Universidade de São Paulo
Instituto de Física de São Carlos
Física Estatística Computacional

Projeto 4

Modelos de Crescimento

Rafael Fernando Gigante - **NºUSP** 12610500

20 de maio de 2023

AUTÔMATOS CELULARES DETERMINÍSTICOS

TAREFA 1

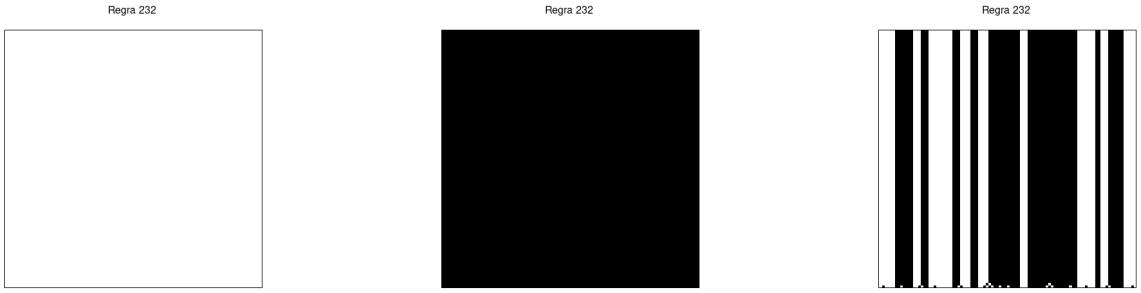
Os autômatos celulares são sistemas evolutivos baseado em regras simples. Eles são formados por uma rede de células, onde cada célula ocupa uma posição e possui um determinado estado inicial. A evolução de cada célula é dada em função do seu estado anterior e do estado anterior de suas células vizinhas. A partir de um ponto inicial e, baseado em uma regra que determina as condições para mudança de estado, a célula com estado inicial, ao ser alterado, interfere na célula vizinha, desencadeando um efeito evolutivo.

Nesta tarefa iremos considerar uma cadeia de L sítios (L $\sim$ 100-200) e que cada sítio da rede é ocupado por uma variável binária $\{b_i, i = 1, ..., L\}$ que assume valores 0 ou 1. Dessa forma, a configuração em um instante t será dada por $C_t = \{b_1^t, b_2^t, ..., b_L^t\}(b_i^t = 0, 1)$. As regras de evolução do autômato utilizadas envolvem apenas três vizinhos próximos, ou seja:

$$b_i^{t+1} = f_A(b_{i-1}^t, b_i^t, b_{i+1}^t), \;\; i = 1, ...L \tag{1}$$

Essa trinca pode ter 8 configurações possíveis, então podemos ter $2^8 = 256$ regras de evolução possíveis.

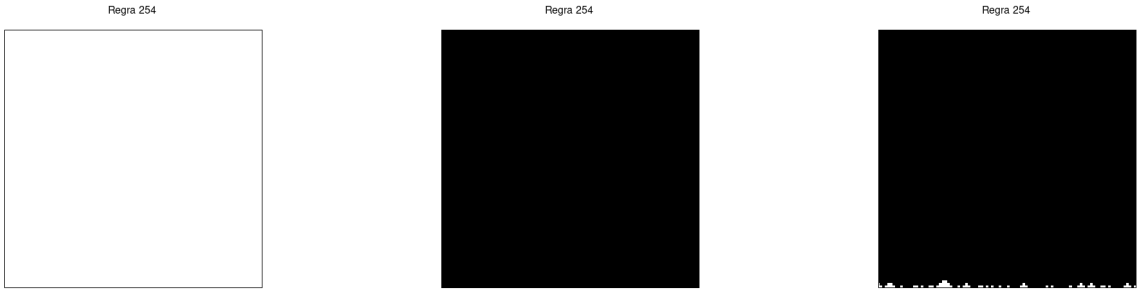
Iremos testar algumas regras de evolução para três configurações iniciais diferentes: todos os sítios com valor 0, $C_0^{(1)} = \{0, ..., 0\}$; todos os sítios com valor 1, $C_0^{(2)} = \{1, ..., 1\}$; e todos os sítios com valores 0 e 1 escolhidos de forma aleatória, $C_0^{(3)} = \{b_1, ..., b_L\}$. Para simplicidade o sistema a ser considerado será periódico e unidimensional.



