// Jun 2019

// 5. prihvatnih, ubacuje se, a uzima B/2

chan reply[N](int[]);

chan request(int, op, args);

process Buffer {

const int B = 20;

int buffer[B];

int head = 0, tail = 0, size = 0;

queue<int, int> waitingPut;

queue<int> waitingGet;

while (true) {

if (waitingGet.size()>0 && size>=B/2) {

clientId = waitingGet.pop();

operation = GET\_CONT;

} else if (waitingPut.size()>0 && size<B) {

clientId, arguments = waitingPut.pop();

operation = PUT\_CONT;

}

else

clientId, operation, argument = request.recevice();

switch (operation) {

case PUT:

if (size == B) {

waitingPut.push({clientId, argument});

break;

}

case PUT\_CONT:

buffer[head] = argument;

head = (head+1)%B;

reply[clientId].send(nullptr); // dummy

break;

case GET:

if (size < B/2) {

waitingGet.push(clientId);

break;

}

case GET\_CONT:

int ret[B/2];

for (int i=0;i<B/2;i++) {

ret[i] = buffer[tail];

tail = (tail+1)%N;

size--;

}

reply[cliendId].send(ret);

break;

}

}

}

// 6. CSP Game of Life

[(int i=0...N)(int j=0...N)C(i,j)::cell]

cell:: [

bool state = ..., other\_state;

int iteration = 0;

const int ITERATION\_NUM = 500;

int i1, i1;

int alive\_neighbours;

\*[

iteration < ITERATION\_NUM ->

[

alive\_neighbours = 0;

// primam od levog ako imam, saljem desnom ako ga imam

[j-1>=0, C(i,j-1)?message(other\_state) -> [other\_state -> alive\_neighbours++;] ]

[j+1<N -> C(i, j+1)!message(state)]

// primam od desnog ako imam, saljem levom ako ga imam

[j+1<N, C(i,j+1)?message(other\_state) -> [other\_state -> alive\_neighbours++;] ]

[j-1>=0 -> C(i, j-1)!message(state)]

// analogno u svim pravcima ...

// primam od gore levog ako ga imam, saljem dole desnom ako ga imam

[i-1>=0 && j-1>=0, C(i-1, j-1)?message(other\_state) -> [other\_state -> alive\_neighbours++;] ]

[j+1<N && i+1<N -> C(i+1, j+1)!message(state)]

[

state && (alive\_neighbours<2 || alive\_neighbours>3) -> state=false;

[] !state && alive\_neighbours==3 -> state = true;

]

iteration++;

]

]

]

// Jul 2019

// 3.

void tick() {

int time;

while (true) {

in("time", ?time);

out("time", time+1);

sleep(1min);

}

}

void control() {

while (true) {

//walk

string registration\_num = ...; // see a car

bool valid = true;

int time, time\_card, ticket;

in("time", ?time);

if (inp("parking\_card", ?ticket, registration\_num, ?time\_card)) {

if (time\_card + 60 > time)

valid = false;

} else

valid = false;

out("time");

if (!valid) {

// naplati kaznu

}

}

}

void cleaner() {

int ticketCheck = 0, time\_card, time;

string registration\_num;

while (true) {

rd("parking\_card", ticketCheck, ?registration\_num, ?time\_card);

rd("time", ?time);

if (time\_card + 60 > time) {

in("parking\_card", ticketCheck, registration\_num, time\_card);

ticketCheck++;

} else {

pause(1min);

}

}

}

void driver() {

string registration\_num = ...;

//...

int time, ticket;

in("time", ?time);

in("ticket", ticket);

out("ticket", ticket+1);

out("parking\_card", ticket, registration\_num, time); // platio parking

out("time", time);

//...

