1- کتابخانه های machine Learnigدر زبان Rustرا نام ببرید ؟یک مثال ساده بنویسید ؟

کتابخانه‌های معروف یادگیری ماشین در Rust:

Linfa: این کتابخانه یکی از جامع‌ترین و پرکاربردترین کتابخانه‌های یادگیری ماشین در Rust است. Linfa الگوریتم‌های مختلفی برای دسته‌بندی، رگرسیون، خوشه‌بندی و کاهش ابعاد داده ارائه می‌دهد. هدف این کتابخانه ارائه یک API کاربرپسند و کارآمد برای یادگیری ماشین است.

Rust-ML: این کتابخانه مجموعه‌ای از الگوریتم‌ها و ابزارها برای یادگیری ماشین است. تمرکز این کتابخانه بیشتر بر روی ارائه الگوریتم‌های پایه‌ای است و برای پروژه‌هایی که نیاز به کنترل دقیق‌تری بر الگوریتم‌ها دارند مناسب است.

Tch-rs: این کتابخانه یک رابط Rust برای کتابخانه PyTorch است. اگر شما با PyTorch آشنا هستید، می‌توانید از این کتابخانه برای استفاده از قدرت PyTorch در پروژه‌های Rust خود استفاده کنید.

Ndarray: این کتابخانه برای عملیات آرایه‌ای چندبعدی بهینه شده است. اغلب کتابخانه‌های یادگیری ماشین از ndarray برای مدیریت داده‌ها استفاده می‌کنند.

SmartCore: یک کتابخانه دیگر که الگوریتم های پایه ای یادگیری ماشین را در اختیار شما قرار می دهد.

مثال ساده با Linfa (دسته‌بندی k-Nearest Neighbors):

در این مثال، یک دسته‌بندی ساده با استفاده از الگوریتم k-Nearest Neighbors (k-NN) انجام می‌دهیم. این مثال شامل تولید داده‌های مصنوعی، آموزش مدل k-NN و پیش‌بینی بر روی داده‌های جدید است.

```
Rust

;*::Use linfa::prelude

;Use linfa::Dataset

;Use linfa::Float

;Use linfa_datasets::generate::{make_blobs, Blobs}

;Use linfa_knn::{Knn, NearestNeighbours}

;Use ndarray::{Array2, Array1, Axis}

// ساخت داده‌های مصنوعی برای دسته‌بندی

Fn generate_data<F: Float>(num_samples: usize, num_features: usize, num_classes:
}<<usize) -> Dataset<Array2<F>, Array1<usize
```

```
;Let blobs: Blobs<F> = make_blobs(num_samples, num_features, num_classes)
;()Let (observations, targets) = blobs.into_inner
Dataset::new(observations, targets)
{

} ()Fn main
    // 1. تولید داده‌های مصنوعی
    ;Let num_samples = 100
    ;Let num_features = 2
    ;Let num_classes = 2
    ;Let dataset = generate_data(num_samples, num_features, num_classes)
    ;()Let (observations, targets) = dataset.into_parts
    // 2. آموزش مدل k-NN (k=3)
    ;Let model = Knn::params(3).fit(&dataset)

    // 3. ایجاد داده های جدید برای پیش بینی
    ,Let new_data = Array2::from_shape_vec((2, num_features)
        Vec![1.0, 1.0, 5.0, 5.0]
    ;().unwrap(

    // 4. پیش بینی با استفاده از مدل آموزش داده شده
    ;Let predictions = model.predict(&new_data)

    // 5. چاپ نتایج
    ;Println!("Observations: \n{:?}", observations)
    ;Println!("Targets: \n{:?}", targets)
    ;Println!("New data: \n{:?}", new_data)
    ;Println!("Predictions: \n{:?}", predictions)
```

{

توضیحات:

ما از کتابخانه‌های linfa_datasets ،linfa و ndarray استفاده می‌کنیم.

تابع generate_data برای تولید داده‌های مصنوعی استفاده می‌شود.

در تابع main، ابتدا داده‌های مصنوعی را تولید می‌کنیم.

سپس مدل k-NN را با پارامتر k=3 آموزش می‌دهیم.

در ادامه، داده‌های جدید را برای پیش‌بینی آماده می‌کنیم.

در نهایت، پیش‌بینی‌ها را با استفاده از مدل آموزش‌داده‌شده انجام می‌دهیم.

در پایان، نتایج را چاپ می کنیم.

نحوه اجرای کد:

مطمئن شوید که Rust و Cargo (مدیر بسته Rust) را نصب کرده‌اید.

یک پروژه جدید Rust ایجاد کنید: cargo new knn_example

به دایرکتوری پروژه بروید: cd knn_example

فایل Cargo.toml را با محتوای زیر ویرایش کنید:

Toml

[package]

"Name = "knn_example

"Version = "0.1.0

"Edition = "2021

[dependencies]

"Linfa = "0.6

"Linfa-datasets = "0.6

"Linfa-knn = "0.6

"Ndarray = "0.15

فایل src/main.rs را با کد بالا جایگزین کنید.

کد را اجرا کنید: cargo run

این مثال ساده، نمایشی از چگونگی استفاده از کتابخانه Linfa برای یک وظیفه دسته‌بندی ساده است. شما می‌توانید با مطالعه مستندات کتابخانه‌ها، الگوریتم‌ها و قابلیت‌های بیشتری را کشف کنید.

2- برای انجام یک پردازش ساده بروی یک Videoبنویسید ؟

. برای این کار از کتابخانه `ffm-next` استفاده خواهیم کرد.

پیش‌نیازها

1. **نصب Rust:** اگر Rust را نصب نکرده‌اید، از وب‌سایت رسمی Rust اقدام کنید.
2. **ایجاد پروژه Cargo:** با استفاده از دستور `cargo new –bin video_processing` یک پروژه جدید Rust ایجاد کنید.
3. **نصب FFmpeg:** قبل از هر چیز شما باید ffmpeg را روی سیستم خود نصب کنید.
4. **افزودن وابستگی‌ها:** به فایل `Cargo.toml` وابستگی `ffmpeg-next` و `image` را اضافه کنید:

```toml
[dependencies]
"Ffmpeg-next = "6.0
"Image = "0.24
```

کد برنامه

در اینجا کد کامل برنامه `src/main.rs` آمده است:

```rust
;Use ffmpeg_next::{format, frame, media, Error, Packet}
;Use image::{ImageBuffer, Rgb}
```

```
;Use std::path::Path
;Use std::io


<Fn extract_frames(input_path: &str, output_dir: &str, frame_rate: f64) -> Result<(), Error
}
    ;?Let input = format::input(&input_path)
        Let video_stream_index = input
            ()streams.
                )find.
,stream| stream.codec().medium() == media::Type::Video&|
                    (
        ;?ok_or(Error::StreamNotFound).
            Let video_stream =
;?input.stream(video_stream_index).ok_or(Error::StreamNotFound)
    ;?()Let mut decoder = video_stream.codec().decoder


    ;Let mut frame_index = 0
    ;Let mut frame_count = 0
    ;Let mut frame_pts = 0



    }()For (stream, packet) in input.packets
    } If stream.index() == video_stream_index
    } ?If let Some(frame) = decoder.decode(&packet)
;Let frame_time = frame.pts() as f64 * video_stream.time_base() as f64
    ;Let frame_skip_time = frame_count as f64 / frame_rate
    }If frame_time >= frame_skip_time
```

```
Let frame_path = Path::new(output_dir).join(format!("frame_{}.png",
                                                    ;frame_index))
                                    ;?Save_frame(&frame, &frame_path)
                                                ;Frame_index += 1
                                                Frame_count += 1
                                                        {
                                                    }Else
                                                ;Frame_count += 1
                                                        {
                                                            {
                                                                {
                                                                    {


                                            ;Println!("Extracted {} frames.", frame_index)
                                                                ((Ok()
                                                                    {

                } <Fn save_frame(frame: &frame::Video, path: &Path) -> Result<(), Error
                                                ;Let width = frame.width() as u32
                                                ;Let height = frame.height() as u32
                                        ;Let mut image_buffer = ImageBuffer::new(width, height)

                                        } ()For (x, y, pixel) in image_buffer.enumerate_pixels_mut
                                                ;Let index = (y * width + x) as usize
                                                ; Let r = frame.data(0)[index * 3]
                                                ; Let g = frame.data(0)[index * 3 + 1]
                                                ; Let b = frame.data(0)[index * 3 + 2]
                                                        pixel = Rgb([r,g,b])*
```

```
                                                                {

    Image_buffer.save(path).map_err(|_| Error::Other { details: "image save
                                                                ;?failed".to_string() })

                                                                    (())Ok
                                                                        {

                                                                } ()Fn print_menu
                                                            ;Println!("\nVideo Processing Menu:")
                                                                ;Println!("1. Extract Frames")
                                                                    ;Println!("2. Exit")
                                                                ;Print!("Enter your choice: ")
                                                                ;()Io::stdout().flush().unwrap
                                                                        {

                                                        } <Fn main() -> Result<(), ffmpeg_next::Error
                                                                    } Loop
                                                                ;()Print_menu

                                                                ;()Let mut choice = String::new
                                                    ;Io::stdin().read_line(&mut choice).expect("Failed to read line")
                                                        ;Let choice: u32 = choice.trim().parse().unwrap_or(0)

                                                                } Match choice
                                                                    } <= 1
                                                            ;Print!("Enter input video path: ")
                                                                ;()Io::stdout().flush().unwrap
```

```
                                                ;()Let mut input_path = String::new
;Io::stdin().read_line(&mut input_path).expect("Failed to read line")
                                                ;()Let input_path = input_path.trim


                                        ;Print!("Enter output frames directory: ")
                                                ;()Io::stdout().flush().unwrap
                                                ;()Let mut output_dir = String::new
;Io::stdin().read_line(&mut output_dir).expect("Failed to read line")
                                                ;()Let output_dir = output_dir.trim

                                            ;Print!("Enter frame rate (fps): ")
                                                ;()Io::stdout().flush().unwrap
                                            ;()Let mut frame_rate = String::new
;Io::stdin().read_line(&mut frame_rate).expect("Failed to read line")
    ;Let frame_rate: f64 = frame_rate.trim().parse().unwrap_or(30.0)



        } Match extract_frames(input_path, output_dir, frame_rate)
                                                    ,() <= (_)Ok
                ,Err€ => eprintln!("Error extracting frames: {}", e)
                                                            {
                                                                {
                                                        } <= 2
                                                ;Println!("Exiting...")
                                                        ;Break
                                                            {
```

```
                ,println!("Invalid choice.") <= _
                    {
                        {

                    (())Ok
                        {
```

توضیحات کد

1. **`extract_frames` تابع:**
   * یک فایل ویدیویی را باز می‌کند.
   * جریان ویدیو را پیدا می‌کند.
   * فریم‌ها را از ویدیو استخراج می‌کند و هر فریم را به صورت فایل تصویر ذخیره می‌کند.
2. **`save_frame` تابع:** یک فریم را از ساختار `frame::Video` به یک ساختار `ImageBuffer` تبدیل کرده و آن را ذخیره می‌کند.
3. **`print_menu` تابع:** منوی عملیات را نمایش می‌دهد.
4. **`main` تابع:**
   * یک حلقه بی‌نهایت ایجاد می‌کند که منو را نمایش می‌دهد و عملیات انتخاب شده توسط کاربر را اجرا می‌کند.
   * از کاربر ورودی برای مسیر ویدیو و مسیر ذخیره فریم‌ها را دریافت کرده و تابع استخراج فریم را فراخوانی می‌کند.

