%Generation\_kQueens\_Composition

%{

Программа служит для формирования композиции из k ферзей, которые

распределены на шахматной доске размера n x n. Под композицией мы понимаем случайное распредление *k* ферзей на произвольной шахматной доске с размером *n x n*, таким образом, что выполняются три условия задачи: *в каждой строке, в каждом столбце, а также на левой и правой диагоналях, проходящих черз позицию, где расположен ферзь, не располагается более одного ферзя.*

%}

%{

License: Attribution-NonCommercial-ShareAlike  
CC BY-NC-SA – “This license lets others remix, adapt, and build upon your work non-commercially, as long as they credit you and license their new creations under the identical terms”.

%}

%{

Автор проекта и разработчик – Григорян Эрос (EricGrig), 2020

Буду рад, если какие-либо участки кода, или вся программа в целом, будут использоваться для научных целей, или для образования. При этом, буду благодарен, если сочтете возможным сослаться на мою публикацию. Это элемент культуры и знак взаимного уважения.

Для использования в коммерческих целях любого участка кода программы, или всей программы в целом, необходимо письменное согласие автора.

%}

% Для работы программы необходимо указать размер стороны шахматной доски (n)

n=1000;

tStr = sprintf(' The size of a chessboard = %d',n);

disp(tStr);

nx=n-1; % nx – максимальный размер композиции

% n2 - размер контрольных массивов

n2=2\*n;

%{

nFix - Фиксированное значение размера матрицы решения. Если n<nFix то

выполнение решения передается в блок-3, минуя блок-1 и блок-2

%}

nFix=17;

%{

bound\_1\_Ar, bound\_2\_Ar - массивы значений для eventBound1 и eventBound2

при n <30. Эти значения определены на основе вычислительных экспериментов

%}

bound\_1\_Ar =[2 2 2 3 3 3 3 3 4 4 4 4 5 5 5 6 6 7];

bound\_2\_Ar =[4 4 5 5 5 6 6 7 7 8 8 9 10 10 11 11 12 13];

%{

Вычисление значений eventBound1 и eventBound2 на основе уравнений

регрессии. Эти результаты получены на основе вычислительных экспериментов

%}

if n<nFix

if n<9

eventBound1=1;

eventBound2=2;

elseif n<12

eventBound1=2;

eventBound2=3;

elseif n<14

eventBound1=3;

eventBound2=4;

else

eventBound1=3;

eventBound2=6;

end

sameRepeatBound=10;

else

sameRepeatBound=5;

if n<30

nInd=n-12;

eventBound1=bound\_1\_Ar(nInd);

eventBound2=bound\_2\_Ar(nInd);

else

u=log10(n);

w=u\*u;

if n<100

b1=293.898676\*w\*u-1495.491673\*w+2578.130423\*u-1470.692935;

b2=91.458481\*w\*u-474.647556\*w+849.173904\*u-497.393064;

elseif n<30000

b1=12.749568\*w\*u -46.535838\*w + 120.011829\*u -89.600272;

b2=9.717958\*w\*u -46.144187\*w + 101.296409\*u -50.669273;

else

b1=-0.886344\*w\*u+56.136743\*w-146.486415\*u+227.967782;

b2=14.959815\*w\*u-253.661725\*w+1584.711376\*u-3060.691342;

end

eventBound1=n-round(b1);

eventBound2=n-round(b2);

end

end

%{

Переменные simBound1, simBound2 и simBound4 определяют

максимальное число повторных вычислений в пределах блоков 1,2 и 4

%}

simBound1=3;

simBound2=5;

simBound4=10;

%{

totSimBound - граничное значение для учета общего количества всех

повторных вычислений

%}

totSimBound=1000;

% Определим случайный размер композиции nComp=(1, ... ,n-1)

nComp=randi(nx);

%{

Цепочка решений будет продолжаться до тех пор, пока не будут установлены

ферзи в nComp строках

%}

%Обнулим рабочий массив Q(1:n)

Q=zeros(1,n,'uint32');

% Для работы обнулим рабочие массивы

% A(1:n) - для контроля индексов свободных строк

A=zeros(1,n,'uint8');

% B(1:n) - для контроля индексов свободных столбцов

B=zeros(1,n,'uint8');

% C(1:n2), D(1:n2) - для контроля занятости диагональных проекций

C=zeros(1,n2,'uint8');

