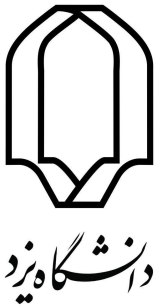


الگوریتم درخت کارتزینی

شروین خیرخواه

خرداد ۱۴۰۳

فهرست مطالب

۱ مقدمه و تاریخچه ایجاد الگوریتم

اولین بار الگوریتم درخت کارتزینی توسط "ژان وویمین" (Jean Vuillemin) در سال ۱۹۸۰ در زمینه ساختارهای داده جستجوی کران (range search data structure) معرفی شد.

هدف اولیه از معرفی این الگوریتم، ایجاد ساختاری کارآمد برای یافتن حداقل یا حداکثر مقدار در یک بازه از آرایه بود.

درخت‌های کارتزینی به دلیل سادگی، کارایی و قابلیت تعمیم به سرعت محبوبیت پیدا کردند و در زمینه‌های مختلف علوم کامپیوتر از جمله جستجوی دودویی، حل مسائل مربوط به بازه‌ها و ساختارهای داده تَرَپ (Treap) مورد استفاده قرار گرفتند.

ژان وویمین

۲ علت ایجاد الگوریتم

در آن زمان، ساختارهای داده مختلفی برای ذخیره و جستجوی اطلاعات وجود داشت. برخی از این ساختارها مانند آرایه‌ها و لیست‌های پیوسته، برای جستجوی عناصر به صورت تصادفی کارآمد بودند، اما برای یافتن حداقل یا حداکثر مقدار در یک بازه از آرایه، کارایی لازم را نداشتند.

ساختارهای داده دیگری مانند درخت‌های دودویی جستجو، برای یافتن حداقل یا حداکثر مقدار در یک بازه کارآمدتر بودند، اما درج و حذف عناصر در آن‌ها می‌توانست عملیات پرهزینه‌ای باشد.

الگوریتم درخت کارتزینی با ارائه روشی برای ساخت یک درخت دودویی متعادل از یک آرایه، این چالش‌ها را حل کرد. این درخت‌های متعادل، عملیات جستجو، درج و حذف را در زمان (log n) انجام می‌دهند، که در مقایسه با سایر ساختارهای داده موجود در آن زمان، بسیار کارآمدتر بود.

علاوه بر کارایی، درخت‌های کارتزینی به دلیل سادگی و قابلیت تعمیم نیز مورد توجه قرار گرفتند. پیاده‌سازی این الگوریتم نسبتاً ساده است و می‌توان از آن برای حل طیف وسیعی از مسائل مربوط به جستجو و پردازش بازه‌ها استفاده کرد.

۳ توضیح الگوریتم

۱.۳ درخت در علوم کامپیوتر

در علوم کامپیوتر، درخت (Tree) یک ساختار داده سلسله مراتبی است که از گره‌ها (Nodes) و یال‌ها (Edges) تشکیل شده است. در این ساختار، هر گره می‌تواند به صفر یا تعداد بیشتری گره دیگر به عنوان فرزند (Child) متصل باشد.

درخت‌ها به دلیل ساختار سازمان‌یافته و قابلیت پیمایش کارآمد، در زمینه‌های مختلف علوم کامپیوتر کاربردهای فراوانی دارند.

ویژگی‌های کلیدی درخت:

ریشه: گره‌ای که در بالاترین سطح درخت قرار دارد و هیچ فرزندی ندارد.

فرزندان: گره‌هایی که به طور مستقیم به یک گره دیگر متصل هستند.

سطح: فاصله گره از ریشه. ریشه در سطح (۰) قرار دارد.

زیر درخت: شامل ریشه و تمام فرزندان و نوادگان آن.

برگ: گره‌ای که هیچ فرزندی ندارد.

درجه: تعداد فرزندان یک گره.

