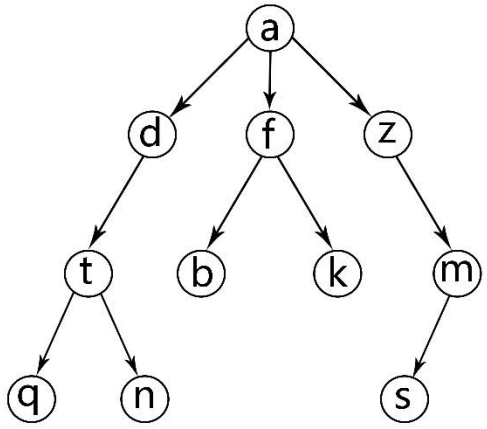
فصل چهارم

ساختمان داده ی درخت

مقدمه

درخت (Tree) یک ساختمان داده ی پرمصرف در رایانه است که از تعدادی گره تشکیل شده که با یال هایی به هم وصل شده اند.



هر گره ممکن است تعدادی فرزند داشته باشد. گره هایی که فرزند دارند برای فرزندانشان پدر محسوب می شوند.

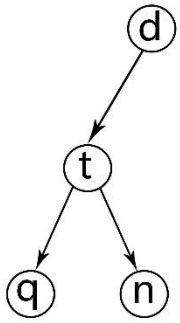
درخت یک گراف همبند بدون دور است و بین هر دو گره یک درخت ، یک مسیر یکتا وجود دارد. مانند :

تعریف بازگشتی درخت

۱- یک گره به تنهایی درخت است(Root)

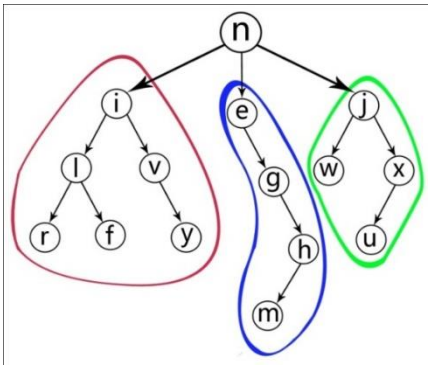
۲- اگر $T_1, T_2, \ldots, T_k$ درخت باشند، از به هم چسباندن آن ها به Root ، یک درخت به صورت بازگشتی تشکیل می شود.

زیر درخت



زیر درخت بخشی از درخت است که خود یه درخت کامل را تشکیل می دهد.
هر گره درخت را می توان به عنوان ریشه ی یک زیردرخت در نظر گرفت.
مانند شکل روبرو که در آن گره d نقش ریشه ی زیردرخت را دارد.

از چسپاندن چند درخت (زیر درخت) به یک گره(مثل n)، درخت جدیدی ایجاد می شود.(در درخت جدید n ریشه است.)



اجزای مهم درخت

۱-ریشه(Root) : در درخت به گره ای که بالاتر از همه قرار دارد و دارای پدر نیست ریشه می گویند.(مانند گره a در درخت بالا)

۲-برگ (Leaf) : به گره ای که هیچ فرزندی ندارند و درنتیجه درجه اش ۰ است ، برگ می گویند. (مثلا گره a دارای سه فرزند d و f و z است اما گره s هیچ فرزندی ندارد.)

۳- گره داخلی (Internal Node) : به گره هایی که برگ نباشند - یعنی گره هایی که حداقل یک فرزند دارند -گره داخلی می گویند.(مانند گره های a و f در درخت بالا)

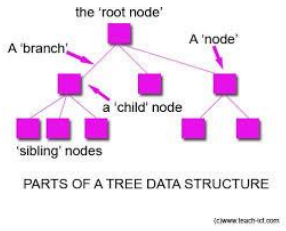
درخت یک ساختمان داده ی غیر خطی (non-linear)است. یعنی عناصر آن را نمی توان ترتیب خاصی بخشید. همچنین درخت یک گونه داده ای مجرد(ADT) است.

تعاریف اولیه

درجه ی گره(Degree) : به تعداد زیردرخت های هر گره (یا تعداد فرزندان هر گره) ، درجه ی آن گره می گویند.

