دانشکده مهندسی کامپیوتر

جزوه درس

ساختمان‌های داده

استاد درس: سید صالح اعتمادی

پاییز ۱۳۹۸

جلسه ۱۵

صف و پشته

محمد صدرا خاموشی فر - ۱۳۹۸/۷/۱۸

جزوه جلسه ۱۵ام مورخ ۱۳۹۸/۷/۱۸درس ساختمان‌های داده تهیه شده توسط محمد صدرا خاموشی فر. در جهت مستند کردن مطالب درس ساختمان های داده، بر آن شدیم که از دانشجویان جهت مکتوب کردن مطالب کمک بگیریم. هر دانشجو می‌تواند برای مکتوب کردن یک جلسه داوطلب شده و با توجه به کیفیت جزوه از لحاظ کامل بودن مطالب، کیفیت نوشتار و استفاده از اشکال و منابع کمک آموزشی، حداکثر یک نمره مثبت از بیست نمره دریافت کند. خواهش‌مند است نام و نام خانوادگی خود، عنوان درس، شماره و تاریخ جلسه در ابتدای این فایل را با دقت پر کنید.

اگه خواستید پیاده سازی .NET. روبرای STACK ببینید به سایت referencesource.microsoft.com[] مراجعه کنید

system.collections.genereic.stack را سرچ کنید.

- stack
- queue

تعریف پشته: نوع داده ای Abstract که عملگر های push ، key key،top راpop دارد.

تعریف: Abstract نوع داده (ADT) نوعی (یا کلاس) برای اشیاء است که رفتار آنها توسط مجموعه ای از ارزش و مجموعه عملیات تعریف می شود.

```
Push (key) : add key to collection
Key top():returns most recently-added key
Empty():are there any elements?
Key pop():removes and return recently-added key
```

برای تشخیص درست بودن ترتیب پرانتز ها میتوان از stack استفاده کرد

```
Data: string
Result: return IsBalanced
Stack stack;
while not at end of string do
    if char in ['[' ,'('] then
        stack.Push(char);
    else
        if stack.Empty() then
            return false;
        else
            top=stack.Pop();
            if top='[' and char!=']' or top='(' and char!=')' then
                return false;
            else
                return stack.Empty();
            end
        end
    end
end
```

Algorithm 1: IsBlalnced code

ما میتوانیم stack را با Type Data های مختلف مثل Linkedlist،Queue،array نمایش بدهیم با آرایه:

برای ()push به انتهای آرایه اضافه میشود . برای اینکه ببینیم خالی هست یا نه از ()Empty استفاده میکنیم . نمیتوان resize کرد آرایه را و اگه ارایه پر باشد و بخواهیم یه عنصر اضافه کنیم Error میدهد.

با لینک لیست:

با داستن Head میتوان به ابتدای linkedlist اضافه push)) کرد.

در stack تمام عملیات ها با پیچیدگی زمانی ۱ انجام میشود.

و همچنین در stack هر عنصری که آخر وارد بشود اول هم خارج میشود.

صف :

نوع داده ای با متد های زیر :

```
Enqueue (key) : add key to collection
Dequeue():removes and return the last recently-added
Empty():are there any elements?
```

همه ی متد ها با پیچیدگی زمانی ۱ انجام میشوند.

و از قاعده ی FIFO پیروی میکند.

یعنی اینکه هر عنصری که آخر وارد میشود اول از بقیه هم خارج میشود.

پیاده سازی صف با لینک لیست:
برای پیادسازی با LinkedList با داشتن tail و head به انتهای linkedlist اضافه میکنیم.هنگام اضافه کردن به انتهای لیست اضافه کرده و هنگام برداشتن از ابتدای لیست بر میداریم . به اسلاید های جلسه ۱۵ صفحه ی ۱۲۲ مراجعه شود.

پیاده سازی با آرایه:
برای پیادسازی با array یه آرایه و دوپوینتر read (به ابتدای ارایه اشاره کرده)و پوینتر write (به آخرین خونه ای که جدیدا اضافه شده اشاره میکند) در نظر گرفته.هر بار که Enqueue میکنیم write را یه واحد جلو برده و هر بار که Dequeue میکنیم read را یه واحد جلوبرده.به اسلاید های جلسه ۱۵ صفحه ی ۱۶۴ مراجعه شود.

