Osnovne akademske studije

PREDMET: Objektno-orijentisana simulacija

TEMA: Strategija raspoređivanja događaja 2

Predmetni nastavnik: Prof. dr Milorad Stanojević
Asistent: mr Marko Đogatović

Lehmerov generator slučajnih brojeva

Lehmerov GSB je takođe poznat kao multiplikativni linearni kongruentni GSB. To je najčešće korišćeni linearni kongruentni generator. Postoje tri parametra:

1. ***m*** **– modulus,**
2. ***a*** **– multiplikator,**
3. ***c*** **– aditivna konstantna (najčešće je jednaka 0).**

Takođe postoji i početna vrednost z_0 (tzv. seme). Pseudoslučajne vrednosti se generišu primenom sledeće formule:

$Z_i = (Z_{i-1}a+c) \bmod m$**. (mod je moduo - ostatak nakon deljenja dva cela broja)**

Ova formula generiše slučajnu sekvencu od *m*-1 brojeva. Za kombinaciju parametara koja vodi ka sekvenci od *m*-1 brojeva se kaže da ima celi period. Najčešće se vrednosti Z_i se normalizuju tako da se nalaze na jediničnom intervalu na sledeći način

$U_i = Z_i / m$ za i=1,2,...

Uspeh ovakvog generatora zavisi od izbora parametara. Npr. izborom m = 11 i multipikatorom a = 8 dobija se sekvenca sa celim periodom, dok multilikator a = 9 daje sekvencu dužine 5.

a = 8 ... 1, 8, 9, 6, 4, 10, 3, 2, 5, 7, 1 ...
a = 9 ...1, 9, 4, 3, 5, 1 ,...

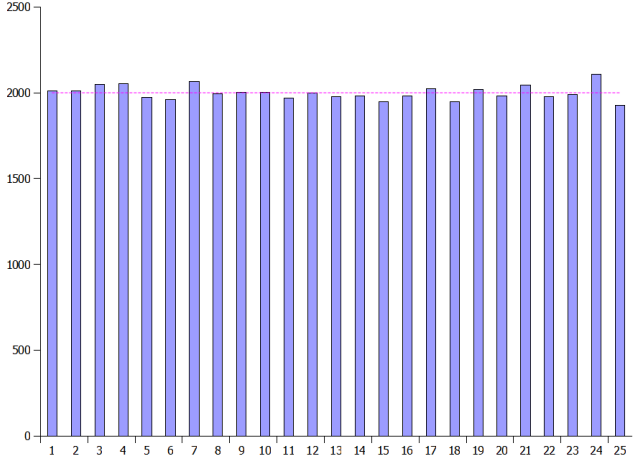
Park i Miller daju odgovarajući izbor parametara $m = 2147483647 = 2^{31}-1$ i $a = 16807 = 7^5$ za generator sa celim periodom. U nastavku su dati programi realizovani u C jeziku kojima je implementiran Lehmerov GSB.

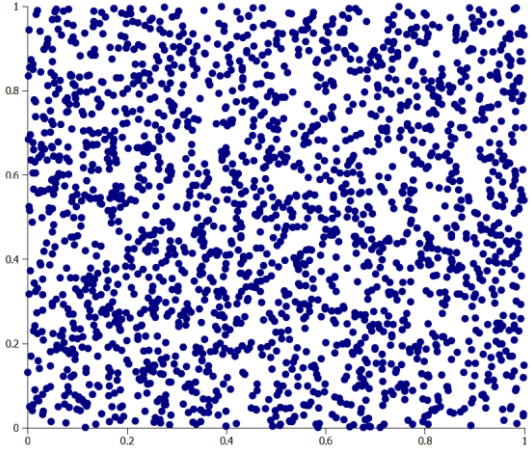
```c
#include <stdio.h>
#include <limits.h>

struct rand {
  int seed,mult;
  double val;
};
void randu(struct rand* rnd) {
  rnd->seed *= rnd->mult;
  if(rnd->seed<0)
    rnd->seed += LONG_MAX+1;
  rnd->val = rnd->seed/(1.0*LONG_MAX);
}
```

```c
int main() {
  int i;
  struct rand rnd;
  rnd.seed = 1;
  rnd.mult = 16807;
  for(i=0; i<50000; ++i) {
    randu(&rnd);
    printf("%f\n", rnd.val);
  }
  return 0;
}
```

```c
int main() {
  int i;
  struct rand rnd;
  double r1, r2;
  rnd.seed = 1;
  rnd.mult = 16807;
  for(i=0; i<2000; ++i) {
    randu(&rnd);
    r1 = rnd.val;
    randu(&rnd);
    r2 = rnd.val;
    printf("%f,%f\n", r1, r2);
  }
  return 0;
}
```





Histogram od 50000 vrednosti generisanih `randu` funkcijom i dvodimenzionalna raspodela 2000 pseudoslučajnih parova generisanih `randu` funkcijom

`rand` funkcija

```
int rand(void);
```

vraća pseudoslučajni ceo broj na intervalu od 0 do `RAND_MAX`. Broj se generiše algoritmom koji vraća sekvencu slučajnih brojeva svaki put kada se funkcija pozove. Najčešće je algoritam neka varijanta linearnog kongruentnog generatora slučajnih brojeva.

Ovaj algoritam koristi seme da generiše seriju. Seme treba da bude inicijalizovano nekom vrednošću korišćenjem funkcije `srand`.

`RAND_MAX` je konstanta definisana u `<stdlib.h>` (`<cstdlib>`). Ova vrednost je najmanje 32767. `rand()` vraća broj na intervalu od 0 do 1 korišćenjem sledećeg dela koda

```
double r = rand()/(RAND_MAX+1.0);
```

`srand` funkcija

```
void srand(unsigned int seed);
```

Ova funkcija inicijalizuje generator slučajnih brojeva. GSB je inicijalizovan korišćenjem argumenta prosleđenog kao seme (`seed`).

Za različite vrednosti semena korišćene prilikom poziva `srand`, generator pseudoslučajnih brojeva će generisati različite sekvence brojeva.

Ukoliko se seme postavi na 1, generator se inicijalizuje na svoju početnu vrednost i vraća iste vrednosti kao i pre poziva `srand`.

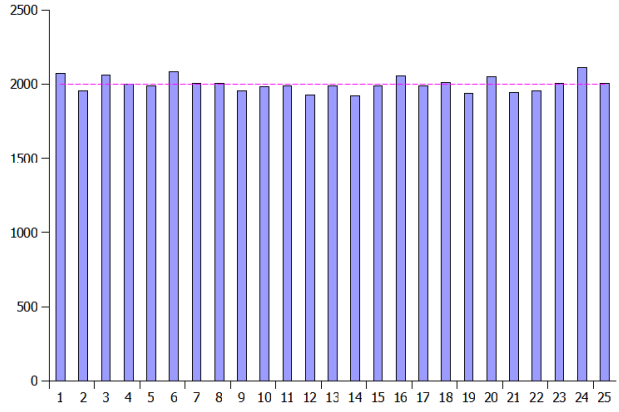
Moguće je seme inicijalizovati i vremenom na sledeći način

```
srand(time(NULL));
```

`time` funkcija se nalazi u zaglavlju `<time.h>` (`<ctime>`).

