گزارش کار

HW3

دانش عبداللهی

9723053

ابتدا در فایل **bst.h** طبق خواسته‌های سوال، کلاس **BST** و در داخل کلاس **BST** به صورت **Public** ، کلاس **Node** را تعریف کردیم.

ابتدا **Constructor** های کلاس **Node** را نوشتیم. اولین**Constructor** مقدار متغیر **Node** و آدرس بچهٔ راست و چپش را می‌گیرید و با استفاده از این ورودی‌ها ، مقادیر متغیرهای کلاس **Node** (**value** ، **left** و **right**) را مقدار دهی می‌کند. اگر آدرس بچه‌های **Node** را ندهیم ، به صورت پیشفرض ، متغیرهای **left** و **right** را برابر **Nullptr** قرار می‌دهیم.

- **Default Constructor** را طوری تعریف می‌کنیم که به صورت پیشفرض مقدار **value** را صفر و متغیرهای **right** و **left** را برابر **Nullptr** قرار دهد.
- برای نوشتن **Copy Cunstructor** هم ، ورودی را از کلاس **Node** تعریف کردیم و مقادیر متغیرهای ورودی را در متغرهای **Object** جدید ریختیم.
- برای پیاده‌سازی عملگر >> ، آنرا به صورت **friend** تعریف کردیم و خروجی را به صورت زیر ساختیم:

```
**********************************************************************************
PRINT A NODE
0x7ffd32b73710     => value:10      left:0x7ffd32b73750     right:0x7ffd32b73730
**********************************************************************************
```

- در تصویر بالا از چپ به راست : آدرس **Node** ، مقدار **value** آن ، آدرس بچهٔ چپ و در آخر آدرس بچهٔ راست می‌باشد.
- برای پیاده‌سازی عملگرهای مقایسه‌ای ، ابتدا کتاب‌خانهٔ **compare** را **include** کردیم و از عملگر سه طرفه (⇔) استفاده کردیم. ورودی آنرا از جنس متغیر **Node** قرار دادیم که اگر جایی خواستیم عبارت دو تا متغیر **Node** را با هم مقایسه کنیم ، با مشکلی روبه رو نشویم. **Compiler** در صورت نیاز می‌تواند **int** را به متغیر **Node** تبدیل کند ، اما برعکسش ممکن نیست. عملگر== را با ورودی از جنس **Node** به صورت جداگانه تعریف کردیم.
- در این مرحله تمام عملگرها و **Constructor**های مورد نیاز را پیاده سازی کردیم.
- **Default Constructor** کلاس **BST** را گونه‌ای پیاده‌سازی کردیم که متغیر **root** را برابر **Nullptr** قرار بدهد.
- در **Destructor** ، یک **Vector** از جنس ***Node** درست کردیم و با استفاده از تابع **bfs** ، آدرس تمام **Node** های درخت رو در **Vector** ریختیم و سپس با زدن یک **for** روی این **Vector** تمام متغیرهای آنرا با دستور **delete** پاک کردیم.
- **Copy Constructor** را با استفاده از تابع **bfs** و **add_node** ساختیم. به این صورت که یک درخت جدید با استفاده تمام مقادیر **Node**های درخت ورودی (به ترتیب از **root** به پایین) می‌سازیم.
- در **Move Construvtor** هم **root** ورودی را در **root** درخت جدید می‌ریزیم و سپس متغیر **root** درخت ورودی را برابر **Nullptr** قرار می‌دهیم.

