داده ساختار های خطی

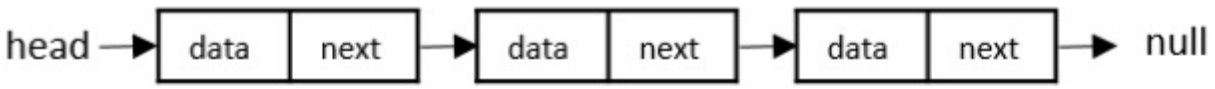
آموزش ساختارهای داده خطی در پایتون

فهرست مطالب

1. لیست پیوندی (Linked List)

تعریف کلی

لیست پیوندی یکی از ساختمان‌های داده‌ی پایه و پرکاربرد در علوم کامپیوتر است که برای ذخیره و مدیریت مجموعه‌ای از داده‌ها به صورت ترتیبی به‌کار می‌رود. برخلاف آرایه‌ها که خانه‌های حافظه‌ی آن‌ها به صورت متوالی در حافظه قرار می‌گیرند، در لیست پیوندی داده‌ها در گره‌هایی (Node) نگهداری می‌شوند که هر گره به صورت مستقل در حافظه ذخیره شده و از طریق پیوندهایی (اشاره‌گرها) به گره‌های دیگر متصل است.



ساختار گره (Node)

هر گره در یک لیست پیوندی حداقل شامل دو بخش اصلی است:

۱. **داده (Data):** مقدار یا اطلاعاتی که می‌خواهیم ذخیره کنیم.
۲. **اشاره‌گر (Pointer):** لینکی که آدرس یا مرجع گره بعدی در لیست را نگه می‌دارد.

در زبان‌های سطح پایین مانند C یا C++، اشاره‌گر به صورت آدرس حافظه گره بعدی نگهداری می‌شود، در حالی که در زبان‌های سطح بالا مانند Python، این اشاره‌گر به صورت ارجاع (Reference) به آبجکت بعدی عمل می‌کند.

گره ابتدایی و انتهایی

لیست پیوندی دارای یک گره‌ی ابتدایی به نام **Head** است که به اولین عنصر لیست اشاره دارد. از طریق Head می‌توان به صورت ترتیبی تمام گره‌ها را پیمایش کرد. در برخی پیاده‌سازی‌ها، برای دسترسی سریع‌تر به انتهای لیست، یک اشاره‌گر به آخرین گره به نام **Tail** نیز نگهداری می‌شود.

انواع لیست پیوندی

بسته به تعداد و نوع اشاره‌گرها در هر گره، لیست‌های پیوندی به انواع مختلفی تقسیم می‌شوند:

۱. لیست تک‌پیوندی (Singly Linked List):

- هر گره فقط به گره بعدی اشاره دارد.
- حرکت در این نوع لیست فقط از ابتدا به انتها ممکن است.

۲. لیست دوپیوندی (Doubly Linked List):

- هر گره شامل دو اشاره‌گر است: یکی به گره قبلی و یکی به گره بعدی.
- حرکت در این نوع لیست هم به جلو و هم به عقب امکان‌پذیر است.

۳. لیست حلقوی (Circular Linked List):

- در این نوع لیست، گره‌ی انتهایی به گره‌ی ابتدایی اشاره می‌کند و لیست به صورت دایره‌ای بسته می‌شود.
- می‌تواند به صورت تک‌پیوندی یا دوپیوندی حلقوی باشد.

مزایا

- **انعطاف‌پذیری در اندازه:** اندازه لیست می‌تواند به صورت پویا تغییر کند (برخلاف آرایه‌ها).

- **افزودن و حذف آسان:** اضافه یا حذف کردن گره‌ها به‌خصوص در ابتدای لیست بسیار سریع و با پیچیدگی زمانی O(1) قابل انجام است.

معایب

- **دسترسی ترتیبی:** برای دستیابی به عنصر خاصی، باید لیست را از ابتدا پیمایش کرد (O(n)).
- **مصرف حافظه‌ی اضافی:** به دلیل وجود اشاره‌گرها، فضای بیشتری نسبت به آرایه اشغال می‌کند.