Histogram od 50000 vrednosti generisanih `rand` funkcijom

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
  int i;
  double r;
  srand(time(NULL));
  for(i=0; i<50000; ++i) {
    r = rand()/(RAND_MAX+1.0);
    printf("%f\n", r);
  }
  return 0;
                }
```



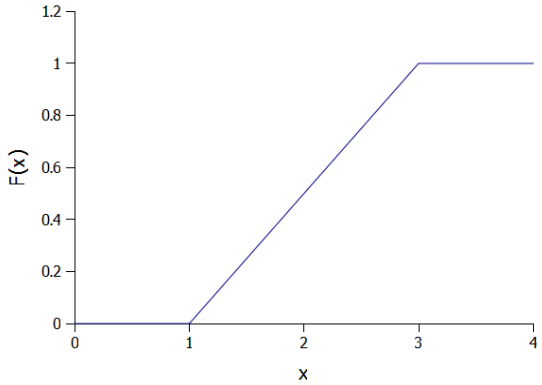
Uniformna raspodela

Gustina $f(x)=\frac{1}{b-a},\ a\le x\le b$, **Raspodela** $F(x)=\frac{x-a}{b-a}$, **Sr. vred.** $E[X]=(b-a)^{-1}$, **Varijansa** $\mathrm{Var}[X]=\frac{1}{12}(b-a)^2$

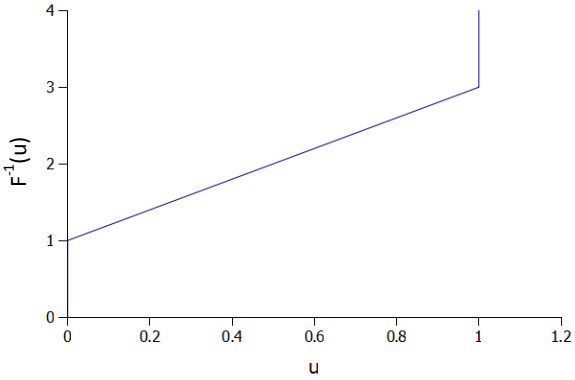
Nalaženje inverzne raspodele

$$u=F(x)\Rightarrow x=F^{-1}(u)$$

$$u=F(x)=\frac{x-a}{b-a}\Leftrightarrow x=a+(b-a)u$$



Uniformna raspodela



Inverzna uniformna raspodela

Eksponencijalna raspodela

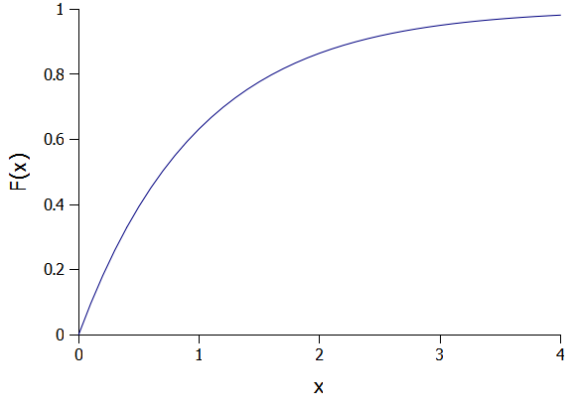
Gustina **Raspodela** **Sr. vred.** **Varijansa**

$f(x)=\lambda e^{-\lambda x},\ x>0$, $F(x)=1-e^{-\lambda x}$, $E[X]=\lambda^{-1}$, $\mathrm{Var}[X]=\lambda^{-2}$

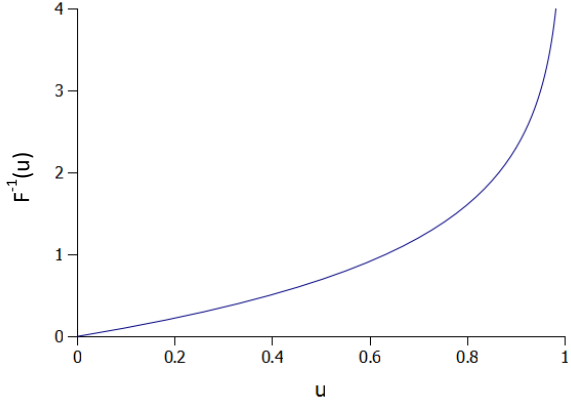
Nalaženje inverzne raspodele

$u=F(x)\Rightarrow x=F^{-1}(u)$

$u=F(x)=1-e^{-\lambda x} \Leftrightarrow x=-\frac{1}{\lambda}\ln(1-u) \overset{v=1-u}{\Rightarrow} x=-\frac{1}{\lambda}\ln v$



Eksponencijalna raspodela

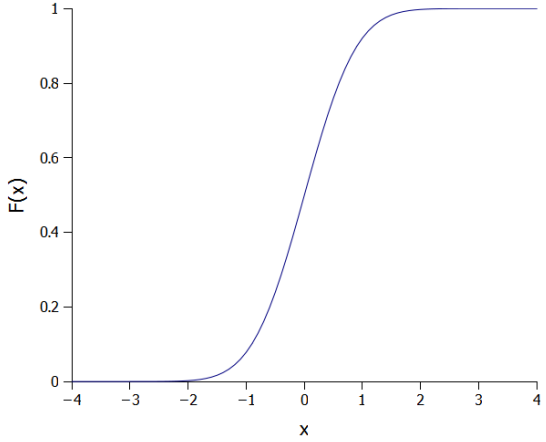


Inverzna eksponencijalna raspodela

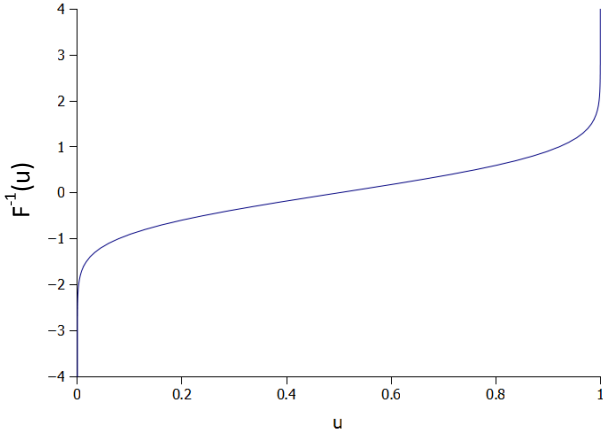
Normalna raspodela

Gustina **Raspodela** **Sr. vred.** **Varijansa**

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{(x-\mu)^2}{2\sigma^2}}\ ,\quad F(x)=\int_{-\infty}^{x}\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{(y-\mu)^2}{2\sigma^2}}dy=\frac{1}{2}\left[1+\operatorname{erf}\left(\frac{x-\mu}{\sqrt{2\sigma^2}}\right)\right]\ ,\quad E[X]=\mu\ ,\quad \operatorname{Var}[X]=\sigma^2$$



Normalna raspodela



Inverzna normalna raspodela

Obzirom da vrednosti iz normalne raspodele nije moguće generisati korišćenjem analitičkog izraza za inverznu funkciju raspodele u GPSS-u smo koristili metodu inverzne transformacije za generisanje normalne raspodele. Ovde ćemo pokazati kako se generišu vrednosti iz normalne raspodele korišćenjem metode sumiranja.

Metoda sumiranja

Ova metoda se koristi za generisanje normalne rspodele. Zasniva se na primeni centralne granične teoreme. Ona se formuliše na sledeći način. Neka je $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_n$ **niz nezavisnih slučajnih promenljivih sa jednakim verovatnoćama, svaka sa srednjom vrednošću** μ_X **i konačnom varijansom** σ_X^2**. Njihova srednja vrednost data je sledećim izrazom**

$$\overline{X} = \frac{1}{n} \sum_{i=1}^{n} \tilde{X}_i \, .$$

Tada novo usvojena promenljiva

$$Y = \frac{\bar{X} - \mu_X}{\sigma_X / \sqrt{n}}.$$

konvergira ka standardnoj normalnoj raspodeli sa srednjom vrednošću μ_X **i standardnoj devijaciji** $\sigma_X / \sqrt{n}$**, tj. za dovoljno veliko n razlika između promenljive i standardne normalne promenljive može se zanemariti iz praktičnih razloga.**

Prema tome da bi generisali uzorak iz standardne normalne raspodele možemo uzeti n nezavisnih uniformno raspodeljenih slučajnih brojeva $\tilde{r}_i \in [0,1)$ **sa srednjom vrednošću** $E(\tilde{r}_i) = \frac{1}{2}$ **i varijansom** $E(\tilde{r}_i - \mu)^2 = \frac{1}{12}$**. Slučajna promenljiva**

$$Z = \frac{\frac{1}{n}\sum_{i=1}^{n} \tilde{r}_i - \frac{1}{2}}{\sqrt{\frac{1}{12}} / \sqrt{n}} = \frac{\sum_{i=1}^{n} \tilde{r}_i - \frac{n}{2}}{\sqrt{\frac{n}{12}}}$$

ima normalnu raspodelu sa srednjom vrednošću 0 i varijansom 1 ($E[Z]=0$**,** $E\left[(Z-\mu)^2\right]=1$**), kada** $n\to\infty$**. Zadovoljavajući rezultati aproksimacije se dobijaju za** $n=12$ **cu kom gornja jednačina postaje**

$$Z=\sum_{i=1}^{12}\tilde{r}_i-6.$$

Ukoliko je potrebno generisati uzorke iz nestandardizovane normalne raspodele $\tilde{Z}$ **koristiti se sledeći izraz.**

$$\tilde{Z}=\mu+\sigma\cdot Z\,.$$

Iako je ova metoda jednostavna za realizaciju ona pati od nekoliko nedostataka. Prvi, i najočigledniji nedostatak je da je za jednu vrednost normalne promenjive potrebno generisati dvanaest uniformnih raspodeljenih slučajnih brojeva. To znači da će korišćenjem metode sumiranja veoma brzo istrošiti sekvenca uniformnih brojeva čijim korišćenjem se generiše normalna raspodela. Drugi problem sa ovom metodom je da se na ovaj način standardna

normalna raspodela ograničava na intervalu od [-6,6] iako je po definiciji neograničena.

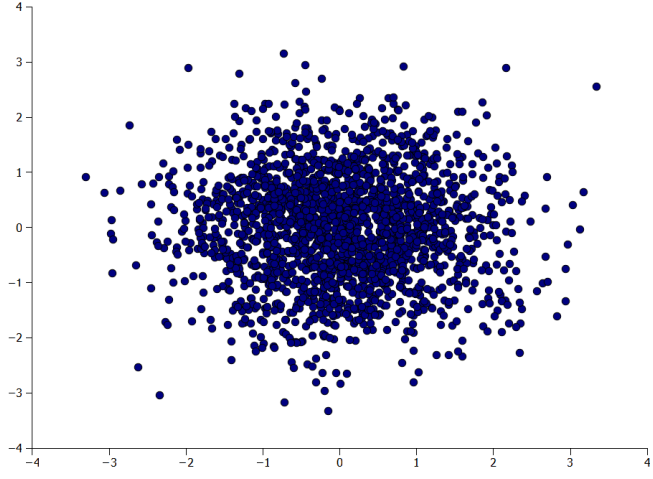
Pored metode inverzne transformacije i metode sumiranja postoje i druge metode za generisanje normalne raspodele. Ovde ćemo kao alternativu navesti Box-Muller-ovu, polarnu i Ziggurat metodu.

Dvodimenzionalna raspodela 2000 pseudoslučajnih parova generisanih `norm(0,1)` funkcijom

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

double norm(double m, double s) {
  int i;
  double z = 0.0;
  for(i=0; i<12; i++)
    z += rand()/(RAND_MAX+1.0);
  return z-6.0;
}

int main() {
  int i;
  double r;
  srand(time(NULL));
  for(i=0; i<2000; ++i)
    printf("%f,%f\n",norm(0,1),norm(0,1));
  return 0;
}
```

