# جزوه جلسه نوزده داده ساختارها و الگوریتم

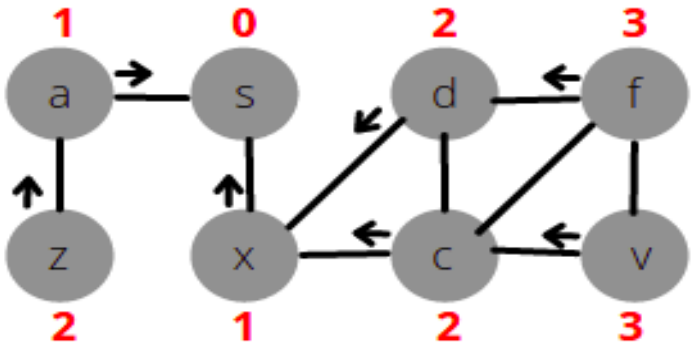
۹ آذر ۱۴۰۰

## فهرست مطالب

# ۱ ادامه مباحث پیمایش BFS

## ۱.۱ مثال از پیمایش BFS یک گراف ۸ راسی

در گراف زیر، مبدا راس s در نظر گرفته شده. لایه ای که هر راس در آن قرار دارد با عدد قرمز روی راس و Parent هر راس نیز با فلش مشخص شده است.




شکل ۱: پیمایش گراف هشت راسی با الگوریتم BFS

در هر مرحله از الگوریتم نیز، frontier و next به شکل زیر مشخص می شوند:

- frontier $= \{s\} \implies$ next $= \{a, x\}$
- frontier $= \{a, x\} \implies$ next $= \{z, d, c\}$
- frontier $= \{z, d, c\} \implies$ next $= \{f, v\}$
- frontier $= \{f, v\} \implies$ next $= \{\}$ (end of algorithm)

در این پیمایش، اگر راسی در هیچ یک از levelها قرار نگرفت، آن راس را در لایه $\infty$ قرار می دهیم. همچنین منظور از لایه iام یعنی با دقیقا i حرکت از راس به مبدا می رسیم.

## ۲.۱ درخت BFS یا درخت کوتاه ترین مسیر

با توجه به مسیر هایی که از هر راس به والدش در شکل بالا داریم (فلش ها) گراف جهت دار متناظر با گراف، به نام درخت BST به وجود می آید. در این درخت، یال های بدون جهت حذف می شوند و از هر راس کوتاه ترین مسیر به مبدا مشخص می شود.

اگر میخواستیم تمام مسیر های منتهی به مبدا را ذخیره کنیم حافظه ای به اندازه $\Theta(n^2)$ نیاز بود. اما در این گراف برای ذخیره سازی نیاز به حافظه ای از مرتبه n $(\Theta(n))$ نیاز داریم.
یال های بدون جهت که در درخت وجود ندارند یا بین رئوس یک لایه (مانند یال fv) یا بین لایه های متوالی (مانند یال fc) قرار دارند.

## ۲ الگوریتم جست و جوی عمق اول یا Depth First Search (DFS)

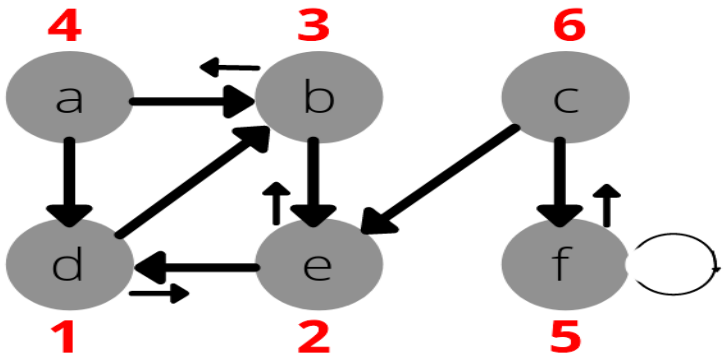
برای خروج از یک Maze رویکرد ما می تواند در پیش گرفتن یک مسیر با قرار دادن یک دست ثابت روی دیوار باشد. در این رویکرد بن بست ها را به صورت خودکار بر میگردیم و مسیر های جدید را بازدید میکنیم. همچنین درصورتی که از ورودی هزارتو وارد شده باشیم در loop نمی افتیم.

```
1.  def DFS (V, Adj):
2.      parent = []
3.      def visit (s)
4.          for v in Adj[s]:
5.              if v not in parent:
6.                  parent[v] = s
7.                  visit(v)
8.      for s in V:
9.          if s not in parent:
10.             parent[s] = None
11.             visit(s)
```

در این الگوریتم بر خلاف BFS راس مبدا نداریم.
حلقه for خط ۸ نیز باعث میشود مطمئن شویم همه رئوس بازدید شده اند.

