STRUKTURE PODATAKA I ALGORITMI

ZBIRKA ZADATAKA

Ishod učenja 1

2024-2025.

Zadatak 1

Isparsirajte sve podatke iz datoteke **air_pollution.csv** u vektor. Prvi stupac sadrži naziv države, drugi naziv grada, treći godinu mjerenja, četvrti izmjerenu vrijednost sitnih čestica u zraku zvanih PM2.5, a peti izmjerenu vrijednost krupnijih čestica u zraku zvanih PM10. Prvi, drugi i treći stupac su obavezni i uvijek sadrže podatke. Ako četvrti ili peti stupac nemaju upisani podatak, pretpostavite da je izmjerena vrijednost jednaka 0. Nakon toga, pronađite najveću vrijednost PM2.5 i najveću vrijednost PM10 pa ih ispišite zajedno s državom, gradom i godinom:

```
Max PM2.5: 191.9, Bangladesh, Rajshahi, 2011
Max PM10: 317.88, Bahrain, Hamad Town, 2012
```

Odredite i argumentirate (objasnite) a priori vremensku složenost vašeg algoritma za pronalazak najvećih vrijednosti PM2.5 i PM10, u najboljem i najgorem slučaju. Odgovor napišite u komentar.

Moguće rješenje:

```
struct AirPollution {
    string country;
    string city;
    string year;
    double pm25;
    double pm10;
};

int main() {
    ifstream dat("air_pollution.csv");
    if (!dat) {
        cout << "Greska pri otvaranju datoteke air_pollution.csv" << endl;
        return 1;
    }

    vector<AirPollution> v;
    string country, city, year, pm25, pm10;

    string temp;
    getline(dat, temp);

    while (getline(dat, country, ';')) {
        getline(dat, city, ';');
        getline(dat, year, ';');
        getline(dat, pm25, ';');
        getline(dat, pm10);

        if (pm25 == "") {
            pm25 = "0";
        }
        if (pm10 == "") {
            pm10 = "0";
        }

        v.push_back({ country, city, year, stof(pm25), stof(pm10) });
    }

    int i1 = 0;
    int i2 = 0;
    for (int i = 0; i < v.size(); i++) {
        if (v[i].pm25 > v[i1].pm25) {
```

```
            i1 = i;
        }
        if (v[i].pm10 > v[i2].pm10) {
            i2 = i;
        }
    }

    cout << "Max PM2.5: " << v[i1].pm25 << ", " << v[i1].country << ", " <<
v[i1].city << ", " << v[i1].year << endl;
    cout << "Max PM10: " << v[i2].pm10 << ", " << v[i2].country << ", " <<
v[i2].city << ", " << v[i2].year << endl;

      // U oba slučaja je vremenska složenost algoritma O(n) jer je potrebno proći
sve elemente vektora kako bi se pronašle najveće vrijednosti.
}
```

Zadatak 2

Na osnovu tablice odaberite i napišite u komentar sve strukture koje je najbolje izbjegavati ako nam je ključno imati efikasno brisanje u srednjem slučaju. Argumentirate (objasnite) svoj odabir.

Data Structure	Time Complexity							
	Average				Worst			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)

Ispis na konzolu je spora operacija, a u ovom zadatku ćete to i dokazati a posteriori analizom:

- Napišite prvu for petlju koja ispisuje sve brojeve između 1 i 100 te izmjerite i ispišite koliko to traje.
- Ispod, napišite drugu for petlju koja zbraja sve brojeve između 1 i 100 te izmjerite i ispišite koliko to traje.

Trajanja ispišite u mikrosekundama.