۲.۳ درخت دودویی

درخت دودویی (binary tree) یک ساختار داده بنیادی در علم کامپیوتر است. این ساختار سلسله مراتبی شبیه به یک درخت با یک گره خاص به نام ریشه (Root) است. گره‌های موجود در یک درخت دودویی می‌توانند حداکثر دو فرزند داشته باشند: یک فرزند چپ و یک فرزند راست. این فرزندان خود می‌توانند درخت‌های دودویی باشند و یک ساختار بازگشتی (Recursive) ایجاد کنند.

ویژگی‌های کلیدی یک درخت دودویی :

گره ریشه: نقطه شروع درخت است. هر درخت دودویی دقیقاً یک گره ریشه دارد.

فرزندان: هر گره به جز برگ‌ها (که در ادامه توضیح داده می‌شود)، می‌تواند تا دو فرزند داشته باشد: فرزند چپ و فرزند راست.

گره‌های برگ: گره‌هایی که هیچ فرزندی ندارند، برگ نامیده می‌شوند.

ساختار بازگشتی: یک درخت دودویی را می‌توان به زیر درخت‌های (Sub-Tree) کوچکتری تقسیم کرد. این زیر درخت‌ها خود می‌توانند درخت‌های دودویی باشند که یک ساختار سلسله مراتبی را ایجاد می‌کنند.

۳.۳ توده (Heap)

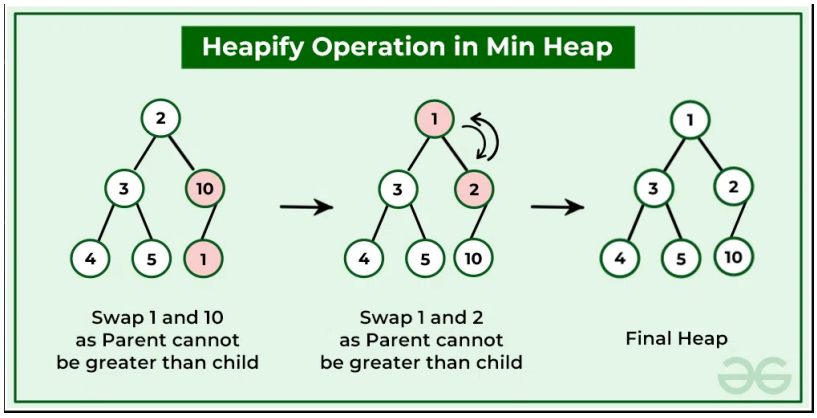
توده (Heap) یک ساختار داده درختی مبتنی بر آرایه است که از خاصیت توده پیروی می‌کند. در توده ماکس، مقدار هر گره بزرگتر یا مساوی با مقادیر هر دو فرزندش است. در مقابل، در توده مین، مقدار هر گره کوچکتر یا مساوی با مقادیر هر دو فرزندش است.

الگوریتم ساخت توده :

ایجاد توده اولیه: آرایه ورودی را به عنوان یک درخت دودویی کامل در نظر بگیرید. هر عنصر در آرایه به عنوان یک گره در درخت عمل می‌کند.

نظیم توده: از پایین‌ترین سطح درخت به سمت بالا حرکت کنید. برای هر گره، مقادیر آن و دو فرزندش را مقایسه کنید.

اگر مقدار گره کوچکتر از یکی از فرزندانش باشد، جای آن را با فرزند بزرگتر عوض کنید. این کار را به طور مکرر برای هر گره تا زمانی که به ریشه برسید، انجام دهید.



الگوریتم هیپ کردن

۴.۳ درخت کارتزینی چیست؟

درخت کارتزینی (cartesian tree) یک ساختار داده است که بر روی یک آرایه از اعداد تعریف می‌شود. در این ساختار، هر عضو آرایه به یک گره در درخت تبدیل می‌شود. ویژگی منحصر به فرد درخت کارتزینی این است که اگر پیمایش آن به صورت درون‌تراز (In-Order) انجام شود، ترتیب عناصر در خروجی دقیقاً با ترتیب عناصر در آرایه اصلی مطابقت دارد.

علاوه بر این، درخت کارتزینی خاصیت هرم کمینه (Heap Min) را نیز دارا است. به این معنی که در هر زیر درخت، کلید ریشه کوچک‌تر از کلید تمام گره‌های فرزند آن است.