}

void init() {

out("time", 0);

eval(tick());

eval(control());

out("ticket", 0);

for (int i = 0; i < Nd; i++)

eval(driver(i));

}

// 4. Distibuted pairing CSP

// slicno kao Linda resenje (prvo cekati zahteve od nizih, a onda slati visima)

// 2019 Avgust

// 3.

while (true) {

int v1, v2;

v1 = in1.recevie();

v2 = in2.receive();

while (v1 != EOS && v2 != EOS) {

if (v1 > v2) {

out.send(v1);

v1 = in1.receive();

}

else if (v1 < v2) {

out.send(v2);

v2 = in2.receive();

} else if (v1 == v2) {

out.send(v1);

v1 = in1.receive();

v2 = in2.receive();

}

}

while (v1 != EOS) {

out.send(v1);

v1 = in1.receive();

}

while (v2 != EOS) {

out.send(v2);

v2 = in2.receive();

}

out.send(EOS);

}

// Povezati mrezu u vidu stabla, 18 procesa/cvorova je potrebno; niz ce biti sortiran na izlasku

// 4. Child care

void leave\_kid(int child\_num) {

int myTurn;

in("turn", ?myTurn);

out("turn", myTurn+1);

in("next", myTurn);

for (int i=0;i<child\_num;i++)

in("space");

for (int i=0;i<child\_num;i++)

out("kid");

out("next", myTurn+1);

}

void take\_kid(int child\_num) {

for (int i=0;i<child\_num;i++)

in("kid");

for (int i=0;i<child\_num;i++)

in("space");

}

void come\_to\_work() {

out("space");

out("space");

out("space");

}

void go\_home() {

int myTurn;

in("turn", ?myTurn);

out("turn", myTurn+1);

in("next", myTurn);

for (int i=0;i<3;i++)

in("space");

out("next", myTurn+1);

}

void init() {

out("turn", 0);

out("next", 0);

}

// 2019 Septembar

// 3.

void process() {

int myTurn;

in("turn", ?myTurn);

out("turn", myTurn+1);

in("next", myTurn);

//CRITICAL SECTION

out("next", myTurn+1);

}

void init() {

out("turn", 0);

out("next", 0);

for (int i=0;i<N;i++)

eval(process());

}

// 4.

messageBox bathroom;

messageBox clients[N];

process Bathroom {

Message m;

int cntm = 0, cntw = 0, cntk = 0, cntj = 0;

queue<Message> q;

Message waiting\_for\_kids = null;

while (true) {

if (waiting\_for\_kids != null && (cntk == 0 || cntm+cntw > 1)){

m = waiting\_for\_kids;

waiting\_for\_kids = null;

} else if (q.size() > 0 && (

(q.front().op == MAN\_IN\_GO && (cntw+cntj == 0)) ||

(q.front().op == WOMAN\_IN\_GO && (cntm+cntj == 0)) ||

(q.front().op == KID\_IN\_GO && (cntj == 0 && cntw + cntm > 0)) ||

(q.front().op == JANITOR\_IN\_GO && (cntm + cntw + cntk == 0))

)

m = q.pop();

else

m = bathroom.get();

switch (m.operation) {

case MAN\_IN:

if (q.size() > 0 || (cntw + cntj > 0)) {

m.op = MAN\_IN\_GO;

q.push\_back(m);

break;

}

case MAN\_IN\_GO:

cntm++;

clients[m.id].put(ackm);

break;

case MAN\_OUT:

if (cntk != 0 && cntm == 1) {

m.op = MAN\_OUT\_GO;

waiting\_for\_kids = m;

break;

}

case MAN\_OUT\_GO:

cntm--;

clients[m.id].put(ackm);

break;

case WOMAN\_IN:

// isto

break;

case WOMAN\_OUT:

// isto

break;

case KID\_IN:

if (q.size() > 0 || cntm + cntw == 0 || cntj > 0) {

m.op = KID\_IN\_GO;

q.push\_back(m);

break;

}

case KID\_IN\_GO:

cntk++;

clients[m.id].put(ackm);

break;

case KID\_OUT:

cntk--;

clients[m.id].put(ackm);

break;

case JANITOR\_IN:

if (q.size() > 0 || cntm + cntw + cntk + cntj > 0) {

m.op = JANITOR\_IN\_GO;

q.push\_back(m);

break;

}

case JANITOR\_IN\_GO:

cntj++;

clients[m.id].put(ackm);

break;

case JANITOR\_OUT:

cntj--;

clients[m.id].put(ackm);

break;

}

}

}

// Jan 2019

// 3.

