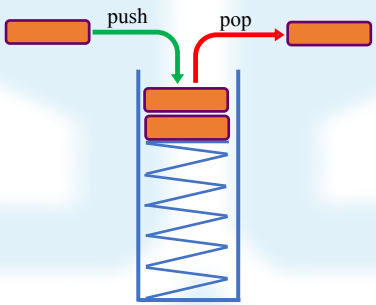
ساختمان داده‌ها و الگوریتم‌ها

Stacks

ا	ه	ه	ت	ش	پ



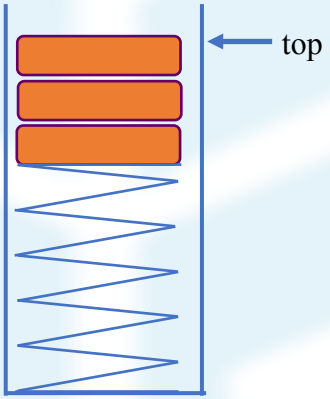
Dr. *Ali Valinejad*

valinejad.ir
valinejad@umz.ac.ir

فهرست مطالب

تعریف پشته

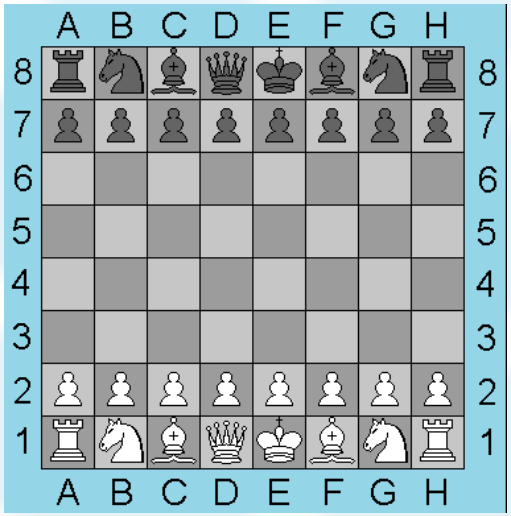
- پشته نوعی لیست خطی است که یک مجموعه از عناصر را طبق ترتیب خاصی ذخیره می‌کند.
- عملیات حذف و اضافه کردن یک عنصر طبق قانون زیر صورت می‌گیرد:
  LAST IN FIRST OUT (LIFO)
  آخرین عنصر ورودی به پشته، اولین عنصری است که خارج می‌شود.
- از نشانگر(اشاره‌گر) *top* برای اشاره به **آخرین** (*بالاترین*) عنصری که داخل پشته قرار می‌گیرد استفاده می‌شود.



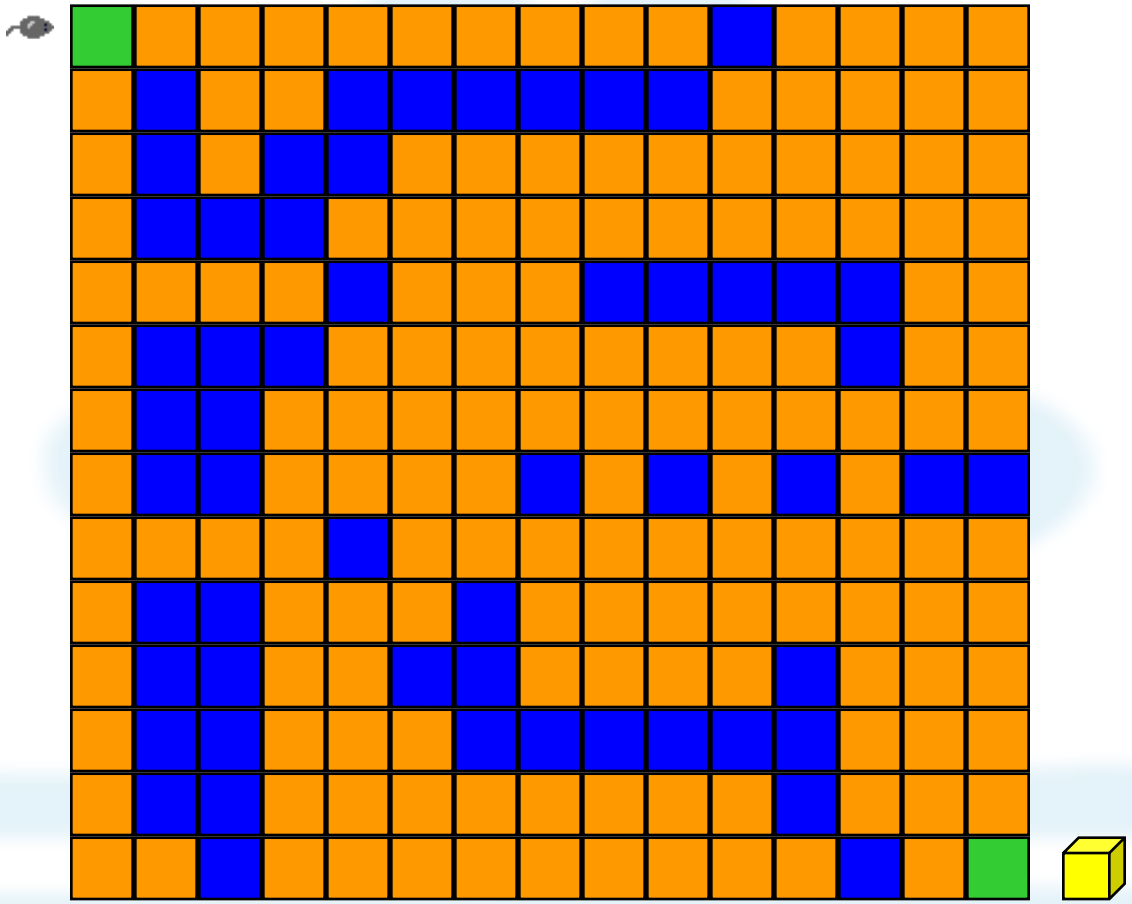
- عملیات حذف و اضافه کردن عناصر فقط از طریق *top* امکان پذیر است.
- عناصر موجود در پشته تنها از طرف بالا و از طریق *top* قابل دسترسی هستند.

کاربردهای پشته

- اجرای برنامه‌ها در کامپیوتر
- برنامه شطرنج
- هزارتو(مسیر پرپیچ وخم)
- ارزیابی عبارات