۴ نحوه ساخت درخت کارتزینی

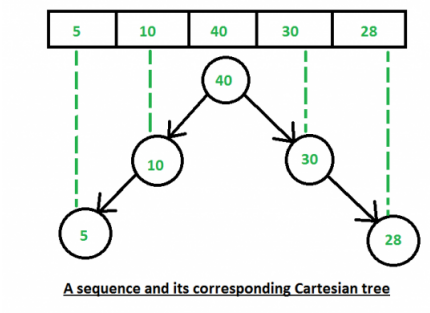
برای ساخت درخت کارتزینی از یک آرایه، مراحل زیر را دنبال می‌کنیم:

مقادیر کمینه را پیدا کنید: ابتدا، کمینه کل آرایه را پیدا می‌کنیم. این مقدار به عنوان ریشه درخت کارتزینی در نظر گرفته می‌شود.

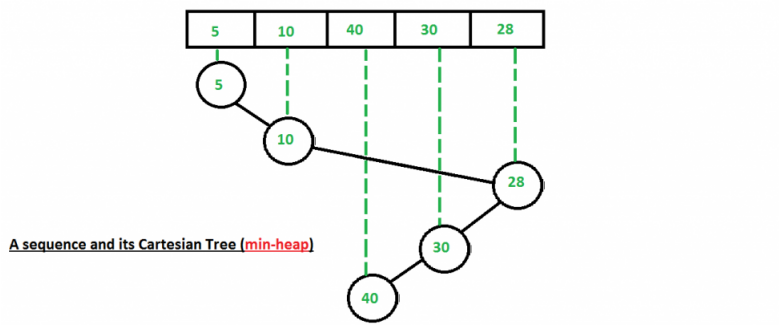
درخت‌های فرعی را بسازید: سپس به طور بازگشتی، دو درخت فرعی برای ریشه ایجاد می‌کنیم. درخت فرعی سمت چپ شامل تمام عناصری در آرایه است که کوچک‌تر از ریشه هستند. درخت فرعی سمت راست نیز شامل تمام عناصری در آرایه است که بزرگ‌تر از ریشه هستند.

درخت را کامل کنید: در نهایت، درخت فرعی سمت چپ را به عنوان فرزند چپ ریشه و درخت فرعی سمت راست را به عنوان فرزند راست ریشه قرار می‌دهیم.

با استفاده از این الگوریتم بازگشتی، می‌توان درخت کارتزینی را برای یک آرایه از n عنصر ساخت.



