درس ساختمان داده‌ها
دانشکده ریاضی. دانشگاه صنعتی خواجه نصیرالدین طوسی — بهار ۹۹
تدریس توسط: حسین جوهری

ساختار داده درخت

آرایه‌ها دسترسی و جستجوی سریع را فراهم می‌آورند در حالیکه برای اعمالی چون حذف و اضافه کند عمل می‌کنند. از طرف دیگر ساختار داده لیست پیوندی اعمالی چون حذف و اضافه را سرعت می‌بخشند اما برای جستجو و دسترسی به داده‌های دلخواه مناسب نیستند. آیا می‌توان ساختار داده‌ای داشت که هم موقع جستجو سریع عمل کند و هم موقع حذف و اضافه کردن عناصر؟ برای رسیدن به چنین ساختاری باید قید خطی بودن و چیدن متوالی داده‌ها را بزنیم. به عبارت دیگر باید سراغ ساختارهایی مثل درخت برویم که داده‌ها را بصورت غیر متوالی و سلسله مراتبی ذخیره می‌کنند. در این قسمت از درس بطور مقدماتی به بررسی ساختار درخت و الگوریتمهای مربوط به آن می‌پردازیم. در قسمتهای آینده درس ساختارهای پیچیده‌تری را بر مبنای درخت معرفی می‌کنیم که اعمال جستجو و حذف و اضافه را بشکل سریع پیاده‌سازی می‌کنند.

۱ مقدمات

درخت ساختاری متداول و پرکاربرد برای نمایش روابط میان نهادها، افراد و اشیای مختلف است. از شجره فامیلی گرفته تا ساختار سازمانی، گونه‌های زیستی و جانوری، فایلها و پوشه‌های ذخیره شده در هارددیسک یک کامپیوتر، فهرست کتاب و بسیار مثالی دیگر را از ساختاری درختی پیروی می‌کنند. شکل زیر یک نمونه شجره فامیلی را نشان می‌دهد که با یک ساختار درختی نمایش داده شده است.