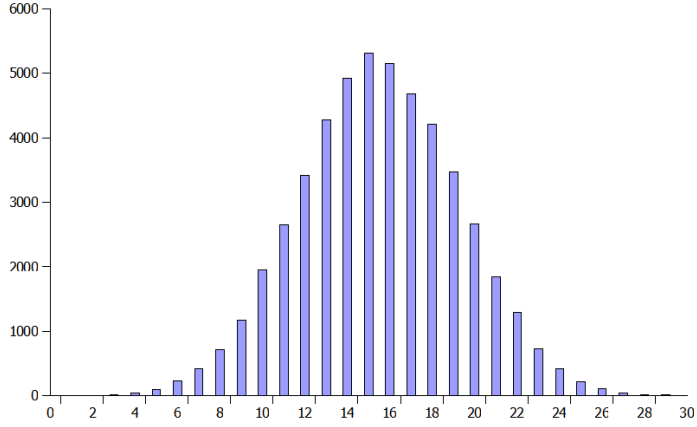


Histogram od 50000 vrednosti generisanih `norm(0,1)` funkcijom

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

double norm(double m, double s) {
  int i;
  double z = 0.0;
  for(i=0; i<12; i++)
    z += rand()/(RAND_MAX+1.0);
  return z-6.0;
}

int main() {
  int i;
  double r;
  srand(time(NULL));
  for(i=0; i<50000; ++i)
    printf("%f\n",norm(0,1));
  return 0;
}
```



Klasa raspodela

```
class raspodela {
  // Seme generatora slucajnih brojeva (GSB)
  unsigned int seme;
  // Multiplikator GSB
  unsigned int a;
  // Modulus GSB
  unsigned int m;
public:
  // Konstruktor
  raspodela(unsigned int s):seme(s) {
    // Postavljamo vrednost multiplikatora
    a = 16807;
    // Postavljamo vrednost modulusa
    m = 2147483647;
  }
  // Lehmerov multiplikativni kongruentni generator
  double operator() () {
    seme = (a*seme)%(m+1);
    return seme/(1.0*m));
  }
  // Uniformna raspodela
  double unif(double a,double b) {
    return a+(b-a)*(*this)();
  }
  // Eksponencijalna raspodela
  double expo(double m) {
    return -m*log((*this)());
  }
  // Normalna raspodela
  double norm(double m, double s) {
    double z = 0.0;
    for(int i=0; i<12; i++)
      z += (*this)();
```

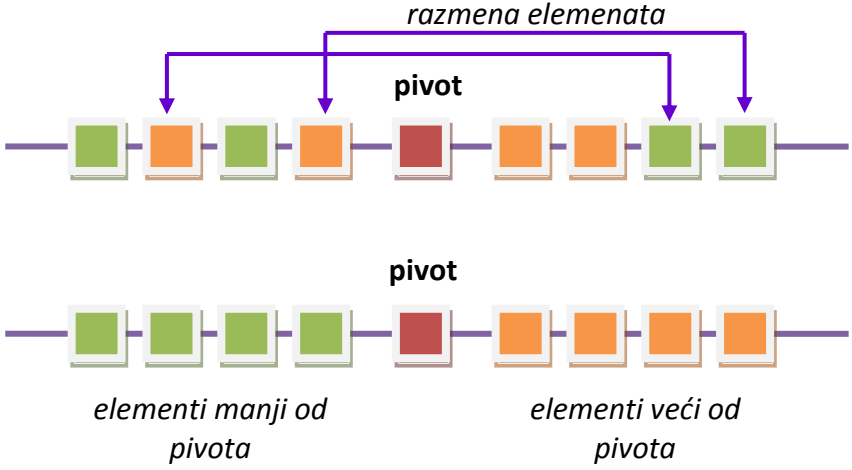
```
    return m+(z-6)*s;
  }
};
```

Klasa `raspodela` sadrži tri privatna člana `seme`, `a` i `m` (tipa `unsigned int`). `seme` je seme Lehmerovog generatora slučajnih brojeva. `a` je multiplikator Lehmerovog multiplikativnog GSB, dok je `m` modulus GSB-a. U konstruktoru se postavljaju vrednosti multiplikatora i modulusa i te vrednosti odgovaraju vrednostima Park-Millerovog minimalnog standardnog GSB-a (`a = 16807` i `m = 2147483647`). Konstruktoru se prosleđuje vrednost početnog semena GSB-a. U klasi postoje još četiri javno vidljive funkcije. Jedna je operatorska funkcija `operator()` čijim korišćenjem se vraća vrednost slučajnog broja korišćenjem Lehmerovog generatora. Ostale tri funkcije odgovaraju uniformnoj, eksponecijalnoj i normalnoj raspodeli. Ove tri funkcije koriste izraze koji su prethodno prikazani i izvedeni.

`this` je ključna reč koja predstavlja pokazivač na tekuću instancu klase u funkciji članice klase u kojoj se `this` koristi. `(*this)()` poziva `operator()` i vraća vrednost slučajnog broja. `log()` je funkcija prirodnog logaritma (zaglavlje `<cmath>`).

„Quick sort“ algoritam

„Quick sort“ algoritam (algoritam brzog sortiranja) je rekurzivni algoritam sortiranja elemenata polja. Odabira se jedan element polja, tzv. pivot i nakon toga se svi ostali elementi razvrstavaju levo ili desno od pivota u zavisnosti od toga da li su elementi manji ili veći od pivota. Oni elementi koji su veći od pivota, a nalaze se sa leve strane, zamenjuju se onima koji su manji od pivota, a nalaze se sa desne strane.



Nakon razmene elemenata polje i dalje nije sortirano. Da bi polje bilo sortirano do kraja potrebno je nezavisno sortirati levo i desno podpolje. Sortiranje podpolja se postiže rekurzivnim pozivima istog algoritma.

Znači, algoritam može biti podeljen u tri koraka:
1. *Izbor pivota*. Najčešće se središnji element u polju bira za pivota. U principu, pivot može biti bilo koja vrednost koja je u opsegu sortiranih vrednosti, čak i ako se ne nalazi kao element u polju.
2. *Raspoređivanje*. Elementi se raspoređuju tako da oni elementi koji su manji od pivota prelaze na levu stranu polja, dok oni elementi koji su veći od pivota se prebacuju na desnu stranu polja. Vrednosti koje su jednake pivotu mogu ostati u bilo kom delu polja.
3. *Rekurzivno sortiranje*. „Quick sort" algoritam se rekurzivno primenjuje na levu i desnu stranu polja.

***Detalji raspoređivanja unutar „quick sort" algoritma.* Postoje dva inteksa *i* i *j* na početku raspoređivanja i *i* ukazuje na prvi element polja, dok *j* ukazuje na poslednji element polja. Indeks *i* se uvećava sve dok se ne nađe element sa vrednošću većom ili jednakom vrednosti pivota. Indeks *j* se umanjuje sve dok**

se ne nađe element sa vrednošću manjom ili jednakom vrednosti pivota. ukoliko je $i \leq j$ ta dva elementa se međusobno razmenjuju, dok se *i* pomera na narednu poziciju (i+1), a *j* na prethodnu poziciju (*j*-1). Algoritam se zaustavlja kada *i* postane veće od *j*.

Nakon raspoređivanja sve vrednosti pre *i*-tog elementa su manje ili jednake pivotu i sve vrednosti nakon *j*-tog elementa su veće ili jednake pivotu.

Primer: Sortirati celobrojno polje 8, 1, 7, 6, 5, 8, 9, 1, 2 korišćenjem „quick sort“ agoritma.

Izabiramo pivota na sredini polja

8, 1, 7, 6, 5, 8, 9, 1, 2

Raspoređivanje

8, 1, 7, 6, 5, 8, 9, 1, 2	8 ≥ 5 ≥ 2 razmeni 8 i 2
2, 1, 7, 6, 5, 8, 9, 1, 8	7 ≥ 5 ≥ 1 razmeni 7 i 1
2, 1, 1, 6, 5, 8, 9, 7, 8	6 ≥ 5 ≥ 5 razmeni 6 i 5
2, 1, 1, 5, 6, 8, 9, 7, 8	zaustavljamo raspoređivanje

Rekurzivno sortiranje

Levo podpolje

2, 1, 1, 5

Izabiramo pivota

2, 1, 1, 5

Raspoređivanje

2, 1, 1, 5 2 ≥ 1 ≥ 1 razmeni 2 i 1

1, 1, 2, 5 zaustavljamo raspoređivanje

Desno podpolje

6, 8, 9, 7, 8

Izabiramo pivota

6, 8, 9, 7, 8

Raspoređivanje

6, 8, 9, 7, 8 9 ≥ 9 ≥ 8 razmeni 9 i 8

6, 8, 8, 7, 9 zaustavljamo raspoređivanje

Rekurzivno sortiranje

Levo podpolje

1, 1

Desno podpolje

2, 5

Levo podpolje

6, 8, 8, 7

Desno podpolje

9

itd.

Na kraju će sortirano polje glasiti

1, 1, 2, 5, 6, 7, 8, 8, 9

Sortiranje celobrojnog polja – „Quick sort“ algoritam

