درخت

داده های سلسله مراتبی توسط ساختمان داده درخت نمایش داده می شوند .انواع مختلفی از درخت ها وجود دارند که پرکاربردترین آنها درخت های دودویی هستند. درخت های BST و heap نوعی درخت دودویی هستند که خواص ویژه ای دارند و در الگوریتم های مختلف برای حل مسائل استفاده می شوند .

اصطلاحات

ساختمان داده درخت برای نمایش داده‌های سلسله مراتبی به کار می رود و چون شبیه درخت رسم می شود ساختار درختی نامگذاری شده است. البته ساختار درختی در مقایسه با درخت واقعی معمولا به صورت وارونه رسم می شود، یعنی ریشه درخت در بالا و برگ های آن در پائین قرار می گیرند .

به طور کلي یک درخت مجموعه ای از گره هاست که از طریق پیوندهایی با هم در رابطه هستند. هر گره دارای داده مرتبط و مجموعه ای از گره های دیگر است .

در نظریه گراف یک درخت یک گراف متصل بدون دور است .

گره

داده ها دردرخت در ساختاری به نام گره (node) قرار دارند. هر گره حاوی اطلاعات و پیوند هایی به دیگر گره های درخت است .

شاخه

خطوطی که گره ها را در درخت به هم متصل می کنند شاخه (branche) نامیده می شوند .

والد و فرزند

گره ای که بلافاصله زیر یک گره قرار می گیرد فرزند (children) آن گره محسوب می شود. یک گره والد گره دیگر (parent) است اگر بلافاصله بالاتر از آن نزدیک تر به ریشه قرار داشته باشد .

گره ای که کلیه گره های سطوح پایین را به هم متصل می کند جد (ancestor) نامیده می شود .

ریشه

هر درخت گره خاصی به نام ریشه (root) دارد که کلیه گره های دیگر درخت در پایین آن قرار دارند. گره ریشه والدی ندارد. هر درخت تنها شامل یک گره ریشه است .

گره های همزاد

گره های همزاد (Sibling) گره هایی هستند که والد یکسانی دارند. به عبارت دیگر فرزندان یك گره با هم همزاد هستند .

درجه گره

تعداد فرزندان يك گره درجه (degree) آن گره ناميده مي‌شود.

درجه درخت

درجه درخت برابر ماکزیمم درجه گره‌ها در درخت است.

برگ

گره های بدون فرزند گره های پایانی (end-nodes) یا برگ (leaf) نامیده می شوند. درجه گره های برگ صفر است .

سطح

مجموعه گره هایی طول مسیر آنها تا ریشه یکسان است را سطح درخت (level) می نامند. اگر ریشه را در سطح يك فرض كنيم برحسب اينكه يك گره نسبت به ريشه در چه رديفی باشد شماره سطح می گيرد .

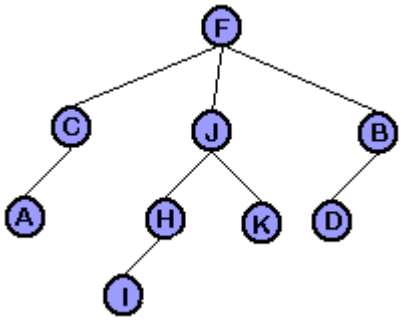
ارتفاع درخت

ارتفاع (height) درخت برابر با بیشترین سطح گره‌ها در درخت یا سطح دورترین برگ است. ارتفاع درختی که تنها گره ریشه را دارد یک است .

هر درخت خواص زیر را نمایش می دهند:

- دقیقا یک ریشه دارد.
- همه گره ها بجز ریشه دقیقا یک والد دارند.
- تنها یک مسیر از بین هردو گره وجود دارد.
- دور وجود ندارد یعنی مسیری وجود ندارد که از یک گره شروع شود و به خود آن ختم شود.
- درختی که دارای n گره است n-1 شاخه دارد.

مثال. در درخت زیر گره F ریشه است و گره هایC ، J و B فرزندان ریشه هستند. گره هایA ، I، Kو D برگ های درخت هستند. درجه درخت ۳ و ارتفاع آن 4 است .

