%Generation\_kQueens\_Composition

%{

Le programme est conçu pour former une composition de k reines, qui

distribué sur un échiquier de taille n x n. Par composition on entend une distribution aléatoire de k reines sur un échiquier arbitraire de taille *n x n*, de telle sorte que trois conditions du problème soient remplies: *dans chaque ligne, dans chaque colonne, ainsi que sur les diagonales gauche et droite passant par la position où se trouve la reine, il n'y a pas plus d'une reine.*

%}

%{

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”.

%}

%{

Auteur et développeur du projet - *Grigoryan Eros (EricGrig), 2019*

Je serai heureux si des sections du code, ou l'ensemble du programme dans son ensemble, seront utilisés à des fins scientifiques ou à des fins éducatives. Dans le même temps, je vous serai reconnaissant de bien vouloir vous référer à ma publication. C'est un élément culturel et un signe de respect mutuel.

Pour une utilisation commerciale d'une partie quelconque du code du programme, ou de l'ensemble du programme dans son ensemble, le consentement écrit de l'auteur est requis.%}

%

Pour que le programme fonctionne, il faut préciser la taille du côté du damier (*n*)

%}

n=100;

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

disp(tStr);

nx=n-1; % *nx* – taille maximale de la composition.

% *n2* - taille des tableaux de contrôle.

n2=2\*n;

%{

*nFix* - Une valeur fixe pour la taille de la matrice de solution. Si un *n <nFix*, alors l'exécution de la décision est transférée au *bloc-3*, en contournant les blocs- 1 et 2.

%}

nFix=17;

%{

*bound\_1\_Ar, bound\_2\_Ar* - tableaux de valeurs pour *eventBound1* et *eventBound2*

à *n <30*. Ces valeurs sont déterminées sur la base d'expériences informatiques.

%}

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];

%{

Calcul des valeurs *eventBound1* et *eventBound2* sur la base d'équations régression. Ces résultats sont obtenus sur la base d'expériences informatiques.

%}

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

%{

Les valeurs limites *simBound1*, *simBound2* et *simBound4* définissent le nombre maximum de recalculs dans les *blocs*- 1, 2 et 4.

%}

simBound1=3;

simBound2=5;

simBound4=10;

%{

*totSimBound* - valeur limite pour comptabiliser le nombre total de tous

recalculs.

%}

totSimBound=1000;

% Définissons une taille de composition aléatoire *nComp --> (1, ... ,n-1)*

nComp=randi(nx);

%{

L'algorithme continuera jusqu'à ce que les reines *nComp* soient distribuées sur l'échiquier.

%}

% Remettons à zéro le tableau *Q(1 : n)*

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

% Pour le travail, nous mettrons à zéro les tableaux de contrôle:

% *A(1:n)* - pour contrôler les indices des lignes libres.

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

% *B(1:n)* - pour surveiller les index des colonnes libres.

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

% *C(1:n2), D(1:n2)* - pour contrôler l'occupation des projections diagonales.

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

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

% *totPos* - compteur du nombre total de tous les recalculs.

totPos=0;

%{

Réinitialisez les compteurs de répétition pour chaque événement: *simCount1, simCount2, simCount4* - compteurs de répétition dans les blocs 1, 2 et 4.

*totSimCount* - compteur du nombre total de toutes les répétitions.

%}

simCount1=0;

simCount2=0;

simCount3=0;

simCount4=0;

totSimCount=0;

%{

Dans ce programme de génération de composition, les calculs commencent à partir du premier bloc. (*eventInd = 1).*

%}

eventInd=1;

% *swiInd* is Switch Index - «switch» pour quitter la boucle.

processInd =1;

%{

Tous les événements se déroulent dans la boucle *while swiInd==1*, jusqu'à ce qu'une solution soit reçue.

%}

tic

while processInd==1

% La variable *eventInd* sert d'interrupteur à bascule entre 4 événements.

switch eventInd

case 1

%{

L'algorithme *rand\_set & rand\_set*.

La composition se forme jusqu'à la taille *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

% Trouvez les indices d'origine des lignes libres restantes dans la matrice de solution.

A=find(A==0);

nFreeRow=length(A);

%{

Trouvez les indices d'origine des colonnes libres restantes dans la matrice de solution.

%}

B=find(B==0);

%{

Créez un tableau *L (1: nFreeRow, 1: nFreeRow)* et remplissez toutes les cellules avec une. De plus, si la cellule *L (p, q)* s'avère libre, alors au lieu de

unités que nous notons à zéro.

