Univerzitet u Tuzli

Fakultet Elektrotehnike

Uvod u Računarske Algoritme

Zadaća 1

Rekurzija i pretraživanje

Tuzla, 23. 4. 2020.

Napomena

U svim problemima koji slijede nije dozvoljena upotreba komandi i funkcija koje dosad nisu korištene na predavanjima ili vježbama. Smije se koristiti std::sort ukoliko je potrebno. Dozvoljena je upotreba kontejnera iz standardne biblioteke (std::vector i std::list), kao i C nizova. Nerekurzivna rješenja se ne mogu smatrati tačnim ukoliko je u zadatku naglašeno da je potrebno koristiti rekurziju.

Zadatak 1

Napisati funkciju *divide* koja uz pomoć rekurzije i oduzimanja određuje rezultat cjelobrojnog dijeljenja dva broja. Funkcija treba da se izvršava u *O(n)* vremenu.

Zadatak 2

Napisati funkciju *fast_divide* koja uz pomoć rekurzije određuje količnik dva cijela broja metodom egipatskog dijeljenja. Ova metoda je slična metodi egipatskog množenja koja je rađena na vježbama.

Mala pomoć: tražiti najveći stepen broja dva, koji kad se pomnoži sa djeliteljem, bude manji od djeljenika, te ga oduzeti.

Zadatak 3

Napisati rekurzivnu funkciju koja provjerava da li je proslijeđeni c string palindrom. Funkcija treba imati potpis *bool palindrome(const char* s, int n)*.

Zadatak 4

Napisati rekurzivnu funkciju koja računa sumu svih elemenata u nizu. Funkcija treba da ima sljedeći prototip: *int sum(const int* array, int n);* gdje je array adresa prvog elementa u nizu, a n ukupan broj elemenata.

Zadatak 5

Napisati rekurzivnu funkciju koja računa proizvod parnih prirodnih brojeva manjih i jednakih broju *n*. Od korisnika se traži unos broja *n* u *main* funkciji.

Zadatak 6

Napisati funkciju *push_unique* sa potpisom:

```
bool push_unique(std::vector<int>&, int)
```

Gdje je prvi parametar niz brojeva, a drugi parametar je element kojeg treba ubaciti na kraj niza samo u slučaju da element već nije u nizu. Funkcija treba da se izvršava u *O(n)* vremenu. Funkcija vraća true ukoliko je element ubačen u niz.

Zadatak 7

Implementirati funkciju *sorted_position* sa potpisom:

```
std::vector<int>::iterator sorted_position(std::vector<int>&, const int&);
```

Gdje je prvi parametar niz brojeva koji je sortiran, a drugi parametar je element kojeg korisnik želi ubaciti. Funkcija treba da u *O(log n)* vremenu pronađe mjesto u nizu gdje treba ubaciti element tako da bi nakon ubacivanja elementa vektor ostao sortiran. Funkcija vraća natrag iterator na poziciju gdje treba ubaciti element. Ova funkcija ne treba da ubaci element, nego samo da pronađe poziciju. Pretpostaviti da korisnik ubaciju elemente sa *vector::insert* metodom.

Zadatak 8

Implementirati verziju algoritma std::partition koja ima sljedeći potpis:

```
template <typename Iter, typename P>
Iter partition(Iter begin, Iter end, const P& p);
```

Ovaj algoritam prima opseg niza opisan pomoću dva iteratora, uzetih putem template parametra, a koji trebaju zadovoljavati karakteristike *ForwardIteratora*. Dodatni parametar *p* je predikat funkcija koja prima element niza, a vraća natrag bool vrijednost. Funkcija treba da izmjeni redoslijed elemenata u nizu na način da se svi elementi za koje je predikat funkcija *p* vratila *true* nalaze ispred elemenata za koje je funkcija vratila *false*. Međusobni poredak elemenata u ove dvije grupe nije bitan. Funkcija vraća iterator na prvi element iz druge grupe, za koje je predikat funkcija p vratila false. Dva primjera upotrebe partition algoritma:

```
std::vector<int> v{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

std::partition(v.begin(), v.end(), [](int elem) { return elem & 2 == 0; });
// sadržaj vektora v nakon poziva:
// v = {0, 8, 2, 6, 4, *5*, 3, 7, 1, 9};
// funkcija bi vratila iterator koji pokazuje na element 5 (boldiran).

std::vector<int> v{2, 1, 3, 8, 4, 5, 6, 9, 7};
auto s = std::partition(v.begin(), v.end(),
  [](const int& e) -> bool { return e < 4; });

for (auto it = begin(v); it < s; ++it)
  std::cout << *it << ' ';
std::cout << '*';
for (; s < end(v); ++s)
  std::cout << ' ' << *s;
std::cout << std::endl;
// Ispis programa bi bio
// 2 1 3 * 8 4 5 6 9 7
```

Zadatak 9

U prilogu zadaće se nalazi file *shakespeare.txt*. Potrebno je implementirati program koji će učitati sve riječi koje se nalaze u tom fileu, a zatim od korisnika tražiti unos jedne po jedne riječi sve do kraja programa. Nakon unosa riječi, program treba da provjeri da li riječ koju je korisnik unio nalazi u fileu, te da ispiše korisniku poruku da li je riječ pronađena. Pored toga potrebno je ispisati vrijeme trajanja pretrage za tom riječi.

Potrebno je implementirati najbrže rješenje sa stanovišta pretrage. Pretpostaviti da se vrijeme unosa i obrade podataka može zanemariti, jer će potencijalni korisnik ovog programa učitati podatke samo jednom, a nakon toga nekoliko stotina puta tražiti riječ.

Način predavanja

Zadaću je potrebno predati u obliku arhive *ime_prezime_zadaca1.zip*, pri čemu sadržaj arhive treba da bude direktorij *ime_prezime_zadaca1*, sa poddirektorijem za svaki zadatak, sa imenima: zadatak1, zadatak2, ..., zadatak9.

Pored toga, potrebno je da se studenti strogo pridržavaju naziva metoda, njihovih parametara i povratnih vrijednosti u zadacima gdje su oni naglašeni. Za izlazak na provjeru potrebno je implementirati i predati minimalno 50% zadaće.

Sve predane zadaće će biti automatski pregledane za plagijate i svaki pokušaj prepisivanja će biti prijavljen i adekvatno sankcionisan.