۱. "علوم‌مهندسی آباد" شهریست شامل $n \leq 10^5$ میدان و m خیابان مستقیم میان بعضی از جفت-میدان‌ها (گراف ساده). بابک که به تازگی شهردار شده است قصد دارد تعدادی از میدان‌های شهر را گل‌کاری کند به طوری که هر میدان یا گل‌کاری شده باشد یا به میدانی که گل‌کاری شده است، خیابان داشته باشد. بنابر پاره‌ای از مشکلات، او حداکثر می‌تواند $\left\lfloor\frac{n}{2}\right\rfloor$ تا از میدان‌ها را انتخاب کند. به او در این امر کمک کنید و الگوریتمی ارائه دهید که میدان‌های انتخابی را خروجی دهد. همچنین آن را از لحاظ زمانی تحلیل کنید.

پاسخ:

ابتدا اجازه دهید یک bfs روی راس دلخواه v گراف داده شده بزنیم و فاصله همه رئوس را از آن محاسبه کنیم. درواقع ما به اندازه خود فاصله‌ها نیازی نداریم بلکه به زوجیت آنها کار داریم. در گام بعدی تمامی رئوسی که فاصله‌شان از v زوج است را مجموعه Z و تمامی رئوسی که فاصله‌شان از v فرد است را مجموعه F می‌نامیم. حال جواب برابر است با مجموعه‌ای که اندازه‌اش کمتر است. اولا که واضح است که حداقل اندازه یکی از این مجموعه ها، از عدد $\left\lfloor\frac{n}{2}\right\rfloor$ فراتر نمی‌رود. دوما چون ما زوجیت فواصل را درنظر می‌گیریم، هر راس از هر مجموعه‌ای به حداقل یک راس از مجموعه روبه‌رو متصل است. $O(n+m)$

```
#include <bits/stdc++.h>
using namespace std;
const int INF = 1e9; int n, m;
vector<int> d; vector<vector<int>> g;
void bfs(int s) {
	d = vector<int>(n, INF); d[s] = 0;
	queue<int> q; q.push(s);
	while (!q.empty()) {
		int v = q.front(); q.pop();
		for (auto to : g[v]) {
			if (d[to] == INF) {
				d[to] = d[v] + 1; q.push(to);
			}
		}
	}
}
int main() {
	int t; cin >> t;
	for (int tc = 0; tc < t; ++tc){
		cin >> n >> m; g = vector<vector<int>>(n);
		for (int i = 0; i < m; ++i) {
			int x, y;
			cin >> x >> y;
			--x, --y;
			g[x].push_back(y);
			g[y].push_back(x);
		}

		bfs(0);
		vector<int> even, odd;
		for (int i = 0; i < n; ++i) {
			if (d[i] & 1) odd.push_back(i);
			else even.push_back(i);
		}

		if (even.size() < odd.size()) {
			cout << even.size() << endl;
			for (auto v : even) cout << v + 1 << " ";
		} else {
			cout << odd.size() << endl;
			for (auto v : odd) cout << v + 1 << " ";
		}
		cout << endl;
	}
	return 0;
}
```

۲. یک گراف n راسی و m یالی داریم. روی هر یال آن یک عدد اعشاری وجود دارد. الگوریتمی ارائه دهید که مشخص کند آیا در این گراف دوری وجود دارد که حاصل ضرب تمامی یال‌هایش از 1 بزرگتر باشد؟ $(nm \leq 10^6)$

پاسخ:

ایده حل الگوریتم، چیزی شبیه به الگوریتم $bellman\ ford$ خواهد بود ($O(nm)$). اما باید یک سری تفاوت را در نظر گرفت. این‌بار می‌خواهیم برای هر راس v_i گشتی را از راسی دلخواه مثل راس v_1 پیدا کنیم که حاصل‌ضرب یال‌های داخل آن گشت بیشینه باشند. فرض کنید این اعداد را در آرایه $zarb$ ذخیره کنیم. در ابتدا $zarb(i)$ به ازای هر راس i صفر خواهد بود. حال مانند الگوریتم بلمن فورد عمل می‌کنیم (در شبه کد تعداد رئوس n است و تعداد یالها m. هر یال سه مولفه دارد که دوتای اول نقاط پایانی آن یال هستند و مولفه سوم وزن آن را ذخیره می‌کند):

```
for (int i=0; i<n-1; i++){
    for (int j=0; j<m; j++){
        int vj= get<0>(edges[j]);
        int uj= get<1>(edges[j]);
        long double weight= get<2>(edges[j]);
        zarb[uj]= max(zarb[uj], zarb[vj]*weight);
    }
}
```