کامپایل و اجرا

1. برای کامپایل برنامه از دستور زیر استفاده کنید:

```bash
Cargo build
```

2. برای اجرای برنامه از دستور زیر استفاده کنید:

```bash
Cargo run
```

بعد از اجرای برنامه، یک منو نمایش داده می‌شود که می‌توانید از آن برای استخراج فریم‌های ویدیویی استفاده کنید.

نکات مهم

* **مدیریت خطا:** در این مثال از `?` برای مدیریت خطاها استفاده شده است.
* **`ffmpeg-next`:** کتابخانه `ffmpeg-next` برای کار با فایل‌های چندرسانه‌ای استفاده می‌شود.
* **پردازش پیچیده:** این مثال یک پردازش بسیار ساده را انجام می‌دهد. برای انجام پردازش‌های پیچیده‌تر و پیشرفته‌تر، نیاز به کدنویسی و دانش بیشتری در مورد پردازش ویدیو خواهید داشت.
* **رابط کاربری:** رابط کاربری این مثال ساده است، اما می‌توان آن را با استفاده از کتابخانه‌های UI بهبود بخشید.
* **frame rate** Frame Rate:** در اینجا فریم هایی که در ثانیه از ویدیو استخراج می شود رو مشخص میکنه

نکته: به دلیل اینکه شما به APIهای وب دسترسی ندارید، برای اجرای این برنامه، نیاز به نصب `ffmpeg-next` و `image` و همچنین `ffmpeg` به صورت دستی خواهید داشت.

۵۳ سوال . بله، حتماً. در اینجا یک برنامه به زبان Rust می‌نویسم که یک فایل را از اینترنت دانلود می‌کند. برای این کار از کتابخانه `reqwest` استفاده خواهیم کرد.

پیش‌نیازها

1. **نصب Rust:** اگر Rust را نصب نکرده‌اید، از وب‌سایت رسمی Rust اقدام کنید.
2. **ایجاد پروژه Cargo:** با استفاده از دستور `cargo new –bin download_file` یک پروژه جدید Rust ایجاد کنید.

3. **افزودن وابستگی‌ها:** به فایل `Cargo.toml` وابستگی `reqwest` و `tokio` را اضافه کنید:

```toml
[dependencies]
Reqwest = { version = "0.11", features = ["blocking"] }
Tokio = { version = "1", features = ["full"] }
```

کد برنامه

در اینجا کد کامل برنامه `src/main.rs` آمده است:

```rust
;Use std::fs::File
;Use std::io::Write
;Use reqwest::blocking::get
;Use std::io

} <<Fn download_file(url: &str, file_path: &str) -> Result<(), Box<dyn std::error::Error
    ;?Let response = get(url)

    } ()If response.status().is_success
        ;?Let mut file = File::create(file_path)
        ;?()Let content = response.bytes
        ;?File.write_all(&content)
        ;Println!("File downloaded successfully to: {}", file_path)
    } else {
        ;Println!("Failed to download file. Status code: {}", response.status())
```

```
                                                                {


                                                        (())Ok
                                                            {



                                                    } ()Fn print_menu
                                            ;Println!("\nFile Download Menu:")
                                                ;Println!("1. Download File")
                                                    ;Println!("2. Exit")
                                                ;Print!("Enter your choice: ")
                                            ;()Io::stdout().flush().unwrap
                                                                {

} <<Fn main() -> Result<(), Box<dyn std::error::Error


                                                                } Loop
                                                        ;()Print_menu
                                                ;()Let mut choice = String::new
                                            ;?Io::stdin().read_line(&mut choice)
                                        ;?()Let choice: u32 = choice.trim().parse


                                                                } Match choice
                                                                    } <= 1
                                                    ;Print!("Enter URL: ")
                                                ;()Io::stdout().flush().unwrap
                                                ;()Let mut url = String::new
                                            ;?Io::stdin().read_line(&mut url)
```

```
                                                    ;()Let url = url.trim



                                        ;Print!("Enter file path to save: ")
                                          ;()Io::stdout().flush().unwrap
                                        ;()Let mut file_path = String::new
                                        ;?Io::stdin().read_line(&mut file_path)
                                          ;()Let file_path = file_path.trim

                                        } Match download_file(url, file_path)
                                                        ,() <= (_)Ok
                                    ,Err€ => eprintln!("Error downloading file: {}", e)
                                                                    {
                                                                       {
                                                                    } <= 2
                                                ;Println!("Exiting...")
                                                                ;Break
                                                                       {
                                                ,println!("Invalid choice.") <= _
                                                                          {
                                                                             {
                                                                    (())Ok
                                                                                {
                                                                                ```
```

**توضیحات کد**

1. **تابع `download_file`:**
```

* یک درخواست `GET` به آدرس URL داده شده می‌فرستد.
* بررسی می‌کند که آیا پاسخ موفق بوده است یا خیر.
* اگر موفق بود، محتوای پاسخ را خوانده و در یک فایل ذخیره می‌کند.
* پیام موفقیت یا شکست را در کنسول نمایش می دهد.

2. **`print_menu` تابع:** منوی عملیات را نمایش می‌دهد.
3. **`main` تابع:**

* یک حلقه بی‌نهایت ایجاد می‌کند که منو را نمایش می‌دهد و عملیات انتخاب شده توسط کاربر را اجرا می‌کند.
* از کاربر URL و مسیر فایل را دریافت کرده و تابع دانلود را فراخوانی می‌کند.

کامپایل و اجرا

1. برای کامپایل برنامه از دستور زیر استفاده کنید:

```bash
Cargo build
```

2. برای اجرای برنامه از دستور زیر استفاده کنید:

```bash
Cargo run
```

بعد از اجرای برنامه، یک منو نمایش داده می‌شود که می‌توانید از آن برای دانلود فایل استفاده کنید.

نکات مهم

* **مدیریت خطا:** در این مثال از `?` برای مدیریت خطاها استفاده شده است.
* **`reqwest`:** کتابخانه `reqwest` برای ارسال درخواست‌های HTTP استفاده می‌شود.

* **دانلود فایل بزرگ:** این مثال ساده برای دانلود فایل‌های کوچک و متوسط مناسب است. برای دانلود فایل‌های بزرگ، ممکن است نیاز به استفاده از روش‌های پیشرفته‌تری مانند دانلود به صورت جریانی باشد.

* **رابط کاربری:** رابط کاربری این مثال ساده است، اما می‌توان آن را با استفاده از کتابخانه‌های UI بهبود بخشید.

نکته: به دلیل اینکه شما به APIهای وب دسترسی ندارید، برای اجرای این برنامه، نیاز به نصب `reqwest` و `tokio` به صورت دستی خواهید داشت. همچنین مطمئن شوید که کامپیوتر شما به اینترنت متصل است.

4- برنامه‌ای برای دانلود یک فایل ازInternetبنویسید؟

بله، حتماً. در اینجا یک برنامه به زبان Rust می‌نویسم که یک فایل را از اینترنت دانلود می‌کند. برای این کار از کتابخانه `reqwest` استفاده خواهیم کرد.

پیش‌نیازها

1. **نصب Rust:** اگر Rust را نصب نکرده‌اید، از وب‌سایت رسمی Rust اقدام کنید.
2. **ایجاد پروژه Cargo:** با استفاده از دستور `cargo new –bin download_file` یک پروژه جدید Rust ایجاد کنید.
3. **افزودن وابستگی‌ها:** به فایل `Cargo.toml` وابستگی `reqwest` و `tokio` را اضافه کنید:

```toml
[dependencies]
Reqwest = { version = "0.11", features = ["blocking"] }
Tokio = { version = "1", features = ["full"] }
```

کد برنامه

در اینجا کد کامل برنامه `src/main.rs` آمده است:

```rust
```

```
;Use std::fs::File
;Use std::io::Write
;Use reqwest::blocking::get
;Use std::io

} <<Fn download_file(url: &str, file_path: &str) -> Result<(), Box<dyn std::error::Error
    ;?Let response = get(url)

    } ()If response.status().is_success
        ;?Let mut file = File::create(file_path)
        ;?()Let content = response.bytes
        ;?File.write_all(&content)
        ;Println!("File downloaded successfully to: {}", file_path)
    } else {
        ;Println!("Failed to download file. Status code: {}", response.status())
    {

    (())Ok
{


} ()Fn print_menu
    ;Println!("\nFile Download Menu:")
    ;Println!("1. Download File")
    ;Println!("2. Exit")
    ;Print!("Enter your choice: ")
    ;()Io::stdout().flush().unwrap
{
```

```
} <<Fn main() -> Result<(), Box<dyn std::error::Error

                                                    } Loop
                                                ;()Print_menu
                                            ;()Let mut choice = String::new
                                        ;?Io::stdin().read_line(&mut choice)
                                    ;?()Let choice: u32 = choice.trim().parse