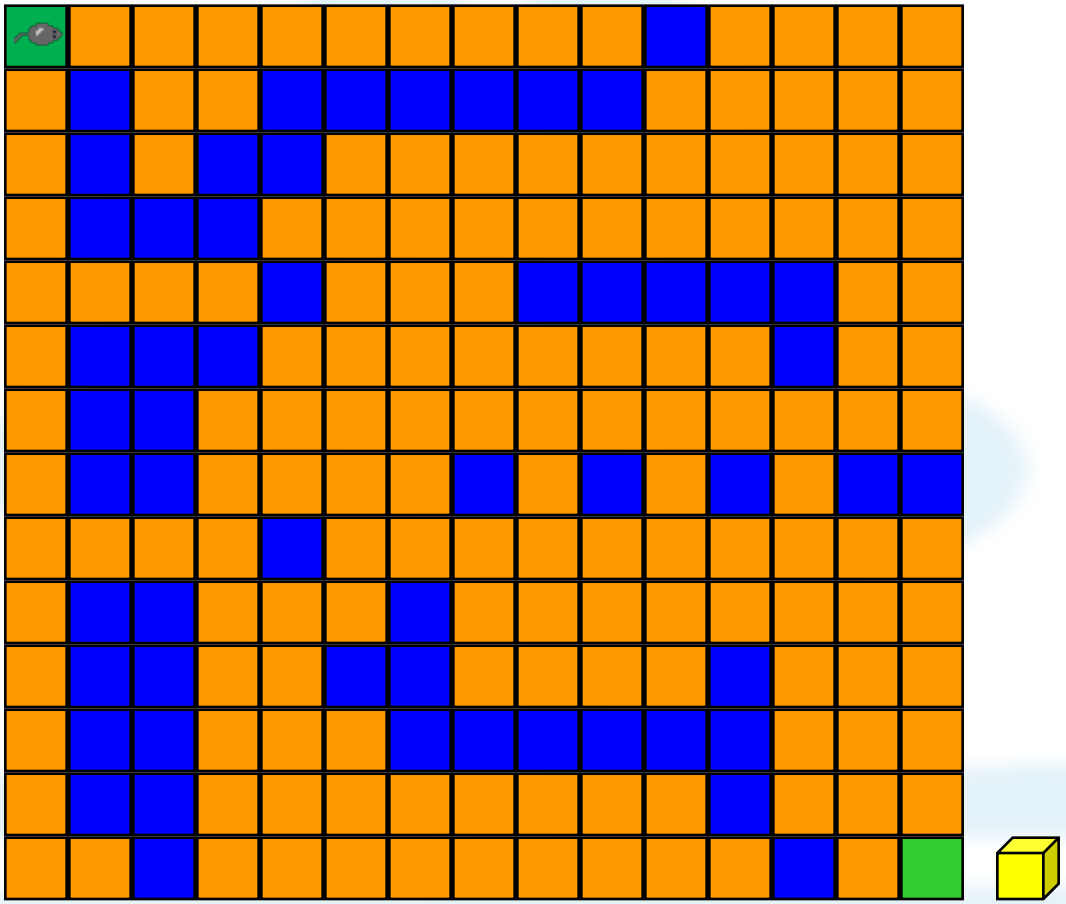


کاربردهای پشته- مساله مسیر پر پیچ و خم



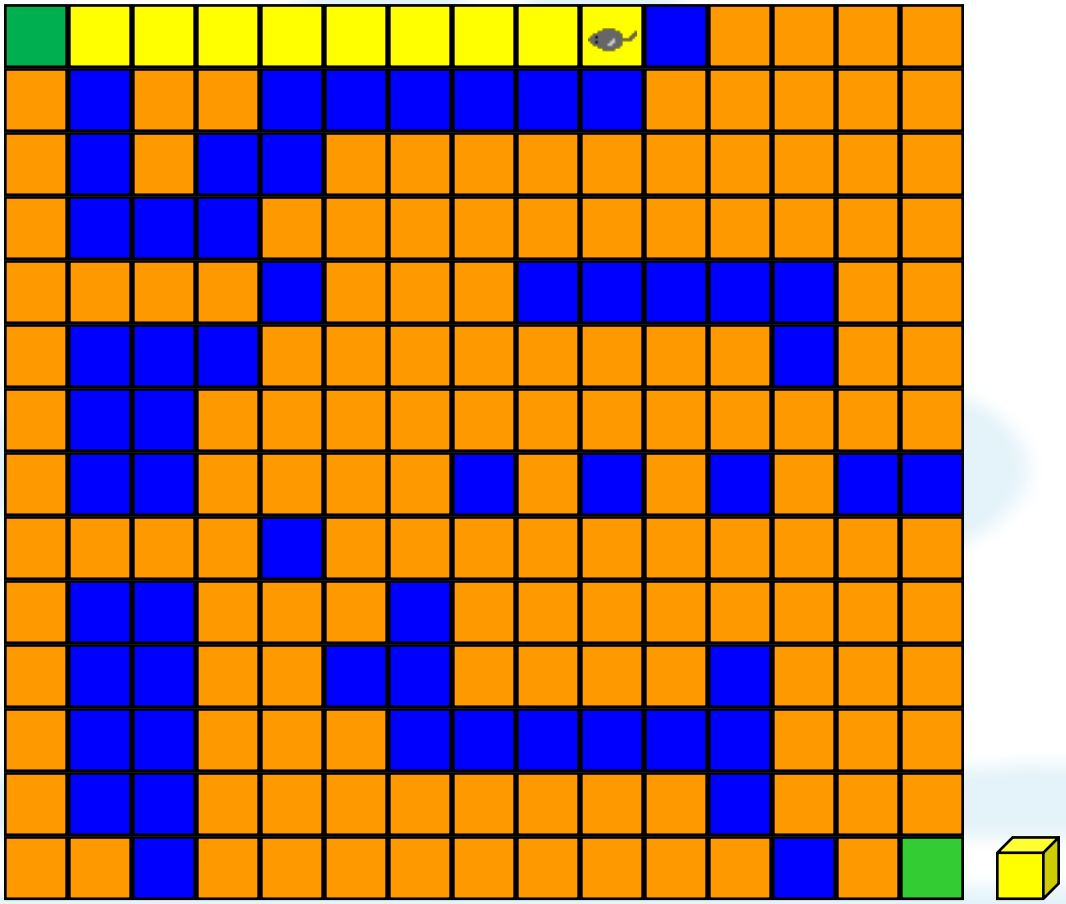
- **ترتیب حرکت:** راست، پایین، چپ، بالا

کاربردهای پشته- مساله مسیر پر پیچ و خم



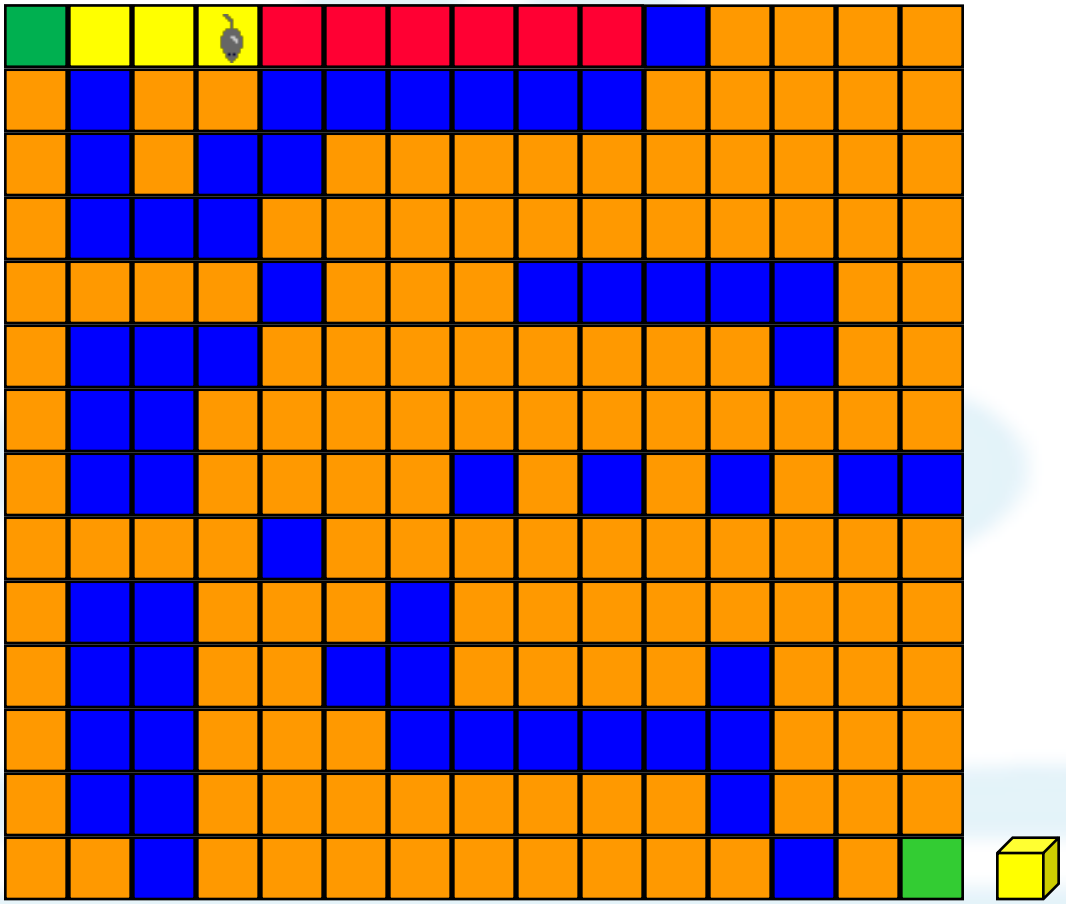
- **ترتیب حرکت:** راست، پایین، چپ، بالا
- حرکت به راست تا جایی که امکان پذیر باشد.

کاربردهای پشته- مساله مسیر پر پیچ و خم



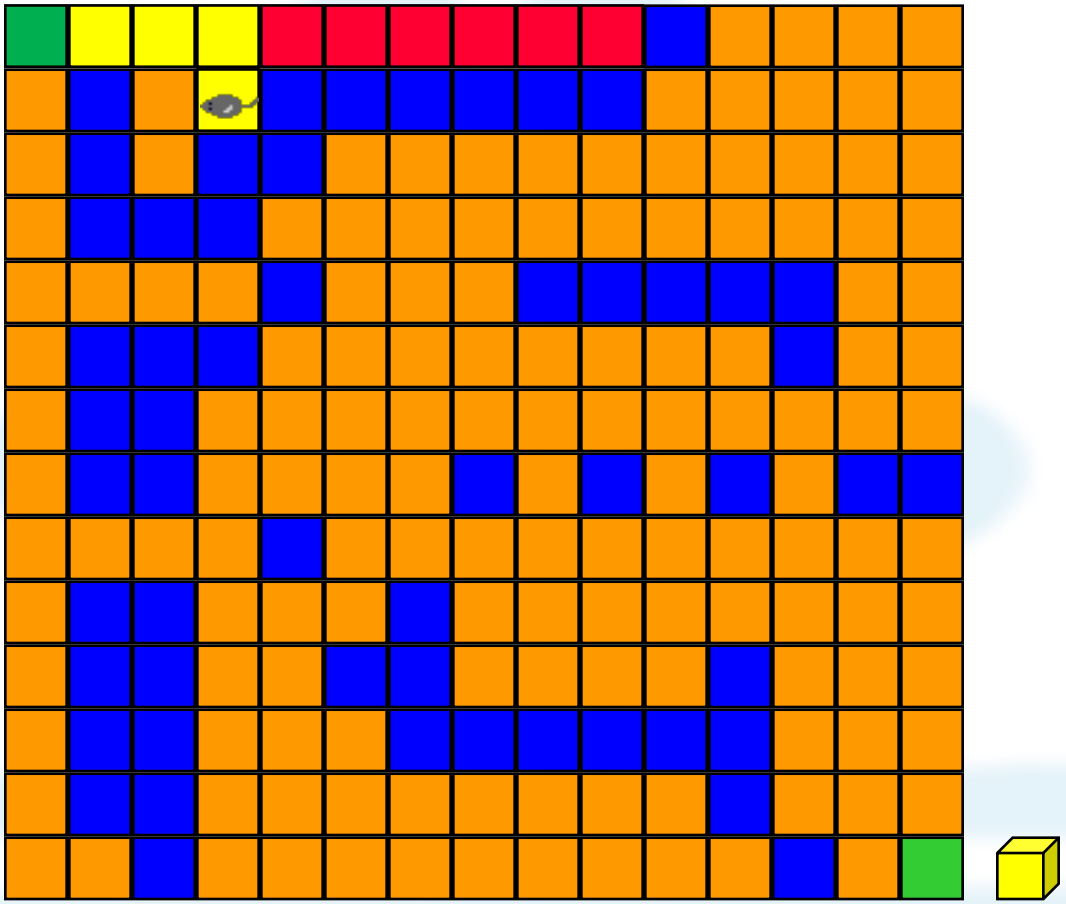
- ترتیب حرکت: ~~راست~~، پایین، چپ، بالا
- حرکت به چپ، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به پایین یا بالا امکان‌پذیر باشد.

