Algoritmos e Estruturas de Dados II

Lucila Bento
lucila.bento [at] ime.uerj.br

As aulas serão realizadas todas as quintas e sextas

Horário:
Quintas de 08:50 às 10:30
Sextas de 10:40 às 12:20

O material do curso será compartilhado no Classroom

Não teremos monitor

A presença nas aulas não será exigida

Disposições gerais

Os alunos se comprometem a ter um comportamento ético e disciplinado ao longo do curso.

Os alunos entregarão as avaliações nos prazos solicitados e não poderão modificar as soluções enviadas.

Não serão aceitas entregas de trabalho após o prazo estabelecido.

Teremos 4 tipos de avaliações: 2 provas (P1 e P2), exercícios em aula (ES), exercícios em casa (EC) e exercício extra (EX1 e EX2).

Disposições gerais

A P1 será realizada em 12/05/23, a P2 em 30/06/23 e a PF no dia 14/07/23.

A prova de reposição será disponibilizada **somente** para aqueles que apresentarem documentação comprobatória de indisponibilidade para realizar a P1 ou a P2.

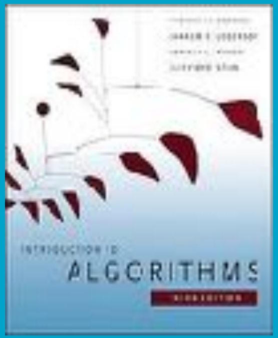
A prova de reposição será junto com a prova final, no dia 14/07/2023. Logo, aquele que fizer prova de reposição e ficar de final terá sua nota da reposição replicada para a final.

O EX1 **somente** será usado para arredondar para 7,0 a N1 que estiver entre 6,5 a 6,9 (inclusive). Tal arredondamento só será realizado se a nota do EX1 for >= 9,0. O uso do EX2 será análogo.

Nota da disciplina: NF = (P1+P2)*0,6 + EC*0,3 + ES*0,1

Bibliografia

R.L. Graham, D.E. Knuth, O. Patashnik, Concrete Mathematics, Addison-Wesley, 1989. (Matemática Concreta, LTC, 1995)



T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2009. (Algoritmos, Elsevier, 2012)

Ziviani, N. Projeto de Algoritmos, Cengage, 2008.

R. Sedgewick & K. Wayne, Algorithms, 4th Edition, Addison-Wesley, 2011.

Contextualização

- O projeto de algoritmos requer abordagens adequadas:
  - A forma como um algoritmo aborda o problema pode levar a um desempenho ineficiente,
  - Em certo casos, o algoritmo pode não conseguir resolver o problema em tempo viável.

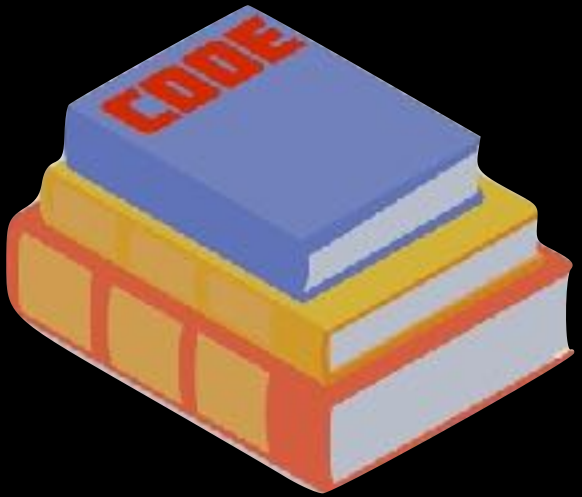
Paradigmas de projeto de algoritmos

- Indução
- Recursividade
- Força bruta e Backtracking
- Divisão e conquista
- Programação dinâmica
- Método guloso
- Algoritmos heurísticos
- Algoritmos aproximativos

P1	P1	P2	P2
Divisão e Conquista	Programação Dinâmica	Backtracking	Método Guloso

Divisão e Conquista

Algoritmos e Estrutura de Dados II



Divisão e Conquista

- É uma técnica para resolver problemas (construir algoritmos) que subdivide o problema em subproblemas menores, de mesma natureza e compõe a solução desses subproblemas, obtendo uma solução do problema original.
- Porém, quando o problema é muito pequeno ("infantil") sua solução completa deve ser especificada.

DIVIDIR PARA CONQUISTAR!!!

