1- حل این سوال با این فرض است که در گراف اصلی، یال‌ها جهت‌دار نباشند. ابتدا تمام یال‌ها را پیمایش کرده و هر یال e که بین دو راس u و v قرار دارد را با دو یال جهت‌دار uv با وزن v و vu با وزن u جایگزین می‌کنیم. حال به کمک الگوریتم Dijkstra می‌توانیم طول کوتاه‌ترین مسیر از s به رئوس دیگر را بدست آوریم. در انتها باید وزن راس s را نیز به مقدار بدست آمده اضافه کنیم.
با توجه به اینکه در ابتدا یک بار یال‌ها را پیمایش می‌کنیم و در ادامه از الگوریتم Dijkstra استفاده می‌کنیم، مرتبه زمانی الگوریتم $O(|E| + |V|\, log|V|)$ خواهد بود.
برای اثبات درستی این راه کافی‌ست به این موضوع دقت کنیم که وقتی از راس a به راس b می‌رویم، به جای اینکه هزینه را در انتهای مسیر (در راس b) پرداخت کنیم می‌توانیم همین هزینه را در خود مسیر (یال ab) پرداخت کنیم که در این حالت مقدار جواب نهایی تغییری نخواهد کرد.
در صورتی که یال‌ها در گراف اصلی جهت‌دار باشد، به جای اینکه در گراف جدید هم از u به v و هم از v به u یال داشته باشیم، یال را فقط در جهتی قرار می‌دهیم که در گراف اصلی وجود دارد. یعنی برای مثال اگر در گراف اصلی یک یال از u به v داشته باشیم، در گراف جدید یک یال از u به v و با وزن راس v تعریف می‌کنیم.

2- یال‌ها را در m مرحله و به صورت جداگانه به گراف اضافه می‌کنیم. در هر مرحله که یالی مثل uv را به گراف اضافه می‌کنیم، با اضافه کردن این یال به MST، دقیقا یک دور تشکیل می‌شود. اگر یال uv یال با بیشترین وزن در این دور باشد، T همچنان یک MST برای گراف است. در غیر این صورت می‌توان درخت پوشای دیگری پیدا کرد که هزینه آن کمتر از T باشد. این درخت که آن را T′ می‌نامیم، همان درخت T است با این تفاوت که یال با بیشترین وزن در این دور حذف شده و یال uv به T اضافه شده است. این درخت T′ یک MST برای گراف مورد نظر است.
برای انجام این کار زمانی که یال uv را به گراف اضافه می‌کنیم، به کمک الگوریتم DFS مسیر یکتا از راس u به راس v را در درخت T پیدا می‌کنیم. در همین حین، طول بزرگ‌ترین یالی که در این مسیر قرار دارد را نیز پیدا می‌کنیم، این یال را x در نظر می‌گیریم. اگر w(uv) > w(x) باشد، درخت T نیازی به تغییر ندارد و MST باقی می‌ماند. در غیر این صورت یال x را از درخت حذف می‌کنیم و یال uv را به درخت اضافه می‌کنیم. درخت بدست آمده MST است.
می‌دانیم که مرتبه زمانی الگوریتم DFS برابر با $O(|E| + |V|)$ است و با توجه به این مورد که در درخت تعداد یال‌ها برابر با $O(|V|)$ است، مرتبه زمانی انجام این الگوریتم به ازای هر یال برابر با $O(n)$ خواهد بود و با تکرار این عمل به تعداد m بار، مرتبه زمانی این الگوریتم با اضافه کردن m یال به گراف برابر با $O(nm)$ خواهد بود.

3- می‌توانیم هر شهر را یک راس و هر جاده را یک یال در نظر بگیریم. در این صورت با استفاده از الگوریتم Floyd-Warshall می‌توانیم کوتاه‌ترین فاصله بین هر دو شهر را محاسبه کنیم، این فاصله بین دو راس u و v را با dist(u, v) نمایش می‌دهیم. حال به ازای هر شهر (راس) u، بیشترین مقدار dist(u, j) را محاسبه می‌کنیم و آن را در MSP(u) ذخیره می‌کنیم.

