Neka je *G* neusmjereni graf, čiji su čvorovi označeni podacima tipa string. Neka je graf prikazan kao mapa koja svakom čvoru pridružuje listu ~~svih~~ njegovih susjednih čvorova, ali će svaki brid biti unesen samo jednom (pogledati prvu Dodatnu napomenu), . Možete pretpostaviti da u listi neće biti ponavljanja i da neće biti vrhova koji su povezani sami sa sobom.

Napišite funkciju prototipa
```
stack<string> susjedni (string n, map<string, list<string> > G)
```
koja prima čvor *n* i graf *G*, te vraća stog čvorova susjednih čvoru *n* u grafu *G*. Povratni stog ne smije sadržavati duplikate.

Napišite funkciju prototipa
```
queue<string> dostupni (string n, map<string, list<string> > G)
```
koja prima čvor *n* i graf *G*, te vraća red svih čvorova do kojih se može doći iz čvora *n* nekim putem u grafu *G*. Povratni red ne smije sadržavati duplikate.

Napišite funkciju prototipa
```
list<string> dostupniN (set<string> N, map<string, list<string> > G)
```
koja prima skup čvorova *N* i graf *G*, te vraća listu svih čvorova do kojih se može doći nekim putem u grafu *G*, tako da početni čvor puta bude u skupu *N*. Povratna lista ne smije sadržavati duplikate.

Napišite funkciju prototipa
```
vector<set<string> > povezani (map<string, list<string> > G)
```
koja prima graf *G*, te vraća sve komponente povezanosti grafa *G*, tj. u svakoj komponenti su vrhovi iz kojih se može međusobno doći do svih ostalih vrhova te komponente nekim putem u grafu *G*. Npr. ako je graf povezan (iz svakog vrha možemo nekako doći u svaki drugi vrh), funkcija vraća jedan skup koji sadrži sve vrhove grafa, a ako graf nema niti jedan brid, funkcija vraća vektor jednočlanih skupova.

Opće napomene:
- Možete pretpostaviti da graf neće imati više od 100 vrhova.
- Elementi u listi/redu/stogu/vektoru mogu biti navedeni u bilo kojem redoslijedu.
- Deklaracije funkcijâ koje napišete stavite u datoteku **graf.h**, a njihove implementacije u datoteku **graf.cpp**.

Ako neku od funkcijâ uopće ne znate napisati, nemojte je spominjati niti u .h niti u .cpp datoteci (nešto bodova se može dobiti i ako ne napišete sve funkcije).
- Funkcije i njihovi argumenti se moraju zvati točno kako piše u zadacima. Također, njihovi povratni tipovi i tipovi argumenata moraju izgledati onako kako je opisano u zadacima.
- Niti jedna funkcija ne smije ništa učitavati s tipkovnice ili neke datoteke, niti išta ispisivati na ekran ili u neku datoteku.
- Naravno, za provjeru radi li implementacija prije nego što je pošaljete, preporučuje se da je testirate pomoću nekog klijentskog programa. No taj klijentski program ne šaljete!
- Za sva pitanja vezana uz ovu zadaću javite se asistentu Petričeviću na **vpetrice@math.hr**

Dodatne napomene:
- U mapi *G*, svaki brid će biti naveden samo jedan put. Tako npr. ukoliko su *a* i *b* povezani, ili će G[„a“] sadržavati „b“ ili obrnuto.
- Mapa *G* nam predstavlja neusmjeren graf, pa funkcija *dostupni* (a isto tako i druge) treba vratiti prazan skup ukoliko vrh *n* nema bridova, a u protivnom treba vratiti sve vrhove za koje postoji *šetnja* do vrha *n*, tj. neka i vrh *n* bude u povratnom rezultatu (budući da se u njega tada može doći šetnjom duljine 2) (tj. treba vratiti komponentu povezanosti kojoj pripada *n*).

Primjer programa:
```
#include <iostream>
#include <string>
#include <stack>
#include <queue>
#include <list>
#include <vector>
#include <map>
#include <set>
```

```
using namespace std;

#include "graf.h"

int main() {
	map<string, list<string> > G;

	stack<string> ss = susjedni("a", G);
	while (!ss.empty()) {
		cout << ss.top() << endl;
		ss.pop();
	} // nista se ne ispise

	queue<string> qs = dostupni("a", G);
	while (!qs.empty()) {
		cout << qs.front() << endl;
		qs.pop();
	} // nista se ne ispise

	G["a"].push_back("b");
	G["c"].push_back("a");

	ss = susjedni("a", G);
	while (!ss.empty()) {
		cout << ss.top() << endl;
		ss.pop();
	} // ispisu se i b i c;
	cout << endl;

	qs = dostupni("a", G);
	while (!qs.empty()) {
		cout << qs.front() << " ";
		qs.pop();
	}
	cout << endl; // a b c

	G["f"].push_back("b");
	qs = dostupni("a", G);
	while (!qs.empty()) {
		cout << qs.front() << " ";
		qs.pop();
	}
	cout << endl; // a b c f

	{
		set<string> ss;
		G["d"];
		G["e"].push_back("g");
		ss.insert("c");
		list<string> ls = dostupniN(ss, G);
		for (list<string>::iterator i = ls.begin(); i != ls.end(); ++i)
			cout << *i << " ";
		cout << endl; // a b c f


		ss.insert("d");
		ls = dostupniN(ss, G);
		for (list<string>::iterator i = ls.begin(); i != ls.end(); ++i)
			cout << *i << " ";
		cout << endl; // a b c f
```

```
		ss.insert("g");
		ls = dostupniN(ss, G);
		for (list<string>::iterator i = ls.begin(); i != ls.end(); ++i)
			cout << *i << " ";
		cout << endl; // a b c e f g
	}

	vector<set<string> > vss = povezani(G);
	for (vector<set<string> >::iterator i = vss.begin(); i != vss.end(); ++i) {
		for (set<string>::iterator j = i->begin(); j != i->end(); ++j)
			cout << *j << " ";
		cout << endl;
	}
	// a b c f
	// d
	// e g

	return 0;
}
```