Exemplos

Problemas clássicos

```
inteiro Fatorial(p):
   se p ≤ 0:
      retornar 1
   senão:
      retornar
       p * Fatorial(p-1)
```

```
inteiro Fib(p):
   se p ≤ 1:
      retornar p
   senão:
      retornar
       Fib(p-1) + Fib(p-2)
```

Visões

Solução de problemas de trás para frente

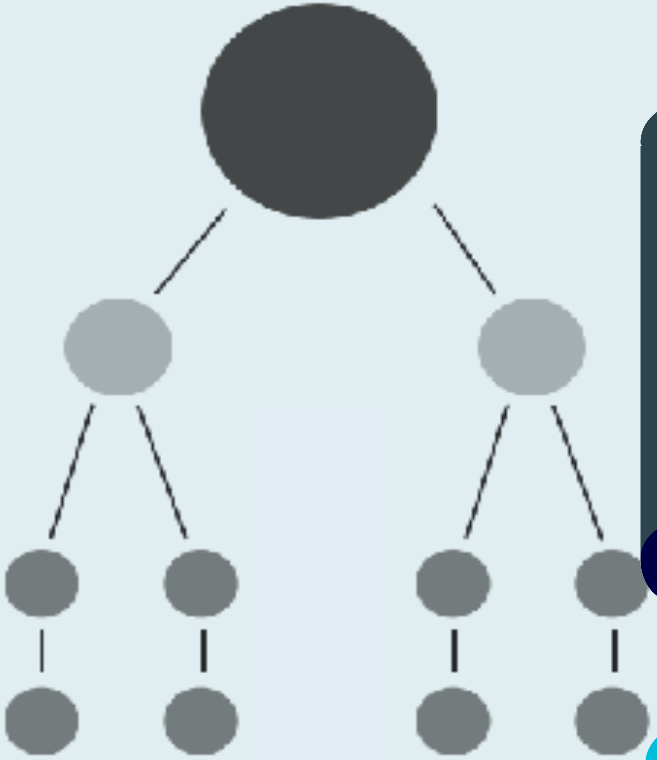
Enfatizam-se os passos finais da solução, após a solução de **problemas menores**. Mas a solução de problemas pequenos (“**problemas infantis**”) tem que ser mostrada.

Analogia com a “Indução finita”

Indução Finita: prova-se resultados matemáticos gerais supondo-os válidos para valores **inferiores a n** e demonstrando que o resultado vale também para **n**. Além disso, mostra-se que o resultado é correto para **casos particulares**.

Equivalente procedural de Recorrências

Recorrências são formulações de funções para **n**, usando resultados da mesma função para valores **inferiores a n**. Além disso uma recorrência deve exibir resultados específicos para **determinados valores**.



Visões

Uso de Recursão

Algoritmos de Divisão e Conquista, em geral, são implementados com recursão, que é um mecanismo onde um procedimento pode "**chamar a sí mesmo**". Esse tipo de procedimento é aceito pelas linguagens de programação e tem esquema de geração de código bastante eficiente.

Escrita do algoritmo criado com Divisão e Conquista

O algoritmo pode ser expresso com uma estrutura padrão, contendo as alternativas para cada problema "infantil" e também para a decomposição do problema "grande".

Alternativamente, quando nada precisa ser feito para resolver o problema "infantil" o código pode assumir a seguinte estrutura:

Escrita do algoritmo com Divisão e Conquista

Alternativas para solução de cada subproblema “infantil” e decomposição do problema “grande”.

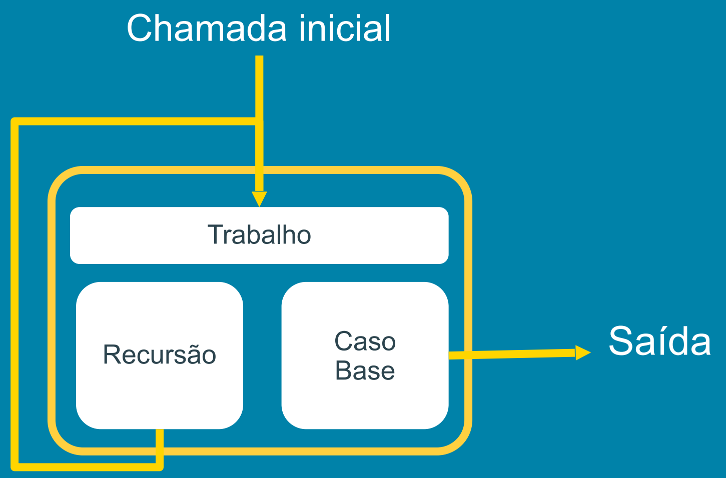
```
AlgoritmoDC(n):
   se (teste de problema “infantil” 1):
      #código com a solução do problema infantil 1
      ...
   senão se (teste do problema “infantil” k):
      #código com a solução do problema infantil n
   senão:
      #código com chamadas de subproblemas e composição
      da solução a partir dessas chamadas
```

Quando nada precisa ser feito para solução do problema infantil.

```
AlgoritmoDC(n):
    se (teste de não ser o problema “infantil”):
       #código com chamadas de subproblemas e
       composição da solução a partir dessas
       chamadas
```

Dinâmica da execução de um procedimento recursivo

Sempre que uma chamada recursiva é executada, o sistema operacional empilha as variáveis locais e a instrução em execução, desempilhando esses elementos no retorno da chamada.