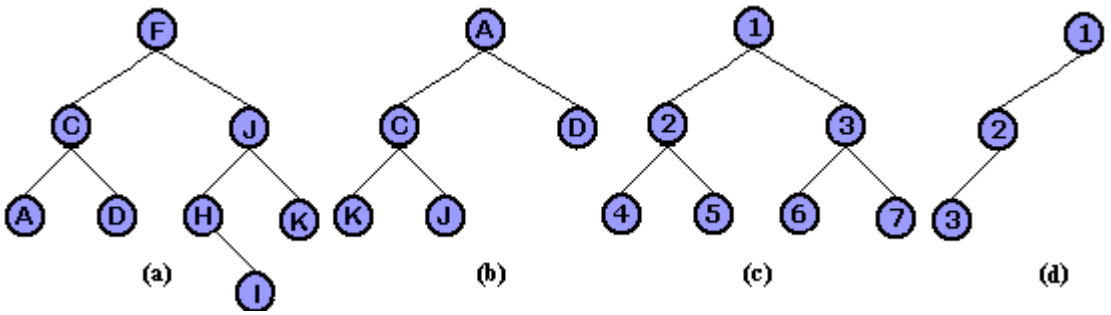


درخت دودوئی

احتمالا پرکاربردترین ترین نوع درختی، درخت دودویی (binary tree) است .درخت های دودویی یک پیاده سازی خاص از یک درخت m-ary است که در آن m=2 است یا به عبارت ساده تر هر گره حداکثر دو فرزند دارد که فرزند چپ و فرزند راست (یا زیردرخت چپ و زیر درخت راست) نامیده می شوند .

ترتیت قرار گرفتن گره ها در درخت دودویی کاملا اختیاری نیست. درحقیقت نحوه درج گره های جدید در درخت دودویی ساختار و کاربرد آنرا تعیین می کند .

مثال. درخت های زیر همگی دودویی هستند.



انواع درخت دودوئی

درخت دودوئی پر

یک درخت دودویي پر (full binary tree) درختی است که هر گره آن صفر یا دو فرزند دارد .
در درخت دودویی پر کلیه برگ ها در یک سطح هستند .
یک درخت دودویی پر به عمق h دارای 2^h-1 گره است.
در درخت دودویی پر کلیه برگ ها در یک سطح هستند.
تعداد گره‌های سطح i ام يك درخت دودوئی پر 2^{i-1} است.

مثال. درخت (c) در شکل فوق یک نمونه درخت دودویی پر است.

درخت دودوئی کامل

یک درخت دودویی کامل (complete binary tree) درختی است که همه سطوح آن به جز احتمالا آخرین سطح حداکثر گره ها را دارند و در سطح آخر گره ها از سمت چپ ظاهر می شوند. یعنی اگر ارتفاع درخت h باشد تعداد کل گره ها تا سطح h-1 برابر با 2^h-1 است و در سطح آخر اگر گره هایی وجود دارند باید از چپ به راست اضافه شوند.

مثال. درخت (b) در شکل فوق یک نمونه درخت دودویی کامل است.

درخت دودویی میزان

درخت دودویی میزان (balanced binary tree) درختی است که کلیه برگ ها حداکثر یک سطح با هم تفاوت دارند .

مثال. در شکل قبل کلیه درخت ها به جز درخت (d) میزان هستند .

نمایش درخت دودوئی

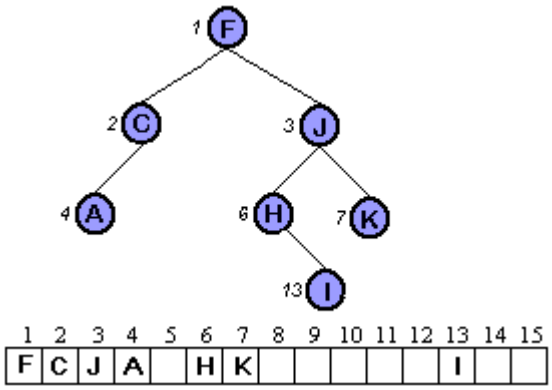
نمایش توسط آرایه

يك درخت دودوئی کامل با n گره را می توان در يك آرایه يك بعدی ذخیره کرد. برای این کار گره های درخت شماره گذاری می شوند. شماره گذاری به ترتیب از بالا به پایین و از چپ به راست انجام می شود و به هر گره شماره ای تعلق می گیرد. سپس گره با شماره i در خانه i ام آرایه قرار می گیرد .

وقتی درخت دودویی در آرایه نمایش داده می شود برای هر گره با اندیس: i

- اگر $i \neq 1$ باشد، والد i در $\lfloor i/2 \rfloor$ است و اگر $i=1$ باشد i ریشه است.
- اگر $2i \leq n$ باشد فرزند چپ i در $2i$ است و اگر $2i+1 \leq n$ باشد، فرزند راست i در $2i+1$ است.

مثال .