/\*

Филтерски процеси имају један улаз и један излаз и раде следеће: примају позитивне вредности на улазу и прослеђују их на излаз ако су веће од запамћеног минимума процеса. Процеси имају само две локације, за сачувани минимум и за задњу примљену вредност. Када на улаз стигне EOS, избацују минималну вредност на излаз и затим EOS. Направите проточну обраду (pipeline) од n процеса који опадајуће сортирају: до n улазних позитивних вредности које се убацују на почетак проточне обраде, а завршавају се са EOS.

\*/

// Potrebno je povezati n-1 filterskih procesa redno i bice izvrseno sortiranje niza opadajuce na izlazu poslednjeg filterskog procesa. Povezuje se n-1 procesa kako bi se omogucilo sortiranje u najgorem slucaju (kada je niz obrnuto sortiran).

// 4.

/\*

[(int i=0...N)C(i)::car || (int i=0...M)V(i)::visitor || CO::coordinator]

car:: \*[

int visitor;

CO?get\_visitor();

CO!ok(visitor);

// ride

V(visitor)?free();

]

visitor:: [

//watch\_dinosaurs

int myCar;

CO!request\_car();

CO?ok(myCar);

//ride

C(myCar)!free();

]

coordinator:: [

int cars\_waiting[N]; int cnt\_cars=0;

int visitors\_waiting[M]; int cnt\_visitors=0;

\*[

(int i=0...N)C(i)?get\_visitor() -> [

\*[cnt\_visitors==0, (int j=0...M)C(j)?request\_car() -> visitors\_waiting[cnt\_visitors++] = j; ]

C(i)!ok(visitors\_waiting[cnt\_visitors-1]);

cnt\_visitors--;

]

[] (int i=0...M)V(i)?request\_car() -> [

\*[cnt\_cars==0, (int j=0...N)C(j)?get\_visitor() -> cars\_waiting[cnt\_cars++] = j; ]

C(i)!ok(cars\_waiting[cnt\_cars-1]);

cnt\_cars--;

]

]

]

\*/

// Feb 2019

// 3.

/\*

3. (20) И У систему се извршавају три процеса F, G и H, при чему сваки процес има локални низ целих бројева. Сваки од низова је уређен неопадајуће. Најмање један исти број се појављује у сваком од ова три низа. Написати код за сваки од ова три процеса у којем три процеса међусобно комуницирају док сваки од њих не одреди најмању заједничку вредност. Поруке које се размењују могу да садрже само по један елемент низа. Користити асинхрону размену порука. Сваки процес треба да има идентичан код.

\*/

chan c[3](int);

void process(int i) {

//init

int array[] = {...};

int N = ...;

int left = i;

int right = (i+1)%N;

int curr = 0;

int element;

int received2 = 0;

while (received2 < 2) {

c[right].send(array[curr]);

element = c[left].receive();

if (array[curr] == element) received2++;

else received2 = 0;

while (array[curr] < element) curr++;

}

}

// 4.

void disp() {

int turn = 0;

while (true) {

in("call", turn, ?location);

out("request");

sleep(n);

in("request");

int taxi, taxiloc;

int besttaxi=-1, bestloc = INF, taxicnt = 0;

while (inp("available", ?taxi, ?taxiloc)) {

if (taxiloc < bestloc) {

besttaxi = taxi;

bestloc = taxiloc;

}

taxicnt++;

}

put("reply", besttaxi);

if (taxicnt>0)

put("driving", besttaxi, taxicnt, 0);

else

in("nextcall");

turn++;

}

}

void user() {

int location = ...;

int ticket;

in("turn", ?ticket);

out("turn", ticket+1);

out("call", ticket, location);

int myTaxi;

in("reply", ?myTaxi);

if (myTaxi == -1) {

// NO FREE VEHICLES

} else {

in("startride", myTaxi);

out("startride\_ack", myTaxi);

// ride

in("endride", myTaxi);

out("endride\_ack", myTaxi);

}

}

void taxi(int i) {

int myLocation;

while (true) {

myLocation = ...;

bool riding = false;

while (!riding) {

in("request");

out("available", i, myLocation);

out("request");

int besttaxi, taxicnt, cnt;

in("driving", ?besttaxi, ?taxicnt, ?cnt);

cnt++;

if (cnt == taxicnt)

out("nextcall");

else

out("driving", besttaxi, taxicnt, cnt+1);

driving = (besttaxi==i);

