مسیر ریاضی دان عاشق

پویا آقاحسینی ۹۵۱۳۰۰۶

میخواهیم مسیری پیدا کنیم که از هر راس دقیقا یک بار رد شود، این یک مسیر همیلتونی است و در مسیر همیلتونی نمیتوانیم از یک راس دوبار رد شد.
برای اینکه این مسیر را پیدا کنیم با Topological Sort گراف را مرتب میکنیم . همچنین طبق تعریف آن داریم:

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering.

لذا در ترتیبی که بدست آورده ایم راس ها به ترتیبی هستند که برای هر یالی که در گراف است راس های قبل و بعد آن برای همه Topological Sort ها به ترتیب آمده اند.
پس از اینکه گراف را سورت کردیم به ترتیب راس ها را پیمایش میکنیم و چک میکنیم که برای هر دو راس متوالی باید یالی جهت دار بین آن ها باشد.

در واقع اگر مسیر همیلتونی موجود باشد آنگاه در توپولوژیکال سورت آن بین هر دو راس متوالی یال جهتدار داریم چون در صورتی که یالی نباشد با توجه به اینکه هر راس یکبار در توپولوژیکال سورت ظاهر شده و اگر هنگام پیمایش نتوانیم آن راس را ببینیم در ادامه نیز دیده نمیشود و به آن نیز برنمیگردیم، بنابراین تنها حالت این است که حتما آن را همینجا رویت کنیم و یال وجود داشته باشد.
در صورتی که توپولوژیکال سورت را پیمایش کنیم و همه رئوس متوالی به هم یال داشته باشند، در واقع همه رئوس را یکبار دیده ایم و مسیر همیلتونی را پیمایش کرده ایم.
از آنجایی که یک گراف میتواند چند توپولوژیکال سورت مختلف داشته باشد باید ثابت کنیم که هر مسیر همیلتونی یک توپولوژیکال سورت یکتا میدهد :
فرض خلف میکنیم که مسیر همیلتونی دو سورت مختلف دارد، دو توپولوژیکال سورت مختلف در حالتی پیش می آید که ترتیب رئوس جابجا باشد و این در حالتی است که یک راس دوشاخه داشته باشیم و اولویتی بین فرزندان آن نباشد که در این صورت در مسیر یکی را نمیبینیم و نمیتوانیم به آن برگردیم پس اصلا نمیتوانیم مسیر همیلتونی داشته باشیم. حالت دیگر وقتی است که دو راس به هم متصل نباشند یا در مولفه های مختلفی باشند که در این صورت نیز با فرض داشتن مسیر همیلتونی در تناقض است.

```
topologicalSort: \\Same as DFS except we push each vertex when returning.
visited[v] = True
for w in adj[v]:
     if w was not visited: \\ recursively call the function
     topologicalSort(w)
when going back, push the vertex into a stack
stack.insert(0,v)

\\ after the topological sort we have it in a stack
\\and we can traverse the nodes by poping :
flag=0
```

```
v=stack.pop()
while stack is not empty:
     w=stack.pop()
     if w in adj(v):
          \\ if we use the matrix this check will be in O(1)
          \\ O.W it's O(degree(v))
         queue.add(v) \\ just to save the path
         v=w
         continue
     else
         flag=1
         break
if flag is 1
     we dont have a hamiltonian path
else
     queue is the path
```

پیچیدگی زمانی : توپولوژیکال سورت $O(V+E)$ زمان میبرد و پیمایش استک $O(V)$ و چک کردن هم $O(degree(V))=O(E)$ ، پس کل الگوریتم $O(V+E)$ زمان میبرد.

تحلیل حافظه: وابسته به ساختمان داده گراف در صورتی که Adjacency List باشد یا Adjacency Matrix به ترتیب $O(V+E)$ و $O(V^2)$ و استک هم به اندازه تعداد رئوس که $O(V)$ است حافظه میبرد.