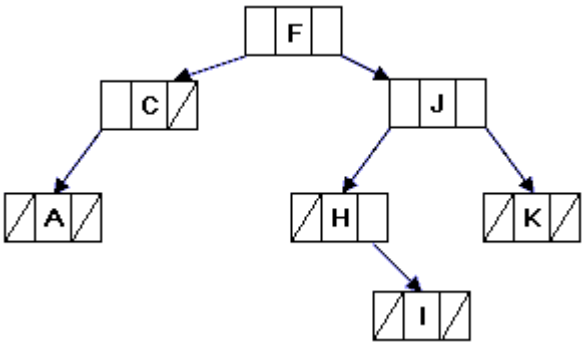


نمایش توسط لیست پیوندی

یک درخت دودویی را می توان توسط لیست پیوندی نمایش داد. به این صورت که برای هر گره درخت یك گره در لیست در نظر گرفته می شود. هر گره یك اشارهگر به فرزند چپ و یك اشارهگر به فرزند راست دارد .

```
typedef struct node *TreePionter;
typedef struct node {
  char Data;
  Treepointer Left,Right;};
```

مثال .



پیمایش درخت دودوئی

اغلب می خواهیم کلیه گره های درخت را بررسی کنیم. چند روش برای پیمایش وجود دارد که در آنها گره ها می توانند پردازش یا ملاقات شوند. هر روش وقتی روی یک درخت دودویی اجرا می شود ویژگی های مفیدی را در اختیار می گذارد .

سه روش معمول پیمایش درخت های دودویی روش هایpreorder ، postorder و inorder است که در آنها هر گره و فرزندانش به طور بازگشتی ملاقات می شوند .هر سه پیمایش از ریشه درخت شروع می شوند. تفاوت آنها در ترتیب ملاقات گره و فرزندانش است. در روش preorder ابتدا گره ملاقات شود سپس فرزندانش. در روش postorder ابتدا فرزندان سپس خود گره ملاقات می شود. در روش inorder گره مابین فرزندان چپ و راست خود ملاقات می شود .

پیمایشPreorder

در روش preorder محتوای گره ریشه قبل از فرزند چپ و راست ملاقات می شود. الگوریتم بازگشتی پیمایش preorder به صورت زیر است :

۱. پردازش ریشه

۲. پیمایش زیردرخت چپ به روش preorder
۳. پیمایش زیردرخت راست به روش preorder

تابع زیر با توجه به الگوریتم فوق پیاده سازی شده است:

```
void Preorder(TreePointer T){
If (T!=NULL) {
  Visit(T->Data);
  Preorder(T->Left);
  Preorder(T->Right);
}}
```

پیمایش از گره جاری T شروع می شود سپس فرزند چپ و بعد فرزند راست آن ملاقات می شود .Visit برای نمایش یا پردازش مقدار گره است و بستگی به هدف از پیمایش می تواند بسادگی دستور نمایش محتوای گره باشد. تابع با فراخوانی گره ریشه آغاز می شود .

پیمایشInorder

پیمایش inorder با ملاقات فرزند چپ گره جاری شروع شده سپس خود گره و بعد فرزند راست را ملاقات می کند. الگوریتم آن به صورت زیر است :

۱.پیمایش زیردرخت چپ به روش inorder
۲.پردازش ریشه
۳.پیمایش زیردرخت راست به روش inorder

تابع زیر مشابه تابع Preorder است با این تفاوت که ترتیب ملاقات گره تغییر کرده است:

```
void Inorder(TreePointer T){
If (T!=NULL) {
  Inorder(T->Left);
  Visit(T->Data);
  Inorder(T->Right);
}}
```

پیمایشPostorder

پیمایش postorder ابتدا محتوای زیردرخت چپ و سپس زیردرخت راست و در نهایت گره ریشه را ملاقات می کند. الگوریتم بازگشتی پیمایش postorder به صورت زیر است :

۱.پیمایش زیردرخت چپ به روش Postorder
۲.پیمایش زیردرخت راست به روش Postorder
۳. پردازش ریشه

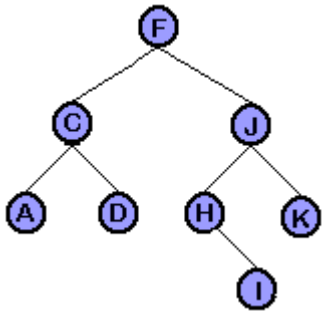
تابع زیر با توجه به الگوریتم فوق پیاده سازی شده است:

```
void Postorder(TreePointer T){
If (T!=NULL) {
  Postorder(T->Left);
  Postorder(T->Right);
  Visit(T->Data);
}}
```

پیمایش اول سطح

پیمایش اول سطح (breadth-first order) مشابه جستجوی اول سطح در گراف است و ابتدا سعی می کند نزدیک ترین گره به ریشه را ملاقات کند .پیمایش از سطح اول درخت آغاز شده، هر بار یك سطح از چپ به راست به طور کامل پیمایش می شود .

