به نام خدا

شرح پروژه ی اول درس ساختمان داده ها

فاز اول

**کاری از :**

**ملیکا عبداللهی / یاسمن میرمحمد**

سلام! در این پروژه از ما خواسته شده که الگوریتم جست و جو را بااستفاده از ساختمان داده های مناسب پیاده سازی کنیم .

این الگوریتم یک درخت جست و جوی چند کاره دارد که کلمات داخل فایل ها ی داده شده در آن قرار میگیرند . هر عنصر درخت به یک لیست فایل اشاره میکند { برای قسمت امتیازی نیاز است که تعداد و محل دقیق وقوع کلمات نیز ذخیره شود }

در کلیه ی مراحل استاپ ورد ها{که در فایل ضمیمه آمده اند } باید نادیده گرفته شوند .

بخش اول پروژه:

**پیاده سازی ساختمان داده های اصلی**

برای این کار ما در ابتدا به سه ساختمان داده ی اصلی نیاز داریم:

1)BST

2) TST

3) TRIE

4) HASH

قسمت چهارم مربوط به فاز دوم پروژه است.

1. BST :

به دو صورت میتوان این درخت را پیاده سازی نمود :

AVL

RED BLACK

که از بین این دو روش اولی روش مناسبتر و بهتری است و پیچیدگی کمتر و فهم بهتر و پیاده سازی ساده تری دارد .

برای پیاده سازی به این روش ، چندین کار لازم داشتیم .

یکی از آنها چرخش درخت بود .

چرخش چپ به راست و چرخش راست به چپ که برای متوازن سازی درخت لازم است .

( در فازاول متوازن سازی لازم نبود ولی برای پیاده سازی بهتر و انسجام فکری بهتر دیدیم که از اول طوری به مساله نگاه کنیم که گویا قرار است همه چیز متوازن باشد .

Avl:

برای پیاده سازی توابع در این بخش ، از چندین تعریف استفاده کردیم :

Class Avl:

* 1)Height
* 2)Diff:
* اختلاف ارتفاع بین د وزیر شاخه را مید هد . اگر این اختلاف 1 نباشد پس درخت ما به بالانس نیاز دارد .
* 3)rr rotation: Right to right
* 4)ll rotation: left to left
* 5)lr rotation : left to right
* 6)rl rotation’: right to left
* 7)balance :
* از 3 و 4 و 5و6 برای پیاده سازی اش استفاده شده است .
* 8)insert(\* root ):
* 10)in order
* 11)remove file ( \* nodes ):
* این الگوریتم برای بقیه ی ساختمان داده ها هم روش مشابهی دارد . بدین صورت که فایل را با استفاده از تابع فایند که قبلا تعریف کرده بودیم پیدا میکند و به ترتیب انرا در زیر درخت های چپ و راست جست و جو میکند و وقتی آنرا یافت حذفش میکند .

همچنین برای راحتی جست و جو ، لیستی از فایل ها در استراکت مان ساختیم و تابع "فایند" را داخل آن تعریف کردیم .

بخش بعدی پیاده سازی درخت جست و جوی دودویی کلید گذاری و حذف کلید هاست .

2) TRIE:

1. Insert
2. Erase
3. Remove file
4. In Order:

از ریشه شروع میکنیم و به اندازه ی سایز عبارت جست و جو شده جلو میرویم ومرتب سازی انجام میدهیم.برخلاف بقیه ی ساختمان داده ها، حلقه را به تعداد حروف الفبا میزنیم .

1. Find

3) TST:

1. Insert
2. Erase
3. Remove file
4. In Order
5. Find

4) Stack:

1. Push:

StackNode \*p = new StackNode( d );

p->next = root; //khoone badi ro rishe bgir

root = p;

sz ++;

1. Pop:

root = root->next; //boro badi

sz --; //size ro kam kon chon doone doone dari pop mikoni

1. Top:

Data را برمیگرداند .

1. Size:

سایز پشته

5) Base :

تعاریف اصلی که نیاز داریم در این قسمت تعریف شده اند .

6) list:

7)main :

قسمت اجرای نهایی برنامه روی آن قراردارد .

