Univerzitet u Tuzli

Fakultet Elektrotehnike

Uvod u Računarske Algoritme

Zadaća 1

Rekurzija i pretraživanje

Tuzla, 29. 4. 2023.

Uvod u Računarske Algoritme, Zadaća 1

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.

Za svaki zadatak potrebno je napraviti funkciju koja se od korisnika traži, i main funkciju koja će testirati rad funkcije na nekoliko primjera. Dozvoljeno je implementirati svaki zadatak inline, u jednom `cpp` fileu. Možete raditi i odvojene implementacije u `hpp cpp` paru.

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, nije potrebno da `vector` bude sortiran. Funkcija vraća `true` ukoliko je element ubačen u niz.

Zadatak 7

Implementirati funkciju *upper_bound* sa potpisom:

```
It upper_bound(It begin, It end, const T& element);
```

`begin` i `end` predstavljaju dva `random_access` iteratora, na primjer iz kontenjera `std::vector`. Niz na koji iteraotri pokazuju je sortiran. Treći argument je `element` kojeg korisnik želi ubaciti u niz.

Funkcija treba da u *O(log n)* vremenu pronađe mjesto u nizu gdje treba ubaciti element tako da bi nakon ubacivanja elementa niz ostao sortiran. Funkcija vraća natrag iterator na poziciju gdje treba ubaciti element. Pretpostaviti da korisnik ubaciju elemente sa *vector::insert* metodom (ona ubacuje prije).

Pomoć: U standardnoj biblioteci postoji algoritam `upper_bound`. Ideja je da algoritam pomoću polovljenja intervala pronalazi prvi element koji je veći od traženog elementa, ili iterator na kraj ako je element veći od svih elemenata u nizu. Kod u prilogu bi radio pravilno ubacivanje u niz:

```
std::vector v{2,4,6,8,10};
auoto it = std::upper_bound(v.begin(), v.end(), 7); // vraća iterator na 8
v.insert(it, 7); // v = {2,4,5,7,8,10};

it = std::upper_bound(v.begin(), v.end(), 20); // vraća iterator na end
v.insert(it, 20); // v = {2,4,5,7,8,10,20};

it = std::upper_bound(v.begin(), v.end(), 7); // vraća iterator na 8
v.insert(it, 7); // v = {2,4,5,7,7,8,10,20};
```

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
```

Izazov za dodatne bodove

Implementirati algoritam *stable_partition*, koji ima isti potpis kao partition, ali će relativni redoslijed elemenata ostati isti u dva dijla niza, dakle

```
std::vector<int> v{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
std::stable_partition(v.begin(), v.end(), [](int elem) { return elem & 2 == 0; });
// v = {0, 2, 4, 6, 8, *1*, 3, 5, 7, 9};
```

Ovaj algoritam treba raditi u O(n logn) vremenu, i trebao bi biti *inline* što znači da se ne može rješiti tako da napravimo novi `vector` u koji ćemo prebaciti elemente.

Zadatak 9

U prilogu zadaće se nalazi file *shakespeare.txt*. Potrebno je napraviti dva main programa koji rješavaju isti problem, ali drugim pristupom. Potrebno je učitati sve riječi iz txt falea u tip podatka `std::vector<std::string>` na način da vector bude sortiran nakon učitavanja.

Prvi program će da učitava riječ po riječ, ubacuje novu riječ na kraj `vectora` te na kraju poziva std::sort.

Drugi program će da učitava riječ po riječ, te će korištenjem upper_bound i insert pristupa, objašnjenog u zadatku 7, da ubacuje riječi tako da `vector` bude

i ostane sortiran.

Potrebno je izmjeriti vrijeme izvršenja prvog i drugog pristupa te odgovoriti na pitanje koji pristup je brži. Pokušajte odrediti algoritamsku složenost prvog i drugog pristupa.

Zadatak 10

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.

Djelimično rješenje ovog problema je rađeno na posljednjim AV, te možete taj kod koristiti kao osnovu.

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.

Dodatne ideje za optimizaciju bi bile:

- Pretvoriti sva slova na ulazu u mala. Potencijalno se u ulazu nalaze riječi **Is** i **is**, koje prestavljaju istu riječ.
- Ukloniti duplikate riječi. Primjera radi, riječ Hamlet se sigurno nalazi nekoliko desetina puta u ulaznom fileu. Pokušajte izbrisati sve duplikate iz `vector`-a. Budući da je `vector` soritran, algoritam koji briše duplikate bi se mogao napraviti u O(n) vremenu.

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, zadatak10.

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.