(a) $C_0^{(1)} = \{0, ..., 0\}$ (b) $C_0^{(2)} = \{1, ..., 1\}$ (c) $C_0^{(3)} = \{b_1, ..., b_L\}$

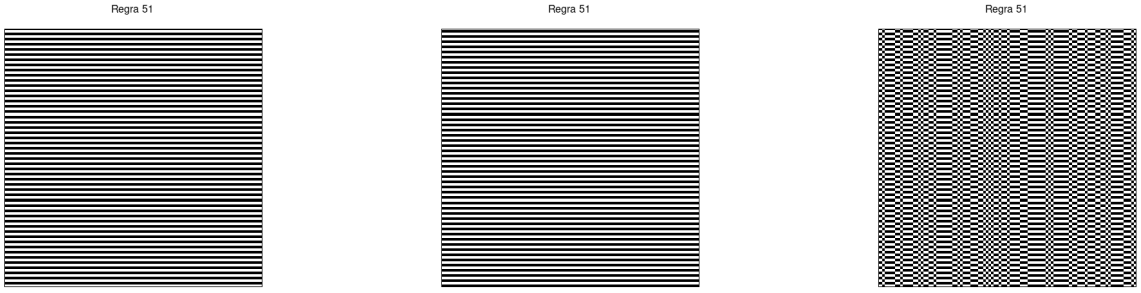
Figura 1: Evolução de um autômato celular dada pela regra 232

As regras de evolução podem ser divididas em 4 classes que possuem propriedades distintas. Autômatos celulares de classe 1 convergem rapidamente para um estado uniforme. Os de classe 2 convergem para um periódico ou estável. Autômatos celulares de classe 3 evoluem para um estado permanente pseudo-aleatório ou caótico. Já os de classe 4 formam áreas de estados periódicos ou estáveis, mas também são formadas estruturas que interagem entre si de forma bem complicada.



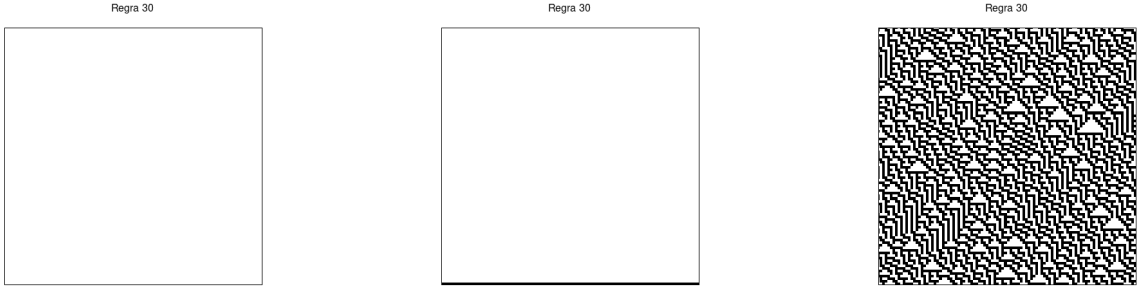
(a) $C_0^{(1)} = \{0, ..., 0\}$ (b) $C_0^{(2)} = \{1, ..., 1\}$ (c) $C_0^{(3)} = \{b_1, ..., b_L\}$

Figura 2: Evolução de um autômato celular dada pela regra 254



(a) $C_0^{(1)} = \{0, ..., 0\}$ (b) $C_0^{(2)} = \{1, ..., 1\}$ (c) $C_0^{(3)} = \{b_1, ..., b_L\}$

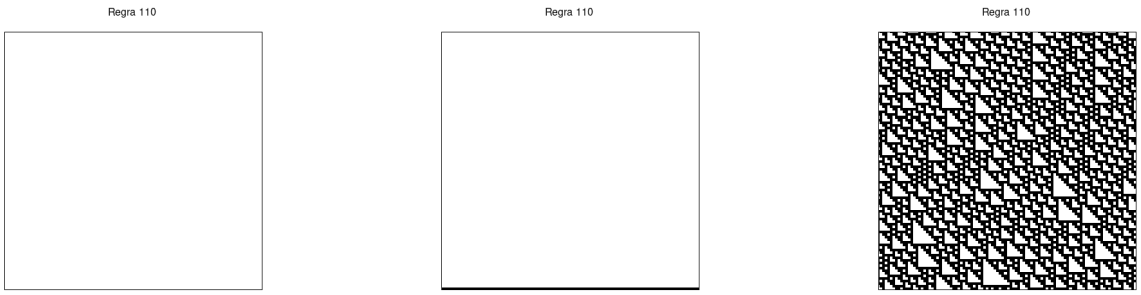
Figura 3: Evolução de um autômato celular dada pela regra 51



(a) $C_0^{(1)} = \{0, ..., 0\}$ (b) $C_0^{(2)} = \{1, ..., 1\}$ (c) $C_0^{(3)} = \{b_1, ..., b_L\}$

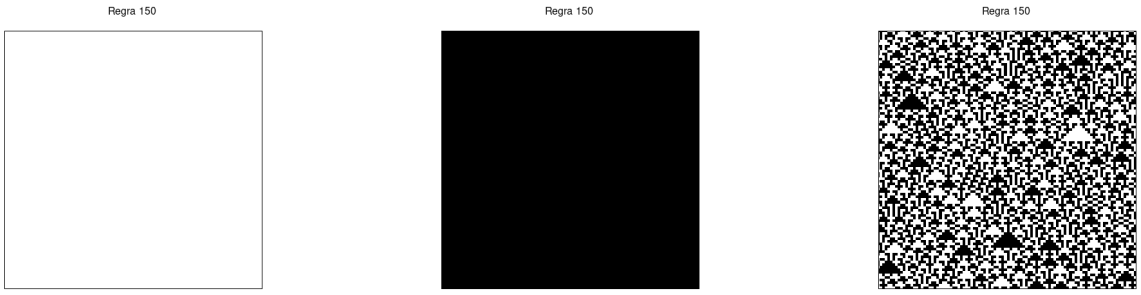
Figura 4: Evolução de um autômato celular dada pela regra 30