کاربردهای پشته- مساله مسیر پر پیچ و خم



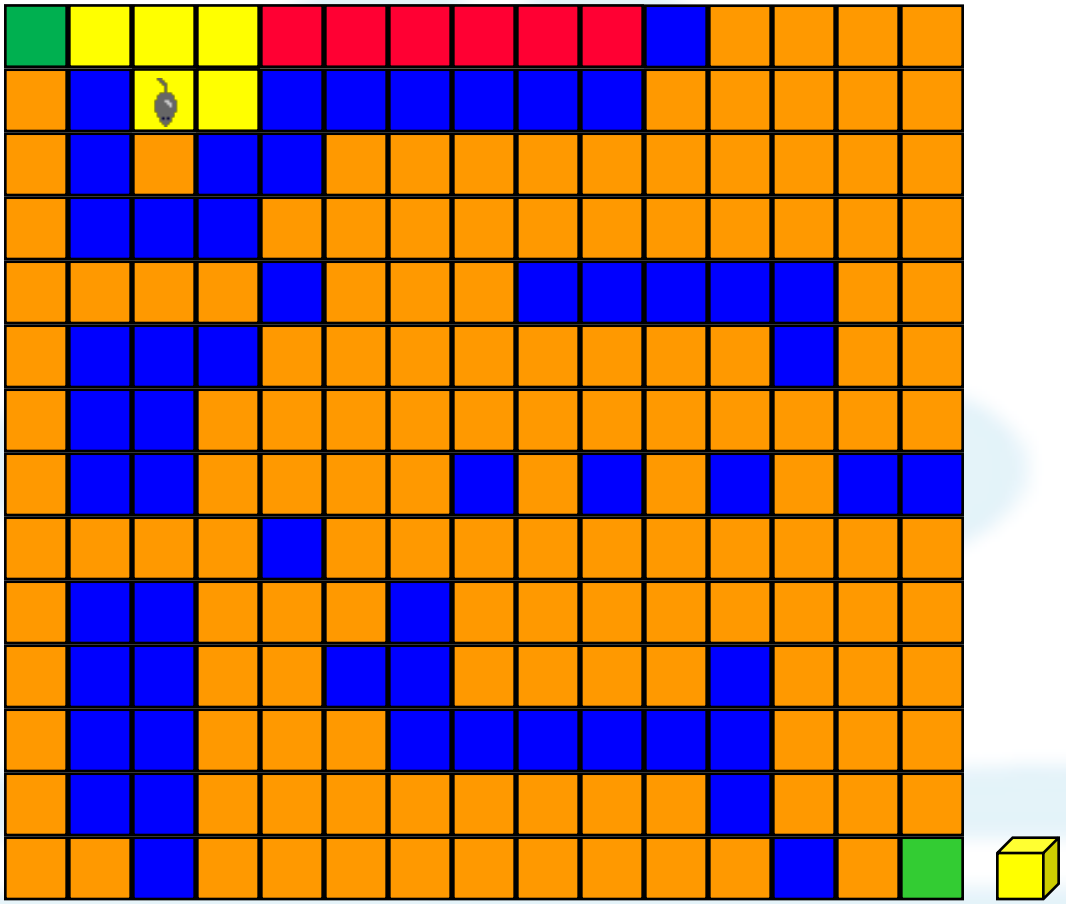
- ترتیب حرکت: ~~راست~~، پایین، چپ، بالا
- حرکت به پایین، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به راست یا چپ امکان‌پذیر باشد.
- ممنوع اعلان کردن موقعیت‌های برگشتی.

کاربردهای پشته- مساله مسیر پر پیچ و خم



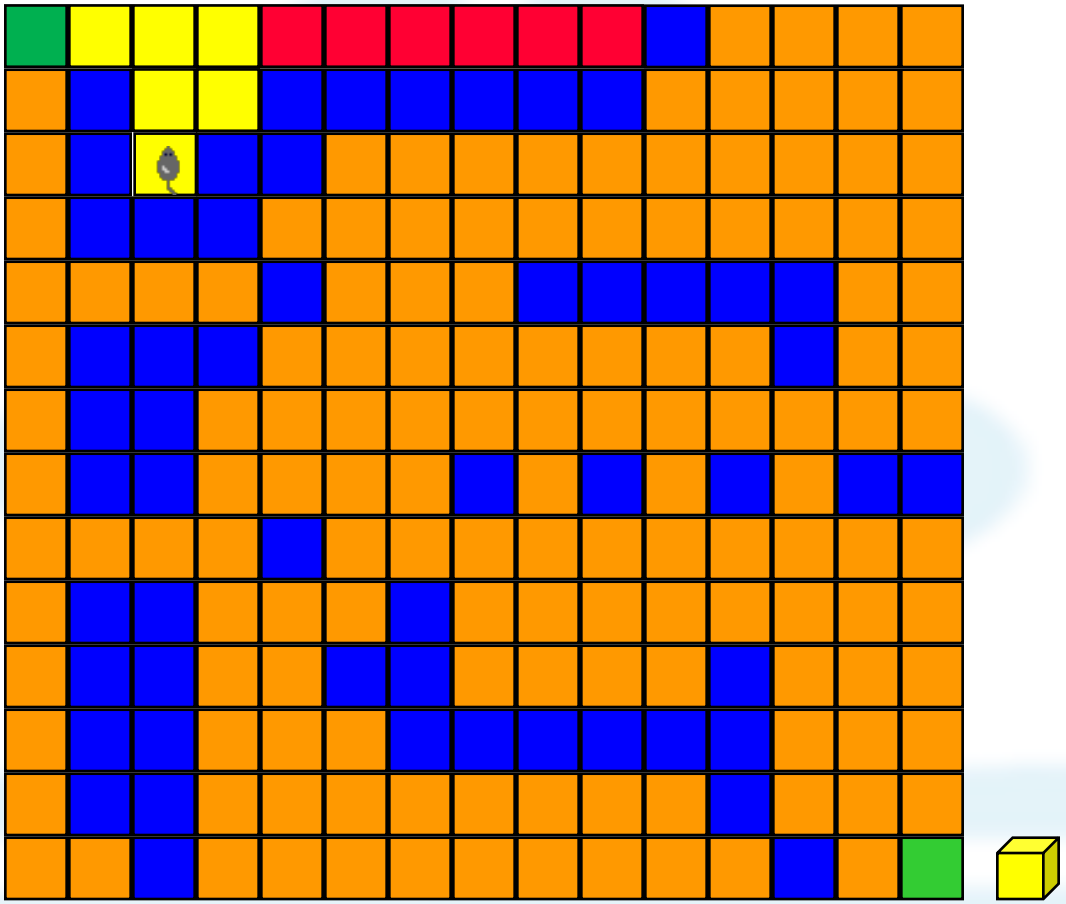
- ترتیب حرکت: ~~راست~~، پایین، چپ، بالا
- حرکت به چپ، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به پایین یا بالا امکان‌پذیر باشد.

کاربردهای پشته- مساله مسیر پر پیچ و خم



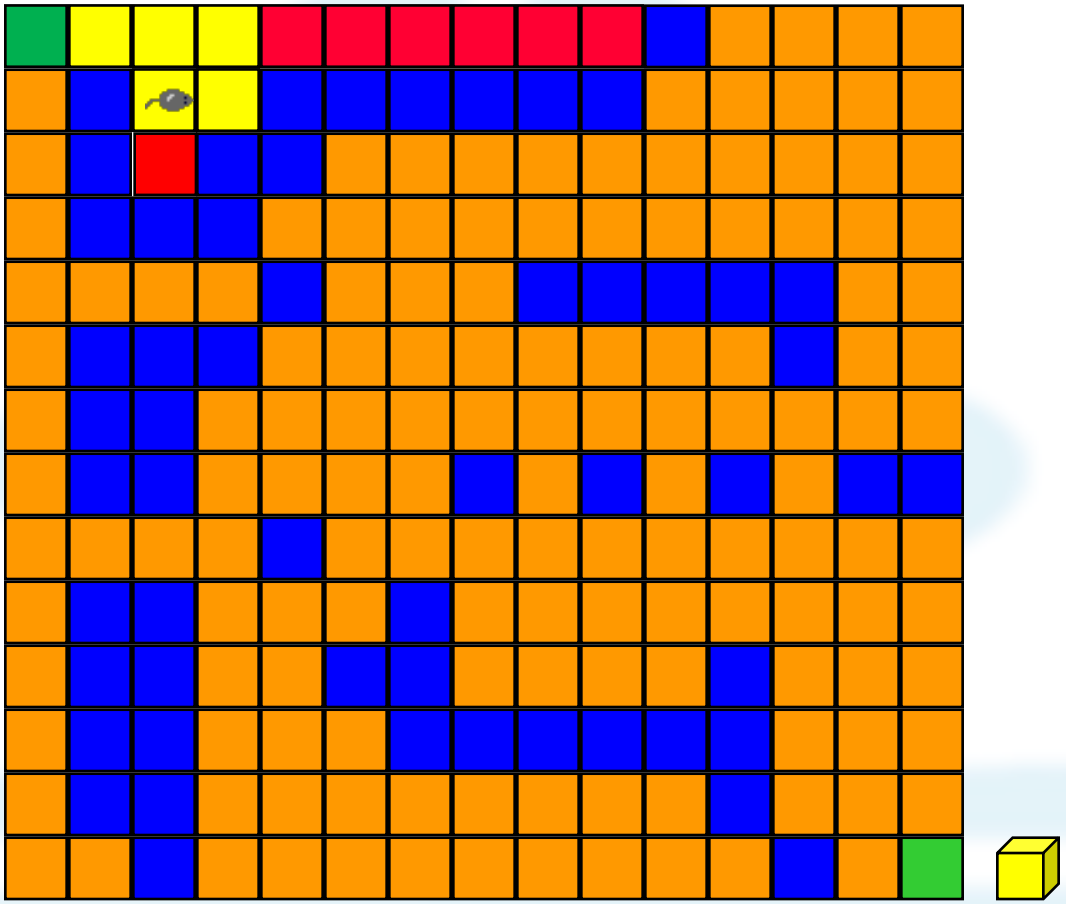
- **ترتیب حرکت:** راست، پایین، چپ، ~~بالا~~
- حرکت به پایین، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به راست یا چپ امکان‌پذیر باشد.

کاربردهای پشته- مساله مسیر پر پیچ و خم



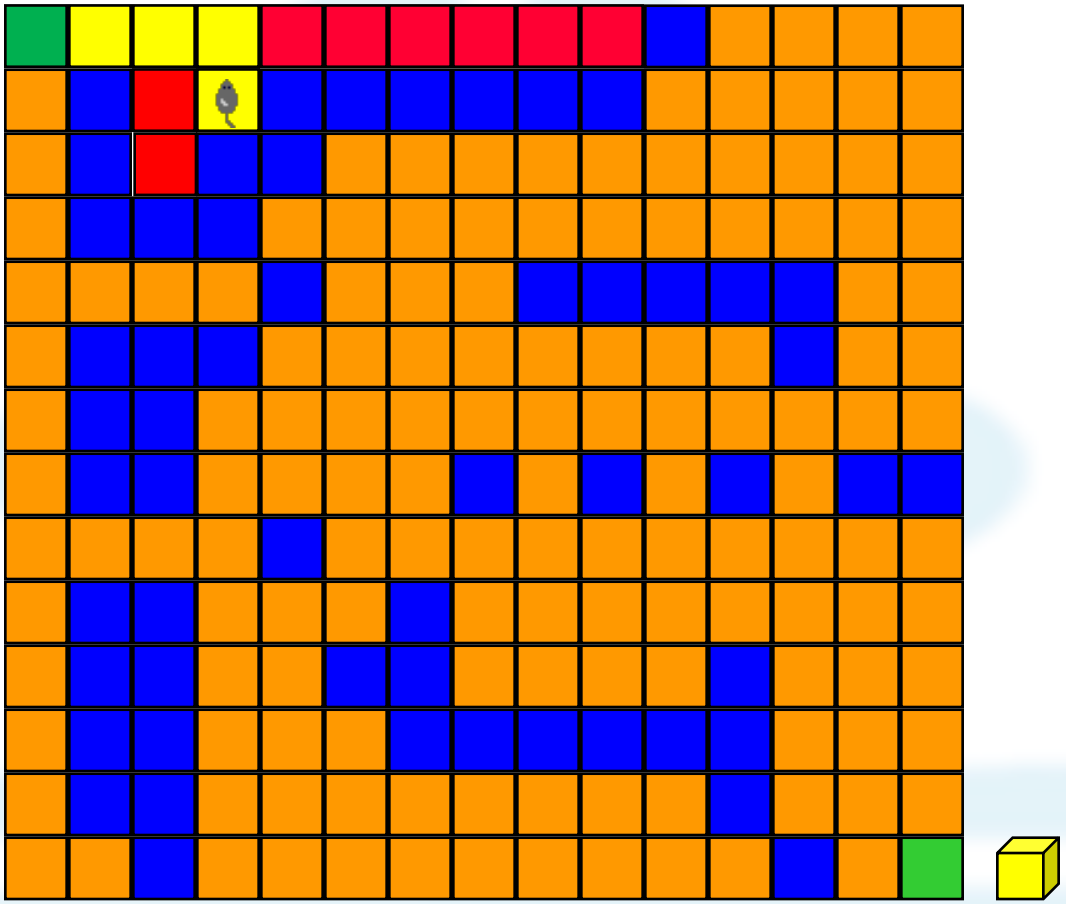
- ترتیب حرکت: راست، ~~پایین~~، چپ، بالا
- حرکت به بالا، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به راست یا چپ امکان‌پذیر باشد.
- ممنوع اعلان کردن موقعیت‌های برگشتی.