D=zeros(1,n2,'uint8');

% totPos - счетчик суммарного числа всех повторных вычислений

totPos=0;

%{

Обнулим счетчики числа повторений simCount для каждого события

simCount1, simCount2, simCount2, simCount4 - счетчики числа повторений

в пределах блоков 1,2 и 4

totSimCount - счетчик суммарного числа всех повторений

%}

simCount1=0;

simCount2=0;

simCount3=0;

simCount4=0;

totSimCount=0;

% eventInd=1 -- в программе генерации расчеты начинаются с 1-го блока

eventInd=1;

% swiInd (switch Index) - "переключатель" для выхода из цикла

processInd =1;

%{

Все события разворачиваются внутри цикла while swiInd==1, пока не будет

получено решение.

%}

tic

while processInd ==1

% Переменная eventInd служит в качестве переключателя между 4-мя событиями

switch eventInd

case 1

% Алгоритм rand\_set & rand\_set. Комплектация композиции до размера simBound1

simCount1=simCount1+1;

if nComp<eventBound1

xEvent=nComp;

else

xEvent=eventBound1;

end

while totPos < xEvent

xInd=find(A==0);

nRow=length(xInd);

aInd=uint32(randperm(nRow));

yInd=find(B==0);

bInd=uint32(randperm(nRow));

for k=1:nRow

i1=aInd(k);

i=xInd(i1);

j1=bInd(k);

j=yInd(j1);

r=n+j-i;

t=j+i;

if C(r)==0 && D(t)==0

C(r)=1;

D(t)=1;

Q(i)=j;

A(i)=1;

B(j)=1;

totPos=totPos+1;

end

if totPos==xEvent

break

end

end

end

if nComp<= eventBound1

processInd =0;

else

% Найдем исходные индексы оставшихся свободных строк в матрице решения.

A=find(A==0);

nFreeRow=length(A);

% Найдем исходные индексы оставшихся свободных столбцов в матрице решения.

B=find(B==0);

%{

Создадим массив L(1:nFreeRow,1:nFreeRow) и заполним все ячейки единицей.

Далее, если ячейка L(p,q) окажется свободной, то в эту ячейку вместо

единицы запишем ноль.

%}

L=ones(nFreeRow,nFreeRow,'uint8');

%{

Создадим рабочие массивы rAr и tAr для сохранения индексов соответствия

контрольным массивам

%}

rAr=zeros(nFreeRow,nFreeRow,'uint32');

tAr=zeros(nFreeRow,nFreeRow,'uint32');

%{

На основе информации об оставшихся свободных строках и свободных

столбцах, запишем нуль в соответствующие свободные ячейки массива L

Формируем массивы учета rAr, tAr

%}

for p=1:nFreeRow

i=A(p);

for q=1:nFreeRow

j=B(q);

r=n+j-i;

t=j+i;

if C(r)==0 && D(t)==0

L(p,q)=0;

rAr(p,q)=r;

tAr(p,q)=t;

end

end

end

%{

Создадим резервные копии всех основных массивов. Они нам будут нужны,

если возникнет необходимость вернуться к началу события 2 для повторных

расчетов

%}

Ay=A;

By=B;

Cy=C;

Dy=D;

Qy=Q;

Ly=L;

rAr\_y=rAr;

tAr\_y=tAr;

yPos=totPos;

eventInd=2;

end

case 2

% В данном блоке отбор производится на основе алгоритма rand & rand

simCount2=simCount2+1;

%{

Далее, продолжим формирование ветви поиска решения на основе данных,

собранных в массиве L

%}

if nComp<eventBound2

xEvent=nComp;

else

xEvent=eventBound2;

end

while totPos < xEvent

% Определим количество свободных строк в массиве на основе массива A

freeRowInd=find(A>0);

freeRow=length(freeRowInd);

% Выберем случайную строку из списка свободных строк

selectRowInd=randi(freeRow);

iInd=freeRowInd(selectRowInd);

% Сформируем список индексов свободных позиций в строке i массива L

freePosAr=find(L(iInd,:)==0);

freePos=length(freePosAr);

if freePos>0

%{

Если в выбранной строке имеются свободные позиции, то продолжаем решение

Здесь, позицию ферзя в строке выбираем случайным образом

%}

selectPosInd=randi(freePos);

jInd=freePosAr(selectPosInd);

j=B(jInd);