$MSP(u) = max\big(dist(u,j)\big) \qquad (j \in V)$

پاسخ مطلوب سوال راسی مانند v است که MSP(v) دارای کمترین مقدار بین تمام مقادیر MSP باشد. به عبارت دیگر می‌توان گفت:

$answer = argmin(MSP)$

می‌دانیم که مرتبه زمانی الگوریتم Floyd-Warshall برابر با $\mathcal{O}(|V|^3)$ است که |V| تعداد کل شهرها را نشان می‌دهد. از طرفی با توجه به اینکه برای محاسبه MSP(u) از دو حلقه تو در تو استفاده می‌کنیم، مرتبه زمانی انجام این بخش $\mathcal{O}(|V|^2)$ خواهد بود. در نتیجه مرتبه زمانی انجام کل الگوریتم برابر با $\mathcal{O}(|V|^3)$ است.

با توجه به اینکه با استفاده از الگوریتم فوق توانستیم راسی را پیدا کنیم که بیشترین مقدار کوتاه‌ترین مسیر آن تا بقیه راس‌ها کمینه باشد و این مورد که توانستیم کشور را با یک گراف مدل کنیم، به مطلوب مسئله می‌رسیم.

همچنین اگر مرتبه زمانی الگوریتم Dijkstra را $\mathcal{O}(|E| + |V|\,log|V|)$ در نظر بگیریم، می‌توانیم به جای الگوریتم Floyd-Warshall از الگوریتم Dijkstra استفاده کنیم. در این روش به ازای هر راس u، طول کوتاه‌ترین مسیر از راس u تا همه راس‌های دیگر را پیدا می‌کنیم و در نهایت راسی را به عنوان پاسخ انتخاب می‌کنیم که بیشترین مقدار طول کمترین مسیر بدست آمده از آن راس کمینه باشد (مشابه روش قبلی). در این حالت مرتبه زمانی انجام الگوریتم برابر با $\mathcal{O}(|V||E| + |V|^2\,log|V|)$ خواهد شد که در صورتی که تعداد یال‌ها (جاده‌ها) برابر با $\mathcal{O}(|V|)$ باشد، مرتبه زمانی انجام الگوریتم $\mathcal{O}(|V|^2\,log|V|)$ خواهد شد.

4- آ) مسئله را با گراف مدل می‌کنیم به این صورت که شهرها را راس و جاده‌های یک طرفه را یال جهت‌دار در نظر می‌گیریم. همچنین برای سادگی توضیح الگوریتم، جاده‌هایی که به تازگی آسفالت شده‌اند را یال رنگی در نظر می‌گیریم. ابتدا تمام یال‌های رنگی را کنار می‌گذاریم. در هر مرحله یک جفت از یال‌های رنگی را انتخاب کرده و به گرافی که یال‌های رنگی از آن حذف شده‌اند، اضافه می‌کنیم. اگر یال‌های رنگی را مجموعه C در نظر بگیریم، واضح است که تعداد حالات انجام این کار (انتخاب 2 یال رنگی) برابر با $\mathcal{O}\left(\binom{|C|}{2}\right)$ خواهد بود. پس از اضافه کردن 2 یال رنگی انتخاب شده، به کمک الگوریتم Dijkstra می‌توانیم کوتاه‌ترین مسیر از راس s به بقیه رئوس را پیدا کنیم. بدیهی است که در این حالت حداکثر 2 یال رنگی در مسیر به مقصد هر یک از رئوس قرار می‌گیرد.

با در نظر گرفتن این مورد که مسئله را با یک گراف مدل کردیم و پاسخ مطلوب مسئله را در گراف بدست آوردیم، این پاسخ برای مسئله اصلی هم صدق می‌کند.

