DSA Stacks مداخن

المكدسات

.المكدس هو بنية بيانات يمكن أن تحتوي على العديد من العناصر

.فكر في كومة مثل كومة من الفطائر

في كومة الفطائر، تُضاف الفطائر وتُزال من الأعلى. لذا عند إزالة فطيرة، ستكون دائمًا آخر فطيرة أضفتها. هذه الطريقة في تنظيم العناصر تسمى LIFO: آخر داخل أول خارج.

العمليات الأساسية التي يمكننا القيام بها على الكومة هي

- الدفع: إضافة عنصر جديد على الكومة.
- البوب: إزالة وإرجاع العنصر العلوي من المكدس.
- نظرة خاطفة : تُرجع العنصر العلوي على المكدس.
- فارغ: يتحقق مما إذا كانت الكومة فارغة.
- الحجم: يبحث عن عدد العناصر في المكدس.

.جرّب هذه العمليات الأساسية في الرسوم المتحركة للمكدس أعلاه

.يمكن تنفيذ المكدسات باستخدام المصفوفات أو القوائم المرتبطة

يمكن استخدام المكدسات لتنفيذ آليات التراجع، أو للعودة إلى الحالات السابقة، أو لإنشاء خوارزميات للبحث بعمق أولًا في الرسوم البيانية، أو للتتبع العكسي.

.غالبًا ما يتم ذكر المكدسات مع قوائم الانتظار، وهي بنية بيانات مشابهة موصوفة في الصفحة التالية

تنفيذ المكدس باستخدام المصفوفات

لفهم فوائد استخدام المصفوفات أو القوائم المرتبطة لتنفيذ المكدسات بشكل أفضل، يجب عليك الاطلاع على ما يشرح كيفية تخزين المصفوفات والقوائم المرتبطة في الذاكرة.

:هكذا يبدو الأمر عندما نستخدم مصفوفة كمكدس

:أسباب تنفيذ الأكوام باستخدام المصفوفات

- كفاءة الذاكرة: لا تحتفظ عناصر المصفوفات بعنوان العناصر التالية كما تفعل عقد القائمة المرتبطة.
- أسهل في التنفيذ والفهم: يتطلب استخدام المصفوفات لتنفيذ المكدسات شيفرة أقل من استخدام القوائم المرتبطة، ولهذا السبب عادةً ما تكون أسهل في الفهم أيضًا.

:سبب لعدم استخدام المصفوفات لتنفيذ المكدسات

- الحجم الثابت: تشغل المصفوفة جزءًا ثابتًا من الذاكرة. هذا يعني أنها قد تشغل ذاكرة أكثر من اللازم، أو إذا امتلأت المصفوفة فلا يمكنها استيعاب المزيد من العناصر.

ملاحظة: عند استخدام المصفوفات في بايثون في هذا البرنامج التعليمي، فإننا نستخدم نوع بيانات "قائمة" بايثون، ولكن في نطاق هذا البرنامج التعليمي يمكن استخدام نوع بيانات "القائمة" بنفس طريقة استخدام المصفوفة. تعرف على المزيد حول قوائم بايثون .

نظرًا لأن قوائم بايثون لديها دعم جيد للوظائف اللازمة لتنفيذ المكدسات، نبدأ بإنشاء مكدس والقيام بعمليات المكدس ببضعة أسطر فقط مثل هذا:

مثال

بايثون

```
مكدس = []

# ادفع
مكدس.append('A')
مكدس.append('B')
stack.append('C')
طباعة("مكدس: "، مكدس)

# البوب
العنصر = stack.pop()
طباعة("البوب: "، عنصر)

# نظرة خاطفة
العنصر العلوي = مكدس[-1]
طباعة("نظرة خاطفة: "، العنصر العلوي)

# هي فارغة
isEmpty = ليست bool(stack)
طباعة("isEmpty: "، isEmpty)

# الحجم
طباعة("الحجم: "، len(stack))
```

لكن لإنشاء بنية بيانات للمكدسات بشكل صريح، مع العمليات الأساسية، يجب أن ننشئ فئة مكدس بدلاً من ذلك. تشبه هذه الطريقة لإنشاء المكدسات في Python مثل C و Java. أيضًا طريقة إنشاء المكدسات في لغات البرمجة الأخرى.

مثال

بايثون

```
صنف المكدس:
    def __init__(self):
        self.stack = [] الذات

    تعريف الدفع (الذات، العنصر):
        ذاتي.stack.append(العنصر)

    تعريف البوب (الذات):
        إذا كان self.isEmpty():
            إرجاع "المكدس فارغ"
        إرجاع self.stack.pop()
```

```
: (تعريف نظرة خاطفة (ذاتي
: self.isEmpty() إذا كان
"الإرجاع "المكدس فارغ
self.stack[-1] إرجاع

: isEmpty(self) تعريف
len(self.stack) == 0 يُعيد

: (تعريف الحجم (ذاتي
len(self.stack) إرجاع

إنشاء مكدس #
()مكدسي = مكدس

myStack.push('A') دفع
myStack.push('B')
myStack.push('C')
(myStack.stack.stack ،" :طباعة("المكدس

(myStack.pop()) ،" :طباعة("البوب

(myStack.peek()) ،" :طباعة(("نظرة خاطفة

("isEmpty: "، myStack.isEmpty()))طباعة

("الحجم :طباعة ", myStack.size()))
```

تنفيذ المكدس باستخدام القوائم المرتبطة

:سبب لاستخدام القوائم المرتبطة لتنفيذ المكدسات

- .الحجم الديناميكي: يمكن للمكدس أن ينمو ويتقلص ديناميكيًا، على عكس المصفوفات

:أسباب عدم استخدام القوائم المرتبطة لتنفيذ المكدسات

- .(ذاكرة إضافية: يجب أن يحتوي كل عنصر مكدس على عنوان العنصر التالي (عقدة القائمة المرتبطة التالية
- .سهولة القراءة: قد يصعب على البعض قراءة الشيفرة وكتابتها لأنها أطول وأكثر تعقيدًا

.هكذا يمكن تنفيذ المكدس باستخدام قائمة مرتبطة

مثال

بايثون

```
: صنف العقدة
: (الذات، القيمة)__init__ تعريف
القيمة الذاتية = القيمة
الذات.التالي = لا شيء

: صنف المكدس
: (ذاتي)___init__ تعريف
```

```
الرأس الذاتي = لا شيء
الحجم الذاتي = 0

:(تعريف الدفع (الذات، القيمة
(عقدة_جديدة = عقدة (قيمة
:self.head إذا كان
new_node.next.new_node = self.head
الرأس الذاتي = العقدة الجديدة
الذات.size += 1

:(تعريف البوب (الذات
:()self.isEmpty إذا كان
"إرجاع "المكدس فارغ
self.head = العقدة المنبثقة
الذات.head = الذات.head.next
الذات.size -= 1
popped_node.value.popped_node.value العودة

Def peek(self):
:()self.isEmpty إذا كان
"إرجاع "المكدس فارغ
self.head.value إرجاع

:(الذات)isEmpty تعريف
self.size = = 0 إرجاع

:(الذات)def stackSize
self.size إرجاع

()المكدس الخاص بي = المكدس
myStack.push('A') دفع
myStack.push('B')
myStack.push('C')

(("البوب")طباعة :" ،myStack.pop())
(("نظرة خاطفة")طباعة :" ،myStack.peek())
(("isEmpty: ")طباعة ،myStack.isEmpty()))
(("الحجم")طباعة :" ،myStack.stack.stackSize()))
```