% Сохраним j-индекс позиции ферзя в массиве решений

i=A(iInd);

Q(i)=j;

% Инкрементируем счетчик учета количества позиций, занятых ферзем

totPos=totPos+1;

% Запишем 0 в ячейку iInd массива A,чтобы зафиксировать, что строка i занята

A(iInd)=0;

% Запишем 0 в ячейку jInd массива B,чтобы зафиксировать, что строка i занята

B(jInd)=0;

%{

Изменим соответствующие ячейки запретных массивов C и D, используя

реальные значения индексов (i,j) (для исходного "большого" массива)

%}

rx=n+j-i;

tx=j+i;

C(rx)=1;

D(tx)=1;

%{

Изменим соответствующие ячейки массива L, используя эквивалентные индексы,

сохраненные в массивах rAr и tAr

%}

rxInd=find(rAr==rx);

L(rxInd)=1;

txInd=find(tAr==tx);

L(txInd)=1;

L(freeRowInd,jInd)=1;

else % if freePos>0

%{

Если в рассматриваемой строке нет свободных позиций,то мы достигли тупика,

поэтому должны закрыть данную ветвь поиска и вернуться назад, в начало

цикла while totPos < simBound2 и повторить формирование новой ветви поиска

Но перед этим, мы должны восстановить все необходимые массивы на основе

резервных копий. Именно с этими массивами мы входили в событие 2

%}

if simCount2 < simBound2

A=Ay;

B=By;

C=Cy;

D=Dy;

Q=Qy;

L=Ly;

rAr=rAr\_y;

tAr=tAr\_y;

totPos=yPos;

eventInd=2;

else

% Обнулим рабочие массивы и передадим управление событию 1

A=zeros(1,n,'uint8');

B=zeros(1,n,'uint8');

C=zeros(1,n2,'uint8');

D=zeros(1,n2,'uint8');

Q=zeros(1,n,'uint32');

totPos=0;

simCount2=0;

eventInd=1;

break

end

end % if freePos>0

end % while totPos < simBound2

if nComp<= eventBound2

processInd=0;

else

if totPos>=xEvent

eventInd=3;

end

end

%{

Мы завершили вторую часть формирования ветви поиска и достигли уровня,

когда в матрице решения правильно распределены simBound2 ферзей. Перейдем

к третьему этапу

%}

case 3

simCount3=simCount3+1;

%{

Далее исключим из расмотрения занятые строки и занятые столбцы. Сформируем

овую компактную матрицу L как пересечение числа оставшихся строк и числа

оставшихся столбцов. Для этого, найдем индексы оставшихся свободных строк,

согласно массива учета занятых строк A

%}

T=find(A>0);

A=A(T);

nRow=length(T);

% Аналогично определим массив индексов свободных столбцов.

T=find(B>0);

B=B(T);

%{

дадим массив L(1:m,1:m) и заполним все ячейки единицей. Далее, если

ячейка L(p,q) окажется свободной, то в эту ячейку вместо единицы запишем

ноль.

%}

L=ones(nRow,nRow,'uint32');

%{

Создадим рабочие массивы для сохранения индексов соответствия контрольным

массивам

%}

rAr=zeros(nRow,nRow,'uint32');

tAr=zeros(nRow,nRow,'uint32');

% Создадим массивы для учета накопительного списка ограничений

Cs=zeros(1,n2,'uint32');

Ds=zeros(1,n2,'uint32');

Bs=zeros(1,n,'uint32');

%{

На основе информации об оставшихся свободных строках и свободных

столбцах, запишем zero в соответствующие свободные ячейки массива L

Формируем массивы Cs, Ds, Bs, а также массивы учета rAr, tAr

Для всех m строк и, соответственно, для оставшихся свободных

позиций в этих строках сформируем накопительный список ограничений для

левой Cs и правой Ds диагональных проекций, а также для проекций столбца

%}

for p=1:nRow

i=A(p);

for q=1:nRow

j=B(q);

r=n+j-i;

t=j+i;

if C(r)==0 && D(t)==0

L(p,q)=0;

rAr(p,q)=r;

tAr(p,q)=t;

Cs(r)=Cs(r)+1;

Ds(t)=Ds(t)+1;

Bs(j)=Bs(j)+1;

end

end

end

% Вычислим сумму элементов каждой строки массива L

rowSum=sum(L==0,2);

