Algorithm Design
Project #3

Hana esfandiar
Mohammad Afshari

ایده کلی الگوریتم:

می‌خواهیم کوتاه ترین مسیری را بیابیم که حمید و محمد باید طی کنند تا متوجه شوند که در کدام روستا بوده اند .

بدین منظور، از الگوریتم **Dijkstra** برای پیدا کردن کوتاه ترین مسیر استفاده می‌کنیم.

شهرها و روستاها را راس های گراف و جاده های بین این شهر ها و روستا ها را یال های گراف در نظر می‌گیریم.

الگوریتم **Dijkstra** برای یافتن کوتاه ترین مسیر از یک رأس تا رئوس دیگر به این صورت عمل می‌کند که یک راس را به عنوان راس شروع یا منبع(source) در نظر میگیرد(در واقع این راس به عنوان ورودی، به همراه گراف،به الگوریتم داده می‌شود.) و کوتاه ترین مسیر از این راس شروع را تا تمام رئوس دیگر پیدا می کند.

دو مجموعه تعریف می‌کنیم. یکی از مجموعه ها شامل رئوسی هست که در کوتاه ترین مسیر قرار دارند و مجموعه دیگه، رئوسی هستند که هنوز در مجموعه کوتاه ترین مسیر قرار نگرفته اند. مراحل اجرای الگوریتم به این صورت است که در هر مرحله، از مجموعه رئوسی که در کوتاه ترین مسیر قرار ندارند، راسی را که کمترین فاصله را تا راس منبع(source) دارد را انتخاب می کنیم. این راس را در مجموعه رئوسی که در کوتاه ترین مسیر قرار دارند، قرار می‌دهیم.

حال باید فاصله‌ی تمام رئوسی که با راس انتخاب شده همسایه هستند را به روز رسانی کنیم و این به روز رسانی به این صورت است که اگر مجموع وزن یالی که بین راس انتخاب شده و راس همسایه‌اش است و فاصله‌ی راس انتخاب شده از راس منبع، کمتر از اندازه مقدار فاصله (distance value) راس همسایه‌اش باشد، مقدار فاصله راس همسایه را به مقدار مجموع گفته شده تغییر می‌دهیم. این روند را تا جایی ادامه می‌دهیم تا تمام رئوس پیمایش شوند و در مجموعه رئوسی که در کوتاه ترین مسیر قرار دارند، قرار گیرند.

حال در مساله‌ی انتخاب کوتاه ترین مسیر توسط حمید و محمد، باید الگوریتم Dijkstra را بر روی گراف داده شده به عنوان ورودی، دوبار اجرا کنیم. یک بار روستای A را راس منبع در نظر می‌گیریم و بار دیگر روستای B را. هر کدام از این دو راس منبع که مسیر کوتاه تری را به عنوان خروجی بدهند، نتیجه میشود که آن شخص حدس درست درباره روستایی که در آن بودند زده و اگر اندازه مسیر برای هر دو راس منبع به یک اندازه بود،نتیجه میگیریم که روستایی که محمد و حمید در آن بودند،قابل تشخیص نیست.

بررسی کد:

تابع **get_input** ورودی ها را از کاربر می‌گیرد و در متغیرهای مورد نظر ذخیره می‌کند. (خط ۹-۲۳)

تابع **dijkstra** الگوریتم Dijkstra را بر روی رئوس منبع A و B اجرا می‌کند.(خط ۳۹-۶۸) در خط ۴۲، مجموعه ای تعریف شده که فاصله‌ی هر راس را تا راس منبع در آن ذخیره می‌کنیم. در خط ۴۹ مجموعه‌ای دیگر داریم که مقادیر true or false را به راس های مختلف اختصاص می‌دهد که true نشان دهنده‌ی این است که راس با این مقدار، جزو رئوسی که در کوتاه ترین مسیر قرار دارند، هست.
حال در حلقه for خط ۵۳، به ترتیب رئوسی که کوتاه ترین فاصله را تا راس منبع دارند انتخاب می‌کنیم و در هر مرحله مقادیر فاصله این رئوس تا رئوس همسایه‌ش را محاسبه، مقایسه و سپس در صورت لزوم به روز رسانی می‌کنیم. این محاسبه و مقایسه در خط ۶۲ تا ۶۷ انجام می‌شود.
در خط ۸۴-۸۷ بررسی میکنیم که از بین کوتاه ترین مسیر هایی که روستای A با شهر و روستا های دیگر دارد و در مجموعه **distance_A** ذخیره شده، فاصله ی روستای A تا نزدیک ترین شهر کدام است. یعنی کدام یک از آن راس ها، شهر هست و کوتاه ترین فاصله روستای A تا آن شهر را پیدا کرده و در متغیر **closest_A** ذخیره میکنیم. همین روند را برای روستای B انجام می‌دهیم. در خط ۹۵-۹۹ بررسی میکنیم که آیا مقادیر **closest_A** و **closest_B** برابر هستند یا خیر!اگر برابر باشند، روستایی که محمد و حمید در آن بودند،قابل تشخیص نیست اما در غیر این صورت، روستایی که مقدار **closest_x** آن کمتر است، روستایی است که محمد و حمید در آن بودند.

پیچیدگی زمانی :

در تابع **dijkstra** دو حلقه for تو در تو داریم که هر حلقه n بار تکرار میشود(روی تمام رئوس) که پیچیدگی زمانی این تابع $O(n^2)$ می‌شود.

حلقه for در تابع **get_input** و **minimun_distance** نیز n بار تکرار می‌شود .

در خط ۸۴ و ۹۰ نیز دو حلقه داریم که آنها نیز n بار تکرار می‌شوند. پیچیدگی زمانی اجرای این حلقه ها، هرکدام O(n) است.

محاسبه پیچیدگی زمانی:

محاسبه پیچیدگی زمانی تابع dijkstra:

اگر فرض کنیم که عبارت

```
distance[v] = distance[u] + matrix[u][v]
```

عملیات پایه است آنگاه داریم:

$$T(n) = \sum_{c=1}^{n} \sum_{v=1}^{n} 1 = \sum_{c=1}^{n} n = n^2$$

$$T(n) \in O(n^2)$$

در نهایت داریم :

$$T'(n) = O(n) + O(n) + O(n) + O(n) + O(n^2)$$

$$T'(n) \in O(n^2)$$