Eratostenov algoritam za nalaženje prostih brojeva

- Za prirodan broj p veći od 1 kažemo da je *prost (prim)* ako nema drugih djelitelja osim 1 i samog broja p.
- Ukoliko želimo naći sve proste brojeve između 2 i n, obično se služimo postupkom kojeg je prvi opisao antički matematičar Eratosten.
- Primjer: naći sve proste brojeve između 2 i 29. Najprije napišemo sve te brojeve jedan iza drugoga:
- 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
- Zatim u gornjem nizu označimo sve višekratnike broja 2 (ali ne i sam broj 2):
- 2, 3, **4**, 5, **6**, 7, **8**, 9, **10**, 11, **12**, 13, **14**, 15, **16**, 17, **18**, 19, **20**, 21, **22**, 23, **24**, 25, **26**, 27, **28**, 29
- Sada se s prvog elementa polja (element 2) pomaknemo na sljedeći *neoznačeni* element. To je broj 3. Označimo sada sve višekratnike broja 3 (ali ne i sam broj 3). Dobijemo ovo:
- 2, 3, **4**, 5, **6**, 7, **8**, **9**, **10**, 11, **12**, 13, **14**, **15**, **16**, 17, **18**, 19, **20**, **21**, **22**, 23, **24**, 25, **26**, **27**, **28**, 29
- Sada se s elementa 3 pomičemo na sljedeći neoznačeni element. To je broj 5.
- 2, 3, **4**, 5, **6**, 7, **8**, **9**, **10**, 11, **12**, 13, **14**, **15**, **16**, 17, **18**, 19, **20**, **21**, **22**, 23, **24**, **25**, **26**, **27**, **28**, 29
- Sljedeći neoznačeni element bio bi broj 7. Međutim njegove višekratnike ne moramo označavati!
- Dovoljno je označiti višekratnike samo onih prirodnih brojeva koji su manji od drugog korijena najvećeg elementa u nizu. Kako je sqrt(29) ≈ 5.3851, označeni su svi potrebni višekratnici.
- Nakon završenog postupka, u nizu će neoznačeni ostati *jedino prosti brojevi*.
- Napisati program koji Eratostenovim algoritmom pronalazi i ispisuje sve proste brojeve manje od 100000.

```
#include <stdio.h>
#include<stdlib.h>
#include<math.h>
#define MAXSIZE 100001
void Eratosten(int a[], int n);
void main(){
	int max;
	int prosti[MAXSIZE] = { 0 };
	printf("Unesi broj manji ili jednak %d: ", MAXSIZE - 1);   scanf("%d", &max);
	printf("\n");
	Eratosten(prosti, max);
	printf("Svi prosti brojevi manji ili jednaki %d su:\n", max);
	for (int i = 2; i <= max; i++)
		if (prosti[i] == 0)    printf("%d ", i);
	printf("\n");
	system("PAUSE");
}
void Eratosten(int a[], int n){
	for(int i = 2; i <= sqrt(n); i++) {
		if (a[i] == 0) {
			int k = 2;
			while (k * i <= n){
				a[k * i] = 1;
				k++;}  }  }
}
```