%{

Сортируем значения суммы в порядке возрастания числа свободных позиций

в каждой строке

%}

[sumSort,rowRangInd]=sort(rowSum);

%{

Здесь, в массиве rowRangInd последовательно сохранены индексы строк с

возрастающим числом свободных позиций в строке. Если окажется, что

что во всех оставшихся строках, "собранных" в массив L есть свободные

позиции, то массив rowRangInd будет использоваться далее, в блоке 4

%}

if sumSort(1)>0

%{

Здесь sumSort(1) -минимальное число свободных позиций в списке всех строк

массива L(m,m). Если минимальное число свободных позиций > 0, то мы

продолжаем построение ветви поиска, т.к. до данного шага построенная

ветвь оставалась перспективной

Создадим контрольный массив учета E размера nRow x nRow, в каждой ячейке

которого сохраним совокупное значение накопительных массивов ограничений

%}

E=zeros(nRow,nRow,'uint32');

%{

Вычислим и сохраним в E совокупное значение накопительных массивов

ограничений

%}

for p=1:nRow

for q=1:nRow

r=rAr(p,q); %Индекс r для массива Cs

t=tAr(p,q); %Индекс t для массива Ds

j=B(q); %Индекс j для массива Bs

if r>0 && t>0

E(p,q)=Cs(r)+Ds(t)+Bs(j);

end

end

end

%{

Далее, вместо массивов Cs,Ds,Bs мы будем пользоваться массивом E

При рассмотрении больших значений n следует удалить эти массивы, чтобы освободить память

%}

%{

Прежде чем перейти к следующему событию, сохраним для повторного

использования копии этих массивов.

%}

Az=A;

Bz=B;

Qz=Q;

Lz=L;

Ez=E;

zPos=totPos;

% Далее, перейдем к событию 4

eventInd=4;

else % if sumSort(1)>0

%{

Если окажется, что среди оставшихся строк есть строка, в которой

отсутствуют свободные позиции, то мы восстанавливаем исходные значения

массивов и передаем управление в событие 2

%}

A=Ay;

B=By;

C=Cy;

D=Dy;

Q=Qy;

L=Ly;

rAr=rAr\_y;

tAr=tAr\_y;

totPos=yPos;

eventInd=2;

end %if sumSort(1)>0

case 4

simCount4=0;

for iRow=1:nRow

selectRowInd=rowRangInd(iRow);

%{

Определим соответствующее(исходное) значение номера строки в массиве L,

используя данные базового уровня

%}

initRowInd=A(selectRowInd);

%Определим число свободных позиций в выбранной строке

T=L(selectRowInd,:);

baseFreePosInd=find(T==0);

baseFreePos=length(baseFreePosInd);

%{

Далее, здесь, на базовом уровне, в пределах рассматриваемой строки

с текущим, минимальным значением числа свободных позиций в строке

будем последовательно, в цикле,рассмотривать каждую свободную позицию

%}

for jCol=1:baseFreePos

% Присвоим i реальный номер выбранной строки на базовом уровне

i=initRowInd;

jPos=baseFreePosInd(jCol);

% Присвоим j значение выбранной свободной позиции на базовом уровне

j=B(jPos);

% Присвоим minRowInd индекс строки в массиве L(1:nRow,1:nRow) с минимальным

% числом свободных позиций в строке

minRowInd=selectRowInd;

%Начало основной части алгоритма в событии 4.

sSame=0;

while totPos < nComp

%Для первого шага в этом цикле значения i,j мы определили выше

%Сохраним j-индекс позиции ферзя в массиве решений

Q(i)=j;

% Инкрементируем счетчик учета количества позиций, занятых ферзем

totPos=totPos+1;

% Проверим, если сформировано полное решение, то завершаем расчеты

if totPos==nComp

totSimCount=totSimCount+1;

processInd =0;

break

end

%{

Мы завершили очередной цикл определения индексов для расположения ферзя

и расположили ферзя в ячейку (i,j) матрицы решения

После этого, мы должны изменить соответствующие ячейки во всех контрольных

массивах, учитывая индексы (minRowInd,colInd) массива L

%}

A(minRowInd)=0;

B(jPos)=0;

%{

Изменим соответствующие ячейки массива L, используя эквивалентные индексы,

сохраненные в массивах rAr и tAr

%}

rx=n+j-i;

tx=j+i;

rxInd=find(rAr==rx);

L(rxInd)=1;

txInd=find(tAr==tx);

L(txInd)=1;

%{

Декрементируем значение накопительного контрольного массива, т.е.

