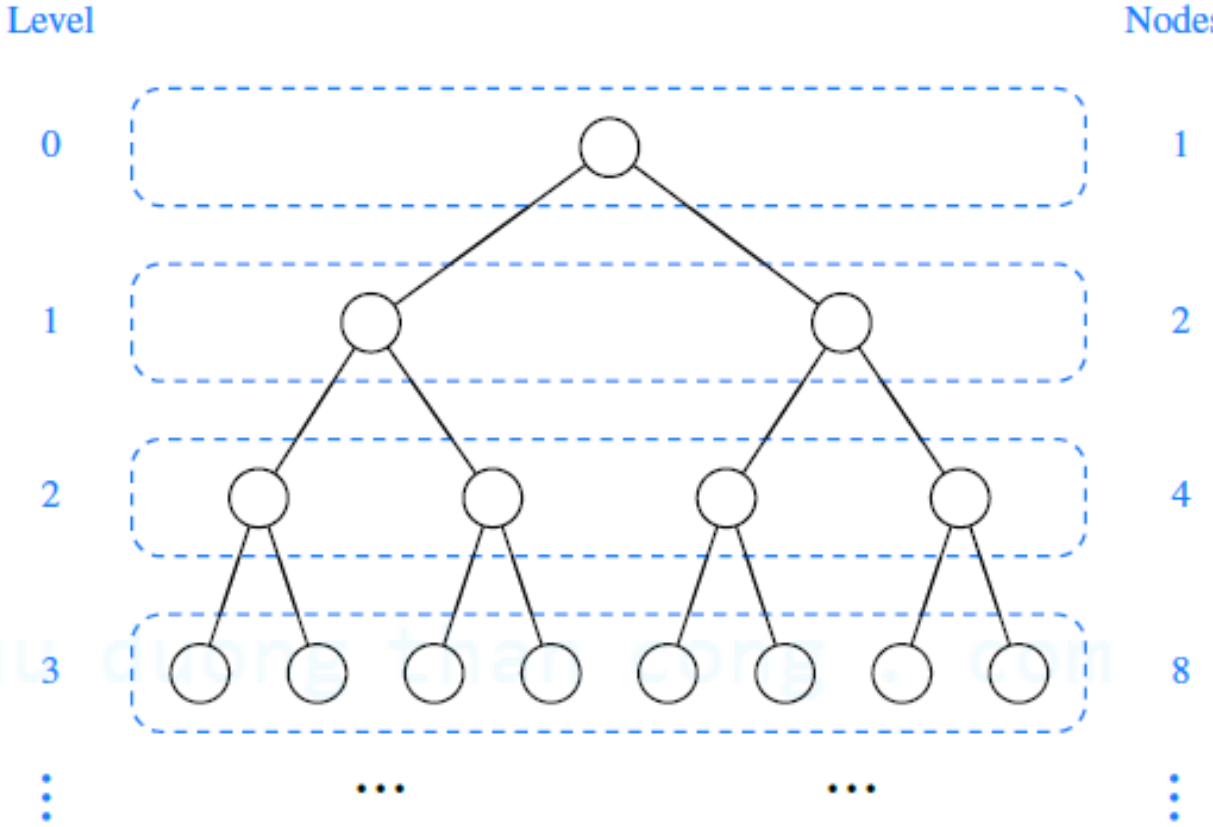
- إذا كانت العقدة خارجية، عندئذ فإن قيمتها هي قيمة المتحول أو الثابت.
- إذا كانت العقدة داخلية، عندئذ، فإن القيمة معرفة من خلال تطبيق العملية على الأبناء.

إن التعبير الثنائي هو عبارة عن شجرة ثنائية تامة proper، بما أن المعاملات +، -، *، / هي معاملات ثنائية، تاخذ عاملين.

الشكل 5-11- شجرة ثنائية تمثل تعبيراً حسابياً. بناء شجرة التعبير

Properties of Binary Trees

خصائص الأشجار الثنائية



الشكل 5-12- العدد الأعظمي للعقد عند كل مستوى.

تملك الأشجار الثنائية عدة خصائص هامة للتعامل مع العلاقات بين إرتفاعاتها وعدد عقدها.

نسمي مجموعة جميع العقد في شجرة T تملك نفس العمق d بالاسم المستوى d لـ T (أو level). وبالتالي، ففي شجرة ثنائية، المستوى 0 يحوي على الأكثر عقدة واحدة (الجذر)، المستوى 1 يحوي على الأكثر عقدتين (أبناء الجذر)، المستوى 2 يحوي على الأكثر 4 عقد، ... وهكذا، يمكن التعميم بالقول أن المستوى d يحوي على الأكثر 2^d عقدة. يبين الشكل 5-12 مفهوم المستويات:

إن العدد الأعظمي للعقد عند مستويات شجرة ثنائية تنمو أسياً كلما نمت الشجرة للأسفل.

سنقبل بالخصائص التالية للشجرة الثائية دون الخوض في برهانها الرياضي (خارج المقرر).

Properties of Binary Trees

خصائص الأشجار الثنائية

- $h+1 \le n \le 2^{h+1} -1$
- $1 \le n_E \le 2^h$
- $h \le n_I \le 2^h -1$
- $\log(n+1) -1 \le h \le n-1$

وإذا كانت الشجرة T تامة proper فإننا نضيف الخصائص التالية:

- $2h+1 \le n \le 2^{h+1} -1$
- $h+1 \le n_E \le 2^h$
- $h \le n_I \le 2^h -1$
- $\log(n+1) -1 \le h \le (n-1)/2$

وأخيراً، إذا كانت الشجرة الثنائية تامة وغير فارغة فإن:

$n_E = n_I + 1$

حيث إن:

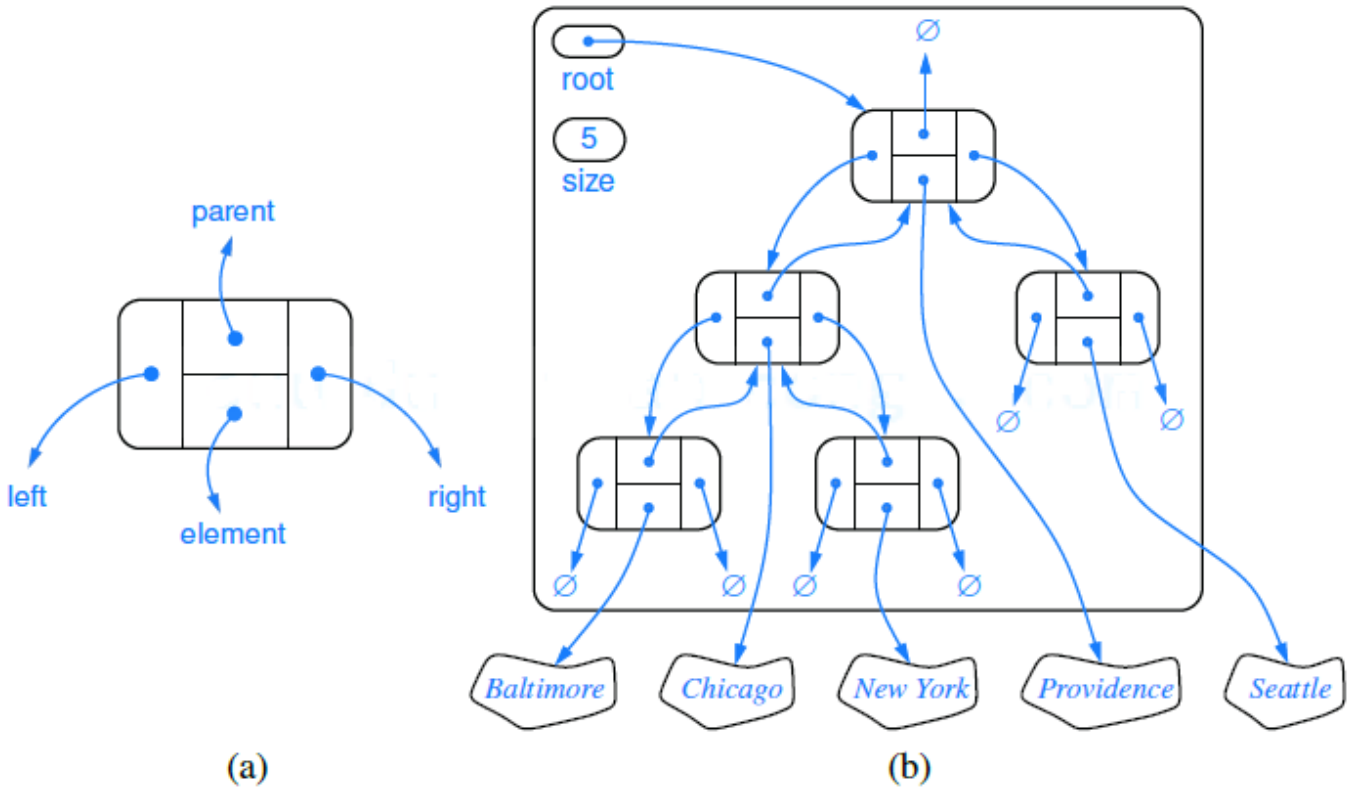
- T شجرة ثنائية غير فارغة.
- n عدد العقد.
- n_E عدد العقد الخارجية.
- n_I عدد العقد الداخلية.
- h إرتفاع الشجرة.

A Linked Structure for Binary Trees

البنية المترابطة للأشجار الثنائية 1

فإن الطريقة الطبيعية لتحقيق شجرة ثنائية T هي أن نستخدم بنية مترابطة linked structure، حيث نمثل كل عقدة v من T من خلال غرض position (الشكل 5-13-a) يحوي حقولاً تتضمن مراجعاً إلى العنصر المخزن في v وإلى الغرض من النوع position المرفق مع أبناء الغرض v و أبوه.

إذا كانت v هي جذر T، فإن حقل أب v هي الغرض الصفري null. وإذا كانت v لا تملك أي إبن يسار Left child، فإن الحقل اليسار لـ v هو null، وبالمثل بالنسبة للابن اليمين. كما أننا نخزن أيضاً عدد العقد في Tفي متحول يدعى Size.



الشكل 5-13- البنية المترابطة لشجرة ثنائية,

البنية المترابطة للأشجار الثنائية 2

A Linked Structure for Binary Trees

سندرس طرائق التحديث التالية:

- addRoot(e): تنشيء وتعيد عقدة جديدة r تخزن العنصر e وتجعل r هي جذر الشجرة. يحدث خطأ إذا كانت الشجرة غير فارغة.
- insertLeft(v,e): تنشيء وتعيد عقدة جديدة w تخزن العنصر e، وتضيف w كابن يسار لـ v وتعيد w. يحدث خطأ إذا كانت العقدة v تملك إبناً يسار.
- insertRight(v,e): مثل سابقتها، ولكن مع الابن اليمين.
- remove(v): تقوم بحذف العقدة v، واستبدالها بابنها (إن وجد)، وتعيد العنصر المخزن في v. يحدث خطأ إذا كانت v تملك إبنين.
- attach(v,T_1,T_2): تصل الشجرتين T_1 و T_2 على التوالي، كشجرتين فرعيتين يسارية ويمينية للعقدة الخارجية v. يحدث خطأ إذا كانت v ليست عقدة خارجية.

يحوي الصنف LinkedBinaryTree تابعاً بانياً بدون وسطاء يعيد شجرة ثنائية فارغة. وانطلاقاً من هذه الشجرة الثنائية الفارغة، يمكن أن نبني أي شجرة ثنائية من خلال إنشاء العقدة الأولى باستخدام الطريقة addRoot وتكرار استدعاء الطريقتين insertLeft و insertRight و /أو الطريقة attach.

عندما يتم تمرير غرض v من النوع position كوسيط لواحدة من طرائق هذا الصنف، يتم التحقق من صلاحيته من خلال استدعاء الطريقة الإضافية checkPosition(v).

تتم زيارة قائمة العقد بالتجوال السابق للترتيب من خلال الطريقة العودية preorderPositions،

البنية المترابطة للأشجار الثنائية 2

A Linked Structure for Binary Trees

سندرس طرائق التحديث التالية:

- addRoot(e): تنشيء وتعيد عقدة جديدة r تخزن العنصر e وتجعل r هي جذر الشجرة. يحدث خطأ إذا كانت الشجرة غير فارغة.
- insertLeft(v,e): تنشيء وتعيد عقدة جديدة w تخزن العنصر e، وتضيف w كابن يسار لـ v وتعيد w. يحدث خطأ إذا كانت العقدة v تملك إبناً يسار.
- insertRight(v,e): مثل سابقتها، ولكن مع الابن اليمين.
- remove(v): تقوم بحذف العقدة v، واستبدالها بابنها (إن وجد)، وتعيد العنصر المخزن في v. يحدث خطأ إذا كانت v تملك إبنين.
- attach(v,T_1,T_2): تصل الشجرتين T_1 و T_2 على التوالي، كشجرتين فرعيتين يسارية ويمينية للعقدة الخارجية v. يحدث خطأ إذا كانت v ليست عقدة خارجية.

يحوي الصنف LinkedBinaryTree تابعاً بانياً بدون وسطاء يعيد شجرة ثنائية فارغة. وانطلاقاً من هذه الشجرة الثنائية الفارغة، يمكن أن نبني أي شجرة ثنائية من خلال إنشاء العقدة الأولى باستخدام الطريقة addRoot وتكرار استدعاء الطريقتين insertLeft و insertRight و /أو الطريقة attach.

عندما يتم تمرير غرض v من النوع position كوسيط لواحدة من طرائق هذا الصنف، يتم التحقق من صلاحيته من خلال استدعاء الطريقة الإضافية checkPosition(v).