مثال. پیمایش های فوق روی درخت زیر انجام شده است:



Pre Order : FCADJHIK
In Order : ACDFHIJK
Post Order : ADCIHKJF
Breadth First Order : FCJADHKI

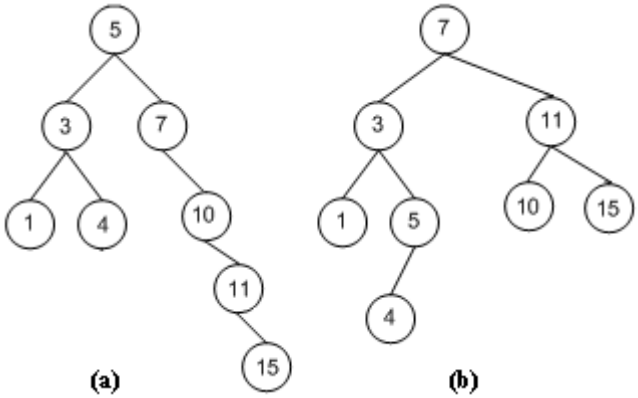
درخت جستجوی دودوئی

وقتي عمليات جستجو، حذف و اضافه مدنظر باشد درخت جستجوی دودوئی از تمام ساختارهای دیگر مناسب‌تر است .

یک درخت جستجوی دودویی (binary search tree) یا BST نوع خاصی از درخت دودویی است که اگر تهی نباشد خواص زیر را دارا است :

- هر گره يك مقدار منحصر بفرد دارد.
- کلیه مقادیر فرزندان زیردرخت چپ هر گره از مقدار خود گره کوچکتر هستند.
- کلیه مقادیر فرزندان زیردرخت راست هر گره از مقدار خود گره بزرگتر هستند.

مثال. درخت های زیر هر یک BST هستند .



جستجو درBST

جستجوی یک مقدار در BST از گره ریشه شروع می شود. آرگومان جستجو با مقدار گره مقایسه می شود. اگر یکسان باشند داده مورد نظر پیدا شده است در غیراینصورت اگر داده از مقدار گره کوچکتر باشد جستجو از زیردرخت چپ و اگر بزرگتر باشد جستجو از زیردرخت راست گره ادامه پیدا می کند .

به طور خلاصه الگوریتم جستجوی BST به صورت زیر بیان می شود .Item داده موردنظر است که در BST جستجو می شود. الگوریتم مقدار تهی یا گره ای که دنبالش هستیم را بر می گرداند .

```
Search ( TreePointer T, DataType Item )
Begin
  If (T = null) Then return null {tree is empty}
  Else If (Item = T.Data ) Then return T {item found}
  Else If (Item < T. Data and T.Left !=NULL ) Then Search(T.Left,.Data)
  Else If (Item > T.Data and T.Right !=NULL ) Then Search (T.Right, Data)
  End If
End
```

مثال. فرض کنید می خواهیم عدد ۱۰ را در درخت شکل (b) مثال قبل جستجو کنیم. جستجو از ریشه شروع می شود. عدد ۷ در ریشه است که کمتر از ۱۰ است .بنابراین اگر ۱۰ وجود داشته باشد باید در زیردرخت راست ریشه باشد. پس جستجو را از گره ۱۱ ادامه می دهیم. در حالت ۱۰ از عدد ۱۱ کمتر است بنابراین اگر ۱۰ در درخت باشد باید در زیردرخت چپ گره ۱۱ باشد. به سمت چپ گره ۱۱ حرکت می کنیم که گره ۱۰ است و گره ای که دنبالش می گشتیم را پیدا کردیم .

ممکن است که اگر گره ای که به دنبالش هستیم در درخت موجود نباشد. برای نمونه اگر بدنبال عدد ۹ هستیم ابتدا به همان طریق بالا عمل می کنیم. وقتی به گره ۱۰ می رسیم باید جستجو را از زیردرخت چپ ادامه دهیم ولی گره ۱۰ فرزند چپ ندارد بنابراین ۹ در درخت وجود ندارد .