درجه ی درخت : به بزرگترین درجه گره های درخت ، درجه ی درخت می گویند.

همزاد (Siblings) : فرزندان یک گره همزاد هستند.

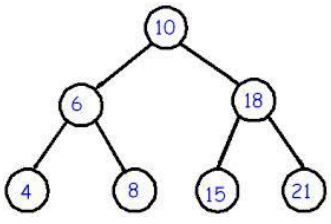


سطح گره (Level) : تعداد یال های مسیر آن گره تا ریشه را سطح یا ارتفاع گره می گویند.

ارتفاع درخت(Height of Tree) : به بیشترین سطح گره های یک درخت ، ارتفاع یا عمق درخت می گویند.

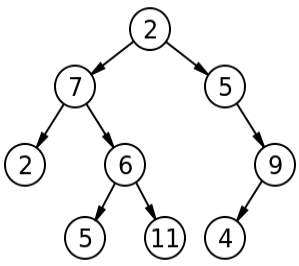
اجداد یک گره (Ancestor) : به گره های مسیر گره تا ریشه اجداد آن ، گره گفته می شود.

درخت مرتب (Ordered Tree) : درختی است که در آن ترتیب خاصی بین فرزندان اعمال می‌شود . (درخت نامرتب درختی است که در آن فرزندان هر رأس ترتیب خاصی ندارند)



مثالی از یک درخت مرتب

این درخت یک درخت جستجوی دودویی نیز هست



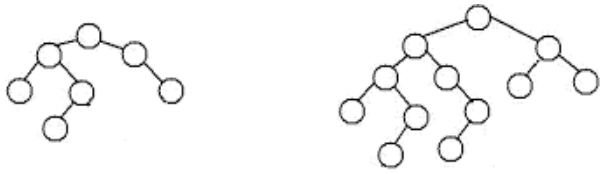
مثالی از یک درخت نامرتب

۲ گره با مقدار ۲ داریم که یکی پدرش ۷ است و دیگری پدری ندارد(ریشه است)

درخت K تایی : درختی که حداکثر درجه ی هر گره K باشد.

درخت K تایی کامل : درختی که درجه ی همه ی گره ها به جز برگ ها دقیقا K باشد.

درخت متوازن(Balanced Tree) : درختی که اختلاف ارتفاع برگ ها در آن حداکثر یک است.



درخت متوازن　　درخت نامتوازن

درخت کاملا متوازن : درختی که در آن اختلاف ارتفاع برگ ها صفر است.

جنگل : به بیشتر از یک درخت کنار هم جنگل می گویند.

قضیه ی اویلر

در درخت تعداد یال ها یکی کمتر از تعداد گره هاست. به بیان ریاضی :

۱ − تعداد گره ها = تعداد یال ها

$$E = V - ۱$$

اثبات با استقرا روی تعداد گره های یک درخت :

$$|V| = ۱ \quad \Rightarrow \quad E = ۰$$

حکم برای $V = ۱$ برقرار است.

فرض استقرا : هر درخت با V-۱ گره دارای $E = V - ۲$ یال است.

اگر از درخت با V گره یک گره کم کنیم درنتیجه $V - ۱$ گره داریم که با توجه به فرض استقرا $V - ۲$ یال داریم.

حال با اضافه کردن همان گره سر جای خودش یک یال اضافه می شود در نتیجه در درخت با V گره $V - ۱$ یال داریم. به این ترتیب قضیه اثبات می شود.

مثال : تعداد برگ های یک درخت K تایی کامل با n گره چقدر است ؟

اگر تعداد برگ ها را B در نظر بگیریم و با توجه به قضیه ی اویلر برای تعداد یال ها داریم :

$$E = n - ۱$$

$$(n - B)K = E \qquad \Rightarrow \qquad B = n - \frac{n - ۱}{K}$$

۴ – ۱ : روش های پیاده سازی درخت

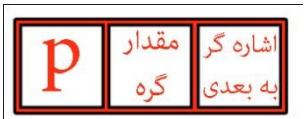
درخت در کامپیوتر به روش های مختلفی پیاده سازی می شود که عبارت اند از :