درخت :
از درخت برای کار های مختلفی استفاده میشود(مناطق جغرافیایی و ...).
تعریف درخت:یه درخت یا خالیه یا مجموعه ای از راس هاست که هر کدام یه کلید دارند با یه لیست از بچه ها.
ریشه:بالاترین راس درخت
بچه :یال مستقیمی که از والد خود دارد
جد:پدر یا پدر پدر یا ...
نوه:بچه یا بچه ی بچه یا ...
خواهر برادری:بچه هایی که والد مشترک دارند
برگ:راسی که بچه ندارد
گره داخلی:راسی که برگ نباشد
لول(level) :۱+تعداد یال های بین ریشه و راس
ارتفاع:حداکثر عمق زیر درخت راس معیین شده و دور ترین برگ

هر راس شامل یه کلید و لیستی از بچه هاش وپدرش است.در درخت دودویی هر راس شامل کلید و راس سمت چپ و راس سمت راست و راس پدر است.
براس محاسبه ارتفاع :

Data: tree
Result: return Height
Height(tree):
if *tree==null* **then**
  return 0;
else
  return 1+Max(Height(tree.left),Height(tree.right));
end

Algorithm 2: Height code

براس محاسبه سایز درخت :

```
Data: tree
Result: return Size
Size(tree):
if tree==null then
    return 0;
else
    return 1+Size(tree.left)+Size(tree.right);
end
```

Algorithm 3: Size code

پیمایش درخت:هر راسی را حداقل یه بار مشاهده کنیم. و دو نوع داریم

پیمایش اول عمق: ما قبل از کاوش در زیر یک درخت خواهر و برادر، کاملاً یک زیر درخت را بپیماییم

پیمایش اول سطح: درخت را لول به لول پیمایش میکنیم.

```
Data: tree
Result: traversal
leveltraversal(tree);
if tree is null;
 then
    return;
else
    Queue q;
    q.Enqueue(tree);
    while q is not null do
        node=q.Dequeue();
        print(node);
        if node.left!=null;
         then
            q.Enqueue(node.left);
        else
            if node.right;
             then
                q.Enqueue(node.right)
            else
            end
        end
    end
end
```

Algorithm 4: Traversal code

مدل های پیمایش اول عمق:

InOrder
PreOrder
PostOrder

در روش Inorder برای هر راس ابتدا بچه ی سمت چپ را نوشته بعدش خودش را نوشته بعدش بچه ی سمت راست.

```
Data: tree
Result: print inorder traversal
InorderTraversal(tree):
if tree is null then
    return ;
else
    InOrderTraversal(tree.left) ;
    Print(tree.key) ;
    InOrderTraversal(tree.right);
end
```

Algorithm 5: InOrderTRaversal code

در روش preorder برای هر راس ابتدا خودش را نوشتهو سپس بچه چپ را نوشته بعدش بچه ی سمت راست را نوشته .

```
Data: tree
Result: print PreOrder traversal
PreOrderTraversal(tree):
if tree is null then
    return ;
else
    Print(tree.key) ;
    PreOrderTraversal(tree.left) ;
    PreOrderTraversal(tree.right);
end
```

Algorithm 6: PreOrderTRaversal code

در روش ابتداpostorder بچه ی سمت چپ وسپس بچه سمت راست را نوشته وسپس خودش را نوشته.

```
Data: tree
Result: print PostOrder traversal
PostOrderTraversal(tree):
if tree is null then
    return ;
else
    PostOrderTraversal(tree.left);
    PostOrderTraversal(tree.right);
    Print(tree.key);
end
```

Algorithm 7: PostOrderTRaversal code

Dfs برای هر جور درختی میشه به جز traversal inorder اما BFS برای هر نوع درختی میتوان اجرا کرد چه باینری باشد چه باینری نباشد.

Bibliography