چالش : برای متوازن شدن این درخت راهی پیدا نشد .

Phase 2 …

بخش دوم پروژه :

**Driver**

آن است . در این قسمت باید کلیه ی قسمت های خواسته شده توسط کاربر پاسخ داده شود .

به طور کلی :

1. افزودن فایل
2. حذف فایل
3. به روز رسانی اطلاعات فایل
4. لیست گرفتن از فایل هایی که برایشان درخت ساخته شده 🡺 list\_1
5. لیست تمام فایل های موجود در پوشه ی فعلی
6. جست و جوی عبارت:

در این بخش استاپ ورد ها نادیده گرفته می شوند.

1. جست و جوی کلمات
2. Arrow key up

:در این بخش باید از یک پشته استفاده کنیم، چرا که به وضوح گفته شده که تاریخچه ی جست و جو باید هر (آنچه که آخرین بار وارد شده را اولین بار نمایش دهد lifo)

1. Init 🡺 reading from stop word
2. Is stop word:

به ترتیب چک میکند که آیا از کلمات استاپ وردی که از فایل خوانده ، کلمه ی مشترکی در فایل وجود دارد یا خیر .

* return StopWords.find( str ) != StopWords.end();

1. Read and insert
2. Read and remove
3. Drive :

* Add
* Delete
* Update
* List

1. Words
2. Files
3. Directories

6. Search

1. Word
2. Phrase

ارتباط با gui:

برای راحتی کار ما در داریور به ترتیب به حالت هایی که کاربر روی رادیو باتن ها انتخاب میکند شماره نسبت دادیم، به ترتیب اگر بی اس تی انتخاب کند :1

اگر تی اس تی انتخاب کند :2

اگر ترای انتخاب کند :3

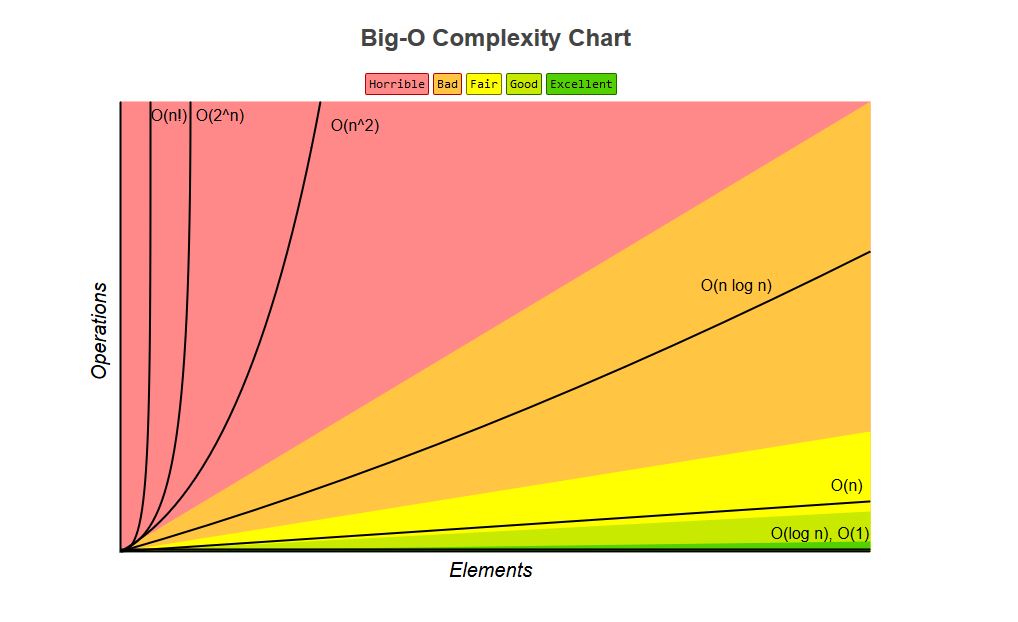
و در مورد هش این مقدار 4 خواهد بود .

این گونه تقسیم بندی و پیمایش حالتها با نظم بیشتری صورت میگیرد و با ترتیب بهتری جلو خواهیم رفت .