## ۱.۲ پیمایش یک گراف ۶ راسی با الگوریتم DFS

پیمایش از راس a شروع شده است. parentها نیز با فلش مشخص شده اند و ترتیب اتمام الگوریتم برای رئوس با عدد قرمز رنگ مشخص شده است.

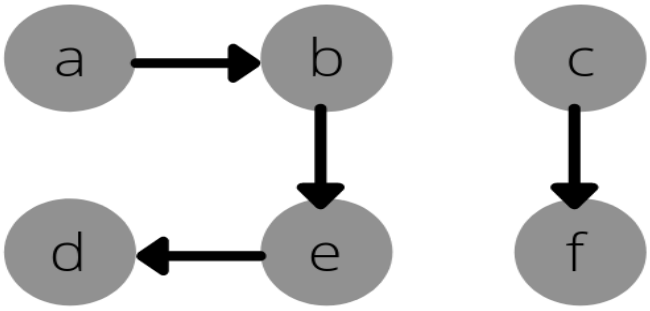



شکل ۲: پیمایش یک گراف ۶ راسی با الگوریتم DFS

در استک حافظه نیز به ترتیب از سر استک در مرحله اول visit(d)، visit(e)، visit(b) و visit(a) قرار دارند. سپس با فرض اینکه تابع visit برای راس c صدا زده شود، در مرحله دوم ترتیب به صورت visit(f) و visit(c) می باشد.

## ۲.۲ درخت **DFS**

مانند درخت BFS، مسیرهایی که در پیمایش DFS طی می کنیم را درخت DFS می نامیم. ممکن است این درخت همبند نباشد و چند مولفه همبند داشته باشد (جنگل باشد).

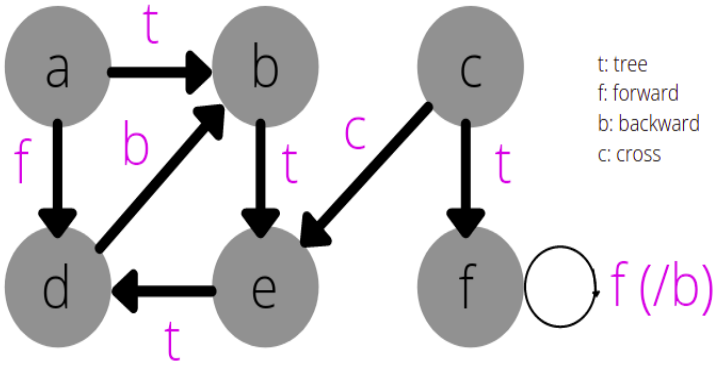



شکل ۳: درخت DFS گراف مثال بالا

### ۱.۲.۲ دسته بندی یالهای یک گراف جهت دار

- یالهای درخت DFS
- یالهای رو به جلو یا forward از ریشه به برگ ها و گره ها
- یالهای بازگشتی یا backward از گره ها و برگ ها به ریشه
- یالهای متقاطع یا cross یین دو شاخه از درخت

شکل صفحه بعد یالهای گراف ۶ راسی ذکر شده را نشان میدهد.




شکل ۴: یالهای گراف جهت دار ۶ راسی

### ۲.۲.۲ دسته بندی یالهای یک گراف بدون جهت

در گراف بدون جهت یالهای backward و backward یکی هستند و به آنها یال backward گفته می شود.
همچنین یال متقاطع نیز نداریم. زیرا اگر فرض کنیم که چنین یالی وجود دارد، امکان ندارد یک شاخه را تا ته پیمایش کنیم و یال متصل به شاخه دیگر یا همان یال کراس را پیمایش نکنیم. پس هر دو راس متصل به هم در یک شاخه قرار دارند.

## ۳.۲ زمان اجرای الگوریتم DFS

هر راس در این الگوریتم تنها یک بار بررسی می شود و تابع visit آن تنها یکبار فراخوانی می شود. هر تابع visit نیز برای یک راس به تعداد همسایه های هر راس زمان می برد تا اجرایش به اتمام برسد. پس در نتیجه زمان اجرای الگوریتم برابر $\Theta(n+e)$ میباشد.
(تابع visit خط سه از اردر e و حلقه خط هشت از اردر n)

# ۳ مقایسه DFS و BFS

BFS مانند یک صف به صورت FiFo رئوس را پیمایش میکند ولی DFS مانند استک و به صورت LiFo پیمایش را انجام میدهد.