هیپ ماکسیمم

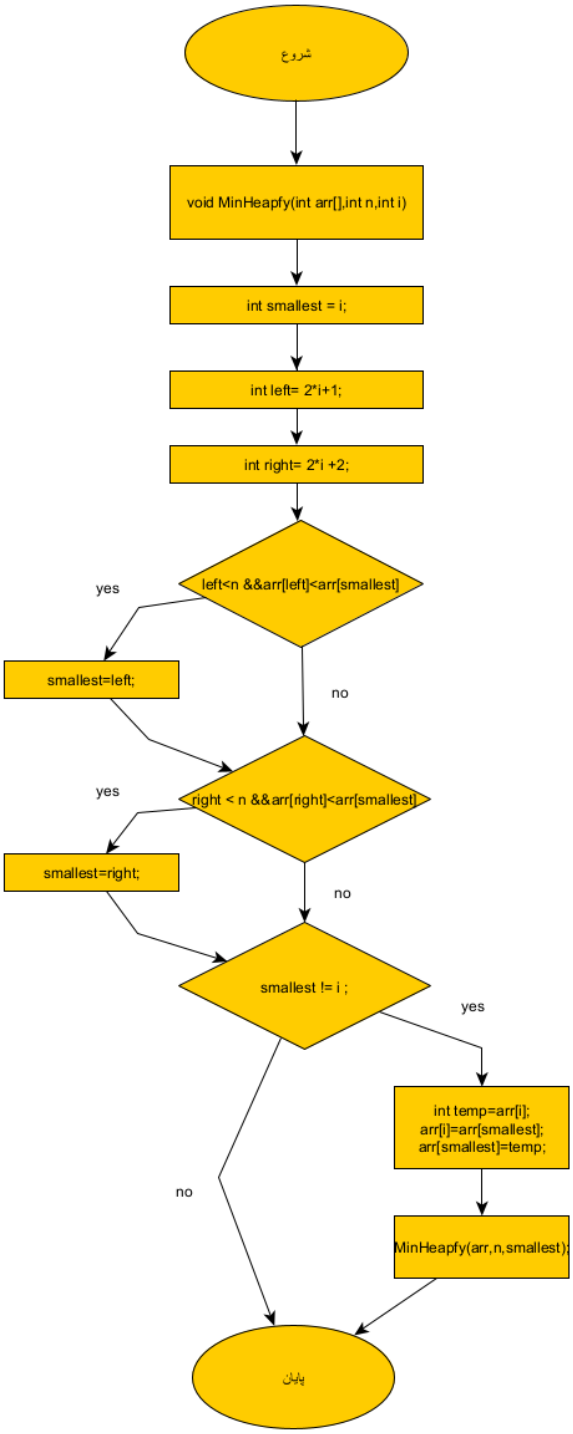


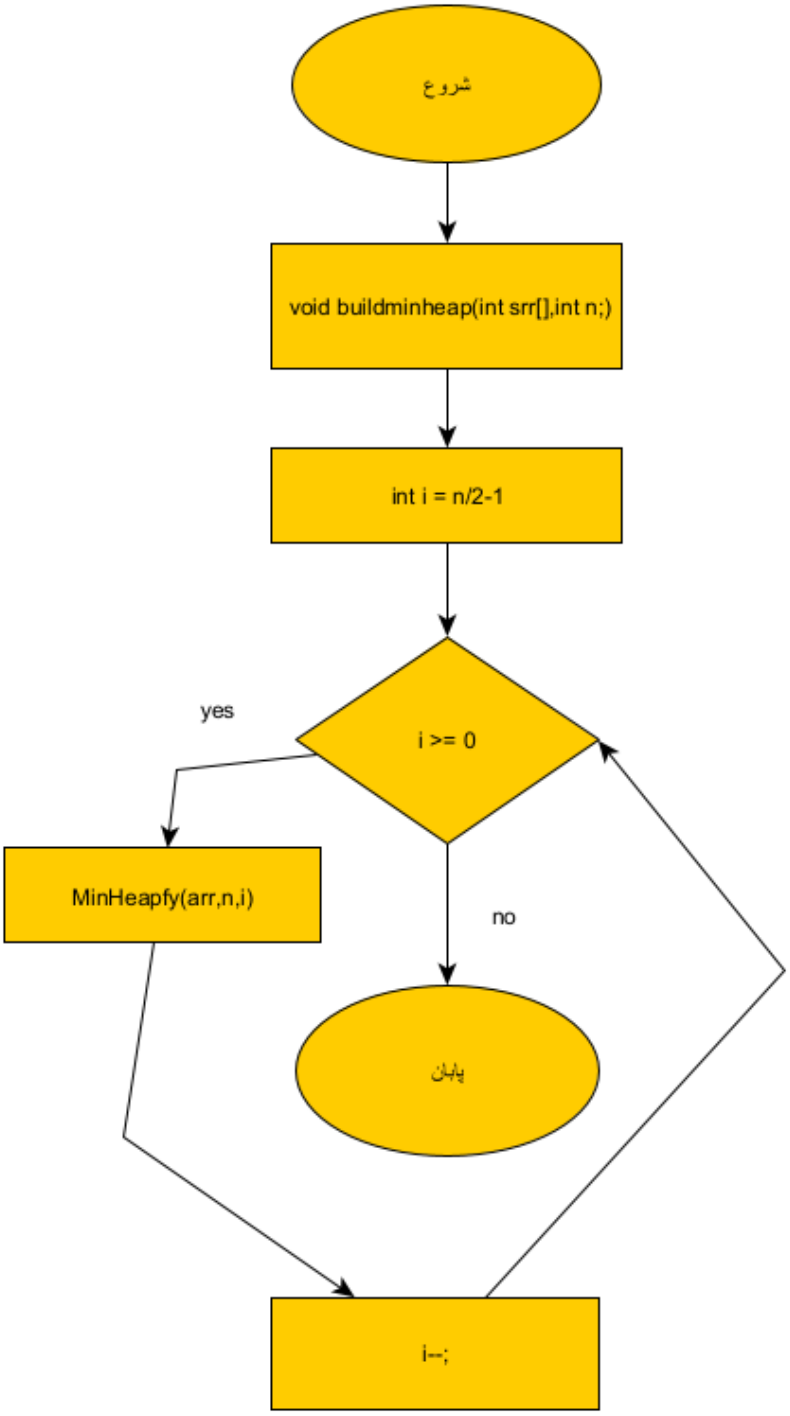
هیپ مینیمم

۵ بررسی الگوریتم و فلوچارت

۱.۵ عملکرد

این الگوریتم با استفاده از تکنیک بازگشتی (Recursion) به صورت زیر عمل می‌کند:
مقادیر کمینه را پیدا کنید: ابتدا، کمینه (min) را در آرایه پیدا می‌کنیم.
یک گره جدید با مقدار min به عنوان ریشه درخت کارتزینی ایجاد می‌کنیم.
زیرآرایه سمت چپ (leftSubarray): از عناصر آرایه اصلی که کوچک‌تر از min هستند، یک زیرآرایه جدید ایجاد می‌کنیم.
زیرآرایه سمت راست (rightSubarray): از عناصر آرایه اصلی که بزرگ‌تر از min هستند، یک زیرآرایه جدید ایجاد می‌کنیم.
به طور بازگشتی: برای هر زیرآرایه (leftSubarray) و (rightSubarray) مراحل ۱ تا ۳ را تکرار می‌کنیم تا زمانی که تمام عناصر آرایه اصلی در درخت کارتزینی قرار بگیرند.
درخت را کامل کنید: در نهایت، گره ریشه (min) را به عنوان ریشه درخت کارتزینی نهایی در نظر می‌گیریم و فرزندان آن را به ترتیب درخت‌های فرعی ساخته شده از leftSubarray و rightSubarray در مراحل قبل، به آن متصل می‌کنیم.


شروع
void MinHeapfy(int arr[],int n,int i)
int smallest = i;
int left= 2*i+1;
int right= 2*i +2;
left<n &&arr[left]<arr[smallest]
yes
smallest=left;
no
right < n &&arr[right]<arr[smallest]
yes
smallest=right;
no
smallest != i ;
yes
int temp=arr[i];
arr[i]=arr[smallest];
arr[smallest]=temp;
MinHeapfy(arr,n,smallest);
no
پایان


شروع
void buildminheap(int srr[],int n;)
int i = n/2-1
i >= 0
yes
MinHeapfy(arr,n,i)
no
پایان
i--;

