# جزوه جلسه هجدهم داده ساختارها و الگوریتم

۷ آذر ۱۴۰۰

## فهرست مطالب

# ۱ گراف

هر گراف مجموعه ای از رئوس و یال ها می باشد و به جز این دو مورد، بقیه اطلاعات دور ریخته می شود. $G = (V, E)$ نشان دهنده گراف $G$ با مجموعه رئوس $V$ و مجموعه یال های $E$ است.
اگر گراف جهت دار باشد، هر یال یک زوج مرتب از ۲ راس $((u, v))$ می باشد که به معنی یک یال از راس $u$ به $v$ است. در گراف بدون جهت نیز هر یال، یک مجموعه دوتایی از راس هاست $(\{u, v\})$ که به معنی وجود اتصال بین ۲ راس $u$ و $v$ است.
در این درس، مباحث مرتبط با پیمایش گراف و بعد از آن، پیدا کردن کوتاه ترین مسیر بررسی می شود.

## ۱.۱ مثال های گراف

کاربرد های پیمایشی:

- پیدا کردن مسیری از یک راس به عنوان مبدا به راس مقصد
- بازدید از همه راس ها یا یال های گراف که از یک راس مبدا به نام S قابل دسترس است.

کاربرد های غیر پیمایشی:

- رنگ آمیزی گراف به نحوی که دو سر هر یال دارای به یک رنگ رنگ آمیزی نشده باشد.
- Matching
- درخت پوشای کمینه
- گراف مسطح

## ۲.۱ کاربردهای گراف

- خزش وب یا Web Crawling: الگوریتم گوگل برای شناسایی صفحات جدید وب اینگونه کار میکند که تمام لینک های موجود در صفحات شناخته شده را پیمایش می کند و در صورت پیدا کردن صفحه جدید که ناشناخته است، آنرا برای موتور جست و جو شناسایی می کند.
- شبکه های اجتماعی: پیمایش شبکه دوستی و پیشنهاد دادن دوستان جدید

- مسیریابی یا Navigation
- پیدا کردن مسیرهای هوایی مستقیم
- زباله روبی در زبان های برنامه نویسی (Garbage Collection): آبجکت هایی که به همدیگر رفرنس دارند، در صورتی که یک رفرنس حذف شود، زباله روب اشیا بعدی که به رفرنس حذف شده، خود رفرنس داشتند را زباله در نظر می گیرد.
- چک کردن مدل یا Model Checking: در علوم مختلف و در صنایع حساس (از آسانسور گرفته تا صنایع نظامی و هسته ای!) مدل های خاصی را طراحی می کنند تا عملکرد پروژه در حالت های مختلف بررسی شود. حالت های مختلف در این حالت، پیمایش و بررسی می شوند.
- حل برخی پازل ها مانند روبیک $2 \times 2 \times 2$: تمام حالت ها با Model Checking بررسی می شوند.
- شواهد دیجیتال: برای مثال حرکت های کارآگاهی! و دنبال کردن پیام ها و تماس ها

## ۳.۱ داده ساختار های مناسب برای ذخیره سازی گراف

فرض می کنیم رئوس گراف با اعداد ۰ تا n-1 نامگذاری شده اند.

### ۱.۳.۱ ماتریس مجاورت

یک ماتریس به ابعاد $n \times n$ به نام A که $A[i][j] = 1$ اگر و تنها اگر $(i,j) \in E$ یا $\{i,j\} \in E$
از مزایای این روش می توان به انجام اعمال جبرخطی، بررسی وجود یک یال و حذف یا اضافه کردن یک یال در $O(1)$ اشاره کرد.
در طرف دیگر زمانبر بودن استخراج لیست همسایه ها ($\Theta(n)$) و حافظه مورد نیاز به اندازه $\Theta(n^2)$ برای ذخیره سازی اطلاعات از معایب این روش هستند.

### ۲.۳.۱ لیست مجاورت

برای هر راس مانند $u$، لیست همسایه هایش یا $Adj(u)$ را نگه داری می کنیم.
در این شیوه، مجموع اندازه لیست ها برای گراف جهت دار از مرتبه $e$ و برای گراف بدون جهت از مرتبه $2e$ است. حافظه مصرفی در این روش نیز برابر $\Theta(n+e)$ می باشد. ($e$ تعداد یال ها و $n$ تعداد رئوس است)
در این روش با توجه به توضیحات فوق، مشکل حافظه و دسترسی به همسایه ها حل

شد. اضافه کردن یک یال نیز در $O(1)$ انجام می گیرد.
اما چک کردن و حذف و اضافه کردن یک یال دیگر در $O(1)$ قابل انجام نیست. (می توان به جای لیست پیوندی از جدول درهم سازی برای ذخیره سازی همسایه ها استفاده کرد که عملیات بررسی و حذف و اضافه در $O(1)$ انجام گیرد اما پیاده سازی با لیست پیوندی اصطلاحا good enough است.)
همسایه های u را میتوان به صورت یک آرایه از همسایه ها ($Adj[u]$) یا یک property در کلاس مربوط به راس ($u.neighbour$) در نظر گرفت.
همچنین نگه داری همسایه های هر راس میتواند صریح (از ابتدا همه لیست ها موجود باشند) یا ضمنی (لیست ها به هنگام نیاز ساخته شوند) باشد.
در ۹۰ درصد مواقع، گراف با این متد پیاده سازی می شود.

## ۴.۱ الگوریتم پیمایش جست و حوی سطح-اول گراف یا Breadth First Search

از یک راس در سطح صفر شروع به پیمایش می کنیم و همسایه های هر سطح، سطح بعدی را می سازند که هر سطح یک لایه (Layer) نام دارد.
در هر مرحله، یک مجموعه مرزی به نام frontier داریم که آخرین لایه پیمابش شده است و از روی لایه مرزی و همسایه هایی که تا کنون بازدید نشده اند، لایه بعدی یا next ساخته می شود ک مجموعه مرزی در مرحله بعدی است.

```
1.  def BFS(V, Adj, s)
2.      level = {s: 0}
3.      parent = {s: None}
4.      i = 1
5.      frontier = [s]
6.      while frontier:
7.          next = []
8.          for u in frontier:
9.              for v in Adj[u]:
10.                 if v not in level:
11.                     level[v] = i
12.                     parent[v] = u
13.                     next.append(v)
14.         frontier = next
15.         i += 1
```

### ۱.۴.۱ تحلیل زمانی الگوریتم **BFS**

هر راس حداکثر یک بار در مجموعه frontier قرار می گیرد. پس حلقه for خط هشتم گویا روی کل رئوس گراف است. برای هر راس نیز روی تمام همسایه هایش لوپ میزنیم که برابر با مجموع طول لیست های مجاورت است. پس شرط خط دهم نیز به اندازه تعداد یالها ($\mathrm{m}$ یا $2\mathrm{m}$) بررسی می شود و بلوک آن نیز برای هر راس یک بار انجام میشود.
با تفاسیر فوق این الگوریتم از مرتبه $\Theta(n+e)$ می باشد.
در نظر داریم که این الگوریتم برای مولفه های همبند، گراف را پیمایش میکند.