%}

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

%{

Créons des tableaux *rAr* et tAr pour stocker les tableaux de contrôle d'index correspondants.

%}

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

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

%{

Sur la base des informations sur les lignes libres restantes et les colonnes libres, écrivez zéro dans les cellules libres correspondantes du tableau *L.*

Nous formons des tableaux de comptabilité *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

%{

Créons des copies de sauvegarde de toutes les baies principales. Nous en aurons besoin s'il devient nécessaire de revenir au début de l'*événement-2* pour répéter

calculs (*Back Tracking*).

%}

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

% Dans ce bloc, la décision est basée sur l'algorithme *rand & rand*.

simCount2=simCount2+1;

%{

Ensuite, nous continuerons à former la branche de la recherche d'une solution basée sur les données, collectées dans le tableau *L*.

%}

if nComp<eventBound2

xEvent=nComp;

else

xEvent=eventBound2;

end

while totPos < xEvent

% Déterminez le nombre de lignes libres sur la base du tableau A.

freeRowInd=find(A>0);

freeRow=length(freeRowInd);

%{

Sélection d'un index de ligne aléatoire basé sur une liste d'index de ligne libre.

%}

selectRowInd=randi(freeRow);

iInd=freeRowInd(selectRowInd);

% Formons une liste d'indices de positions libres dans la ligne *i* du tableau *L*

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

freePos=length(freePosAr);

if freePos>0

%{

S'il y a des positions libres dans la ligne sélectionnée, nous continuons la solution. Ici, la position de la reine dans la rangée est choisie au hasard.

%}

selectPosInd=randi(freePos);

jInd=freePosAr(selectPosInd);

j=B(jInd);

% Stockez le *j-index* de la position de la reine dans le tableau de solution.

i=A(iInd);

Q(i)=j;

% On incrémente le compteur du nombre de lignes occupées par la reine.

totPos=totPos+1;

%{

Écrivez 0 dans la cellule *iInd* du tableau *A* pour corriger le fait que la ligne i du tableau init est occupée.

%}

A(iInd)=0;

%{

Écrivez *0* dans la cellule *jInd* du tableau *B* pour corriger le fait que la colonne *j* du tableau init est occupée.

%}

B(jInd)=0;

%{

Modifiez les cellules correspondantes des tableaux interdits *C* et *D* en utilisant valeurs réelles des indices *(i, j)*.

%}

rx=n+j-i;

tx=j+i;

C(rx)=1;

D(tx)=1;

%{

Modifiez les cellules correspondantes du tableau *L* en utilisant des indices équivalents, stockés dans les tableaux *rAr* et *tAr*.

%}

rxInd=find(rAr==rx);

L(rxInd)=1;

txInd=find(tAr==tx);

L(txInd)=1;

L(freeRowInd,jInd)=1;

else % if freePos>0

%{

S'il n'y a pas de positions libres dans la ligne considérée, alors nous avons atteint une impasse, nous devons donc fermer la branche donnée et revenir au début tout en boucle *totPos <simBound2* et répéter la formation d'une nouvelle branche de recherche. Mais avant cela, nous devons restaurer toutes les baies requises en fonction de sauvegardes (*Back Tracking*).

%}

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

%{

Mettons à zéro les tableaux de contrôle et transférons le processus à *l'événement-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;

elseif totPos>=xEvent

eventInd=3;

end

%{

Nous avons terminé la deuxième partie de la formation de la branche de recherche et atteint le niveau, lorsque les reines *simBound2* sont correctement réparties dans la matrice de décision. Passons à la troisième étape.

%}

case 3

simCount3=simCount3+1;

%{

Ensuite, nous exclurons les lignes occupées et les colonnes occupées de la considération. Formons une nouvelle matrice compacte *L* comme l'intersection du nombre de lignes restantes et du nombre des colonnes restantes. Pour ce faire, recherchez les indices des lignes libres restantes, selon le tableau de comptabilisation des lignes occupées *A*.

%}

T=find(A>0);

A=A(T);

nRow=length(T);

% Définissons le tableau d'indices des colonnes libres de la même manière.

T=find(B>0);

B=B(T);

%{

Créez un tableau *L (1: m, 1: m)* et remplissez toutes les cellules avec une. De plus, si la cellule *L (p, q)* se révèle être libre, alors au lieu d'une seule, nous écrivons dans cette cellule zéro.