Com isso, podemos analisar graficamente os resultados obtidos para seis regras distintas e classificá-las de acordo com as quatro classes citadas anteriormente. A regra 254 (Figura 2) faz as células convergirem para um estado homogêneo estável, então ela é de classe 1. A regra 232 (Figura 1) faz a rede evoluir para um estado estável, porém não homogêneo, enquanto que a regra 51 (Figura 3) faz o sistema evoluir para um estado oscilatório, logo ambas as regras são de classe 2. Podemos ver que a evolução dos sistemas dados pelas regras 30 e 150 (Figuras 4 e 6) evoluem de forma completamente caótica, todas as estruturas formadas são rapidamente destruídas pelas células em volta, dessa forma essas regras são de classe 3. Para a regra 110 (Figura 5) vemos a formação de várias estruturas e de



(a) $C_0^{(1)} = \{0, ..., 0\}$ (b) $C_0^{(2)} = \{1, ..., 1\}$ (c) $C_0^{(3)} = \{b_1, ..., b_L\}$

Figura 5: Evolução de um autômato celular dada pela regra 110



(a) $C_0^{(1)} = \{0, ..., 0\}$ (b) $C_0^{(2)} = \{1, ..., 1\}$ (c) $C_0^{(3)} = \{b_1, ..., b_L\}$

Figura 6: Evolução de um autômato celular dada pela regra 150

algumas partes estáveis, portanto essa regra é de classe 4.

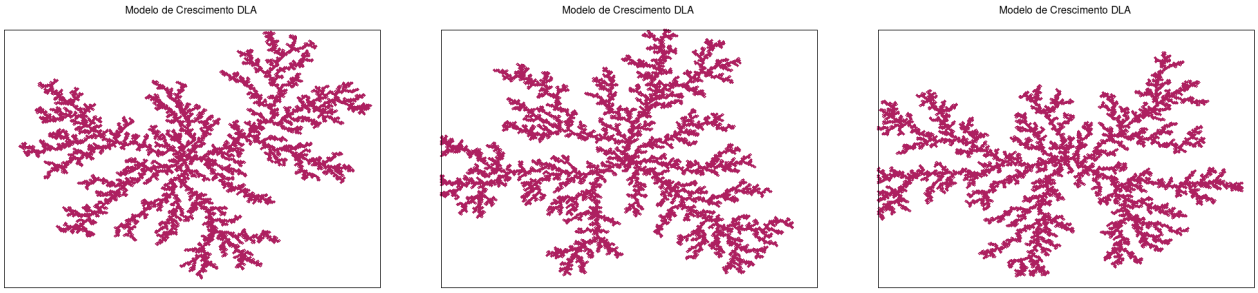
MODELOS DE CRESCIMENTO COM ALEATORIEDADE

TAREFA 2

Nesta tarefa realizaremos um programa que simula o crescimento de um agregado de partículas em duas dimensões de acordo com a dinâmica do DLA (Diffusion Limited Agregation). Consideramos que temos uma partícula fixa na origem (x,y) = (0,0), então iniciaremos novas partículas em um determinado raio R_{inic} (Pelo menos 5 unidades de distância da partícula pertencente ao agregado mais distante em relação à origem). Então, a partícula irá se mover de forma aleatória até que ela encontre alguma partícula pertencente ao agregado ou se afaste uma distância de $R_{final} = 1.5R_{inic}$ da origem.

Na figura 7 pode-se observar o resultado obtido ao realizar a simulação para um número de 10 mil partículas lançadas. Para cada crescimento foi utilizada uma semente diferente da função SRAND do Fortran, desse modo a função gera uma ordem diferente de números pseudo-aleatórios e isso faz com que a trajetória das partículas lançadas seja diferente e por consequência a forma do agregado.