تتم زيارة قائمة العقد بالتجوال السابق للترتيب من خلال الطريقة العودية preorderPositions،

Performance of the Linked Binary Tree Implementation

أداء الشجرة الثنائية المترابطة

يبين الجدول التالي تلخيصاً لأزمنة التنفيذ لطرائق الصنف LinkedBinaryTree:

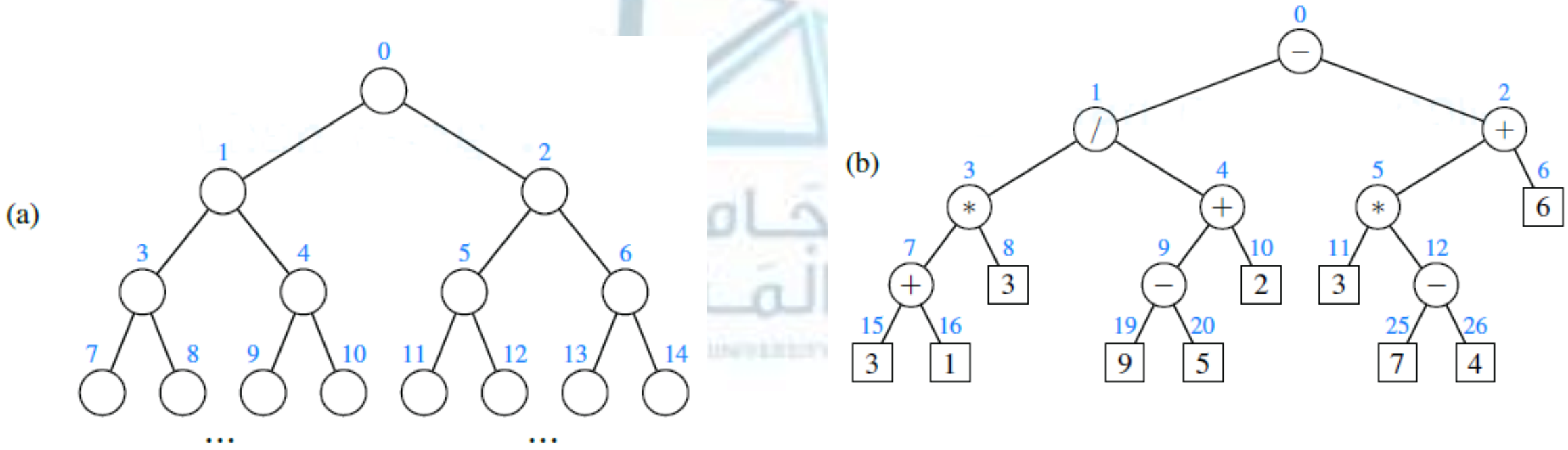
الطريقة	زمن التنفيذ
size ،isEmpty	O(1)
iterator ،position	O(n)
replace.	O(1)
root ،parent ،children ،left ،right ،sibling	O(1)
hasLeft ،hasRight ،isInternal ،isExternal ، isRoot	O(1)
insertLeft ،insertRight ،attach ،remove	O(1)

Performance of the Linked Binary Tree Implementation

أداء الشجرة الثنائية المترابطة

إن تابع الترقيم p يعرف باسم ترقيم المستويات level numbering للعقد في شجرة ثنائية T، فهو يرقم العقد في كل مستوى بترتيب تصاعدي من اليسار إلى اليمين، على الرغم من أنه يتجاوز بعض الأرقام.

يبين الشكل 5-14 ترقيم المستويات:



الشكل 5-14- ترقيم المستويات في شجرة ثنائية (a : الأسلوب العام، b: مثال).

Traversals of Binary Trees

التجول عبر الأشجار الثنائية 1

كما في حال الأشجار العامة، غالباً ما تتطلب أي عمليات معالجة لعناصر الأشجار الثنائية تجولاً عبر هذه الأشجار.

التجول السابق للترتيب لشجرة ثنائية: بإمكاننا تبسيط الخوارزمية في حال التجول على شجرة ثنائية لتصبح كما يلي:

Algorithm binaryPreorder(T,v):
  perform the "visit" action for node v
  if v has a left child u in T **then** binaryPreorder(T,u) {recursively traverse left subtree}
  if v has a right child w in T **then** binaryPreorder(T,w) {recursively traverse right subtree}

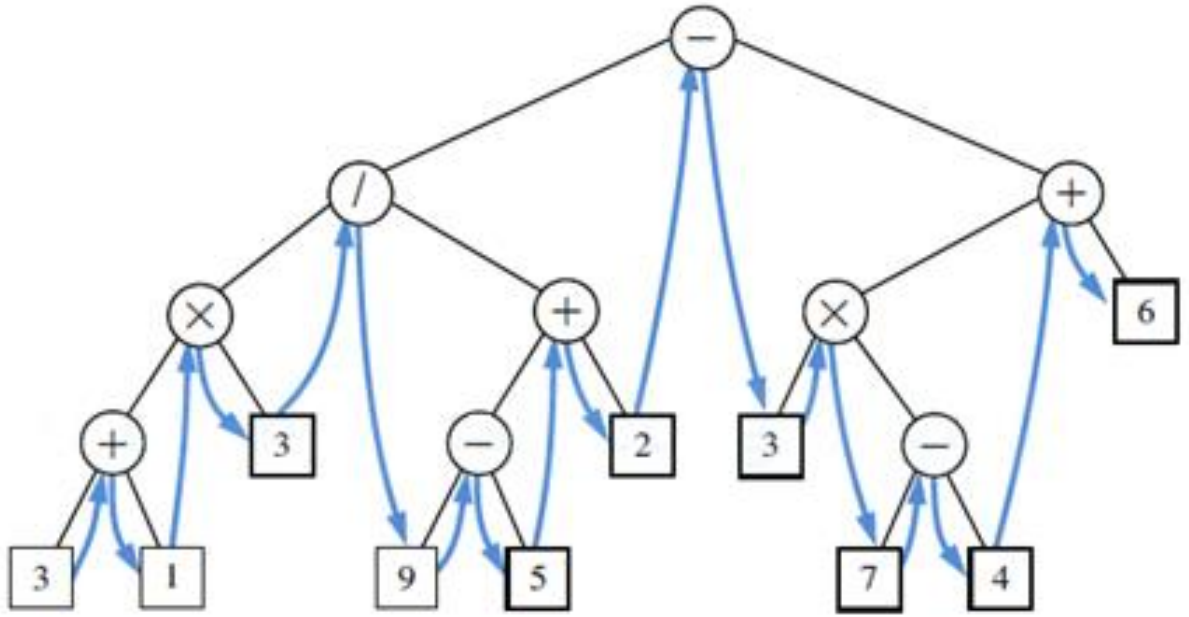
خوارزمية binaryPreorder

التجول اللاحق للترتيب لشجرة ثنائية:

Algorithm binaryPostorder(T,v):
  if v has a left child u in T **then** binaryPostorder(T,u) {recursively traverse left subtree}
  if v has a right child w in T **then** binaryPostorder(T,w) {recursively traverse right subtree}
  perform the "visit" action for node v

Inorder Traversal of a Binary Tree

التجول المرتب عبر شجرة ثنائية



الشكل 5-16- التجول المرتب عبر شجرة ثنائية LPR.

طريقة إضافية للتجول عبر شجرة ثنائية، هي التجول المرتب **inorder traversal**. في هذا التجول، نقوم بزيارة عقدة بين التجولات العودية من أشجارها اليسارية واليمينية. يبين المقطع التالي خوارزمية هذا النوع من التجول:

Algorithm inorder(T,v):
  if v has a left child u in T **then**
    inorder(T,u) {recursively traverse left subtree}
  perform the "visit" action for node v
  if v has a right child w in T **then**
    inorder(T,w) {recursively traverse right subtree}

خوارزمية inorder

Binary Search Trees

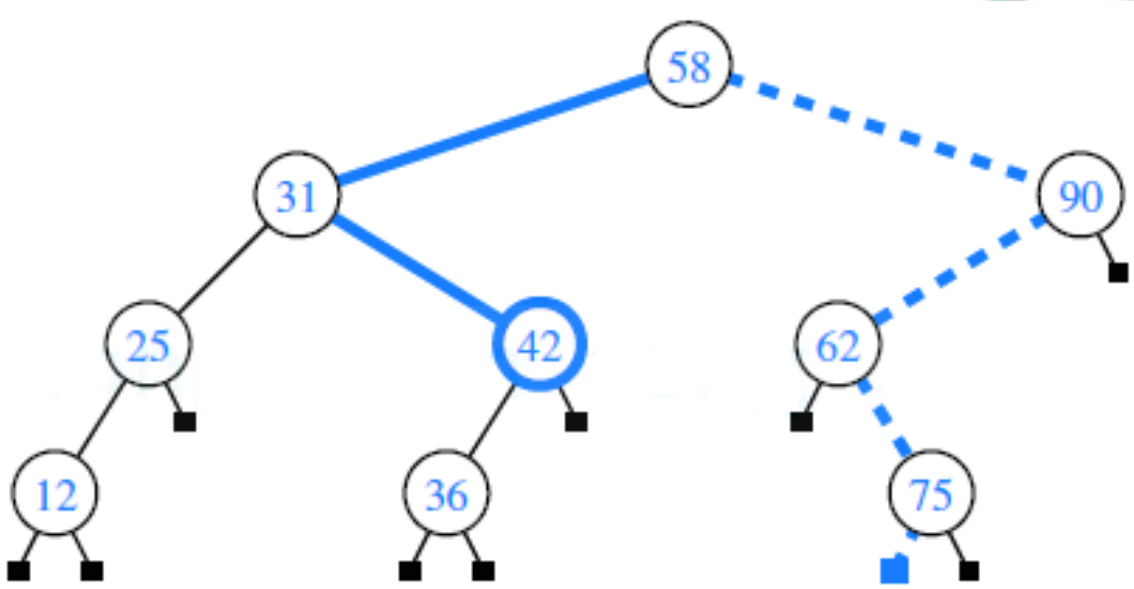
أشجار البحث الثنائية 1

لتكن S هي عبارة عن مجموعة عناصرها تملك علاقة ترتيب. مثلاً يمكن لـ S أن تكون مجموعة من الأعداد الصحيحة. عندها فإن شجرة البحث الثنائية binary search tree لـ S هي عبارة عن شجرة ثنائية تحوي:

- كل عقدة داخلية v من T تخزن عنصراً من عناصر S ويرمزله x(v).
- من أجل كل عقدة داخلية v من T، فإن العناصر المخزنة في الشجرة اليسارية لـ v هي أقل من أو تساوي x(v) والعناصر المخزنة في الشجرة اليمينية لـ v هي أكبر x(v).
- لا تخزن العقد الخارجية لـ T أي عنصر.

يقوم التجول المرتب على العقد الداخلية في شجرة البحث الثنائية T بزيارة العناصر بترتيب غير تنازلي (كما يبين الشكل المرفق):

القيمة في كل عقدة أكبر من القيم الموجودة في شجرتها الفرعية اليسرى (إن وجدت) و أقل من القيم الموجودة في شجرتها الفرعية اليمنى (إن وجدت). الشجرة الثنائية التي تتمتع بهذه الخاصية تدعى شجرة البحث الثنائية binary search tree (BST)



الشكل 5-17- شجرة بحث ثنائية.

يمكن استخدام شجرة بحث ثنائي T من أجل مجموعة S من أجل إيجاد فيما إذا كانت قيمة بحث ما y موجودة ضمن S أم لا، وذلك من خلال التجول عبر مسار هابط من الشجرة بدءاً من الجذر (الشكل 5-17 الذي يظهر مسار البحث عن القيمة 36 بخط غامق (عملية ناجحة) ومسار البحث عن لقيمة 70 بخط منقط (نتيجه سلبية)).

تتم عملية البحث كمايلي: عند كل عقدة داخلية تتم مصادفتها، نقارن قيمة المبحوث عنه y مع العنصر x(v) المخزن في v، فإن كانت القيمتان متساويتان ينتهي البحث، أما إن كانت قيمة البحث أصغر من x(v) عندها يستمر البحث في الشجرة الفرعية اليسارية وإلا يستمر في الشجرة الفرعية اليمينية. فإذا وصلنا إلى عقدة خارجية بدون أن نجد القيمة التي نبحث عنها، تكون القيمة ليست ضمن الشجرة.

هناك العديد من أشكال التجول الأخرى عبر الأشجار الثنائية إلا أننا نكتفي بالمقدار الذي تم عرضه في هذا البحث، ونترك للمهتمين التعرف على الأشكال الأخرى.

انتهت محاضرة الأسبوع 8

جَامعة المَنارة
MANARA UNIVERSITY

كلية الهندسة قسم المعلوماتية

بنى معطيات 1

Data Structure 1

ا. د. علي عمران سليمان

محاضرات الأسبوع التاسع

الأشجار2

Tree 2

الفصل الثاني 2023-2024

محتوى الفصل

الأشجار 1 + 2	Trees 1 +2
1- مقدمة	
2- الأشجار العامة.	2- General Trees.
3- خوارزميات التجول عبر الشجرة.	
4- الأشجار الثنائية.	
5- تعريف شجرة البحث الثنائية كقاموس بيانات.	
6- تحقيق خوارزميات البحث.	
7- حساب أداء خوارزميات البحث.	
8 - تحقيق طرائق التبديل على شجرة بحث ثنائية.	
9 - تحقيق شجرة البحث الثنائية بلغة C++	

References

- Deitel & Deitel, Java How to Program, Pearson; 10th Ed(2015)

- د.علي سليمان، بنى معطيات بلغة JAVA، بنى معطيات بلغة C++، بنى معطيات بلغة Pascal جامعة تشرين 2014، 2007، 1998

شجرة البحث الثنائي –بتعريف بسيط- هي عبارة عن بنية معطيات من نوع شجرة tree، يمكن أن تستخدم لبناء أو تحقيق قاموس dictionary.

أما القاموس فهو عبارة عن بنية معطيات مجردة تملك الطرائق methods التالية:

- find(k): تعيد العنصر ذو المفتاح k إذا كان موجوداً.
- findAll(k): تعيد مجموعة قابلة للتجول iterable collection تضم جميع العناصر ذات مفتاح مساوي لـ k.
- insert(k,x): تقوم بحشر عنصر ذو مفتاح k وقيمة x.
- remove(e): تقوم بحذف العنصر e وإعادته.
- removeAll(k): تقوم بحذف جميع العناصر ذات المفتاح k، وتعيد مكرر iterator إلى قيمها.