                                                } Match choice
                                                    } <= 1
                                            ;Print!("Enter URL: ")
                                        ;()Io::stdout().flush().unwrap
                                        ;()Let mut url = String::new
                                    ;?Io::stdin().read_line(&mut url)
                                            ;()Let url = url.trim



                                    ;Print!("Enter file path to save: ")
                                        ;()Io::stdout().flush().unwrap
                                    ;()Let mut file_path = String::new
                                ;?Io::stdin().read_line(&mut file_path)
                                    ;()Let file_path = file_path.trim

                                } Match download_file(url, file_path)
                                                    ,() <= (_)Ok
                        ,Err€ => eprintln!("Error downloading file: {}", e)
                                                    {
```

```
                {
            } <= 2
        ;Println!("Exiting...")
            ;Break
                {
    ,println!("Invalid choice.") <= _
                    {
                        {
                    ())Ok
                        {
                    ```
```

**توضیحات کد**

1. **`download_file` تابع:**
   * یک درخواست `GET` به آدرس URL داده شده می‌فرستد.
   * بررسی می‌کند که آیا پاسخ موفق بوده است یا خیر.
   * اگر موفق بود، محتوای پاسخ را خوانده و در یک فایل ذخیره می‌کند.
   * پیام موفقیت یا شکست را در کنسول نمایش می دهد.
2. **`print_menu` تابع:** منوی عملیات را نمایش می‌دهد.
3. **`main` تابع:**
   * یک حلقه بی‌نهایت ایجاد می‌کند که منو را نمایش می‌دهد و عملیات انتخاب شده توسط کاربر را اجرا می‌کند.
   * از کاربر URL و مسیر فایل را دریافت کرده و تابع دانلود را فراخوانی می‌کند.

**کامپایل و اجرا**

1. برای کامپایل برنامه از دستور زیر استفاده کنید:

```bash
```
```

Cargo build

```

2. برای اجرای برنامه از دستور زیر استفاده کنید:

bash```

Cargo run

```

بعد از اجرای برنامه، یک منو نمایش داده می‌شود که می‌توانید از آن برای دانلود فایل استفاده کنید.

نکات مهم

* **مدیریت خطا:** در این مثال از `?` برای مدیریت خطاها استفاده شده است.
* **`reqwest`:** کتابخانه `reqwest` برای ارسال درخواست‌های HTTP استفاده می‌شود.
* **دانلود فایل بزرگ:** این مثال ساده برای دانلود فایل‌های کوچک و متوسط مناسب است. برای دانلود فایل‌های بزرگ، ممکن است نیاز به استفاده از روش‌های پیشرفته‌تری مانند دانلود به صورت جریانی باشد.
* **رابط کاربری:** رابط کاربری این مثال ساده است، اما می‌توان آن را با استفاده از کتابخانه‌های UI بهبود بخشید.

نکته: به دلیل اینکه شما به APIهای وب دسترسی ندارید، برای اجرای این برنامه، نیاز به نصب `reqwest` و `tokio` به صورت دستی خواه

5- بله، حتماً. در اینجا یک سرویس RESTful API با استفاده از زبان Rust و کتابخانه `actix-web` می‌نویسم که درخواست‌های JSON را پردازش می‌کند. این مثال یک API ساده برای مدیریت یک لیست از وظایف (todos) را نشان می‌دهد.

پیش‌نیازها

1. **نصب Rust:** اگر Rust را نصب نکرده‌اید، از وب‌سایت رسمی Rust اقدام کنید.

2. **نصب Actix-Web:** از طریق Cargo می‌توانید `actix-web` را به پروژه خود اضافه کنید.

ایجاد یک پروژه جدید Cargo

```bash
Cargo new todos-api
Cd todos-api
```

افزودن وابستگی‌ها به `Cargo.toml`

```toml
[package]
"Name = "todos-api
"Version = "0.1.0
"Edition = "2021

[dependencies]
"Actix-web = "4
Serde = { version = "1", features = ["derive"] }
"Serde_json = "1
Uuid = { version = "1", features = ["v4", "serde"] }
```

کد اصلی سرویس `src/main.rs`

```rust
;Use actix_web::{web, App, HttpServer, Responder, HttpResponse, Error}
```

```
;Use serde::{Serialize, Deserialize}
;Use uuid::Uuid
;Use std::sync::{Mutex, Arc}

[derive(Serialize, Deserialize, Clone, Debug)]#
} Struct Todo
    ,Id: Uuid
    ,Title: String
    ,Completed: bool
{

;<<<Type TodoList = Arc<Mutex<Vec<Todo

    Handler for getting all todos //
} <Async fn get_todos(data: web::Data<TodoList>) -> Result<impl Responder, Error
    ;()Let todos = data.lock().unwrap
        Ok(web::Json(todos.clone()))
{


    Handler for getting a specific todo //
    )Async fn get_todo
        ,<Id: web::Path<Uuid
        ,<Data: web::Data<TodoList
    } <Result<impl Responder, Error <- (
    ;()Let todos = data.lock().unwrap

} If let Some(todo) = todos.iter().find(|todo| todo.id == *id)
```

```
                Ok(web::Json(todo.clone()))
            } else {
Ok(HttpResponse::NotFound().body("Todo not found"))
        {
    {

Handler for creating a new todo //
)Async fn create_todo
,<Todo: web::Json<Todo
,<Data: web::Data<TodoList
} <Result<impl Responder, Error <- (
;()Let mut todos = data.lock().unwrap
} Let new_todo = Todo
,()Id: Uuid::new_v4
,()Title: todo.title.clone
,Completed: false
;{
;Todos.push(new_todo.clone())
Ok(web::Json(new_todo))
{

Handler for updating a todo //
)Async fn update_todo
,<Id: web::Path<Uuid
,<Updated_todo: web::Json<Todo
,<Data: web::Data<TodoList
} <Result<impl Responder, Error <- (
;()Let mut todos = data.lock().unwrap
```

```
} If let Some(todo) = todos.iter_mut().find(|todo| todo.id == *id)
                    ;()Todo.title = updated_todo.title.clone
                ;Todo.completed = updated_todo.completed
                        Ok(web::Json(todo.clone()))
                                    } else {
    Ok(HttpResponse::NotFound().body("Todo not found"))
                                            {
                                                {

                                    Handler for deleting a todo //
                                        )Async fn delete_todo
                                    ,<Id: web::Path<Uuid
                                ,<Data: web::Data<TodoList
                            } <Result<impl Responder, Error <- (
                        ;()Let mut todos = data.lock().unwrap
                                    ;()Let initial_len = todos.len

                                ;Todos.retain(|todo| todo.id != *id)

                                    } If todos.len() != initial_len
                            Ok(HttpResponse::Ok().body("Todo deleted"))
                                                }else{
    Ok(HttpResponse::NotFound().body("Todo not found"))
                                            {
                                                {



                                    [actix_web::main]#
```

```
} <()>Async fn main() -> std::io::Result

;Let todos: TodoList = Arc::new(Mutex::new(Vec::new()))

} || HttpServer::new(move

()App::new

app_data(web::Data::new(todos.clone())).

)service.

Web::resource("/todos")

route(web::get().to(get_todos)).

route(web::post().to(create_todo)).

(

)service.

Web::resource("/todos/{id}")

route(web::get().to(get_todo)).

route(web::put().to(update_todo)).

route(web::delete().to(delete_todo)).

(

({

?bind("127.0.0.1:8080").

()run.

await.

{
```

توضیحات کد

1. **وابستگی‌ها**:
   * `actix-web`: برای ایجاد سرور وب و مدیریت مسیرها (Routing).

* `serde`: برای تبدیل ساختار داده‌ها به JSON و برعکس.
* `serde_json`: برای کار با JSON.
* `uuid`: برای تولید شناسه های منحصر به فرد.
* `std::sync::{Mutex, Arc}`: برای مدیریت دسترسی همزمان به داده‌ها.

2. **ساختار داده `Todo`**:
   * `id`: یک شناسه یکتا برای هر وظیفه.
   * `title`: عنوان وظیفه.
   * `completed`: نشان دهنده انجام یا عدم انجام وظیفه.

3. **`TodoList`**:
   * `Arc<Mutex<Vec<Todo>>>`: یک لیست از وظایف که از طریق `Arc` و `Mutex` برای مدیریت دسترسی همزمان به داده ها به اشتراک گذاشته می‌شود.

4. **مسیرها (Routes)**:
   * `/todos`:
     * `GET`: دریافت تمام وظایف.
     * `POST`: ایجاد یک وظیفه جدید.
   * `/todos/{id}`:
     * `GET`: دریافت یک وظیفه خاص بر اساس شناسه.
     * `PUT`: به‌روزرسانی یک وظیفه خاص.
     * `DELETE`: حذف یک وظیفه خاص.

5. **توابع مدیریت کننده (Handlers)**:
   * `get_todos`: بازیابی لیست وظایف.
   * `get_todo`: بازیابی یک وظیفه خاص.
   * `create_todo`: ایجاد یک وظیفه جدید.
   * `update_todo`: به روز رسانی یک وظیفه خاص.

* `delete_todo`: حذف یک وظیفه خاص.

6. **`main` تابع**:
   * مقداردهی اولیه لیست وظایف.
   * راه‌اندازی سرور HTTP با استفاده از `HttpServer`.
   * تعریف مسیرهای API با استفاده از `App` و `service`.
   * متصل کردن سرور به آدرس `127.0.0.1:8080`.

اجرا

برای اجرا، از دستور زیر در ترمینال استفاده کنید:

```bash
Cargo run
```

تست API

می‌توانید از ابزارهایی مانند `curl` یا Postman برای تست API استفاده کنید.

- **دریافت تمام وظایف:**

```bash
Curl http://127.0.0.1:8080/todos
```

- **ایجاد یک وظیفه جدید:**

bash```

Curl -X POST -H "Content-Type: application/json" -d '{"title": "Buy groceries"}' http://127.0.0.1:8080/todos

```

- **دریافت یک وظیفه خاص (جایگزین `id` با مقدار واقعی کنید):**

bash```

Curl http://127.0.0.1:8080/todos/{id}

```

- **به‌روزرسانی یک وظیفه خاص (جایگزین `id` با مقدار واقعی کنید):**

bash```

Curl -X PUT -H "Content-Type: application/json" -d '{"title": "Pay bills", "completed": true}' http://127.0.0.1:8080/todos/{id}

```

- **حذف یک وظیفه خاص (جایگزین `id` با مقدار واقعی کنید):**

bash```

Curl -X DELETE http://127.0.0.1:8080/todos/{id}

```

این مثال یک سرویس RESTful API پایه را نشان می‌دهد که درخواست‌های JSON را پردازش می‌کند و می‌تواند به عنوان نقطه شروع برای توسعه APIهای پیچیده‌تر مورد استفاده قرار گیرد.