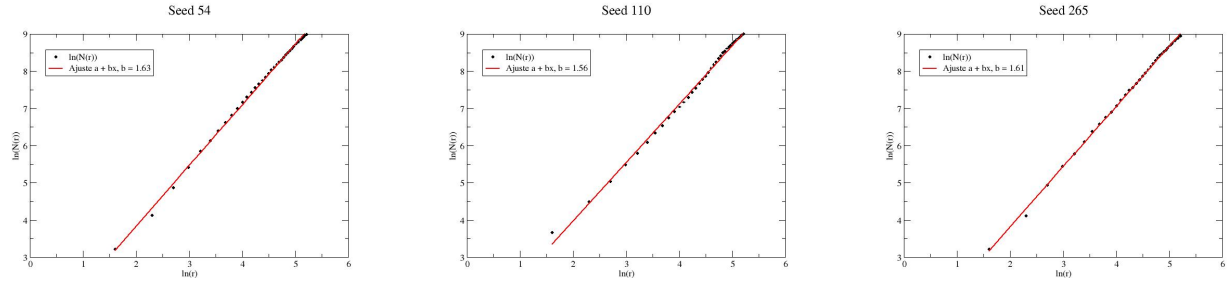
Algo interessante que podemos fazer com os resultados obtidos é determinar a dimensão fractal



(a) Seed 54 (b) Seed 110 (c) Seed 265

Figura 7: Crescimentos de um agregado de partículas em duas dimensões para diferentes seeds

dos três agregados. Para isso, a partir da origem traçamos círculos de raio r variável, então contamos o número de partículas N(r) inseridas dentro do círculo. Fazendo um fitting dos pontos obtidos, temos que $N(r) \sim r^{D_f}$, onde D_f é denominado de dimensão fractal. Para os modelos de crescimento DLA, espera-se obter um fractal com $1 < D_f < 2$ da ordem de $D_f \sim 1.65$.



(a) $D_f = 1.63$ (b) $D_f = 1.56$ (c) $D_f = 1.61$

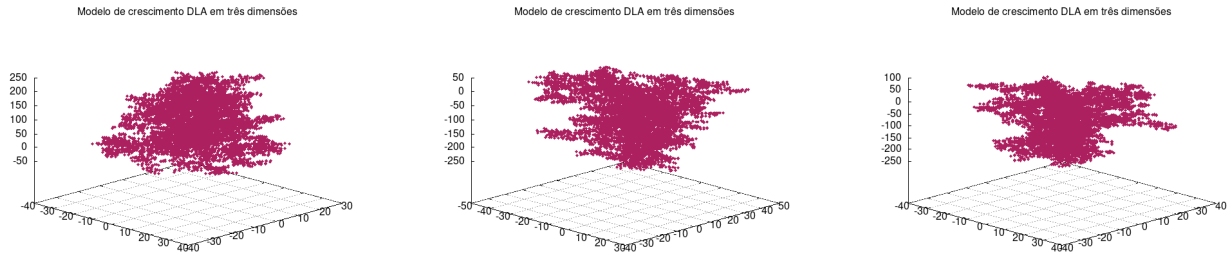
Figura 8: Determinação da dimensão fractal dos agregados obtidos anteriormente

Na figura 8 é possível observar os gráficos realizados para os três crescimentos mencionados anteriormente. Colocando-se o gráfico na escala loglog devemos considerar uma função da forma $y(x) = A + Bx$ para realizar o ajuste, de modo que o coeficiente B será a nossa dimensão fractal D_f. Quantos aos resultados, obtemos valores de 1.56 até 1.63, fazendo a média deles obtemos que $\bar{D_f} = 1.6$, sendo portanto comparável com o valor esperado da ordem de 1.65.

TAREFA 3

Nesta tarefa iremos refazer o mesmo modelo de crescimento DLA realizado na tarefa 2 para o caso em três dimensões. Fazendo as devidas adaptações ao código da tarefa 2, produzimos os seguintes agregados ilustrados na Figura 9.

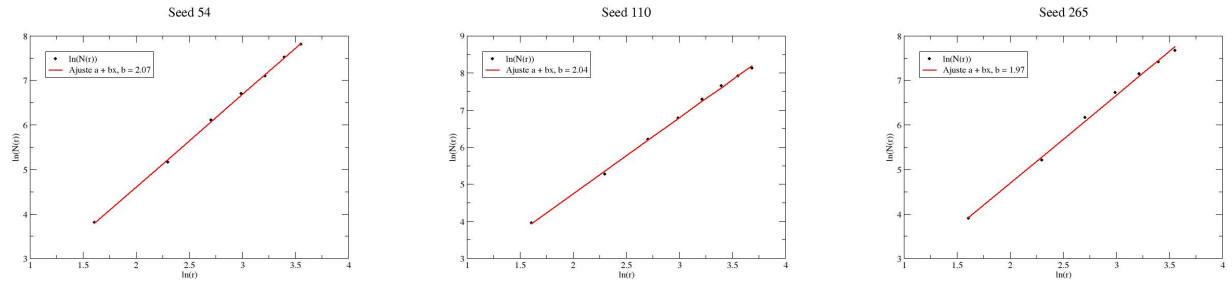
Da mesma forma que na tarefa 2, utilizamos os resultados obtidos para determinar a dimensão fractal do agregado. Para o caso em três dimensões, espera-se obter valores de D_f tal que $2 < D_f < 3$.



