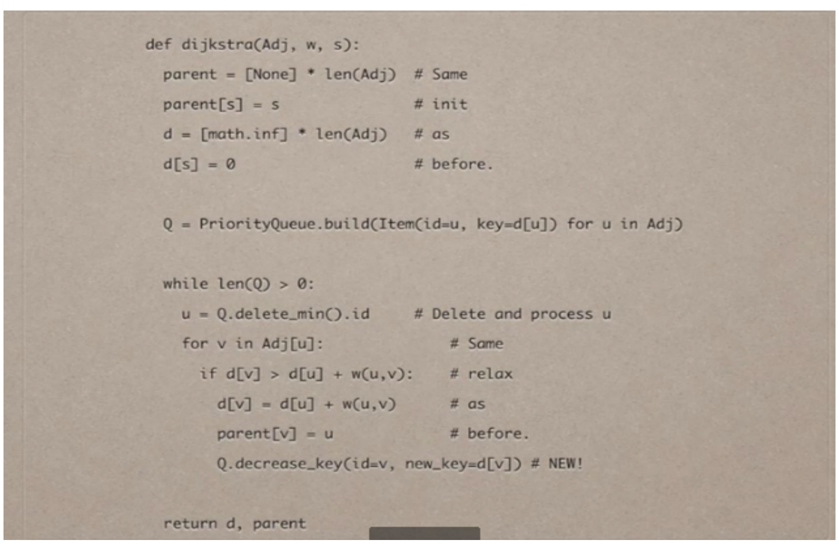
جزوه جلسه بیست و سوم داده ساختارها و الگوریتم

۲۱ آذر ۱۴۰۰

فهرست مطالب

۱ الگوریتم دکسترا برای حل مسئله SSSP

در این الگوریتم در گراف یال منفی نداریم. به همین دلیل زمان $\theta(ne)$ الگوریتم بلمن-فورد به $\Theta(e+nlogn)$ (در بهترین حالت پیاده سازی) می رسد.
ایده این الگوریتم در نظر گرفتن یک مجموعه مرزی که شامل رئوسی است که فاصله آنها از مبدا s حداکثر x می باشد و افزایش گام به گام x می باشد. رئوسی که درون مجموعه قرار دارند را relax شده در نظر می گیریم و با افزایش x، رئوس خارج از مجموعه مرزی را نیز relax می کنیم و با ورود همه رئوس به مجموعه مرزی، الگوریتم به پایان می رسد.
حالت پایه این الگوریتم زمانی است که $x = 0$ باشد و تنها مبدا در مجموعه مرزی قرار بگیرد. الگوریتم باید برای این حالت نیز درست کار کند. (فرض می کنیم که یال با وزن صفر نیز نداریم)
اگر وزن همه رئوس یک بود، دقیقا مسئله BFS را داشتیم؛ اما در این مسئله وزن یالها لزوما صحیح نیستند و عمل افزایش x باید با دقت و احتیاط انجام گیرد. به این صورت که افزایش زیاد x باعث ignore شدن بعضی رئوس و افزایش کم آن باعث عدم پایان یافتن الگوریتم در زمان معمول می شود.
بهترین گام افزایش x، فاصله رئوس بیرون مجموعه مرزی از مبدا منهای x است. در این حالت تنها event های مهم (رئوس جدید) پیمایش می شوند. به این منظور نیز در هر مرحله، فاصله مجموعه مرزی تا رئوس باقی مانده را به روز نگه می داریم و آن ها را در یک صف اولویت قرار می دهیم.
شبه کد این الگوریتم در تکه کد زیر آمده است: در گام اول مجموعه مرزی خالی است و



```
def dijkstra(Adj, w, s):
  parent = [None] * len(Adj)  # Same
  parent[s] = s               # init
  d = [math.inf] * len(Adj)   # as
  d[s] = 0                    # before.

  Q = PriorityQueue.build(Item(id=u, key=d[u]) for u in Adj)

  while len(Q) > 0:
    u = Q.delete_min().id     # Delete and process u
    for v in Adj[u]:              # Same
      if d[v] > d[u] + w(u,v):    # relax
        d[v] = d[u] + w(u,v)      # as
        parent[v] = u             # before.
        Q.decrease_key(id=v, new_key=d[v]) # NEW!

  return d, parent
```

شکل ۱: شبه کد الگوریتم دکسترا

با شروع از مبدا s، راس از صف حذف می شود و وارد مجموعه مرزی می شود. همچنین در این الگوریتم هر یال حداکثر ۲ بار و در اردر $\Theta(e)$ یال ها پیمایش می شوند.

