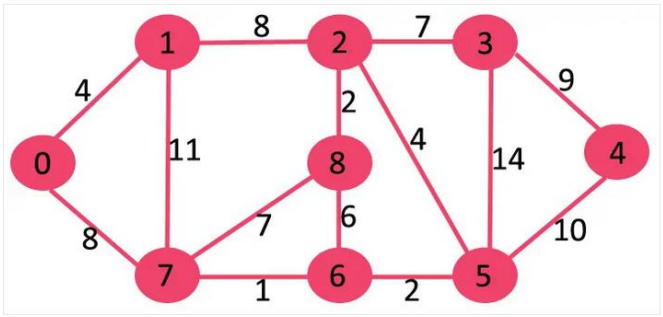
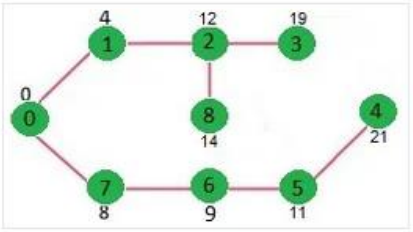
سینا به مسافرت رفته است. تعدادی شهر و هزینه سفر بین این شهرها را مشخص کرده است. برای این کار از گراف استفاده می کند. (برای راحتی به جای نام شهرها یک عدد را قرار داده‌ایم.) مانند شکل زیر.



شکل ۱

در این گراف ۹ شهر و هزینه سفر بین دو شهر (در صورت وجود مسیر) مشخص شده است. هر راس یک شهر و خطوط بین راسها نشانده هزینه سفر بین دو شهر هست. در شکل۱، هزینه سفر از شهر ۵ به شهر ۶ مقدار ۲ هست یا هزینه سفر از شهر ۶ به شهر ۴، ۱۲ هست که این مسیر از شهر ۵ می گذرد.

سینا دنبال راهی می گردد تا در هر شهری هست کم هزینه‌ترین مسیرها از این شهر تا هر شهر دیگری را بیابد. به طور مثال اگر در شهر ۰ هست کم هزینه ترین سفر به هر شهرهای ۰و۱و۲و۳و...۸ به ترتیب ۰و۴و۱۲و۱۹و۲۱و۱۱و۹و۸و۱۴ می باشد که در شکل ۲ مسیرها و هزینه سفر از شهر ۰ به هر شهر، بر روی آن شهر، نوشته شده است.



شکل ۲

برنامه‌ای بنویسید که یک گراف را به عنوان ورودی در قالب یک آرایه دو بعدی و همچنین شماره راس مبدا را دریافت کند و برای خروجی کم هزینه ترین سفر از راس مبدا تا بقیه رئوس را چاپ کند. (هزینه ها به ترتیب شماره راس چاپ شوند به طور مثال هزینه رفتن به شهر ۰، هزینه رفتن به شهر ۱و ...)

توضیح ورودی:

گراف شکل ۱ با یک آرایه دو بعدی قابل نمایش است. در این آرایه سطرها و ستون‌ها نشان‌دهنده راس‌ها هستند، و در صورتی که بین راس i و j مسیری با هزینه w وجود داشته باشد، درایه [j][i] و درایه [i][j]

در این آرایه برابر با w می‌شود: g[i][j] = g[j][i] = w. همچنین اگر بین دو راس i و j هیچ مسیری وجود نداشته باشد، درایه مربوطه برابر با صفر مقداردهی می‌شود: g[i][j] = g[j][i] = 0. بنابراین g یک ماتریس مربعی متقارن است، و درایه‌های روی قطر اصلی آن صفر هستند.

0	4	0	0	0	0	0	8	0
4	0	8	0	0	0	0	11	0
0	8	0	7	0	4	0	0	2
0	0	7	0	9	14	0	0	0
0	0	0	9	0	10	0	0	0
0	0	4	14	10	0	0	0	0
0	0	0	0	0	2	0	1	6
8	11	0	0	0	0	1	0	7
0	0	2	0	0	0	6	7	0

هدف : پیدا کردن طول کوتاهترین مسیرها از راس مبدا s به دیگر راس‌های گراف

متغیرهای اولیه:

- گراف ورودی که با آرایه ۲بعدی g نمایش داده می‌شود.
- آرایه dist: یک آرایه یک بعدی که تعداد عناصر آن برابر است با تعداد رئوس، و فاصله هر راس از راس مبدا را در خود ذخیره می‌کند. به این معنی که اگر فاصله راس i از راس صفر برابر با x باشد، داریم: dist[i] = x
- آرایه visited برای مشخص شدن راس‌های بررسی شده (این آرایه را می‌توانید از نوع int یا bool تعریف کنید به ازای هر راس i در صورتی که راس i در الگوریتم بررسی شده باشد مقدار visited[i] =1 و در غیر این صورت visited[i] = 0 مقداردهی می‌شود.