}

out("startride", i);

in("startride\_ack", i);

// ride

out("endride", i);

in("endride\_ack", i);

}

}

void init() {

out("turn", 0);

}

// 2020 Jan

// 3.

enum op {WAIT, SIGNAL, WAIT\_N, SIGNAL\_N};

chan request(int, op, arg);

chan reply[N](dummy);

void Semaphore() {

int clientId;

operation op;

int arg;

int s = 0;

queue<int, int> waiting;

while (true) {

if (waiting.size()>0 && s >= arg) {

clientId, arg = waiting.pop();

op = WAIT\_CONT;

} else {

clientId, op, arg = request.receive();

if (op == WAIT) { op = WAIT\_N; arg = 1; }

if (op == SIGNAL) { op = SIGNAL\_N; arg = 1; }

}

switch (operation) {

case WAIT\_N:

if (waiting.size()>0 || s < arg) {

waiting.push\_back({clientId, arg});

break;

}

case WAIT\_CONT:

s -= arg;

reply[clientId].send(dummy);

break;

case SIGNAL\_N:

s += arg;

reply[clientId].send(dummy);

break;

}

}

}

// 4. Gravitaciono telo CSP

// 2020 Feb

// 3.

// n-1 procesa vezanih redno koji vrse sortiranje niza tako sto pamte najmanji element pronadjen do sada, a izbacuju veci element; ako dodje EOS, izbacuje se min element i EOS

// poslednji proces izvaja element na poziciji n/2. koji je medijana

chan in(int);

chan out(int);

void sort\_process() {

int min = None;

while (true) {

int num = in.receive();

if (num == EOS) {

if (min != None)

out.send(min);

out.send(EOS);

} else {

if (min != None) {

if (num < min) {

out.send(min);

min = num;

}

else

out.send(num);

} else

min = num;

}

}

}

void median() {

int current = 0;

while (true) {

int num = in.receive();

if (num == EOS) { current = 0; continue; }

current++;

if (current == (n-1)/2) out.send(num);

}

}

// 4.

void barber() {

int sleeping\_turn;

while (true) {

in("sleeping\_turn", ?sleeping\_turn);

out("sleeping\_turn", sleeping\_turn+1);

in("barber", sleeping\_turn);

out("sit", sleeping\_turn);

// waiting customer to sit

in("sit\_ack", sleeping\_turn);

// haircutting

out("pay", sleeping\_turn);

// waiting for money

in("paid", sleeping\_turn);

}

}

void customer() {

int myTicket;

in("ticket", ?myTicket);

in("waiting", ?waiting);

if (waiting==20) {

out("waiting", waiting);

out("ticket", myTicket);

// go home

}

out("waiting", waiting+1);

out("ticket", myTicket+1);

// stojim

in("next\_haircutting", myTicket);

out("next\_haircutting", myTicket+1);

in("waiting", ?waiting);

out("waiting", waiting-1);

int barber\_next;

in("barber\_next", ?barber\_next);

out("barber\_next", barber\_next+1);

out("gethaircut", barber\_next);

in("sit", barber\_next);

out("sit\_ack", barber\_next);

// getting haircut

in("pay", barber\_next);

out("paid", barber\_next);

// go home

}

void init() {

out("ticket", 0);

out("waiting", 0);

out("barber\_turn", 0);

out("barber\_next", 0);

out("next\_standing", 0);

out("next\_sitting", 0);

out("next\_haircutting", 0);

}

// Sep 2018

// 3.

process P() {

int v1, v2;

v1 = in1.receive();

v2 = in2.receive();

if (v1<v2) out.send(v1);

else out.send(v2);

in1.receive();

in2.receive();

out.send(EOS);

}

// Povezati u vidu stabla, potrebno je 10 cvorova/procesa

//4 nema sanse, previse podslucajeva

/\*

bathroom:: [

int cntm = 0, cntw = 0, cntk = 0, cntj = 0;