***مقایسه ی ساختمان داده های مورد استفاده :***

یکی از بخش های اصلی پروژه ، مقایسه و بررسی پیچیدگی زمانی ساختمان داده های مختلف استقاده شده است .

تئوری :

الگوریتم ها همگی حالت متوسط درج و حذف و آپدیت در ساختمان های داده می باشند)

# BST: O (log N)

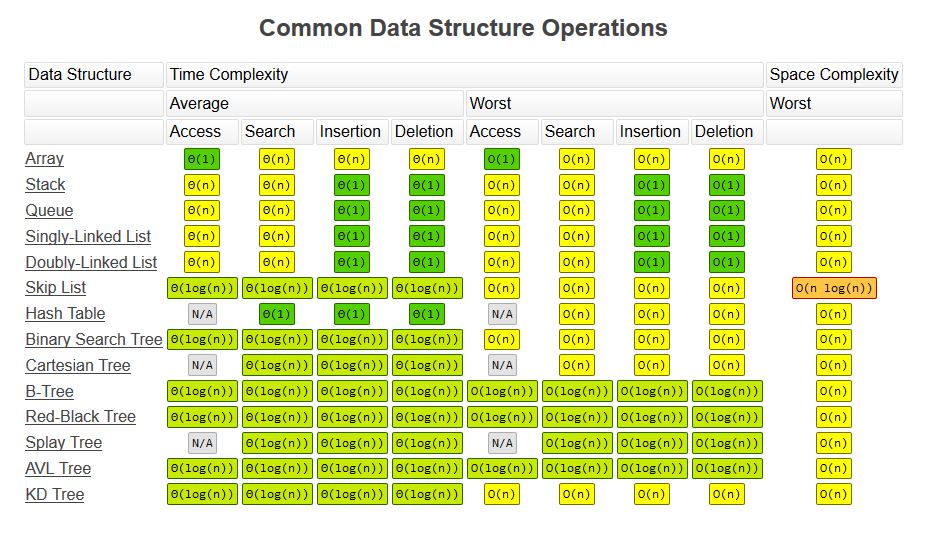
# 2) TST: O (height of tree)

# 3) TRIE: M\* log N (M: String length, N: Number of keys)

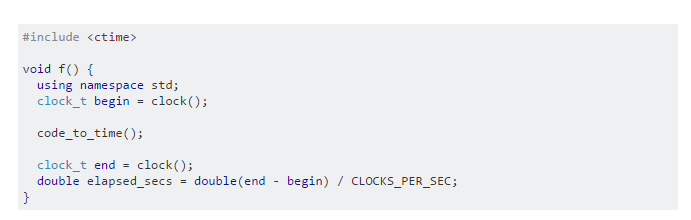
# 4) HASH: O(1) [ O(N) in worst Case]

# 5)Stack: O(1) [ O(N) in worst Case]

**خلاصه ای از مقایسه ی تمام ساختمان داده ها :**



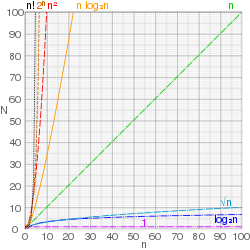
تحلیل زمانی الگوریتم های برنامه :



زمان پاسخ به سرچ یا کوئری

مقایسه عملی :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data Structure | Average case | | | Worst case | | |
|  | Insert | Delete | Search | Insert | Delete | Search |
| Array  Stack  Queue | O(1) | O(1) | O(n) | O(1) | O(1) | O(n) |
| Linked list | O(1) | O(1) | O(n) | O(1) | O(1) | O(n) |
| Hash table | O(1) | O(1) | O(1) | O(n) | O(n) | O(n) |
| Binary search tree | O(log(n)) | O(log(n)) | O(log(n)) | O(n) | O(n) | O(n) |
| TST | O(height) | O(height) | O(height) | O(height) | O(height) | O(height) |
| TRIE | O(m\*log(n)) | O(m\*log(n)) | O(m\*log(n)) | O(m\*log(n)) | O(m\*log(n)) | O(m\*log(n)) |



**توضیحات فاز دوم پروژه:**

**(تابع درهم سازی و توازن)**

1)تابع درهم سازی

بخش پایانی :

**رابط کاربری**



برای راحتی کار ما در داریور به ترتیب به حالت هایی که کاربر روی رادیو باتن ها انتخاب میکند شماره نسبت دادیم، به ترتیب اگر بی اس تی انتخاب کند :1

اگر تی اس تی انتخاب کند :2

اگر ترای انتخاب کند :3

و در مورد هش این مقدار 4 خواهد بود .