عملیات‌های اصلی روی لیست پیوندی

درج (Insertion)

← درج در ابتدای لیست (Insert at Beginning)

- یک گره جدید ساخته می‌شود.
- اشاره‌گر آن به گره‌ی فعلی Head تنظیم می‌شود.
- اشاره گر Head به گره جدید به‌روزرسانی می‌شود.
- پیچیدگی زمانی: **O(1)**

← درج در انتهای لیست (Insert at End)

- لیست تا رسیدن به آخر پیمایش می‌شود (مگر اینکه Tail موجود باشد).
- اشاره‌گر گره آخر به گره جدید تنظیم می‌شود.
- در صورت وجود Tail، این کار در **O(1)** انجام می‌شود، در غیر این صورت **O(n)**

← درج در موقعیت خاص (Insert at Position)

- لیست تا موقعیت مورد نظر پیمایش می‌شود.
- گره جدید بین دو گره درج می‌شود و اشاره‌گرها به‌روزرسانی می‌شوند.
- پیچیدگی زمانی: **O(n)**

حذف (Deletion)

← حذف از ابتدای لیست (Delete from Beginning)

- اشاره گر Head به گره‌ی دوم منتقل می‌شود.
- گره اول آزاد می‌شود.
- پیچیدگی زمانی: **O(1)**

← حذف از انتهای لیست (Delete from End)

- لیست پیمایش می‌شود تا به گره قبل از آخر برسیم.
- اشاره‌گر آن Null می‌شود.
- پیچیدگی زمانی: **O(n)** (مگر اینکه دوپیوندی باشد)

← حذف از موقعیت خاص (Delete from Position)

- پیمایش تا رسیدن به گره‌ی قبل از موقعیت انجام می‌شود.
- اشاره‌گر آن تغییر داده می‌شود تا گره‌ی مورد حذف از لیست کنار گذاشته شود.
- پیچیدگی زمانی: **O(n)**

جستجو (Search)

← جستجو بر اساس مقدار

- گره‌ها به ترتیب بررسی می‌شوند تا گره‌ای با مقدار مورد نظر پیدا شود.
- در صورت پیدا شدن، مکان یا مرجع گره برگردانده می‌شود.
- پیچیدگی زمانی: **O(n)**

پیمایش (Traversal)

← پیمایش از ابتدا تا انتها

- از Head شروع کرده و تا رسیدن به Null (یا در لیست حلقوی تا برگشت به Head) حرکت می‌کنیم.
- معمولاً برای چاپ مقادیر یا بررسی‌ها استفاده می‌شود.

- پیچیدگی زمانی: **O(n)**

به‌روزرسانی (Update / Modify)

← تغییر مقدار یک گره

- ابتدا گره مورد نظر با جستجو یا موقعیت مشخص پیدا می‌شود.
- مقدار داده‌ی آن تغییر داده می‌شود.
- پیچیدگی زمانی: **O(n)**

محاسبات کمکی

← شمارش تعداد گره‌ها (Length)

- از Head شروع کرده، هر گره را می‌شماریم تا به انتها برسیم.
- پیچیدگی زمانی: **O(n)**

← بررسی تهی بودن لیست (Is Empty)

- بررسی اینکه آیا Head برابر با Null است یا نه.
- پیچیدگی زمانی: **O(1)**

← یافتن میانه، مینیمم یا ماکسیمم

- بسته به نیاز، با پیمایش لیست می‌توان این مقادیر را پیدا کرد.
- پیچیدگی زمانی: **O(n)**

عملیات	توضیح	مرتبه زمانی
درج در ابتدا	فقط Head تغییر می‌کند	**O(1)**
درج در انتها (بدون Tail)	نیاز به پیمایش کل لیست	**O(n)**
درج در انتها (با Tail)	مستقیماً به Tail اشاره داریم	**O(1)**

عملیات	توضیح	مرتبه زمانی
درج در وسط	پیمایش تا مکان مورد نظر	**O(n)**
حذف از ابتدا	Head را به گره بعدی اشاره می‌دهیم	**O(1)**
حذف از انتها	پیمایش تا گره قبل از آخر	**O(n)**
حذف از وسط	پیمایش و تغییر اشاره‌گرها	**O(n)**
جستجو	باید خانه‌به‌خانه بگردیم	**O(n)**
پیمایش کل لیست	بازدید از همه‌ی گره‌ها	**O(n)**
بررسی تهی بودن	چک کردن Head == None	**O(1)**
شمارش عناصر	نیاز به پیمایش	**O(n)**
به‌روزرسانی داده	جستجو و تغییر مقدار	**O(n)**

