Programiranje 2

Laboratorijske vježbe

LV9
Pretraživanje i sortiranje podataka

Fakultet elektrotehnike računarstva i informacijskih tehnologija Osijek
Kneza Trpimira 2b
www.ferit.unios.hr

Uvod

Algoritam predstavlja proceduru kojom se u određenom broju koraka može riješiti problem s računalom. Koraci algoritma mogu uključiti ponavljanje, ovisno o problemu koji algoritam treba riješiti. Najčešće se koriste algoritmi za pretraživanja i sortiranja podataka, unutar neke strukture podataka kao što su polja, povezani popis... . Na ovim laboratorijskim vježbama spomenut će se rad s dva algoritma za pretraživanje i dva algoritma za sortiranje, objasnit će se koncepti rada i njihova konkretna implementacija u C programskom jeziku.

Algoritmi pretraživanja podataka

Algoritmi za pretraživanje podataka su algoritme kojima se dohvaća informacija spremljena unutar određene strukture podataka. Ovisno o strukturi podataka, mogu se koristiti različiti algoritmi za pretraživanje podataka.

Na ovim laboratorijskim vježbama pokazat će se primjena dva algoritma za pretraživanje podataka nad cjelobrojnim poljem. To je algoritam sekvencijalnog pretraživanja, poznat pod imenom linearno pretraživanje i algoritam binarnog pretraživanja.

Linearno pretraživanje (Sekvencijalno pretraživanje)

Linearno pretraživanje je vrlo jednostavan algoritam pretraživanja podataka i ne zahtjeva sortirano polje. Za pronalazak traženog broja unutar polja, potrebno je radi usporedbu traženog broja sa svakim članom polja, počevši od nultog indeksa do zadnjeg indeksa n – 1. U slučaju pronalaska traženog broja unutar polja, vraća se indeks pronađenog broja, u suprotno pretraživanje se nastavlja do kraja polja i ako se traženi broj ne pronađe, vraća se vrijednost -1.

Primjer 1: Pronalaženje broja 8 unutar polja od 10 elemenata.

indeks	0	1	2	3	4	5	6	7	8	9
usporedba 1	5	7	3	4	6	8	2	9	1	0
usporedba 2	5	7	3	4	6	8	2	9	1	0
usporedba 3	5	7	3	4	6	8	2	9	1	0
usporedba 4	5	7	3	4	6	8	2	9	1	0
usporedba 5	5	7	3	4	6	8	2	9	1	0
usporedba 6	5	7	3	4	6	8	2	9	1	0

Implementacija algoritma linearnog pretraživanja kroz funkciju u C programskom jeziku.

```
int linearnoPretrazivanje(const int polje[], const int n, const int trazeniBroj) {

	for (int i = 0; i < n; i++)
	{
		if (polje[i] == trazeniBroj) {
			return i;
		}
	}
	return -1;
}
```

Cjelokupni kôd koji prikazuje linearno pretraživanje na jednostavnom primjeru.

```c
#include <stdio.h>

int linearnoPretrazivanje(const int[], const int, const int);

int counter;

int main(void) {

	int polje[] = { 5, 7, 3, 4, 6, 8, 2, 9, 1, 0 };
	int n = -1;
	int trazeniBroj = 8;
	int indeks = -1;

	n = sizeof(polje) / sizeof(int);

	indeks = linearnoPretrazivanje(polje, n, trazeniBroj);

	if (indeks == -1) {
		printf("Trazeni broj %d nije pronadjen unutar polja.\n", trazeniBroj);
	}
	else {
		printf("Trazeni broj %d je pronadjen unutar polja nakon %d koraka na
indeksu %d\n", trazeniBroj, counter, indeks);

	}

	return 0;
}


int linearnoPretrazivanje(const int polje[], const int n, const int trazeniBroj) {

	for (int i = 0; i < n; i++)
	{
		counter++;
		if (polje[i] == trazeniBroj) {
			return i;
		}
	}
	return -1;
}
```

Binarno pretraživanje

Binarno pretraživanje je brzi algoritam koji radi na principu podijeli i vladaj. Kako bi ovaj algoritam ispravno radio potrebno je sortirati polje (npr. uzlazno ili silazno).