\*[

(int i=0...Nw)W(i)?goIn() -> [

janitor\_out

man\_out

cntw++;

W(i)!ok();

]

[] (int i=0...Nm)M(i)?goIn() -> [

janitor\_out

woman\_out

cntm++;

M(i)!ok();

]

[] (int i=0...Nk)K(i)?goIn() -> [

janitor\_out

\*[

cntm==0 && cntw==0, M(i)? (int i=0...Nm)M(i)?goIn() -> [ cntm++; M(i)!ok; ]

cntm==0 && cntw==0, W(i)? (int i=0...Nw)W(i)?goIn() -> [ cntm++; W(i)!ok; ]

]

cntk++;

K(i)!ok();

]

[] (int i=0...Nk)K(i)?goOut() -> cntk--;

[] (int i=0...Nj)J(i)?goIn() -> [

cntj++;

J(i)!ok();

]

[] (int i=0...Nj)J(i)?goOut() -> cntj--;

]

]

janitor\_out::[cntj>0, (int i=0...Nj)J(i)?goOut() -> cntj--; ]

manempty::

\*[

cntm>0 && cntk>0, (int i=0...Nk)K(i)?goOut() -> cntk--;

(int i=0...Nw)W(i)?goOut() -> [

[cntk>0 && cntw==1, (int i=0...Nk)K(i)?goOut() -> cntk--;]

]

]

womanempty:: ...

drugo resenje, nije dobro (izgladnjivanje/nije fifo koliko je moguce)

[

int cntm = 0, cntw = 0, cntk = 0, cntj = 0;

\*[

cntw + cntj == 0, (int i=0...Nm)M(i)?goIn() -> [ cntm++; ]

[] cntm + cntj == 0, (int i=0...Nw)W(i)?goIn() -> [ cntw++; ]

[] cntm + cntw + cntk + cntj == 0, (int i=0...Nj)J(i)?goIn() -> [ cntj++; ]

[] cntm + cntw > 0, (int i=0...Nk)K(i)?goIn() -> [ cntk++; ]

[] (int i=0...Nk)K(i)?goOut() -> [ cntk--; ]

[] (int i=0...Nj)J(i)?goOut() -> [ cntj--; ]

[] cntk == 0 || cntm > 1, (int i=0...Nm)M(i)?goOut() -> [ cntm--; ]

[] cntk == 0 || cntw > 1, (int i=0...Nw)W(i)?goOut() -> [ cntw--; ]

]

]

\*/

// Jul 2018

// 4.

/\*

childcare:: [

int deca = 0, vaspitacice = 0;

int k, k1;

\*[

(int i=0...Nr)R(i)ostaviDecu(k)?->[

\*[

deca+k > 3\*vaspitacice, (int i=0...Nv)V(i)dodji()?-> vaspitacice++;

[] deca+k > 3\*vaspitacice, (int i=0...Nr)R(i)odvediDecu(k1)?-> deca -= k1;

]

deca += k;

R(i)!ok();

]

[] (int i=0...Nr)R(i)odvediDecu(k)?-> deca -= k;

[] (int i=0...Nv)V(i)dodji()?-> vaspitacice++;

[] (int i=0...Nv)V(i)idi()?->[

\*[

deca > 3\*(vaspitacice-1), (int i=0...Nv)V(i)dodji()?-> vaspitacice++;

[] deca > 3\*(vaspitacice-1), (int i=0...Nr)R(i)odvediDecu(k1)?-> deca -= k1;

]

vaspitacice--;

V(i)!ok();

]

]

]

\*/

// Filterski procesi, izbacivanje duplikata (i sortiranje)

while (true) {

int v1, v2;

v1 = in1.recevie();

v2 = in2.receive();

while (v1 != EOS && v2 != EOS) {

if (v1 > v2) {

out.send(v1);

v1 = in1.receive();

}

else if (v1 < v2) {

out.send(v2);

v2 = in2.receive();

} else if (v1 == v2) {

out.send(v1);

v1 = in1.receive();

v2 = in2.receive();

}

}

while (v1 != EOS) {

out.send(v1);

v1 = in1.receive();

}

while (v2 != EOS) {

out.send(v2);

v2 = in2.receive();

