Neka je *G* neusmjereni graf, čiji su čvorovi označeni podacima tipa int. Neka je graf prikazan kao skup uređenih parova čvorova koji su povezani. Možete pretpostaviti da u neće biti ponavljanja (npr. neće biti par *a,b* i *b,a*) i da neće biti vrhova koji su povezani sami sa sobom.

Napišite funkciju prototipa
`queue<int> susjedni (int n, set<pair<int, int> > G)`
koja prima čvor *n* i graf *G*, te vraća red čvorova susjednih čvoru *n* u grafu *G*. Povratni red ne smije sadržavati duplikate.

Napišite funkciju prototipa
`stack<int> dostupni (int n, set<pair<int, int> > G)`
koja prima čvor *n* i graf *G*, te vraća stog svih čvorova do kojih se može doći iz čvora *n* nekim putem u grafu *G*. Povratni stog ne smije sadržavati duplikate.

Napišite funkciju prototipa
`vector<int> dostupniN (list<int> L, set<pair<int, int> > G)`
koja prima listu čvorova *L* i graf *G*, te vraća vektor svih čvorova do kojih se može doći nekim putem u grafu *G*, tako da početni čvor puta bude u listi *L*. Povratni vektor ne smije sadržavati duplikate, a možete pretpostaviti da i lista *L* neće imati duplikata.

Napišite funkciju prototipa
`bool povezan (set<pair<int, int> > G)`
koja prima graf *G*, te vraća true, ako je graf povezan, tj. u se iz svakog vrha grafa *G* može doći u svaki drugi vrh nekim putem, a inače vraća false.

Opće napomene:
- Možete pretpostaviti da graf neće imati više od 100 vrhova.
- Elementi u vektoru/redu/stogu 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:
- Ukoliko graf *G* nema niti jedan brid, smatra se da je povezan.
- Skup *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 <stack>
#include <queue>
#include <list>
#include <vector>
#include <map>
#include <set>


using namespace std;

#include "graf.h"
```

```
int main() {
	set<pair<int, int> > G;

	queue<int> qi = susjedni(1, G);
	while (!qi.empty()) {
		cout << qi.front() << endl;
		qi.pop();
	} // nista se ne ispise

	stack<int> si = dostupni(1, G);
	while (!si.empty()) {
		cout << si.top() << endl;
		si.pop();
	} // nista se ne ispise

	G.insert(make_pair(1, 2));
	G.insert(make_pair(3, 1));

	qi = susjedni(1, G);
	while (!qi.empty()) {
		cout << qi.front() << endl;
		qi.pop();
	} // ispisu se i 2 i 3;
	cout << endl;

	si = dostupni(1, G);
	while (!si.empty()) {
		cout << si.top() << " ";
		si.pop();
	}
	cout << endl; // 3 2 1

	G.insert(make_pair(100,2));
	si = dostupni(1, G);
	while (!si.empty()) {
		cout << si.top() << " ";
		si.pop();
	}
	cout << endl; // 100 3 2 1

	if (povezan(G))cout << "povezan" << endl; // ispise se

	{
		list<int> li;
		li.push_back(15);
		vector<int> vi = dostupniN(li, G);
		for (vector<int>::iterator i = vi.begin(); i != vi.end(); ++i)
			cout << *i << " ";
		cout << endl; // prazna linija

		G.insert(make_pair(50, 15));
		vi = dostupniN(li, G);
		for (vector<int>::iterator i = vi.begin(); i != vi.end(); ++i)
			cout << *i << " ";
		cout << endl; // 15 50

		if (povezan(G))cout << "povezan" << endl; // ne ispise se

		li.push_back(1);
		vi = dostupniN(li, G);
		for (vector<int>::iterator i = vi.begin(); i != vi.end(); ++i)
			cout << *i << " ";
```

```
		cout << endl; // 1 2 3 15 50 100

		li.pop_back();
		G.insert(make_pair(15, 1));
		vi = dostupniN(li, G);
		for (vector<int>::iterator i = vi.begin(); i != vi.end(); ++i)
			cout << *i << " ";
		cout << endl; // 1 2 3 15 50 100
	}

	if (povezan(G))cout << "povezan" << endl; // ispise se

	return 0;
}
```