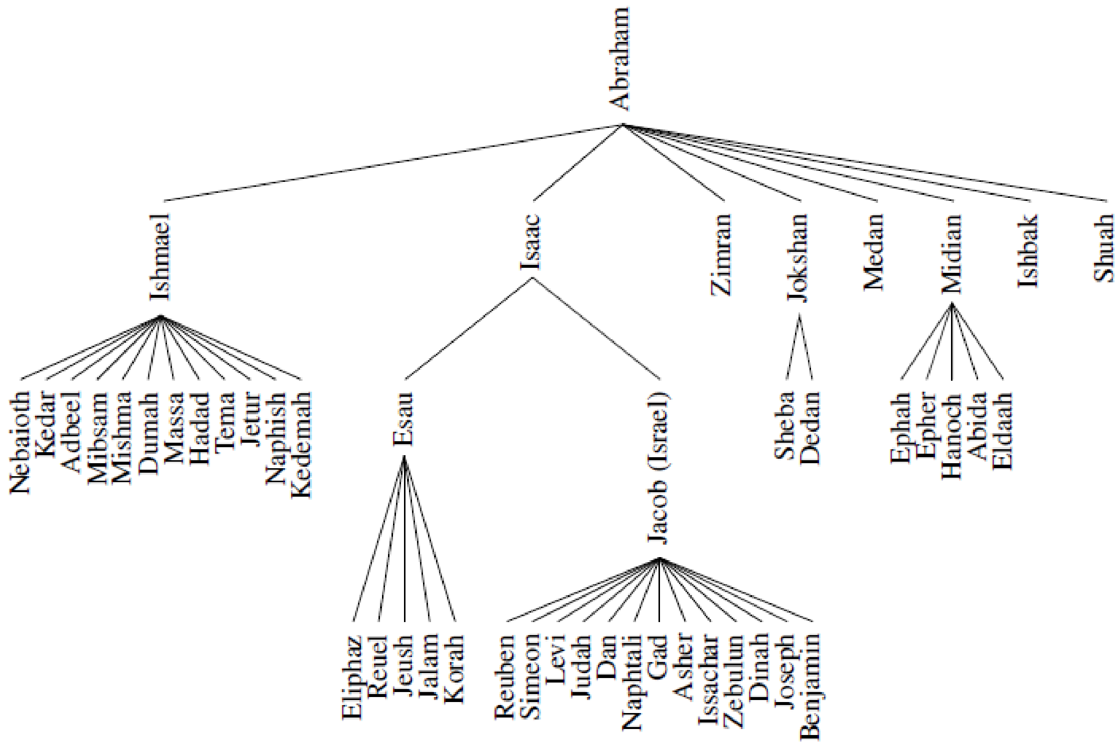


Figure ۱: شجره فامیلی با ریشه ابراهیم نبی برگرفته از کتاب مقدس

درخت ریشه‌دار مرتب یک درخت ریشه‌دار مرتب T درختی است که میان عناصر آن رابطه پدر_ فرزندی برقرار است. هر عنصر T بجز ریشه یک پدر دارد. هر راس می‌تواند چندین فرزند داشته باشد. فرزندان از سمت چپ به راست مرتب شده‌اند. اگر بخواهیم به طور بازگشتی درخت ریشه دار را تعریف کنیم، یک راس به تنهایی یک درخت ریشه‌دار مرتب است. اگر k درخت ریشه‌دار مستقل، T_1 تا T_k، به ترتیب با ریشه‌های r_1 تا r_k را داشته باشیم، می‌توانیم درخت ریشه‌دار مرتب T با ریشه r را بسازیم بطوری که r_1 تا r_k به ترتیب فرزندان r از چپ به راست هستند.

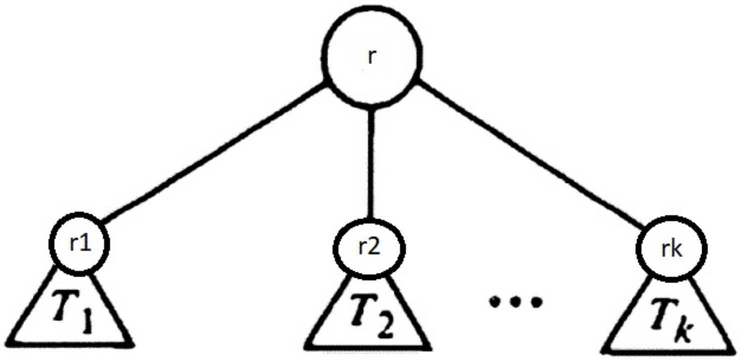


Figure ۲: یک درخت ریشه‌دار مرتب

۱.۱ چند تعریف در مورد درختها

- ریشه root : راسی در درخت ریشه‌دار که دارای پدر نیست. ریشه منحصر بفرد است.
- برگ leaf : راس بدون فرزند
- برادر sibling : راسهایی که پدر مشترک دارند برادر هستند.
- فرزند اول first child: سمت چپ ترین فرزند یک راس
- برادر بعدی next sibling : در میان برادران یک راس برادری که در سمت راست آن قرار دارد (در صورت وجود)
- ارتفاع height یک درخت : طول طولانی‌ترین مسیر از ریشه درخت تا یک برگ. درخت با تنها یک راس ارتفاع صفر دارد.
- ارتفاع یک راس : ارتفاع راس v ارتفاع زیردرخت با ریشه v است.
- سطح (عمق) یک راس depth یا level : طول مسیر از ریشه درخت تا آن راس
- درخت k تایی : درختی که هر راس آن حداکثر k فرزند دارد.
- درخت دودویی (باینری): درخت ریشه دار مرتب که هر راس آن حداکثر دو فرزند دارد.
- نوادگان یک راس descendants : نوادگان راس v راسهای زیردرخت با ریشه v هستند که خود v شامل آن نمی‌شود.
- اجداد یک راس ancestors : اجداد راس v راسهای موجود در مسیر از v به ریشه درخت است که شامل خود v نمی‌شود.

۲ ساختار داده انتزاعی درخت

ساختار داده انتزاعی درخت T اعمال اصلی زیر را پشتیبانی می‌کند.

- `T.root()`
  عنصر ذخیره شده در ریشه درخت را برمی‌گرداند.
- `T.is_root(p)`
  در صورتی که p ریشه باشد مقدار True و در غیر اینصورت False را برمی‌گرداند.
- `T.parent(p)`
  پدر p را در صورت وجود برمی‌گرداند.
- `T.num_children(p)`
  تعداد فرزندان p را برمی‌گرداند.
- `T.children(p)`
  لیستی از فرزندان p را برمی‌گرداند.
- `T.is_leaf(p)`
  در صورتی که p یک برگ باشد مقدار True و در غیر اینصورت False را برمی‌گرداند.
- `T.depth(p)`
  عمق راس p را برمی‌گرداند.
- `T.height(p)`
  ارتفاع راس p را برمی‌گرداند.
- `len(T)`
  تعداد عناصر ذخیره شده در درخت (تعداد رئوس درخت) را برمی‌گرداند.
- `T.is_empty()`
  در صورتی که T تهی باشد، مقدار True و در غیر اینصورت مقدار False را برمی‌گرداند.
- `T.nodes()`
  لیستی حاوی عناصر درخت را برمی‌گرداند.

در کتاب مرجع درسی (صفحه ۳۰۷) کلاس پایه Tree برای پیاده‌سازی ساختار داده انتزاعی درخت تعریف شده است. البته چون این کلاس یک کلاس پایه base است و کلاسهای دیگری از آن مشتق می‌شوند متدهای مربوطه در آن پیاده‌سازی نشده‌اند. در زیر بعضی از متدهای این کلاس را بصورت شبه کد پیاده‌سازی می‌کنیم.