}

out.send(EOS);

}

// CSP Readers writers

/\*

[(int i=1..N)R::reader || (int i=1..M)W::writer || RW::rw]

rw:: [

int nr = 0;

\*[

(int i=1..N)R(i)?startRead() -> [ nr++; ]

[] (int i=1..N)R(i)?endRead() -> [ nr--; ]

[](int i=1..M)W(i)?startWrite() -> [

\*[ nr>0, (int i=1..N)R(i)?endRead() -> [ nr--; ]]

W(i)!okToWrite();

W(i)?endWrite();

]

]

]

R:: \*[

RW!startRead();

RW?ok();

READING;

RR!endRead();

]

W:: [

RW!startWrite();

RW?ok();

READING;

RR!endWrite();

] \*/

// Jun 2020

// 3.

chan req(int, op);

chan reply[N](dummy);

process Server {

bool forks[N] = {true, ..., true};

queue<int, int> waiting;

int id;

op operation;

while (true) {

if (waiting.size()>0 && canEat(waiting.front().id))

id, operation = waiting.pop\_front();

else

receive req(id, operation);

switch(operation) {

case GET\_FORKS:

if (waiting.size()>0 || !canEat(id)) {

waiting.push\_back({id, GET\_CONT});

break;

}

// nema break jer ako ne udje u if propada se u sledeci case da uzme viljuske

case GET\_CONT:

forks[id] = false;

forks[(id+1)%N] = false;

send reply[id](0);

break;

case RELEASE\_FORKS:

forks[id] = true;

forks[(id+1)%N] = true;

send reply[id](0); // nije neophodno

break;

}

}

}

bool canEat(int i) {

return forks[i] && forks[(i+1)%N];

}

process Philosopher(int i) {

dummy d;

while (true)

{

think();

send req(i, GET\_FORKS);

receive reply[i](d);

eat();

send req(i, RELEASE\_FORKS);

receive reply[i](d); // nije neophodno

}

}

// 4.

/\*

[

int cnt\_a = 0, cnt\_p = 0;

int sem = AUTO;

int proslo = 0;

int w\_auto[N], w\_pesak[T];

int ha=0, ta=0, sa=0, hp=0, tp=0, sp=0;

int promena = system\_current\_time;

int prvi\_pesak;

\*[

// ukoliko dodje pesak i propustaju se pesaci, propusti ga

// ako se ne propustaju, cekaj u redu, i zabelezi vreme dolaska prvog pesaka koji ceka

(int i=0..T)P(i)?prodji() -> [

[

sem == PESAK -> [ cnt\_p++; P(i)!ok(); ]

sem != PESAK -> [

w\_pesak[hp] = i;

hp = (hp+1)%T;

sp++;

[sp == 1 -> prvi\_pesak = system\_current\_time; ]

]

]

]

// ukoliko je semafor otvoren za pesake, propustaj one pesake koji cekaju

[] sem == PESAK && sp>0 -> [

P(w\_pesak[tp])!ok();

tp = (tp+1)%T;

sp--;

]

// ukoliko dodje automobil i propustaju se automobili, propusti ga

// ako se ne propustaju, ceka u redu

[] (int i=0..N)A(i)?prodji() -> [

[

sem == AUTO -> [ cnt\_a++; A(i)!ok(); proslo++; ]

sem != AUTO -> [

w\_auto[ha] = i;

ha = (ha+1)%N;

sa++;

]

]

]

// ukoliko je semafor otvoren za automobile, propustaj one automobile koji cekaju

[] sem == AUTO && sa>0 -> [

P(w\_auto[ta])!ok();

ta = (ta+1)%N;

sa--;

proslo++;

]

// ako je za pesake i isteklo je G, kreni da menjas

// sacekaj da odu svi pesaci pa onda propusti automobile

[] sem == PESAK && system\_current\_time - promena >= G -> [ sem = PROMENA\_PESAK; ]

[] sem == PROMENA\_PESAK && cnt\_p==0 -> [ sem = AUTO; promena = system\_current\_time; ]

// ako je za automobile, neko od pesaka ceka i isteklo je K, kreni da menjas

// ako je za automobile, neko ceka i proslo je vise od C u suprotnom smeru od promene svetla, kreni da menjas