کاربردهای پشته- مساله مسیر پر پیچ و خم



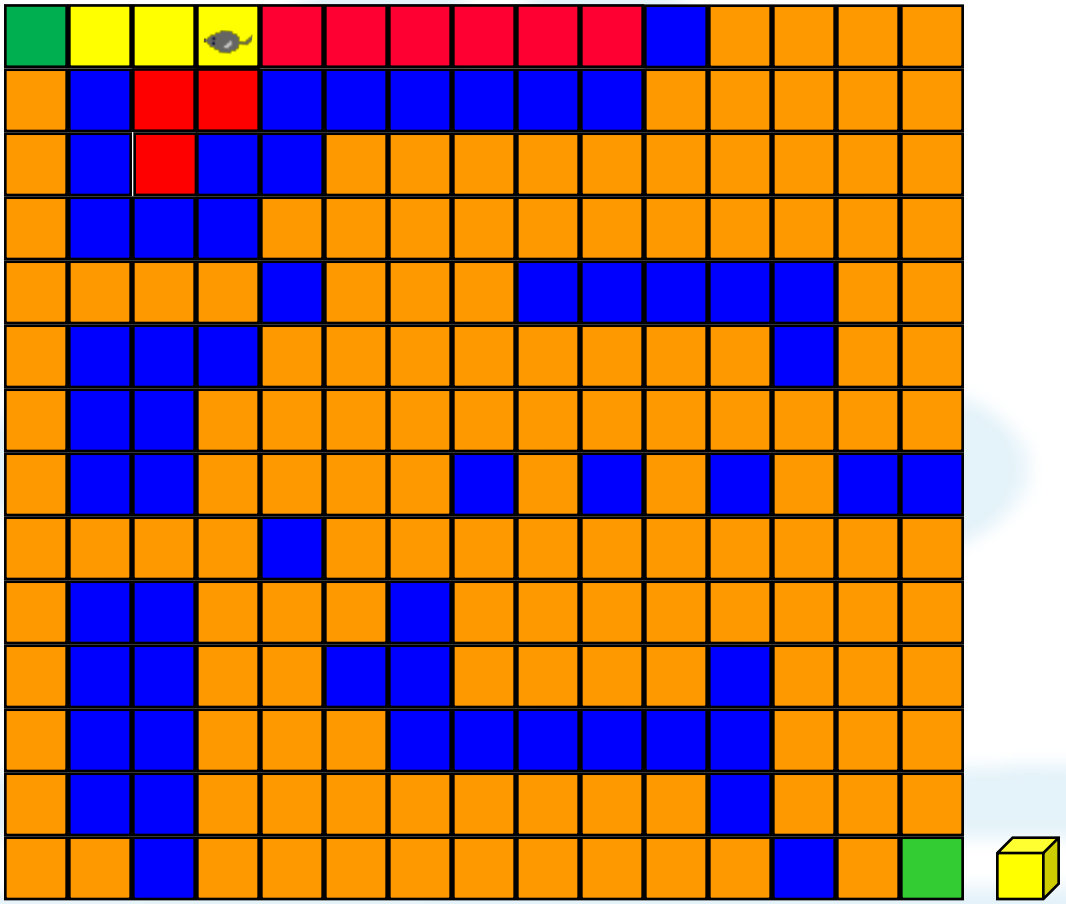
- ترتیب حرکت: راست، پایین، ~~چپ~~، بالا
- حرکت به راست، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به پایین یا بالا امکان‌پذیر باشد.
- ممنوع اعلان کردن موقعیت‌های برگشتی.

کاربردهای پشته- مساله مسیر پر پیچ و خم



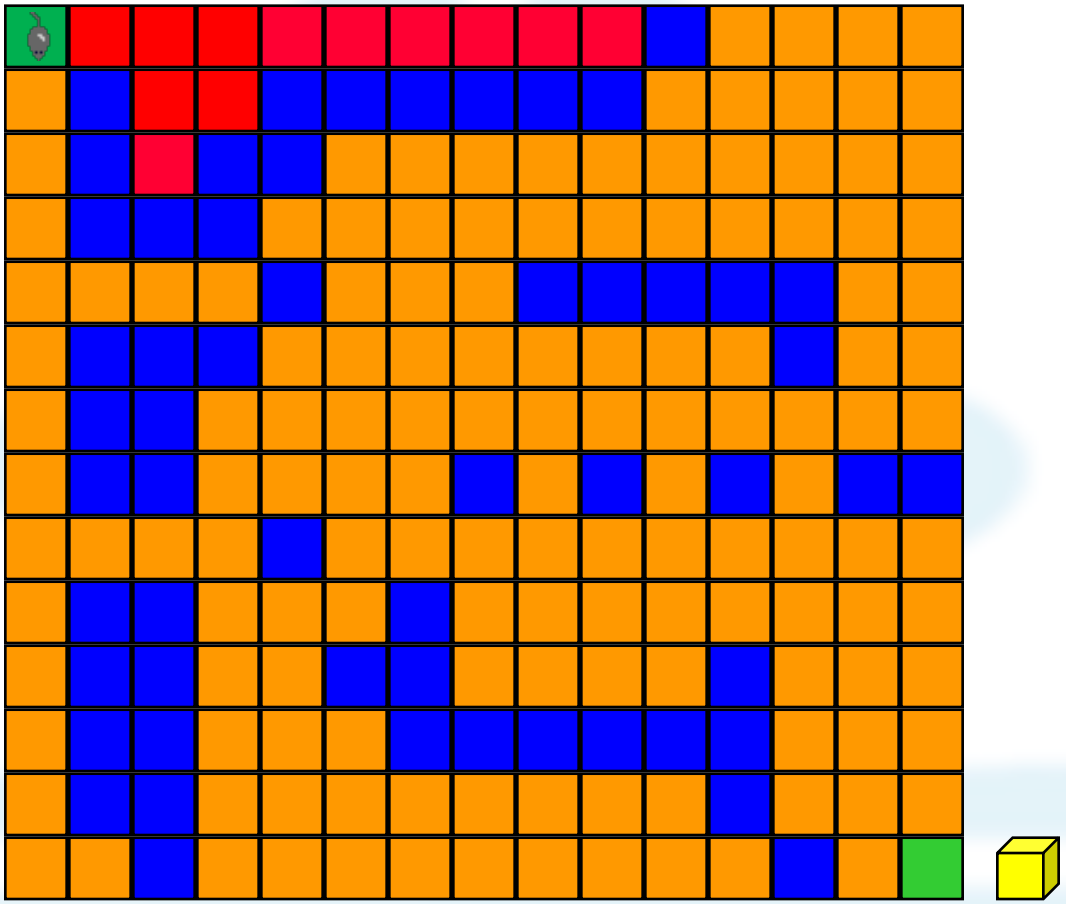
- ترتیب حرکت: ~~راست~~، ~~پایین~~، چپ، بالا
- حرکت به بالا، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به چپ امکان‌پذیر باشد.
- ممنوع اعلان کردن موقعیت‌های برگشتی.

کاربردهای پشته- مساله مسیر پر پیچ و خم



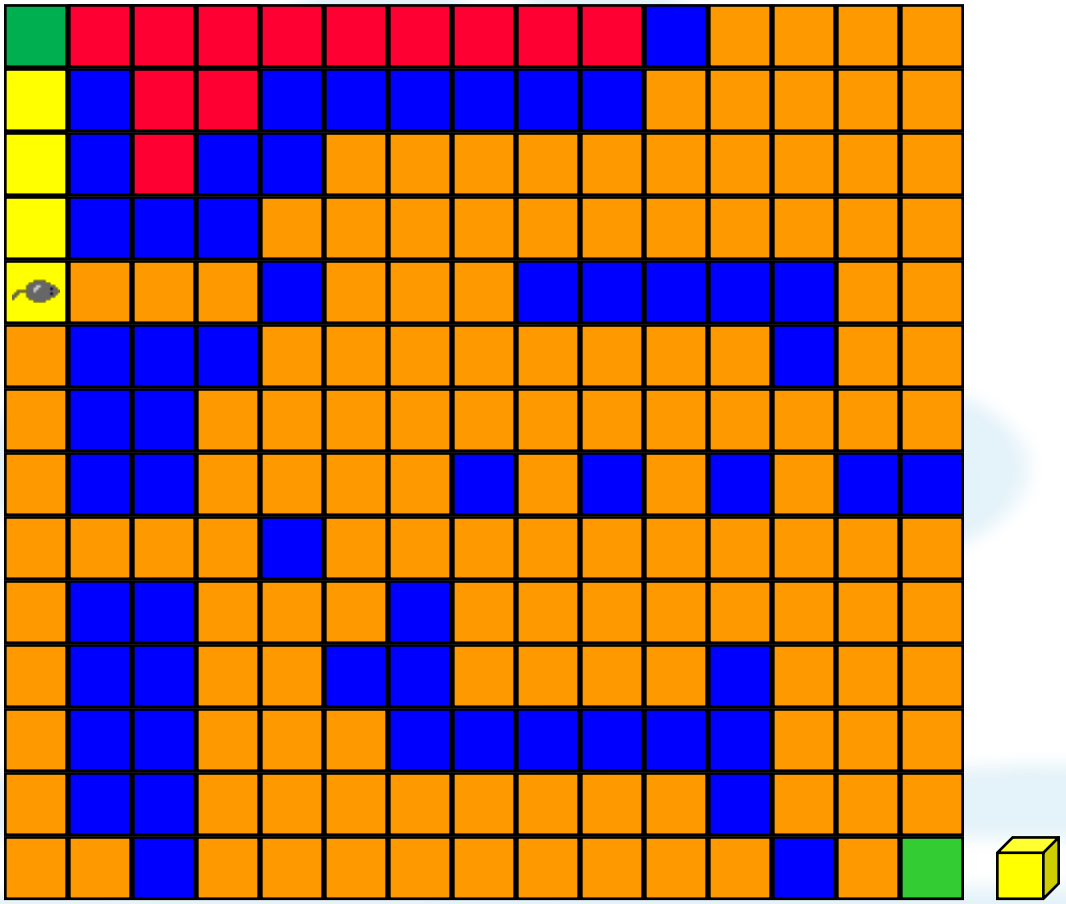
- ترتیب حرکت: ~~راست~~، پایین، چپ، بالا
- حرکت به چپ، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به پایین یا چپ امکان‌پذیر باشد.
- ممنوع اعلان کردن موقعیت‌های برگشتی.