تعيد الطريقة find القيمة null إذا كان k غير موجود.

يتضمن القاموس المرتب ordered dictionary كبنية معطيات مجردة بعض الطرائق الإضافية للبحث من خلال العنصرين السابق predecessor واللاحق successor لمفتاح أو عنصر، إلا أن أدائها مشابه لأداء الطريقة find، لذا فإننا سنركز على الطريقة find كعملية بحث أساسية في هذا الفصل.

إن الأشجار الثنائية binary trees هي بنية معطيات ممتازة لتخزين عناصر قاموس، بافتراض أن لدينا علاقة ترتيب معرفة على المفاتيح. وكما سبق وأشرنا سابقاً, فإن شجرة البحث الثنائية binary search tree هي شجرة ثنائية T يكون فيها

كل عقدة داخلية v من الشجرة مخصصة لتخزين عنصر (k,x) بحيث إن:

- المفاتيح المخزنة في العقد التي في الشجرة الفرعية اليسارية لـ v أصغر من أو تساوي k.
- المفاتيح المخزنة في العقد التي في الشجرة الفرعية اليمينية لـ v أكبر من k.

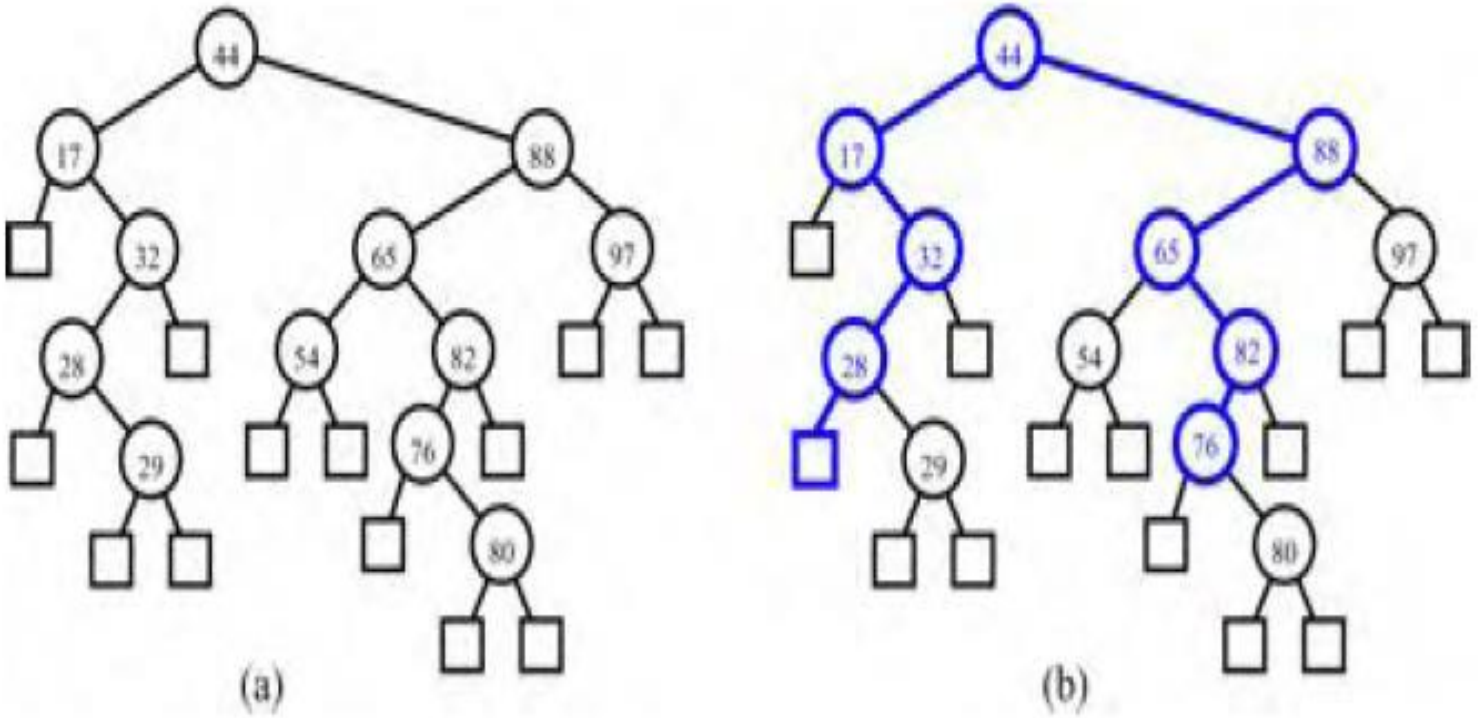
إن المفاتيح المخزنة في عقد T تتيح طريقة لتنفيذ البحث من خلال إجراء مقارنة عند كل عقدة داخلية v، ويمكن أن تتوقف عند v أو تتابع عند الابن اليساري أو اليميني لـ v. وبالتالي، فإن أشجار البحث الثنائية هي أشجار ثنائية مكتملة proper غير فارغة، وتستخدم العقد الخارجية كمقابض placeholders. إن هذا الأسلوب يبسط الكثير من خوارزميات البحث والتعديل الخاصة بنا.

يمكن في بعض الأحيان أن نسمح باستخدام أشجار بحث ثنائية غير مكتملة improper، فهي تتيح استخداماً أفضل لمساحة الذاكرة، إلا أنها أكثر كلفة وتعقيداً في تحقيق خوارزميات البحث والتعديل.

وبغض النظر فيما إذا كنا نرى شجرة البحث الثنائية كشجرة مكتملة أو غير مكتملة، فإن الميزة الهامة لشجرة البحث الثنائية هي تحقيق قاموس مرتب (أو خريطة). أي أن شجرة البحث الثنائية يجب أن تمثل أو تعبر بشكل شجري أو هرمي hierarchically عن ترتيب لمفاتيحها، باستخدام علاقة بين الأب والابن. وبتحديد أكبر، فإن التجول المرتب inoreder traversal لجميع عقد شجرة بحث ثنائية T يجب أن يقوم بزيارة جميع المفاتيح بالترتيب.

البحث 1

لتنفيذ العملية find(k) في قاموس D ممثل بشجرة بحث ثنائية T، ننظر إلى الشجرة على أنها شجرة قرار decision tree في هذه الحالة، السؤال الذي يطرح عند كل عقدة داخلية v هو فيما إذا كان مفتاح البحث k أصغر من أو يساوي المفتاح المخزن في العقدة v، المشار إليه بـ key(v). فإن كان الجواب (أصغر)، عندئذ يستمر البحث في الشجرة الفرعية اليسارية، أما إن كان الجواب (يساوي) عندئذ فإن البحث ينتهي بنجاح، وإن كان الجواب (أكبر) عندئذ يستمر البحث في الشجرة الفرعية اليمينية. أخيراً، إذا وصلنا إلى عقدة خارجية، فإن البحث ينتهي بشكل غير ناجح (نتيجة سلبية).



الشكل 7-1- (a) شجرة بحث ثنائية T تمثل قاموساً D،
(b) مسار تنفيذ find(76) بنجاح و find(25) بفشل.

يبين المقطع الخوارزمي 7-1 وصفاً مفصلاً لهذا الأسلوب، حيث k هو مفتاح البحث، وv هي عقدة من T.

تعيد الطريقة TreeSearch عقدة (موقع) w من الشجرة الفرعية T(v) من الشجرة T والتي جذرها v، حيث يحصل أحد الأمور التالية:

- w هي عقدة داخلية ويملك العنصر w مفتاحاً مساوي لـ k.
- w هي عقدة خارجية تمثل موقع k في حالة تجول ترتيب عبر T(v)، إلا أن k ليست مفتاحاً متضمناً في T(v).

وبالتالي، يمكن إنجاز الطريقة find(k) من خلال استدعاء TreeSearch(k,root()).

لتكن w هي العقدة من الشجرة T المعادة بهذا الاستدعاء. إذا كانت w هي عقدة داخلية، عندئذ نعيد قيمة w وإلا فإننا نعيد null.

ندرج خوارزمية البحث العودية

```
Algorithm TreeSearch(k,v):
    if T.isExternal(v) then      return v
    if k<key(v) then             return TreeSearch(k,T.left(v))
    else if k>key(v) then        return TreeSearch(k,T.right(v))
    return v                     {we know k=key(v)}
```

شجرة البحث الثنائية BST كنمط بيانات مجرد

مجموعة من عناصر البيانات:

الشجرة الثنائية التي يكون فيها من أجل كل عقدة x:

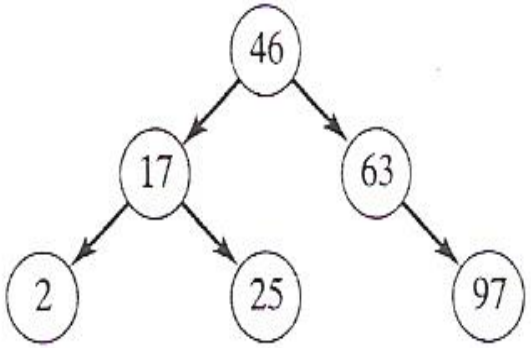
القيمة في الابن الأيسر لـ x أصغر أو تساوي القيمة في x التي هي أصغر أو تساوي القيمة في الابن الأيمن لـ x.

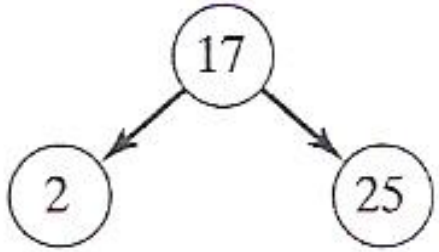
العمليات الأساسية:

- بناء شجرة بحث ثنائية BST فارغة.
- تحديد فيما إذا كانت شجرة البحث الثنائية BST فارغة.
- البحث ضمن شجرة البحث الثنائية BST عن قيمة معطاة.
- حشر عنصر جديد في شجرة البحث الثنائية BST مع المحافظة على خاصية شجرة البحث الثنائية BST.
- حذف عنصر من شجرة البحث الثنائية BST مع المحافظة على خاصية شجرة البحث الثنائية BST.

التجول عبر شجرة البحث الثنائية BST وزيارة كل عقدة مرة واحدة فقط، على الأقل أحد عمليات المرور يدعى التجول الترتيبي inorder traversal يجب أن يزور القيم في العقد بترتيب تصاعدي.

مثال عن البحث ضمن شجرة البحث الثنائية





نفرض أننا نريد البحث ضمن الشجرة السابقة عن القيمة 25. نبدأ بالجذر وبما أن 25 اصغر من القيمة في الجذر أي 46، فإننا نستنتج أن القيمة المرغوبة فد تكون موجودة في الجزء الذي على يسار الجذر وبالتالي تصبح العملية هي البحث ضمن الشجرة الفرعية اليسارية والتي جذرها هو 17.

نتابع الآن عملية البحث بمقارنة القيمة المراد البحث عنها 25 مع القيمة في جذر الشجرة الفرعية وبما أنها أكبر منها وبالتالي يجب أن يتم البحث في الجزء الذي على يمين العقدة الجذر أي:

باختبار القيمة في الجذر في الشجرة الفرعية المؤلفة من عقدة واحدة نصل إلى إيجاد العقدة المطلوبة.

وعندما يصل البحث إلى ورقة خارجية ولاتملك القيمة المبحوث عنها عندها نصل لنتيجة البحث السلبيه القسمة غير موجوده.

مثال عن البحث ضمن شجرة البحث الثنائية

وبالتالي يمكن تعديل التعريف السابق للصنف BST لتعريف التابع ( )Search وذلك بإضافة التصريح التالي ضمن القسم public للصنف:

```
bool Search(const DataType & item) const;
```

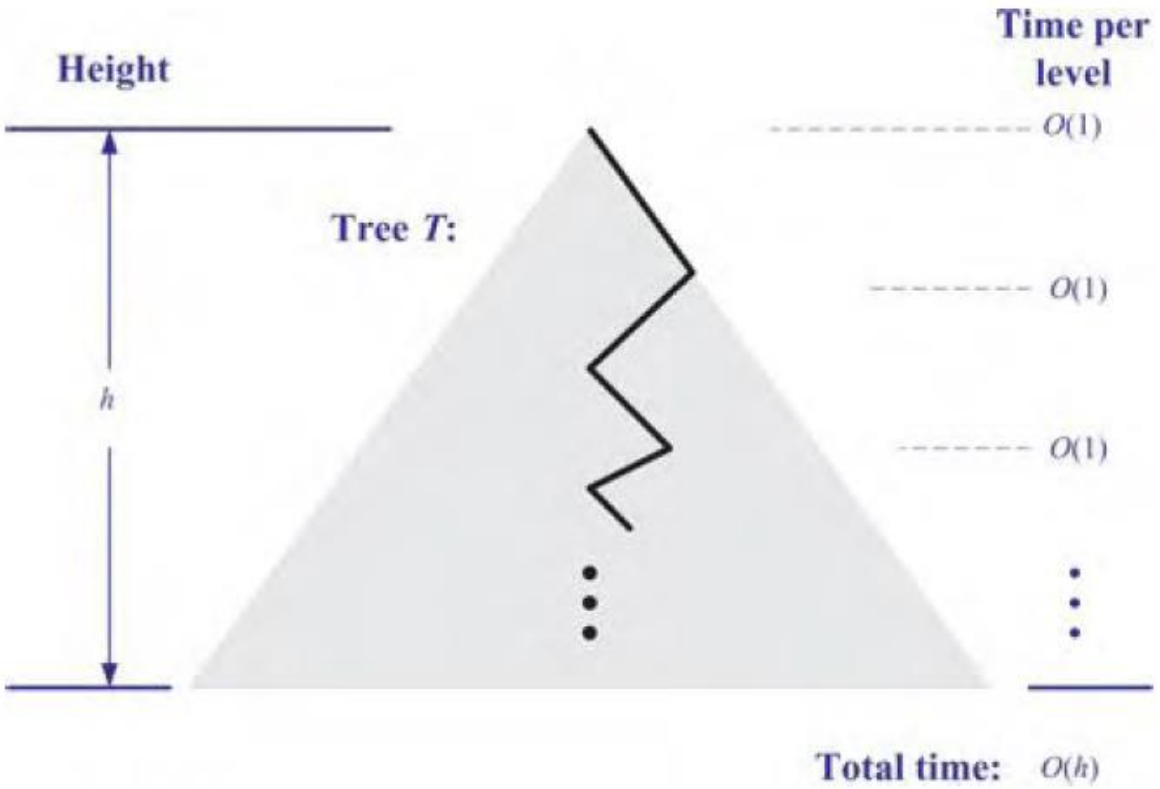
ومن ثم إضافة الشيفرة التالية إلى نهاية الملف:

```
template <typename DataType>
bool BST<DataType> :: Search(const DataType & item) const
{   BinNodePointer locptr = root;   bool found = false;
 for ( ; ; ) {    if (found || locptr == 0) break;
  if (item < locptr->data)         // descend left
   locptr = locptr->left;
  else if (item > locptr->data)   // descend right
   locptr = locptr->right;
  else              found = true;   // item found
}
 return found;
}
```

حيث يبدأ المؤشر locptr عند جذر شجرة البحث الثنائي ومن ثم ينتقل إلى الرابط اليميني أو اليساري للعقدة الحالية بحسب كون العنصر الذي نبحث عنه أصغر أو أكبر من القيمة المخزنة في العقدة. تستمر هذه العملية حتى يتم إيجاد العنصر المرغوب أو يصبح locptr صفرياً مشيراً إلى شجرة فرعية فارغة والتي تعني أن العنصر ليس في الشجرة.