مقداردهی اولیه:

- آرایه dist: برای راس s (برای این گراف s=0) ، مقدار dist[0]= 0 و برای دیگر راس‌ها در ابتدای کار مقدار

dist(v) = ∞ مقداردهی می‌شود. (در این برنامه برای نشان دادن ∞ از یک عدد بزرگ برای مثال ۱۰۰۰ استفاده کنید. dist[i] = 1000)

- آرایه visited: این آرایه در ابتدا با صفر مقداردهی می‌شود. چراکه در ابتدای کار هیچکدام از راس‌ها بررسی نشده‌اند.

الگوریتم:

- ✓ تا زمانی که هنوز راس v در گراف وجود دارد که بررسی نشده (visite[v] = 0) دقت کنید شرط پایان الگوریتم این است که تمام مقادیر آرایه visited برابر با ۱ شود.
- ✓ از بین رئوس v که visited[v] = 0 راس v0 را که مقدار dist(v0) مینیمم است، انتخاب کنید.

برای مثال اگر آرایه visited و آرایه dist به ترتیب مقادیر زیر را داشته باشند:

- ✓ `visited: [1 0 0 1 1 0 0 1 1]`
- ✓ `dist: [0 3 7 10 12 6 8 15]`

- ✓ در این صورت با توجه به آرایه visited، راس‌های ۱،۲،۵،۶ بررسی نشده‌اند. برای هرکدام از این راس‌ها مقدار آرایه dist به شرح زیر است:
  - ✓ `dist[1]=3, dist[2]=7, dist[5]=6, dist[6]=8`

- ✓ با توجه به این مقادیر، مقدار مینیمم برابر است با dist[1]=3 ، بنابراین راس v0 = 1 انتخاب می‌شود.
- ✓ راس v0 را به لیست رئوس بررسی شده اضافه کنید. (`visited[v0] = 1`)
- ✓ برای هر راس u که مجاور v0 است، مقدار dist[u] را به این صورت به‌روزرسانی نمایید:
- ✓ `dist[u] = min(dist[v0]+g[u][v0], dist[u])`

- ✓ به این معنی که اگر مقدار `dist[v0]+g[u][v0]` مقدار قبلی `dist[u]` کمتر بود، مقدار `dist[u]` مقدار جدید `dist[v0]+g[u][v0]` آپدیت شود، در غیر این صورت نیاز به آپدیت مقدار `dist[u]` نیست.
- ✓ به مرحله گام ۱ بازگرد. (ابتدای حلقه)
- ✓ در نهایت آرایه dist را که نشان‌دهنده طول کوتاهترین مسیر از راس s به دیگر رئوس گراف است در خروجی چاپ نمایید.

نکته:

۱. برای راحتی کار می توانید مانند کد زیر در زمان پیاده سازی گراف را مقداردهی اولیه کنید تا برای هر اجرا نیاز نباشد که اطلاعات گراف را از ورودی بگیرید.. (اما وقتی کد بر روی سایت قرار می گیرد باید از ورودی اطلاعات را بگیرد.)

```
int g[9][9] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},
               {4, 0, 8, 0, 0, 0, 0, 11, 0},
                {0, 8, 0, 7, 0, 4, 0, 0, 2},
                {0, 0, 7, 0, 9, 14, 0, 0, 0},
                {0, 0, 0, 9, 0, 10, 0, 0, 0},
                {0, 0, 4, 14, 10, 0, 2, 0, 0},
                {0, 0, 0, 0, 0, 2, 0, 1, 6},
                {8, 11, 0, 0, 0, 0, 1, 0, 7},
                {0, 0, 2, 0, 0, 0, 6, 7, 0}
               };
```

۲. زمانی که پیاده سازی تمام شد ابعاد گراف را به صورت متغیر تعریف کنید. می توانید در ابتدای برنامه یک متغیر صحیح ثابت با اندازه ۵۰ برای ماکزیمم تعداد راس‌ها تعریف کنید. آرایه دو بعدی با همین ابعاد ایجاد می‌شود. از کاربر تعداد راسها (v<51) گرفته می شود. و از این به بعد v سطر و ستون از آرایه پردازش می شود.

به عنوان مثال تابع گرفتن گراف به این صورت هست:

```
int const n = 50;
void getGraph( int g[n][n], int v)
{
     {
    for (int i=0; i<v; i++)
    {
       for (int j=0; j<v; j++)
           cin >> g[i][j] ;
    }
}
```

در خط انتهایی ورودی، یک عدد می‌آید که بیانگر شماره راس ورودی است.

۳. گروهها به صورت حداکثر سه نفره تشکیل شود. همه اعضای گروه کد را آپلود کنند و نام همه اعضای گروه در بالای کد نوشته شود. در زمان تقلب یابی اگر نام اعضای کدهای مشابه یکسان نباشد، نمره صفر تعلق می‌گیرد.