Primjer za različite složenosti istog problema

- Zadano je polje cijelih brojeva A_0, A_1,…,A_{n-1}. Brojevi mogu biti i negativni. Potrebno je pronaći najveću vrijednost sume niza brojeva. Pretpostavit će se da je najveća suma 0 ako su svi brojevi negativni. Problem je razmatran na predavanjima.
- Kubna složenost: Ispituju se svi mogući podnizovi. U vanjskoj petlji se varira prvi član podniza, u srednjoj petlji varira se zadnji član podniza, u unutrašnjoj petlji varira se duljina niza od prvog člana do zadnjeg člana. Sve 3 petlje se za najgori slučaj obavljaju n puta: apriorna složenost $O(n^3)$.
- Kvadratna složenost: ako uočimo da je treća petlja nepotrebna (ne treba se suma svaki put računati iznova), složenost se može reducirati na $O(n^2)$.
- Linearna * logaritamska složenost: $O(n\log_2 n)$ - relativno složeni rekurzivni postupak. Ako se ulazno polje podijeli približno po sredini, rješenje može biti takvo da je maksimalna suma u lijevom dijelu polja, ili je u desnom dijelu polja ili prolazi kroz oba dijela. Prva dva slučaja mogu biti riješena rekurzivno. Zadnji slučaj se može realizirati tako da se nađe najveća suma u lijevom dijelu koja uključuje njegov zadnji član i najveća suma u desnom dijelu koja uključuje njegov prvi član. Te se dvije sume zbroje i uspoređuju s one prve dvije.
- Linearna složenost: zbrajaju se svi članovi polja redom, a pamti se ona suma koja je u cijelom tijeku tog postupka bila najveća, pa je složenost algoritma $O(n)$

```
#include <stdio.h>
#include <stdlib.h>

// vraca niz znakova c u zadanoj duljini n
char* nc (int c, int n) {
     static char s[80+1];
     s[n] = '\0';                    // prirubi
  while (--n >= 0) s[n] = c;     // popuni
     return s;
}

// ispis polja
void ispisi(int A[], int n) {
     int i;
     printf("\n");
     for (i = 0; i < n; i++) printf(" A[%d]",i);
     printf("\n");
     for (i = 0; i < n; i++) printf("%5d", A[i]);
     printf("\n");
}
```

```
// Kubna slozenost
int MaxPodSumaNiza3 (int A[], int N) {
    int OvaSuma, MaxSuma, i, j, k;
    int iteracija = 0;

    MaxSuma = 0;
    for (i = 0; i < N; i++) {
        printf ("i=%d\n", i);

        for (j = i; j < N; j++) {
            OvaSuma = 0;
            for (k = i; k <= j; k++) {
                OvaSuma += A [k];
                ++iteracija;
            }
     if (OvaSuma > MaxSuma)
        MaxSuma = OvaSuma;
        printf ("Suma clanova [%d, %d] = %d, a najveca = %d\n",i, j, OvaSuma, MaxSuma);
        }
    }
    printf ("Broj iteracija: %d\n", iteracija);
    return MaxSuma;
}
```

```
// Kvadratna slozenost
int MaxPodSumaNiza2 (int A[ ], int N) {
    int OvaSuma, MaxSuma, i, j;
    int iteracija = 0;

    MaxSuma = 0;
    for (i = 0; i < N; i++) {
        printf ("i=%d\n", i);

        OvaSuma = 0;
        for (j = i; j < N; j++) {
            OvaSuma += A[ j ];
            ++iteracija;
            if (OvaSuma > MaxSuma)
               MaxSuma = OvaSuma;
           printf ("Suma clanova [%d, %d] = %d, a najveca = %d\n",i, j, OvaSuma, MaxSuma);
        }
    }
    printf ("Broj iteracija: %d\n", iteracija);
    return MaxSuma;
}
```

```
// NlogN slozenost - koristi funkcije Max3 i MaxPodSuma
// racuna najveci od 3 broja
int Max3 (int A, int B, int C) {
     return A > B ? A > C ? A : C : B > C ? B : C;
}
// trazi najvecu podsumu clanova od Lijeva do Desna