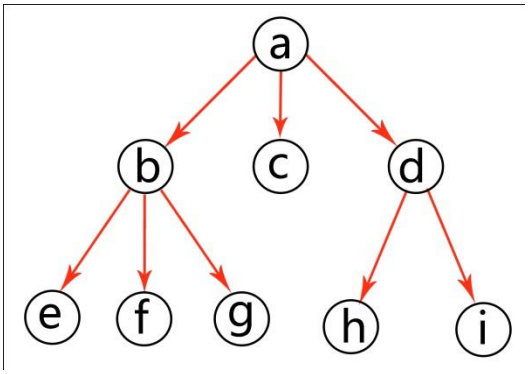
۱ – استفاده از لیست پیوندی (Linked List) :

در این روش برای پیاده سازی از دو نوع گره استفاده می کنیم.

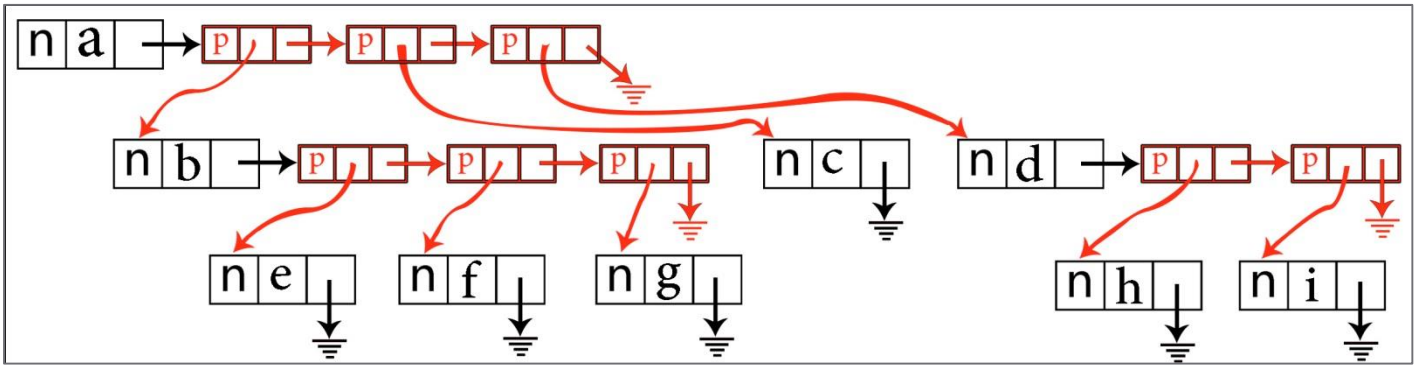
یک نوع متناظر با گره های درخت (n)

و یک نوع متناظر با یال ها (p)





مثلا پیاده سازی درخت روبه رو با این روش به شکل زیر است:

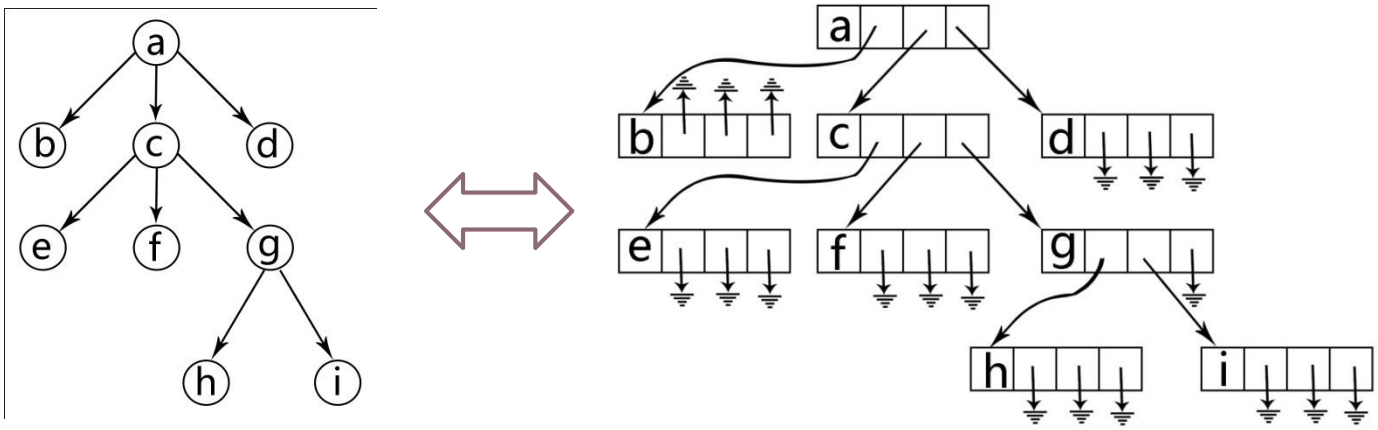


به تعداد E+V نود در لینک لیست داریم.

۲ – استفاده از K اشاره گر برای درختK تایی :

در این روش هر کدام از گره ها دارای K اشاره گره برای اشاره کردن به فرزندان خود هستند.

مثلا پیاده سازی درخت رو به رو در این روش به شکل زیر است:



این روش دو مشکل اساسی دارد که عبارت اند از :

۱ – تلفات تعداد اشاره گرهای NULL زیاد است. زیرا:

تعداد یال ها – تعداد کل اشاره گرها = تعداد اشاره گرهای NULL

$$Kn - (n - ۱) = Kn - n + ۱ \quad \text{تعداد اشاره گرهای NULL} = \text{تعدادگره ها} \times K - e$$

۲ – برای K محدودیت داریم یعنی اگر تعداد اشاره گر های یک گره بیشتر از K باشد نمی توان همه را ذخیره کرد.

مثال : فرض کنید درختی که میخواهیم ذخیره کنیم ، فولدرها در یک رایانه باشد.اگر از روش فوق استفاده کنیم، باید از قبل پیش بینی کنیم که قرار است هر فولدر چند subfolder داشته باشد.که این خوب نیست و حداقل از لحاظ تئوری k بی نهایت است.

توجه:البته اگر k کوچیک باشد و حداکثر آن را بدانیم،این روش روش خوبی است.

۳ – استفاده از درخت دودویی معادل (Leftmost Child-Right Sibling (LMC-RS))

هر درختی قابلیت تبدیل به درخت دودویی معادل را دارد.

برای تبدیل درخت به درخت دودویی معادل به تعداد گره های درخت اصلی در درخت دودویی معادل گره داریم، فرزند چپ هر گره در درخت دودویی چپ ترین فرزند درخت اصلی است و فرزند راست در درخت دودویی همزاد راست گره در درخت اصلی است.

مثال:الگوریتمی برای تبدیل یک درخت معمولی به درخت دودویی ارائه دهید.

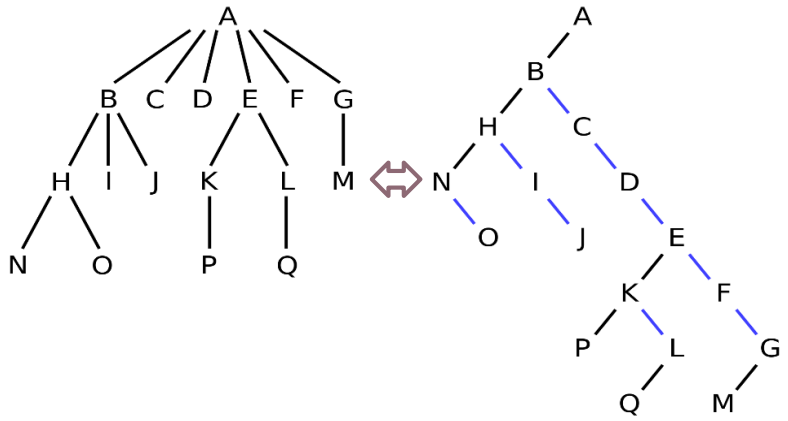
حل:

۱-از ریشه درخت شروع میکنیم(ریشه درخت معمولی ، ریشه درخت دودویی نیز هست.)