Binarno pretraživanje prvo odredi srednji indeks polja i na temelju tog indeksa dohvaća se element polja koji se uspoređuje s traženom vrijednosti. U slučaju pronalaska traženog broja unutar polja, vraća se indeks pronađenog broja. Ako je element polja pod srednjim indeksom veći od traženog broja, pretraživanje se nastavlja u pod-polju lijevo od srednjeg indeksa. Ako je element polja pod srednjim indeksom manji od traženog broja, pretraživanje se nastavlja u pod-polju desno od srednjeg indeksa. Ovi koraci se ponavljaju sve dok se ne pronađe traženi broja unutar polja, ili dok se veličina pod-polja ne smanji na nulu, gdje se u tom slučaju vraća vrijednost -1.

Primjer 2: Pronalaženje traženog broja čija je vrijednost 8 unutar polja od 10 elemenata.

indeks	0	1	2	3	4	5	6	7	8	9
usporedba 1	0	1	2	3	4	5	6	7	8	9
usporedba 2	0	1	2	3	4	5	6	7	8	9
usporedba 3	0	1	2	3	4	5	6	7	8	9

Implementacija algoritma binarnog pretraživanje kroz funkciju u C programskom jeziku.

```
int binarnoPretrazivanje(const int polje[], const int n, const int trazeniBroj) {

	int dg = 0;
	int gg = n - 1;
	int s = -1;

	while (dg <= gg) {
		s = (dg + gg) / 2;

		if (polje[s] == trazeniBroj) {
			return s;
		}
		else if (polje[s] > trazeniBroj) {
			gg = s - 1;
		}
		else {
			dg = s + 1;
		}
	}

	return -1;
}
```

Cjelokupni kôd koji prikazuje binarno pretraživanje na jednostavnom primjeru.

```c
#include <stdio.h>

int binarnoPretrazivanje(const int[], const int, const int);

int counter;

int main(void) {

	int polje[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
	int n = -1;
	int trazeniBroj = 8;
	int indeks = -1;

	n = sizeof(polje) / sizeof(int);

	indeks = binarnoPretrazivanje(polje, n, trazeniBroj);

	if (indeks == -1) {
		printf("Trazeni broj %d nije pronadjen unutar polja.\n", trazeniBroj);
	}
	else {
		printf("Trazeni broj %d je pronadjen unutar polja nakon %d koraka na
indeksu %d\n", trazeniBroj, counter, indeks);
	}

	return 0;
}


int binarnoPretrazivanje(const int polje[], const int n, const int trazeniBroj) {

	int dg = 0;
	int gg = n - 1;
	int s = -1;

	while (dg <= gg) {
		s = (dg + gg) / 2;
		counter++;

		if (polje[s] == trazeniBroj) {
			return s;
		}
		else if (polje[s] > trazeniBroj) {
			gg = s - 1;
		}
		else {
			dg = s + 1;
		}
	}

	return -1;
}
```

Algoritmi sortiranja podataka

Algoritmi za sortiranje podataka su algoritmi kojima se elementi unutar neke strukture podataka sortiraju po određenom kriteriju. Kriterij sortiranja može biti numerički redoslijed od najmanje vrijednosti prema najvećoj (uzlazno sortiranje) ili obrnuto od najveće vrijednosti prema najmanjoj (silazno sortiranje). Sortiranje podataka je vrlo važno, jer neki algoritmi pretraživanja podataka zahtijevaju unutar određene strukture podataka sortirane elemente prema određenom kriteriju.

Na ovim laboratorijskim vježbama pokazat će se primjena dva algoritma za sortiranje podataka nad cjelobrojnim poljima. To su algoritmi sortiranje biranjem (engl. *Selection sort*) i mjehurićasto sortiranje (engl. *Bubble sort*).