```
#include <iostream>
void quicksort(int x[],int l,int d) {
  int i = l, j = d, t;
  // Izbor pivota
  int p = x[(l+d)/2];
  // Rasporedjivanje
  while(i<=j) {
    // Pomeramo i dok ne naidjemo na element veci ili jednak pivotu
    while(x[i]<p) i++;
    // Pomeramo j dok ne naidjemo na element manji ili jednak pivotu
    while(x[j]>p) j--;
    // Ukoliko je i<=j
    if(i<=j) {
      // Razmena elementa (swap)
      t = x[i];
      x[i] = x[j];
      x[j] = t;
      // Pomeramo  i i j na naredni, odnosno, prethodni element
      i++; j--;
    }
  };
  // Rekurzivni pozivi algoritma sortiranja
  // nad levim i desnim podpoljem
  if(l<j) quicksort(x,l,j);
  if(d>i) quicksort(x,i,d);
}
int main() {
  int x [] = {8, 1, 7, 6, 5, 8, 9, 1, 2 };
  int n = sizeof(x)/sizeof(int);
  quicksort(x,0,n-1);
  for(int i=0; i<n; i++)
    std::cout << x[i] << std::endl;
  return 0;
}
```

Klasa simulacija

```
class simulacija {
  … // Isto kao u SRD1
  void quicksort(entitet* x[],int l,int d) {
    int i = l, j = d;
    entitet* t;
    // Izbor pivota
    entitet* p = x[(l+d)/2];
    // Rasporedjivanje
    while(i<=j) {
      // Pomeramo i dok ne naidjemo na element veci ili jednak pivotu
      while(x[i]->vreme_nastupanja_dogadjaja < p->vreme_nastupanja_dogadjaja)
        i++;
      // Pomeramo j dok ne naidjemo na element manji ili jednak pivotu
      while(x[j]->vreme_nastupanja_dogadjaja > p->vreme_nastupanja_dogadjaja)
        j--;
      if(i<=j) {
        // Razmena
        t = x[i];
        x[i] = x[j];
        x[j] = t;
        // Pomeramo se na naredni element
        i++; j--;
      }
    };
    // Rekurzivni pozivi algoritma sortiranja nad levim i desnim podpoljem
    if(l<j) quicksort(x,l,j);
    if(i<d) quicksort(x,i,d);
  }
  void sortiraj_listu_nastupanja() {
    quicksort(lista_nastupanja_entiteta,0,broj-1);
  }
  … // Isto kao u SRD1
};
```

Klasa `simulacija` je uglavnom ista kao i klasa `simulacija` iz prethodne verzije programa (vidi Strategija raspoređivanja događaja 1). Ono što je promenjeno u odnosu na prethodnu verziju to da je dodata privatna funkcija `void quicksort(entitet*,int,int)` koja se poziva iz postojeće privatne funkcije `void sortiraj_listu_nastupanja()`.

Primer 2. Posmatrajmo poštu u kojoj se nalazi četiri univerzalna šaltera ispred koga se formira zajednički red čekanja. Vreme opsluge je eksponencijalno raspodeljeno sa srednjim vremenom 200 sekundi.

Klijenti dolaze po puasonovom toku sa srednjim vremenom od 100 sekundi.

Simulirati rad pošte za period od 12 časova. Vremenska jedinica je 1 sekunda. Takođe, potrebno je izračunati statistike resursa i reda čekanja.

Bezuslovni događaji u ovom primeru su gotovo isti kao i u primeru 1. Ponovo postoje dva bezuslovna događaja *DolazakKlijenta* i *OdlazakKlijenta*. Pseudokodovi ova dva događaja su isti kao i u prethodnom primeru. Pored ova dva događaja uvodi se i treći događaj koji se aktivira nakon isteka vremena simulacije i služi da prekine simulaciju. Ovaj događaj nosi naziv *KrajSimulacije*. Pseudokod tog dođaja je jednostavan i on glasi:

procedure *KrajSimulacije*
begin
    Zaustavi simulaciju.
end

Za realizaciju primera koristićemo zaglavlje simu2.h. Zaglavlje simu2.h je jako slično zaglavlju simu1.h, osim što je dodata klasa `raspodela` i promenjena klasa `simulacija` (funkcija `quicksort`). Takođe se menja i početak datoteke, na taj način da se pored `<iostream>` i `<cstdlib>`, uvozi i zaglavlje `<cmath>`. Isto tako se menja i naziv prostora imena u `simu2`.

```
// Zaglavlje simu2.h
#include <iostream>
#include <cstdlib>
#include <cmath>

// Koristimo prostor imena STL biblioteke
using namespace std;

// Prostor imena simu2
namespace simu2 {
  // Realizicije klasa raspodela, entitet, fifo_red, resurs i simulacija
  // ...
}
```

Takođe, iz klase `fifo_red` i `simulacija` izbačene su funkcije `stampa` i `stampaj_listu`.

simu2.cpp

U ovoj izvornoj datoteci neće biti mnogo izmena u odnosu na simu1.cpp. Kao i u simu1.cpp, na početku se učitavaju odgovarajuća zaglavlja i dozvoljava se korišćenje odgovarajućih prostora imena.

```
#include "simu2.h"  // Treba nam zbog raspodela, entitet, fifo_red, resurs i
                    // simulacija
#include <iostream> // Treba nam zbog cout

// Ukljucujemo prostore imena simu2 i std
using namespace simu2;
using namespace std;
```

Zatim postavljamo redne brojeve događaja i potpise funkcija koje realizuju te bezuslovne događaje. Nakon toga realizujemo funkciju `simulacija::izvrsavanje_dogadjaja`.

```
// Redni broj dogadjaja DolazakKlijenta
const int DOLAZAK_KLIJENTA = 0;
// Redni broj dogadjaja OdlazakKlijenta
const int ODLAZAK_KLIJENTA = 1;
// Redni broj dogadjaja KrajSimulacije
const int KRAJ_SIMULACIJE  = 2;

// Potpis funkcije bezuslovnog dogadjaja DolazakKlijenta
void dolazak_klijenata(entitet *e);
// Potpis funkcije bezuslovnog dogadjaja OdlazakKlijenta
void odlazak_klijenata(entitet *e);
// Potpis funkcije bezuslovnog dogadjaja KrajSimulacije
void kraj_simulacije(entitet *e);

// Izvrsavanje dogadjaja
void simulacija::izvrsavanje_dogadjaja(int dog, entitet* e) {
  switch(dog) {
    case DOLAZAK_KLIJENTA:
      dolazak_klijenata(e);
      break;
    case ODLAZAK_KLIJENTA:
      odlazak_klijenata(e);
      break;
    case KRAJ_SIMULACIJE:
      kraj_simulacije(e);
  }
}
```

Dalje, kreiramo objekte raspodele, reda čekanja, četiri šatera i simulacije koji će kontrolisati izvršenje simulacije.

```
// Objekat reda cekanja
fifo_red red;
// Salteri
resurs salteri(4);
// Objekat simulacije
simulacija simu;
// Raspodele
raspodela rn1(11615), rn2(36517);
```

Takođe, uvodimo i dve konstante koje će predstavljati srednje vreme dolaska i srednje vreme opsluge entiteta.

