Pintando con distancias

Autor del problema: Félix Moreno Peñarrubia (Miembro del comité científico)

Resumen del enunciado

Dados m enteros $a_1, \ldots, a_m$, ver si se puede pintar todo grafo de n vértices siguiendo un proceso que empieza en un vértice y va pintando los vértices a distancia a_i de alguno ya pintado.

Explicación de la solución

- Los vértices a distancias que sean sumas de varios valores de a_i se acaban pintando.
- Calculamos el menor valor d que no sea suma con programación dinámica.
- Si $n < 2d \Longrightarrow$ el radio de cualquier grafo de n vértices es $< d \Longrightarrow$ la respuesta es SÍ.
- $n \geq 2d \Longrightarrow$ la respuesta es NO, el ciclo de longitud $2d$ (con algunos vértices duplicados) es un contraejemplo.

Solución en C++

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <set>
#include <queue>
#include <complex>
#include <sstream>

using namespace std;
typedef long long int ll;
typedef vector <int> vi;
typedef vector <vi> vvi;
typedef vector <vvi> vvvi;
typedef vector <vvvi> vvvvi;
typedef pair <int, int> ii;
typedef vector <ii> vii;
typedef vector <vii> vvii;
typedef vector <vvii> vvvii;

void Floyd_Warshall(int n, vvi& dist){
    for (int k = 0; k < n; ++k){
        for (int i = 0; i < n; ++i){
            for (int j = 0; j < n; ++j){
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}
```

```
bool correct(int pos, vvi& dist, vi& V){
	int n = dist.size();
	vi col(n, 0);

	col[pos] = 1;

	for (int x: V){
		vi nextcol = col;

		for (int i = 0; i < n; ++i){
			if (not col[i]) continue;

			for (int j = 0; j < n; ++j){
				if (dist[i][j] == x){
					nextcol[j] = 1;
				}
			}
		}

		col = nextcol;
	}

	for (int x: col) {
		if (not x) return false;
	}

	return true;
}

bool valid(int n, vvi& G, vi& V){
	if (n != G.size()) return false;

	vvi dist(n, vi(n, 1e9));

	for (int i = 0; i < n; ++i){
		dist[i][i] = 0;

		for (int j : G[i]){
			dist[i][j] = dist[j][i] = 1;
		}
	}

	Floyd_Warshall(n, dist);

	for (int i = 0; i < n; ++i){
		for (int j = 0; j < n; ++j){
			if (dist[i][j] == 1e9) return false;
		}
	}

	for (int i = 0; i < n; ++i){
		if (correct(i, dist, V)) return false;
	}
```

```
        return true;

}

void build_cycle(int n, int m){
        vvi edges;
        if(m == 2) {
                for(int i=0; i < n; ++i) {
                        for(int j=i+1; j < n; ++j) {
                                edges.push_back({i, j});
                        }
                }
        }
        else {
                for (int i = 0; i < m; ++i){
                        edges.push_back({i, (i + 1) % m});
                }

                for (int i = m; i < n; ++i){
                        edges.push_back({i, (i + 1) % m});
                        edges.push_back({i, (m + i - 1) % m});
                }
        }
        cout << edges.size() << '\n';
        for (vi& edge: edges) cout << edge[0] << ' ' << edge[1] << '\n';
}

void try_path(int n, vi& V){
        vvi G(n);

        for (int i = 0; i < n - 1; ++i) {
                G[i].push_back(i+1);
        }

        if (not valid(n, G, V)){
                cout << "SI" << '\n';
                return;
        }

        cout << "NO" << '\n';
        cout << n - 1 << '\n';

        for (int i = 0; i < n - 1; ++i){
                cout << i << ' ' << i + 1 << '\n';
        }
}

void solve(){
        int n, m;
        cin >> n >> m;

        vi V(m);
        for (int i = 0; i < m; ++i) cin >> V[i];
```

```
        vvi dp(m+1, vi(n+1, 0));
        dp[0][0] = 1;

        for (int i = 0; i < m; ++i){
                for (int j = 0; j <= n; ++j){
                        if (dp[i][j]){
                                dp[i+1][j] = 1;
                                if (j + V[i] <= n) dp[i+1][j+V[i]] = 1;
                        }
                }
        }

        int maximum = -1;
        for (int i = 1; i <= n and maximum == -1; ++i){
                if (not dp[m][i]) maximum = i;
        }

        if (maximum == -1){
                cout << "SI" << '\n';
                return;
        }

        if (n >= 2 * maximum){
                cout << "NO" << '\n';
                build_cycle(n, maximum * 2);
                return;
        }

        cout << "SI" << '\n';
}

int main() {
        int T;
        cin >> T;

        while(T--) {
                solve();
        }
}
```

Autor de la solución: Félix Moreno Peñarrubia (Miembro del comité científico)