Neka je *G* neusmjereni graf, čiji su čvorovi označeni uređenima parovima char, int. Neka je graf prikazan kao mapa koja svakom čvoru pridružuje skup njemu susjednih čvorova (pogledati prvu Dodatnu napomenu). Možete pretpostaviti da neće biti vrhova koji su povezani sami sa sobom i da neće biti vrhova koji nisu povezani s nikim.

Napišite funkciju prototipa

```
vector<pair<char, int> > susjedni (pair<char, int> n,
     map<pair<char, int>, set<pair<char, int> > > G)
```

koja prima čvor *n* i graf *G*, te vraća vektor čvorova susjednih čvoru *n* u grafu *G*. Povratni vektor ne smije sadržavati duplikate.

Napišite funkciju prototipa

```
list<pair<char, int> > dostupni (pair<char, int> n,
     map<pair<char, int>, set<pair<char, int> > > G)
```

koja prima čvor *n* i graf *G*, te vraća listu svih čvorova do kojih se može doći iz čvora *n* nekim putem u grafu *G*. Povratna lista ne smije sadržavati duplikate.

Napišite funkciju prototipa

```
stack<pair<char, int> > dostupniN (queue<pair<char, int> > Q,
     map<pair<char, int>, set<pair<char, int> > > G)
```

koja prima red *Q* (u kojem možete pretpostaviti da neće biti duplikata) i graf *G*, te vraća stog svih čvorova do kojih se može doći nekim putem u grafu *G*, tako da početni čvor puta bude u redu *Q*. Povratni stog ne smije sadržavati duplikate.

Napišite funkciju prototipa

```
set<char> povezani (map<pair<char, int>, set<pair<char, int> > > G)
```

koja prima graf *G*, te vraća skup svih početnih slova vrhova grafa *G* koji su povezani nekim putem sa svim ostalim vrhovima koji počinju istim slovom (moguće da put sadrži i vrhove koji ne počinju tim slovom). Npr. ako imamo graf `a1-b1, a2-b2, b1-b2, c1-d5, c2-d10, e1-f1` i `e1-f2`, funkcija treba vratiti skup `{a,b,e,f}`.

Opće napomene:

- Možete pretpostaviti da graf neće imati više od 100 vrhova.
- Elementi u vektoru/listi/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:

- U mapi *G*, svaki brid će biti naveden samo jedan put. Tako npr. ukoliko su *a1* i *b2* povezani, ili će G[„a1“] sadržavati „b2“ ili obrnuto.
- Mapa *G* nam predstavlja neusmjeren graf, pa funkcija *dostupni* (a isto tako i druge) treba vratiti prazan skup ukoliko vrh *n* ne postoji u grafu (i graf treba ostati nepromijenjen), 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<pair<char, int>, set<pair<char, int> > > G;

	vector<pair<char, int> > vp = susjedni(make_pair('a', 1), G);
	for (int i = 0; i < vp.size(); ++i)
		cout << vp[i].first << vp[i].second << endl;
	// nista se ne ispise

	list<pair<char, int> > lp = dostupni(make_pair('a', 1), G);
	for (list<pair<char, int> >::iterator i = lp.begin(); i != lp.end(); ++i)
		cout << i->first << i->second << endl;
	// nista se ne ispise


	G[make_pair('a', 1)].insert(make_pair('b', 2));
	G[make_pair('c', 3)].insert(make_pair('b', 2));

	lp = dostupni(make_pair('a', 1), G);
	for(list<pair<char, int> >::iterator i = lp.begin(); i != lp.end(); ++i)
		cout << i->first << i->second << " ";
	cout << endl; // a1 b2 c3

	lp = dostupni(make_pair('b', 2), G);
	for (list<pair<char, int> >::iterator i = lp.begin(); i != lp.end(); ++i)
		cout << i->first << i->second << " ";
	cout << endl; // a1 b2 c3

	queue<pair<char, int> > q;
	q.push(make_pair('c', 3));
	stack<pair<char, int> > sp = dostupniN(q, G);

	while (!sp.empty()) {
		pair<char, int> t = sp.top();
		sp.pop();
		cout << t.first << t.second << " ";
	}
	cout << endl; // c3 b2 a1

	set<char> sc = povezani(G);
	for(set<char>::iterator i = sc.begin(); i != sc.end(); ++i)
		cout << *i;
	cout << endl; // abc

	G[make_pair('a', 5)].insert(make_pair('e', 7));
	sc = povezani(G);
	for (set<char>::iterator i = sc.begin(); i != sc.end(); ++i)
		cout << *i;
	cout << endl; // bce

	sp = dostupniN(q, G);

	while (!sp.empty()) {
		pair<char, int> t = sp.top();
```

```
        sp.pop();
        cout << t.first << t.second << " ";
    }
    cout << endl; // c3 b2 a1

    q.push(make_pair('e', 7));
    sp = dostupniN(q, G);
    while (!sp.empty()) {
        pair<char, int> t = sp.top();
        sp.pop();
        cout << t.first << t.second << " ";
    }
    cout << endl; // e7 c3 b2 a5 a1

    return 0;
}
```