```
// Srednje vreme dolaska entiteta
const double SRVRDOL = 100;
// Srednjevreme opsluge entiteta
const double SRVROPS = 200;
```

Događaji *DolazakKlijenta* i *OdlazakKlijenta* su gotovo isti kao i u primeru 1. Jedino se menja raspoređivanje događaja obzirom da vremena ne uzimamo iz polja već vreme dobijamo korišćenjem objekata klase raspodela.

U pozivima funkcije rasporedi

```
// Postavljamo vreme narednog dolaska
simu.rasporedi(ne,DOLAZAK_KLIJENTA,
                  simu.vrati_vreme_simulacije()+rn1.expo(SRVRDOL));

// Rasporedi klijenta za kraj opsluge
simu.rasporedi(fe,ODLAZAK_KLIJENTA,
                  simu.vrati_vreme_simulacije()+rn2.expo(SRVROPS));
```

`rn1.expo(SRVRDOL)` i `rn2.expo(SRVROPS)` vraćaju vreme između dva dolaska entiteta i vreme trajanja opsluge. Vremenski trenuci nastupanja događaja dolaska novog entiteta i događaja završetka opsluge dobijaju se kada se na vreme između dolazaka i vreme opsluge doda vreme simulacionog časovnika `simu.vrati_vreme_simulacije()`.

Pored događaja *DolazakKlijenta* i *OdlazakKlijenta* potrebno realizovati i bezuslovni događaj *KrajSimulacije*. U funkciji `kraj_simulacije` pozivamo funkciju klase `simulacija unisti_entitet` kojoj se prosleđuje entitet za koji se izvršava događaj i brojna vrednost za koju se umanjuje terminacioni brojač. Obzirom da ćemo pozivom funkcije `simulacija::izvrsi` postaviti vrednost terminacionog brojača na 1, i obzirom da se terminacioni brojač umanjuje za 1, jasno je da će prvo izvršenje događaja *KrajSimulacije* dovesti do završetka simulacije.

```
// Dogadjaj kraj simulacije
void kraj_simulacije(entitet* e) {
  simu.unisti_entitet(e,1);
}
```

```
procedure KrajSimulacije
begin
      Zaustavi simulaciju.
end
```

main funkcija

Kao što smo i u prethodnom primeru rekli da bi simulacija uopšte mogla da započne neophodno je napraviti i rasporediti prvi entitet za događaj dolaska klijenta. Postoje dva načina raspoređivanja prvog entiteta. Prvi način je da se prvi entitet rasporedi na događaj dolaska u trenutku *t*=0, dok je drugi način, da se dolazak prvog entiteta rasporedi na vreme koje se dobija iz raspodele vremena dolaska (u našem slučaju to je eksponecijalna raspodela sa srednjim vremenom 100 sek). Mi ćemo za raspoređivanje dolaska prvog entiteta koristiti drugi način:

```
// Stvaramo prvog klijenta
entitet *e1 = simu.napravi_entitet();

// Postavljamo vreme dolaska prvog klijenta
simu.rasporedi(e1,DOLAZAK_KLIJENTA,rn1.expo(SRVRDOL));
```

Takođe potrebno je na početku potrebno rasporediti i entitet koji će dovesti do završetka simulacije, odnosno koji će izvršiti događaj *KrajSimulacije*.

```
// Stvaramo entitet za kraj simulacije
entitet *e2 = simu.napravi_entitet();

// Postavljamo vreme zavrsetka simulacije
simu.rasporedi(e2,KRAJ_SIMULACIJE,43200);
```

Funkcija `main` će glasiti

```
int main() {
  // Stvaramo prvog klijenta
  entitet *e1 = simu.napravi_entitet();
  // Postavljamo vreme dolaska prvog klijenta
  simu.rasporedi(e1,DOLAZAK_KLIJENTA,rn1.expo(SRVRDOL));
  // Stvaramo entitet za kraj simulacije
  entitet *e2 = simu.napravi_entitet();
  // Postavljamo vreme zavrsetka simulacije
  simu.rasporedi(e2,KRAJ_SIMULACIJE,43200);
  // Izvrsavamo simulaciju
  simu.izvrsi(1);
}
```

Štampanje izveštaja

Da bi smo mogli da izračunamo statistike neophodno je zabeleziti trenutke u kojima dolazi do promene stanja sistema. Dva stanja pratimo i to su dužina reda čekanja i broj zauzetih kanala opsluge u resursu. Promene koje se dešavaju su sledeće: ulazak u red čekanja (ured), izlazak iz reda čekanja (izred), ulazak u resurs (usal) i izlazak iz resursa (izsal). Takođe je potrebno zabeležiti i trenutak prekida simulacije, tj. vreme simulacije (vsim). Pored vrste promene i vremena nastupanja potrebno je zabeležiti i redni broj entiteta koji dovodi do promene stanja.

```
cout << "ured" << "," << e->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;

cout << "izred" << "," << fe->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;

cout << "usal" << "," << fe->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;

cout << "izsal" << "," << e->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;

cout << "vsim" << "," << -1 << "," << simu.vrati_vreme_simulacije() << endl;
```

Kompajliranje i izvršenje programa

```
cl simu2.cpp /EHsc

simu2 > simu2.txt
```

Zaglavlje simu2.h

```
// Zaglavlje simu2.h
#include <iostream>
#include <cstdlib>
#include <cmath>

// Koristimo prostor imena STL biblioteke
using namespace std;

// Prostor imena simu2
namespace simu2 {
  // Konstante
  const int max_broj_uredu  = 10;
  const int max_broj_ulisti = 50;
  // Klasa raspodela
  class raspodela {
    // Seme generatora slucajnih brojeva (GSB)
    unsigned int seme;
    // Multiplikator GSB
    unsigned int a;
    // Modulus GSB
    unsigned int m;
  public:
    // Konstruktor
    raspodela(unsigned int s = 1):seme(s) {
      // Postavljamo vrednost multiplikatora
      a = 16807;
      // Postavljamo vrednost modulusa
      m = 2147483647;
    }
    // Lehmerov multiplikativni kongruentni generator
    double operator() () {
      seme = (a*seme)%(m+1);
      return seme/(1.0*m);
```

```
  }
  // Uniformna raspodela
  double unif(double a,double b) {
    if(b>a)
      return a+(b-a)*(*this)();
    else {
      cerr << "Gornja granica mora da bude veca od donje" << endl;
      exit(1);
    }
  }
  // Eksponencijalna raspodela
  double expo(double m) {
    if(m>0)
      return -m*log((*this)());
    else {
      cerr << "Ocekivanje mora da bude vece od nule" << endl;
      exit(1);
    }
  }
  // Normalna raspodela
  double norm(double m, double s) {
    double z = 0.0;
    if(s>0) {
      for(int i=0; i<12; i++)
        z += (*this)();
      return m+(z-6)*s;
    }
    else {
      cerr << "Devijacija ne sme da bude manja ili jednaka nuli" << endl;
      exit(1);
    }
  }
};
// Klasa entiteta
class entitet {
  // Omogucavamo klasi simulacija da pristupi privatnim clanovima klase entitet
  friend class simulacija;
```

```
  // Omogucavamo klasi fifo_red da pristupi privatnim clanovima klase entitet
  friend class fifo_red;
  // Redni broj entiteta
  int id;
  // Buduci dogadjaj
  int naredni_dogadjaj;
  // Vreme nastupanja buduceg dogadjaja
  double vreme_nastupanja_dogadjaja;
public:
  // Konstruktor
  entitet(int eid):id(eid) { }
  // Vraca id entiteta
  int vrati_id() { return id; }
};
// Klasa FIFO red cekanja
class fifo_red {
  // Broj entiteta u redu cekanja
  int broj;
  // Polje pokazivaca na entitete
  entitet* red[max_broj_uredu];
public:
  // Konstruktor
  fifo_red(): broj(0) {}
  // Smestamo entitet u red cekanja
  void ured(entitet* e) {
    if( broj < max_broj_uredu) {
      // Smestamo entitet na kraj reda cekanja
      red[broj]=e;
      // Uvecavamo broj entiteta za 1
      broj++;
    }
    else {
      cerr << "Previse korisnika u redu cekanja" << endl;
      exit(1);
    }
  }
  // Vadimo entitet iz reda cekanja
```

```
  entitet* izred() {
    entitet *e;
    if(broj>0) {
      // Vadimo prvi entitet iz reda cekanja
      e = red[0];
      for(int i=1;i<broj;i++)
        red[i-1] = red[i];
      // Umanjujemo broj entiteta iz reda cekanja
      broj--;
    }
    else {
      cerr << "Nema entiteta u redu cekanja" << endl;
      exit(1);
    }
    // Vraca entitet
    return e;
  }
  // Vracamo velicinu reda cekanja
  int velicina() { return broj; };
  void stampa() {
    for(int i=0; i<broj; i++)
      cout << red[i]->id << ",";
  }
};
// Klasa resursa
class resurs {
  // Tekuci broj zauzetih mesta u resursa
  int broj_zauzetih_mesta;
  // Maksimalni broj  mesta u resursu
  int ukupni_broj_mesta;
public:
  // Konstruktor
  resurs(int b):ukupni_broj_mesta(b),broj_zauzetih_mesta(0) {}
  // Zauzima mesto u resursu
  void zauzmi() {
    if(broj_zauzetih_mesta<ukupni_broj_mesta)
      // Uvecavamo broj zauzetih mesta
```