این گونه تقسیم بندی و پیمایش حالتها با نظم بیشتری صورت میگیرد و با ترتیب بهتری جلو خواهیم رفت .

BUILD:

کلیه ی مراحل برنامه دیکشنری با توجه به شرایط گفته شده و اطلاعاتی که کاربر وارد کرده اجرا میشوند.

RESET:

کل هیستوری پاک شده و از اول برنامه اجرا میشود .

CANCEL:\_

EXIT:­­\_

**نمودار ها و مقایسه ها**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **COMMAND** | **BUILD** | **SEARCH** |  |  |
| BST | 3.85E+06 | 3.70094e+0.6 | 127805000 |  |  |
| TST | 2.26E+08 | 1.25E+08 | 2.32E+08 |  |  |
| TRIE | 3.5 | 1.84E+08 | 1.16E+08 |  |  |
| HASH | 1.15E+08 | 1.16E+08 | 5.17E+65 |  |  |
|  |  |  |  |  |  |

# **چالش های پروژه :**

این که در فاز اول توازن لازم نبود کمی کار را راحت تر کرد و بنابراین ما اول رفتیم سراغ سرچ کردن در مورد ساختمان داده هایی که نیاز داشتیم .

به جز درخت جست و جوی دودویی ما درباره ی هیچ کدام از دوتا ساختمان داده ی دیگه اطلاع زیادی نداشتیم و بنابراین وقت زیادی روی مطالعه در مورد آن ها گذاشتیم. بعد از این کار ، به جای اینکه سراغ بدنه ی اصلی پروژه برویم، تصمیم گرفتیم روی رابط کاربری و گزارش یکی دو روز تمرکز کنیم تا هم فشار کار تعدیل شود و هم اینکه جذابیت کار بالاتر برود .

پروزه با این که طراحی رابط کاربری در مرحله ی اولیه هیچ ارتباطی با فایل ها و ساختمان داده ها ندارد و ظاهرا کمکی به پیشبرد پروژه نمیکند ، اما ا استخوان بندی پروژه را محکم تر میکند و دید بهتری به برنامه نویس میدهد

. شاید مهمترین ویژگی این کار این باشد که شما وقتی به عنوان برنامه نویس پای کامپیوتر مینشینید و یک سری الگوریتم را به وسیله ی کد پیاده سازی میکنید دید شما به عنوان برنامه نویس صرفا ران شدن و خوش ساخت و مرتب بودن کدتان است؛

. اما وقتی شما شروع میکنید به طراحی و امتحان کردن یک رابط کاربری ، در واقع شما زاویه دید متفاوتی را انتخاب کردید و دارید از جایگاه یه کاربر که میخواهد با این برنامه کار کند به قضیه نگاه میکنید .

بعد از رابط کاربری ما رفتیم سراغ انجام بخش پرکار پروژه یعنی درایور .{ این را هم بگویم که برای بخشی که باید فایل آپدیت میشد توضیح پروژه کمی گنگ و نامفهوم بود و ما گیج شده بودیم که محتوای فایل آیا قرار است تغییر کند یا خیر و..

بالاخره بعد از پرسش و پاسخ های فراوان و مشورت با بقیه و سوال از تدریس یاران این مشکل هم برطرف شد و ما متوجه شدیم که این کار فقط قراراست حین تحویل حضوری انجام بشود و ما قرار نیست محتوای دیکشنری مان را حین کار تغییر بدهیم .

در مورد امتخاب زبان هم ، ما تصمیم گرفتیم از زبان سی پلاس پلاس استفاده کنیم زیرا کدها هم سر کلاس به این زبان تدریس میشدند . مراجعه به کتاب هوروویتز هم کمک خوبی به ما کرد .

چالش های فاز 2 :

***توازن :***