با توجه به اینکه الگوریتم Dijkstra در این مسئله $\mathcal{O}(|C|^2)$ بار اجرا می‌شود، مرتبه زمانی انجام الگوریتم برابر با $\mathcal{O}(|C|^2|E| + |C|^2|V|\,log|V|)$ است.

ب) اگر شرط عبور از حداکثر 2 جاده آسفالت شده در نظر گرفته نشود، کافی‌ست الگوریتم Dijkstra را بر روی گراف معادل با نقشه شهرها اجرا کنیم.

```
function Dijkstra(V, w, s) do
    set S = []
    declare dist[V.size]
    declare heap

    for v in V do
        dist[v] = infinity
    end
    dist[s] = 0

    for v in V do
        heap.push(v, dist[v])
    end

    while heap.size > 0 do
        u = heap.pop()
        S.push(u)
        for v in u.adjacents do
            if dist[v] > dist[u] + w[u][v] then
                dist[v] = dist[u] + w[u][v]
                heap.decrease(v, dist[v])
            end
        end

    return dist
end
```

5- آ) همانطور که در صورت سوال اشاره شده، می‌توانیم شهر را با یک گراف ساده با n راس و m یال مدل کنیم. در این گراف نقاطی که دزدها قرار دارند و نقاطی که ایستگاه‌های پلیس قرار دارند، هر کدام با یک راس نشان داده می‌شوند. با توجه به اینکه میخواهیم کوتاه‌ترین مسیر تعدادی راس را از نزدیک‌ترین ایستگاه پلیس محاسبه کنیم، می‌توانیم از یک راس جدید کمک بگیریم. راس N را به گراف اضافه می‌کنیم به طوری که از این راس فقط به راس‌هایی که در آن‌ها ایستگاه پلیس قرار دارد یک یال به وزن 0 قرار می‌دهیم. حال در صورتی که کوتاه‌ترین فاصله این راس از هر یک از راس‌هایی که دزدها در آن قرار دارند را پیدا کنیم، می‌توان گفت کوتاه‌ترین فاصله هر یک از دزدها از نزدیک‌ترین ایستگاه پلیس را پیدا کردیم. در این بخش از الگوریتم Dijkstra و برنامه‌ریزی پویا استفاده می‌کنیم. ابتدا آرایه dist به طول n را تعریف می‌کنیم به طوری که dist[i] مقدار کوتاه‌ترین فاصله تا راس N یا به عبارت دیگر مقدار کوتاه‌ترین فاصله تا نزدیک‌ترین ایستگاه پلیس را نشان می‌دهد. برای چاپ پاسخ هم آرایه parent را به طول n تعریف می‌کنیم به طوری که parent[i] نشان‌دهنده نزدیک‌ترین ایستگاه پلیس تا راس i باشد. برای آپدیت کردن راس v از طریق راس u به این طریق عمل می‌کنیم:

```
If (dist[u]+w[u][v] < dist[v])
        dist[v] = dist[u] + w[u][v]
        parent[v] = parent[u]
```

همچنین نیاز به ذکر است که مقدار parent برای ایستگاه‌های پلیس برابر با خودشان (همان ایستگاه پلیس) است. پس از اجرای الگوریتم Dijkstra از راس N، مقادیر مطلوب همان $parent[T_i]$ها هستند. پیچیدگی زمانی مسئله نیز برابر با پیچیدگی زمانی الگوریتم Dijkstra خواهد بود که با مقدار $\mathcal{O}(m + n\,log\,n)$ برابر است (یا در نوع دیگری از پیاده‌سازی برابر با مقدار $\mathcal{O}(m\,log\,n)$ است).