کاربردهای پشته- مساله مسیر پر پیچ و خم



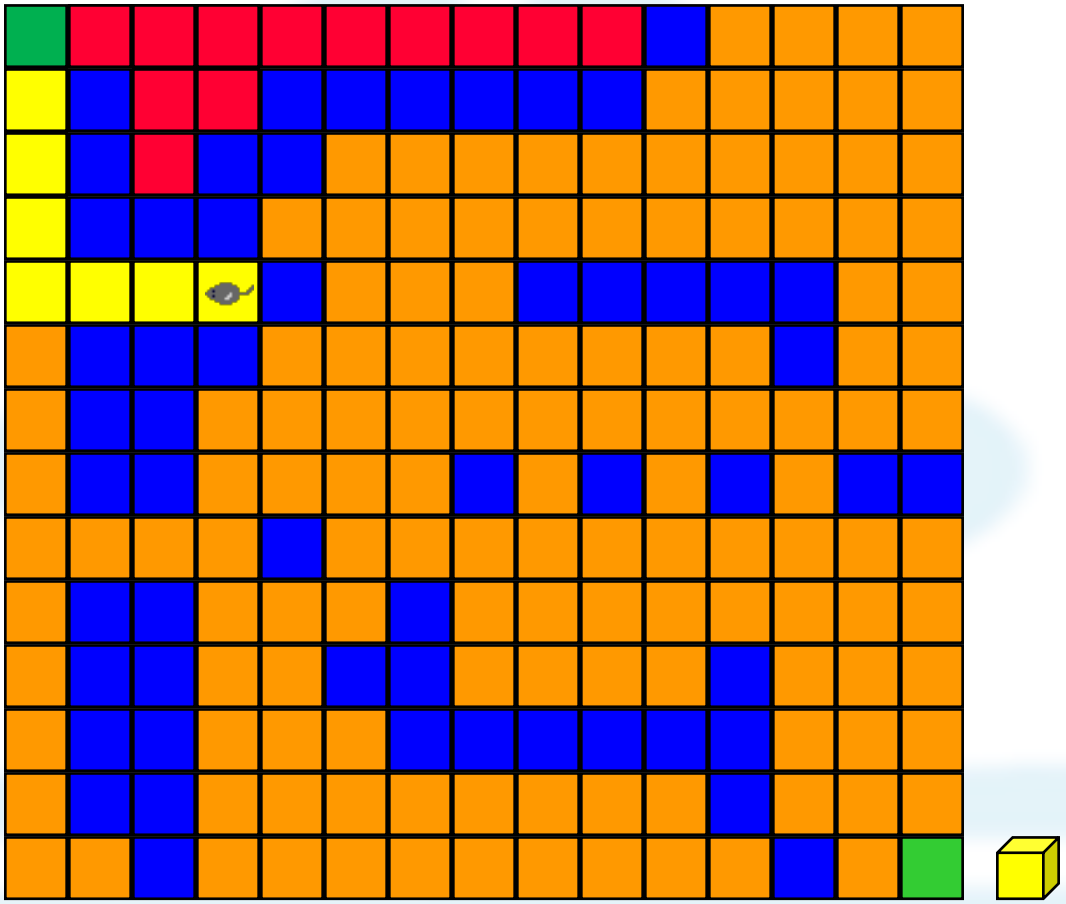
- **ترتیب حرکت:** راست، پایین، چپ، ~~بالا~~
- حرکت به پایین، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به راست یا چپ امکان‌پذیر باشد.

کاربردهای پشته- مساله مسیر پر پیچ و خم



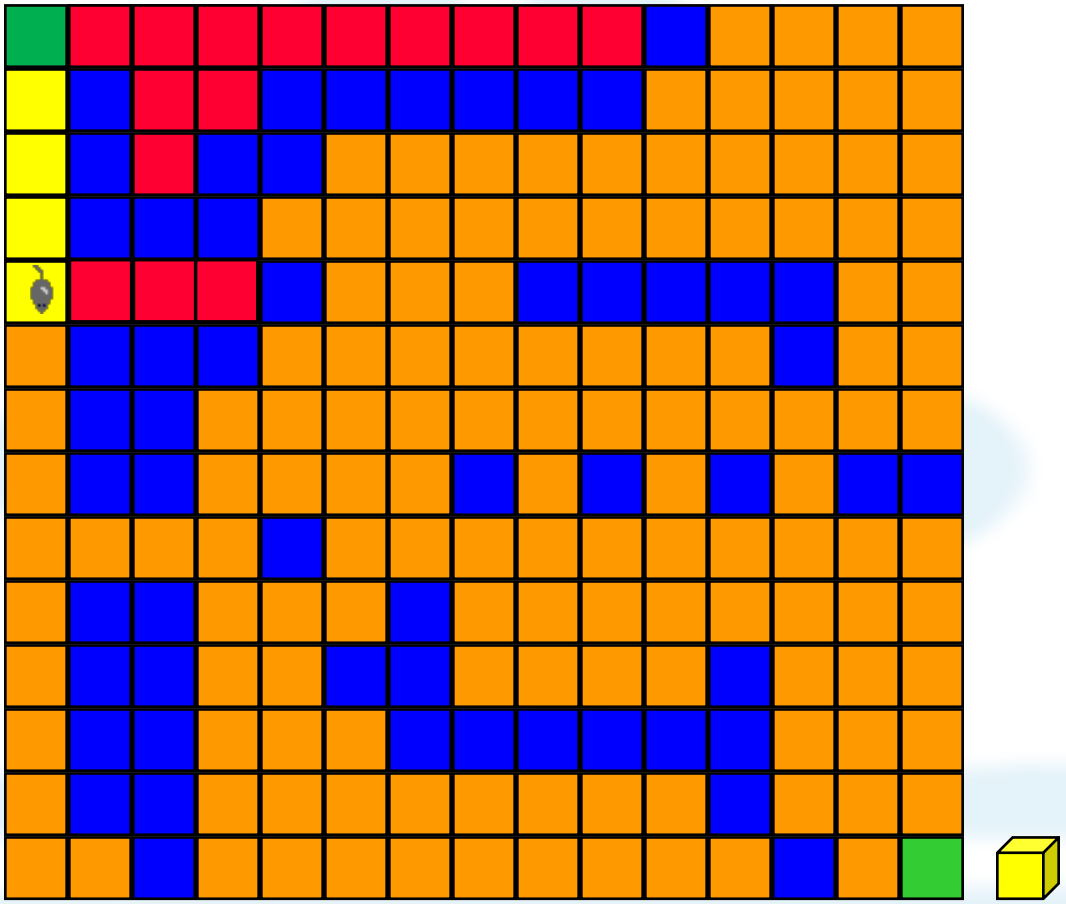
- **ترتیب حرکت:** راست، پایین، چپ، بالا
- حرکت به راست تا جایی که امکان پذیر باشد.

کاربردهای پشته- مساله مسیر پر پیچ و خم



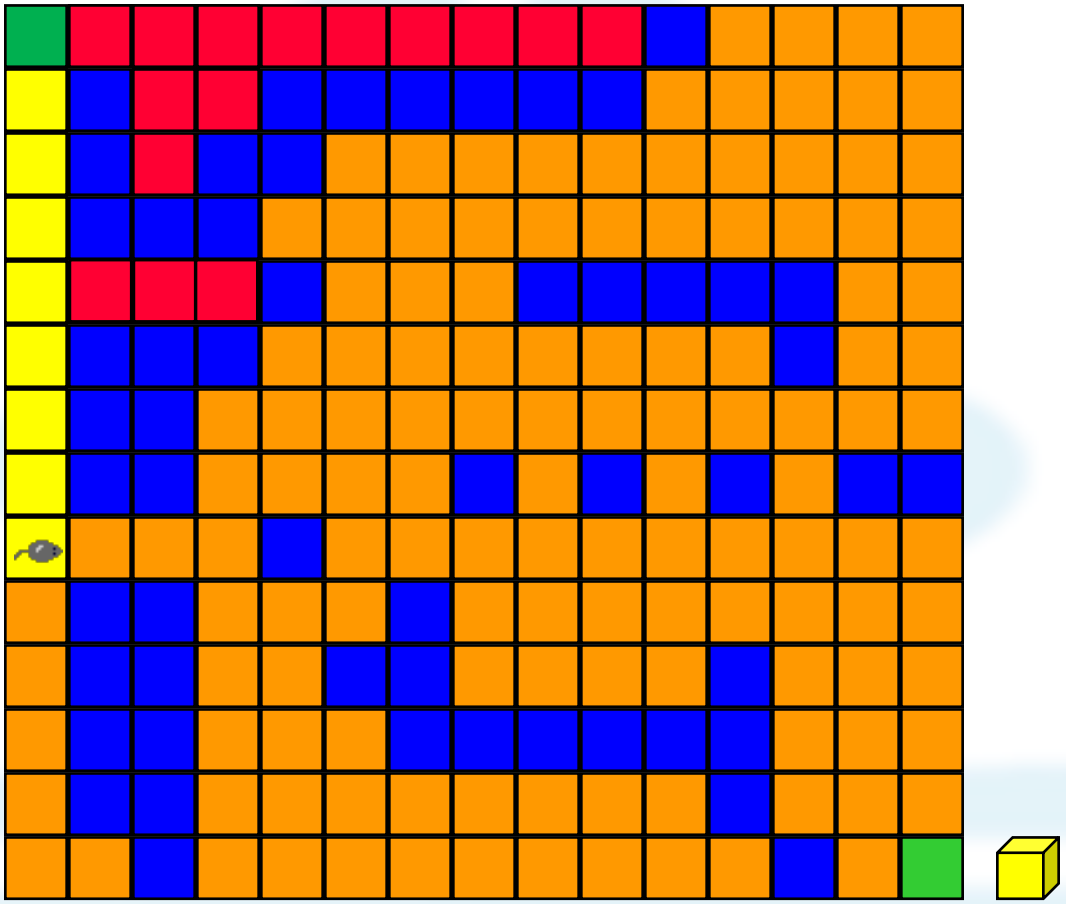
- ترتیب حرکت: ~~راست~~، پایین، چپ، بالا
- حرکت به چپ، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به پایین یا بالا امکان‌پذیر باشد.