%}

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

%{

Créons des tableaux pour stocker les index de conformité avec les tableaux de contrôle.

%}

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

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

% Créons des tableaux pour tenir compte de la liste cumulative des restrictions.

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

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

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

%{

Sur la base des informations sur les lignes libres restantes et les colonnes libres, nous écrivons zéro dans les cellules libres correspondantes du tableau *L*. Former les tableaux *Cs, Ds, Bs*, ainsi que les tableaux comptables *rAr, tAr*. Pour toutes les m lignes et, en conséquence, pour les positions libres restantes dans ces lignes, formez une liste cumulative de contraintes pour les projections diagonales *Cs* gauche et *Ds* droite, ainsi que pour les projections de colonne (*Bs*).

%}

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

% Calculons la somme des éléments de chaque ligne du tableau *L*.

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

%{

Triez les valeurs de somme dans l'ordre croissant du nombre de positions libres dans chaque ligne.

%}

[sumSort,rowRangInd]=sort(rowSum);

%{

Ici, dans le tableau *rowRangInd*, les index de ligne avec un nombre croissant de positions libres dans la ligne. S'il s'avère que dans toutes les lignes restantes "*collectées*" dans le tableau *L* il y a des positions libres, alors le tableau *rowRangInd* sera utilisé plus loin, dans le *bloc-4*.

%}

if sumSort(1)>0

%{

Ici, *sumSort (1)* est le nombre minimum de positions libres dans la liste de toutes les lignes du tableau *L (m, m)*. Si le nombre minimum de positions libres > 0, alors nous continuons à construire la branche de recherche, car jusqu'à cette étape, la branche construite restait prometteuse.

Créons un tableau de contrôle de comptabilité *E* de taille *nRow x nRow*, dans chaque cellule dont nous stockons la valeur cumulée des tableaux cumulatifs de restrictions.

%}

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

%{

Nous calculons et stockons dans *E* la valeur cumulée des tableaux cumulatifs de contraintes.

%}

for p=1:nRow

for q=1:nRow

r=rAr(p,q); % Index *r* for array *Cs*

t=tAr(p,q); % Index *t* for array *Ds*

j=B(q); % Index *j* for array *Bs*

if r>0 && t>0

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

end

end

end

% De plus, au lieu des tableaux *Cs, Ds, Bs*, nous utiliserons le tableau *E*

%{

Avant de passer à l'événement suivant, sauvegardons des copies de ces tableaux pour les réutiliser.

%}

Az=A;

Bz=B;

Qz=Q;

Lz=L;

Ez=E;

zPos=totPos;

% Ensuite, passons à *event- 4*

eventInd=4;

else % *if sumSort(1)>0*

%{

S'il s'avère que parmi les lignes restantes, il y a une ligne dans laquelle il n'y a pas de positions libres, alors nous restaurons les valeurs initiales des tableaux et transférons le contrôle à *event-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);

%{

Déterminez la valeur (d'origine) correspondante de l'index de ligne dans le tableau *L* en utilisant l'index des données inite.

%}

initRowInd=A(selectRowInd);

% Déterminez le nombre de positions libres dans la ligne sélectionnée.

T=L(selectRowInd,:);

baseFreePosInd=find(T==0);

baseFreePos=length(baseFreePosInd);

%{

De plus, ici, au niveau de base, à l'intérieur de la ligne considérée avec la valeur minimale actuelle du nombre de positions libres dans la ligne, nous allons séquentiellement, dans une boucle, considérer chaque position libre.

%}

for jCol=1:baseFreePos

%{

Attribuons i le nombre réel de la ligne sélectionnée (en fonction de l'index des données source).

%}

i=initRowInd;

jPos=baseFreePosInd(jCol);

%{

Affectez *j* à la valeur de la position libre sélectionnée (selon l'index des données source).

%}

j=B(jPos);

%{

Affectons à *minRowInd* l'index de ligne dans le tableau *L (1: nRow, 1: nRow)* avec le nombre minimum de positions libres dans la ligne.

%}

minRowInd=selectRowInd;

% *Event-4.* Le début de la partie principale de l'algorithme.

sSame=0;

while totPos < nComp

%{

Pour la première étape de ce cycle, les valeurs *i*, *j* sont définies ci-dessus. Stockez le *j-index* de la position de la reine dans le tableau de solution.

