گزارش تمرین سوم برنامه نویسی پیشرفته

علیرضا طباطبائیان 9723052

*دستورات دارای کامنت میباشند.

در ابتدا تمامی **کانستراکتور** ها را تعریف کردیم.

برای مقدار node و پوینتر های راست و چپ تک تک کپی شوند.

سپس **دیستراکتور** را مطابق تعریف سوال آوردیم.

برای **کپی کانستراکتور** : از bfs استفاده نمودیم که بر روی تک تک نود ها حرکت میکند و تک تک آنها را به درخت اضافه میکند.

سپس اپراتور >> را برای نمایش نود و اطلاعات موجود در آن تعریف کردیم : برای نمایش زیباتر از setw با اعداد مناسب استفاده نمودیم.

سپس **اپراتور مساوی را برای کپی کردن** تعریف کردیم : که با استفاده از bfs بر روی تک تک نود ها حرکت کردیم و مقادیر آنها را با add_node بر روی شبکه جدید کپی کردیم.

سپس **اپراتور مساوی را برای move کردن** تعریف کردیم : میبایست پوینتر ورودی را در آبجکت جدید بریزیم و سپس آبجکت قبلی را پاک کنیم و آبجکت جدید را به خروجی ببریم.

در قسمت بعد ، **عملگر بشقاب پرنده و مساوی** را تعریف نمودیم که برای جلوگیری از تکرار زیاد انواع عملگر های مقایسه ای ، میتوان این دو عملگر را تعریف کرد و برای سایر عملگر ها از ترکیب همین دو استفاده کرد.

سپس **bfs** را تعریف میکنیم : به واسطه یک بردار و گرفتن بچه های آن و ذخیره کردن آن در برداری جدید ، میتوان بر روی تمام نود ها حرکت کرد. بردار را با استفاده از push back و erase تغییر میدادیم.

سپس **تابع length** را تعریف کردیم : که تعداد نود ها را با استفاده از bfs میشمارد. در تابع مربوطه کافیست این شمارش را بواسطه یک متغیر صحیح بشماریم.

سپس **add node** را تعریف کردیم : جهت این کار باید هر مرحله مطمئن شویم که نود مورد نظر از قبل موجود است یا خیر. یا اینکه آیا بچه ای به آن متصل است یا خیر. سپس با مقایسه مقادیر ، نود مربوطه را اضافه میکنیم.

سپس **اپراتور >> را برای BST** تعریف نمودیم : از تعریف قبلی اپراتور >> استفاده کرده و با ترکیب آن با bfs ، میتوانیم کل درخت را نمایش دهیم.

تعریف **find node** : در اینجا مجاز به استفاده از bfs نبودیم و باید با استفاده از ویژگی های درخت که همان نابرابری است ، نود خاصی را پیدا میکردیم و پوینتری به پوینتر آن را بر میگرداندیم. در اینجا نیز میبایستی شرایط nullptr ها را رعایت میکردیم تا به مشکل نخوریم.

در بخش **find parent** : ابتدا میبایست پرنت هر نودی که در حال بررسی آن میبودیم را در آبجکتی جدا ذخیره میکردیم و اگر مقدار مربوطه با مقدار مورد جستجو برابر بود ، آبجکتی که مربوط به یک مرحله قبل تر بود را برمیگرداندیم. همواره شرایط nullptr ها را باید رعایت نماییم.

در **find successor** : باید شرایط مختلفی مثل نبود نود مربوطه یا نداشتن بچه چپ یا نداشتن بچه راست برای بچه چپ را رعایت میکردیم و سپس در صورت داشتن بچه چپ ، وارد یک حلقه میشویم تا به آخرین بچه راست از آن بچه چپ برسیم و آن را بصورت پوینتری به یک پوینتری دیگر بازگردانیم.

در **delete node** : باید سه حالت ممکن را رعایت کنیم :

1 - در حال حذف یک برگ باشیم : کافیست بچه ی والد مربوطه را nullptr نماییم.

2- در حال حذف نودی با یک بچه باشیم : باید والد را به فرزند وصل کنیم.

3- در حال حذف نودی با یک بچه باشیم که successor آن خودش یک برگ است : ابتدا باید successor را به دو فرزند نودی که در حال حذف هستیم وصل کنیم و سپس successor را به والد نودی که در حال حذف هستیم وصل کنیم و سپس ارتباط successor را با والدش قطع کنیم. ترتیب این مراحل مهم میباشد. همچنین اگر نودی که در حال حذف بودیم ، نود root باشد ، دیگر نیاز نیست successor را به والد خاصی مرتبط نماییم که همه این موارد در کد لحاظ شده اند.

همچنین **اپراتور های ++** را برای دو حالت ممکن آن نوشتیم که در هر دو از bfs استفاده نمودیم. برای آن که ++ در سمت راست آبجکت است ، نیاز بود که آبجکت را در متغیر temp ذخیره کنیم و این نسخه بدون تغییر یعنی temp را برگردانیم.

در آخر برای **initialize** کردن : یک لیست را دریافت کرده و بر روی اعضای آن ، تابع add node را پیاده میکنیم.

لینک Git