Analysis of Binary Tree Searching

تحليل البحث في شجرة ثنائية 1



الشكل 7-2- كلفة البحث على شجرة ثنائية.

إن القيمه المقدمة الخوارزمية السابقة لتنفيذ العملية findAll(k) هى زمن O(h+s)، حيث s عدد القيم التي يتم إيجادها (إعادتها). إلا أن هذه الطريقة أكثر تعقيداً بقليل وستدرسها لاحقاً.

إن تحليل زمن تنفيذ الحالة الأسوأ لعملية البحث في شجرة بحث ثنائية T يعتبر أمراً بسيطاً. فالخوارزمية TreeSearch هي خوارزمية عودية وتنفذ عدداً ثابتاً من العمليات البسيطة في كل عملية استدعاء عودي.

يتم تنفيذ كل استدعاء عودي للطريقة TreeSearch على إبن من أبناء العقدة السابقة. أي أن، الطريقة TreeSearch يتم استدعاؤها على عقد مسار من T يبدأ عند الجذر وينزل هبوطاً مستوى واحد في كل مرة. وبالتالي، عدد هذه العقد محدد بـ h+1، حيث h هو ارتفاع T.

بكلام آخر، بما أننا ننفق زمناً مقداره O(1) عند كل عقدة في البحث، فإن الطريقة find تنفذ على قاموس D في زمن O(h)، حيث h هو ارتفاع شجرة البحث الثنائية T المستخدمة لتحقيق D. (يبين الشكل 7-2 هذه العملية).

Update Operations(Insertion)

طرائق التعديل (الحشر)

تتيح أشجار البحث الثنائية تحقيقات للعمليات insert و remove باستخدام خوارزميات مباشرة تماماً.

الحشر Insertion : لنفرض شجرة ثنائية مكتملة T تتضمن عملية التعديل التالية:

- insertAtExternal(v,e): إضافة العنصر e في عقدة خارجية v وتوسيع v لتكون داخلية من خلال إعطائها عقدتين ابنتين خارجيتين فارغتين حيث نواجه حدوث خطأ إذا كانت v عقدة داخلية.

باستخدام هذه الطريقة، يمكن تنفيذ insert(k,x) من أجل قاموس تم تحقيقه بواسطة شجرة بحث ثنائية T من خلال استدعاء TreeInsert(k,x,root()) المبينة في المقطع 7-2.

Algorithm TreeInsert(k,x,v):

Input A search key k, an associated value, x, and a node of T.

Output A new node w in the subtree T(v) that stores the entry (k,x)

←TreeSearch(k,v)

if k=key(w) **then** {the key at w is equal to k, so recurse at a child}

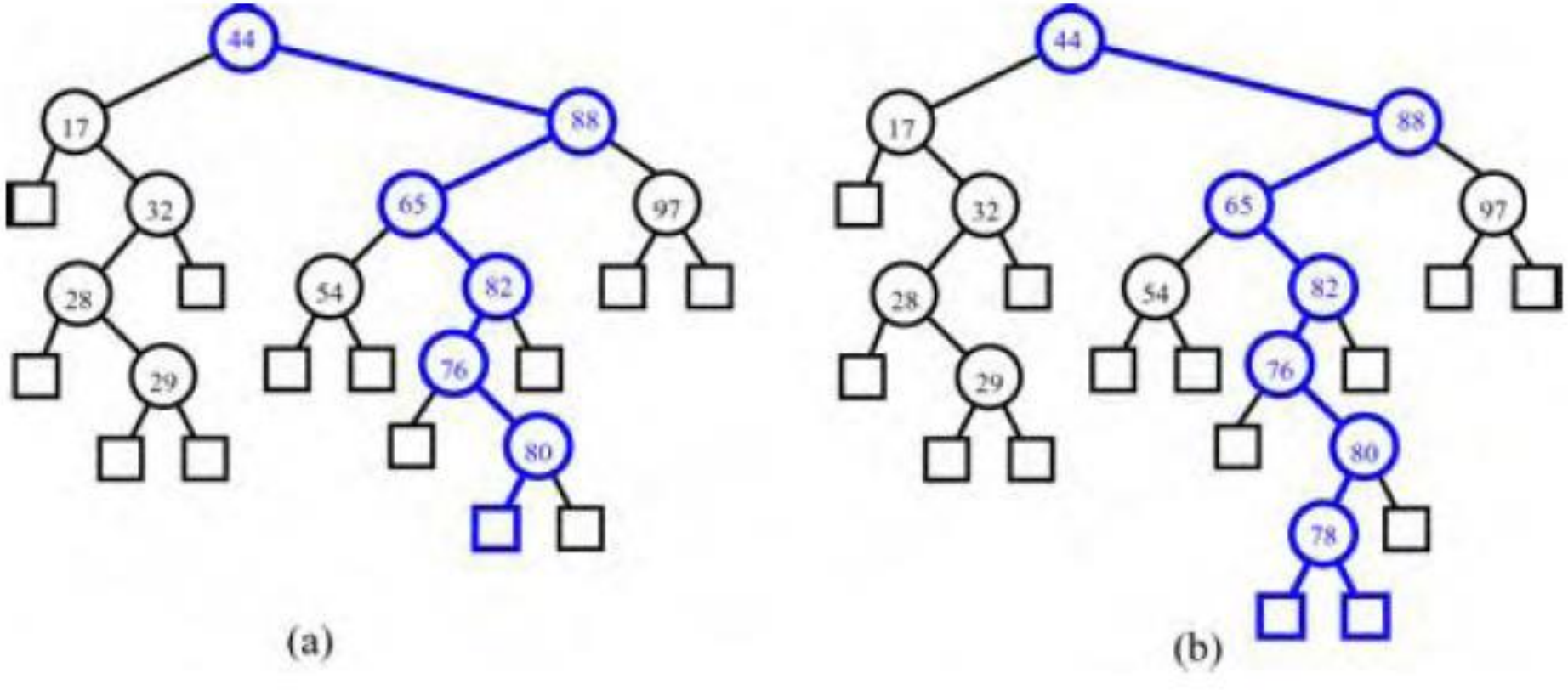
return TreeInsert(k,x,T.left(v)) {going to the right would be correct too}

T.insertAtExternal(w,(k,x)) {this is an appropriate place to put (k,x)} **return** w

Update Operations(Insertion)

طرائق التعديل (الحشر) 2

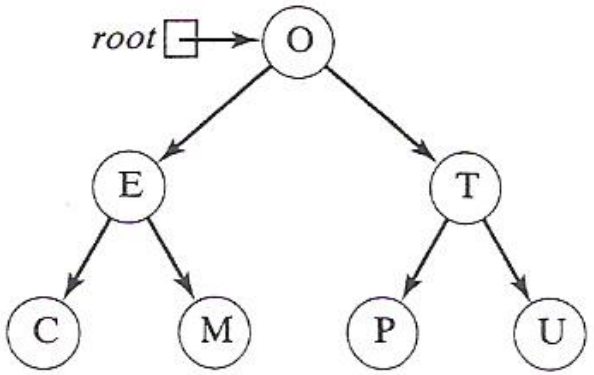
تقوم هذه الخوارزمية بتتبع مسار بدءاً من جذر T إلى عقدة خارجية، والتي توسع إلى عقدة داخلية جديدة لاحتواء القيمة الجديدة. يبين الشكل 7-3 مثالاً للحشر في شجرة بحث ثنائية.

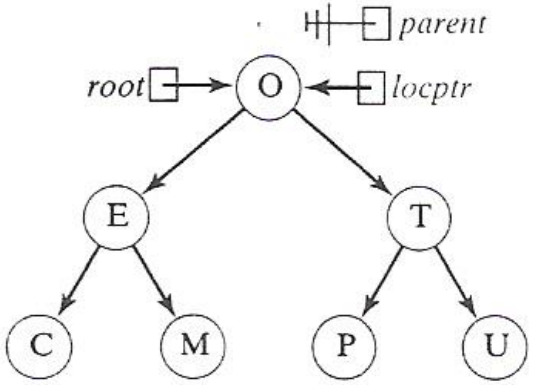


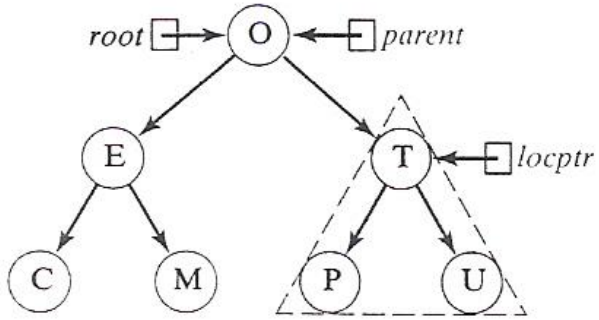
الشكل 7-3- حشر عنصر قيمته 78 في شجرة البحث المبينة في الشكل 7-1. إيجاد الموقع (a)

(b) الشجرة الناتجة.

مثال عن الحشر1







يمكن بناء شجرة البحث الثنائية بالاستدعاء المتكرر لتابع يقوم بحشر العناصر في شجرة البحث الثنائية التي هي فارغة في الحالة الابتدائية (root يشير إلى المؤشر الصفري). الطريقة المستخدمة لتحديد المكان الذي سيتم حشر العنصر فيه مشابهة لتلك المستخدمة في عملية البحث، وبالتالي نحتاج فقط لأن نعدل التابع ( )Search للمحافظة على مؤشر إلى العقدة الأب للعقدة المختبرة حالياً عندما ننزل عبر الشجرة باحثين عن مكان لحشر العنصر.

لتوضيح ذلك، نفرض أن شجرة البحث الثنائية المجاوره تم بناؤها:

و أننا نريد إضافة الحرف 'R'. نبدأ عند الجذرونقارن 'R' مع الحرف الموجود ضمنها:

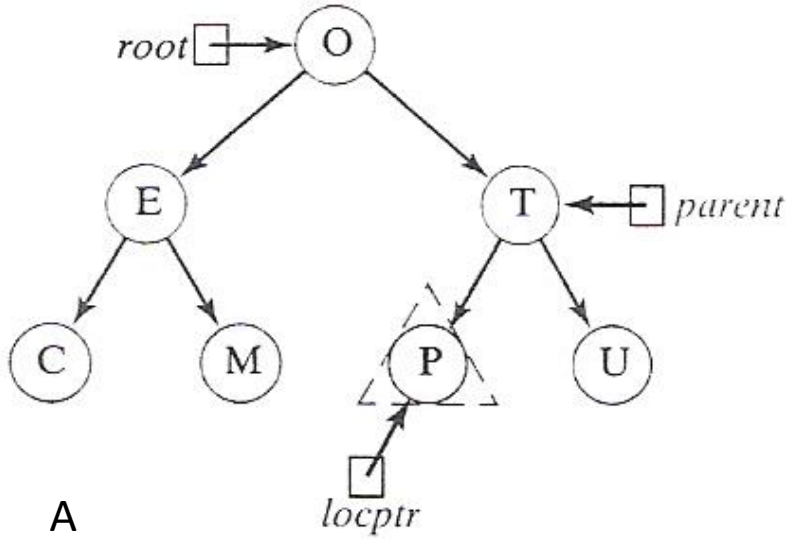
بما أن 'O'<'R' ننزل إلى الشجرة الفرعية اليمنى:

وبعد مقارنة 'R' مع 'T' المخزن في جذر هذه الشجرة الجزئية المشار إليها بواسطة locptr، نهبط إلى الشجرة الجزئية اليسرى بما أن 'R'<'T':

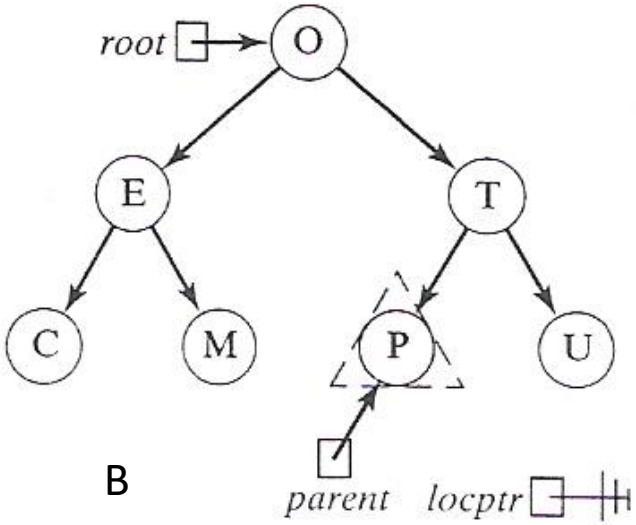
مثال عن الحشر2

وبما أن 'R'>'P' ننزل إلى الشجرة الجزئية اليمنى من هذه الشجرة الجزئية ذات العقدة الوحيدة الحاوية على 'P':

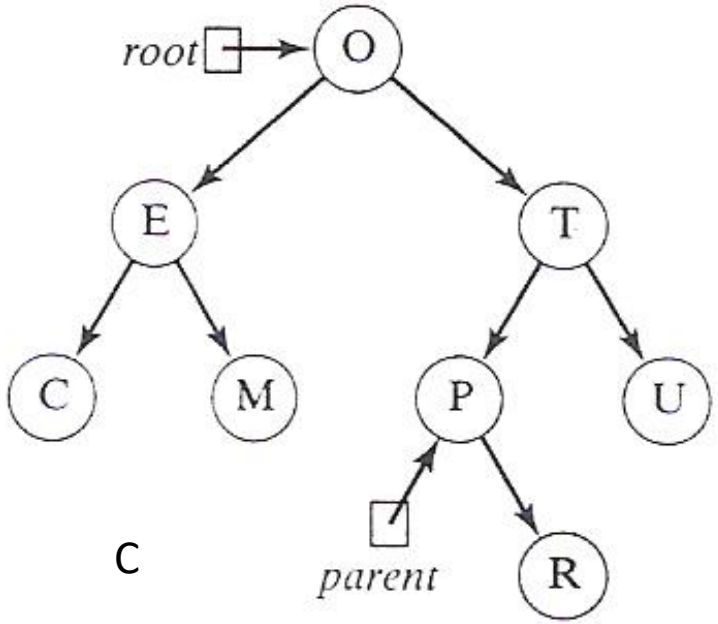
إن حقيقة أن الشجرة الجزئية اليمنى فارغة (أي locptr==null) تعني أن 'R' ليست موجودة في شجرة البحث الثنائية ويجب أن تحشركابن يميني لهذه العقدة الأب.