کاربردهای پشته- مساله مسیر پر پیچ و خم



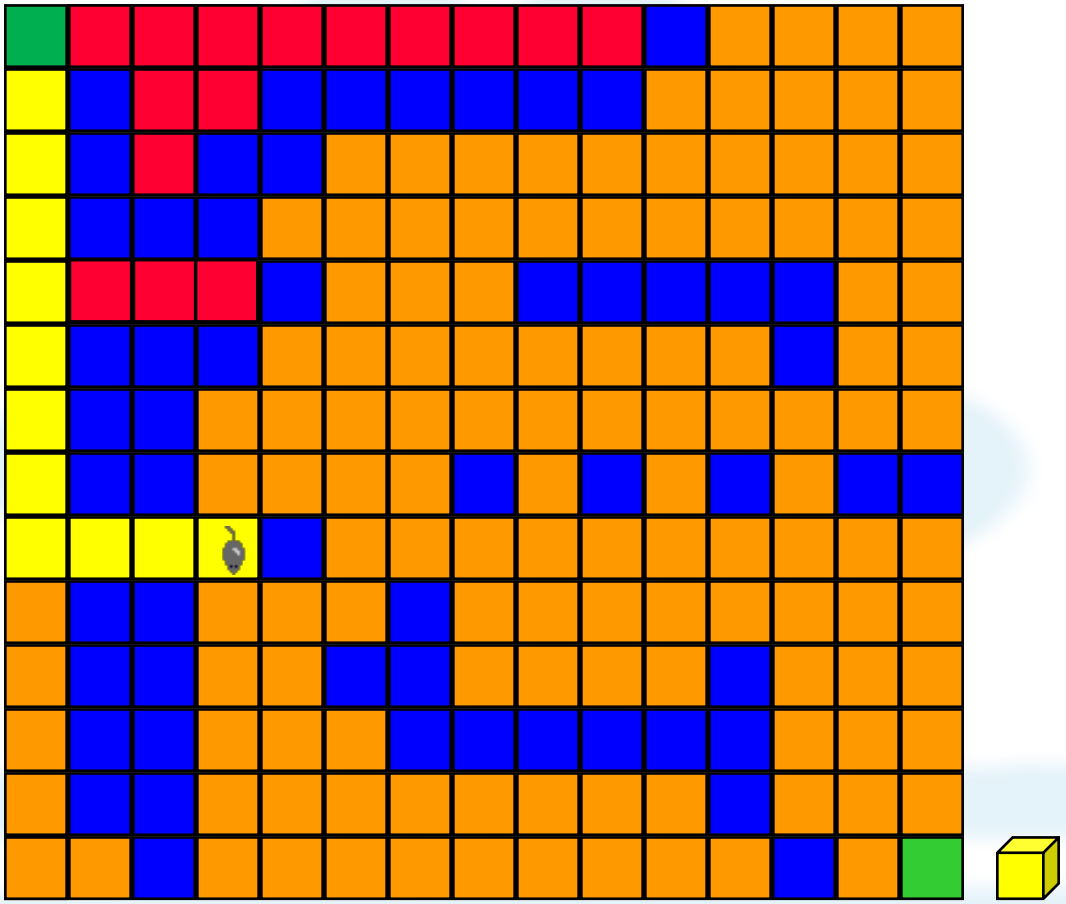
- **ترتیب حرکت:** راست، پایین، چپ، ~~بالا~~
- حرکت به پایین، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به راست یا چپ امکان‌پذیر باشد.

کاربردهای پشته- مساله مسیر پر پیچ و خم



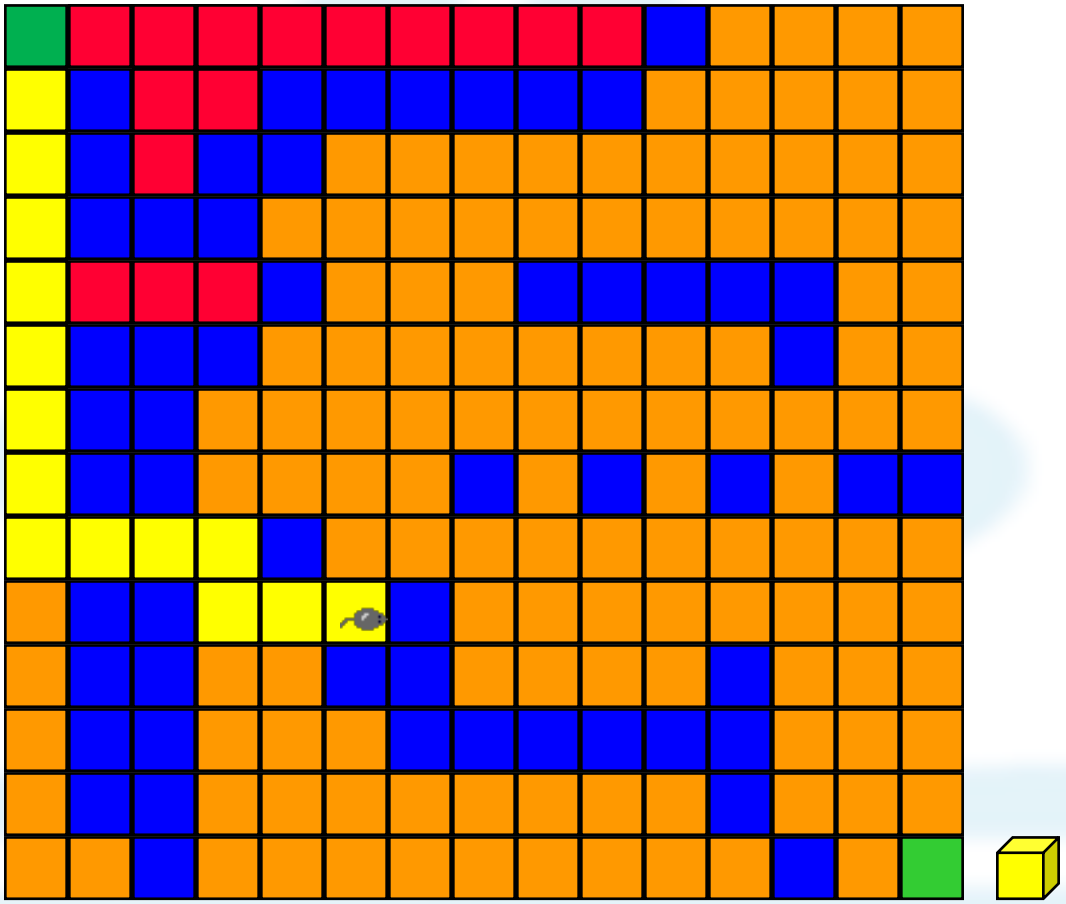
- **ترتیب حرکت:** راست، پایین، چپ، بالا
- حرکت به راست تا جایی که امکان پذیر باشد.

کاربردهای پشته- مساله مسیر پر پیچ و خم

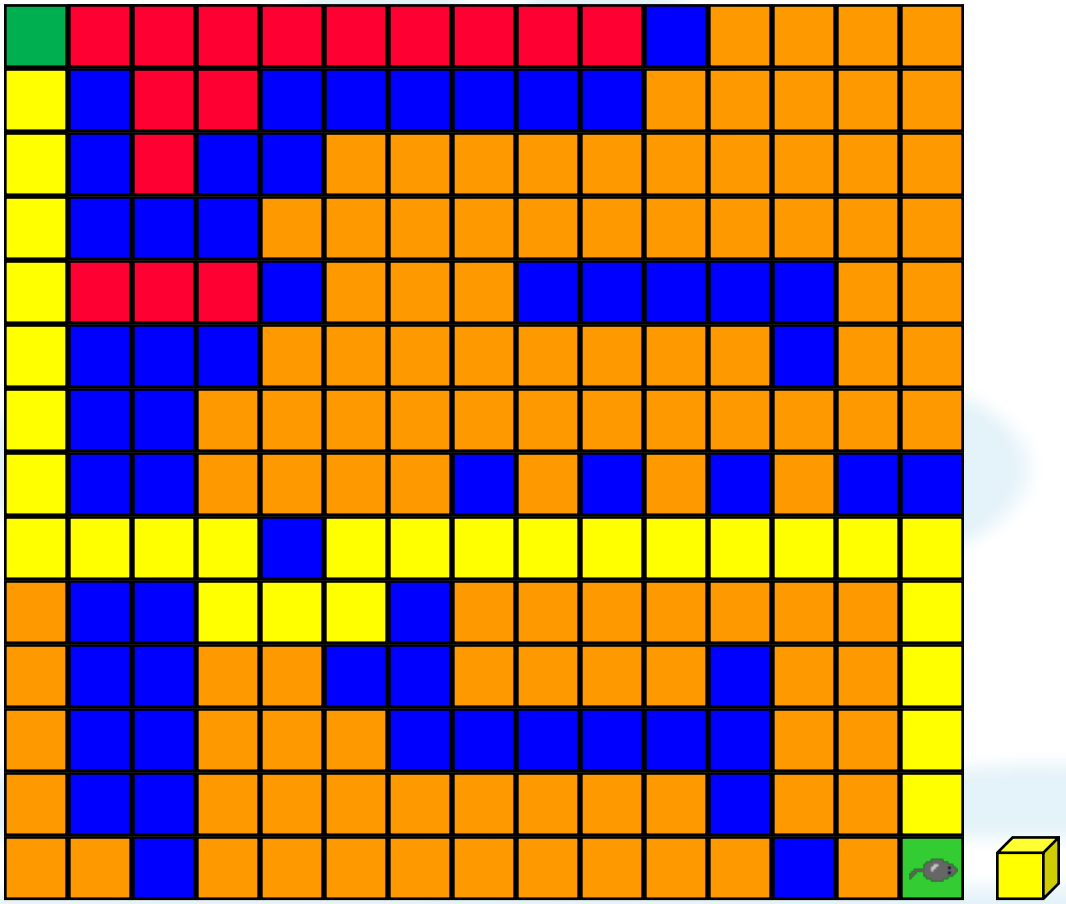


- **ترتیب حرکت:** راست، پایین، چپ، ~~بالا~~
- حرکت به پایین، تا رسیدن به بن بست یا توقف در اولین موقعیتی که از آن جا حرکت به راست یا چپ امکان‌پذیر باشد.