- یک **Constructor** هم با ورودی **initializer_list<int>** تعریف کردیم (برای حل قسمت **challenge** سوال) که با یک حلقه روی متغیرهای لیست ورودی حرکت می‌کند و با استفاده از تابع **()add_node** درخت جدید را با مقادیر موجود در لیست ورودی می‌سازیم. (کتاب-خانهٔ **<initializer_list >** را باید **include** می‌کردیم.)
- تابع **()get_root** : فقط متغیر **root** را برمی‌گردانیم.
- تابع **()bfs** : در این تابع ، ابتدا یک **vector** از جنس ***Node** می‌سازیم. سپس **root** را در **Vector Tree** می‌ریزیم. سپس وارد یک حلقه می‌شویم و تمام **Node**های درخت را تک تک داخل **Tree** میریزیم و تابع را روی آن اجرا می‌کنیم. باید توجه کنیم که ما برای اضافه کردن **Node**های درخت به **Vector Tree** ، از **root** به سمت پایین حرکت می‌کنیم پس باید تابع ورودی را به ترتیب از ریشه به پایین روی **Node**ها اجرا کنیم. برای اجرای همین مورد , در هر تکرار حلقه ، تابع را روی اولین المان **Tree** اجرا می‌کنیم و سپس آنرا از **Vector** پاک می-کنیم. شرط **(node!= nullptr) if** را باید بگذاریم تا به ارور **Segmentation fault** مواجه نشویم.حلقه هم تا زمانی ادامه دارد که **vector Tree** خالی نباشد.
- تابع **()add_node** : در این تابع ابتدا بررسی می‌کنیم که متغیر **root** ، **Nullptr** هست یا نه و اگر بود ، یک **Node** جدید می‌سازیم و در **root** می‌ریزیم. در غیر این صورت با مقایسهٔ مقدار ورودی با هر**Node** وارد بچهٔ چپ یا راستش می‌شویم و اگر هر کدام از آنها که وارشان شدیم ، مقداری نداشتند ، **Node** جدید را می‌سازیم و داخل آن می‌ریزیم. در هر **Node** هم می‌رویم بررسی می‌کنیم که مقدار ورودی تابع با مقدار آن **Node** برابر است یا خیر که در صورت برابری **false** را به عنوان خروجی تابع برمی‌گردانیم.

- تابع **length()** : با استفاده از تابع **bfs()** ، به ازای تمام **Node**های درخت ، متغیر **size_t length()** را به علاوهٔ **1** می‌کنیم.
- تابع **find_node()** : با استفاده از حلقه و مقایسهٔ مقدار ورودی ، وارد **Node** های جدید می‌شویم و مقادیر آنها را با مقدار ورودی مقایسه می‌کنیم و اگر مساوی بودند ، آدرس اشاره‌گر به آن **Node** را به عنوان خروجی برمی‌گردانیم.
- تابع **find_parrent()** : مثل تابع **find_node()** عمل می‌کنیم فقط در آخر به جای برگرداندن آدرس اشره‌گر به خود **Node**، آدرس اشاره‌گر به **Node** بالائی اش را برمی‌گردانیم.
- تابع **find_successor ()** : در این تابع **3** حالت را بررسی می‌کنیم ، اگر **Node** ئی که می‌خواهیم **successor** اش را پیدا کنیم شاخهٔ چپش خالی بود ، خودش را بر می‌گردانیم. اگر شاخهٔ چپ داشته باشد اما ساخهٔ چپ آن ، **Node** راست نداشته باشد همان شاخهٔ چپ را برمی‌گرداند و در حالت کلی هم که الگریتم را کامل پیاده می‌کند.
- تابع **delete_node()** : اول از همه با استفاده از توابع **find_node()** و **find_parrent()** آدرس خود **Node** و **Parrent** اش را پیدا می‌کنیم. سپس به ترتیب چک می‌کنیم که **Node** ئی که می‌خواهیم آنرا پاک کنیم ، برگ است یا فقط شاخهٔ چپ دارد یا فقط شاخهٔ راست دارد یا اینکه دو تا شاخهٔ چپ و راست را دارد و برای هر حالت الگوریتم را طبق شرط پیاده می‌کنیم. در همهٔ حالات اتصالات درخت به آن **Node** ئی که می‌خواهیم پاک را پاک می‌کنیم (**Nullptr** قرار می‌دهیم.) و در صورت نیاز اتصالات جدید را بسته به حالت آن **Node** تعریف می‌کنیم.