A



B



C

مثال عن الحشر3

يضاف التعريف على الصنف BST في قسم public : void Insert(const DataType & item); وتضمين التعريف التالي الى الملف:

```
template <typename DataType>
void BST<DataType> :: Insert(const DataType & item)
{  BinNodePointer    locptr = root,      parent = 0;  bool found = false;
 for ( ; ; )  {   if (found || locptr == 0) break;
   parent = locptr;
   if (item < locptr->data)   locptr = locptr->left;
   else if (item > locptr->data)  locptr = locptr->right;
   else         found = true;  }
 if (found)   cout << "Item already in the tree\n";
 else  {   locptr = new BinNode(item);
   if (parent == 0)              root = locptr;
   else if (item < parent->data)    parent->left = locptr;
   else                      parent->right = locptr;  }
}
```

مثال اختبارتسجيل الدخول إلى الحاسوب1

example – validating computer logins

ندرس مسألة تنظيم مجموعة من معرفات مستخدمي الحاسوب. في كل مرة يقوم المستخدم بتسجيل الدخول إلى النظام الحاسوبي بإدخال معرف المستخدم وكلمة السر، يجب على النظام اختبار صلاحية هذا المعرف وكلمة السر للتأكد من أن هذا المستخدم شرعي. بما عملية التحقق يجب أن تتم عدة مرات في اليوم، فمن المهم أن تنظم هذه المعلومات بحيث يمكن أن يتم البحث عنها بسرعة، وأكثر من ذلك فإنها يجب أن تنظم في بنية ديناميكية لأن المستخدمين الجدد يضافون بانتظام إلى النظام.

تصميم الحل - الأغراض اللازمة في هذه المسألة هي:

- معلومات المستخدم (المعرف وكلمة السروهي سلاسل محرفية): من النمط UserInfo.
- مجموعة من أغراض UserInfo تشكل BST.
- مجاري الدخل والخرج المعروفة.

سنستخدم شجرة بحث ثنائية من أجل مجموعة أغراض UserInfo لأنها يمكن أن تبحث بسرعة ولها بنية ديناميكية. وبالتالي فالعمليات الأساسية التي نحتاجها مضمنة في الصنف BST وهذه العمليات هي:

- بناء شجرة بحث ثنائية من أغراض UserInfo.
- البحث ضمن شجرة البحث الثنائية عن غرض ما من النمط UserInfo مدخل من لوحة المفاتيح.
- عرض رسائل تشير إلى أن المستخدم مقبول.

يمكن بناء خوارزمية الحل كما يلي:

example – validating computer logins

مثال اختبارتسجيل الدخول إلى الحاسوب2

algorithm for computer-login validation

1-open a stream to a file containing the valid user information.

2-create an empty BST of type UserInfo.

3-read the UserInfo objects from the file and insert them into the BST.

4-repeat the following until shutdown:

a. read UserInfo object.

b. search the BST for this object.

c. if it is found ,display a "valid" message

else display a "not valid " message.

وبالتالي يمكن صياغة الحل بلغة C++ كما يلي: الملف الرأسي BST.h:

```
#include <iostream>
using namespace std;
#ifndef BINARY_SEARCH_TREE
#define BINARY_SEARCH_TREE
template <typename DataType>
```

example – validating computer logins

مثال اختبارتسجيل الدخول إلى الحاسوب3

```
class BST {        private:   /*** Node structure ***/
class BinNode {
public:              DataType data;     BinNode * left,* right;        // BinNode constructors
 BinNode( )        {            left = right = 0;  }
BinNode(DataType item)  {            data = item;   left = right = 0;  }       };
typedef BinNode * BinNodePointer;
 public: BST( );
bool Empty() const;
.....   /*** Data members ***/
private:             BinNodePointer root;      bool Search(const DataType & item) const;
void Insert(const DataType & item);          }; // end of class template declaration
 //--- Definition of constructor
template <typename DataType>    inline BST<DataType>::BST()          { root = 0; }
```

example – validating computer logins

مثال اختبارتسجيل الدخول إلى الحاسوب4

```
//--- Definition of Empty()
template <typename DataType>
inline bool BST<DataType>::Empty() const  { return root == 0; }
template <typename DataType>
bool BST<DataType>::Search(const DataType & item) const
{          BinNodePointer locptr = root;          bool found = false;
 for ( ; ; ) {          if (found || locptr == 0) break;
  if (item < locptr->data)     locptr = locptr->left;
  else if (item > locptr->data)    locptr = locptr->right;
  else  found = true; }
 return found; }
template <typename DataType>
void BST<DataType>::Insert(const DataType & item)
```

example – validating computer logins

مثال اختبارتسجيل الدخول إلى الحاسوب5

```
{ BinNodePointer    locptr = root,       parent = 0;    bool found = false;
 for ( ; ; )  {   if (found || locptr == 0) break;
   parent = locptr;
   if (item < locptr->data)         locptr = locptr->left;
   else if (item > locptr->data)    locptr = locptr->right;
   else                             found = true; }
 if (found)    cout << "Item already in the tree\n";
 else  {    locptr = new BinNode(item);
   if (parent == 0) root = locptr;
   else if (item < parent->data) parent->left = locptr;
   else             parent->right = locptr;  }
}
#endif
```

example – validating computer logins

مثال اختبارتسجيل الدخول إلى الحاسوب 6

ملف الاختبارللصنف BST.cpp:

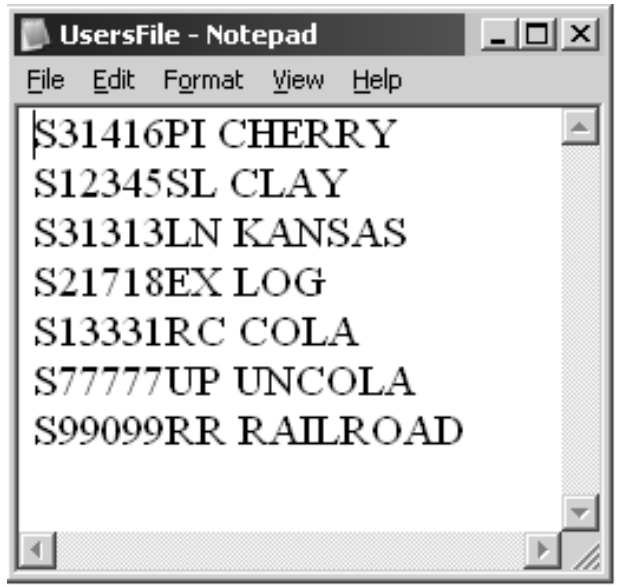
```
#include "BST"
  #include <iostream>
  #include <fstream>
  #include <string>
  using namespace std;
  class UserInfo {  public:    friend istream & operator>>(istream &in,UserInfo &user);
  bool operator ==(const UserInfo &user) {Return myId==user.myId && myPassword==user.myPassword; }

  bool operator <(const UserInfo & user)   {    return myId<user.myId; }
          bool operator >(const UserInfo & user) {return myId>user.myId;  }
  private:  string myId,myPassword; };
  istream & operator>>(istream &in,UserInfo &user)
          { in>>user.myId>>user.myPassword;  return in;
          }
```

example – validating computer logins

مثال اختبارتسجيل الدخول إلى الحاسوب 7

```
void main()
{       ifstream userFile("UsersFile");
        if (!userFile.is_open())       { cerr<<"Cannot open UserFile\n";    exit(-1);}
        BST<UserInfo> userTree;    UserInfo user;
        for( ; ; )   {userFile>>user;
                      if (userFile.eof()) break;     userTree.Insert(user);}
                      cout<<"Enter Q Q to stop processing.\n";
        for( ; ; )   {cout<<"\n User id & password:"; cin>>user;
          if (user.myId=="Q") break;
          if (userTree.Search(user))cout<<"Valid user \n";
          else      cout<<"Not a valid user \n";
        }
}
```

جرب تنفيذ هذا البرنامج على الملف UsersFile التالي:



طرائق التعديل 3

الحذف Removal

يعتبر تحقيق العملية remove(k) على قاموس D تم تحقيقه باستخدام شجرة بحث ثنائية T أمراً أكثر تعقيداً بقليل، وذلك لأننا لانرغب بإنشاء أي ثقوب أو فجوات في الشجرة T. نفرض في هذه الحالة، أن أي شجرة ثنائية مكتملة تدعم عملية التحديث الإضافية التالية:

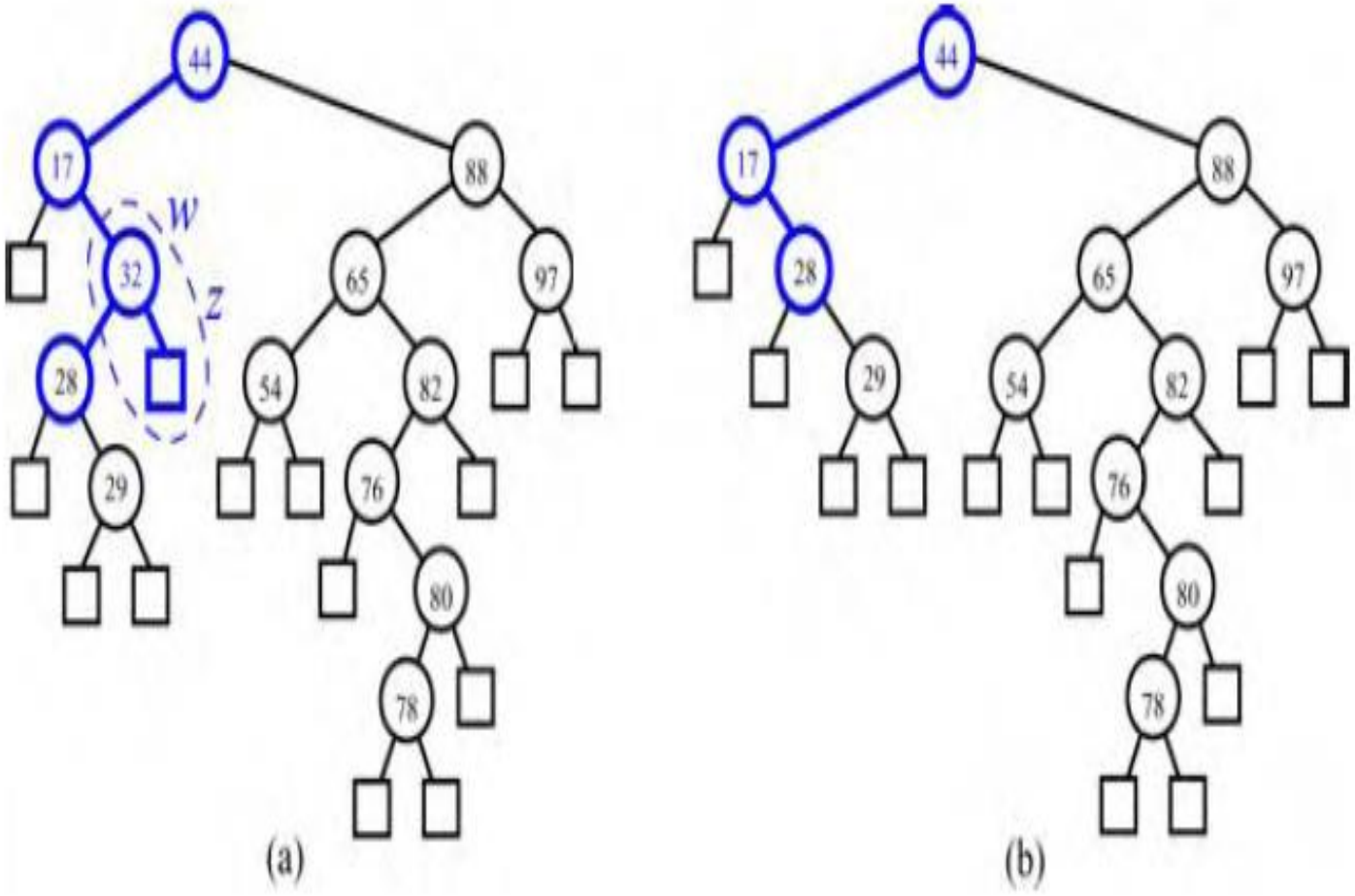
- (والتي تحذف عقدة خارجية v ووالدها، وتستبدل أب العقدة vبأخ هذه العقدة. يحصل خطأ إذا كانت v ليست خارجية.