با دقت در الگوریتم جستجو می توان مشاهده کرد که درهرمرحله تعداد گره هایی که باید بررسی شوند نصف می شوند. بنابراین چنین زمان اجرای الگوریتم $\log_2 n$ می شود. البته زمان جستجو به توپولوژی درخت بستگی دارد و در بهترین حالت $O(\log n)$ است، در بدترین حالت $O(n)$ می شود .

اگر یک BST به روش InOrder پیمایش شود مقادیر گره‌های درخت به صورت مرتب بدست می آید. زمان اجرای پیمایش روی درخت $O(n)$ است.

درج درBST

درج یک گره جدید در BST در دو مرحله انجام می گیرد. ابتدا داده جدید در BST طبق الگوریتمی که ذکر شد جستجو می شود سپس در محل خاتمه جستجو گره جدید اضافه می شود .

با فرض اینکه داده تکراری در درخت مجاز نیست، هنگام درج گره جدید دو شرط باید مدنظر باشد :

- درج در یک درخت خالی. در این حالت گره ای که درج می شود ریشه درنظر گرفته می شود.
- درج در یک درخت غیر خالی. درخت باید جستجو شود همانطور که در بالا ذکر شد تا محل درج گره تعیین شود. گره جدید ابتدا با ریشه درخت مقایسه می شود. اگر مقدار گره جدید کمتر از ریشه باشد و زیردرخت چپ خالی باشد گره جدید در محل فرزند چپ ریشه درج می شود. درغیراینصورت جستجو از زیردرخت چپ ادامه پیدا می کند. اگر مقدار گره جدید بزرگتر از ریشه باشد و زیردرخت راست خالی باشد گره جدید در محل فرزند راست ریشه اضافه می شود. درغیر اینصورت فرآیند جستجو از زیردرخت راست ادامه پیدا می کند.

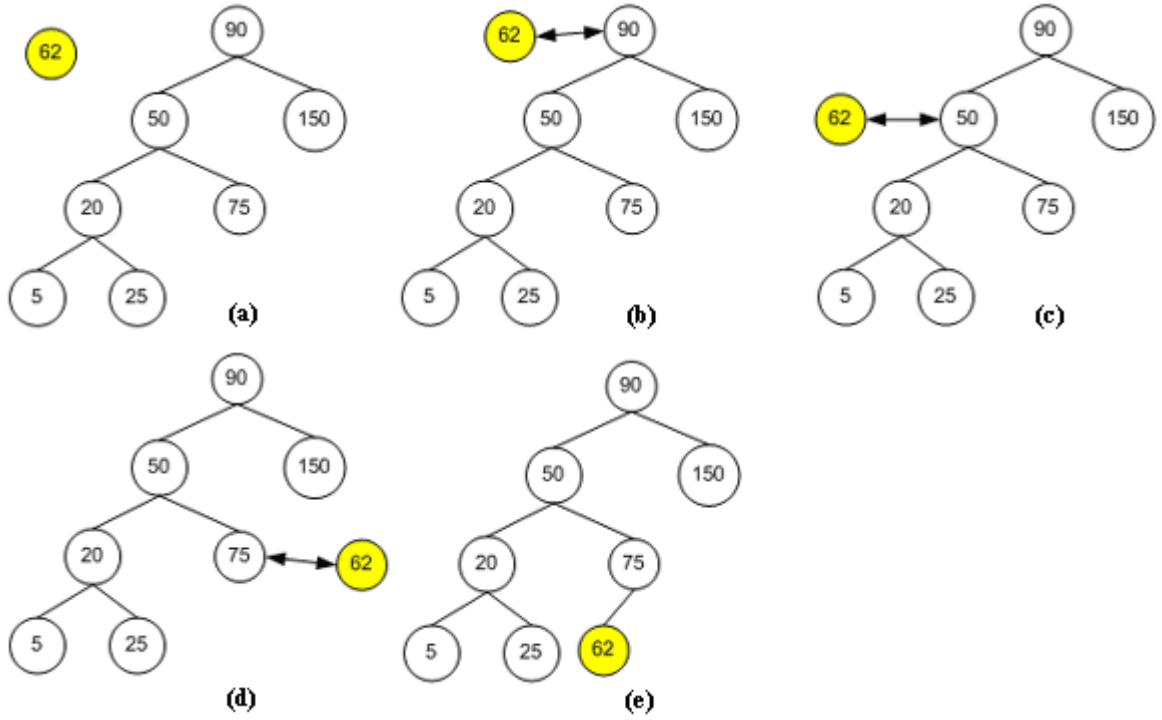
اگر زیربرنامه جستجو متوجه شود که گره جدید قبلا در BST وجود دارد پایان می یابد و دوباره آنرا درج نمی کند .