(a) Seed 54 (b) Seed 110 (c) Seed 265

Figura 9: Crescimentos de um agregado de partículas em três dimensões para diferentes seeds

Na figura 10, pode-se observar que apenas uma das sementes teve um valor abaixo do previsto, apesar de o valor obtido para as outras estar bem próximo de 2. A razão disso é devido à baixa densidade de partículas no agregado, mesmo iniciando um número de 50 mil partículas não são todas que se agregam. Devido à isso, o espaço ocupado pelo agregado é consideravelmente menor do que o espaço em que ele está contido.



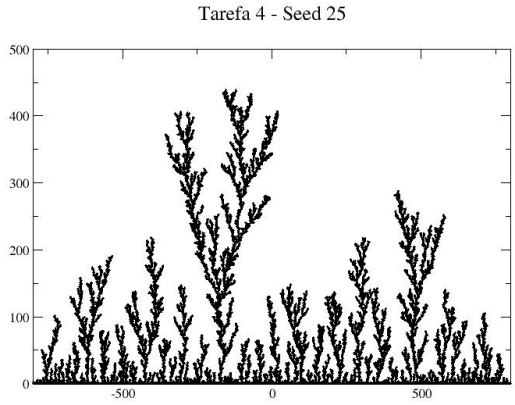
(a) $D_f = 2.07$ (b) $D_f = 2.04$ (c) $D_f = 1.97$

Figura 10: Determinação da dimensão fractal dos agregados obtidos anteriormente

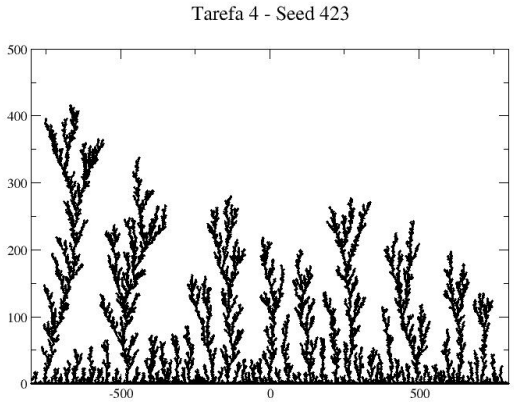
TAREFA 4

Nesta tarefa realizaremos um modelo de crescimento semelhante ao da tarefa 2, porém agora ao invés termos apenas uma semente inicial posicionada na origem teremos sementes em todos os sítios localizados no eixo x, ou seja, $(x, y)|y = 0, x, y \in \mathbf{Z}$. As partículas serão iniciadas a uma determinada distância do eixo x e em uma posição aleatória, então a partícula se moverá de forma aleatória até que ela encontre alguma outra partícula do aglomerado.

Para que as partículas não ficassem muito amontoadas foi feita uma grade tal que $x \in [-800, 800]$. Os resultados obtidos podem ser vistos na Figura 10, onde foi realizado dois crescimentos distintos. Nota-se que o gráfico final obtido desses crescimentos assemelha-se com o efeito observado em descargas elétricas de fios condutores.



(a) Seed 54

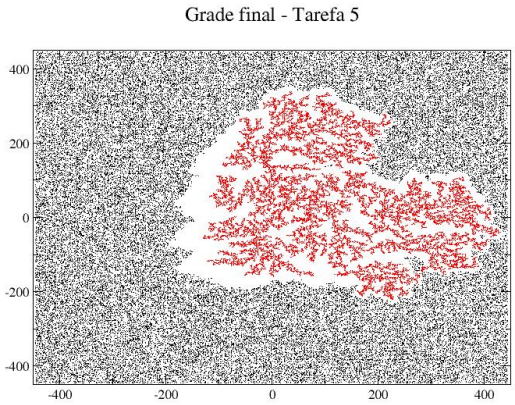


(b) Seed 110

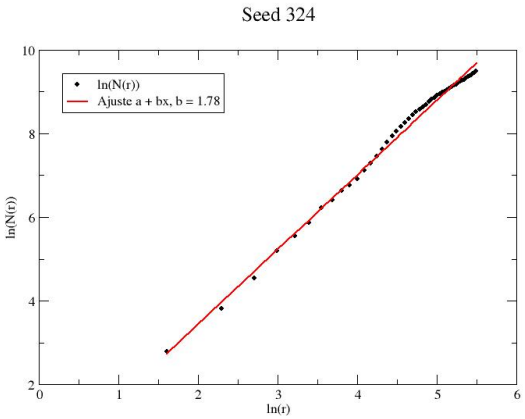
Figura 11: Crescimentos de um agregado de partículas para diferentes seeds

TAREFA 5

Nesta tarefa faremos um modelo de crescimento que simula revoluções populares. Neste caso, iniciamos nossa rede já com partículas espalhadas por ela, onde todos os sítios do reticulado serão sorteados com uma probabilidade p de conter uma partícula. Feito isso, teremos uma única partícula do agregado na origem em (0,0), que passará a se mover de forma aleatória enquanto o restante das partículas permanecem em sua posição inicial. Durante o movimento deste agregado ele irá alcançar as outras partículas, de modo que estas se juntam a ele e passam a se movimentar de maneira conjunta.