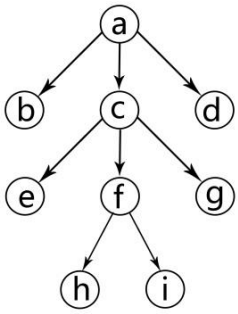
۲-اولین فرزند سمت چپ ریشه یعنی C_1 را فرزند چپی ریشه در درخت دودویی قرار میدهیم و همزاد C_1 را فرزند سمت راست C_1 در درخت دودویی قرار میدهیم(اگر همزاد نداشت null قرار میدهیم) و ....

۳- گام دوم را برای هر گره جدید تکرار میکنیم

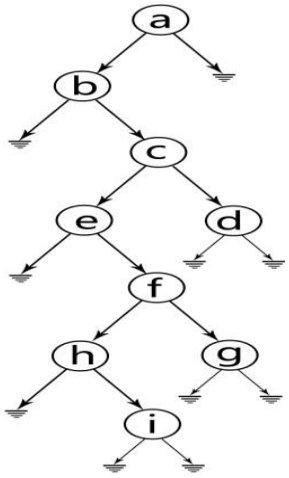
مثال:در شکل زیر مثالی از تبدیل درخت معمولی به درخت دودویی مشاهده میکنید.



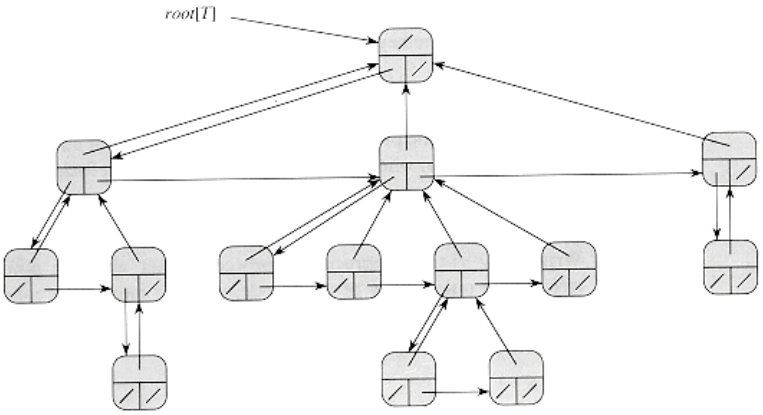
مثال : درخت روبرو را به درخت دودویی تبدیل کنید.



حل: پیاده سازی درخت دودویی معادل آن به صورت زیر است:



شکل زیر نشان میدهد چگونه میتوان یک درخت معمولی را به صورت درجا به درخت باینری تبدیل کرد



توجه: با درخت دودویی میتوان جنگلی از درخت ها را نیز پیاده سازی کرد.

مثال : الگوریتمی با مرتبه O(n) پیشنهاد دهید که مقادیر تمام گره های یک درخت ریشه دار دلخواه را که به صورت **LMC-RS** ذخیره شده است را چاپ کند.

حل:کد الگوریتم به زبان c به صورت زیر می باشد

```c
#define MAX_SIZE 10


struct tree_t {
    struct tree_t *child;
    struct tree_t *sibling;
    struct tree_t *parent;
    int key;
};
typedef struct tree_t tree_t;
void store(int);


void print_tree(tree_t *tree) {
    store(tree->key);
```

```
    if (tree->child)
        print_tree(tree->child);

    if (tree->sibling)
        print_tree(tree->sibling);
}

int keys[MAX_SIZE];
int count = 0;

void reset_storage() {
    count = 0;
}

void store(int key) {
    keys[count++] = key;
}
```

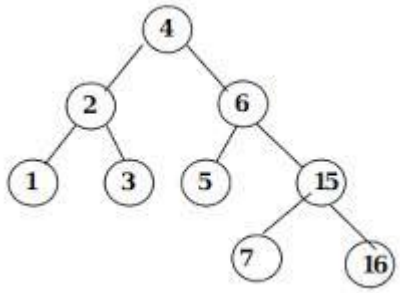
۴ – پیاده سازی درخت با استفاده از آرایه

برای این کار دو روش وجود دارد:

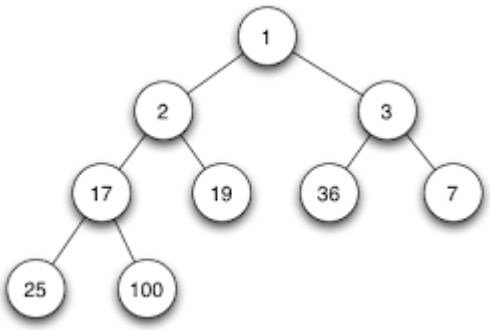
روش اول : در این روش از پیمایش سطحی(Level Order) استفاده میکنیم.