Moguće rješenje:

```
#include <iostream>
#include <string>
#include <chrono>
using namespace std;
using namespace std::chrono;

int main() {
    // Potrebno je izbjegavati Array, Skip list i Binary Search Tree jer su njihove složenosti O(n) i O(logn), što je lošije od O(1) što imaju preostale strukture.

    auto t1 = std::chrono::high_resolution_clock::now();
    for (int i = 1; i <= 100; i++) {
        cout << i << endl;
    }
    auto t2 = chrono::high_resolution_clock::now();
    cout << "Trajanje a) dijela u mikrosekundama: " << chrono::duration_cast<chrono::microseconds>(t2 - t1).count() << endl;

    t1 = std::chrono::high_resolution_clock::now();
    int suma = 0;
    for (int i = 1; i <= 100; i++) {
        suma += i;
    }
    t2 = chrono::high_resolution_clock::now();
    cout << "Trajanje b) dijela u mikrosekundama: " << chrono::duration_cast<chrono::microseconds>(t2 - t1).count() << endl;
}
```

Zadatak 3

Dizajnirajte i implementirajte novu klasu Namjestaj. Neka Namjestaj može čuvati naziv (npr. Sendsberg), kategoriju (npr. Stolovi), cijenu (npr. 39.99), boju (npr. crna) te je li dostupan ili ne. Omogućite da korisnik prilikom izrade objekta tipa Namjestaj ima sljedeće opcije:

a. Može kreirati objekt zadavanjem samo naziva i cijene. U tom slučaju, pretpostavite da se radi o bijelom namještaju iz kategorije Razno koji je dostupan.
b. Može kreirati objekt zadavanjem naziva, kategorije i cijene. U tom slučaju, pretpostavite da se radi o bijelom namještaju koji je dostupan.
c. Može kreirati objekt zadavanjem svih vrijednosti.

Dodatno, na tipu podataka Namjestaj definirajte još dvije operacije:

a. Omogućite korisniku da jednim pozivom metode postavi dostupnost namještaja.
b. Omogućite korisniku ispis svih podataka o namještaju u formatu prema vašoj želji.

Kreirajte tri objekta u main()-u (svaki na jedan od mogućih načina) te na svakom pozovite obje metode.

Moguće rješenje:

Namjestaj.h

```
#pragma once
#include <string>
using namespace std;

class Namjestaj {
private:
    string naziv;
    string kategorija;
    double cijena;
    string boja;
    bool dostupan;

public:
    Namjestaj(string naziv, double cijena);
    Namjestaj(string naziv, string kategorija, double cijena);
    Namjestaj(string naziv, string kategorija, double cijena, string boja, bool
dostupan);
    void set_dostupnost(bool vrijednost);
    string ispis();
};
```

Namjestaj.cpp

```
#include "Namjestaj.h"

Namjestaj::Namjestaj(string naziv, double cijena) {
    this->naziv = naziv;
    this->kategorija = "Razno";
    this->cijena = cijena;
    this->boja = "Bijela";
    this->dostupan = true;
}

Namjestaj::Namjestaj(string naziv, string kategorija, double cijena) {
    this->naziv = naziv;
    this->kategorija = kategorija;
    this->cijena = cijena;
```

```
    this->boja = "Bijela";
    this->dostupan = true;
}

Namjestaj::Namjestaj(string naziv, string kategorija, double cijena, string boja,
bool dostupan) {
    this->naziv = naziv;
    this->kategorija = "Razno";
    this->cijena = cijena;
    this->boja = boja;
    this->dostupan = dostupan;
}

void Namjestaj::set_dostupnost(bool vrijednost) {
    dostupan = vrijednost;
}

string Namjestaj::ispis() {
    return naziv + " " + kategorija + " " + to_string(cijena) + " " + boja + " " +
(dostupan ? "DOSTUPAN" : "NIJE DOSTUPAN");
}
```

Source.cpp

```
#include "Namjestaj.h"

int main() {
    Namjestaj n1{ "Sendsberg", 199.99 };
    n1.set_dostupnost(false);
    cout << n1.ispis() << endl;

    Namjestaj n2{ "Sendsberg", "Stolovi", 199.99 };
    n2.set_dostupnost(false);
    cout << n2.ispis() << endl;

    Namjestaj n3{ "Sendsberg", "Stolovi", 199.99, "Plava", false };
    cout << n3.ispis() << endl;
    n3.set_dostupnost(true);
    cout << n3.ispis() << endl;
}
```