حال یکبار دیگر از یال‌ها استفاده می‌کنیم تا ببینیم می‌شود آرایه ضرب را برای راسی آپدیت کرد یا خیر. اگر این اتفاق افتاد، مشابه پیدا کردن دور منفی در الگوریتم بلمن فورد خواهد بود (چرا؟). در این صورت گراف ورودی شامل دوری خواهد بود که ضرب وزن یالهایش بیشتر از 1 خواهد بود و در غیر این صورت دوری با این ویژگی وجود ندارد.

```
bool res= false;
for (int j=0; j<m; j++){
    int vj= get<0>(edges[j]);
    int uj= get<1>(edges[j]);
    long double weight= get<2>(edges[j]);
    if (distance[uj]<distance[vj]*weight){
        res= true;
        break;
    }
}
```

مقدار نهایی در متغیر res ذخیره خواهد شد. اگر دوری وجود داشته باشد که ضرب وزن یالهایش بیشتر از 1 بود مقدار res برابر با $true$ و در غیر این صورت برابر با $false$ خواهد بود.

۳. ماتریس D کوتاه‌ترین مسیر بین هر دو راس در گراف وزن‌دار G را نگهداری می‌کند به طوری که $D[u,v]$ کوتاه‌ترین مسیر از راس u به راس v است. فرض کنید وزن یک یال در گراف G از w_e به $\acute{w}_e$ تغییر پیدا کند. الگوریتمی از مرتبه زمانی $O(n^2)$ طراحی کنید که مقادیر ماتریس D را به‌روزرسانی کند.

پاسخ:

فرض کنید دو سر یال e رئوس v و u باشند. چون فقط هزینه یکی از یال‌ها کاهش یافته است، برای محاسبه فاصله جدید کافی است که برای هر دو گره دلخواه فاصله آنها تا دو سر یال uv محاسبه کنیم و با وزن کنونی آنها مقایسه کنیم. هرکدام که کوچکتر بود را به عنوان مسیر کمینه بین دو گره معرفی می‌کنیم. به این ترتیب داریم:

```
for (int i=0; i<N; i++){
    for (int j=0; j<N; j++){
        d[i][j] = min(d[i][j], d[i][u]+w(uv)+d[v][j]);
        d[i][j] = min(d[i][j], d[j][u]+w(uv)+d[v][i]);

```

```
24.            }
25.        }
```

هزینه الگوریتم $O(n^2)$ است.

اثبات: برای دو راس دلخواه i و j:

الف) اگر کوتاه‌ترین مسیر بین این دو راس دلخواه از یال uv عبور نکند چون فقط وزن همین یال عوض شده است پس مقدار کوتاه‌ترین مسیر با مقدار در گراف قبلی برابر است. حال باید ثابت کنیم مسیر $\mathrm{i} \rightarrow \mathrm{uv} \rightarrow \mathrm{j}$ یا عکس آن مقدار کمتری از کوتاه‌ترین مسیر ندارد. اگر مقدار عبارت

$$\mathrm{d[i][u]} + \mathrm{w(uv)} + \mathrm{d[v][j]} < \mathrm{shortest\ path(i,j,G')}$$

برقرار باشد، در این صورت مسیر $\mathrm{i} \rightarrow \mathrm{uv} \rightarrow \mathrm{j}$ کوتاه‌ترین خواهد بود که با فرض تناقض دارد که کوتاه‌ترین مسیر شامل uv نیست.

ب) اگر کوتاه‌ترین مسیر بین این دو راس دلخواه از یال uv عبور کند، با فرض عدم وجود دور منفی خواهیم داشت که مسیرهای $\mathrm{u} \rightarrow \mathrm{i}$ و $\mathrm{j} \rightarrow \mathrm{v}$ مسیر ساده هستند و دوری ندارد. دو مسیر $\mathrm{u} \rightarrow \mathrm{i}$ و $\mathrm{j} \rightarrow \mathrm{v}$ کوتاه‌ترین مسیر بین جفت گره‌های نام برده هستند. این موضوع را فقط برای $\mathrm{u} \rightarrow \mathrm{i}$ ثابت می‌کنیم و برای دیگری نیز مشابه است. فرض کنید مسیر دیگری مثل P' باشد که از مسیر $\mathrm{u} \rightarrow \mathrm{i}$ که آن را P می‌نامیم و در گراف اولیه است کوتاه‌تر باشد. P' از uv عبور نمی‌کند چون اگر می‌کرد می‌شد با عدم عبور از آن اندازه مسیر را کوتاه‌تر کرد. بنابراین اندازه P و P' برابر است. حال کافی است نشان دهیم که $\mathrm{d[i][j]} > \mathrm{d[i][u]} + \mathrm{W(uv)} + \mathrm{d[v][j]}$ است. $\mathrm{d[i][j]}$ مقدار فاصله کمینه این دو راس در گراف قبلی است. از طرفی ثابت کردیم مقدار فاصله $\mathrm{u} \rightarrow \mathrm{i}$ و $\mathrm{j} \rightarrow \mathrm{v}$ در دو گراف برابر و کمینه است؛ پس تنها تفاوت دو مسیر اندازه یال uv است که در گراف دوم کمتر است پس این مسیر کمینه است.