إذا أعطينا هذه العملية، نبدأ تحقيقنا للعملية remove(k) للقاموس كنمط بيانات مجرد باستدعاء TreeSearch(k,T.root()) على T لإيجاد العقدة من T التي تخزن عنصراً ذو مفتاح يساوي k. إذا أعاد TreeSearchعقدة خارجية، عندئذ ليس هناك عنصر ذو مفتاح k في القاموس D، ونعيد حينها القيمة null. أما إذا أعاد TreeSearch عقدة داخلية w بدلاً من ذلك، عندئذ تخزن w العنصر الذي نرغب بحذفه، ونميز حالتين (ذات صعوبة أكبر):

- إذا كان أحد ابناء العقدة w هو عقدة خارجية، ولتكن العقدة z، فإننا نقوم –بكل بساطة- بحذف w و z من T من خلال العملية removeExternal(z) على T. تقوم هذه العملية بإعادة تشكيل T من خلال استبدال wبأخ z، وإزالة كل من w و z من T (الشكل 7-4).

طرائق التعديل 4

Removal الحذف



الشكل 7-4- الحذف من شجرة البحث الثنائية المبينة في الشكل 7-3 حيث العنصر المراد حذفه (ذو المفتاح 32) مخزن في عقدة w ذات إبن خارجي (a قبل، b بعد)

طرائق التعديل 5

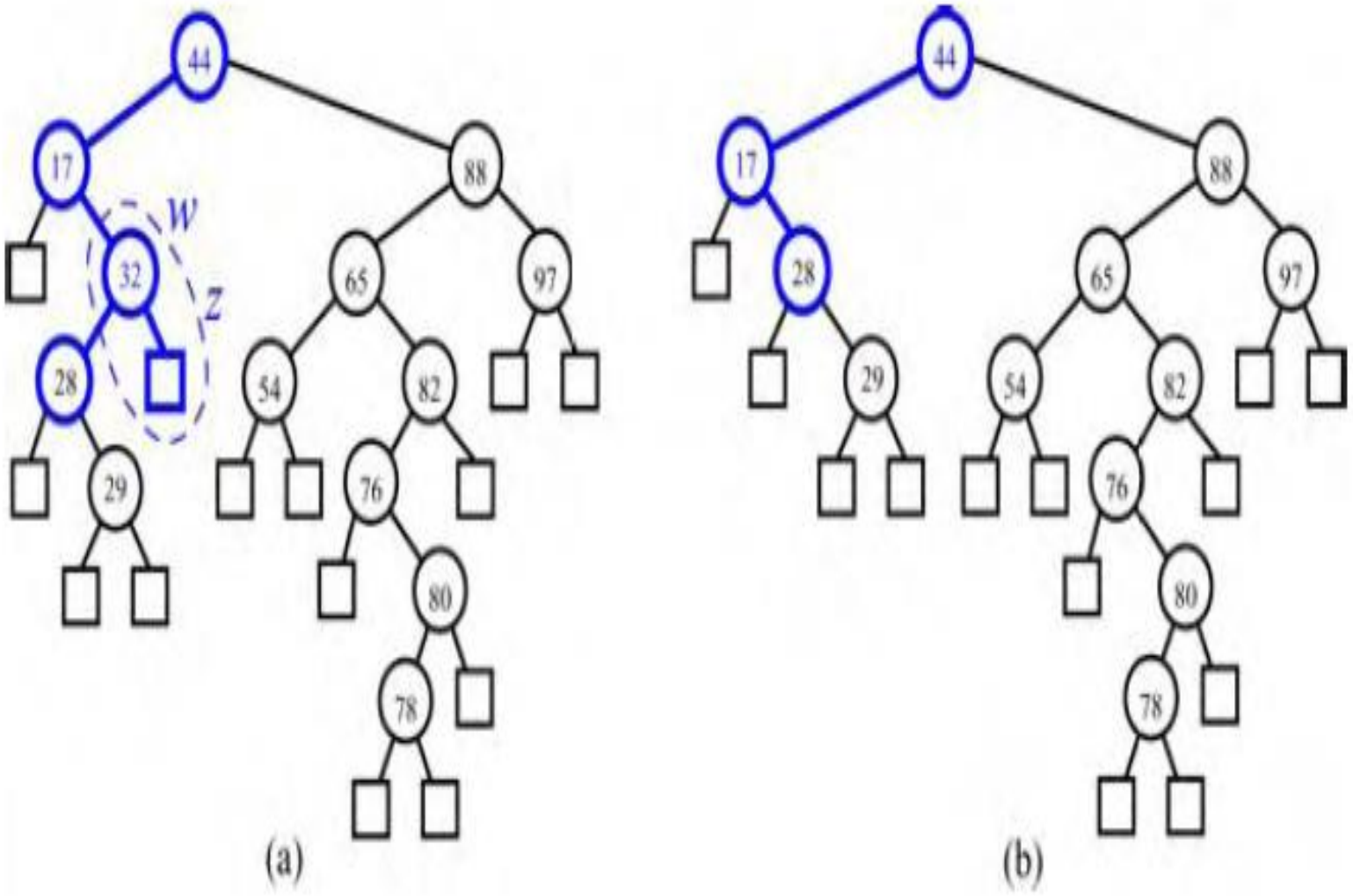
الحذف Removal

- إذا كان كلا من إبني العقدة w هي عقد داخلية، لانستطيع ببساطة حذف العقدة w من T لأن ذلك سيؤدي إلى إنشاء فجوة أو ثقب في T. بدلاً من ذلك فإننا نقوم بمايلي (الشكل 7-5):
  - نقوم بإيجاد أول عقدة داخلية y تلي w في تجول مرتب عبر T. تكون العقدة y هي العقدة الداخلية التي في أقصى اليسار من الشجرة الفرعية اليمينية لـ w، ويتم إيجادها من خلال الذهاب إلى الابن اليميني لـ w أولاً ومن ثم هبوط T من هناك، بتتبع الأبناء اليساريين. الابن اليساري x لـ y هو أيضاً عقدة خارجية تلي مباشرة العقدة w في التجول المرتب عبر T.
  - نقوم بحفظ العنصر المخزن في w في متحول مؤقت t، ونقل العنصر y إلى w. إن هذا التصرف له مفعول حذف العنصر السابق المخزن في w.
  - نقوم بحذف العقد x و y من T من خلال استدعاء removeExternal(x) على T. هذا التصرف يستبدل yبأخ x ويحذف كلاً من x و y من T.
  - نقوم بإعادة العنصر المخزن سابقاً في w والذي قمنا بحفظه في المتحول المؤقت t.

وكما في حالتي البحث والحشر، فإن خوارزمية الحذف تتجول عبر مسار من الجذر إلى عقدة خارجية، وتقوم ربما بنقل عنصر بين عقدتين من هذا المسار، ومن ثم تنفذ عملية removeExternal عند تلك العقدة الخارجية.

طرائق التعديل 6

الحذف



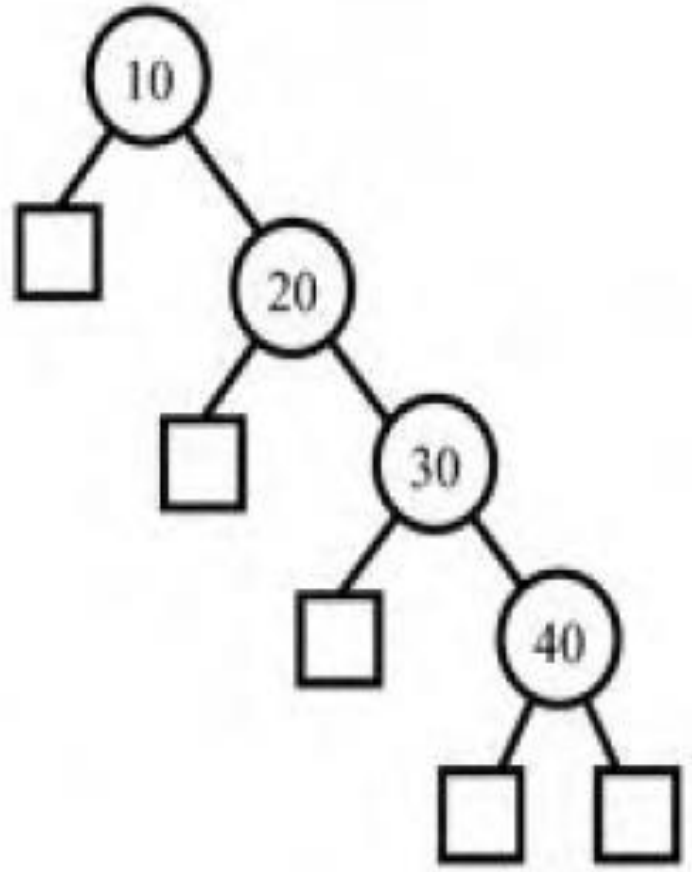
الشكل 7-5- الحذف من شجرة البحث الثنائية المبينة في الشكل 7-3 حيث العنصر المراد حذفه (ذو المفتاح 65) مخزن في عقدة w كلا إبنيها داخليين(a قبل، b بعد).

Performance of a Binary Search Tree

أداء شجرة بحث ثنائية

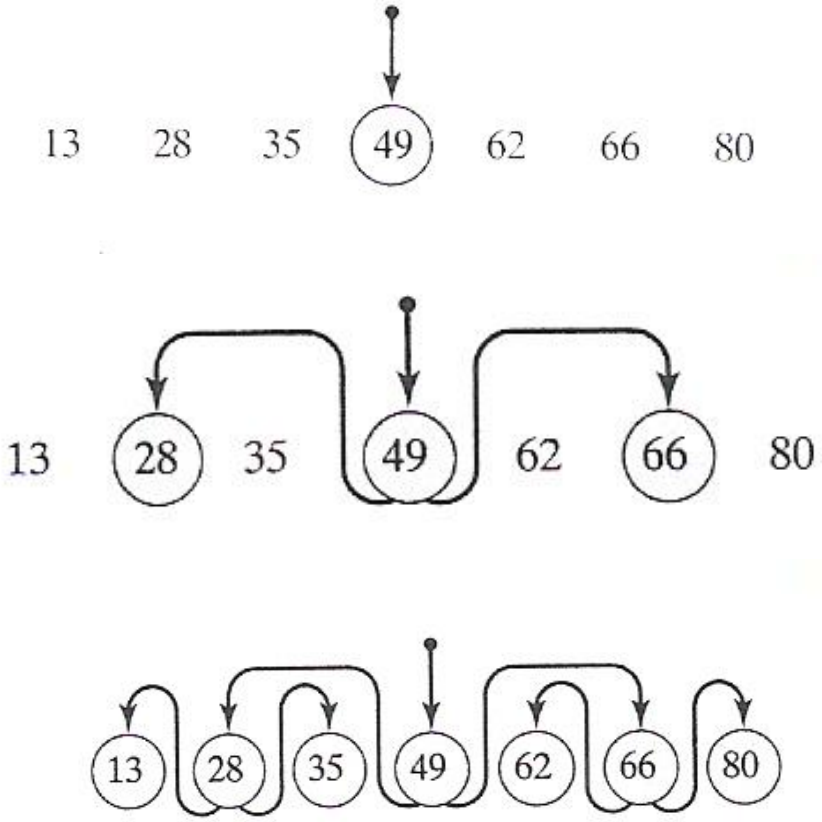
إن تحليل خوارزميات البحث، الحشر والحذف متشابه. نحن ننفق زمناً O(1) عند كل عقدة نزورها، وفي الحالة الأسوأ، عدد العقد التي تتم زيارتها تتناسب مع الارتفاع h للشجرة T. وبالتالي، في قاموس D تم تحقيقه بشجرة بحث ثنائية T فإن الطرائق find، insert، و remove تنفذ في زمن O(h)، حيث h هو ارتفاع T. وبالتالي، إن شجرة البحث الثنائية T هي تحقيق فعال أو مجدي لقاموس ذي n عنصر فقط إذا كان ارتفاع الشجرة صغيراً. في الحالة الأفضل، يكون ارتفاع T هو h=log(n+1) الأمر الذي ينتج أداء ذو زمن لوغاريتمي لجميع عمليات القاموس. في حين أنه، في الحالة الأسوأ، يكون ارتفاع T هو n، وفي هذه الحالة قد يكون أفضل أن نستخدم تحقيقاً للقاموس على شكل لائحة مرتبة.

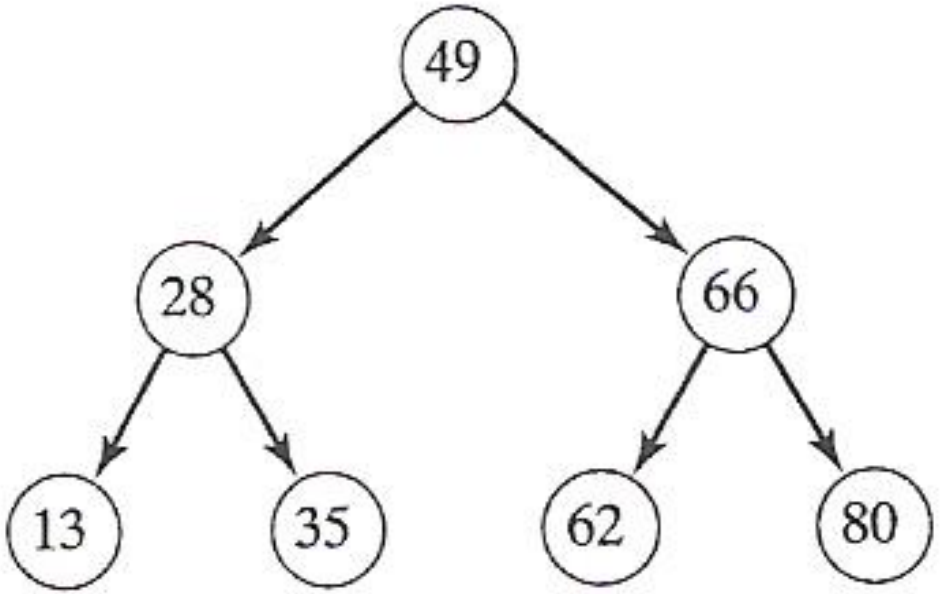
تحدث الحالة الأسوأ عندما نقوم بإدخال سلسلة من العناصر ذات مفاتيح بترتيب متزايد أو متناقص (الشكل 7-6).