Zadatak 4

Parsirajte sve podatke iz datoteke **weather_classification_data.csv** u vektor. Datoteka sadrži puno stupaca, ali vam trebaju samo tri: stupac Temperature sadrži decimalni broj, stupac Season sadrži naziv godišnjeg doba (Summer, Winter, Spring, Autumn), a stupac Weather Type sadrži opis vremena (Rainy, Cloudy, Sunny, Snowy). Zanemarite sve ostale stupce. Nakon toga pronađite najnižu temperaturu ljeti i najnižu temperaturu zimi i ispišite ih ovako:

```
Najniza ljetna temperatura je -20 stupnjeva (Snowy)
Najniza zimska temperatura je -25 stupnjeva (Snowy)
```

Moguće rješenje:

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
using namespace std;

struct WeatherData {
	double temperature;
	string season;
	string weather;
};

int main() {
	ifstream dat("weather_classification_data.csv");
	if (!dat) {
		cout << "Greska pri otvaranju datoteke" << endl;
		return 1;
	}

	vector<WeatherData> data;
	string temperature;
	string season;
	string weather;
	string temp;
	getline(dat, temp);

	while (getline(dat, temperature, ',')) {
		getline(dat, temp, ',');
		getline(dat, temp, ',');
		getline(dat, temp, ',');
		getline(dat, temp, ',');
		getline(dat, temp, ',');
		getline(dat, temp, ',');
		getline(dat, season, ',');
		getline(dat, temp, ',');
		getline(dat, temp, ',');
		getline(dat, weather, '\n');

		data.push_back({ stod(temperature), season, weather });
	}

	dat.close();

	int min_summer = 0;
	int min_winter = 0;
```

```
	for (int i = 0; i < data.size(); i++) {
		if (data[i].season == "Summer" and data[i].temperature <
data[min_summer].temperature) {
			min_summer = i;
		}
		if (data[i].season == "Winter" and data[i].temperature <
data[min_winter].temperature) {
			min_winter = i;
		}
	}

	cout << "Najniza ljetna temperatura je " << data[min_summer].temperature << "
stupnjeva (" << data[min_summer].weather << ")" << endl;
	cout << "Najniza zimska temperatura je " << data[min_winter].temperature << "
stupnjeva (" << data[min_winter].weather << ")" << endl;
}
```

Zadatak 5

Dizajnirajte i implementirajte novu klasu Pas. Kad klasu koristite u main() ovako:

```
Pas dog1("Barney");
dog1.feed();
dog1.print(); // ispisuje "Barney (Mutt) has happiness: 1"
dog1.play();
dog1.print(); // ispisuje "Barney (Mutt) has happiness: 2"

Pas dog2("Ruffus", "Golden retriever");
dog2.scratch();
dog2.print(); // ispisuje "Ruffus (Golden retriever) has happiness: 1"
dog2.scratch();
dog2.print(); // ispisuje "Ruffus (Golden retriever) has happiness: 2"
dog2.yell();
dog2.print(); // ispisuje "Ruffus (Golden retriever) has happiness: 0"
```

Mora ispisati sljedeće:

```
Barney (Mutt) has happiness: 1
Barney (Mutt) has happiness: 2
Ruffus (Golden retriever) has happiness: 1
Ruffus (Golden retriever) has happiness: 2
Ruffus (Golden retriever) has happiness: 0
```

Moguće rješenje:

Pas.h

```
#pragma once
#include <iostream>
#include <string>
using namespace std;

class Pas {
private:
	string name;
	string kind;
	int happiness;

public:
	Pas(string name);
	Pas(string name, string kind);
	void scratch();
	void print();
	void feed();
	void play();
	void yell();
};
```

Pas.cpp

```
#include "Pas.h"

Pas::Pas(string name) {
	this->name = name;
	this->kind = "Mutt";
```

```
	this->happiness = 0;
}

Pas::Pas(string name, string kind) {
	this->name = name;
	this->kind = kind;
	this->happiness = 0;
}

void Pas::scratch() {
	happiness++;
}

void Pas::print() {
	cout << name << " (" << kind << ") has happiness: " << happiness << endl;
}

void Pas::feed() {
	happiness++;
}

void Pas::play() {
	happiness++;
}

void Pas::yell() {
	happiness = 0;
}
```

Source.cpp

```
#include <iostream>
#include <string>
#include "Pas.h"
using namespace std;

int main() {
	Pas dog1("Barney");
	dog1.feed();
	dog1.print(); // ispisuje "Barney (Mutt) has happiness: 1"
	dog1.play();
	dog1.print(); // ispisuje "Barney (Mutt) has happiness: 2"