(a) Grade final obtida



(b) Dimensão fractal $D_f = 1.78$ (seed 324)

Figura 12: Resultados obtidos para a tarefa 5

Na figura 12 (a) pode-se observar o resultado final da rede após a realização da simulação. Para uma probabilidade de p = 0.1 a dimensão fractal da forma crescida deve ser da ordem de 1.7, o que é

condizente com o resultado obtido de $D_f = 1.78$.

APÊNDICE

(a) Código da Tarefa 1

```
program tarefa_1 !aut mato celular determ nistico
    integer, parameter :: L = 100
    integer :: valor_regra
    integer, dimension(1:L) :: Cc, Cn, mais, menos !C current e C next
    character(8) :: string_regra

    !Inputs do usu rio para definir a regra e a cadeia inicial
    write(*,*) 'Digite o n mero da regra (0 - 256): '
    read(*,*) valor_regra !Inteiro de 0 at  256
    write(string_regra,'(B8.8)') valor_regra !Transforma o valor da regra em
   bin rio e o guarda em uma string
    write(*,*) 'Escolha a cadeia inicial (0, 1 ou aleat rio (2)):'
    read(*,*) j !Definindo a cadeia inicial |0, 1|

    open(1, file='data.out')

    !Defini  o da periodicidade da cadeia
    do i = 2, L-1
        mais(i) = i + 1
        menos(i) = i - 1
    end do
    mais(L) = 1
    mais(1) = 2
    menos(1) = L
    menos(L) = L-1

    !Configura es iniciais
    if (j == 0) then
        Cc = 0
    else if (j == 1) then
        Cc = 1
    else
        iseed = 25
        call srand(iseed)
        do i = 1, L
            a = rand()
            if (a < 0.5d0) then
                Cc(i) = 0
            else
```

```
            Cc(i) = 1
        end if
    end do
end if

write(1,*) Cc

!Itera  es da cadeia
do j = 1, 100
    do i = 1, L
        Cn(i) = func_regra(Cc(menos(i)), Cc(i), Cc(mais(i)), string_regra)
    end do

    Cc = Cn
    write(1,*) Cc

end do

contains

!Fun  o que recebe os valores de 3 s tios da cadeia e a string contendo o
 n mero da regra
!e ent o a aplica para o valor dos s tios a, b e c
integer function func_regra(a,b,c,string)
    integer :: a, b, c, valor
    character(8) :: string

    if (a+b+c == 0) then
        read(string(8:8), '(I10)') valor
        func_regra = valor

    else if (a+b+c == 1) then
        if (a == 0 .and. b == 0) then
            read(string(7:7), '(I10)') valor
            func_regra = valor
        else if (a == 0 .and. c == 0) then
            read(string(6:6), '(I10)') valor
            func_regra = valor
        else
            read(string(4:4), '(I10)') valor
            func_regra = valor
        end if
    else if (a+b+c == 2) then
        if (b == 1 .and. c == 1) then
            read(string(5:5), '(I10)') valor
```

```
                func_regra = valor
            else if (a == 1 .and. c == 1) then
                read(string(3:3), '(I10)') valor
                func_regra = valor
            else
                read(string(2:2), '(I10)') valor
                func_regra = valor
            end if
        else
            read(string(1:1), '(I10)') valor
            func_regra = valor
        end if
    end function


end program tarefa_1
```

(b) Código da Tarefa 2

```
program tarefa_2 !Modelo de crescimento DLA 2d
    integer, parameter :: L = 400, N = 15000
    integer, dimension(-L:L,-L:L) :: grade
    integer, dimension(0:3) :: px = (/1,-1,0,0/), py = (/0,0,1,-1/)
    real(8), parameter :: pi = acos(-1.d0)
    real(8) :: Rinic, Rfin, theta, soma
    logical :: cond = .true.

    call srand(265)

    !Defini  o dos raios inicial e final
    Rinic = 5.0d0
    Rfin = 1.5d0 * Rinic

    open(1, file='seed.out')
    open(2, file='fractal.out')

    grade(0,0) = 1
    write(1,*) 0, 0

    do i = 1, N
        cond = .true.
        soma = 0

        !Posi  o que as part culas  s  o iniciadas
        theta = 2 * pi * rand()
        ix = Rinic * cos(theta)
```

```
28          iy = Rinic * sin(theta)
29
30          do while (cond .eqv. .true.)
31
32              !random walk para a part cula iniciada
33              ia = 4 * rand()
34              ix = ix + px(ia)
35              iy = iy + py(ia)
36
37              !Checagem no entorno da part cula iniciada
38              do j = -1, 1
39                  do k = -1, 1
40                      soma = soma + grade(ix + j, iy + k)
41                  end do
42              end do
43
44              !Dist ncia da part cula iniciada em  rela   o    origem
45              d = sqrt(real(ix ** 2 + iy ** 2))
46
47              !C o n d i   o para caso a part cula se afaste demais do agregado
48              if (d >= Rfin) then
49                  cond = .false.
50              !C o n d i   o para a part cula se juntar ao agregado
51              else if (soma >= 1) then
52                  grade(ix, iy) = 1
53                  write(1,*) ix, iy
54                  cond = .false.
55                  !R e d e f i n i   o do raio inicial para iniciar novas part culas
56                  if (d > Rinic) then
57                      Rinic = d + 5
58                      Rfin = Rinic * 1.5
59                  end if
60              end if
61          end do
62      end do
63
64      !Trecho de c d i g o  nada otimizado para realizar a contagem de part culas
    para
65      !determinados valores de raio
66      id = 5
67      icount = 0
68      do while (id <= Rinic)
69          do i = -id, id
70              do j = -id, id
71                  if (sqrt(real(i ** 2 + j ** 2)) <= real(id)) then
```

```
                    if (grade(i,j) == 1) then
                        icount = icount + 1
                    end if
                end if
            end do
        end do

        write(2,*) log(real(id)), log(real(icount))
        icount = 0
        id = id + 5
    end do
end program tarefa_2
```