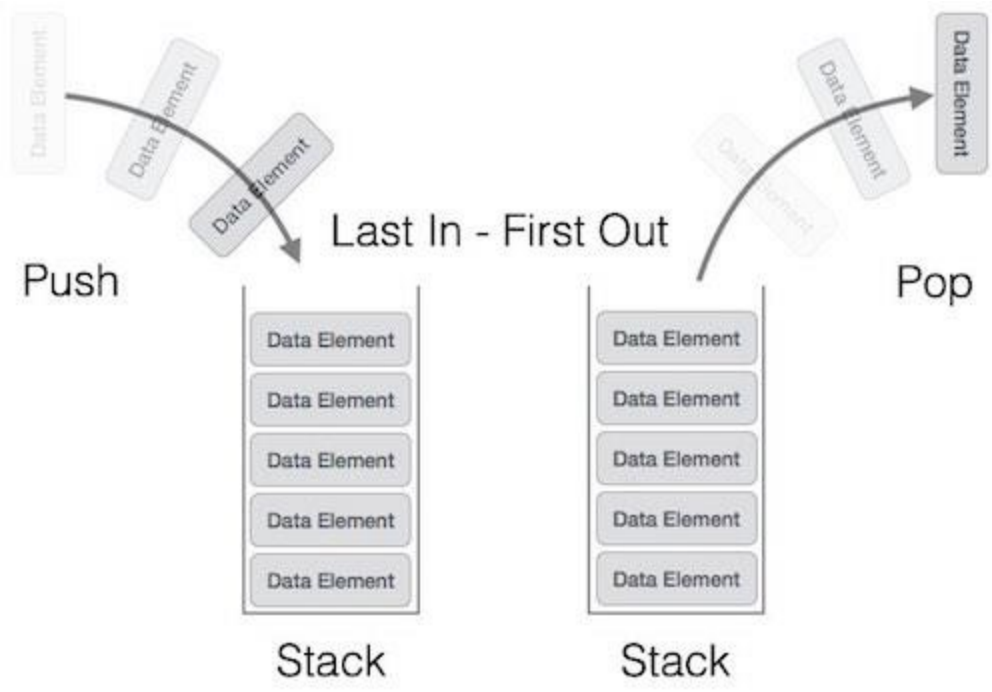
2. پشته (Stack)

تعریف:

تعریف کلی

پشته (Stack) یکی از ساختمان‌های داده‌ی بنیادی در علوم کامپیوتر است که از **قانون LIFO (Last In, First Out)** پیروی می‌کند؛ یعنی **آخرین عنصری که وارد می‌شود، اولین عنصری است که خارج می‌شود**.

پشته را می‌توان مانند یک سطل تصور کرد: هر چیزی را که آخر از همه در سطل می‌گذارید، اولین چیزی است که هنگام خالی کردن خارج می‌شود.



ساختار و نحوه پیاده‌سازی

پشته می‌تواند به دو روش پیاده‌سازی شود:

۱. **با استفاده از آرایه (Array/List):**

- در این حالت از یک آرایه یا لیست معمولی استفاده می‌شود و از انتهای آن داده‌ها اضافه یا حذف می‌شوند.
- در زبان‌هایی مانند Python می‌توان از `list` یا بهتر از آن، `collections.deque` استفاده کرد.

۲. **با استفاده از لیست پیوندی (Linked List):**

- هر بار که داده‌ای اضافه می‌شود، یک گره در ابتدای لیست اضافه می‌شود.
- حذف داده‌ها نیز از ابتدا صورت می‌گیرد.

عملیات های مهم

- عملیات **Push:** عمل اضافه کردن یک عنصر به بالای پشته.
- عملیات **Pop:** عمل حذف و بازگرداندن عنصر بالای پشته.
- عملیات **Top / Peek:** مشاهده عنصر بالای پشته بدون حذف آن.

- عملیات **IsEmpty:** بررسی اینکه پشته خالی است یا نه.

عملیات	توضیح	مرتبه زمانی
`Push(item)`	اضافه‌کردن یک عنصر به بالای پشته	**O(1)**
`Pop()`	حذف عنصر بالای پشته و بازگرداندن آن	**O(1)**
`Top() / Peek()`	مشاهده عنصر بالای پشته بدون حذف	**O(1)**
`IsEmpty()`	بررسی خالی بودن پشته	**O(1)**
`Size()`	برگرداندن تعداد عناصر موجود	**O(1)** یا **O(n)** بسته به پیاده‌سازی

مزایا و معایب پشته

مزایا:

- پیاده‌سازی ساده و کم‌هزینه
- بسیار مناسب برای مسائلی مثل: بازگشت (Recursion)، parsing، بررسی توازن پرانتزها و ...