	Pas dog2("Ruffus", "Golden retriever");
	dog2.scratch();
	dog2.print(); // ispisuje "Ruffus (Golden retriever) has happiness: 1"
	dog2.scratch();
	dog2.print(); // ispisuje "Ruffus (Golden retriever) has happiness: 2"
	dog2.yell();
	dog2.print(); // ispisuje "Ruffus (Golden retriever) has happiness: 0"
}
```

Zadatak 6

Isparsirajte sve podatke iz datoteke **lego-database-colors.csv** u vektor. Nakon toga, izračunajte i ispišite odgovor na pitanje: koliko ukupno ima transparentnih boja za kockice (podatak o transparentnosti neke boje piše u zadnjem stupcu „is_trans“ – vrijednost „f“ znači da nije transparentan, vrijednost „t“ znači da jest transparentan). Odredite i argumentirate (objasnite) a priori vremensku složenost vašeg algoritma za brojanje transparentnih kockica.

Moguće rješenje:

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
using namespace std;

struct LegoDatabaseColor {
	int id;
	string name;
	string rgb;
	char is_trans;
};

int main() {
	ifstream dat("lego-database-colors.csv");
	if (!dat) {
		cout << "Greska pri otvaranju datoteke" << endl;
		return 1;
	}

	vector<LegoDatabaseColor> data;
	string id;
	string name;
	string rgb;
	string is_trans;
	string temp;
	getline(dat, temp);

	while (getline(dat, id, ',')) {
		getline(dat, name, ',');
		getline(dat, rgb, ',');
		getline(dat, is_trans, '\n');

		data.push_back({ stoi(id), name, rgb, is_trans[0] });
	}

	dat.close();

	int ukupno_transparentnih = 0;

	for (const auto& color : data) {
		if (color.is_trans == 't') {
			ukupno_transparentnih++;
		}
	}

	cout << "Ukupno transparentnih boja: " << ukupno_transparentnih << endl;
```

```
	// Algoritam za brojanje transparentnih kockica ima i u najboljem i u
najgorem slučaju složenost O(n)
	// jer je uvijek potrebno proći kroz sve podatke.
}
```

Zadatak 7

Na osnovu tablice odaberite i napišite sve strukture koje je najbolje koristiti ako nam je ključno imati efikasno umetanje u prosječnom slučaju. Argumentirate (objasnite) svoj odabir.

Data Structure	Time Complexity							
	Average				Worst			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)

Izmjerite i ispišite koliko vremena traje zbrajanje svih brojeva od 1 do 2.100.000.000 (a posteriori analiza). Ispišite dobiveni zbroj tih brojeva.

Moguće rješenje:

```
#include <iostream>
#include <vector>
#include <string>
#include <chrono>
using namespace std;
using namespace std::chrono;

int main() {
      // Ako nam je ključno imati efikasno umetanje u prosječnom slučaju, najbolje
je koristiti Stack, Queue, Singly-linked List, Doubly-linked List ili Hash Table.
Razlog za to je složenost O(1) koja je konstanta i ne ovisi o broju elemenata u
kontejneru.

    auto t1 = chrono::high_resolution_clock::now();
    long long suma = 0;
    for (int i = 1; i <= 2100000000; i++) {
        suma += i;
    }
    cout << "Zbroj je: " << suma << endl;
    auto t2 = chrono::high_resolution_clock::now();

    cout << "Trajanje u mikrosekundama: " <<
chrono::duration_cast<chrono::microseconds>(t2 - t1).count() << endl;
}
```

Zadatak 8

Dizajnirajte i implementirajte dva tipa podataka koristeći klase Drzava i Grad. Neka Grad može čuvati svoj naziv i broj stanovnika, a neka Drzava može čuvati svoj naziv i proizvoljan broj gradova. Omogućite da se prilikom izrade objekta tipa Grad mora zadati naziv i broj stanovnika, a prilikom izrade objekta tipa Drzava mora zadati samo naziv države. Dodatno, na tipu podataka Drzava definirajte još dvije operacije: dodavanje novog grada na temelju proslijeđenog naziva i broja stanovnika te ispis ukupnog broja stanovnika u državi koji se dobije zbrajanjem stanovnika iz svih gradova dodijeljenih državi (prema potrebi dodajte getere na klasu Grad). Demonstrirajte korištenje tako da kreirate državu, u nju ubacite tri grada te ispišete ukupan broj stanovnika u državi.