۱.۲ محاسبه عمق در درخت

عمق یک راس از درخت می‌تواند بصورت بازگشتی پیاده‌سازی شود. اگر راس داده شده ریشه باشد، عمق برابر با 0 است، در غیر اینصورت عمق راس داده شده برابر با عمق پدر به اضافه 1 است.

```python
def depth(self, p):
  """Return the number of levels separating Position p from the root."""
  if self.is_root(p):
    return 0
  else:
    return 1 + self.depth(self.parent(p))
```

Code Fragment 8.3: Method depth of the Tree class.

زمان اجرا زمان اجرا متناسب با عمق راس داده شده است. در بدترین حالت زمان اجرا برابر با ارتفاع درخت است. اگر n تعداد رئوس درخت باشد، ارتفاع درخت حداکثر برابر با $n - 1$ است. لذا می‌توان گفت زمان اجرای الگوریتم بالا $O(n)$ است.

۲.۲ محاسبه ارتفاع در درخت

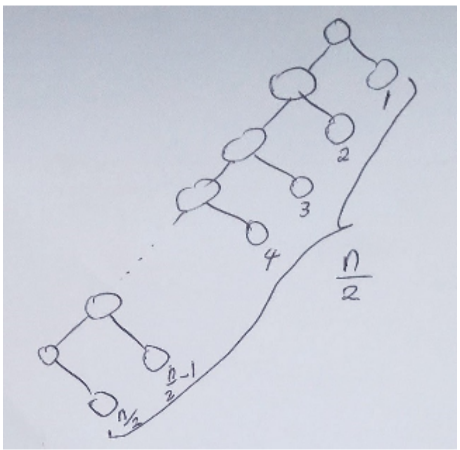
ارتفاع یک راس از درخت را نیز می‌توان بصورت بازگشتی پیاده‌سازی کرد. اگر راس داده شده برگ باشد، ارتفاع برابر با 0 است، در غیر اینصورت ارتفاع راس داده شده برابر ماکزیمم ارتفاع میان فرزندان راس داده شده به اضافه 1 است.

ارتفاع یک درخت برابر با ارتفاع ریشه آن است.

از تعریف بالا نتیجه زیر گرفته می‌شود:
ارتفاع درخت برابر با ماکزیمم عمق میان برگهای آن است. به عبارت دیگر، ارتفاع درخت برابر با بیشترین فاصله ریشه تا یک برگ است.
یک شیوه محاسبه ارتفاع درخت در قطعه کد زیر نشان داده شده است. در این شیوه همه رئوس درخت بررسی می‌شوند. رئوسی که برگ هستند، عمقشان محاسبه می‌شود و از این میان ماکزیمم گرفته می‌شود.

```python
def _height1(self):                    # works, but O(n^2) worst-case time
  """Return the height of the tree."""
  return max(self.depth(p) for p in self.nodes ( )  if self.is_leaf(p))
```

زمان اجرا در بدترین حالت، زمان اجرای کد بالا $\Theta(n^2)$ است. به شکل زیر دقت کنید.



این درخت n راس دارد. تعداد برگها برابر با $\frac{n}{2}$ است. عمق هر برگ در زیر آن نوشته شده است. الگوریتم عمق هر برگ را محاسبه می‌کند و سپس از این میان ماکزیمم می‌گیرد. با توجه به اینکه زمان محاسبه عمق متناسب با مقدار

عمق است، اگر $T(n)$ زمان اجرای الگوریتم باشد داریم

$$T(n) \geq 1 + 2 + \ldots + \frac{n}{2} = \Omega(n^2)$$

از طرفی دیگر

$$T(n) \leq (n-1)^2 = O(n^2)$$

چون حداکثر $n-1$ برگ داریم و هر برگ حداکثر عمقش برابر با $n-1$ است. در نتیجه

$$T(n) = \Theta(n^2)$$

یک شیوه دیگر محاسبه ارتفاع درخت برگرفته از همان تعریف بازگشتی ارتفاع است که در قطعه کد زیر آمده است. ارتفاع یک راس برابر با ماکزیمم ارتفاع میان فرزندان به اضافه 1 است.

```
def _height2(self, p):                    # time is linear in size of subtree
  """Return the height of the subtree rooted at Position p."""
  if self.is_leaf(p):
    return 0
  else:
    return 1 + max(self._height2(c) for c in self.children(p))
```

زمان اجرا در بدترین حالت، زمان اجرای کد بالا $\Theta(n)$ است. برگها ارتفاع صفر دارند و لذا زمان محاسبه ارتفاع آنها $O(1)$ است. از پایین که به بالا نگاه کنیم، به محض اینکه ارتفاع فرزندان محاسبه شد، ارتفاع پدر نیز قابل محاسبه است. اگر ارتفاع فرزندان محاسبه شده باشد، زمان ارتفاع پدر متناسب با تعداد فرزندان است. پس در مجموع زمان محاسبه همه ارتفاعها، متناسب با تعداد یالهای درخت است که همان $n-1$ است.