```
      broj_zauzetih_mesta++;
    else {
      cerr << "Sva mesta u resursu su zauzeta" << endl;
      exit(1);
    }
  }
  // Oslobadja mesto u resursu
  void oslobodi() {
    if(broj_zauzetih_mesta>0)
      // Umanjujemo broj zauzetih mesta
      broj_zauzetih_mesta--;
    else {
      cerr << "Nema entiteta u resursu" << endl;
      exit(1);
    }
  }
  // Raspolozivost resursa (ima makar jedno slobodno mesto u resursu)
  bool raspoloziv() { return (broj_zauzetih_mesta<ukupni_broj_mesta); }
};

// Klasa simulacija
class simulacija {
  // Brojac entiteta
  int eid;
  // Terminacioni brojac
  int tb;
  // Broj entiteta u LBD listi
  int broj;
  // Vreme simulacije
  double vreme_simulacije;
  // Polje entiteta u LBD listi
  entitet* lista_nastupanja_entiteta[max_broj_ulisti];
  // Algoritam brzog sortiranja
  void quicksort(entitet* x[],int l,int d) {
    int i = l, j = d;
    entitet* t;
    // Izbor pivota
```

```
    entitet* p = x[(l+d)/2];
    // Rasporedjivanje
    while(i<=j) {
      // Pomeramo i dok ne naidjemo na element veci ili jednak pivotu
      while(x[i]->vreme_nastupanja_dogadjaja < p->vreme_nastupanja_dogadjaja)
        i++;
      // Pomeramo j dok ne naidjemo na element manji ili jednak pivotu
      while(x[j]->vreme_nastupanja_dogadjaja > p->vreme_nastupanja_dogadjaja)
        j--;
      // Ukoliko je i<=j
      if(i<=j) {
        // Razmena elemenata (swap)
        t = x[i];
        x[i] = x[j];
        x[j] = t;
        // Pomeramo  i i j na naredni, odnosno, prethodni element
        i++; j--;
      }
    };
    // Rekurzivni pozivi algoritma sortiranja nad levim i desnim podpoljem
    if(l<j) quicksort(x,l,j);
    if(i<d) quicksort(x,i,d);
  }
  // Sortiranje LBD - Sortiranje izborom
  void sortiraj_listu_nastupanja() {
    quicksort(lista_nastupanja_entiteta,0,broj-1);
  }
public:
  // Konstruktor
  simulacija():eid(1),tb(0),broj(0),vreme_simulacije(0.0) {}
  // Destruktor
  ~simulacija() {
    // Uklanjamo preostale entitete iz LBD na kraju simulacije.
    for(int i=0; i<broj; i++)
      delete lista_nastupanja_entiteta[i];
  }
  // Pravljenje entiteta
```

```
entitet* napravi_entitet() {
  // Pravimo novi entitet
  return new entitet(eid++);
}
// Unistavanje entiteta
void unisti_entitet(entitet *e, int b) {
  // Unistavamo entitet
  delete e;
  // Umanjujemo terminacioni brojac
  tb -= b;
}
// Rasporedjivanje dogadjaja
void rasporedi(entitet* e,int dog,double vreme) {
  if( broj < max_broj_ulisti) {
    // Postavljamo dogadjaj
    e->naredni_dogadjaj = dog;
    // Postavljamo vreme nastupanja dogadjaja
    e->vreme_nastupanja_dogadjaja = vreme;
    // Smestamo entitet u listu
    lista_nastupanja_entiteta[broj] = e;
    broj++;
    // Sortiramo LBD
    sortiraj_listu_nastupanja();
  }
  else {
    cerr << "Previse entiteta u listi dogadjaja" << endl;
    exit(1);
  }
}
// Izvrsavanje simulacije
void izvrsi(int b) {
  entitet * tekuci;
  // Postavljamo terminacioni brojac
  tb = b;
  // Izvrsavamo simulaciju. Simulacija se zavrsava ukoliko
  // nema entiteta u LBD ili ukoliko je terminacioni brojac 0.
  do {
```

```
        // Vadimo prvi entitet iz liste. Taj entitet postaje tekuci entitet.
        tekuci = lista_nastupanja_entiteta[0];
        for(int i=1;i<broj;i++)
          lista_nastupanja_entiteta[i-1] = lista_nastupanja_entiteta[i];
        broj--;
        // Faza 1: Azurirarmo vreme simulacije
        vreme_simulacije = tekuci->vreme_nastupanja_dogadjaja;
        // Faza 2: Izvrsavamo dogadjaj
        izvrsavanje_dogadjaja(tekuci->naredni_dogadjaj,tekuci);
      } while(broj && tb>0);
    }
    // Izvrsava dogadjaj
    void izvrsavanje_dogadjaja(int dog, entitet* e);
    // Vraca vreme simulacije
    double vrati_vreme_simulacije() { return vreme_simulacije; }
  };
}
```

Izvorna datoteka simu2.cpp

```
// Datototeka simu2.cpp
#include "simu2.h"
#include <iostream>

// Ukljucujemo prostore imena simu2 i std
using namespace simu2;
using namespace std;

// Redni broj dogadjaja DolazakKlijenta
const int DOLAZAK_KLIJENTA = 0;
// Redni broj dogadjaja OdlazakKlijenta
const int ODLAZAK_KLIJENTA = 1;
// Redni broj dogadjaja KrajSimulacije
const int KRAJ_SIMULACIJE  = 2;

// Potpis funkcije bezuslovnog dogadjaja DolazakKlijenta
void dolazak_klijenta(entitet *e);
// Potpis funkcije bezuslovnog dogadjaja OdlazakKlijenta
void odlazak_klijenta(entitet *e);
// Potpis funkcije bezuslovnog dogadjaja KrajSimulacije
void kraj_simulacije(entitet *e);

// Izvrsavanje dogadjaja
void simulacija::izvrsavanje_dogadjaja(int dog, entitet* e) {
  switch(dog) {
    case DOLAZAK_KLIJENTA:
      dolazak_klijenta(e);
      break;
    case ODLAZAK_KLIJENTA:
      odlazak_klijenta(e);
      break;
    case KRAJ_SIMULACIJE:
      kraj_simulacije(e);
  }
```

```
}

// Objekat reda cekanja
fifo_red red;
// Salteri
resurs salteri(4);
// Objekat simulacije
simulacija simu;
// Raspodele
raspodela rn1(11615), rn2(36517);
// Srednje vreme dolaska entiteta
const double SRVRDOL = 100;
// Srednjevreme opsluge entiteta
const double SRVROPS = 200;

// Dogadjaj dolazak klijenta
void dolazak_klijenta(entitet *e) {
  entitet *fe, *ne;
  cout << "ured" << "," << e->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;
  // Postavi klijenta u red cekanja
  red.ured(e);
  // Ukoliko je salter raspoloziv
  if(salteri.raspoloziv()) {
    // Izvadi prvog klijenta iz reda
    fe = red.izred();
    cout << "izred" << "," << fe->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;
    // Zauzmi jedno mesto u salteru
    salteri.zauzmi();
    cout << "usal" << "," << fe->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(fe,ODLAZAK_KLIJENTA,simu.vrati_vreme_simulacije()+rn2.expo(SRVROPS));
  }
  // Stvaramo narednog klijenta
  ne = simu.napravi_entitet();
  // Postavljamo vreme narednog dolaska
  simu.rasporedi(ne,DOLAZAK_KLIJENTA,simu.vrati_vreme_simulacije()+rn1.expo(SRVRDOL));
}
```