Moguće rješenje:

Grad.h

```
#pragma once
#include <string>
using namespace std;

class Grad {
private:
	string naziv;
	int broj_stanovnika;
public:
	Grad(string naziv, int broj_stanovnika);
	int get_broj_stanovnika() const;
};
```

Grad.cpp

```
#include "Grad.h"

Grad::Grad(string naziv, int broj_stanovnika) {
	this->naziv = naziv;
	this->broj_stanovnika = broj_stanovnika;
}

int Grad::get_broj_stanovnika() const
{
	return broj_stanovnika;
}
```

Drzava.h

```
#pragma once
#include <iostream>
#include <vector>
#include <string>
#include "Grad.h"
using namespace std;

class Drzava {
private:
	string naziv;
	vector<Grad> gradovi;
public:
	Drzava(string naziv);
	void dodaj_grad(string naziv, int broj_stanovnika);
```

```
	void ispisi_ukupni_broj_stanovnika();
};
```

Drzava.cpp

```
#include "Drzava.h"

Drzava::Drzava(string naziv) {
	this->naziv = naziv;
}

void Drzava::dodaj_grad(string naziv, int broj_stanovnika) {
	gradovi.push_back({ naziv, broj_stanovnika });
}

void Drzava::ispisi_ukupni_broj_stanovnika() {
	int ukupno = 0;
	for (const Grad& grad : gradovi) {
		ukupno += grad.get_broj_stanovnika();
	}
	cout << "Ukupno stanovnika u drzavi " << naziv << " je " << ukupno << endl;
}
```

Source.cpp

```
#include "Drzava.h"
using namespace std;

int main() {
	Drzava d("Bosna i Hercegovina");
	d.dodaj_grad("Sarajevo", 340000);
	d.dodaj_grad("Mostar", 100000);
	d.dodaj_grad("Zenica", 120000);
	d.ispisi_ukupni_broj_stanovnika();
}
```

Zadatak 9

Isparsirajte podatke iz datoteke **pokedex.csv** u vektor. Koristite samo stupce name, height, weight i info. Nakon toga, za svakog pokemona ispišite ime, opis te BMI i BMI kategoriju. BMI se računa formulom visina/težina^2, a kategorije su:

- bmi< 18.5: underweight
- 18.5 <= bmi < 24.9: healthy weight
- bmi >= 24.9: overweight

Pazite na činjenicu da se zarez može pojaviti i u opisu.

Moguće rješenje:

Pokemon.h

```
#pragma once
#include <string>
#include <vector>
#include "Pokemon.h"
using namespace std;

class Pokemon {
private:
	string name;
	int height;
	int weight;
	string info;
	double bmi() const;

public:
	Pokemon(const string& name, const int& height, const int& weight, const
string& info);
	void print() const;
};
```

Pokemon.cpp

```
#include <iostream>
#include <string>
#include "Pokemon.h"
using namespace std;

double Pokemon::bmi() const
{
	return static_cast<double>(weight) / pow(height, 2);
}

Pokemon::Pokemon(const string& name, const int& height, const int& weight, const
string& info)
	: name(name), height(height), weight(weight), info(info)
{
}

void Pokemon::print() const
{
	double b = bmi();a
	cout << name << " (" << info << ")" << endl;
	cout << "BMI: " << b << " (";
```

```
	if (b < 18.5) {
		cout << "underweight";
	}
	else if (b < 24.9) {
		cout << "healthy weight";
	}
	else {
		cout << "overweight";
	}
	cout << ")" << endl << endl;
}
```