گره جدید به نحوی باید به درخت اضافه شود که BST خواص خود را حفظ کند.

گره جدید همیشه به صورت یک برگ اضافه می شود. بنابراین ترتیب درج روی توپولوژی درخت تاثیر می گذارد .

زمان اجرای الگوریتم درج مشابه الگوریتم جستجو است .

مثال. مراحلی که باید طی شود تا گره ۶۲ به BST اضافه شود در شکل های زیر نشان داده شده است .

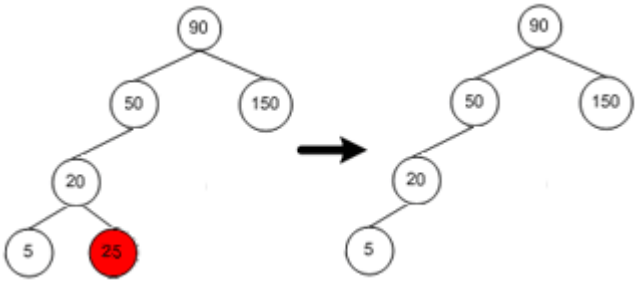


حذف ازBST

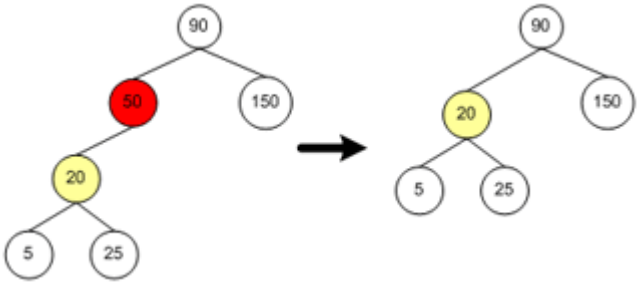
حذف یک گره از BST نسبتا دشوارتر از درج است. زیرا وقتی گره ای حذف می شود که دارای فرزند است باید گره دیگری انتخاب شود تا جایگزین گره حذف شده شود. اگر این انتخاب درست انجام نشود خواص BST نقض می شود .

گره ای که باید حذف شود ابتدا در BST جستجو می شود سپس به نحوی جایگزین می شود خواص درخت حفظ شود. چند حالت برای جایگزین کرده گره وجود دارد :

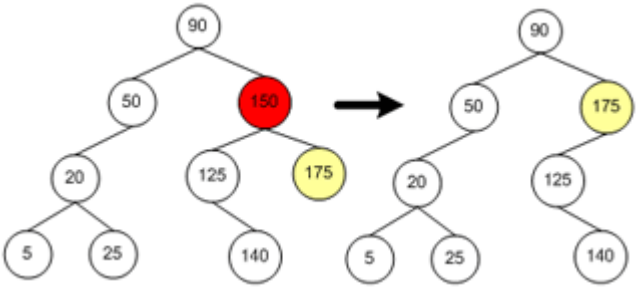
حالت ۱. گره که باید حذف شود برگ باشد و فرزندی نداشته باشد. در این حالت حذف به سادگی انجام می پذیرد و کافی است اشاره گر والد برابر تهی شود. برای مثال در شکل گره ۲۵ که حذف می شود فرزندی ندارد .



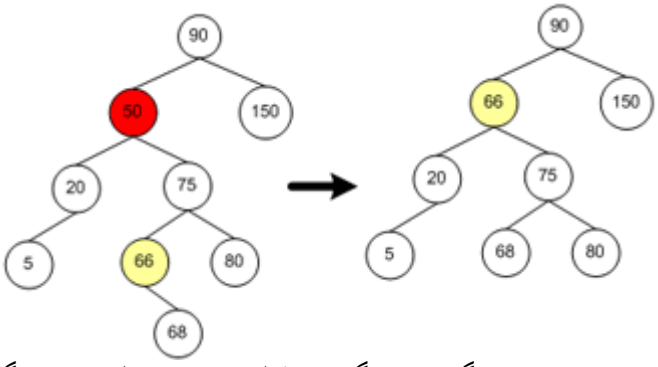
حالت ۲. گره ای که باید حذف شود تنها دارای یک فرزند چپ است که می تواند جایگزین آن شود. برای مثال فرض کنید بخواهیم گره ۵۰ را حذف کنیم .چون ۵۰ فرزند راستی ندارد ۲۰ با آن جایگزین می شود .



حالت ۳. فرزند راست گره ای که باید حذف شود فرزند چپی ندارد بنابراین فرزند راست گره جایگزین آن می شود. برای مثال اگر بخواهیم گره ۱۵۰ را حذف کنیم چون فرزند راست آن فرزند چپی ندارد فرزند راستش یعنی ۱۷۵ جایگزین می شود .