```
// Dogadjaj odlazak klijenta
void odlazak_klijenta(entitet *e) {
  entitet *fe;
  // Vrati salter;
  salteri.oslobodi();
  cout << "izsal" << "," << e->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;
  // Klijent odlazi iz poste - unistavamo tekuceg klijenta
  simu.unisti_entitet(e,0);
  // Ukoliko red nije prazan
  if(red.velicina()>0) {
    // Izvadi prvog klijenta iz reda
    fe = red.izred();
    cout << "izred" << "," << fe->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;
    // Zauzmi jedno mesto u salteru
    salteri.zauzmi();
    cout << "usal" << "," << fe->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(fe,ODLAZAK_KLIJENTA,simu.vrati_vreme_simulacije()+rn2.expo(SRVROPS));
  }
}
// Dogadjaj kraj simulacije
void kraj_simulacije(entitet* e) {
  simu.unisti_entitet(e,1);
}
int main() {
  // Stvaramo prvog klijenta
  entitet *e1 = simu.napravi_entitet();
  // Postavljamo vreme dolaska prvog klijenta
  simu.rasporedi(e1,DOLAZAK_KLIJENTA,rn1.expo(SRVRDOL));
  // Stvaramo entitet za kraj simulacije
  entitet *e2 = simu.napravi_entitet();
  // Postavljamo vreme zavrsetka simulacije
  simu.rasporedi(e2,KRAJ_SIMULACIJE,43200);
  // Izvrsavamo simulaciju
  simu.izvrsi(1);
  // Stampamo vreme simulacije
  cout << "vsim" << "," << -1 << "," << simu.vrati_vreme_simulacije() << endl; }
```

Statistike simulacionog programa

Statistike resursa

$R(t)$, **trenutni broj klijenata u resursu**

J_R , **broj promena stanja u resursu u toku simulacije**

t_R^i , **vremenski trenuci u kojima se menja stanje resursa**

N_R , **broj entiteta koji je ušao u resurs u toku simulacije**

n_R , **broj kanala opsluge**

W_R^i , **vreme opsluge *i*-tog entiteta**

T , **vreme simulacije**

1. Srednji broj zauzetih kanala: $\overline{R} = \frac{1}{T}\int_0^T R(\tau)\,d\tau = \frac{1}{T}\left[\sum_{i=1}^{J_R-1} R\left(t_R^i\right)\left(t_R^{i+1} - t_R^i\right) + R\left(t_R^{J_R}\right)\left(T - t_R^{J_R}\right)\right]$

2. Srednje vreme opsluge: $\overline{W}_R = \frac{\sum_{i=1}^{N_R - R(T)} W_R^i}{N_R - R(T)}$

3. Iskorišćenost: $\rho = \frac{\overline{R}}{n_R}$

Statistike reda čekanja

$Q(t)$, **trenutni broj klijenata u redu čekanja**

J_Q , **broj promena stanja u redu čekanja u toku simulacije**

t_Q^i , **vremenski trenuci u kojima se menja stanje reda čekanja**

N_Q , **broj entiteta koji je ušao u red čekanja u toku simulacije**

N'_Q , **broj entiteta koji je ušao u red čekanja, a nije se zadržavao u redu**

W_Q^i , **vreme čekanja *i*-tog entiteta**

T , **vreme simulacije**

1. Srednja dužina reda čekanja: $\bar{Q} = \frac{1}{T}\int_0^T Q(\tau)\,d\tau = \frac{1}{T}\left[\sum_{i=1}^{J_Q-1} Q\left(t_Q^i\right)\left(t_Q^{i+1} - t_Q^i\right) + Q\left(t_Q^{J_Q}\right)\left(T - t_Q^{J_Q}\right)\right]$

2. Srednje vreme čekanja: $\bar{W}_Q = \frac{\sum_{i=1}^{N_Q - Q(T)} W_Q^i}{N_Q - Q(T)}$

3. Srednje vreme čekanja u redu isključujući one entitete koji se nisu zadržavali u redu $\bar{W}'_Q = \frac{\sum_{i=1}^{N_Q - Q(T)} W_Q^i}{N_Q - N'_Q - Q(T)}$

Parametri SMO M/M/n/∞

λ , **intenzitet dolazaka**

μ , **intenzitet opsluživanja**

1. Verovatnoća da je sistem prazan: $P_0 = \left[\sum_{j=1}^{n-1} \left(\frac{\lambda}{\mu} \right)^j \frac{1}{j!} + \frac{1}{n!} \left(\frac{\lambda}{\mu} \right)^n \frac{n\lambda}{n\mu - \lambda} \right]^{-1}$

2. Srednja dužina reda čekanja: $\overline{k}_r = \frac{P_0 \lambda \mu (\lambda/\mu)^n}{(n-1)!(n\mu - \lambda)^2}$

3. Srednje vreme čekanja u redu: $\overline{t}_r = \frac{\overline{k}_r}{\lambda}$

4. Srednje vreme opsluge: $\overline{t}_o = \frac{1}{\mu}$

5. Iskorišćenost: $\rho = \frac{\lambda}{n\mu}$

Listing simulacionog programa realizovanog u GPSS/H

```
  Student GPSS/H Release 3.70 (EC052)      5 Feb 2013   00:31:39     File: simu2.gps

    Line# Stmt#  If Do  Block#  *Loc Operation      A,B,C,D,E,F,G  Comments

        1     1                        SIMULATE
        2     2                        STORAGE S(1),4
        3     3              1         GENERATE RVEXPO(3,100)
        4     4              2         QUEUE 1
        5     5              3         ENTER 1
        6     6              4         DEPART 1
        7     7              5         ADVANCE RVEXPO(7,200)
        8     8              6         LEAVE 1
        9     9              7         TERMINATE
       10    10                 *
       11    11              8         GENERATE 43200
       12    12              9         TERMINATE 1
       13    13                        START 1
       14    14                        END

Entity Dictionary (in ascending order by entity number; "*" => value conflict.)

           Queues: 1

         Storages: 1

   Random Numbers: 3               7

Symbol    Value    EQU Defns  Context      References by Statement Number

1             1               Queue            4     6

1             1            2  Storage          5     8
```

```
3             3              Random Nmbr      3
7             7              Random Nmbr      7


Storage Requirements (Bytes)

Compiled Code:       336
Compiled Data:        80
Miscellaneous:         0
Entities:            896
Common:            10000
-----------------------
Total:             11312


GPSS/H Model Size:

Control Statements       4
Blocks                   9


Simulation begins.

Relative Clock: 43200.0000   Absolute Clock: 43200.0000


Block Current     Total
1                   409
2                   409
3                   409
4                   409
5           1       409
```

6	408
7	408
8	1
9	1

Storage	--Avg-Util-During-- Total Time	Avail Time	Unavl Time	Entries	Average Time/Unit	Current Status	Percent Avail	Capacity	Average Contents	Current Contents	Maximum Contents
1	0.478			409	202.096	AVAIL	100.0	4	1.913	1	4

Queue	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Unit	$Average Time/Unit	Qtable Number	Current Contents
1	5	0.109	409	354	86.6	11.496	85.491		0

Random Stream	Antithetic Variates	Initial Position	Current Position	Sample Count	Chi-Square Uniformity
3	OFF	300000	300410	410	0.69
7	OFF	700000	700409	409	0.84

Status of Common Storage

9448 bytes available
552 in use
1528 used (max)

Simulation complete. Absolute Clock: 43200.0000

```
Total Block Executions: 2863

Blocks / second:        9067207

Microseconds / Block:   0.11


Elapsed Time Used (Sec)

Pass1:            0.00
Sym/Xref          0.00
Pass2:            0.00
Load/Ctrl:        0.00
Execution:        0.00
Output:           0.00
---------------------
Total:            0.00
```

Rezultati dobijeni u programskom jeziku C++

```
Vreme simulacije: 43200.000

STATISTIKE REDA CEKANJA
Broj entiteta koji je usao u red cekanja: 428
Preostali broj entiteta u redu cekanja: 0
Maksimalni broj entiteta u redu: 5
Srednje vreme cekanja u redu: 18.254
Srednji broj entiteta u redu: 0.252
Broj entiteta koji je prosao kroz red bez zadrzavanja: 348
Procenat ulaza bez zadrzavanja: 81.308
Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu zadrzali): 97.661

STATISTIKE RESURSA
Broj entiteta koji je usao u resurs: 428
Preostali broj entiteta u resursu: 1
Maksimalni broj zauzetih kanala opsluge: 4
Srednje vreme opsluge: 214.647
Srednji broj zauzetih kanala: 2.128
Iskoriscenost: 0.532
```

Uporedne statistike reda čekanja i resursa (skladišta) dobijene simulacionim programom realizovanim u GPSS/H i C++

Uporedne statistike reda čekanja

Statistike reda čekanja	GPSS/H	C++	Analitičke vrednosti
Broj entiteta koji je ušao u red čekanja	409	428	432
Preostali broj entiteta u redu čekanja	0	0	-
Maksimalni broj entiteta u redu	5	5	-
Srednje vreme čekanja u redu	11.496	18.254	16.667
Srednji broj entiteta u redu	0.109	0.252	0.167
Broj entiteta koji je prošao kroz red bez zadržavanja	354	348	-
Procenat ulaza bez zadržavanja	86.6	81.308	-
Srednje vreme čekanja u redu (isključ. ent. koji se nisu zadržali)	85.491	97.661	-

Uporedne statistike resursa (skladišta)

Statistike resursa (skladišta)	GPSS/H	C++	Analitičke vrednosti
Broj entiteta koji je ušao u resurs	409	428	432
Preostali broj entiteta u resursu	1	1	-
Maksimalni broj zauzetih kanala opsluge	4	4	-
Srednje vreme opsluge	202.096	214.647	200.000
Srednji broj zauzetih kanala	1.913	2.129	2.167
Iskorišćenost	0.478	0.532	0.500

Kompajliranje i izvršenje programa