الشكل 7-6- مثال لشجرة ثنائية ذات ارتفاع خطي.

إن تمثيل لائحة مرتبة كشجرة ثنائية يذكرنا بعملية البحث الثنائي جيث أن mad تمثل العقد على أن يتم التجوال بكلا الجزئين.



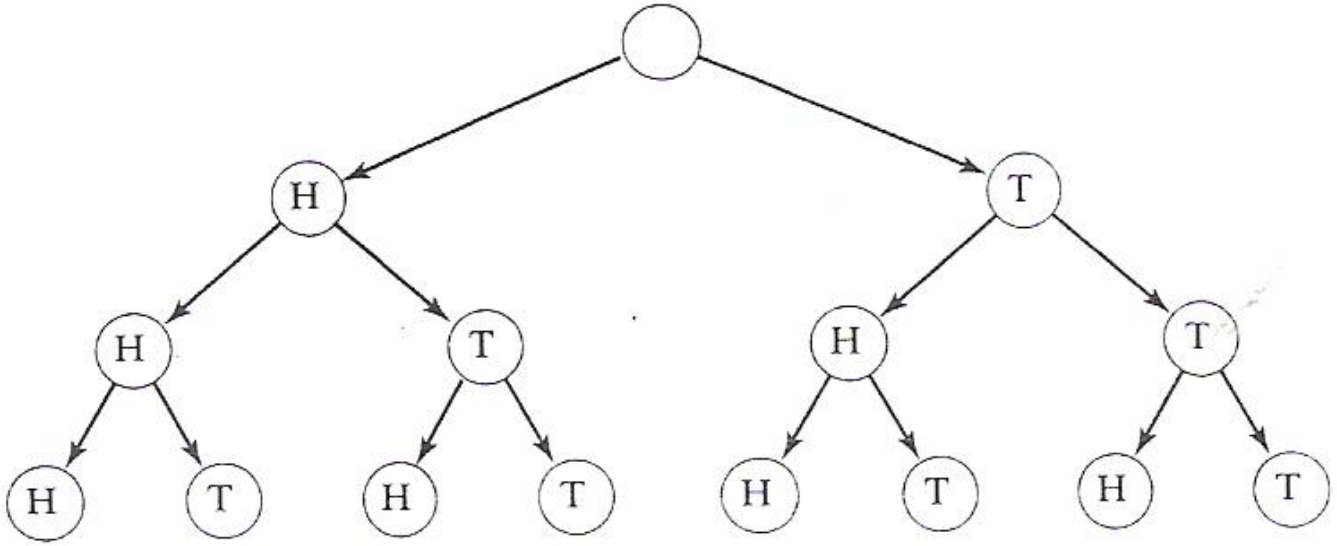


الشكل 7-7- يمثل شجرة البحث الثنائي.

Binary Search Tree Applications

تطبيقات بحث ثنائية 2

شجرة ثنائية لتمثيل الاحتمالات الممكنة لعملية رمي قطعة نقدية ثلاث مرات (وتطبق على التجارب والاختبارات التي يكون لها احتمالين ممكنين (مثل off أو on، 0 أو 1، false أو true، down أو up).

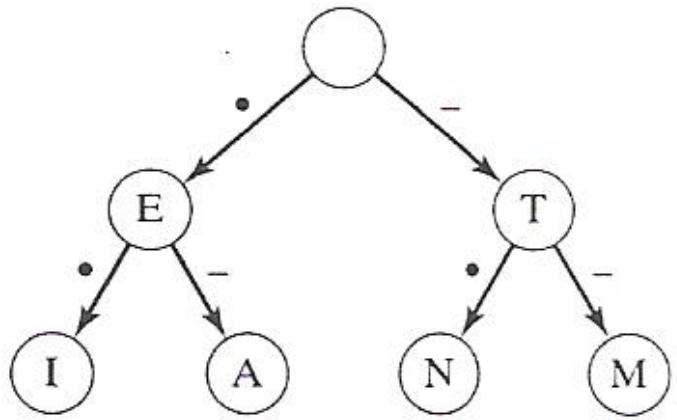


يمثل كل مسار بدءاً من الجذر وحتى إحدى الأوراق ناتجاً محتملاً مثل THT (نقش ثم طرة ثم نقش) كما هو مبين في المخطط.

الشكل 7-8- يمثل شجرة البحث الثنائي رمي قطعة نقدية ثلاث مرات.

في مسائل التشفير مثل تشفير و فك تشفير coding and decoding الرسائل المرسلة باستخدام شيفرة مورس Morse code، وفي هذه الطريقة يتم تمثيل المحارف كتتال من النقط والشرطات كما هو مبين في الجدول التالي:

A ·—	M ——	Y —·——
B —···	N —·	Z ——··
C —·—·	O ———	1 ·————
D —··	P ·——·	2 ··———
E ·	Q ——·—	3 ···——
F ··—·	R ·—·	4 ····—
G ——·	S ···	5 ·····
H ····	T —	6 —····
I ··	U ··—	7 ——···
J ·———	V ···—	8 ———··
K —·—	W ·——	9 ————·
L ·—··	X —··—	0 —————



الشكل 7-9- يمثل جزء من شيفرة مورس .

تستخدم العقد في الشجرة الثنائية لتمثيل المحارف وكل قوس من عقدة إلى أبنائها تميز إما بنقطة أو شرطة بحسب فيما إذا كانت تقود إلى ابن يميني أو ابن يساري على التوالي والتالي فإن أجزاء الشجرة الخاصة بشيفرة مورس هي:
تتالي النقط والشرطات المميز للمسار من الجذر إلى عقدة ما يمثل شيفرة مورس لذلك الحرف

تطبيق على الأشجار الثنائية: شيفرة هوفمان1

application of binary trees: Huffman codes

تستخدم الأشجار الثنائية في العديد من مسائل التشفيروفك التشفير.

شيفرة مورس، التي تمثل كل حرف بتتال من النقط والشرطات. تستخدم شيفرة مورس متتاليات متغيرة الطول بخلاف ترميزات الـ ASCII، EBCDIC، UNICODE التي يكون فيها طول الرمز هو نفسه لكل المحارف.

ندرس في هذه الفقرة طريقة أخرى للترميز هي شيفرة هوفمان Huffman code والتي تستخدم رموزاً بأطوال متغيرة.

الفكرة الأساسية في طرق التشفير متغير الطول هي استخدام شيفرات أقصر للأحرف الأكثر استخدماً، وشيفرات أطول للأحرف الأقل استخداماً. على سبيل المثال 'E' في شيفرة مورس هي عبارة عن نقطة واحدة، في حين أن 'Z' ممثلة بالشكل (.- -.).

إن الهدف هو تقليل الطول المتوقع لشيفرة الحرف، هذا يقلل عدد البتات الواجب إرسالها عند نقل الرسائل المشفرة. إن هذه الطرق في الترميز المتغيرة الطول مفيدة عند ضغط البيانات لأنها تقلل عدد البتات الواجب تخزينها

لتوضيح المسألة بشكل أدق، نفرض أننا أعطينا مجموعة المحارف $\{C_1, C_2, \ldots, C_n\}$ وأن أوزاناً محددة أرفقت مع هذه المحارف $w_1, w_2, \ldots, w_n$، حيث w_i هي الوزن المرفق بالمحرف C_i وهو يدل على تكرار استخدام هذا المحرف في الرسائل المراد تشفيرها. إذا كانت $l_1, l_2, \ldots, l_n$ هي أطوال الشيفرات الخاصة بالأحرف $C_1, C_2, \ldots, C_n$ على التوالي، عندئذ فإن الطول المتوقع لشيفرة أي من الأحرف تحسب كما يلي:

$$\exp ectedlength = w_1 l_1 + w_2 l_2 + \ldots\ldots + w_n l_n = \sum_{i=1}^{n} w_i l_i$$

تطبيق على الأشجار الثنائية: شيفرة هوفمان2

كمثال بسيط، لندرس الأحرف الخمسة A,B,C,D,E ونفرض أنها تحصل بالأوزان التالية:

character	A	B	C	D	E
weight	0.2	0.1	0.1	0.15	0.45

في شيفرة مورس إذا استبدلنا النقطة بصفروالشرطة بواحد فإن هذه المحارف مشفرة كما يلي:

character	A	B	C	D	E
code	01	1000	1010	100	0

وبالتالي فإن الطول المتوقع لكل من هذه الأحرف الخمسة في هذه الطريقة هو:

$$0.2\times2+0.1\times4+0.1\times4+0.15\times3+0.45\times1=2.1$$

قابلية فك التشفير مباشرة immediate decidability:

الميزة المفيدة الأخرى لبعض طرق التشفير هو أنها قابلة لفك التشفير مباشرة immediately decodable. وهذا يعني ليس هناك تتالي من البتات يمثل محرفاً يمكن أن يكون جزءاً سابقاً prefixمن تتالي أطول لأحرف أخرى. وبالتالي عند استقبال تتالي من البتات فهو شيفرة لحرف، فهو يمكن أن يفك تشفيره إلى ذلك المحرف مباشرة بدون انتظار فيما إذا كانت الخانات التي تلي تجعل منه شيفرة لمحرف آخر.

لاحظ أن شيفرة مورس السابقة ليست قابلة لفك التشفير مباشرة، لأنه، وعلى سبيل المثال، الشيفرة للمحرف E هي (0) وهي جزء سابق من شيفرة الحرف A أي (01)، وكذلك شيفرة الحرف D أي (100) هي جزء سابق من شيفرة الحرف B أي (1000).
تستخدم شيفرة مورس من أجل عملية فك التشفير خانة إضافية هي الفراغ للفصل بين المحارف.

شيفرة هوفمان Huffman codes:

يمكن استخدام الخوارزمية التالية المقدمة من قبل D.A.Huffman في عام 1952 لبناء طريقة تشفير قابلة للفك مباشرة وتملك طولاً أصغرياً متوقعاً لشيفرة كل حرف:

HUFFMAN'S ALGORITHM

1-initialize a list of one-node binary trees containing the weights $w_1,w_2,\dots,w_n$ one for each of the characters $C_1,C_2,\dots,C_n$.

2-do the following n-1 times:

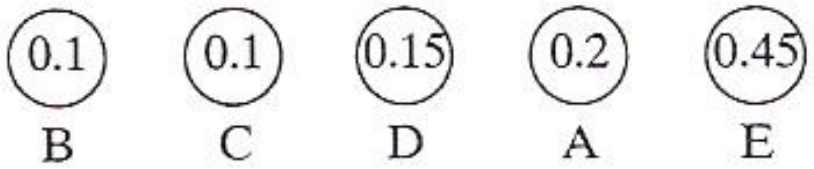
a-find two trees T' and T'' in this list with roots of minimal weights w' and w''.

b-replace these two trees with a binary tree whose root is $w'+w''$, and whose subtrees are T' and T'' , and label the pointers to these subtrees 0 and 1 respectively.

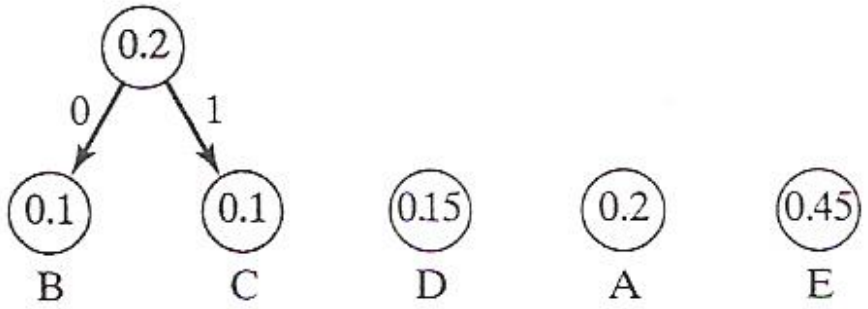
3-the code for character C_i is the bit string labeling a path in the final binary tree from the root to the leaf for C_i.

