(تمارین درس مباحث ویژه)

بخش سوم

مدرس: مهندس احمدزاده

دانشجویان:

عباس کارگرجهرمی / مهدی فرازنده شهرکی

دانشکده ملی مهارت میناب

بهمن ۱۴۰۳

تفاوت اصلی بین List **و** Array **بستگی به زبان برنامه‌نویسی‌ای دارد که در موردش صحبت می‌کنیم، اما به‌طور کلی در بسیاری از زبان‌ها (مثل** Python**،** Java **و ...)، تفاوت‌هایی کلی بین این دو وجود دارد. در ادامه، تفاوت‌ها رو به شکل ساده توضیح می‌دم:**

۱. نوع داده‌ها (Data Type**)**

– Array**: معمولاً فقط می‌تونه شامل یک نوع داده باشه (مثلاً فقط عدد یا فقط رشته).**

– مثال: در جاوا یا C++**، آرایه‌ای از** int **فقط می‌تونه عدد صحیح داشته باشه.**

– List**: معمولاً می‌تونه شامل چند نوع داده مختلف باشه (در زبان‌هایی مثل** Python**).**

– مثال: [**۱,** "3.14 ,hello"]

۲. اندازه (Size**)**

– Array**: معمولاً اندازه‌اش ثابت هست؛ یعنی بعد از ایجاد، نمی‌تونی به راحتی آیتم جدید اضافه یا حذف کنی.**

– List**: اندازه‌اش پویاست؛ یعنی می‌تونی آیتم اضافه یا حذف کنی.**

۳. کارایی (Performance**)**

– Array**: چون ساده‌تر و کم‌حجم‌تر هست، معمولاً سریع‌تره، مخصوصاً در پردازش‌های عددی یا حلقه‌ها.**

– List**: ممکنه کمی کندتر باشه ولی قابلیت‌های بیشتری داره.**

۴. توابع و قابلیت‌ها

– List**: معمولاً متدهای بیشتری برای کار با داده‌ها داره (مثل** append, remove, sort**).**

– Array**: امکانات کمتری داره، مگه این‌که از کتابخانه‌های خاصی استفاده بشه (مثل** NumPy **در** Python**).**

مثال در Python**:**

```python
# List
my_list = [1, 2, 3, "hello"]

# Array
import array
my_array = array.array('i', [1, 2, 3])  # فقط عدد صحیح
```

اگه می‌خوای تفاوت‌ها رو در زبان خاصی مثل Java یا C++ یا Python بدونی، بگو تا دقیق‌تر بررسی کنم.

در پایتون، دیکشنری (Dictionary) یک نوع ساختار داده‌ست که اطلاعات را به صورت جفت‌های کلید و مقدار ذخیره می‌کند. یعنی هر مقدار، با یک کلید منحصر به‌فرد قابل دسترسی است.

به‌جای اینکه مثل لیست‌ها فقط ترتیب عناصر مهم باشد، در دیکشنری تمرکز روی این است که به هر داده با استفاده از یک کلید دلخواه (مانند نام، عدد، یا رشته) دسترسی پیدا کنیم. مثلاً می‌توان برای شخصی، کلیدی به نام «نام» داشت که مقدار آن «علی» باشد.

دیکشنری‌ها بسیار انعطاف‌پذیر هستند و می‌توان در آن‌ها انواع داده‌ها را به عنوان مقدار ذخیره کرد. کلیدها باید نوع‌هایی باشند که تغییر نمی‌کنند (مانند رشته، عدد یا تاپل‌های بدون تغییر)، ولی مقدارها می‌توانند هر چیزی باشند، حتی لیست یا دیکشنری دیگر.

با استفاده از دیکشنری، می‌توان اطلاعات را سریع‌تر و سازمان‌یافته‌تر جست‌وجو، اضافه، حذف یا تغییر داد. این ساختار در خیلی از کاربردهای برنامه‌نویسی، به‌ویژه وقتی داده‌ها با نام یا ویژگی خاصی مشخص می‌شوند، بسیار پرکاربرد است.