6- یک سرویس ساده جهت پردازش درخواست‌های مبتنی بر پروتکلgRPCبنویسید

بله، حتماً. در اینجا یک سرویس ساده برای پردازش درخواست‌های مبتنی بر پروتکل gRPC با استفاده از زبان Rust و کتابخانه `tonic` می‌نویسم. این مثال یک سرویس ساده برای مدیریت یک لیست از وظایف (todos) را نشان می‌دهد.

پیش‌نیازها

1. **نصب Rust:** اگر Rust را نصب نکرده‌اید، از وب‌سایت رسمی Rust اقدام کنید.
2. **نصب Protobuf Compiler:** برای کامپایل فایل های `.proto` به کد Rust، کامپایلر Protobuf را نصب کنید. (برای مثال در لینوکس `sudo apt install protobuf-compiler` و در مک با استفاده از `brew install protobuf`).
3. **نصب `protoc-gen-grpc-web` plugin:** این پلاگین برای تولید کد های تحت وب grpc مورد استفاده قرار می‌گیرد، که با دستور `npm install -g protoc-gen-grpc-web` قابل نصب است.
4. **نصب وابستگی‌ها:** از طریق Cargo می‌توانید `prost` ، `tonic` و `tokio` را به پروژه خود اضافه کنید.

ایجاد یک پروژه جدید Cargo

```bash
Cargo new grpc-todos
Cd grpc-todos
```

افزودن وابستگی‌ها به `Cargo.toml`

```toml
[package]
"Name = "grpc-todos
"Version = "0.1.0
"Edition = "2021

[dependencies]
"Tonic = "0.10
"Prost = "0.12
```

Tokio = { version = "1", features = ["macros", "rt-multi-thread"] }

Serde = { version = "1", features = ["derive"] }

Uuid = { version = "1", features = ["v4", "serde"] }

```

**تعریف پیام‌ها و سرویس در `proto/todos.proto`**

ابتدا فایل `proto/todos.proto` را ایجاد کنید و تعاریف مربوط به پیام‌ها و سرویس را در آن قرار دهید:

protobuf```

;"Syntax = "proto3

;Package todos

} Message Todo

;String id = 1

;String title = 2

;Bool completed = 3

{

} Message GetTodoRequest

;String id = 1

{

} Message CreateTodoRequest

;String title = 1

{
```

```
} Message UpdateTodoRequest
    ;String id = 1
    ;String title = 2
    ;Bool completed = 3
{

} Message DeleteTodoRequest
    ;String id = 1
{

{} Message Empty

} Service Todos
    ;Rpc GetTodos (Empty) returns (stream Todo)
    ;Rpc GetTodo (GetTodoRequest) returns (Todo)
    ;Rpc CreateTodo (CreateTodoRequest) returns (Todo)
    ;Rpc UpdateTodo (UpdateTodoRequest) returns (Todo)
    ;Rpc DeleteTodo (DeleteTodoRequest) returns (Empty)
{
```

کامپایل فایل `.proto`

برای کامپایل کردن فایل `.proto` به کد Rust، یک فایل `build.rs` در ریشه پروژه ایجاد کنید:

```rust
build.rs //
} <<Fn main() -> Result<(), Box<dyn std::error::Error
```

```
    ()Tonic_build::configure
        build_server(true).
    ;?compile(&["proto/todos.proto"], &["proto"]).
    (())Ok
}
```

کد اصلی سرویس `src/main.rs`

```rust
;Use tonic::{transport::Server, Request, Response, Status}
;Use std::sync::{Mutex, Arc}
;Use uuid::Uuid
;Use serde::{Serialize, Deserialize}
;Use futures::Stream
;Use std::pin::Pin
;Use std::time::Duration
;Use std::collections::HashMap


} Pub mod todos
    ;Tonic::include_proto!("todos")
{
}::Use todos
    ,Todos_server::{Todos, TodosServer}
    ,Empty
    ,Todo
    ,GetTodoRequest
```

```
                                                                ,CreateTodoRequest
                                                                ,UpdateTodoRequest
                                                                ,DeleteTodoRequest
                                                                                ;{

                                                    [derive(Debug, Clone, Serialize, Deserialize)]#
                                                                        }Struct TodoItem
                                                                            ,Id: Uuid
                                                                        ,Title: String
                                                                    ,Completed: bool
                                                                                {

                                    ;<<<Type TodoList = Arc<Mutex<HashMap<Uuid, TodoItem

                                                                    [derive(Default)]#
                                                                } Pub struct MyTodos
                                                                    Todos: TodoList
                                                                                {



                                                                    [tonic::async_trait]#
                                                                } Impl Todos for MyTodos
Type GetTodosStream = Pin<Box<dyn Stream<Item = Result<Todo, Status>> +
                                                                                ;<<Send

                                                                )Async fn get_todos
                                                                        ,self&
                                                                ,<request: Request<Empty_
                                    } <Result<Response<Self::GetTodosStream>, Status <- (
```

```
                                                    ;()Let todos_clone = self.todos.clone


                                                } !Let output = async_stream::stream
                                            ;()Let todos = todos_clone.lock().unwrap
                                                    } ()For todo in todos.values
                                                        } Let todo_res = Todo
                                                    ,()Id: todo.id.to_string
                                                    ,()Title: todo.title.clone
                                                ,Completed: todo.completed
                                                                        ;{


                                                        ;Yield Ok(todo_res)


                                                                            {


                                                                        ;{


                                    Ok(Response::new(Box::pin(output) as Self::GetTodosStream))
                                                                        {


                                                                )Async fn get_todo
                                                                    ,self&
                                                            ,<Request: Request<GetTodoRequest
                                                            } <Result<Response<Todo>, Status <- (
                                                Let id = Uuid::parse_str(request.get_ref().id.as_str())
                                    ;?map_err(|_| Status::invalid_argument("Invalid UUID format")).


                                                        ;()Let todos = self.todos.lock().unwrap
```

```
}If let Some(todo) = todos.get(&id)
                } Let res = Todo
            ,()Id: todo.id.to_string
           ,()Title: todo.title.clone
         ,Completed: todo.completed
                                ;{
              Ok(Response::new(res))
                            } else {
Err(Status::not_found("Todo not found"))
                                  {
                                    {

                    )Async fn create_todo
                                ,self&
          ,<Request: Request<CreateTodoRequest
        } <Result<Response<Todo>, Status <- (
;()Let mut todos = self.todos.lock().unwrap
                    ;()Let id = Uuid::new_v4
                } Let new_todo = TodoItem
                                   ,Id
          ,()Title: request.get_ref().title.clone
                         ,Completed: false
                                ;{

            ;Todos.insert(id, new_todo.clone())

                    } Let res = Todo
```

```
                ,()Id: new_todo.id.to_string
                ,()Title: new_todo.title.clone
                Completed: new_todo.completed
                                        ;{

                                Ok(Response::new(res))
                                        {



                        )Async fn update_todo
                                        ,self&
                ,<Request: Request<UpdateTodoRequest
                        } <Result<Response<Todo>, Status <- (
        Let id = Uuid::parse_str(request.get_ref().id.as_str())
;?map_err(|_| Status::invalid_argument("Invalid UUID format")).

                        ;()Let mut todos = self.todos.lock().unwrap

                        }If let Some(todo) = todos.get_mut(&id)
                ;()Todo.title = request.get_ref().title.clone
                ;Todo.completed = request.get_ref().completed
                                        } Let res = Todo
                                ,()Id: todo.id.to_string
                                ,()Title: todo.title.clone
                        Completed: todo.completed
                                                ;{
                                Ok(Response::new(res))
                                        } else {
```

```
                Err(Status::not_found("Todo not found"))
                                                    {

                                                      {


                                    )Async fn delete_todo
                                                 ,self&
                                ,<Request: Request<DeleteTodoRequest
                                  } <Result<Response<Empty>, Status <- (

                        Let id = Uuid::parse_str(request.get_ref().id.as_str())
    ;?map_err(|_| Status::invalid_argument("Invalid UUID format")).

                                    ;()Let mut todos = self.todos.lock().unwrap

                                          } ()If todos.remove(&id).is_some
                                            Ok(Response::new(Empty {}))
                                                          } else{
                                 Err(Status::not_found("Todo not found"))
                                                           {

                                                             {
                                                               {

                                                  [tokio::main]#
                                } <<Async fn main() -> Result<(), Box<dyn std::error::Error
                                                  ;?()Let addr = "[::1]:50051".parse
```

```
}Let todos = MyTodos

,Todos:  Arc::new(Mutex::new(HashMap::new()))

;{


;Println!("Server listening on: {:?}", addr)

()Server::builder

.add_service(TodosServer::new(todos))

.serve(addr)

;?await.


(())Ok

{
```

توضیحات کد

1. **وابستگی‌ها**:
   * `tonic`: برای ایجاد سرور gRPC و مدیریت درخواست‌ها.
   * `prost`: برای تبدیل ساختار داده‌ها از فایل `.proto` به Rust.
   * `tokio`: برای اجرای کد آسنکرون.
   * `serde`: برای سریالیزه کردن داده ها در صورت نیاز
   * `uuid`: برای تولید شناسه های منحصر به فرد
   * `futures::Stream`: برای تعریف ساختار استریم در gRPC
   * `std::pin::Pin`: برای ایجاد پین شده به استریم های تولید شده
2. **فایل `.proto`**:
   * تعریف سرویس `Todos` با متدهای `GetTodos`، `GetTodo`، `CreateTodo`، `UpdateTodo`` و `DeleteTodo`.
   * تعریف پیام‌های مورد نیاز برای این سرویس مانند `GetTodoRequest`، `Todo` و...
3. **کامپایل فایل `.proto`**:

* فایل `build.rs` باعث ایجاد کد های `Rust` از روی فایل `proto` میشود.