۱.۱ پیاده سازی های مختلف الگوریتم دکسترا

۱.۱.۱ پیاده سازی صف اولویت با **binary heap**

اگر صف اولویت با binary heap پیاده سازی شود، در اینصورت زمان عملیات delete_min و decrease_key هردو در زمان $O(logn)$ انجام میشود و در نهایت به دلیل انجام n عمل delete_min و e عمل decrease_key، این الگوریتم در زمان $O((n+e)logn)$ انجام می شود.
در صورت وجود یال منفی، در هنگام relax کردن با پیمایش مکرر یال منفی، طول مسیر را کوتاه کرد. به همین دلیل وجود یال منفی در این الگوریتم منع شده است.
همچنین اگر گراف جهت دار باشد نیز در صورت وجود یال منفی، تضمینی بر درست بودن الگوریتم وجود ندارد.

۲.۱.۱ پیاده سازی الگوریتم در زمان $O(n^2)$

اگر تعداد یال ها زیاد باشد می توان بدون استفاده از صف اولویت نیز الگوریتم را پیاده سازی کرد. در این حالت با for زدن روی رئوس بیرون مجموعه و لیست همسایه ها می توان در زمان $On^2)$ الگوریتم را پیاده سازی کرد. در این حالت زمان delete_min برابر $O(n)$ و زمان decrease_min نیز برابر $O(1)$ است. پس در نهایت اردر زمانی الگوریتم برابر می شود با $O(n^2)$

۳.۱.۱ پیاده سازی با هرم فیبوناچی

هرم فیبوتاچی نیز یک داده ساختار ارائه شده برای interface هرم است. در این داده ساختار عملیات delete_min در زمان $O(logn)$ (به طور کلی به دلیل کران پایین مرتبط سازی، این عملیات در هر هرمی حداقل از اردر $logn$ می باشد) و decrease_key در زمان سرشکن $O(1)$ انجام می گیرد. پس زمان اجرای الگوریتم برابر می شود با:
$O(n * O(logn) + e * O(1)) = O(nlogn + e)$
توجه شود که این مدل پیاده سازی برای n های کوچک به دلیل ضریب بالای عملیات سرشکن مناسب نیست و استفاده از هرم دودویی توصیه می شود. ولی برای n های بزرگ استفاده از هرم فیبوتاچی بهتر است.

۲.۱ اثبات درستی الگوریتم دکسترا

با relax کردن یال ها، d راس ها همواره بیشتر یا مساوی فاصله واقعی آن ها از مبدا است. حال ادعا می کنیم وقتی یک راس به مجموعه مرزی اضافه می شود، d آن نهایی شده است. با اثبات این ادعا، درستی الگوریتم دکسترا نیز ثابت می شود.
با فرض خلف، فرض می کنیم راس w اولین راسی است که با اضافه شدن به مجموعه مرزی، فاصله آن از مبدا نهایی نشده است و صحیح نیست. کوتاه ترین مسیر از s به w را در گراف داده شده در نظر می گیریم. با حرکت از s به w، اولین یالی را در نظر بگیرید که از داخل مجموعه به خارج آن میرویم و اسم آن را (v, u) در نظر می گیریم. چون یال منفی نداریم پس فاصله واقعی u از مبدا کمتر از فاصله واقعی w از مبدا است. d[v] به دلیل قرار گرفتن درون مجموعه مرزی درست محاسبه شده است. همچنین به دلیل relax کردن یال (v, u)، d[u] نیز درست محاسبه شده است؛ پس $d[w] \geq d[u]$ همچنین اگر یال صفر نداشتیم، باید راس u به مجموعه مرزی اضافه می شد. همچنین در صورت وجود یال صفر نیز مشکلی برای الگوریتم به وجود نمی آید؛ زیرا در این صورت فرقی بین u و w به منطور اضافه کردن به مجموعه مرزی وجود ندارد.
$d[u] \geq d[w] \geq dis(s, w) \geq dis(s, u) = d[u]$
به دلیل برابر شدن ابتدا و انتهای نامساوی، پس حالت تساوی همه نامساوی ها رخ داده و در نتیجه: $d[w] = dis(s, w)$ پس فرض خلف نادرست است و اثبات به پایان می رسد.

۳.۱ اجرای هوشمندانه تر الگوریتم دکسترا!

در عمل برای رسیدن به راس مشخص t از مبدا s، شروع کردن از مبدا و بزرگ کردن مجموعه مرزی تا رسیدن به t از لحاظ زمانی مقرون به صرفه نیست. البته در بدترین حالت، برای مسیریابی بین دو نقطه کار بهتری نیز نمیتوان انجام داد. به جای این کار از مبدا در جهت یال ها و از مقصد (t) در خلاف جهت یال ها الگوریتم دکسترا را اجرا می کنیم تا به اولین راس مشترک در مجموعه های مرزی برسیم. کوتاه ترین مسیر لزوما از راس مشترک نمی گذرد و به همین دلیل باید تمام یال های موجود بین ۲ مجموعه را بررسی کنیم تا کوتاه ترین مسیر پیدا شود