- **عملگر = (Copy Version) : ابتدا بررسی می‌کنیم که اگر آدرس Object های دو طرف = یکسان باشند ، همان Object را به عنوان خروجی برگرداند. در ادامه با استفاده از کد Copy Constructor متغیر جدید را می‌سازیم. فقط قبل از آن متغیر root آنرا پاک می‌کنیم که اگر چیزی از قبل در آن بود پاک شود.**

  در این جا از خود Copy Constructor استفاده نکردیم تا تعداد کپی کردن ها کمتر باشد.
- **عملگر = (Move Version) : ابتدا متغیر root را پاک می‌کنیم و سپس root ورودی را در آن می‌ریزیم و بعد از آن root ورودی را برابر Nullptr قرار می‌دهیم.**
- **عملگر ++ چپ : با استفاده از تابع ()bfs روی تمام Node های درخت ورودی حرکت می-کنیم و مقادیرشان را به اضافهٔ 1 می‌کنیم و همان درخت را برمی‌گردانیم.**
- **عملگر ++ راست : ابتدا درخت ورودی را در یک متغیر Temp کپی می‌کنیم و سپس با استفاده از تابع ()bfs مقادیر تمام Node های درخت ورودی را به اضافهٔ 1 می‌کنیم و در آخر هم متغیر Temp را به عنوان خروجی برمی‌گردانیم.**
- **عملگر << : با استفاده از عملگر << که برای کلاس Node نوشته بودیم و تابع ()bfs تمام Nodeهای درخت را در متغیر std::ostream& _output می‌ریزیم و در آخر آنرا بر می-گردانیم.**

خروجی به شکل زیر است :

```
********************************************************************************
0x1f45e10       => value:25     left:0x1f45e30      right:0x1f45d30
0x1f45e30       => value:10     left:0x1f45db0      right:0x1f45df0
0x1f45d30       => value:50     left:0              right:0x1f45d50
0x1f45db0       => value:7      left:0              right:0
0x1f45df0       => value:15     left:0              right:0
0x1f45d50       => value:53     left:0              right:0
binary search tree size: 6
********************************************************************************
```

و در اخر هم تمام تست‌ها به درستی انجام شدند :

```
[ RUN      ] HW3Test.TEST17
[       OK ] HW3Test.TEST17 (0 ms)
[ RUN      ] HW3Test.TEST18
[       OK ] HW3Test.TEST18 (0 ms)
[ RUN      ] HW3Test.TEST19
[       OK ] HW3Test.TEST19 (0 ms)
[ RUN      ] HW3Test.TEST20
[       OK ] HW3Test.TEST20 (0 ms)
[ RUN      ] HW3Test.TEST21
[       OK ] HW3Test.TEST21 (0 ms)
[ RUN      ] HW3Test.TEST22
[       OK ] HW3Test.TEST22 (0 ms)
[ RUN      ] HW3Test.TEST23
[       OK ] HW3Test.TEST23 (0 ms)
[ RUN      ] HW3Test.TEST24
[       OK ] HW3Test.TEST24 (0 ms)
[ RUN      ] HW3Test.TEST25
[       OK ] HW3Test.TEST25 (0 ms)
[ RUN      ] HW3Test.TEST26
[       OK ] HW3Test.TEST26 (0 ms)
[ RUN      ] HW3Test.TEST27
[       OK ] HW3Test.TEST27 (0 ms)
[ RUN      ] HW3Test.TEST28
[       OK ] HW3Test.TEST28 (0 ms)
[ RUN      ] HW3Test.TEST29
[       OK ] HW3Test.TEST29 (0 ms)
[ RUN      ] HW3Test.TEST30
[       OK ] HW3Test.TEST30 (0 ms)
[ RUN      ] HW3Test.TEST31
[       OK ] HW3Test.TEST31 (0 ms)
[----------] 31 tests from HW3Test (1 ms total)

[----------] Global test environment tear-down
[==========] 31 tests from 1 test suite ran. (1 ms total)
[  PASSED  ] 31 tests.
<<<SUCCESS>>>
```

لینک **github**