Sortiranje biranjem (engl. *Selection sort*)

Sortiranje biranjem je jednostavni algoritam kojim se polje dijeli na dva dijela. Prvi dio je sortirani dio, koji se nalazi s lijeve strane polja, a drugi dio je nesortirani dio koji se nalazi na desnoj strani polja. Na početku je sortirani dio prazan, a ne sortirani dio predstavlja cijelo polje.

Algoritam radi tako da pronađe najmanji element u nesortiranom dijelu polja i zamijeni ga s početnim lijevim elementom polja. Prvo se u nesortiranom dijelu polja pronađe element koji je manji od početnog lijevog elementa te mu se zapamti indeks, tzv. *min* indeks. Nakon toga se do kraja nesortiranog dijela polja nastavlja provjera za mogućim manjim elementom te ako postoji njegov se indeks pamti kao *min*. Kada se dođe do kraja nesortiranog dijela polja, taj najmanji element s indeksom *min* postaje dio sortiranog dijela polja. Proces zamijene se nastavlja u svakom sljedećem koraku pomičući granicu sortiranog dijela polja za jedan element u desno.

Primjer 3: Sortiranje cjelobrojnog polja od 10 elemenata uzlazno primjenom algoritma *Selection sort*.

indeks	0	1	2	3	4	5	6	7	8	9
prolaz 1	5	7	3	4	6	8	2	9	1	0
prolaz 1	5	7	3	4	6	8	2	9	1	0
prolaz 1	5	7	3	4	6	8	2	9	1	0
prolaz 1	5	7	3	4	6	8	2	9	1	0
zamjena 1	0	7	3	4	6	8	2	9	1	5
prolaz 2	0	7	3	4	6	8	2	9	1	5
prolaz 2	0	7	3	4	6	8	2	9	1	5
prolaz 2	0	7	3	4	6	8	2	9	1	5
zamjena 2	0	1	3	4	6	8	2	9	7	5
prolaz 3	0	1	3	4	6	8	2	9	7	5
zamjena 3	0	1	2	4	6	8	3	9	7	5
prolaz 4	0	1	2	4	6	8	3	9	7	5
zamjena 4	0	1	2	3	6	8	4	9	7	5
prolaz 5	0	1	2	3	6	8	4	9	7	5
zamjena 5	0	1	2	3	4	8	6	9	7	5
prolaz 6	0	1	2	3	4	8	6	9	7	5
prolaz 6	0	1	2	3	4	8	6	9	7	5
zamjena 6	0	1	2	3	4	5	6	9	7	8

Implementacija algoritma *Selection sort* kroz funkciju u C programskom jeziku.

```
void zamjena(int* const veci, int* const manji) {

int temp = 0;

	temp = *manji;
	*manji = *veci;
	*veci = temp;
}
void selectionSort(int polje[], const int n) {

	int min = -1;

	for (int i = 0; i < n - 1; i++)
	{
		min = i;

		for (int j = i + 1; j < n; j++)
		{
			if (polje[j] < polje[min]) {
				min = j;
			}
		}
		zamjena(&polje[i], &polje[min]);
	}
}
```

Cjelokupni kôd koji prikazuje sortiranje cjelobrojnog polja pomoću algoritma *Selection sort*.

```
#include <stdio.h>

void ispis(const int[], const int);
void zamjena(int* const, int* const);
void selectionSort(int[], const int);

int main(void) {

	int polje[] = { 5, 7, 3, 4, 6, 8, 2, 9, 1, 0 };
	int n = -1;

	n = sizeof(polje) / sizeof(int);

	ispis(polje, n);
	selectionSort(polje, n);
	ispis(polje, n);

	return 0;
}

void ispis(const int polje[], const int n) {

	for (int i = 0; i < n; i++)
	{
		if (i == 0) {
			printf("polje[%d ", polje[i]);
		}
		else if (i > 0 && i < n - 1) {
			printf("%d ", polje[i]);
		}
		else {
			printf("%d]\n", polje[i]);
		}
	}
}
void zamjena(int* const veci, int* const manji) {
int temp = 0;

	temp = *manji;
	*manji = *veci;
	*veci = temp;
}
void selectionSort(int polje[], const int n) {

	int min = -1;

	for (int i = 0; i < n - 1; i++)
	{
		min = i;

		for (int j = i + 1; j < n; j++)
		{
			if (polje[j] < polje[min]) {
				min = j;
			}
		}
		zamjena(&polje[i], &polje[min]);
	}
}
```