ب) برای این بخش نیز از الگوریتم Dijkstra و برنامه‌ریزی پویا استفاده می‌کنیم. الگوریتم Dijkstra را از راس D آغاز می‌کنیم. آرایه dist به طول n را تعریف می‌کنیم به طوری که نشان‌دهنده کمترین فاصله راس i از راس D باشد.(در صورتی که در مسیری موتورسیکلت ویژه هم قرار داشته باشد، کاهش مسیر پلیس در این آرایه لحاظ می‌شود). همچنین یک آرایه دیگر به نام L از جنس bool و به طول n تعریف می‌کنیم به طوری که L[i] در صورتی true است که در راس i و یا یکی از جدهایش حداقل یک موتور سیکلت وجود داشته باشد و در غیر این صورت false است. زمان آپدیت کردن راس v از راس u به این صورت عمل می‌کنیم:

if (u $\in$ L)

$\quad$ $\mathrm{L[u] = true}$

if (L[u])

$\quad$ if (dist[v] > dist[u] + $\frac{1}{2}$w[u][v])

$\quad\quad$ dist[v] = dist[u] + $\frac{1}{2}$w[u][v]

$\quad\quad$ L[v] = true

else

$\quad$ if (dist[v] > dist[u] + w[u][v])

$\quad\quad$ dist[v] = dist[u] + w[u][v]

$\quad\quad$ L[v] = false

در نهایت پس از اجرای این الگوریتم به ازای هر دزد، اگر حتی مقدار dist برای یک پلیس کمتر یا مساوی مقدار dist برای دزد مربوطه باشد، برای این دزد عبارت can’t escape چاپ می‌شود. در غیر این صورت مقدار dist دزد چاپ می‌شود. با توجه به اینکه حل سوال از طریق الگوریتم Dijkstra صورت گرفته است، پیچیدگی زمانی مسئله نیز $\mathcal{O}(m + n\,log\,n)$ است.

6- آ) ابتدا روستاها را با یک گراف مدل می‌کنیم به صورتی که هر روستا را یک راس در نظر می‌گیریم. این گراف را یک گراف کامل در نظر می‌گیریم، یعنی بین هر دو راسی یک یال وجود دارد. همچنین وزن یال‌ها را برابر با فاصله 2 روستای معادل در نظر می‌گیریم. در نهایت با اجرای الگوریتم Prim بر روی گراف ذکر شده می‌توانیم یک MST برای این گراف پیدا کنیم. در ادامه با بازگرداندن گراف به روستاها، بین هر 2 روستایی که یال قرار دارد، یک سیم برق قرار می‌دهیم. با توجه به اینکه درخت بدست آمده کمینه بوده است، هزینه سیم‌کشی برق بین روستاها هم کمینه خواهد بود.
مرتبه زمانی انجام این الگوریتم همان مرتبه زمانی انجام الگوریتم Prim خواهد بود که برابر با $\mathcal{O}(n^2)$ است.

ب) برای اثبات این مورد از برهان خلف کمک می‌گیریم. فرض می‌کنیم طول بزرگ‌ترین یالی که در MST مورد قبل بدست آوردیم کمینه نیست. در این صورت می‌توان گفت درخت پوشایی مثل T وجود دارد که طول بزرگ‌ترین یال آن کمتر از طول بزرگ‌ترین یال MST است. در این صورت بزرگ‌ترین یال MST را حذف می‌کنیم. درخت به دو مولفه همبند تقسیم می‌شود. این دو مولفه را در درخت T در نظر می‌گیریم. با توجه به پوشا بودن T مطمئن هستیم که یالی بین این دو مولفه در T وجود دارد و با توجه به فرض خلف می‌دانیم وزن این یال کمتر از وزن یال حذف شده در MST است. این یال را به MST اضافه می‌کنیم. بدیهی است که با اضافه کردن این یال دو مولفه همبند به همدیگر متصل شده و یک درخت پوشا را تشکیل می‌دهند. با توجه به اینکه از MST یک یال را حذف کردیم و یک یال دیگر با وزن کمتر را جایگزین کردیم، فرض کمینه بودن MST اولیه نقض می‌شود و در نتیجه می‌توان گفت طول بزرگ‌ترین یال موجود در MST همواره کمینه است. با توجه به اینکه مسئله اصلی با این گراف مدل شده است، می‌توان نتیجه گرفت طول بلندترین سیم برق کشیده شده بین روستاها نیز همواره کمینه است.