%}

Q(i)=j;

% On incrémente le compteur du nombre de positions occupées par les reines.

totPos=totPos+1;

%{

Vérifions si une solution complète est formée, puis nous terminons les calculs.

%}

if totPos==nComp

totSimCount=totSimCount+1;

processInd=0;

break

end

%{

Nous avons terminé un autre cycle de détermination des indices pour l'emplacement de la reine et placé la reine dans la cellule *(i, j)* de la matrice de décision. Après cela, nous devons changer les cellules correspondantes dans tous les tableaux de contrôle, en tenant compte des indices *(minRowInd, colInd)* du tableau *L*.

%}

A(minRowInd)=0;

B(jPos)=0;

%{

Modifiez les cellules correspondantes du tableau L en utilisant les indices équivalents stockés dans les tableaux rAr et *tAr*.

%}

rx=n+j-i;

tx=j+i;

rxInd=find(rAr==rx);

L(rxInd)=1;

txInd=find(tAr==tx);

L(txInd)=1;

%{

Nous décrémentons la valeur du tableau de contrôle cumulatif, c'est-à-dire réduisons «*l'effet d'influence*» des positions libres dans la ligne sélectionnée, après que la reine y a été placée.

%}

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

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

%{

Écrivons 1 à toutes les cellules actives de la colonne *colInd*. Les cellules actives sont spécifiées par le tableau *A1*.

%}

A1=find(A>0);

L(A1,jPos)=1;

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

%{

Déterminez l'indice de ligne *minRowInd* avec le nombre minimum de positions libres.

%}

[freePosAr,rowIndAr]=sort(rowSum);

if freePosAr(1)>0

%{

Si deux lignes ont le même nombre minimum de positions libres, choisissez au hasard l'index de l'une de ces lignes.

%}

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

randPos=1;

else

randPos=randi(2);

end

minRow=rowIndAr(randPos);

minRowInd=A1(minRow);

i=A(minRowInd);

% Déterminez le nombre de positions libres dans cette ligne.

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

freePos=length(freePosAr);

%{

Choisissez parmi ces positions celle qui ferme le nombre minimum de positions libres dans les lignes restantes. Pour ce faire, nous utilisons le tableau *E (m, m)*.

%}

if freePos==1

jPos=freePosAr(1);

else

T=E(minRowInd,freePosAr);

[tSort,tInd]=sort(T);

if tSort(1)<tSort(2)

jPos=freePosAr(tInd(1));

else

jInd=randi(2);

jPos=freePosAr(tInd(jInd));

end

end

j=B(jPos);

%{

Ainsi, choisir jPos dans la liste *freePosInd (1: freePos)* dans la ligne courante fermera le nombre minimum de positions libres dans les lignes restantes.

%}

else % if minFreePos>0

%{

S'il n'y a pas de positions libres dans la ligne, (*minFreePos = 0*), fermez la branche de recherche et incrémentez la valeur du compteur de recalcul.

%}

sSame=sSame+1;

simCount4=simCount4+1;

totSimCount=totSimCount+1;

%{

Restaurons les valeurs des tableaux d'origine et revenons au début de la boucle: *while jCol <= colPos*

%}

A=Az;

B=Bz;

Q=Qz;

L=Lz;

E=Ez;

totPos=zPos;

%{

Si le nombre de répétitions internes dépasse la limite autorisée de *sameRepeatBound*, alors nous interrompons et allons au début de la boucle: *while jCol <= colPos.*

%}

if sSame>sameRepeatBound

sSame=0;

break % Go to the beginning of the cycle:

% *for jCol=1:baseFreePos*

end

end % *if freePosAr(1)>0*

end % *while totPos < n*

if processInd==0 % Exiting the loop: w*hile jCol <= colPos*

break

end

%{

Si le nombre de calculs répétés *simCount4* à l'intérieur de la boucle: *while* *jCol <= colPos* dépasse le seuil de *repeatBound4*, alors cette boucle est interrompue.

%}

if simCount4 > simBound4

break

end

end % while jCol<=colPos

if processInd==0

break

end

if totSimCount > totSimBound

processInd=0;

break

end

%{

Si, après avoir exécuté séquentiellement les procédures appropriées dans les *blocs- 1,2,3,4* nous n'obtenons pas de solution, alors nous répétons la recherche d'une solution à partir du *bloc-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 swiInd==1*

toc

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

disp(tStr);

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

disp(Q(1:nDisp))

% Sauvegardons la composition générée.

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

save(outputFileName,'Q');

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

disp(iInfo);