کاربردهای پشته- مساله مسیر پر پیچ و خم

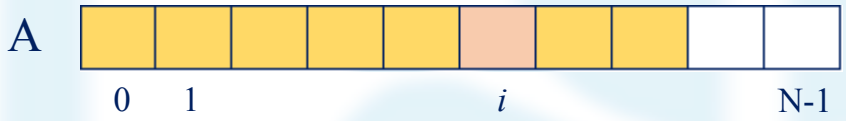


کاربردهای پشته- مساله مسیر پر پیچ و خم

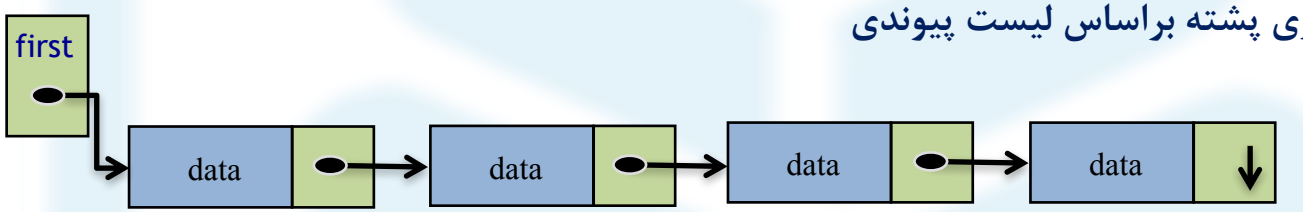


پیاده سازی پشته

- پیاده‌سازی پشته براساس آرایه



- پیاده‌سازی پشته براساس لیست پیوندی



نوع داده انتزاعی پشته

نوع داده انتزاعی پشته(Stack ADT)

تعریف پشته:

پشته، لیستی خطی است که مجموعه‌ای از عناصر را طوری ذخیره می‌کند که فقط از طرف بالای پشته قابل دستیابی‌اند.

عملیات اصلی:

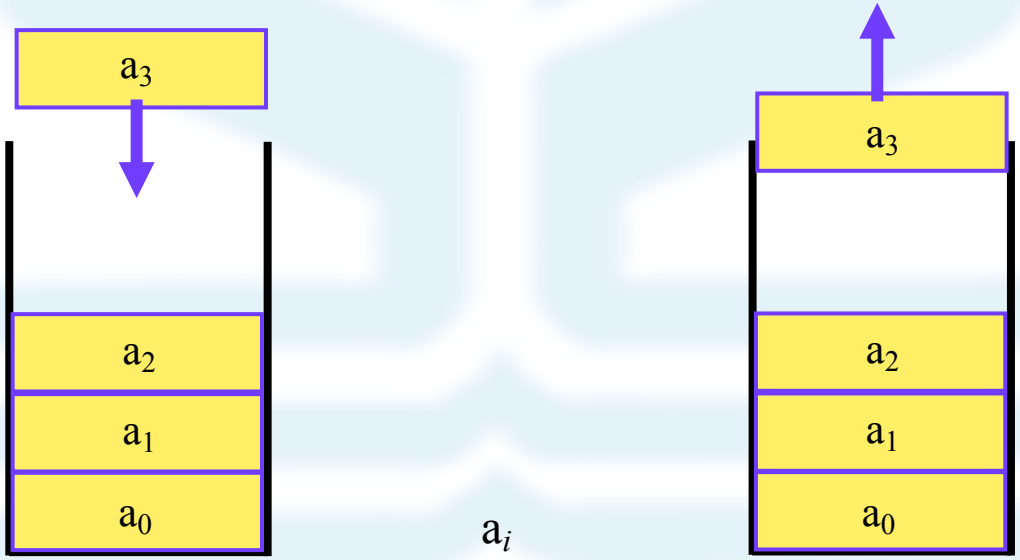
- ایجاد پشته خالی،
- امتحان خالی بودن پشته،
- امتحان پر بودن پشته (*)،
- اضافه کردن عنصر به بالای پشته،
- حذف عنصر از بالای پشته،
- بازیابی عنصر بالایی پشته (بدون حذف آن).

(*) برای پیاده‌سازی پشته بر اساس لیست پیوندی، نیازی به امتحان پر بودن پشته نداریم.

پیاده‌سازی پشته بر اساس آرایه

نکات کلیدی:

- استفاده از آرایه‌ای با اندازه ثابت برای ذخیره‌سازی عناصر پشته.
- تعریف متغیر صحیح *top* برای اشاره به اندیسی از آرایه که متناظر با بالاترین عنصر پشته است.
- ذخیره‌سازی پایین‌ترین عنصر پشته در درایه‌ی با اندیس صفر از آرایه.
- یک واحد افزایش به مقدار *top* با اضافه شدن یک عنصر به پشته،
- یک واحد کاهش از مقدار *top* با حذف شدن یک عنصر از پشته،



a_i

$0 \leq i < n$

پیاده‌سازی پشته براساس آرایه

- **اعضای داده‌ای کلاس پشته:**
  - آرایه با اندازه ثابت جهت ذخیره‌سازی عناصر پشته.
  - تعریف متغیر صحیح *top* با مقدار اولیه ۱- برای اشاره به بالاترین عنصر پشته است.

پیاده‌سازی پشته براساس آرایه

- **اعضای تابعی کلاس پشته**:
  - ایجاد پشته خالی،
  - امتحان خالی بودن پشته،
  - امتحان پر بودن پشته،
  - اضافه کردن عنصر به بالای پشته،
  - حذف عنصر از بالای پشته،
  - بازیابی عنصر بالایی پشته (بدون حذف آن).

پیاده‌سازی پشته براساس آرایه

```
class ArrayStack1:
    def __init__(self,n):
        self._N = n
        self._data = [None]*self._N
        self.top = -1

    def is_empty(self):

    def is_full(self):

    def push(self, e):

    def top_element(self):

    def pop(self):

    def show(self, text):
```

پیاده‌سازی پشته براساس آرایه

- **تابع ایجاد پشته خالی،**

وظایف تابع:

۱- تعریف یک آرایه

۲- تعریف متغیر top با مقدار اولیه ۱- برای نشان دادن اندیس عنصر بالایی پشته.

```
def __init__(self,n):
    self._N=n
    self._data = [None]*self._N
    self.top=-1
```

پیاده‌سازی پشته براساس آرایه

- تابع امتحان خالی بودن پشته،

```
def is_empty(self):
    return self.top == -1
```

- تابع امتحان پر بودن پشته،

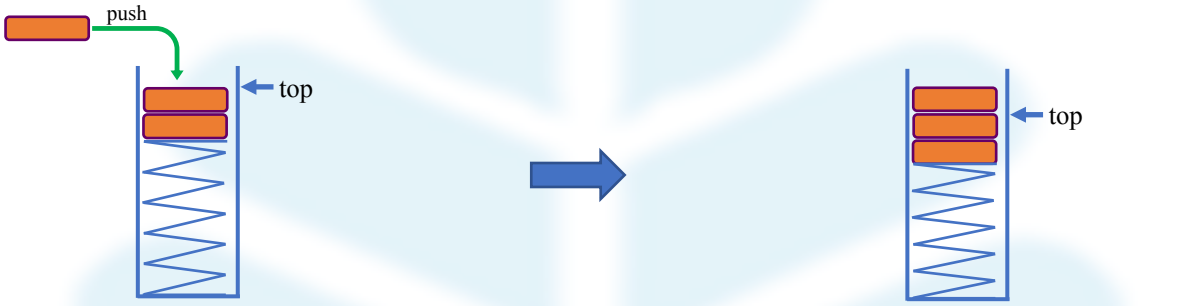
```
def is_full(self):
    return self.top+1 == self._N
```

- بازیابی عنصر بالایی پشته (بدون حذف آن).

```
def top_element(self):
    if self.is_empty():
        return 0,None
    return 1, self._data[self.top]
```

پیاده‌سازی پشته براساس آرایه

- تابع اضافه کردن عنصر به پشته،



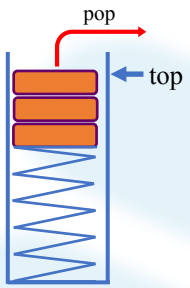
وضعیت پشته قبل از اضافه شدن عنصر جدید

وضعیت پشته بعد از اضافه شدن عنصر جدید

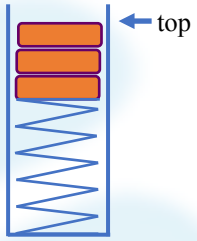
```
def push(self, e):
    if not self.is_full():
        self._data[self.top+1]=e
        self.top+=1
        return 1
    else:
        return 0
```

پیاده‌سازی پشته براساس آرایه

- تابع حذف عنصر از پشته،



وضعیت پشته قبل از حذف عنصر



وضعیت پشته بعد از حذف عنصر

```
def pop(self):
    if self.is_empty():
        return 0,None
    else:
        x=self._data[self.top]
        self.top-=1
    return 1,x
```

پیاده‌سازی پشته براساس آرایه

- تابع نمایش عناصر پشته

```
def show(self, text):
    print(text, end = '')
    for x in range(self.top+1):
            print(self._data[x], end = '->')
    print()
```

پیاده‌سازی پشته براساس آرایه

- تابع اصلی

```
if __name__ == '__main__':
    S = ArrayStack1(5)

    print('elements pushed to stack:')
    for i in range(10):
        flag=S.push(i)
        if (flag==0):
            print('Can\'t push new element(%d) on stack: Stack is full' % (i))
            break
        print(i)
    S.show('Stack after elements are pushed:    ')
```

```
elements pushed to stack:
0
1
2
3
4

Can't push new element(5) on stack: Stack is full
Stack after elements are pushed:    0→1→2→3→4→
```

پیاده‌سازی پشته براساس آرایه

- تابع اصلی (ادامه)

```
flag,element=S.top_element()
if flag:
    print('\n top element:',element)

print('\nelements popped from stack:')
for i in range(10):
    flag, element =S.pop()
    if (flag== 0):
        print('Can\'t pop  element from stack: Stack is empty')
        break
    print(element)
S.show('Stack after elements are popped:')
```

```
 top element: 4

elements popped from stack:
4
3
2
1
0

Can't pop  element from stack: Stack is empty
Stack after elements are popped:
```

پیاده‌سازی پشته براساس آرایه(با استفاده از قابلیت کلاس list)

کلاس ArrayStack را می‌توان با استفاده ار قابلیت کلاس لیست پایتون بازنویسی کرد.