معایب:

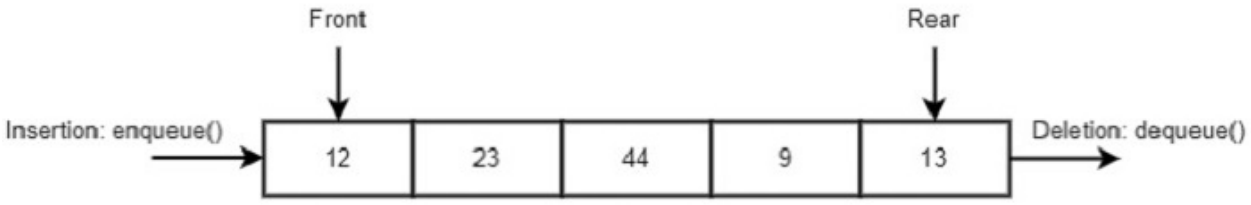
- فقط به عنصر بالای پشته دسترسی داریم (نه به بقیه عناصر)
- ترتیب LIFO برای همه‌ی کاربردها مناسب نیست

3. صف (Queue)

تعریف کلی

صف (Queue) یکی از ساختمان‌های داده‌ی پایه‌ای در برنامه‌نویسی است که از **قانون FIFO (First In, First Out)** پیروی می‌کند. یعنی **اولین عنصری که وارد صف می‌شود، اولین عنصری است که خارج می‌شود**.

صف مشابه صف ایستادن در فروشگاه است: کسی که اول می‌آید، اول هم سرویس می‌گیرد.



Queue: FIFO Operation

نحوه‌ی پیاده‌سازی

صف را می‌توان به چند روش پیاده‌سازی کرد:

۱. **با آرایه (Array / List):**

- از ابتدای لیست برای حذف و از انتها برای افزودن استفاده می‌شود.
- در زبان‌هایی مثل Python استفاده از `collections.deque` پیشنهاد می‌شود چون عملیات‌ها در آن بهینه‌اند.

۲. **با لیست پیوندی (Linked List):**

- گره‌های جدید در انتهای لیست افزوده می‌شوند.
- حذف از ابتدای لیست انجام می‌شود.
- معمولاً Head و Tail برای دسترسی سریع استفاده می‌شود.

اجزای صف

- ابتدای صف یا **Front:** جایی که داده‌ها از آن خارج می‌شوند.
- انتهای صف یا **Rear (یا Back):** جایی که داده‌ها به آن افزوده می‌شوند.

عملیات	توضیح	مرتبه زمانی
`Enqueue(item)`	افزودن یک عنصر به انتهای صف	**O(1)**
`Dequeue()`	حذف عنصر از ابتدای صف و بازگرداندن آن	**O(1)**
`Front() / Peek()`	مشاهده‌ی اولین عنصر بدون حذف	**O(1)**

عملیات	توضیح	مرتبه زمانی
`IsEmpty()`	بررسی خالی بودن صف	**O(1)**
`Size()`	بازگرداندن تعداد عناصر	**O(1)** یا **O(n)** بسته به ساختار

مزایا و معایب صف

مزایا:

- بسیار مناسب برای مدیریت منابع، پردازش وظایف در صف، صف چاپگر، صف مشتریان و ...
- قابل پیاده‌سازی با لیست پیوندی یا `deque`

معایب:

- در پیاده‌سازی با آرایه، امکان مصرف ناکارآمد حافظه در صورت حذف‌های مکرر (مگر در صف حلقوی)
- دسترسی مستقیم به عناصر میانی وجود ندارد

نکات تکمیلی:

▼ **تعریف مرتبه زمانی یا `Big O notation`**

مرتبه زمانی (Time Complexity) یک روش برای توصیف میزان زمانی است که یک الگوریتم برای اجرا شدن به آن نیاز دارد، **بر اساس اندازه ورودی (n)**.

در واقع، ما نمی‌خواهیم بدونیم اجرای برنامه چند ثانیه طول می‌کشه، بلکه می‌خواهیم بدونیم با **بزرگ شدن داده‌ها**، زمان اجرای برنامه چطور رشد می‌کنه.

برخی از نمادهای رایج:

- زمان ثابت یا **O(1):** مستقل از اندازه داده‌ها
- زمان خطی یا **O(n):** رشد متناسب با اندازه داده‌ها
- زمان لگاریتمی یا **O(log n):** معمولاً وقتی ورودی نصف میشه در هر مرحله

- زمان درجه دوم یا **$O(n^2)$:** مثلاً در الگوریتم‌های دو حلقه‌ای تو در تو