// sacekaj da odu svi automobili pa kreni da propustas pesake

[] sem == AUTO && sp>0 && system\_current\_time-prvi\_pesak >= K [ sem = PROMENA\_AUTO; ]

[] sem == AUTO && sp>0 && proslo >= C -> [ sem = PROMENA\_AUTO]

[] sem == PROMENA\_AUTO && cnt\_a == 0 -> [ sem = PESAK; promena = system\_current\_time; proslo = 0; ]

// kada izadju iz raskrsnice umanje

[] (int i=0..N)A(i)?prosao() -> cnt\_a--;

[] (int i=0..T)P(i)?prosao() -> cnt\_p--;

]

]

\*/

// Game of life (razmena poruka)

messageBox cell[N][N];

proccess Node(int i, int j) {

bool state = true;

int alive[2] = {0,0}, received[2] = {0,0};

int neighbours;

while (numGeneration < K) {

m.i = i;

m.j = j;

m.state = state;

m.iteration = numGeneration;

neighbours = 0;

for (int di=-1; di<=1; di++) {

for (int dj=-1; dj <= 1; dj++) {

if (di == 0 && dj == 0) continue;

if (i+di >= 0 && i+di < N && j+dj >= 0 && j+dj < N)

cell[i+di][j+dj].put(m);

neighbours++;

}

}

while (received[numGeneration%2] < neighbours) {

m = cell[i][j].get(INF, status);

alive[m.iteration%2] += m.state;

received[m.iteration%2]++;

}

if (state && (alive[m.iteration%2] < 2 || alive[m.iteartion%2] > 3))

state = false;

else if (!state && alive[m.iteration%2] == 2 || alive[m.iteration%2] == 3)

state = true;

alive[m.iteration%2] = 0;

received[m.iteration%2] = 0;

numGeneration++;

}

}

// Game of Life (Linda)

inline bool valid(int i, int j, int di, int dj) {

return i+di >= 0 && i+di < N && j+dj >= 0 && j+dj < N;

}

int num\_neighbours(int i, int j) {

int k = 0;

for (int di=-1;di<=1;di++) {

for (int dj=-1;dj<=1;dj++) {

if (di == 0 && dj == 0) continue;

k += valid(i, j, di, dj);

}

}

return k;

}

void gof(int i, int j) {

bool alive = (bool)(rand()%2);

int iteration = 0;

int neighbours = num\_neighbours(i, j);

while (iteration < NUM\_IT) {

for (int di=-1;di<=1;di++) {

for (int dj=-1;dj<=1;dj++) {

if (di == 0 && dj == 0) continue;

if (valid(i, j, di, dj))

out(i+di, j+dj, iteration, state); // javim svim komsijama

}

}

int neighbours\_alive = 0;

for (int i=0;i<neighbours;i++) {

bool state\_ij;

in(i, j, iteration, ?state\_ij);

neighbours\_alive += state\_ij;

}

if (alive && !(neighbours\_alive == 2 || neighbours\_alive == 3)) alive = false;

else if (!alive && neighbours == 3) alive = true;

iteration++;

}

}

void init() {

for (int i=0;i<N;i++)

for (int j=0;j<N;j++)

eval(cell(i, j));

}

// Distributed pairing (Linda)

// cvorovi su od 0 do N

void node(i) {

bool adj[N];

bool canPair;

int myPair = -1;

// cekam od nizih zahtev, uparujem se sa prvim, svima odgovaram prihvatam li

for (int j=0;j<i;j++)

if (adj[j]) {

in("req", i, adj[j], ?canPair);

if (canPair) {

if (myPair==-1) myPair = adj[j];

out("ack", adj[j], i, myPair == adj[j]);

}

}

// svima visima od sebe saljem zahtev za uparivanje/neuparivanje, i ako nisam

// uparen, uparujem se kad neko prihvati

for (int j=i+1;j<N;j++)

if (adj[j]) {

out("req", adj[j], i, myPair == -1);

if(myPair==-1) {

in("ack", i, adj[j], ?canPair);

if (canPair) myPair = adj[j];

}

}

}