حالت ٤ . فرزند راست گره ای که باید حذف شود فرزند چپ دارد. در این حالت چپ ترین فرزند راست گره جایگزین آن می شود. یعنی کوچکترین مقدار زیر درخت راست گره. برای مثال اگر گره ٥٠ را در شکل زیر حذف کنیم چپ ترین فرزند آن یعنی ٦٦ را انتخاب کرده و جایگزین می کنیم .



در هر BST کوچکترین مقدار در سمت چپ ترین گره و بزرگترین مقدار در سمت راست ترین گره وجود دارد .

اولین مرحله در حذف گره پیدا کردن محل گره ای است که می خواهیم حذف کنیم که توسط الگوریتم جستجو که قبلا توضیح داده شد انجام می گیرد .بنابراین زمان حذف همان زمان O(log n) در حالت متوسط و O(n) در بدترین حالت است .

معایبBST

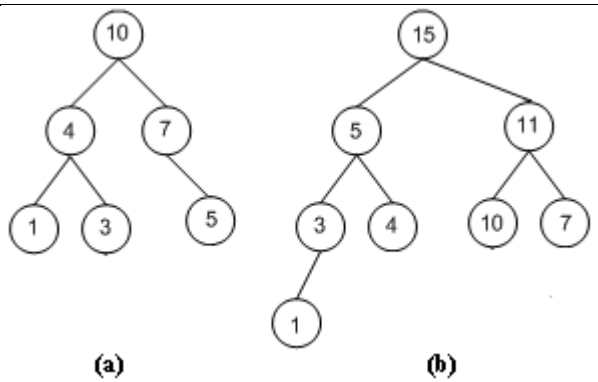
باوجودیکه درخت های جستجوی دودویي به طور ایده ال زمان زیرخطی برای درج، جستجو و حذف می دهند، زمان اجرا بستگی به توپولوزی BST دارد .توپولوژی درخت وابسته به ترتیبی است که داده در BST درج می شود. اگر داده ورودی مرتب باشد توپولوژی BST یک درخت نامیزان طولانی و باریک می شود که بدترین زمان را در اجرای الگوریتم ها می دهد و معمولا در دنیای واقعی داده ها بطور طبیعی مرتب یا نزدیک به مرتب هستند .

درختHeap

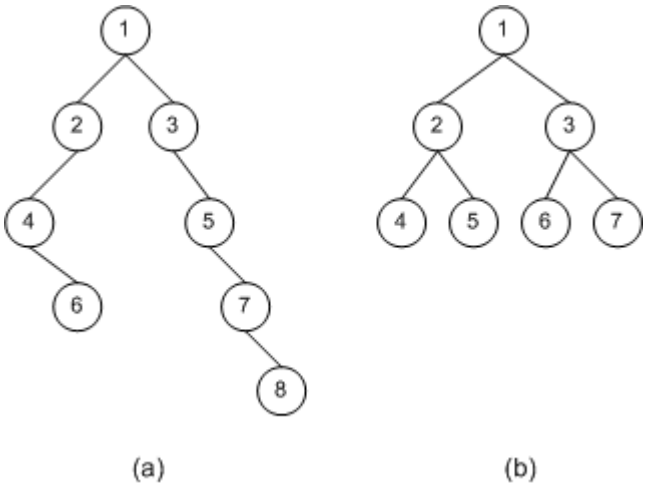
heap یک ساختمان داده درختی که خواص ویژه ای را داراست .
heap را می توان به صورت زیر تعریف کرد:

- یک max meap يك درخت دودوئي كامل است که بزرگ هم باشد. در max heap بزرگترین مقدار همیشه در ریشه قرار می گیرد.
- یک min meap يك درخت دودوئي كامل است که کوچك هم باشد. در min heap کوچکترین مقدار در ریشه قرار می گیرد.

يك درخت بزرگ (max tree) درختی است که مقادیر کلید هر گره بزرگتر از مقادیر فرزندانش باشد. یعنی اگر B فرزند A است آنگاه key(A) ≥ key(B) است .
يك درخت کوچك (min tree) درختی است که مقادیر کلید هر گره کوچکتر از مقادیر فرزندانش باشد. یعنی اگر B فرزند A است آنگاه key(A)≤ key(B) است .
مثال. در شکل زیر درخت (b) یک max heap است. درخت (a) یک درخت بزرگ است ولی max heap نیست چون یک درخت دودویی کامل نیست .