۲.۵ فلوچارت

۳.۵ کد

```
    import java.util.Scanner;

class CTNode {
    CTNode left, right;
    int value;

    public CTNode() {
        left = null;
        right = null;
        value = 0;
    }
}

class CartesianTree {
    private CTNode root;

    public CartesianTree(int[] data) {
        root = build(data);
    }

    public CTNode build(int[] data) {
        if (data == null || data.length == 0) {
            return null;
        }
        return build(data, 0, data.length - 1);
    }

    private CTNode build(int[] data, int start, int end) {
        if (end < start) {
            return null;
        }
        int min = Integer.MAX_VALUE;
        int minIndex = -1;
        for (int i = start; i <= end; i++) {
            if (data[i] < min) {
                min = data[i];
                minIndex = i;
            }
        }
        CTNode node = new CTNode();
        node.value = min;
        node.left = build(data, start, minIndex - 1);
```

```
    node.right = build(data, minIndex + 1, end);
    return node;
}

public boolean isEmpty() {
    return root == null;
}

public int countNodes() {
    return countNodes(root);
}

private int countNodes(CTNode r) {
    if (r == null) {
        return 0;
    } else {
        int l = 1;
        l += countNodes(r.left);
        l += countNodes(r.right);
        return l;
    }
}

public void inorder() {
    inorder(root);
}

private void inorder(CTNode r) {
    if (r != null) {
        inorder(r.left);
        System.out.print(r.value + " ");
        inorder(r.right);
    }
}

public void preorder() {
    preorder(root);
}

private void preorder(CTNode r) {
    if (r != null) {
        System.out.print(r.value + " ");
        preorder(r.left);
        preorder(r.right);
    }
}
```

```
    public void postorder() {
        postorder(root);
    }

    private void postorder(CTNode r) {
        if (r != null) {
            postorder(r.left);
            postorder(r.right);
            System.out.print(r.value + " ");
        }
    }
}

public class CartesianTreeTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Cartesian Tree Test\n");
        System.out.println("Enter number of integer values");
        int N = scan.nextInt();
        int arr[] = new int[N];
        System.out.println("\nEnter " + N + " integer values");
        for (int i = 0; i < N; i++) {
            arr[i] = scan.nextInt();
        }
        CartesianTree ct = new CartesianTree(arr);
        System.out.println("\nTree Details :");
        System.out.println("Empty status - " + ct.isEmpty());
        System.out.println("No of nodes - " + ct.countNodes());
        System.out.print("Post order : ");
        ct.postorder();
        System.out.print("\nPre order : ");
        ct.preorder();
        System.out.print("\nIn order : ");
        ct.inorder();
        System.out.println();
    }
}
```

۴.۵ کلاس ها

CTNode : این کلاس نشان دهنده یک گره (Node) واحد در درخت کارتزینی است. سه ویژگی دارد:

- left : اشاره به گره فرزند چپ.

- right : اشاره به گره فرزند راست.
- value : مقدار عددی ذخیره شده در گره.

CartesianTree : این کلاس کل درخت کارتزی را نشان می دهد. قابلیت های زیر را دارد:

- سازنده : یک آرایه اعداد صحیح را به عنوان ورودی می گیرد و از آن درخت کارتزی را می سازد.
- build : این تابع به طور بازگشتی درخت کارتزی را از یک آرایه داده شده می سازد. کمترین عنصر آرایه را پیدا می کند و آن را به عنوان ریشه (Root) قرار می دهد. سپس به طور بازگشتی زیر درخت چپ را با عناصر کوچکتر از ریشه و زیر درخت راست را با عناصر بزرگتر از ریشه می سازد.
- isEmpty : بررسی می کند که آیا درخت خالی است (گره ریشه ندارد).
- countNodes : تعداد کل گره های درخت را می شمارد.
- postorder preorder، inorder، : این توابع به ترتیب پیمایش های درون بر (Inorder)، پیش بر (Preorder) و پس بر (Postorder) درخت را انجام می دهند. هر تابع پیمایش یک تابع کمکی با همین نام را برای انجام پیمایش بازگشتی و چاپ مقادیر گره ها فراخوانی می کند.

۵.۵ کلاس اصلی

- از کاربر می خواهد تعداد مقادیر عددی را که می خواهد در درخت ذخیره کند وارد کند.
- مقادیر صحیح را از کاربر می خواند و آنها را در یک آرایه ذخیره می کند.
- با ارسال آرایه به سازنده، یک شیء CartesianTree ایجاد می کند.
- بررسی می کند که آیا درخت خالی است و نتیجه را چاپ می کند.
- تعداد گره های درخت را می شمارد و تعداد را چاپ می کند.
- یک پیمایش پس بر از درخت انجام می دهد و مقادیر گره های بازدید شده را چاپ می کند.
- یک پیمایش پیش بر از درخت انجام می دهد و مقادیر گره های بازدید شده را چاپ می کند.
- یک پیمایش درون بر از درخت انجام می دهد و مقادیر گره های بازدید شده را چاپ می کند.