```
cl stat_simu2.cpp /EHsc

stat_simu2
```

Program stat_simu2.cpp

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

const int max_id_num = 20;

class resurs_stat {
  public:
    // Ukupni, tekuci i maksimalni broj entiteta u resursu
    int broj, tek_broj, max_broj;
    // Ukupno vreme za koje je resur bio zauzet
    double ukupno_vreme;
    // Vreme za koje se entitet zadrzava u resursu
    double povrsina_cekanja;
    // Poslednji trenutak promene stanja entiteta u resurs
    double vreme_promene;
    // Ukupno vreme simulacije
    double vreme_sim;
    // Polje rednih brojeva entiteta koji se nalaze u resursu
    int id[max_id_num];
    // Vremenski trenutak promene stanja resursa od strane entiteta
    double vreme[max_id_num];
  private:
    // Broj entiteta u polju id i polju vreme
    int broj_id;
  public:
    resurs_stat(): broj(0),tek_broj(0),max_broj(0),
                   ukupno_vreme(0),povrsina_cekanja(0),
                   vreme_promene(0), broj_id(0) {}
    // Dodajemo redni broj entiteta u polje id
    void dodaj_id(int novi_id,double novo_vreme) {
      if(broj_id<max_id_num) {
        id[broj_id] = novi_id;
        vreme[broj_id] = novo_vreme;
        broj_id++;
      }
      else {
        cerr << "Greska dodaj id" << endl;
```

```
        exit(1);
      }
    }
    // Nalazimo redni broj entiteta u polju id
    int nadji_id(int trazi_id) {
      int i;
      for(i=0; i<broj_id; i++)
        if(id[i]==trazi_id)
          break;
      return i;
    }
    // Uklanjamo redni broj i tekuce vreme entiteta iz polja id i vreme sa pozicije pos
    void ukloni_id(int pos) {
      for(int i=pos+1; i<broj_id; i++) {
        id[i-1] = id[i];
        vreme[i-1] = vreme[i];
      }
      broj_id--;
    }
    void vreme_simulacije(double vs) { vreme_sim = vs; }
    inline double srednje_vreme() {
      return ukupno_vreme/(broj-tek_broj);
    }
    inline double srednji_broj_entiteta() {
      return povrsina_cekanja/vreme_sim;
    }
    inline double iskoriscenost(int broj_kanala) {
      return srednji_broj_entiteta()/broj_kanala;
    }
};
// Klasa statistika reda cekanja
class fifo_red_stat: public resurs_stat {
  public:
    // Broj klijenata koji nisu cekali
    int broj_bezcekanja;
  public:
    fifo_red_stat():resurs_stat(),broj_bezcekanja(0) {}
    inline double procenat_entiteta_bez_zadrzavanja() {
      return (100.0*broj_bezcekanja)/broj;
    }
    inline double srednji_broj_entiteta_bez_zadrzavanja() {
      return ukupno_vreme/(broj-tek_broj-broj_bezcekanja);
```

```
    }
};
int main() {
  int pos,posn,idv;
  string str,tip,id,vreme;
  char line[1024];
  resurs_stat salt;
  fifo_red_stat red;
  double vremev, vreme_sim,
         vreme_u_redu, vreme_u_res;
  // Ucitavamo izvestaj
  ifstream in("simu2.txt",ifstream::in);
  // Ucitavamo podatke iz izvestaja sve dok ne dodjemo do kraja datoteke
  while(in.good()) {
    // Ucitavamo liniju iz izvestaja
    in.getline(line,1024);
    // Dodeljujemo liniju stringu
    str = line;
    // Izdvajamo tip obavestenja
    posn = str.find_first_of(',');
    tip = str.substr(0,posn);
    pos = posn;
    // Izdvajamo vrednost rednog broja entiteta
    posn = str.find_first_of(',',pos+1);
    id = str.substr(pos+1,posn-pos-1);
    pos = posn;
    // Izdvajamo vremenski trenutak
    vreme = str.substr(pos+1);
    // Prevodimo id i vremenski trenutak u broj
    idv = atoi(id.c_str());
    vremev = atof(vreme.c_str());
    // Proveravamo tip obavestenja
    if(!tip.compare("ured")) {
      // Broj entiteta koji su usli u red cekanja
      red.broj++;
      // Uvecavamo ukupno cekanje u redu
      red.povrsina_cekanja += red.tek_broj*(vremev-red.vreme_promene);
      // Pamtimo trenutak dolaska entiteta u red
      red.vreme_promene = vremev;
      //  Uvecavamo broj entiteta u redu
      red.tek_broj++;
      // Ukoliko je tekuci broj klijenata u redu  veci od maksimalnog
```

```
  // tekuci broj postaje maksimalni broj
  if(red.tek_broj>red.max_broj)
    red.max_broj = red.tek_broj;
  // Pamtimo redni broj i trenutak ulaska entiteta u redu
  red.dodaj_id(idv,vremev);
}
else if(!tip.compare("izred")) {
  // Nalazimo poziciju entiteta u redu
  int n = red.nadji_id(idv);
  // Izracunavamo vreme cekanja u redu
  vreme_u_redu = vremev-red.vreme[n];
  // Ukupno vreme koje su entiteti proveli u redu
  red.ukupno_vreme += vreme_u_redu;
  //
  red.povrsina_cekanja += red.tek_broj*(vremev-red.vreme_promene);
  // Ukoliko je vreme koje je entitet proveo u redu 0
  if(vreme_u_redu==0.0)
    // Uvecavamo broj entiteta koji nisu cekali u redu
    red.broj_bezcekanja++;
  // Uklanjamo redni broj entiteta sa pozicije n
  red.ukloni_id(n);
  //  Umanjujemo tekuci broj entiteta u redu
  red.tek_broj--;
}
else if(!tip.compare("usal")) {
  // Uvecamo broj entiteta koji su usli resurs
  salt.broj++;
  //
  salt.povrsina_cekanja += salt.tek_broj*(vremev-salt.vreme_promene);
  // Pamtimo trenutak poslednje promene broja zauzetih mesta
  salt.vreme_promene = vremev;
  //  Uvecavamo tekuci broj zauzetih mesta u salteru
  salt.tek_broj++;
  // Ukoliko je tekuci broj zauzetih mesta veci od maksimalnog
  // tekuci broj postaje maksimalni broj
  if(salt.tek_broj>salt.max_broj)
    salt.max_broj = salt.tek_broj;
  // Pamtimo redni broj entiteta u resursu i trenutak ulaska entiteta u resurs
  salt.dodaj_id(idv,vremev);
}
else if(!tip.compare("izsal")) {
  // Nalazimo poziciju entiteta u resursu
```

```
    int n = salt.nadji_id(idv);
    // Vreme koje entitet provodi u resursu
    vreme_u_res = vremev-salt.vreme[n];
    // Ukupno vreme koje su entiteti proveli u salteru
    salt.ukupno_vreme += vreme_u_res;
    //
    salt.povrsina_cekanja += salt.tek_broj*(vremev-salt.vreme_promene);
    // Pamtimo vreme ulaska entiteta u resurs
    salt.vreme_promene = vremev;
    // Uklanjamo redni broj entiteta sa pozicije n
    salt.ukloni_id(n);
    //  Umanjujemo tekuci broj entiteta u resursu
    salt.tek_broj--;
  }
  else if(!tip.compare("vsim")) {
    //  Pamtimo vreme simulacije
    vreme_sim = vremev;
    // Postavljamo vreme simulacije
    red.vreme_simulacije(vremev);
    salt.vreme_simulacije(vremev);
    //
    salt.povrsina_cekanja += salt.tek_broj*(vremev-salt.vreme_promene);
    red.povrsina_cekanja += red.tek_broj*(vremev-red.vreme_promene);
  }
}
// Zatvaramo datoteku
in.close();
// Podesavamo preciznost izlaznih rezulata
cout.precision(3);
cout.setf(ios_base::fixed);
// ISPISUJEMO STATISTIKE
// Ispisujemo vreme simulacije
cout << "Vreme simulacije: " << vreme_sim << endl;
cout << endl;
// Statistike reda cekanja
cout << "STATISTIKE REDA CEKANJA" << endl;
cout << "Broj entiteta koji je usao u red cekanja: " << red.broj << endl;
cout << "Preostali broj entiteta u redu cekanja: " << red.tek_broj << endl;
cout << "Maksimalni broj entiteta u redu: " << red.max_broj << endl;
cout << "Srednje vreme cekanja u redu: " << red.srednje_vreme() << endl;
cout << "Srednji broj entiteta u redu: " << red.srednji_broj_entiteta() << endl;
cout << "Broj entiteta koji je prosao kroz red bez zadrzavanja: " << red.broj_bezcekanja << endl;
```

```
  cout << "Procenat ulaza bez zadrzavanja: " << red.procenat_entiteta_bez_zadrzavanja() << endl;
  cout << "Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu zadrzali): "
       << red.srednji_broj_entiteta_bez_zadrzavanja() << endl;
  cout << endl;
  // Statistike resursa
  cout << "STATISTIKE RESURSA" << endl;
  cout << "Broj entiteta koji je usao u resurs: " << salt.broj << endl;
  cout << "Preostali broj entiteta u resursu: " << salt.tek_broj << endl;
  cout << "Maksimalni broj zauzetih kanala opsluge: " << salt.max_broj << endl;
  cout << "Srednje vreme opsluge: " << salt.srednje_vreme() << endl;
  cout << "Srednji broj zauzetih kanala: " << salt.srednji_broj_entiteta() << endl;
  cout << "Iskoriscenost: " << salt.iskoriscenost(4) << endl;
  return 0;
}
```