4. **ساختار داده `Todo`**:
   * `id`: یک شناسه یکتا برای هر وظیفه.
   * `title`: عنوان وظیفه.
   * `completed`: نشان دهنده انجام یا عدم انجام وظیفه.

5. **`TodoList`**:
   * `Arc<Mutex<HashMap<Uuid, TodoItem>>>`: یک لیست از وظایف که از طریق `Arc` و `Mutex` برای مدیریت دسترسی همزمان به داده ها به اشتراک گذاشته می‌شود.

6. **`MyTodos`**:
   * ساختار اصلی سرور است که داده ها را نگهداری میکند.

7. **پیاده‌سازی سرویس `Todos`**:
   * `get_todos`: دریافت تمام وظایف در قالب یک استریم.
   * `get_todo`: بازیابی یک وظیفه خاص بر اساس شناسه.
   * `create_todo`: ایجاد یک وظیفه جدید.
   * `update_todo`: به روز رسانی یک وظیفه خاص.
   * `delete_todo`: حذف یک وظیفه خاص.

8. **`main` تابع**:
   * مقداردهی اولیه ساختار داده `MyTodos`.
   * راه‌اندازی سرور gRPC با استفاده از `Server::builder()`.
   * متصل کردن سرویس به آدرس `[::1]:50051`.

اجرا

برای اجرا، از دستور زیر در ترمینال استفاده کنید:

```bash
Cargo run
```

```

**تست API**

برای تست API، می‌توانید از ابزارهایی مانند `grpcurl` یا یک کلاینت gRPC استفاده کنید.

**نصب `grpcurl`**

اگر `grpcurl` را ندارید، می‌توانید آن را نصب کنید:

```bash
Go install github.com/fullstorydev/grpcurl/cmd/grpcurl@latest
```

تست با `grpcurl`

- **دریافت تمام وظایف:**

```bash
Grpcurl -plaintext -d '{}' [::1]:50051 todos.Todos/GetTodos
```

- **ایجاد یک وظیفه جدید:**

```bash
Grpcurl -plaintext -d '{"title": "Buy groceries"}' [::1]:50051 todos.Todos/CreateTodo
```
```

- **دریافت یک وظیفه خاص (جایگزین `id` با مقدار واقعی کنید):**

bash```

Grpcurl -plaintext -d '{"id": "{id}"}' [::1]:50051 todos.Todos/GetTodo

```

- **به‌روزرسانی یک وظیفه خاص (جایگزین `id` با مقدار واقعی کنید):**

bash```

Grpcurl -plaintext -d '{"id":"{id}","title": "Pay bills", "completed": true}' [::1]:50051 todos.Todos/UpdateTodo

```

- **حذف یک وظیفه خاص (جایگزین `id` با مقدار واقعی کنید):**

bash```

Grpcurl -plaintext -d '{"id": "{id}"}' [::1]:50051 todos.Todos/DeleteTodo

```

**نکات مهم**

* **استریم در gRPC**: در این مثال از استریم در متد `GetTodos` برای ارسال تمام وظایف به صورت یک استریم استفاده شده است، که باعث می شود عملکرد بهتری در ارسال داده های زیاد ایجاد شود.

* **مدیریت خطا**: با استفاده از `Result` و `Status` در gRPC، مدیریت خطا به صورت استاندارد انجام می شود.

* **دسترسی همزمان**: استفاده از `Arc<Mutex<HashMap<>>>` برای دسترسی همزمان به داده ها در سرور.

این مثال یک سرویس gRPC پایه را نشان می‌دهد که می‌تواند به عنوان نقطه شروع برای توسعه سرویس‌های پیچیده‌تر gRPC در Rust مورد استفاده قرار گیرد
```

7- ساختمان دادهMaxheapTree را در زبان Rust پیاده سازی نمایید ؟

بله، حتماً. در اینجا پیاده‌سازی یک Max Heap (درخت Maxheap) در زبان Rust به همراه توضیح هر بخش را ارائه می‌دهم. این پیاده‌سازی شامل توابع اصلی برای ایجاد، درج، حذف و مرتب‌سازی است.

ساختار داده Max Heap

ابتدا ساختار Max Heap را تعریف می‌کنیم. Max Heap معمولاً با استفاده از یک آرایه پیاده‌سازی می‌شود، جایی که هر گره در موقعیت `i`، فرزندانش در موقعیت‌های `2*1 + l` و `2*2 + l` قرار دارند.

rust```

[derive(Debug)]#

} <Pub struct MaxHeap<T: Ord

,<Heap: Vec<T

{

```

در اینجا:

* `T: Ord`: یعنی نوع داده‌ای که در heap ذخیره می‌شود باید قابل مقایسه باشد.
* `<heap: Vec<T`: یک بردار (Vector) که عناصر heap را نگه می‌دارد.

**توابع کمکی**

چند تابع کمکی برای محاسبه موقعیت پدر و فرزندان و همچنین برای جابجایی عناصر نیاز داریم.

rust```

} <Impl<T: Ord> MaxHeap<T

} Fn parent_index(index: usize) -> usize

(index – 1) / 2
```

{

} Fn left_child_index(index: usize) -> usize

index + 1 * 2

{

} Fn right_child_index(index: usize) -> usize

index + 2 * 2

{

} Fn swap(&mut self, i: usize, j: usize)

;Self.heap.swap(I, j)

{

{

```

**ایجاد Max Heap**

تابع `new` یک Max Heap خالی ایجاد می‌کند.

```rust

} <Impl<T: Ord> MaxHeap<T

} Pub fn new() -> Self

MaxHeap { heap: Vec::new() }

{

{

```
```

درج عنصر (Insertion)

برای درج عنصر در Max Heap، ابتدا عنصر را به انتهای heap اضافه می‌کنیم و سپس آن را به سمت بالا تا جایی که شرط Max Heap برقرار شود، منتقل می‌کنیم.

```rust
impl<T: Ord> MaxHeap<T> {
    pub fn insert(&mut self, value: T) {
        self.heap.push(value);
        self.heapify_up();
    }

    fn heapify_up(&mut self) {
        let mut index = self.heap.len() - 1;
        while index > 0 && self.heap[index] > self.heap[Self::parent_index(index)] {
            let parent_index = Self::parent_index(index);
            self.swap(index, parent_index);
            index = parent_index;
        }
    }
}
```

حذف عنصر (Removal)

برای حذف عنصر ریشه (بزرگترین عنصر) از Max Heap، ابتدا عنصر ریشه را با آخرین عنصر در heap جابجا می‌کنیم، سپس آخرین عنصر را حذف کرده و عنصر جدید در ریشه را به سمت پایین تا جایی که شرط Max Heap برقرار شود، منتقل می‌کنیم.

```rust
} <Impl<T: Ord> MaxHeap<T
} <Pub fn remove(&mut self) -> Option<T
} ()If self.heap.is_empty
;Return None
{

;Let last_index = self.heap.len() – 1
;Self.swap(0, last_index)

;()Let removed_value = self.heap.pop
;Self.heapify_down(0)

Removed_value
{

} Fn heapify_down(&mut self, mut index: usize)
} Loop
;Let left_child_index = Self::left_child_index(index)
;Let right_child_index = Self::right_child_index(index)
;Let mut largest = index

If left_child_index < self.heap.len() && self.heap[left_child_index] >
} self.heap[largest]
;Largest = left_child_index
{

If right_child_index < self.heap.len() && self.heap[right_child_index] >
} self.heap[largest]
```

```
                ;Largest = right_child_index
                                    {


                            } If largest != index
                    ;Self.swap(index, largest)
                            ;Index = largest
                                } else {
                            ;Break
                                    {
                                {
                                    {
                                    {
```

تابع peek برای دیدن بزرگترین مقدار

```rust
} <Impl<T: Ord + Copy> MaxHeap<T
} <Pub fn peek(&self) -> Option<T
        ()Self.heap.first().copied
                                    {
                                    {
```

مرتب‌سازی (Sort)

با استفاده از توابع `insert` و `remove` می‌توان یک مجموعه داده را به صورت صعودی مرتب کرد.

```rust
```

```
} <Impl<T: Ord + Copy> MaxHeap<T
} <Pub fn sort(mut vec: Vec<T>) -> Vec<T
;()Let mut max_heap = MaxHeap::new
} For item in vec
;Max_heap.insert(item)
{
;()Let mut sorted_vec = Vec::new
} ()While let Some(item) = max_heap.remove
;Sorted_vec.push(item)
{
Sorted_vec
{
{
```

مثال استفاده

```rust
} ()Fn main
;()Let mut max_heap: MaxHeap<i32> = MaxHeap::new
;Max_heap.insert(5)
;Max_heap.insert(3)
;Max_heap.insert(8)
;Max_heap.insert(1)
;Max_heap.insert(9)

;Println!("Max Heap: {:?}", max_heap)
```

```
    ;Println!("Peek: {:?}", max_heap.peek())

    } ()While let Some(value) = max_heap.remove
        ;Println!("Removed Value: {}", value)
    {

;Println!("Max Heap after all removals: {:?}", max_heap)

;Let unsorted_vec = vec![5, 3, 8, 1, 9]
;Let sorted_vec = MaxHeap::sort(unsorted_vec)
Println!("Sorted vector: {:?}", sorted_vec); // Output: [1, 3, 5, 8, 9]
{
```

نکات مهم

* **Generic Type `T`**: با استفاده از `T: Ord`، این ساختار می‌تواند برای انواع مختلف داده‌ها کار کند به شرطی که قابل مقایسه باشند.

* **آرایه**: استفاده از بردار (`Vec`) برای پیاده‌سازی heap امکان دسترسی سریع و آسان به عناصر را فراهم می‌کند.

* **پیاده‌سازی کارآمد**: الگوریتم‌های `heapify_up` و `heapify_down` پیاده‌سازی کارآمدی برای حفظ ساختار heap هستند.

* **مرتب‌سازی**: تابع `sort` نشان می‌دهد که چگونه می‌توان از Max Heap برای مرتب‌سازی مجموعه‌ای از داده‌ها استفاده کرد.

این پیاده‌سازی یک شروع خوب برای کار با Max Heap در Rust است. می‌توانید آن را گسترش داده و بر اساس نیازهای خاص خود بهینه کنید.