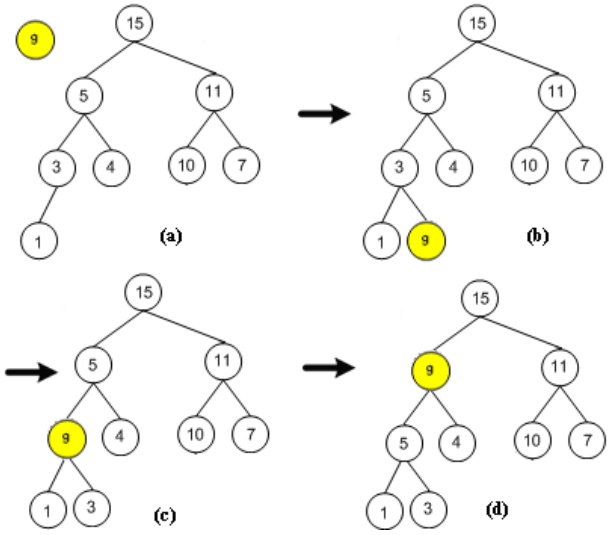
مثال. در شکل زیر درخت (b) یک min heap است. درخت (a) یک درخت کوچک است ولی min heap نیست چون یک درخت دودویی کامل نیست .



درج درHeap

یک گره جدید در heap در محلی اضافه می شود که درخت دودوئی کامل باقی بماند. پس از درج درخت تنظیم می شود تا خواص heapحفظ شود. برای مثال اگر max heap باشد و اگر مقدار کلید گره جدید از والدش بیشتر باشد جای آن با والد عوض می شود. این عمل ممکن است تا ریشه ادامه پیدا کند .

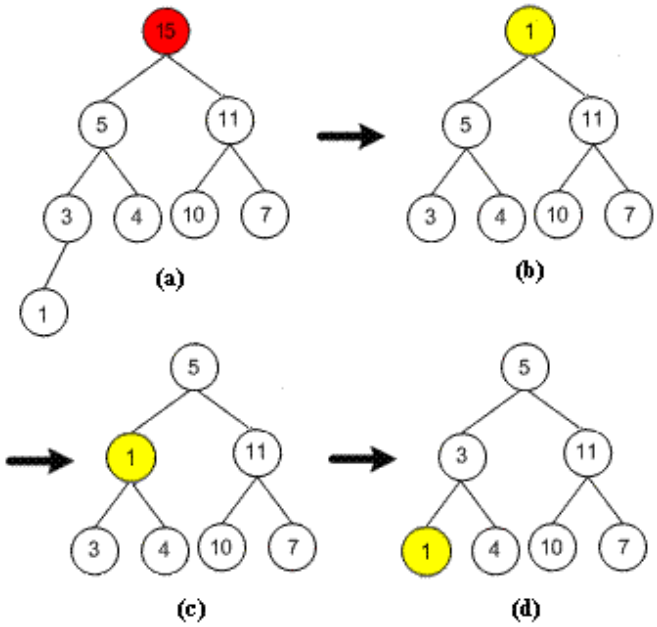
مثال. در درخت زیر گره جدید ۹ به max heap اضافه شده است. پس از درج heap تنظیم می شود و جای ۹ ابتدا با گره ۳ و سپس با ۵ جابه جا می شود .



حذف ازHeap

در Heap حذف همیشه از ریشه صورت می گیرد. بعد از حذف ریشه، آخرین گره درخت جایگزین ریشه می شود. سپس درخت تنظیم می شود تا خواص heap را حفظ کند .

مثال. در max heap شکل زیر گره ۱۵ که در ریشه قرار دارد حذف می شود .گره ۱ که سمت چپ ترین گره درسطح آخر درخت است جایگزین ریشه می شود. سپس درخت تنظیم می شود تا به صورت max heap در بیاید. ابتدا گره ۱ با ۵ و سپس با ۳ جا به جا می شود .



زمان اجرای عملیات حذف و درج در heap O(log n) است.

کاربردهایHeap

در maxh heap بزرگترین مقدار همیشه در ریشه است . در min heap کوچکترین مقدار همیشه در ریشه قرار می گیرد. به خاطر همین ویژگی heap برای پیاده سازی صف‌های الویت (priority queues) بسیار مناسب است. در صف الویت هر عنصر با الویت دلخواه را می‌توان اضافه کرد اما عنصر ماکزیمم یا مینیمم همیشه ابتدا از صف حذف می شود .

Heapها در الگوریتم مرتب سازی heapsort هم استفاده می شوند .