int MaxPodSuma (int A[], int Lijeva, int Desna, int dubina) {
     int MaxLijevaSuma, MaxDesnaSuma;
     int MaxLijevaRubnaSuma, MaxDesnaRubnaSuma;
     int LijevaRubnaSuma, DesnaRubnaSuma;
     int Sredina, i, ret;

  printf ("%s> MaxPodSuma(%d, %d) ...\n", nc(' ', dubina*2), Lijeva, Desna);
     if (Lijeva == Desna) { // Osnovni slucaj
       if (A [Lijeva] > 0)
              ret = A [Lijeva]; // podniz od clana A[Lijeva]
       else
              ret = 0; // suma je 0 ako su svi brojevi negativni
    printf ("%s< MaxPodSuma(%d, %d) = %d\n",nc(' ', dubina*2), Lijeva, Desna, ret);
            return ret;
     }
// racun lijeve i desne podsume s obzirom na Sredina
     Sredina = (Lijeva + Desna) / 2;
     MaxLijevaSuma = MaxPodSuma (A, Lijeva, Sredina, dubina+1);
     MaxDesnaSuma = MaxPodSuma (A,Sredina + 1, Desna, dubina+1);
```

```
// najveca gledano ulijevo od sredine
    MaxLijevaRubnaSuma = 0; LijevaRubnaSuma = 0;
    for (i = Sredina; i >= Lijeva; i--) {
            LijevaRubnaSuma += A [i];
            if (LijevaRubnaSuma > MaxLijevaRubnaSuma)
                    MaxLijevaRubnaSuma = LijevaRubnaSuma;
    }
    // najveca gledano udesno od sredine
  MaxDesnaRubnaSuma = 0; DesnaRubnaSuma = 0;
  for (i = Sredina + 1; i <= Desna; i++) {
    DesnaRubnaSuma += A [i];
    if (DesnaRubnaSuma > MaxDesnaRubnaSuma)
     MaxDesnaRubnaSuma = DesnaRubnaSuma;
  }
    printf ("%s  Lijeva=%d Desna=%d Rubna=%d\n",
                nc (' ', dubina*2), MaxLijevaSuma, MaxDesnaSuma,
                MaxLijevaRubnaSuma + MaxDesnaRubnaSuma);

    // najveca od lijeva, desna, rubna
  ret = Max3 (MaxLijevaSuma, MaxDesnaSuma,
                    MaxLijevaRubnaSuma + MaxDesnaRubnaSuma);
    printf ("%s< MaxPodSuma(%d, %d) = %d\n",
                            nc(' ', dubina*2), Lijeva, Desna, ret);
    return ret;
}
```

```
// NlogN slozenost
int MaxPodSumaNizaLog (int A [], int N) {
    return MaxPodSuma (A, 0, N - 1, 0);
}

// Linearna složenost
int  MaxPodSumaNiza1 (int A[], int N) {
    int OvaSuma, MaxSuma, j;

    OvaSuma = MaxSuma = 0;
    for (j = 0; j < N; j++) {
        OvaSuma += A[ j ];
        if (OvaSuma > MaxSuma)
            MaxSuma = OvaSuma;
        else if (OvaSuma < 0)
            OvaSuma = 0;        // povecanje izgleda sljedeceg podniza
        printf ("j=%d OvaSuma=%2d MaxSuma=%2d\n", j, OvaSuma, MaxSuma);
    }
    return MaxSuma;
}
```

```
void main (void) {
  int A [] = {2,5,-8,4,-2,5,-5,4,3,-2};
  int rez;
      printf("\n\nKubna slozenost\n");
      ispisi(A, sizeof (A) / sizeof (A [0]));
      rez = MaxPodSumaNiza3 (A, sizeof (A) / sizeof (A [0]));
       printf("\nMaxSuma3 = %d", rez);
       getchar();
      printf("\n\nKvadratna slozenost\n");
      ispisi(A, sizeof (A) / sizeof (A [0]));
      rez = MaxPodSumaNiza2 (A, sizeof (A) / sizeof (A [0]));
       printf("\nMaxSuma2 = %d", rez);
      getchar();
      printf("\n\nLogaritamska slozenost\n");
      ispisi(A, sizeof (A) / sizeof (A [0]));
      rez = MaxPodSumaNizaLog (A, sizeof (A) / sizeof (A [0]));
       printf("\nMaxSumaLog = %d", rez);
       getchar();
      printf("\n\nLinearna slozenost\n");
      ispisi(A, sizeof (A) / sizeof (A [0]));
      rez = MaxPodSumaNiza1 (A, sizeof (A) / sizeof (A [0]));
       printf("\nMaxSuma1 = %d\n", rez);
       system("PAUSE");
      exit (0);
}
```