8- ساختمان داده AVLTreeرادرزبان Rustپیاده سازی نمایید؟

باشه، حتماً. در اینجا پیاده‌سازی یک درخت AVL در زبان Rust را به همراه توضیح هر بخش می‌آورم. این کد یک ساختار کلی و پایه‌ای را ارائه می‌دهد که می‌توانید بر اساس نیازهای خاص خود آن را گسترش دهید.

ساختار گره (Node Structure)

ابتدا ساختار گره را تعریف می‌کنیم. هر گره شامل یک مقدار، ارتفاع (height) و اشاره‌گر به گره‌های فرزند چپ و راست است.

```
Rust

[derive(Debug, PartialEq, Eq)]#

} <Struct Node<T: Ord + Copy

    ,Value: T

    ,Height: usize

    ,<<<Left: Option<Box<Node<T

    ,<<<Right: Option<Box<Node<T

{


} <Impl<T: Ord + Copy> Node<T

    } Fn new(value: T) -> Self

        } Node

            ,Value

            ,Height: 1

            ,Left: None

            ,Right: None

        {

    {

{
```

در اینجا:

T: Ord + Copy: یعنی نوع داده گره باید قابل مقایسه و کپی باشد.

T>Node>Option>Box>>>: برای اشاره به گره‌های فرزند، استفاده از Box برای مدیریت حافظه و Option برای نشان دادن اینکه ممکن است فرزند وجود نداشته باشد.

ساختار درخت AVL (AVLTree Structure)

ساختار اصلی درخت AVL شامل یک ریشه است.

Rust

```
#[derive(Debug)]
} <Pub struct AVLTree<T: Ord + Copy {
    ,<<<Root: Option<Box<Node<T
{
```

```
} <Impl<T: Ord + Copy> AVLTree<T {
    } Pub fn new() -> Self {
        AVLTree { root: None }
    {
{
```

توابع کمکی

چند تابع کمکی برای دریافت ارتفاع، به‌روزرسانی ارتفاع و انجام چرخش‌ها لازم داریم.

Rust

```
} <Impl<T: Ord + Copy> AVLTree<T {
    } Fn get_height(node: &Option<Box<Node<T>>>) -> usize {
        } Match node {
            ,Some(n) => n.height
            ,None => 0
        {
```

```
{

} Fn update_height(node: &mut Node<T>)

Node.height = std::cmp::max(Self::get_height(&node.left),
;Self::get_height(&node.right)) + 1

{

} Fn balance_factor(node: &Node<T>) -> isize

Self::get_height(&node.left) as isize – Self::get_height(&node.right) as isize

{

{
```

چرخش‌ها (Rotations)

برای حفظ تعادل درخت AVL، به چرخش‌های راست و چپ نیاز داریم.

```
Rust

} <Impl<T: Ord + Copy> AVLTree<T

} <<Fn rotate_right(mut node: Box<Node<T>>) -> Box<Node<T

;()Let mut new_root = node.left.take().unwrap

;()Node.left = new_root.right.take

;New_root.right = Some(node)

;Self::update_height(&mut new_root.right.as_mut().unwrap())

;Self::update_height(&mut new_root)

New_root

{

} <<Fn rotate_left(mut node: Box<Node<T>>) -> Box<Node<T
```

```
;()Let mut new_root = node.right.take().unwrap
;()Node.right = new_root.left.take
;New_root.left = Some(node)

;Self::update_height(&mut new_root.left.as_mut().unwrap())
;Self::update_height(&mut new_root)

New_root
{
{
```

متعادل‌سازی (Balancing)

بعد از هر درج یا حذف، باید درخت را متعادل کنیم.

```
Rust
} <Impl<T: Ord + Copy> AVLTree<T
} <<Fn balance(mut node: Box<Node<T>>) -> Box<Node<T
;Self::update_height(&mut node)

;Let balance_factor = Self::balance_factor(&node)

} If balance_factor > 1
} If Self::balance_factor(node.left.as_ref().unwrap()) < 0
;Node.left = Some(Self::rotate_left(node.left.take().unwrap()))
{
;Return Self::rotate_right(node)
{
} If balance_factor < -1
```

```
} If Self::balance_factor(node.right.as_ref().unwrap()) > 0
;Node.right = Some(Self::rotate_right(node.right.take().unwrap()))
{
;Return Self::rotate_left(node)
{
Node
{
{
```

درج گره (Insertion)

تابع درج گره به صورت بازگشتی انجام می‌شود و پس از درج، درخت را متعادل می‌کند.

Rust

```
} <Impl<T: Ord + Copy> AVLTree<T
Fn insert_recursive(node: Option<Box<Node<T>>>, value: T) ->
} <<<Option<Box<Node<T
} Match node
,None => Some(Box::new(Node::new(value)))
} <= Some(mut n)
} If value < n.value
;n.left = Self::insert_recursive(n.left, value)
} else {
;n.right = Self::insert_recursive(n.right, value)
{
Some(Self::balance(n))
{
{
{
```

} Pub fn insert(&mut self, value: T)

;Self.root = Self::insert_recursive(self.root.take(), value)

{

{

جستجو (Search)

9- برنامه‌ای به زبان Rustبنویسیدکه عملیات CRUDبرای یک پایگاه داده انجام دهد

بله، حتماً. در اینجا یک برنامه به زبان Rust می‌نویسم که عملیات CRUD (ایجاد، خواندن، به‌روزرسانی، حذف) را بر روی یک پایگاه داده SQLite انجام می‌دهد. این برنامه از کتابخانه `rusqlite` برای تعامل با پایگاه داده استفاده می‌کند.

پیش‌نیازها

1. **نصب Rust:** اگر Rust را نصب نکرده‌اید، از وب‌سایت رسمی Rust اقدام کنید.
2. **ایجاد پروژه Cargo:** با استفاده از دستور `cargo new –bin crud_app` یک پروژه جدید Rust ایجاد کنید.
3. **افزودن وابستگی‌ها:** به فایل `Cargo.toml` وابستگی `rusqlite` را اضافه کنید:

toml```

[dependencies]

"Rusqlite = "0.30

```

**کد برنامه**

در اینجا کد کامل برنامه `src/main.rs` آمده است:

rust```
```

```
;Use rusqlite::{Connection, Result, params}
;Use std::io::{self, Write}


} Struct User
    ,Id: i32
,Name: String
,Email: String
{


} <()>Fn create_table(conn: &Connection) -> Result
    )Conn.execute
        ) CREATE TABLE IF NOT EXISTS users"
    ,Id INTEGER PRIMARY KEY AUTOINCREMENT
        ,Name TEXT NOT NULL
    Email TEXT NOT NULL UNIQUE
        ,"(
        ,[]
        ;?(
    (())Ok
        {


} <()>Fn create_user(conn: &Connection, name: &str, email: &str) -> Result
    )Conn.execute
    ,"(? ,?) INSERT INTO users (name, email) VALUES"
        ,Params![name, email]
        ;?(
```

```
(())Ok
{

} <<Fn read_user(conn: &Connection, id: i32) -> Result<Option<User
;?Let mut stmt = conn.prepare("SELECT id, name, email FROM users WHERE id = ?")
;?Let mut rows = stmt.query(params![id])

} ?()If let Some(row) = rows.next
} Let user = User
,?Id: row.get(0)
,?Name: row.get(1)
,?Email: row.get(2)
;{
Ok(Some(user))
} else {
Ok(None)
{
{

} <()>Fn update_user(conn: &Connection, id: i32, name: &str, email: &str) -> Result
)Conn.execute
,"? = UPDATE users SET name = ?, email = ? WHERE id"
,Params![name, email, id]
;?(
(())Ok
{

} <()>Fn delete_user(conn: &Connection, id: i32) -> Result
```

```
;?Conn.execute("DELETE FROM users WHERE id = ?", params![id])
                                                        (())Ok
                                                            {

                                                } ()Fn print_menu
                                    ;Println!("\nCRUD Operations Menu:")
                                            ;Println!("1. Create User")
                                             ;Println!("2. Read User")
                                           ;Println!("3. Update User")
                                            ;Println!("4. Delete User")
                                                 ;Println!("5. Exit")
                                          ;Print!("Enter your choice: ")
                                         ;()Io::stdout().flush().unwrap
                                                            {



                                        } <()>Fn main() -> Result
                                    ;?Let conn = Connection::open("users.db")
                                             ;?Create_table(&conn)



                                                        } Loop
                                                  ;()Print_menu


                                          ;()Let mut choice = String::new
                    ;Io::stdin().read_line(&mut choice).expect("Failed to read line")
                                    ;Let choice: u32 = choice.trim().parse().unwrap_or(0)
```

```
                } Match choice
                    } <= 1
                ;Print!("Enter name: ")
            ;()Io::stdout().flush().unwrap
            ;()Let mut name = String::new
;Io::stdin().read_line(&mut name).expect("Failed to read line")
            ;()Let name = name.trim



                ;Print!("Enter email: ")
            ;()Io::stdout().flush().unwrap
            ;()Let mut email = String::new
;Io::stdin().read_line(&mut email).expect("Failed to read line")
            ;()Let email = email.trim



            ;?Create_user(&conn, name, email)
        ;Println!("User created successfully.")
                                {
                    } <= 2
                ;Print!("Enter user id: ")
            ;()Io::stdout().flush().unwrap
                ;()Let mut id = String::new
    ;Io::stdin().read_line(&mut id).expect("Failed to read line")
        ;Let id: i32 = id.trim().parse().unwrap_or(0)



                } ?Match read_user(&conn, id)
```

```
                } <= Some(user)
            ;Println!("User ID: {}", user.id)
          ;Println!("Name: {}", user.name)
           ;Println!("Email: {}", user.email)
                                    {
        ,None => println!("User not found.")
                                      {
                                        {
                                    } <= 3
                ;Print!("Enter user id to update: ")
                    ;()Io::stdout().flush().unwrap
                    ;()Let mut id = String::new
    ;Io::stdin().read_line(&mut id).expect("Failed to read line")
    ;Let id: i32 = id.trim().parse().unwrap_or(0)


                ;Print!("Enter new name: ")
                ;()Io::stdout().flush().unwrap
               ;()Let mut name = String::new
;Io::stdin().read_line(&mut name).expect("Failed to read line")
                ;()Let name = name.trim


                ;Print!("Enter new email: ")
                ;()Io::stdout().flush().unwrap
               ;()Let mut email = String::new
;Io::stdin().read_line(&mut email).expect("Failed to read line")
                ;()Let email = email.trim
```

```
                ;?Update_user(&conn, id, name, email)
                    ;Println!("User updated successfully.")
                                                {
                                            } <= 4
                        ;Print!("Enter user id to delete: ")
                            ;()Io::stdout().flush().unwrap
                                ;()Let mut id = String::new
;Io::stdin().read_line(&mut id).expect("Failed to read line")
                    ;Let id: i32 = id.trim().parse().unwrap_or(0)


                                ;?Delete_user(&conn, id)
                    ;Println!("User deleted successfully.")
                                                {
                                            } <= 5
                                ;Println!("Exiting...")
                                        ;Break
                                                {
                                ,println!("Invalid choice.") <= _
                                                    {
                                                        {


                                                ((())Ok
                                                        {
                                                    ```
```