Source.cpp

```
#include <iostream>
#include <vector>
#include <fstream>
#include "Pokemon.h"
using namespace std;

vector<string> split(string& original) {
	vector<string> results;

	int begin = 0;
	int end = 0;
	bool inside = false;
	for (int i = 0; i < original.size(); i++) {
		end++;

		if (original[i] == '"') {
			inside = !inside;
		}

		if (original[i] == ',' and !inside) {
			if (original[i - 1] == '"') {
				results.push_back(string(original.begin() + begin + 1,
original.begin() + end - 2));
			}
			else {
				results.push_back(string(original.begin() + begin,
original.begin() + end - 1));
			}
			begin = i + 1;
		}
	}

	if (*(original.begin() + begin) == '"') {
		results.push_back(string(original.begin() + begin + 1, original.begin()
+ end - 1));
	}
	else {
		results.push_back(string(original.begin() + begin, original.begin() +
end));
	}

	return results;
}
```

```
int main() {
	ifstream dat("pokedex.csv");
	if (!dat) {
		cout << "Error opening file" << endl;
		return 1;
	}

	vector<Pokemon> pokemons;
	string line;
	getline(dat, line);

	while (getline(dat, line)) {
		auto fields = split(line);
		pokemons.push_back({
			fields[1],
			stoi(fields[2]),
			stoi(fields[3]),
			fields[12]
		});
	}
	dat.close();

	for (const auto& pok : pokemons) {
		pok.print();
	}
}
```

Zadatak 10

Dizajnirajte i implementirajte novu klasu `Train`. Kad klasu koristite u main() ovako:

```
Train passenger_train1("Zagreb", "Vienna");
passenger_train1.speed_up(10);
passenger_train1.speed_up(10);
passenger_train1.speed_up(10);
passenger_train1.print(); // prints "Passenger train Zagreb-Vienna has speed: 30 km/h"
passenger_train1.slow_down();
passenger_train1.print(); // prints "Passenger train Zagreb-Vienna has speed: 25 km/h"
passenger_train1.slow_down();
passenger_train1.slow_down();
passenger_train1.print(); // prints "Passenger train Zagreb-Vienna has speed: 15 km/h"

Train cargo_train1("Zagreb", "Vienna", false);
cargo_train1.speed_up(50);
cargo_train1.print(); // prints "Cargo train Zagreb-Vienna has speed: 50 km/h"
cargo_train1.slow_down();
cargo_train1.print(); // prints "Cargo train Zagreb-Vienna has speed: 45 km/h"
cargo_train1.stop();
cargo_train1.print(); // prints "Cargo train Zagreb-Vienna has speed: 0 km/h"
```

Mora ispisati sljedeće:

```
Passenger train Zagreb-Vienna has speed: 30 km / h
Passenger train Zagreb-Vienna has speed: 25 km / h
Passenger train Zagreb-Vienna has speed: 15 km / h
Cargo train Zagreb-Vienna has speed: 50 km / h
Cargo train Zagreb-Vienna has speed: 45 km / h
Cargo train Zagreb-Vienna has speed: 0 km / h
```

Moguće rješenje:

Train.h

```
#pragma once
#include <string>
#include <iostream>
using namespace std;

class Train {
private:
	int speed;
	bool is_passenger;
	string from;
	string to;

public:
	Train(string pfrom, string pto);
	Train(string pfrom, string pto, bool pis_passenger);
	void speed_up(int speed);
	void slow_down();
	void stop();
	void print();
};
```

Train.cpp

```
#include "Train.h"

Train::Train(string pfrom, string pto)
{
	speed = 0;
	is_passenger = true;
	from = pfrom;
	to = pto;
}

Train::Train(string pfrom, string pto, bool pis_passenger)
{
	speed = 0;
	is_passenger = pis_passenger;
	from = pfrom;
	to = pto;
}

void Train::speed_up(int speed)
{
	this->speed += speed;
}

void Train::slow_down()
{
	speed -= 5;
}

void Train::stop()
{
	speed = 0;
}

void Train::print()
{
	if (is_passenger) {
		cout << "Passenger ";
	}
	else {
		cout << "Cargo ";
	}
	cout << "train " << from << "-" << to << " has speed: " << speed << " km / h"
<< endl;
}
```