تطبيق على الأشجار الثنائية: شيفرة هوفمان4

كتوضيح لخوارزمية هوفمان، ندرس مجدداً المحارف A,B,C,D,E بالأوزان المحددة سابقاً. نبدأ ببناء لائحة من أشجار ثنائية مؤلفة من عقدة واحدة، واحدة لكل محرف:



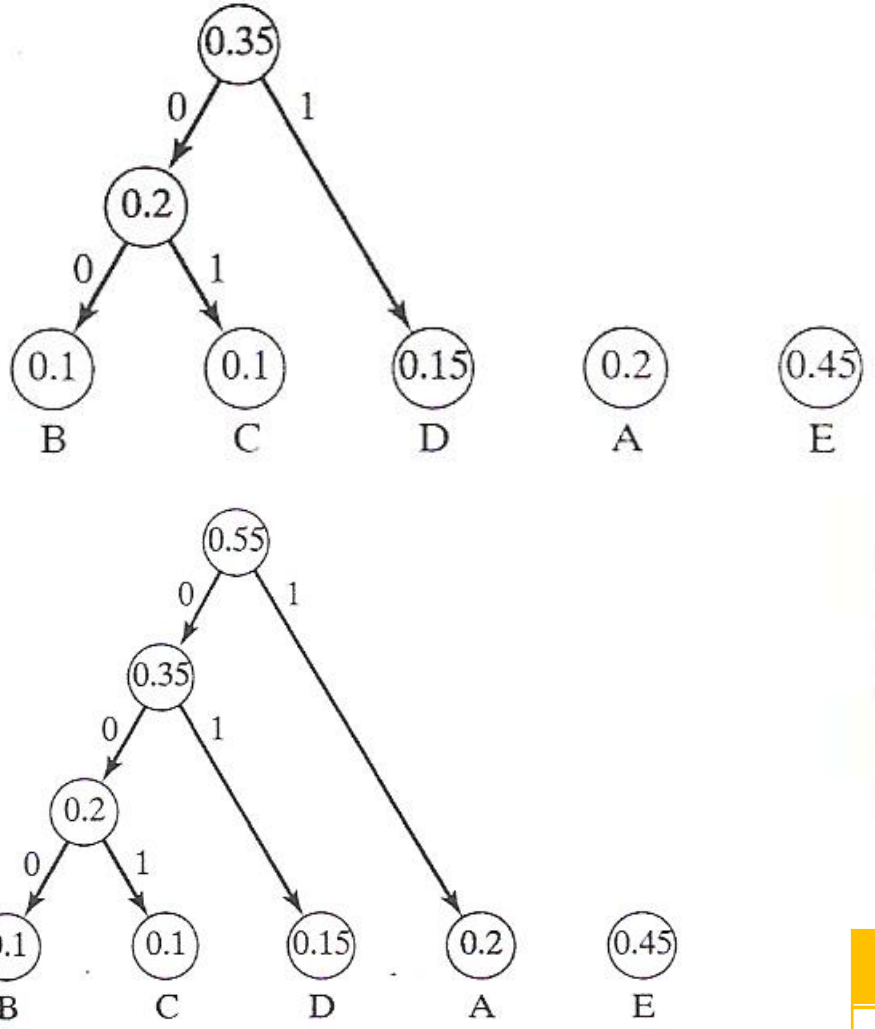
لشجرتين الأولى والثانية التي سيتم اختيارهما هي تلك الممثلة للمحارف B و C وذلك لأنها تملك الأوزان الأصغر. وبتجميع هاتين الشجرتين نشكل شجرة تملك الوزن 0.2=0.1+0.1 ولها شجرتين كشجرتين جزئيتين:



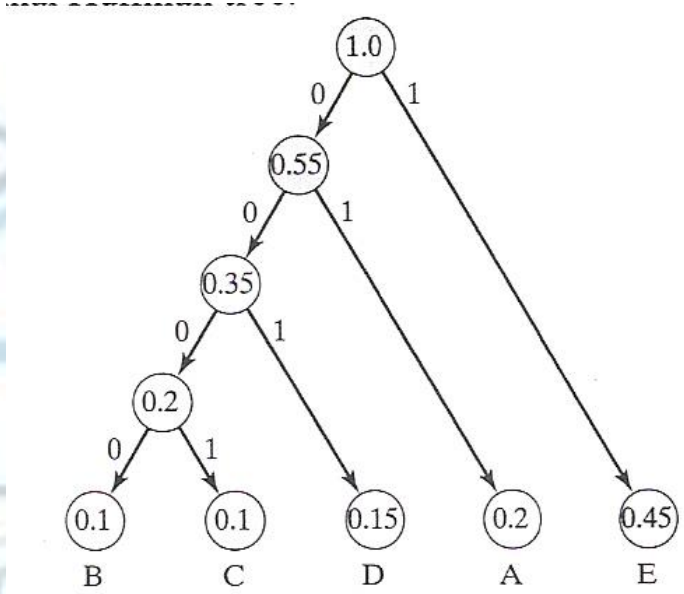
نختار من جديد من لائحة الأشجار الأربعة وزنين أصغريين، الأولى والثانية (أو الثانية والثالثة) ونقوم باستبدالهما كما في المرة السابقة بشجرة أخرى وزنها 0.35=0.15+0.2 ولها هاتين الشجرتين كشجرتين جزئيتين:

تطبيق على الأشجار الثنائية: شيفرة هوفمان5

application of binary trees: Huffman codes



وبالمثل نتابع فنحصل على الشجرة التالية:

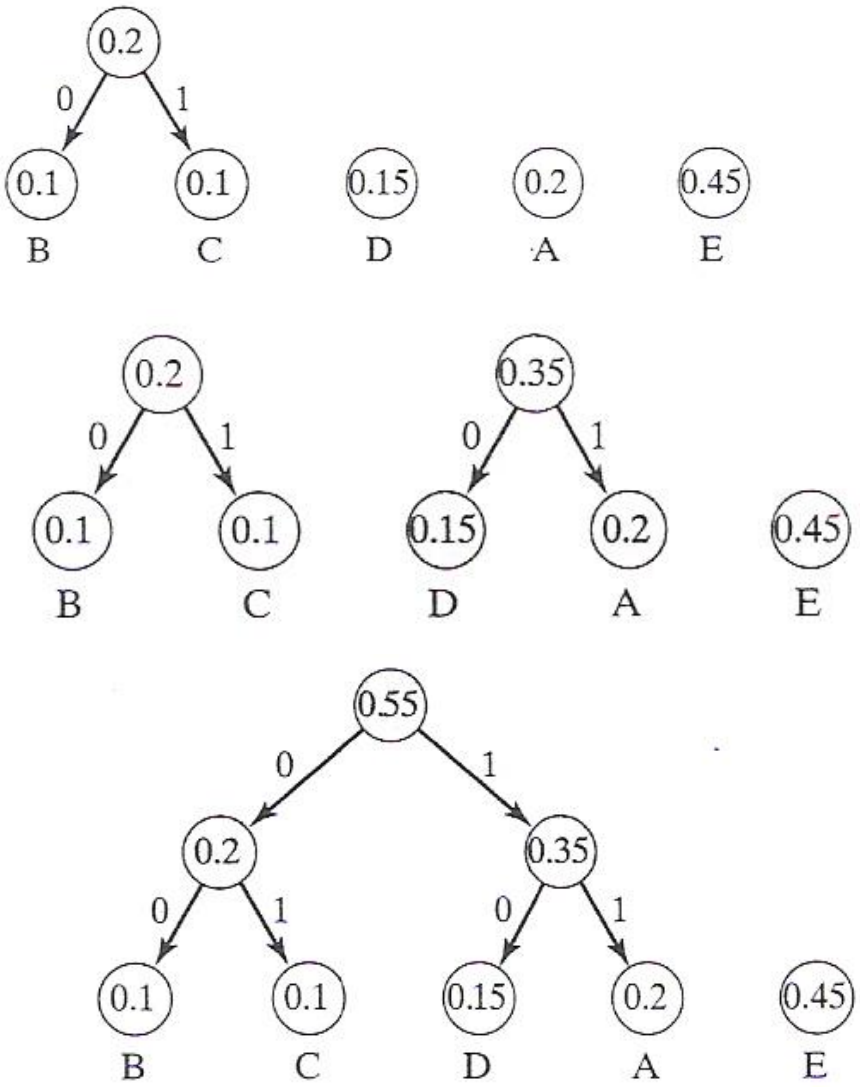


وبالتالي تكون شيفرة هوفمان الناتجة عن هذه الشجرة لهذه الأحرف كما يلي:

وكما بينا سابقاً فإن الطول المتوقع لشيفرة كل حرف هو2.1.

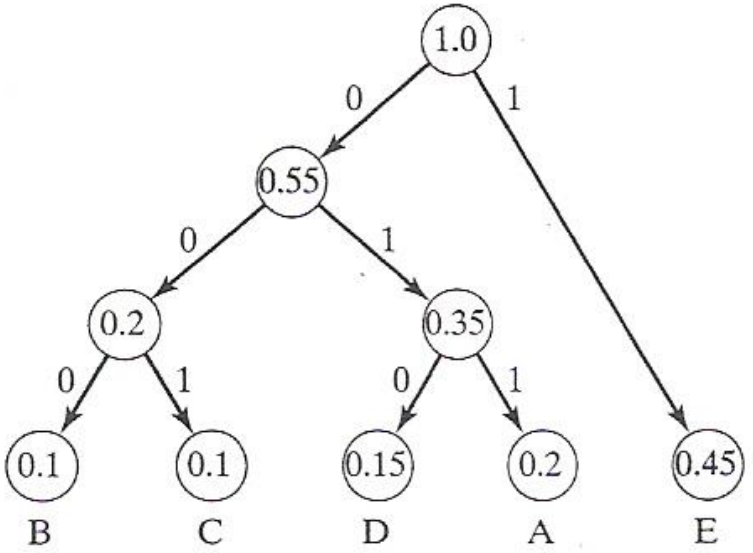
character	A	B	C	D	E
Huffman code	01	0000	0001	001	1

application of binary trees: Huffman codes



تطبيق على الأشجار الثنائية: شيفرة هوفمان6

يمكن تكوين شيفرة هوفمان بطريقة أخرى
نكتفي بعرضها من خلال الرسوم كما يلي:



وتكون الشيفرة الناتجة كما يلي:

character	A	B	C	D	E
Huffman code	011	000	001	010	1

تطبيق على الأشجار الثنائية: شيفرة هوفمان7

إن قابلية فك التشفير مباشرة واضحة في شيفرة هوفمان. كل محرف مرتبط بعقدة ورقة في شجرة هوفمان وهناك مسار وحيد من الجذر إلى كل ورقة. وبالتالي ليس هناك تتالي من البتات يتضمن شيفرة لمحرف يمكن أن يكون جزءاً سابقاً من تتالي أطول من البتات لمحرف آخر.

إن خوارزمية فك التشفير سهلة جداً بسبب ميزة قابلية فك التشفير مباشرة:

HUFFMAN DECODING ALGORITHM

1-initialize pointer p to the root of the Huffman tree.

2-while the end of the message string has not been reached , do the following:

a- let x be the next bit in the string.

b- if x=0 then set p equal to its left child pointer.

else set p equal to its right child pointer.

c-if p points to a leaf then

i-display the character associated with that leaf.

ii-reset p to the root of the Huffman tree.

تطبيق على الأشجار الثنائية: شيفرة هوفمان8

كتوضيح، لنفرض أن الرسالة التالية:

0	1	0	1	0	1	1	0	1	0

قد استلمت، وأن هذه الرسالة مشفرة باستخدام شجرة هوفمان الثانية المبنية سابقاً، يتبع المؤشر المسار التالي 010 من جذر هذه الشجرة يتم اكتشافه وجعله جذر الشجرة:

0	1	0	1	0	1	1	0	1	0
	D								

0	1	0	1	0	1	1	0	1	0
	D		E						

الخانة التالية 1 تقود مباشرة إلى المحرف E:

0	1	0	1	0	1	1	0	1	0
	D		E		A			D	

وهكذا إلى أن تصبح الرسالة من الشكل:

الملف الرأسي BST.h:

```
// C++ program to demonstrate insertion
// in a BST recursively
#include <iostream>
using namespace std;

class BST {
public:
 int data; BST *left, *right;

BST(); // Default constructor.
BST(int); // Parameterized constructor.
BST* Insert(BST*, int); // Insert function.
void Inorder(BST*); // Inorder traversal.
BST* deleteNode(BST*, int); /// Delete function.
};
```

الملف الرأسي BSTF.CPP

```
#include <iostream>
#include "BST.h"
using namespace std;
// Default Constructor definition.
BST::BST(): data(0) , left(NULL), right(NULL){ }
// Parameterized Constructor definition.
BST::BST(int value){data = value;left = right = NULL;}
// Insert function definition.
BST* BST::Insert(BST* root, int value)
{if (!root) {// Insert the first node, if root is NULL.
return new BST(value);}
// Insert data.
if (value > root->data) {/*Insert right node data, if the 'value‘
to be inserted is greater than 'root' .Process right nodes.
```

```
root->right = Insert(root->right, value);}
else if (value < root->data) {
/* Insert left node data, if the 'value‘ to be inserted is smaller
than 'root' node data. Process left nodes.*/

root->left = Insert(root->left, value);}
// Return 'root' node, after insertion.
return root;}
// Inorder traversal function.This gives data in sorted order.
void BST::Inorder(BST* root){
if (!root) {return;}
Inorder(root->left);cout << root->data << " ";
Inorder(root->right);}
/// Function that returns the node with minimum
```

```
// key value found in that tree
BST * minValueNode(BST * node)
{    BST * current = node;// Loop down to find the leftmost leaf
 while (current && current->left != NULL)
        current = current->left;  return current;}
// Function that deletes the key and returns the new root
BST* BST::deleteNode(BST* root, int value)
{    // base Case
    if (root == NULL)          return root;
    // If the key to be deleted is smaller than the root's key,
    // then it lies in left subtree
if (value < root->data)
{root->left = deleteNode(root->left, value);}
  // If the key to be deleted is greater than the root's key,
```

الملف الرأسي BSTF.CPP

```
// then it lies in right subtree
else if (value > root->data)
{ root->right= deleteNode(root->right, value);}
// If key is same as root's key, then this is the node to be del
 else { // Node with only one child or no child
 if (root->left == NULL)
{ BST * temp = root->right;free(root);return temp;}
        else if (root->right == NULL)
{ BST * temp = root->left;free(root); return temp; }
   // Node with two children:Get the inorder successor(smallest
  // in the right subtree)
BST * temp = minValueNode(root->right);
// Copy the inorder successor's content to this node
root->data = temp->data;
```

الملف الرأسي BSTF.CPP

```
// Delete the inorder successor
root->right= deleteNode(root->right, temp->data);    }
    return root;
}
#include "BST.h"
using namespace std;
int main() {BST b, *root = NULL; root = b.Insert(root, 50);
b.Insert(root, 30);   b.Insert(root, 20);   b.Insert(root, 40);
b.Insert(root, 70);   b.Insert(root, 60);   b.Insert(root, 80);
b.Inorder(root);
cout<<"del Node\n";   b.deleteNode(root, 20);
b.Inorder(root);      system("pause");
return 0;
}
```

الملف الرأسي Huffman.h:

الشيفرة التالية تبين حلاً لهذه المسألة باستخدام لغة C++:

الملف الرأسي Huffman.h:

انتهت محاضرة الأسبوع 2