۶ کاربردها

۱.۶ ساختارهای داده

درخت‌های جستجوی بهینه: درخت‌های کارتزینی می‌توانند به عنوان ساختار داده‌ای کارآمد برای جستجوی عناصر در یک مجموعه مرتب شده عمل کنند. زمان جستجو در این درختان در بدترین حالت (log n) است که با درخت‌های جستجوی دودویی متعادل مانند درخت‌های AVL یا درخت‌های سرخ همسان است.

۲.۶ ماشین لرنینگ

انتخاب ویژگی: درخت‌های کارتزینی می‌توانند برای انتخاب ویژگی‌ها در الگوریتم‌های یادگیری ماشین استفاده شوند. این کار با شناسایی ویژگی‌هایی که اطلاعات بیشتری در مورد هدف ارائه می‌دهند و حذف ویژگی‌های تکراری یا بی‌فایده انجام می‌شود.
طبقه‌بندی: درخت‌های کارتزینی می‌توانند برای طبقه‌بندی داده‌ها به دو دسته یا بیشتر استفاده شوند. این کار با تقسیم داده‌ها بر اساس ویژگی‌های انتخاب شده و تکرار فرآیند در زیرمجموعه‌ها انجام می‌شود.
رگرسیون: درخت‌های کارتزینی می‌توانند برای پیش‌بینی مقادیر پیوسته (رگرسیون) استفاده شوند. این کار با ساختن یک مدل رگرسیونی در هر زیرشاخه از درخت و ترکیب پیش‌بینی‌ها در سطوح بالاتر انجام می‌شود.

۳.۶ پردازش زبان طبیعی

تجزیه و تحلیل نحوی: درخت‌های کارتزینی می‌توانند برای تجزیه و تحلیل ساختار نحوی جملات زبان طبیعی استفاده شوند. این کار با شناسایی وابستگی‌های بین کلمات و ساختن یک درخت نحوی از جمله انجام می‌شود.
استخراج اطلاعات: درخت‌های کارتزینی می‌توانند برای استخراج اطلاعات از متن، مانند نام افراد، مکان‌ها یا تاریخ‌ها استفاده شوند. این کار با شناسایی الگوهای خاص در متن و مرتبط کردن آنها با اطلاعات مربوطه انجام می‌شود.

۴.۶ گرافیک کامپیوتری

مدل‌سازی سه‌بعدی: درخت‌های کارتزینی می‌توانند برای مدل‌سازی سه‌بعدی اشیاء و صحنه‌ها استفاده شوند. این کار با ساختن یک سلسله مراتب از گره‌ها انجام می‌شود که هر کدام نشان‌دهنده بخشی از مدل هستند.
رندرینگ: درخت‌های کارتزینی می‌توانند برای رندرینگ گرافیک سه‌بعدی استفاده شوند. این کار با پیمایش درخت و رسم هر گره و فرزندان آن انجام می‌شود.

۷ نتیجه گیری

در مجموع، درخت‌های کارتزینی به دلیل ساختار ساده و کارآمد خود، ابزاری قدرتمند برای حل طیف وسیعی از مسائل در علوم کامپیوتر، به ویژه یادگیری ماشین، هستند. آنها به طور فزاینده‌ای در تحقیقات و کاربردهای مختلف مورد استفاده قرار می‌گیرند.

۸ منابع

- GeeksforGeeks - Cartesian Tree
- Bender, Michael A.; Farach-Colton, Martin (2000), "The LCA problem revisited",
- Berkman, Omer; Schieber, Baruch; Vishkin, Uzi (1993), "Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values",

- ishimoto, Akio; Fujisato, Noriki; Nakashima, Yuto; Inenaga, Shunsuke (2021), "Position heaps for Cartesian-tree matching on strings and tries", in Lecroq,
- Levcopoulos, Christos; Petersson, Ola (1989), "Heapsort - Adapted for Presorted Files"