تفاوت اصلی بین List (لیست) و Tuple (تاپل) در پایتون در قابلیت تغییر آن‌هاست. در ادامه تفاوت‌ها را به زبان ساده توضیح می‌دهم:

۱. قابلیت تغییر (Mutability):

– لیست قابل تغییر است. یعنی می‌توان بعد از ساخت، عناصرش را تغییر داد، عنصر جدید اضافه کرد یا حذف کرد.

– تاپل غیرقابل تغییر است. یعنی بعد از ساخته شدن، دیگر نمی‌توان عناصرش را تغییر داد یا چیزی به آن اضافه یا حذف کرد.

۲. نحوه تعریف:

– لیست با براکت مربع تعریف می‌شود: [ ]

– تاپل با پرانتز معمولی تعریف می‌شود: ( )

۳. کاربردها:

– از لیست وقتی استفاده می‌شود که نیاز به تغییر در داده‌ها باشد.

– از تاپل وقتی استفاده می‌شود که داده‌ها باید ثابت بمانند و از تغییر در امان باشند (مثلاً مختصات یک نقطه، یا داده‌هایی که نباید تصادفی تغییر کنند).

۴. سرعت و کارایی:

– تاپل‌ها معمولاً سریع‌تر و سبک‌تر از لیست‌ها هستند، چون ثابت‌اند و سیستم بهتر می‌تواند آن‌ها را مدیریت کند.

۵. امنیت داده:

– چون تاپل تغییرناپذیر است، برای انتقال داده‌هایی که نباید تغییر کنند (مثلاً تنظیمات یا مقادیر ثابت)، امن‌تر و مناسب‌تر است.

در پایتون، **set (مجموعه) برای حذف داده‌های تکراری استفاده می‌شود چون به‌طور پیش‌فرض فقط عناصر یکتا (unique) را نگه می‌دارد.**

دلیل اصلی:

set **در پایتون یک ساختار داده‌ای بدون ترتیب و بدون عناصر تکراری است. وقتی لیستی از داده‌ها را به** set **تبدیل می‌کنی، پایتون خودش همه‌ی مقدارهای تکراری رو حذف می‌کنه و فقط یک نسخه از هر مقدار رو نگه می‌داره.**

مثال:

python

numbers = [۱, ۲, ۲, ۳, ۴, ۴, ۵]

unique_numbers = set(numbers)

print(unique_numbers)

خروجی:

{۱, ۲, ۳, ۴, ۵}

چرا مفید است؟

– نیازی به نوشتن حلقه یا شرط برای چک کردن تکراری بودن نیست.

– سریع و بهینه است.

– کد را ساده‌تر و تمیزتر می‌کند.

اگه خواستی دوباره لیست داشته باشی (مثلاً با ترتیب مهم)، می‌تونی set **رو دوباره به** list **تبدیل کنی:**

python

unique_list = list(set(numbers))

Stack **و** Queue **هر دو ساختار داده‌ای هستند، اما تفاوت اصلی‌شان در نحوه‌ی اضافه و حذف کردن داده‌ها است:**

– Stack **(پشته):**

روش دسترسی به داده‌ها در Stack **به صورت** LIFO **است، یعنی آخرین عنصری که وارد می‌شود، اولین عنصری است که خارج می‌شود.**

مثل یک دسته کتاب که روی هم چیده شده‌اند، کتابی که آخر گذاشته شده، اول برداشته می‌شود.

– Queue **(صف):**

روش دسترسی در Queue **به صورت** FIFO **است، یعنی اولین عنصری که وارد می‌شود، اولین عنصری است که خارج می‌شود.**

مثل صف نانوایی؛ هر کس زودتر آمده، زودتر نان می‌گیرد.