- برای ذخیره‌سازی عناصر پشته، از شی‌ای از کلاس list پایتون استفاده می‌کنیم.
- برای اضافه و حذف کردن عنصر از توابع append و pop استفاده می‌کنیم.
- نیازی به تعریف متغیر صحیح *top* برای اشاره به بالاترین عنصر پشته نیست.

پیاده‌سازی پشته براساس آرایه(با استفاده از قابلیت کلاس list)

```
class Empty(Exception):
    pass

class ArrayStack2:
    def __init__(self):
        self._data = []

    def __len__(self):
        return len(self._data)

    def is_empty(self):
        return len(self._data) == 0

    def push(self, e):
        self._data.append(e)

    def pop(self):
        if self.is_empty():
            raise Empty('Stack is empty ..')
        return self._data.pop()

    def top(self):
        if self.is_empty():
            raise Empty('Stack is empty')
        return self._data[-1]

    def show(self, text):
        print(text, end='')
        for x in self._data:
            print(x, end='')
            print('->', end='')
        print()

if __name__ == '__main__':
    S = ArrayStack2()
    S.push(5)
    S.show('after push(5): ')
    S.pop()
    S.show('after pop(): ')
```

استفاده از کلاس ArrayStack

آیا در عبارات زیر به ازای هر نشانه ابتدایی، نشانه انتهایی متناظر در جای درست قرار دارد؟

'{[()]}' '{[({})]}'

```
def is_matched(expr):
  """Return True if all delimiters are properly match; False otherwise."""
  lefty =  '({['                          # opening delimiters
  righty = ')}]'                          # respective closing delims
  S = ArrayStack2()
  for c in expr:
    if c in lefty:
      S.push(c)                           # push left delimiter on stack
    elif c in righty:
      if S.is_empty():
        return False                      # nothing to match with
      if righty.index(c) != lefty.index(S.pop()):
        return False                      # mismatched
  return S.is_empty()                     # were all symbols matched?

is_matched('{[()]}')
```

```
output:
    False
```

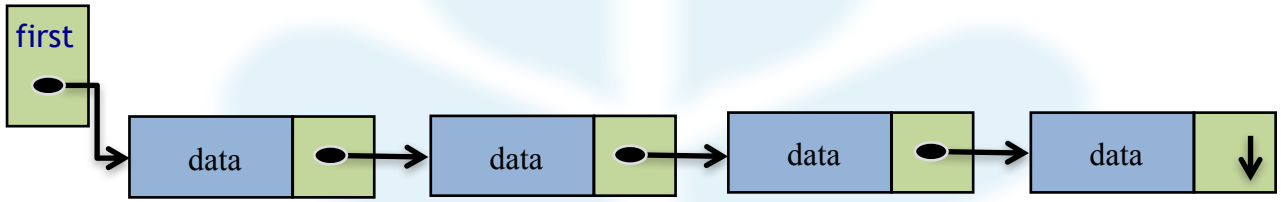
استفاده از کلاس ArrayStack

برنامه‌ای بنویسید که اطلاعات یک فایل را از انتها به ابتدا به صورت خط به خط و برعکس در یک فایل جدید رونویسی کند.

```
def reverse_file(filename):
    """Overwrite given file with its contents line-by-line reversed."""
    S = ArrayStack2()
    original = open(filename)
    for line in original:
        S.push(line.rstrip('\n'))     # we will re-insert newlines when writing
    original.close()

    # now we overwrite with contents in LIFO order
    output = open(filename, 'w')    # reopening file overwrites original
    while not S.is_empty():
        output.write(S.pop() + '\n')  # re-insert newline characters
    output.close()
```

پیاده‌سازی پشته براساس لیست پیوندی



کاربرد پشته- ارزیابی عبارات

انواع عبارات:

۱. عبارت میانوندی: A+B

۲. عبارت پسوندی: AB+

۳. عبارت پیشوندی: +AB

کاربرد پشته- ارزیابی عبارات میانوندی

الگوریتم ارزیابی عبارت میانوندی

گام ۱. الگوریتم تبدیل عبارت میانوندی به پسوندی

گام ۲. ارزیابی عبارت پسوندی

کاربرد پشته- ارزیابی عبارات

الگوریتم تبدیل عبارت میانوندی به پسوندی(روی کاغذ)

a / b - c + d * e - a * c

گام۱: پرانتزگذاری عبارت میانوندی:

a / b - c + d * e - a * c → ((((a / b) - c) + (d * e)) – (a * c))

گام۲: جایگذاری همه عملگرها به جای پرانتز بسته متناظر با آن ها:

((((a / b) - c) + (d * e)) - (a * c)) → ((((a b / c -(d e * + (a c *-

گام۳: حذف پرانتزها:

ab/c-de*+ac*-

کاربرد پشته- ارزیابی عبارات

الگوریتم تبدیل عبارت میانوندی به پسوندی

ورودی: عبارت میانوندی

خروجی: عبارت پسوندی

گام۱: یک پشته خالی برای ذخیره‌سازی عملگرها ایجاد کنید.

گام۲(گام تکرار): عملیات زیر را تا رسیدن به انتهای عبارت میانوندی و یا رخ دادن خطا تکرار کنید:

الف: نشانه بعدی (ثابت، متغیر، عملگر، پرانتز باز، پرانتز بسته) را از عبارت میانوند بخوانید.

ب: اگر نشانه

پرانتز بسته باشد، عناصر پشته را تا رسیدن به پرانتز باز حذف کنید و در خروجی قرار دهید. پرانتز باز را در نظر نگیرید. (اگر پشته خالی شد و به پرانتز باز نرسیدید، خطایی در عبارت میانوند وجود دارد).

عملگر باشد، اگر پشته خالی و یا تقدم این عملگر از تقدم عملگر بالای پشته بیشتر باشد، این عملگر را درپشته قرار دهید، در غیر این صورت، عنصر بالای پشته را حذف کنید و در عبارت خروجی قرار دهید. این عمل را برای بالاترین عنصر پشته در صورت لزوم تکرار کنید. (تقدم پرانتز بازی که در پشته قرار دارد، از تقدم سایر عملگرها کمتر است.)

عملوند باشد، آن را در خروجی قرار دهید.

گام۳: درصورت رسیدن به انتهای پشته، عناصر موجود در پشته را تا خالی شدن پشته، حذف کنید و در خروجی قرار دهید.

کاربرد پشته- ارزیابی عبارات

جدول تقدم عملگرها

Token	Operator	Precedence[1]	Associativity
( ) [ ] -> .	function call array element struct or union member	17	left-to-right
-- ++	increment, decrement[2]	16	left-to-right
-- ++ ! - - + & * sizeof	decrement, increment[3] logical not one's complement unary minus or plus address or indirection size (in bytes)	15	right-to-left
(type)	type cast	14	right-to-left
* / %	mutiplicative	13	Left-to-right

کاربرد پشته- ارزیابی عبارات

جدول تقدم عملگرها

+ -	binary add or subtract	12	left-to-right
<< >>	shift	11	left-to-right
> >= < <=	relational	10	left-to-right
== !=	equality	9	left-to-right
&	bitwise and	8	left-to-right
^	bitwise exclusive or	7	left-to-right
|	bitwise or	6	left-to-right
&&	logical and	5	left-to-right
||	logical or	4	left-to-right

کاربرد پشته- ارزیابی عبارات

جدول تقدم عملگرها

?:	conditional	3	right-to-left
= += -= /= *= %= <<= >>= &= ^= |=	assignment	2	right-to-left
,	comma	1	left-to-right

کاربرد پشته- ارزیابی عبارات

تبدیل عبارت میانوندی به پسوندی

Infix	Postfix
2+3*4	234*+
a*b+5	ab*5+
(1+2)*7	12+7*
a*b/c	ab*c/
(a/(b-c+d))*(e-a)*c	abc-d+/ea-*c*
a/b-c+d*e-a*c	ab/c-de*+ac*-

کاربرد پشته- ارزیابی عبارات

تبدیل عبارت میانوندی به پسوندی

a+b*(c/(d+e))*f

infix input	Stack [0]	[1]	[2]	[3]	[4]	[5]	postfix
a							a
+	+						a
b	+						ab
*	+	*					ab
(	+	*	(				ab
c	+	*	(				abc
/	+	*	(	/			abc
(	+	*	(	/	(		abc
d	+	*	(	/	(		abcd
+	+	*	(	/	(	+	abcd
e	+	*	(	/	(	+	abcde
)	+	*	(	/			abcde+
)	+	*					abcde+/
*	+	*					abcde+/*
f	+						abcde+/*f*
eos							abcde+/*f*+

کاربرد پشته- ارزیابی عبارات

الگوریتم ارزیابی عبارت پسوندی

ورودی: عبارت پسوندی

خروجی: مقدار ارزیابی شده

گام۱: یک پشته خالی برای ذخیره‌سازی عملوندهاایجاد کنید.

گام۲(گام تکرار): عملیات زیر را تا رسیدن به انتهای عبارت پسوندی تکرار کنید:

الف: نشانه بعدی را از عبارت پسوندی بخوانید.

ب: اگر نشانه

عملوند باشد، آن را در پشته قرار دهید.

عملگر باشد، دو مقدار بالایی پشته را خارج نموده و عملگر را روی آن ها اعمال کنید و نتیجه حاصل را در پشته قرار دهید.

گام۳: درصورت رسیدن به انتهای پشته، مقداری که در آن قرار دارد را به عنوان مقدار ارزیابی عبارت ورودی در خروجی قرار دهید.

کاربرد پشته- ارزیابی عبارات

ارزیابی عبارت پسوندی

12 2 15 4 1 + / * 5 * +

postfix input	Stack [0]	[1]	[2]	[3]	[4]
12	12				
2	12	2			
15	12	2	15		
4	12	2	15	4	
1	12	2	15	4	1
+	12	2	15	4+1=5	
/	12	2	15/5=3		
*	12	2*3=6			
5	12	6	5		
*	12	6*5=30			
+	12+30=42				
eos	**42**				

[1] Text Book, *Data Structures and Algorithms in Python* [Goodrich, Tamassia & Goldwasser 2013.
[2] http://xpzhang.me/teach/DS18_Fall
[۳] ساختمان داده‌ها و الگوریتم- ناصر آیت