Mjehurićasto sortiranje (engl. *Bubble sort*)

Mjehurićasto sortiranje je jednostavan algoritam koji se temelji na uspoređivanju susjednih elemenata. Ako susjedni elementi nisu sortirani, sadržaj na njihovim mjestima će se zamijeniti.

Algoritam radi tako da u prvom prolazu uzastopno uspoređuje parove susjednih elemenata i ako je drugi element manji od prvog elementa, zamijenit će im se sadržaj na njihovim mjestima. Nakon prvog prolaska, najveći element će biti postavljen na kraju polja. U svakom sljedećem prolasku kroz cjelobrojno polje, ponovno se uzastopno uspoređuju parovi susjednih elemenata i ako je potrebno vrši se zamjena sadržaja na njihovim mjestima, dok se sortirani dio polja ne dira.

Primjer 3: Sortiranje cjelobrojnog polja od 10 elemenata uzlazno primjenom algoritma *Bubble sort*.

indeks	0	1	2	3	4	5	6	7	8	9
prolaz 1	5	7	3	4	6	8	2	9	1	0
prolaz 1	5	7	3	4	6	8	2	9	1	0
prolaz 1	5	3	7	4	6	8	2	9	1	0
prolaz 1	5	3	4	7	6	8	2	9	1	0
prolaz 1	5	3	4	6	7	8	2	9	1	0
prolaz 1	5	3	4	6	7	8	2	9	1	0
prolaz 1	5	3	4	6	7	2	8	9	1	0
prolaz 1	5	3	4	6	7	2	8	9	1	0
prolaz 1	5	3	4	6	7	2	8	1	9	0
zamjena 1	5	3	4	6	7	2	8	1	0	9
prolaz 2	5	3	4	6	7	2	8	1	0	9
prolaz 2	3	5	4	6	7	2	8	1	0	9
prolaz 2	3	4	5	6	7	2	8	1	0	9
prolaz 2	3	4	5	6	7	2	8	1	0	9
prolaz 2	3	4	5	6	7	2	8	1	0	9
prolaz 2	3	4	5	6	2	7	8	1	0	9
prolaz 2	3	4	5	6	2	7	8	1	0	9
prolaz 2	3	4	5	6	2	7	1	8	0	9
zamjena 2	3	4	5	6	2	7	1	0	8	9
prolaz 3	3	4	5	6	2	7	1	0	8	9
prolaz 3	3	4	5	6	2	7	1	0	8	9
prolaz 3	3	4	5	6	2	7	1	0	8	9
prolaz 3	3	4	5	6	2	7	1	0	8	9
prolaz 3	3	4	5	2	6	7	1	0	8	9
prolaz 3	3	4	5	2	6	7	1	0	8	9
prolaz 3	3	4	5	2	6	1	7	0	8	9
zamjena 3	3	4	5	2	6	1	0	7	8	9
prolaz 4	3	4	5	2	6	1	0	7	8	9
prolaz 4	3	4	5	2	6	1	0	7	8	9
prolaz 4	3	4	5	2	6	1	0	7	8	9

Implementacija algoritma *Bubble sort* kroz funkciju u C programskom jeziku.

```
void zamjena(int* const veci, int* const manji) {

int temp = 0;

	temp = *manji;
	*manji = *veci;
	*veci = temp;
}

void bubbleSort(int polje[], const int n) {

	for (int i = 0; i < n - 1; i++)
	{
		for (int j = 0; j < n - 1 - i; j++)
		{
			if (polje[j + 1] < polje[j]) {
				zamjena(&polje[j + 1], &polje[j]);
			}
		}
	}
}
```