پس:

Stack → **آخر وارد شده، اول خارج می‌شود**

Queue → **اول وارد شده، اول خارج می‌شود**

Hash Table **(جدول هش) یک ساختار داده‌ای است که برای ذخیره‌سازی داده‌ها به شکلی سریع و کارآمد از یک تابع هش استفاده می‌کند. این ساختار به شما امکان می‌دهد که داده‌ها را با سرعت بالایی (در بهترین حالت** O(1)**) پیدا، وارد یا حذف کنید.**

نحوه‌ی کار:

در Hash Table**، هر عنصر از داده‌ها به یک کلید (**key**) و مقدار (**value**) متصل است. برای ذخیره‌سازی داده‌ها، از یک تابع هش استفاده می‌شود که کلید را به یک شاخص (**index**) در آرایه یا لیست تبدیل می‌کند. این شاخص مشخص می‌کند که داده‌ها در کجا ذخیره شوند.**

مزایا:

– سرعت جستجو، درج و حذف: با استفاده از یک تابع هش مناسب، این عملیات‌ها در O**(۱) زمان انجام می‌شوند (در شرایط ایده‌آل).**

– کارایی بالا: بسیار سریع‌تر از ساختارهای داده‌ای دیگر مانند لیست‌ها یا درخت‌ها در مواردی است که به جستجوی سریع نیاز داریم.

کاربردها:

۱. جستجو: برای ذخیره و بازیابی داده‌ها با سرعت بالا.

۲. مدیریت داده‌ها: در پایگاه‌های داده برای ایندکس کردن داده‌ها.

۳. ایجاد کش (Cache**): ذخیره‌سازی مقادیر محاسباتی یا نتایج پرس‌وجوهای تکراری.**

۴. شمارش تکراری (Counting**): مثلاً در الگوریتم‌هایی که می‌خواهند تعداد وقوع کلمات در یک متن را بشمارند.**

۵. دیکشنری‌ها: در زبان‌های برنامه‌نویسی مانند پایتون، دیکشنری‌ها از hash table **برای ذخیره‌ی کلیدها و مقادیر استفاده می‌کنند.**

تفاوت‌های اصلی بین B-tree **و** Binary Tree **(درخت دودویی) به ساختار، عملکرد و کاربرد آن‌ها مربوط می‌شود. در ادامه، به‌صورت خلاصه و شفاف تفاوت‌ها رو توضیح می‌دم:**

۱. تعداد فرزندان گره‌ها

– Binary Tree: **هر گره حداکثر ۲ فرزند دارد (چپ و راست).**

– B-tree: **هر گره می‌تواند چندین فرزند (بیشتر از ۲) داشته باشد. درخت** B **با مرتبه‌ی** m **می‌تواند تا** m **فرزند برای هر گره داشته باشد.**

۲. تعداد کلیدها در هر گره

– Binary Tree: **هر گره فقط یک مقدار یا کلید دارد.**

– B-tree: **هر گره چندین کلید دارد که به ترتیب مرتب شده‌اند.**

۳. کاربردها

– Binary Tree **(به‌ویژه** BST **یا** AVL Tree**): بیشتر در ساختار داده‌های حافظه اصلی (**RAM**) برای جستجوی سریع استفاده می‌شوند.**

– B-tree: **معمولاً برای دستگاه‌های ذخیره‌سازی ثانویه مثل دیسک‌ها و پایگاه داده‌ها استفاده می‌شود چون طراحی شده تا تعداد دفعات دسترسی به دیسک را کاهش دهد.**

۴. عمق درخت

– Binary Tree: **ممکن است بسیار عمیق باشد (بخصوص اگر متوازن نباشد).**

– B-tree: **طراحی شده تا عمق کمی داشته باشد، بنابراین جستجو در آن سریع‌تر در حافظه دیسکی انجام می‌شود.**

۵. مرتب‌سازی و جستجو

– Binary Search Tree: **جستجو با استفاده از مقایسه با گره میانی و رفتن به چپ یا راست.**

– B-tree: **در هر گره چندین کلید داریم، پس جستجو درون گره و سپس انتخاب یکی از فرزندان.**

ساختار داده Graph **(گراف) برای شبکه‌های اجتماعی استفاده می‌شه چون خیلی طبیعی و کارآمد می‌تونه روابط بین افراد یا موجودیت‌ها رو نمایش بده.**

اینجا چند دلیل اصلیش رو می‌گم:

---

۱. مدل‌سازی روابط (Relationship Modeling)

در شبکه‌های اجتماعی، افراد با هم دوستی، فالو، لایک، کامنت و... دارند. این‌ها همه روابط بین موجودیت‌ها هستند.