معمولا درخت های کاملا پُر را با این روش پیاده سازی می کنند.(در غیر این صورت اتلاف زیادی دارد)

درخت پر(Heap Tree) درخت متوازنی است که برگ های سمت چپ عمق بیشتری دارند.



درخت پر نیست

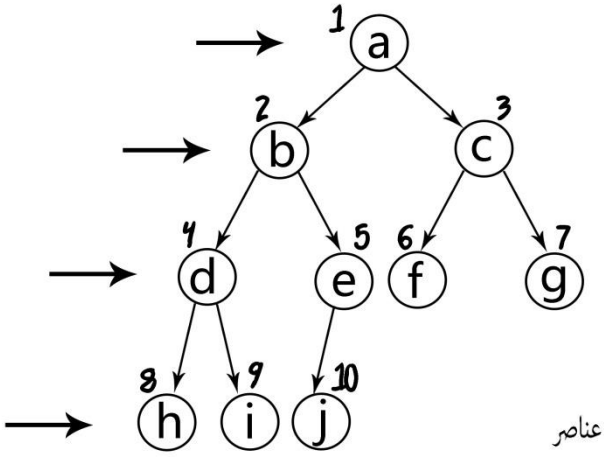


درخت پر

درخت رو به رو یک درخت پُر دوتایی است :

شماره های روی هر گره متناظر با ترتیب پیمایش سطحی (از چپ به راست) درخت است.

پیاده سازی این درخت با آرایه به شکل زیر است :



	1	2	3	4	5	6	7	8	9	10
آرایه ی عناصر	a	b	c	d	e	f	g	h	i	j

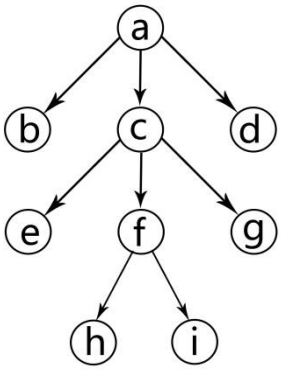
در پیاده سازی درخت های پُر با آرایه فقط به یک آرایه برای ذخیره ی عناصر نیاز داریم، زیرا با توجه به متوازن بودن درخت می توان با استفاده از اندیس فرزند اندیس پدر را محاسبه کرد.

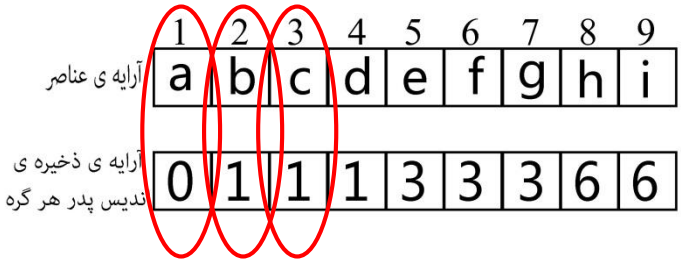
برای مثال بالا اگر اندیس فرزند را i در نظر بگیریم اندیس پدر برابر است با : $\left\lfloor \frac{i}{2} \right\rfloor$

روش دوم:

در این روش دو آرایه داریم، یکی برای ذخیره سازی داده ی داخل گره ها و یکی دیگر برای ذخیره سازی اندیس پدر هر گره.

مثلا پیاده سازی درخت روبه رو با این روش به صورت زیر است :





در این روش ریشه در اندیس ۱ آرایه ی عناصر ذخیره می شود.

هر کدام از بیضی های قرمز معادل یک یال است.

این روش بیشتر برای ذخیره سازی درخت در فایل مناسب است و نه در حافظه ی اصلی.