۴. یک گراف وزن دار که که در آن وزن هر یال ۱ یا ۲ است را در نظر بگیرید، کوتاه ترین مسیر را از یک راس منبع داده شده s تا راس مقصد t پیدا کنید به طوری که پیچیدگی زمانی آن از $O(V+E)$ باشد.

پاسخ:

راه ساده تر آن از الگوریتم $dijkstra$ استفاده می‌کند که ما را به $O(E+VLogV)$ می‌رساند. اما در حالت $O(V+E)$ ایده استفاده از الگوریتم BFS است. نکته مهمی که باید در مورد BFS درنظر گرفت این است که مسیری که در BFS پیدا می‌شود همیشه روی کمترین تعداد یال‌ها تمرکز می‌کند پس اگر وزن یال‌ها برابر بود، BFS جواب می‌دهد. برای این مسئله ما می‌توانیم گراف را تغییر دهیم به طوریکه همه یال‌هایی که وزنشان دو هست را به دو یال با وزن یک تبدیل می‌کنیم به این نحو که یک راس جدید می‌سازیم که بین این دو یال قرار دارد. (یعنی هر یال به طول 2 می‌شود یک مسیر به طول ۲) حال می‌توانیم روی گراف جدید BFS بزنیم.

توجه کنید در هر بار تبدیل کردن یک یال به طول ۲ به یک مسیر، برای گراف جدید یک راس جدید ساخته می‌شود. پس حداکثر پیچیدگی زمانی از $O(V+E+2*E) = O(V+E)$ خواهد بود.

۵. فرض کنید T یک زیر درخت فراگیر کمینه از G و $\acute{T}$ زیردرخت فراگیر دیگر از G باشد هر حرکت یک یال $\acute{T}$ از را با یک یال از T جایگزین می‌کند. الگوریتمی ارائه دهید که با دنباله ای از حرکات، $\acute{T}$ را به T تبدیل کند با این شرط که با هر تغییری که انجام می‌دهیم همچنان درخت، فراگیر باشد و مجموع وزن یالهای آن هیچوقت بیشتر نشود.

پاسخ:

ایده الگوریتم جذاب است. اولا که مثلا اگر ما یک گراف H داشته باشیم و یک یال uv به آن اضافه کنیم و یک یال wxای از آن حذف کنیم تا به گراف $\acute{H}$ برسیم بدیهی است که به صورت برعکس هم می‌توان $\acute{H}$ را به H تبدیل کنیم.
حال T را به ورژن متناسب با درخت کمینه حاصل از اجرای $kruskal$ روی G (مثلا نام آن K است) تبدیل می‌کنیم.
بدین گونه که هر بار کوچک‌ترین یالی که در K قرار دارد را به T اضافه می‌کنیم. حال یک دور تشکیل می‌شود.

قطعا در این دور یالی وجود دارد که از یال جدید اضافه شده ما بزرگتر یا مساوی با آن است و در K نیامده. (چرا که ما هر بار کوچکترین را انتخاب می‌کنیم. متناسب با ایده از پیش اثبات شده الگوریتم $kruskal$) پس ما آن یال را حذف می‌کنیم. انقدر این کار را ادامه می‌دهیم تا T به K تبدیل می‌شود.
همین کار را برای تبدیل $\acute{T}$ به K انجام می‌دهیم. حال برای تبدیل $\acute{T}$ به T ابتدا آن را به K تبدیل می‌کنیم و دقیقا به ترتیب از آخر به اول تغییرات T به K را به صورت برعکس انجام می‌دهیم تا تبدیل $\acute{T}$ به T با ویژگی مدنظر اعمال شود.