**توضیحات کد**
```

1. **ساختار `User`:** ساختاری برای نگهداری اطلاعات کاربر.
2. **`create_table` تابع:** جدول `users` را در پایگاه داده ایجاد می‌کند.
3. **`create_user` تابع:** یک کاربر جدید در جدول `users` ایجاد می‌کند.
4. **`read_user` تابع:** یک کاربر را با استفاده از شناسه آن از جدول `users` خوانده و برمی‌گرداند.
5. **`update_user` تابع:** یک کاربر موجود را با استفاده از شناسه آن در جدول `users` به‌روزرسانی می‌کند.
6. **`delete_user` تابع:** یک کاربر را با استفاده از شناسه آن از جدول `users` حذف می‌کند.
7. **`main` تابع:**
   * یک اتصال به پایگاه داده `users.db` ایجاد می‌کند.
   * جدول `users` را ایجاد می‌کند (اگر وجود نداشته باشد).
   * یک حلقه بی‌نهایت ایجاد می‌کند که منو را نمایش می‌دهد و عملیات انتخاب شده توسط کاربر را اجرا می‌کند.
   * از کاربر اطلاعات ورودی را دریافت و عملیات مورد نظر را بر روی پایگاه داده انجام می‌دهد.

کامپایل و اجرا

1. برای کامپایل برنامه از دستور زیر استفاده کنید:

```bash
Cargo build
```

2. برای اجرای برنامه از دستور زیر استفاده کنید:

```bash
Cargo run
```

بعد از اجرای برنامه، یک منو نمایش داده می‌شود که می‌توانید از آن برای انجام عملیات CRUD استفاده کنید. پایگاه داده SQLite در فایل `users.db` در همان مسیر اجرایی برنامه ذخیره می‌شود.

نکات مهم

* **مدیریت خطا:** در این مثال از `?` برای مدیریت خطاها استفاده شده است. این یک روش معمول در Rust برای ساده‌سازی کد است.

* **پارامترهای پرس و جو:** استفاده از `params!` برای انتقال پارامترها به پرس و جوها، از حملات SQL Injection جلوگیری می‌کند.

* **رابط کاربری:** رابط کاربری این مثال ساده است، اما می‌توان آن را با استفاده از کتابخانه‌های UI بهبود بخشید.

* **پایداری داده:** پایگاه داده SQLite یک راه حل ساده و سبک برای ذخیره داده‌های محلی است. برای برنامه‌های بزرگتر، ممکن است نیاز به استفاده از پایگاه داده‌های قدرتمندتری باشد.

10- یک سرویس RESTfulAPIجهت پردازش درخواست‌های JSONبنویسید؟

بله، حتماً. در اینجا یک سرویس RESTful API با استفاده از زبان Rust و کتابخانه `actix-web` می‌نویسم که درخواست‌های JSON را پردازش می‌کند. این مثال یک API ساده برای مدیریت یک لیست از وظایف (todos) را نشان می‌دهد.

پیش‌نیازها

1. **نصب Rust:** اگر Rust را نصب نکرده‌اید، از وب‌سایت رسمی Rust اقدام کنید.
2. **نصب Actix-Web:** از طریق Cargo می‌توانید `actix-web` را به پروژه خود اضافه کنید.

ایجاد یک پروژه جدید Cargo

```bash
Cargo new todos-api
Cd todos-api
```

افزودن وابستگی‌ها به `Cargo.toml`

```toml
[package]
"Name = "todos-api
"Version = "0.1.0
"Edition = "2021

[dependencies]
"Actix-web = "4
Serde = { version = "1", features = ["derive"] }
"Serde_json = "1
Uuid = { version = "1", features = ["v4", "serde"] }
```

کد اصلی سرویس `src/main.rs`

```rust
Use actix_web::{web, App, HttpServer, Responder, HttpResponse, Error};
Use serde::{Serialize, Deserialize};
Use uuid::Uuid;
Use std::sync::{Mutex, Arc};

#[derive(Serialize, Deserialize, Clone, Debug)]
Struct Todo {
    Id: Uuid,
    Title: String,
    Completed: bool,
```

```
{

;<<<Type TodoList = Arc<Mutex<Vec<Todo

Handler for getting all todos //
} <Async fn get_todos(data: web::Data<TodoList>) -> Result<impl Responder, Error
;()Let todos = data.lock().unwrap
Ok(web::Json(todos.clone()))
{


Handler for getting a specific todo //
)Async fn get_todo
,<Id: web::Path<Uuid
,<Data: web::Data<TodoList
} <Result<impl Responder, Error <- (
;()Let todos = data.lock().unwrap

} If let Some(todo) = todos.iter().find(|todo| todo.id == *id)
Ok(web::Json(todo.clone()))
} else {
Ok(HttpResponse::NotFound().body("Todo not found"))
{
{

Handler for creating a new todo //
)Async fn create_todo
,<Todo: web::Json<Todo
```

```
    ,<Data: web::Data<TodoList
) <- Result<impl Responder, Error> {
    ;()Let mut todos = data.lock().unwrap
    } Let new_todo = Todo
        ,()Id: Uuid::new_v4
        ,()Title: todo.title.clone
        ,Completed: false
    ;{
    ;Todos.push(new_todo.clone())
    Ok(web::Json(new_todo))
}

// Handler for updating a todo
)Async fn update_todo
    ,<Id: web::Path<Uuid
    ,<Updated_todo: web::Json<Todo
    ,<Data: web::Data<TodoList
) <- Result<impl Responder, Error> {
    ;()Let mut todos = data.lock().unwrap
} If let Some(todo) = todos.iter_mut().find(|todo| todo.id == *id) {
    ;()Todo.title = updated_todo.title.clone
    ;Todo.completed = updated_todo.completed
    Ok(web::Json(todo.clone()))
} else {
    Ok(HttpResponse::NotFound().body("Todo not found"))
}
{
```

```
                                                    Handler for deleting a todo //
                                                        )Async fn delete_todo
                                                    ,<Id: web::Path<Uuid
                                                ,<Data: web::Data<TodoList
                                            } <Result<impl Responder, Error <- (
                                        ;()Let mut todos = data.lock().unwrap
                                                ;()Let initial_len = todos.len


                                                ;Todos.retain(|todo| todo.id != *id)


                                                    } If todos.len() != initial_len
                                            Ok(HttpResponse::Ok().body("Todo deleted"))
                                                                    }else{
                                    Ok(HttpResponse::NotFound().body("Todo not found"))
                                                                        {
                                                                          {



                                                                [actix_web::main]#
                                                    } <()>Async fn main() -> std::io::Result
                                    ;Let todos: TodoList = Arc::new(Mutex::new(Vec::new()))


                                                        } || HttpServer::new(move
                                                                ()App::new
                                                app_data(web::Data::new(todos.clone())).
                                                                )service.
                                                        Web::resource("/todos")
                                                route(web::get().to(get_todos)).
```

```
                route(web::post().to(create_todo)).
                (
                )service.
                Web::resource("/todos/{id}")
                route(web::get().to(get_todo)).
                route(web::put().to(update_todo)).
                route(web::delete().to(delete_todo)).
                (
            ({
        ?bind("127.0.0.1:8080").
            ()run.
            await.
        {
    ```

توضیحات کد

1. **وابستگی‌ها**:
    * `actix-web`: برای ایجاد سرور وب و مدیریت مسیرها (Routing).
    * `serde`: برای تبدیل ساختار داده‌ها به JSON و برعکس.
    * `serde_json`: برای کار با JSON.
    * `uuid`: برای تولید شناسه های منحصر به فرد.
    * `std::sync::{Mutex, Arc}`: برای مدیریت دسترسی همزمان به داده‌ها.

2. **ساختار داده `Todo`**:
    * `id`: یک شناسه یکتا برای هر وظیفه.
    * `title`: عنوان وظیفه.
    * `completed`: نشان دهنده انجام یا عدم انجام وظیفه.

3. **`TodoList`**:

   * `<<<Arc<Mutex<Vec<Todo>>>>>`: یک لیست از وظایف که از طریق `Arc` و `Mutex` برای مدیریت دسترسی همزمان به داده ها به اشتراک گذاشته می‌شود.

4. **مسیرها (Routes)**:
   * `/todos`:
     * `GET`: دریافت تمام وظایف.
     * `POST`: ایجاد یک وظیفه جدید.
   * `/todos/{id}`:
     * `GET`: دریافت یک وظیفه خاص بر اساس شناسه.
     * `PUT`: به‌روزرسانی یک وظیفه خاص.
     * `DELETE`: حذف یک وظیفه خاص.

5. **توابع مدیریت کننده (Handlers)**:
   * `get_todos`: بازیابی لیست وظایف.
   * `get_todo`: بازیابی یک وظیفه خاص.
   * `create_todo`: ایجاد یک وظیفه جدید.
   * `update_todo`: به روز رسانی یک وظیفه خاص.
   * `delete_todo`: حذف یک وظیفه خاص.

6. **تابع `main`**:
   * مقداردهی اولیه لیست وظایف.
   * راه‌اندازی سرور HTTP با استفاده از `HttpServer`.
   * تعریف مسیرهای API با استفاده از `App` و `service`.
   * متصل کردن سرور به آدرس `127.0.0.1:8080`.