(c) Código da Tarefa 3

```
program tarefa_3 !Modelo de crescimento DLA 3d
    integer, parameter :: L = 400, N = 50000
    integer, dimension(-L:L,-L:L,-L:L) :: grade
    integer, dimension(0:5) :: px = (/1,-1,0,0,0,0/), py = (/0,0,1,-1,0,0/), pz
   = (/0,0,0,0,1,-1/)
    real(8), parameter :: pi = acos(-1.d0)
    real(8) :: Rinic, Rfin, theta, phi, soma
    logical :: cond = .true.

    call srand(265)

    !Defini  o dos raios inicial e final
    Rinic = 5.0d0
    Rfin = 1.5d0 * Rinic

    open(1, file='seed.out')
    open(2, file='fractal.out')

    grade(0,0,0) = 1
    write(1,*) 0, 0, 0
    count = 1

    do i = 1, N
        cond = .true.
        soma = 0

        !Posi  o que as part culas  s  o iniciadas
        theta = pi * rand()
        phi = 2.d0 * pi * rand()
        ix = Rinic * sin(theta) * cos(phi)
        iy = Rinic * sin(theta) * sin(phi)
```

```
31        iz = Rinic * cos(theta)
32
33        do while (cond .eqv. .true.)
34
35            !random walk para a part cula iniciada
36            ia = 6 * rand()
37            ix = ix + px(ia)
38            iy = iy + py(ia)
39            iz = iz + pz(ia)
40
41            !Checagem do entorno da part cula iniciada
42            do j = -1, 1
43                do k = -1, 1
44                    do m = -1, 1
45                        soma = soma + grade(ix + j, iy + k, iz + m)
46                    end do
47                end do
48            end do
49
50            !Dist ncia da part cula da origem
51            d = sqrt(real(ix ** 2 + iy ** 2 + iz ** 2))
52
53            !Condi  o para caso a part cula se afaste demais
54            if (d >= Rfin) then
55                cond = .false.
56            !Condi  o para a part cula se juntar ao agregado
57            else if (soma >= 1) then
58                grade(ix, iy, iz) = 1
59                write(1,*) ix, iy, iz
60                cond = .false.
61                !Redefini  o do raio inicial
62                if (d > Rinic) then
63                    Rinic = d + 5
64                    Rfin = Rinic * 1.5
65                end if
66            end if
67        end do
68    end do
69
70    !Trecho de c digo nada otimizado para realizar a contagem de part culas
    para
71    !determinados valores de raio
72    id = 5
73    icount = 0
74    do m= 0, 10
```

```
        do i = -id, id
            do j = -id, id
                do k = -id, id
                    if (sqrt(real(i ** 2 + j ** 2 + k ** 2)) <= real(id)) then
                        if (grade(i,j,k) == 1) then
                            icount = icount + 1
                        end if
                    end if
                end do
            end do
        end do

        write(2,*) log(real(id)), log(real(icount))
        icount = 0
        id = id + 5
    end do

end program tarefa_3
```

(d) Código da Tarefa 4

```
program tarefa_4 !Efeito corona, basicamente o mesmo c digo da tarefa 2
    integer, parameter :: L = 800, N = 50000
    integer, dimension(-L:L,0:4000) :: grade = 0
    integer, dimension(0:3) :: px = (/1,-1, 0, 0/), py = (/0,0,1,-1/)
    real(8) :: Y_inic, Y_max, soma
    logical :: condition

    call srand(423)

    open(1, file='data.out')

    do i = -L, L
        grade(i,0) = 1
        write(1,*) i, 0
    end do

    Y_inic = 5
    Y_max = 1.5 * Y_inic
    do i = 1, N
        condition = .true.

        ix = (- L) + (2 * L * rand())
        iy = Y_inic


