توضیحات پروژه:

برای پیاده سازی این پروژه ابتدا از الگوریتم dijkstra و همچنین single source shortest path استفاده کردیم اما کاربرد آنها زمانی است که گره مبدا مشخص است و هدف minimize کردن فاصله این گره تا همه گره هاست. اما هدف ما در این پروژه minimize کردن هزینه گراف و به عبارتی کمینه کردن فاصله بین هر دو زوج مرتب انتخابی است بنابراین برای پیاده سازی این پروژه از الگوریتم فلوید وارشال با کمی تغییر استفاده می‌کنیم.

همانطور که میدانیم الگوریتم فلوید وارشال برای پیدا کردن کمترین هزینه برای جابه جایی بین نود های یک گراف به کمک یال واسطه است.

تغییرات ما در این الگوریتم به این صورت بوده است که ابتدا برای ماتریس اولیه الگوریتم را به صورت کامل اجرا میکنیم سپس در هر ۵ ثانیه پس از اضافه شدن نود جدید الگوریتم را فقط برای نود جدید اجرا میکنیم این کار باعث می‌شود مرتبه زمانی الگوریتم از O(n^3) به O(n^2) کاهش پیدا کند و مساله زودتر حل شود.

برای پیاده سازی ابتدا در تابع مربوط در این پروژه الگوریتم فلوید وارشال را نوشته ایم با این تفاوت که تابع یک ورودی id نیز دریافت میکند .این ورودی مشخص میکند که الگوریتم باید برای همه نود ها برسی را انجام دهد یا فقط برای نود جدید که اضافه شده است.

به این صورت که اگر id صفر باشد الگوریتم تماماً کامل اجرا میشود و اگر بخواهیم فقط برای نود آخر فراخوانی شود باید id را برابر با طول ماتریس منهای یک قرار دهیم.

```
void cost_minimizer(vector<Node*>& nodesList, int id){
    // It's your turn!
    int v_size = nodesList.size();
    for(int k = v_size-1; k >= 0; k--){
        for(int i = v_size-1; i >= id; i--){
            for(int j = v_size-1; j >= 0; j--){
                if(nodesList[i]->nodesCosts[k] != INF && nodesList[k]->nodesCosts[j] != INF){
                    if (nodesList[i]->nodesCosts[k] + nodesList[k]->nodesCosts[j] < nodesList[i]->nodesCosts[j]){
                        nodesList[i]->nodesCosts[j] = nodesList[i]->nodesCosts[k] + nodesList[k]->nodesCosts[j];
                        nodesList[j]->nodesCosts[i] = nodesList[i]->nodesCosts[k] + nodesList[k]->nodesCosts[j];
                    }
                }
            }
        }
    }
}
```

الگوریتم فلوید وارشال پیاده سازی شده .در تابع مورد نظر.

```
if(id-2 == n)
    cost_minimizer(nodesList, 0);
else
    cost_minimizer(nodesList, id-1);
```

شرطی برای اجرای الگوریتم به صورت کامل یا فقط برای نود آخر.