اجرا

برای اجرا، از دستور زیر در ترمینال استفاده کنید:

```bash
Cargo run
```

تست API

می‌توانید از ابزارهایی مانند `curl` یا Postman برای تست API استفاده کنید.

- **دریافت تمام وظایف:**

```bash
Curl http://127.0.0.1:8080/todos
```

- **ایجاد یک وظیفه جدید:**

```bash
Curl -X POST -H "Content-Type: application/json" -d '{"title": "Buy groceries"}' http://127.0.0.1:8080/todos
```

- **دریافت یک وظیفه خاص (جایگزین `id` با مقدار واقعی کنید):**

```bash
Curl http://127.0.0.1:8080/todos/{id}
```

- **به‌روزرسانی یک وظیفه خاص (جایگزین `id` با مقدار واقعی کنید):**

```bash
Curl -X PUT -H "Content-Type: application/json" -d '{"title": "Pay bills", "completed": true}' http://127.0.0.1:8080/todos/{id}
```

- **حذف یک وظیفه خاص (جایگزین `id` با مقدار واقعی کنید):**

```bash
Curl -X DELETE http://127.0.0.1:8080/todos/{id}
```

این مثال یک سرویس RESTful API پایه را نشان می‌دهد که درخواست‌های JSON را پردازش می‌کند و می‌تواند به عنوان نقطه شروع برای توسعه APIهای پیچیده‌تر مورد استفاده قرار گیرد.

10- برنامه نویسی multi-Threadingدرزبان Rustرا با ذکر یک مثال ساده توضیح دهید ؟

توضیحات:

Use std::thread; و use std::time::Duration;: این خطوط، ماژول‌های مورد نیاز را برای کار با رشته‌ها و زمان‌بندی وارد می‌کنند.

Thread::spawn(|| { ... });: این تابع یک رشته جدید ایجاد می‌کند. داخل || { ... } یک closure (تابع بی‌نام) است که کدی که در رشته جدید اجرا می‌شود را تعریف می‌کند.

For I in 1..=5 { ... }: در هر رشته، یک حلقه داریم که از 1 تا 5 می‌شمارد.

println!("Thread 1: Count {}", i); و println!("Thread 2: Count {}", i);: این خطوط، خروجی شمارش را در هر رشته چاپ می‌کنند.

Thread::sleep(Duration::from_millis(100));: برای شبیه‌سازی انجام کار و ایجاد مکث، رشته به مدت 100 میلی‌ثانیه می‌خوابد.

handle1.join().unwrap(); و handle2.join().unwrap();: این خطوط باعث می‌شوند که رشته اصلی منتظر بماند تا رشته‌های فرعی (thread1 و thread2) کار خود را به پایان برسانند. اگر از join استفاده نکنید، رشته اصلی ممکن است قبل از اتمام کار رشته‌های فرعی پایان یابد.

println!("Main thread: All threads finished.");: بعد از اینکه تمام رشته‌ها کار خود را انجام دادند، رشته اصلی این پیام را چاپ می‌کند.

نحوه اجرای کد:

مطمئن شوید که Rust و Cargo را نصب کرده‌اید.

یک پروژه جدید Rust ایجاد کنید: cargo new multi_thread_example

به دایرکتوری پروژه بروید: cd multi_thread_example

فایل src/main.rs را با کد بالا جایگزین کنید.

کد را اجرا کنید: cargo run

خروجی:

خروجی ممکن است به ترتیب دقیق زیر نباشد، زیرا رشته‌ها به صورت همزمان اجرا می‌شوند و ترتیب خروجی آن‌ها ممکن است متغیر باشد. ولی چیزی شبیه به این است:

Thread 1: Count 1

Thread 2: Count 1

Thread 1: Count 2

Thread 2: Count 2

Thread 1: Count 3

Thread 2: Count 3

Thread 1: Count 4

Thread 2: Count 4

Thread 1: Count 5

Thread 2: Count 5

Main thread: All threads finished.

این مثال ساده نشان می‌دهد که چگونه می‌توان از چندین رشته در Rust برای انجام کارهای همزمان استفاده کرد. Rust با داشتن سیستم مالکیت (Ownership) و قرض‌گیری (Borrowing) از خطاهای متداول در برنامه‌نویسی همروند جلوگیری می‌کند. همچنین، ابزارهای دیگری مانند Mutex و Channel برای مدیریت دسترسی به داده‌ها و تبادل پیام بین رشته‌ها در Rust موجود است که می‌توانید از آن‌ها استفاده کنید.

11- در مورد برنامه‌نویسی موازی ((Parallel Programmingدر زبان ،Rustباید بگویم که Rustابزارهای قدرتمندی برای استفاده از توان پردازشی چند هسته‌ای فراهم می‌کند. تفاوت اصلی بین برنامه‌نویسی همروند ((Concurrentو موازی ((Parallelدر این است که:

* **همروندی (Concurrency):** به ساختاری مربوط می‌شود که در آن چندین کار می‌توانند به نظر برسند که همزمان در حال اجرا هستند، اما لزوماً به معنای اجرای واقعی همزمان در چندین هسته نیست.
* **موازی (Parallelism):** به اجرای واقعی همزمان چندین کار در چندین هسته پردازنده اشاره دارد.

Rust از مفاهیم هم‌روندی برای ساخت برنامه‌های موازی استفاده می‌کند، اما تمرکز اصلی آن بر روی ایجاد ایمنی و کارایی در این نوع برنامه‌نویسی است.

مفاهیم کلیدی در برنامه‌نویسی موازی Rust:

* **Thread:** واحد اصلی اجرای همزمان که در Rust استفاده می‌شود.
* **Rayon:** یک کتابخانه محبوب برای موازی‌سازی راحت‌تر و کارآمدتر برنامه‌ها در Rust.
* **Iterators (تکرارکننده‌ها):** توابعی که به شما اجازه می‌دهند روی داده‌ها به صورت ترتیبی پیمایش کنید. Rayon به شما اجازه می‌دهد تا تکرارکننده‌ها را به صورت موازی اجرا کنید.
* **Data Parallelism (موازی‌سازی داده):** رویکردی که در آن یک عملیات روی بخش‌های مختلف داده‌ها به صورت موازی اعمال می‌شود.
* **Ownership and Borrowing (مالکیت و قرض‌گیری):** مفاهیمی که Rust برای مدیریت حافظه و جلوگیری از خطاهای مربوط به همروندی از آنها استفاده می‌کند.

مثال ساده برنامه‌نویسی موازی با Rayon در Rust:

در این مثال، ما یک آرایه بزرگ از اعداد را ایجاد می‌کنیم و سپس مجموع مربع‌های اعداد را به صورت موازی محاسبه می‌کنیم.

```rust
use rayon::prelude::*;

fn main() {
```

```
// 1. ایجاد یک آرایه بزرگ از اعداد
let numbers: Vec<i32> = (0..100000).collect();

// 2. محاسبه موازی مربع اعداد و جمع آنها
let sum_of_squares: i64 = numbers
    .par_iter() // تبدیل به یک تکرارکننده موازی
    .map(|&x| x as i64 * x as i64) // مربع اعداد
    .sum(); // جمع اعداد

// 3. چاپ نتیجه
println!("Sum of squares: {}", sum_of_squares);
}
```

توضیحات:

1. **`use rayon::prelude::*;`**: این خط، ماژول‌های مورد نیاز از کتابخانه Rayon را وارد می‌کند.
2. **`let numbers: Vec<i32> = (0..100000).collect();`**: یک آرایه از اعداد صحیح از 0 تا 99,999 ایجاد می‌شود.
3. **`numbers.par_iter()`**: این متد یک تکرارکننده موازی ایجاد می‌کند که به شما اجازه می‌دهد عملیات روی عناصر آرایه را به صورت همزمان در چندین هسته اجرا کنید.
4. **`map(|&x| x as i64 * x as i64)`**: این متد یک تابع را روی هر عنصر تکرارکننده اعمال می‌کند و مربع عدد را محاسبه می‌کند. به یاد داشته باشید که برای جلوگیری از سرریز شدن اعداد از نوع `i32` به `i64` تبدیل می شوند.
5. **`sum()`**: این متد مجموع تمامی اعداد محاسبه شده را محاسبه می‌کند.
6. **`println!("Sum of squares: {}", sum_of_squares);`**: در نهایت، نتیجه جمع مربع‌ها چاپ می‌شود.

نحوه اجرای کد:

1. مطمئن شوید که Rust و Cargo را نصب کرده‌اید.
2. یک پروژه جدید Rust ایجاد کنید: ``cargo new parallel_example
3. به دایرکتوری پروژه بروید: ``cd parallel_example
4. فایل `Cargo.toml` را با محتوای زیر ویرایش کنید:

```toml
[package]
"Name = "parallel_example
"Version = "0.1.0
"Edition = "2021

[dependencies]
"Rayon = "1.8
```

5. فایل `src/main.rs` را با کد بالا جایگزین کنید.
6. کد را اجرا کنید: ``cargo run

چرا Rayon؟

* **سادگی:** Rayon استفاده از موازی‌سازی را بسیار ساده می‌کند. شما فقط باید از `par_iter()` و دیگر متدهای مشابه استفاده کنید.
* **کارایی:** Rayon به طور خودکار از تعداد هسته‌های موجود در سیستم استفاده می‌کند و بهینه‌سازی‌های لازم را برای اجرای موازی انجام می‌دهد.
* **ایمنی:** Rayon بر مبنای سیستم مالکیت و قرض‌گیری Rust بنا شده است، بنابراین از خطاهای احتمالی در برنامه‌نویسی موازی جلوگیری می‌کند.

خروجی:

خروجی برنامه به شکل زیر است:

```
Sum of squares: 333328333350
```

این مثال ساده نشان می‌دهد که چگونه می‌توان با استفاده از کتابخانه Rayon برنامه‌های موازی در Rust نوشت و کارایی برنامه‌ها را به میزان قابل توجهی افزایش داد. Rayon به شما اجازه می‌دهد تا به راحتی روی داده‌ها به صورت موازی عملیات انجام دهید بدون اینکه نیاز به مدیریت پیچیده thread ها داشته باشید.

گروه

سمانه بهاری

سمیرا صالحی

مریم شهدادی