```

```
        do while (condition .eqv. .true.)

            ia = 4 * rand()
            ix = ix + px(ia)
            iy = iy + py(ia)

            do j = -1, 1
                do k = 0, 1
                soma = soma + grade(ix + j, iy - k)
                end do
            end do

            if (ix >= L .or. ix < -L .or. iy > Y_max) then
                condition = .false.
            else if (soma > 0) then
                grade(ix, iy) = 1
                write(1,*) ix, iy
                condition = .false.
                soma = 0
                if (iy == Y_inic) then
                    Y_inic = Y_inic + 5
                    Y_max = Y_inic * 1.5d0
                end if
            end if
        end do
    end do

end program tarefa_4
```

(e) Código da Tarefa 5

```
program tarefa_5 !Modelo de crescimento de revolu  es populares
    integer, parameter :: L = 1000
    real, parameter :: p = 0.1d0
    integer, dimension(-L:L,-L:L) :: grade_estac,grade_movel, grade_aux
    integer, dimension(0:3) :: px = (/1,-1,0,0/), py = (/0,0,1,-1/)
    logical :: condition = .true.

    call srand(324)

    open(1, file='grade_movel.out')
    open(2, file='grade_estac.out')
    open(3, file='fractal.out')

    grade_estac(0,0) = 0
    grade_movel(0,0) = 1
```

```
16
17      !Definindo os s t i o s que s e r o ocupados inicialmente
18      do i = -L, L
19          do j = -L, L
20              if (rand() <= 0.10) then
21                  grade_estac(i,j) = 1
22              end if
23          end do
24      end do
25
26      !Aqui definimos as coordenadas ix_count e iy_count que basicamente s o a
        l o c a l i z a o
27      !da part c u l a inicial, utilizamos uma d i s t n c i a id como r e f e r n c i a para
        observar a grade.
28      ! E n t o , basicamente ao i n v s de olhar as grades por completo para
        verificar a p o s i o das
29      !part c u l a s n s simplesmente observamos uma f r a o da grade com escala
        compar v e l ao tamanho
30      !do reticulado.
31      id = 5
32      ix_count = 0
33      iy_count = 0
34      do m = 1, 6000
35          do i = - id, id
36              do j = - id, id
37                  if (grade_movel(ix_count + i, iy_count + j) == 1) then
38                      do k = -1, 1
39                          do n = -1, 1
40                              if (grade_estac(i+k,j+n) == 1) then
41                                  grade_movel(i+k,j+n) = 1
42                                  grade_estac(i+k,j+n) = 0
43                                  !Esse trecho    para verificar se a d i s t n c i a
        da nova part c u l a agregada
44                                  !em r e l a o    part c u l a inicial    maior do
        que a d i s t n c i a id definida previamente.
45                                  s = sqrt(real(((i+k) - ix_count) ** 2 + ((j+n) -
         iy_count) ** 2))
46                                  if (s >= id) then
47                                      id = id + 5
48                                  end if
49                              end if
50                          end do
51                      end do
52                  end if
53              end do
```

```
54          end do
55
56          ia = 4 * rand()
57          do i =  ix_count - id,  ix_count + id
58              do j = iy_count - id, iy_count + id
59                  if (grade_movel(i,j) == 1) then
60                      !Nova posi  o das part culas ap s se movimentarem
61                      grade_aux(i+px(ia),j +py(ia)) = grade_movel(i, j)
62                  end if
63              end do
64          end do
65
66          ix_count = px(ia)
67          iy_count = py(ia)
68
69          grade_movel = grade_aux
70          grade_aux = 0
71      end do
72
73      do i = -(ix_count + id), (ix_count + id)
74          do j = -(iy_count + id), (ix_count + id)
75              if (grade_movel(i,j) == 1) then
76                  !Usamos a grade auxiliar para transladar o agregado de volta
    para a origem
77                  grade_aux(i - ix_count, j - iy_count) = 1
78                  write(1, *) i, j
79              end if
80
81              if (grade_estac(i,j) == 1) then
82                  write(2,*) i, j
83              end if
84          end do
85      end do
86
87      !Trecho de c digo nada otimizado para realizar a contagem de part culas
    para
88      !determinados valores de raio
89      id = 5
90      icount = 0
91      iaux = 0
92      do while (id <  L)
93          do i = -id, id
94              do j = -id, id
95                  if (sqrt(real(i ** 2 + j ** 2)) <= real(id)) then
96                      if (grade_aux(i,j) == 1) then
```

```
97                          icount = icount + 1
98                      end if
99                  end if
100             end do
101         end do
102
103         if (iaux /= icount) then
104             write(3,*) log(real(id)), log(real(icount))
105             icount = 0
106             id = id + 5
107         else
108             id = L
109         end if
110     end do
111
112 end program tarefa_5
```