Cjelokupni kôd koji prikazuje sortiranje cjelobrojnog polja pomoću algoritma *Bubble sort*.

```c
#include <stdio.h>

void ispis(const int[], const int);
void zamjena(int* const, int* const);
void bubbleSort(int[], const int);


int main(void) {

	int polje[] = { 5, 7, 3, 4, 6, 8, 2, 9, 1, 0 };
	int n = -1;

	n = sizeof(polje) / sizeof(int);

	ispis(polje, n);
	bubbleSort(polje, n);
	ispis(polje, n);

	return 0;
}

void ispis(const int polje[], const int n) {

	for (int i = 0; i < n; i++)
	{
		if (i == 0) {
			printf("polje[%d ", polje[i]);
		}
		else if (i > 0 && i < n - 1) {
			printf("%d ", polje[i]);
		}
		else {
			printf("%d]\n", polje[i]);
		}
	}
}

void zamjena(int* const veci, int* const manji) {
int temp = 0;

	temp = *manji;
	*manji = *veci;
	*veci = temp;
}
void bubbleSort(int polje[], const int n) {

	for (int i = 0; i < n - 1; i++)
	{
		for (int j = 0; j < n - 1 - i; j++)
		{
			if (polje[j + 1] < polje[j]) {
				zamjena(&polje[j + 1], &polje[j]);
			}
		}
	}
}
```

Zadaci

1. Napisati C program koji će kreirati tekstualnu datoteku pod imenom *dat.txt* i u koju će se spremiti 500 cijelih brojeva koje je potrebno pseudo-slučajno generirati unutar intervala [0, 1000]. Učitati broj ***n*** s tipkovnice koji također mora biti unutar intervala [0, 1000]. Pročitati sve podatke iz tekstualne datoteke *dat.txt* u cjelobrojno polje, te provjeriti sekvencijalnim pretraživanjem nalazi li se taj broj ***n*** u ranije učitanom polju. Ukoliko je broj pronađen ispisati poruku "Broj %d je pronadjen nakon %d koraka na indeksu %d." Ukoliko broj nije pronađen ispisati poruku "Broj %d nije pronadjen.".

2. Napisati C program koji će kreirati tekstualnu datoteku pod imenom *dat.txt* i u koju će se spremiti 500 cijelih brojeva koje je potrebno pseudo-slučajno generirati unutar intervala [0, 1000]. S tipkovnice učitava 5 cijelih brojeva iz intervala [1, 1000]. Zatim se učitava 500 cijelih brojeva iz datoteke *dat.txt*. Izvršiti pretragu svakog od unesenih brojeva s tipkovnice u učitanom polju koristeći algoritme sekvencijalnog i binarnog pretraživanja. Na kraju je potrebno izračunati i ispisati srednji broj pretraživanja za svaki algoritam pojedinačno (pri izračunu srednjeg broja pretraživanja uzeti u obzir samo ona pretraživanja koja su bila uspješna). Ispis na ekran prilagoditi formatu prema sljedećem primjeru:

   Za unesene brojeve 23, 67, 45, 89, 347
   REZULTATI:
   Sekvencijalno:
   1. broj 23 je pronadjen u 377 koraka.
   2. broj 67 nije pronadjen.
   3. broj 45 nije pronadjen.
   4. broj 89 nije pronadjen.
   5. broj 347 nije pronadjen.
   Binarno:
   1. broj 23 je pronadjen u 9 koraka.
   2. broj 67 nije pronadjen.
   3. broj 45 nije pronadjen.
   4. broj 89 nije pronadjen.
   5. broj 347 nije pronadjen.
   Srednji broj koraka za sekvencijalno pretrazivanje je: 377.00
   Srednji broj koraka za binarno pretrazivanje je: 9.00