– هر گره (Node) **می‌تونه یک کاربر یا پست یا موضوع باشه.**

– هر یال (Edge) **نشون‌دهنده‌ی رابطه بین اون‌هاست (مثلاً "کاربر** A**، کاربر** B **رو دنبال می‌کنه").**

---

۲. پشتیبانی از جستجوهای پیچیده

گراف‌ها اجازه می‌دن سوالاتی مثل این‌ها رو راحت انجام بدی:

– "دوستانِ دوستان من کی هستن؟"

– "چه کسی بیشترین ارتباط غیرمستقیم با من رو داره؟"

– "کدوم پست‌ها توسط دوستان من لایک شده؟"

در گراف اینا با مسیرها (paths) **و درجه گره‌ها** (node degree) **خیلی راحت قابل محاسبه‌ست.**

---

۳. تحلیل شبکه اجتماعی (Social Network Analysis)

با گراف می‌تونیم کارهای جالبی مثل:

– تشخیص اینفلوئنسرها (با centrality **یا** hub scores**)**

– تشخیص جوامع (community detection)

– پیشنهاد دوست جدید (با الگوریتم‌هایی مثل PageRank **یا** Common Neighbors**)**

---

۴. کارایی بالا در روابط پیچیده

اگه بخوای این روابط رو توی دیتابیس‌های معمولی مثل جداول relational **ذخیره کنی،** queryها **خیلی پیچیده و کند می‌شن.**

اما ساختار گرافی مثل Neo4j **یا** GraphDB **دقیقاً برای این نوع داده طراحی شدن و خیلی سریع کار می‌کنن.**

Dynamic Programming **(برنامه‌نویسی پویا) در مسائل پیچیده کاربرد داره چون:**

---

۱. حل بهینه‌ی مسائل دارای زیرمسئله‌ی تکراری

خیلی از مسائل پیچیده (مثل مسیرهای بهینه، ترتیب ضرب ماتریس‌ها، چینش سکه‌ها و...) شامل زیرمسئله‌هایی هستن که بارها تکرار می‌شن.

به‌جای اینکه اون‌ها رو هر بار دوباره حل کنیم، Dynamic Programming **نتایج رو ذخیره می‌کنه (**memoization **یا** tabulation**) و فقط یک بار حل می‌شه، پس سرعت حل خیلی می‌ره بالا.**

---

۲. کاهش چشمگیر زمان اجرا

مسائلی که بدون DP **پیچیدگیشون نمایی (**Exponential**) هست، با** DP **به پیچیدگی چندجمله‌ای (**Polynomial**) کاهش پیدا می‌کنن.**

مثال:

– Fibonacci **ساده: زمان نمایی**

– Fibonacci **با** DP**: زمان خطی**

---

۳. مدیریت انتخاب‌های وابسته (Optimal Substructure)

در بسیاری از مسائل، انتخاب بهینه در کل، به معنی ترکیب انتخاب‌های بهینه‌ی کوچکتر هست. DP **دقیقاً این ساختار رو مدیریت می‌کنه.**

---

۴. کاربرد در مسائل واقعی و معروف

Dynamic Programming **برای حل طیف بزرگی از مسائل کاربرد داره:**

– مسائل مسیریابی (مثل shortest path**)**

– تشخیص توالی در زیست‌شناسی

– فشرده‌سازی داده

– تشخیص گفتار و پردازش زبان طبیعی

– بازی‌ها و تصمیم‌گیری‌های مرحله‌ای

———

۵. کاهش پیچیدگی ذهنی الگوریتم

گرچه اولش ممکنه سخت به نظر برسه، اما با نوشتن حالت بازگشتی و بعد تبدیلش به حالت بهینه (با حافظه)، ساختار حل خیلی شفاف‌تر و قابل پیاده‌سازی می‌شه.