Prikaz liste pomoću polja

- Na predavanjima je prikazan apstraktni tip podataka općenite liste i C pseudokod za operacije definirane na listi u implementaciji pomoću polja. Napisati program u kojem se napravi lista cijelih brojeva maksimalne duljine 100 članova, u listu upisati proizvoljan broj slučajno generiranih cijelih brojeva, ubaciti novi član na kraj i sredinu liste te izbrisati izabrani član, članove koji se nalaze ispred i nakon prethodno izbrisanog, te na kraju izbrisati cijelu listu. Nakon svake od ovih operacija ispisati sadržaj liste.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAXLENGTH 100

typedef int elementtype;
typedef int position;
typedef struct list * listptr;
typedef struct list LIST;

position End(listptr lst);
position MakeNull(listptr lst);
void Insert(elementtype x, position p, listptr lst);
void Delete(position p, listptr lst);
position First(listptr lst);
position Next(position p, listptr lst);
position Previous(position p, listptr lst);
elementtype Retrieve(position p, listptr lst);
void PrintErrorAndTerminate(char errormsg[]);
void PrintList(listptr lst);

struct list{
    elementtype elements[MAXLENGTH];
    int last;
};
```

```
position End(listptr lst) {
    return lst->last + 1;
}

position MakeNull(listptr lst) {
    lst->last = -1;
    return 0;
}

void Insert(elementtype x, position p, listptr lst) {
    position q;

    if(lst->last >= MAXLENGTH - 1)
        PrintErrorAndTerminate("Insert: Lista je do kraja popunjena!");
    else {
        if(p > lst->last + 1 || p < 0)
            PrintErrorAndTerminate("Insert: Pozicija ne postoji!");
        else {
            for(q = lst->last; q >= p; q--)
                lst->elements[q + 1] = lst->elements[q];
            lst->last++;
            lst->elements[p] = x;
        }
    }
}
```

```
void Delete(position p, listptr lst) {
    position q;
    if (p > lst->last || p < 0)
        PrintErrorAndTerminate("Delete: pozicija ne postoji!");
    else   {
        lst->last--;
        for(q = p; q <= lst->last; q++)
            lst->elements[q] = lst->elements[q + 1];
    }
}

position First(listptr lst) {
        return 0;
}

position Next(position p, listptr lst) {
    return ++p;
}

position Previous(position p, listptr lst) {
    return --p;
}
```

```
elementtype Retrieve(position p, listptr lst) {
    if (p >= 0 && p <= lst->last)
        return lst->elements[p];
    else
        PrintErrorAndTerminate("Retrieve: Nepostojeca pozicija!");
        return 0;
}

void PrintErrorAndTerminate(char errormsg[]) {
    printf("%s\n", errormsg);
    exit(-1);
}

void PrintList(listptr lst) {
    printf("< ");
    for(position p = 0; p < End(lst); p++)
        printf("%d ", Retrieve(p, lst));
    printf(">\n");
}

int main() {
    LIST lst;
    elementtype clan;
    position mjesto, len;
```

```
MakeNull(&lst);
PrintList(&lst);
printf ("Koliko clanova zelite upisati u listu\n");
scanf("%d",&len);
printf("\n");

   srand(time(NULL)*4);
   for (mjesto=0; mjesto < len; mjesto++) {
         clan = (elementtype) 100 * ((float)rand() / (RAND_MAX + 1));
         Insert(clan, mjesto, &lst);
     }
PrintList(&lst);

printf ("Clan koji zelite upisati na zadnje mjesto u listu\n");
scanf("%d",&clan);
printf("\n");
         Insert(clan, End(&lst), &lst);
PrintList(&lst);

printf ("Clan koji zelite upisati u sredinu liste\n");
scanf("%d",&clan);
printf("\n");
```

```
    Insert(clan, (End(&lst) - First(&lst))/2, &lst);
    PrintList(&lst);

    printf ("Koju poziciju zelite izbrisati iz liste\n");
    scanf("%d",&mjesto);
    Delete(mjesto, &lst);
    PrintList(&lst);

    printf("A sad cu izbrisati clan prije i poslije izbrisanog\n");
     if (mjesto > 0) Delete(Previous(mjesto--, &lst),&lst);
     if (mjesto < End(&lst)) Delete(Next(--mjesto,&lst),&lst);
    PrintList(&lst);

     printf("A sad brisem cijelu listu\n");
         for (mjesto=First(&lst); mjesto < End(&lst); mjesto++) {
             Delete(mjesto--, &lst);
             }
    PrintList(&lst);
    system("PAUSE");
    return 0;
}
```