уменьшим "эффект влияния" свободных позиций в отобранной строке, после

того как там расположили ферзя

%}

E(rxInd)=E(rxInd)-1;

E(txInd)=E(txInd)-1;

% Запишем 1 во все активные ячейки столбца colInd

% Активные ячейки задаются массивом A1

A1=find(A>0);

L(A1,jPos)=1;

rowSum=sum(L(A1,:)==0,2);

% Определим индекс строки minRowInd с минимальным числом свободных позиций

[freePosAr,rowIndAr]=sort(rowSum);

if freePosAr(1)>0

%{

Если две строки имеют одинаковое минимальное число свободных позиций, то

случайно выбираем индекс одной из этих строк

%}

if numel(freePosAr)==1||freePosAr(1)<freePosAr(2)

randPos=1;

else

randPos=randi(2);

end

minRow=rowIndAr(randPos);

minRowInd=A1(minRow);

i=A(minRowInd);

% Определим число свободных позиций в этой строке

rowFreePosAr=find(L(minRowInd,:)==0);

nfreePos=length(rowFreePosAr);

%{

Выберем среди этих позиций ту, которая закрывает минимальное число

свободных позиций в оставшихся строках. Для этого воспользуемся

массивом E(m,m)

%}

if nfreePos==1

jPos= rowFreePosAr(1);

else

T=E(minRowInd, rowFreePosAr);

[tSort,tInd]=sort(T);

if tSort(1)<tSort(2)

jPos= rowFreePosAr(tInd(1));

else

jInd=randi(2);

jPos= rowFreePosAr(tInd(jInd));

end

end

j=B(jPos);

%{

Таким образом, выбор jPos из списка freePosInd(1:freePos) в текушей строке

закроет минимальное число свободных позиций в оставшихся строках

%}

else % if minFreePos>0

%{

Если в строке нет свободных позиций (minFreePos=0), то закрываем ветвь

поиска и инкрементируем значение счетчика повторных вычислений

%}

sSame=sSame+1;

simCount4=simCount4+1;

totSimCount=totSimCount+1;

%{

Восстановим значения исходных массивов и вернемся назад на начало цикла

while jCol<=colPos

%}

A=Az;

B=Bz;

Q=Qz;

L=Lz;

E=Ez;

totPos=zPos;

%{

Если количество внутренних повторов превышает допустимую границу

sameRepeatBound, то выполняем прерывание и епереходим на начало

цикла while jCol<=colPos

%}

if sSame>sameRepeatBound

sSame=0;

break % Переход на начало цикла

% for jCol=1:baseFreePos

end

end % if freePosAr(1)>0

end %while totPos < n

if processInd ==0 % Выход из цикла while jCol<=colPos

break

end

%{

Если число повторных вычислений simCount4 внутри цикла while jCol<=colPos

превышает пороговое значение repeatBound4 то данный цикл прерывается

%}

if simCount4 > simBound4

break

end

end % while jCol<=colPos

if processInd==0

break

end

if totSimCount > totSimBound

processInd=0;

break

end

%{

Если, последовательно выполнив соответствующие процедуры в блоках 1,2,3,4

мы не получаем решение, то повторяем поиск решения начиная с блока 2

%}

if simCount4 > simBound4

A=Ay;

B=By;

C=Cy;

D=Dy;

Q=Qy;

L=Ly;

rAr=rAr\_y;

tAr=tAr\_y;

totPos=yPos;

eventInd=2;

break

end

if totSimCount > totSimBound

falseNegative=falseNegative+1;

end

end %for iRow=1:nRow

otherwise

processInd =0;

end % switch eventInd

if processInd==0

break

end

end % while processInd==1

toc

nDisp =50;

if n< nDisp

nDisp =n;

disp('Positions of all Queens on the chesboard:');

else

disp('Positions of the first 50 Queens on the chesboard:');

end

Q(1: nDisp)

tStr = sprintf(' The size of Composition = %d', nComp);

disp(tStr);

nComp

% Сохраним сформированную композицию

outputFileName= 'kQueens\_Test\_Composition.mat';

save(outputFileName,'Q');

iInfo=['Composition is saved in file: ' outputFileName];

disp(iInfo);