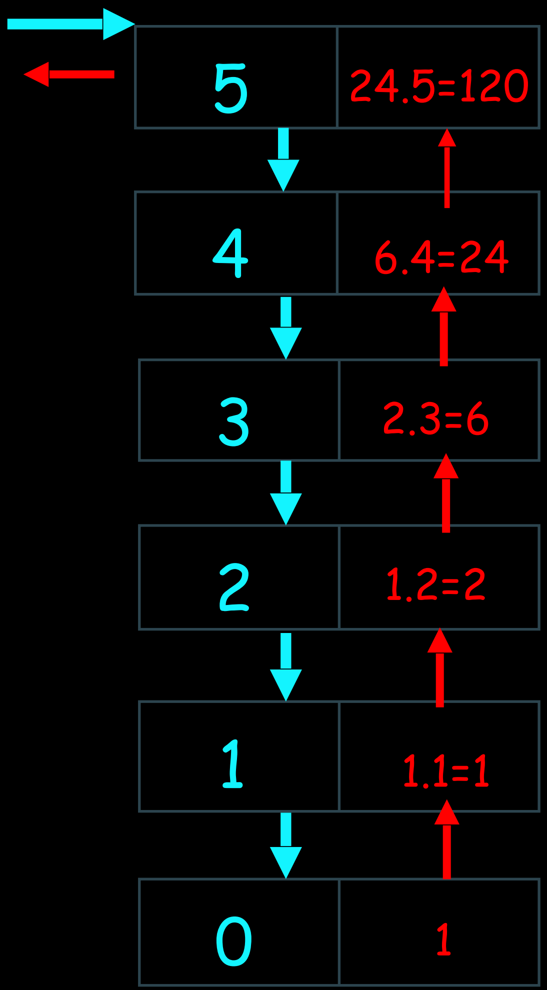


Chamadas recursivas podem ser expressas através de uma árvore de recursão.

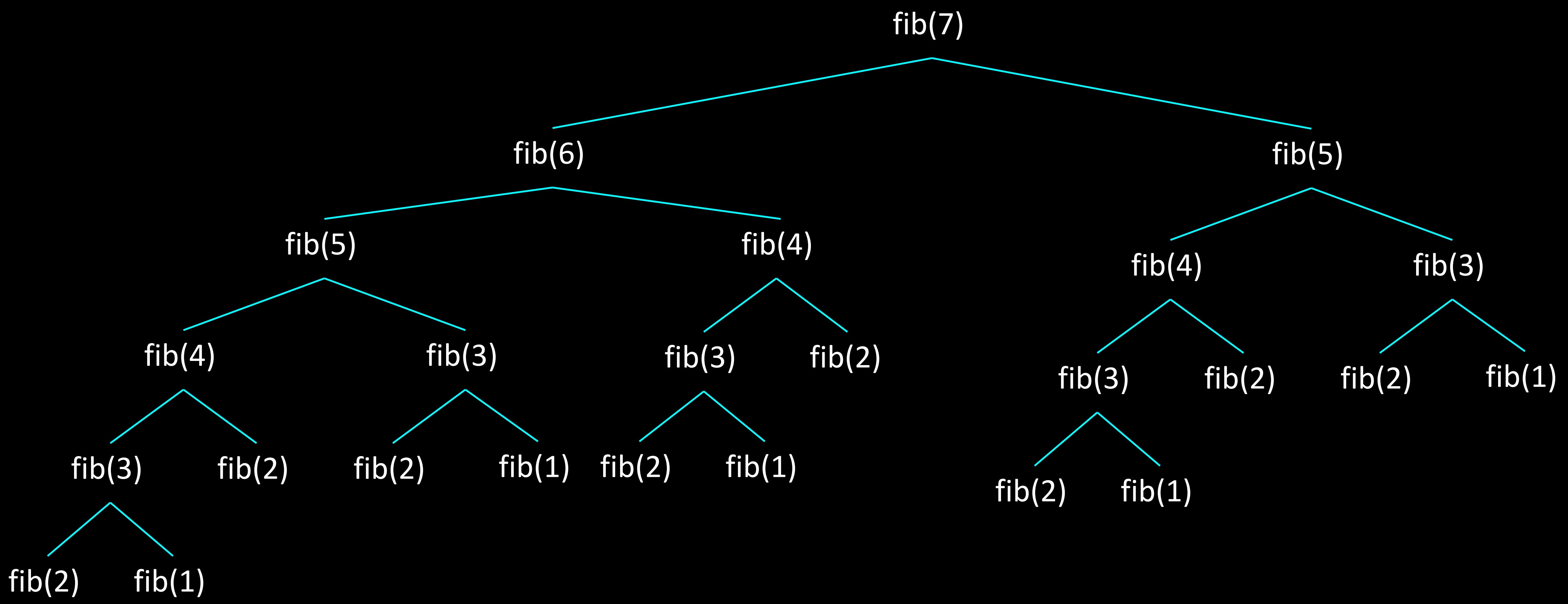
Árvore de recursão para Fatorial

X ← Fatorial(5)

```
inteiro Fatorial(p);
   se p = 0:
      retornar 1
   senão:
      retornar
      p.Fatorial(p-1)
```



Árvore de recursão para Fibonacci

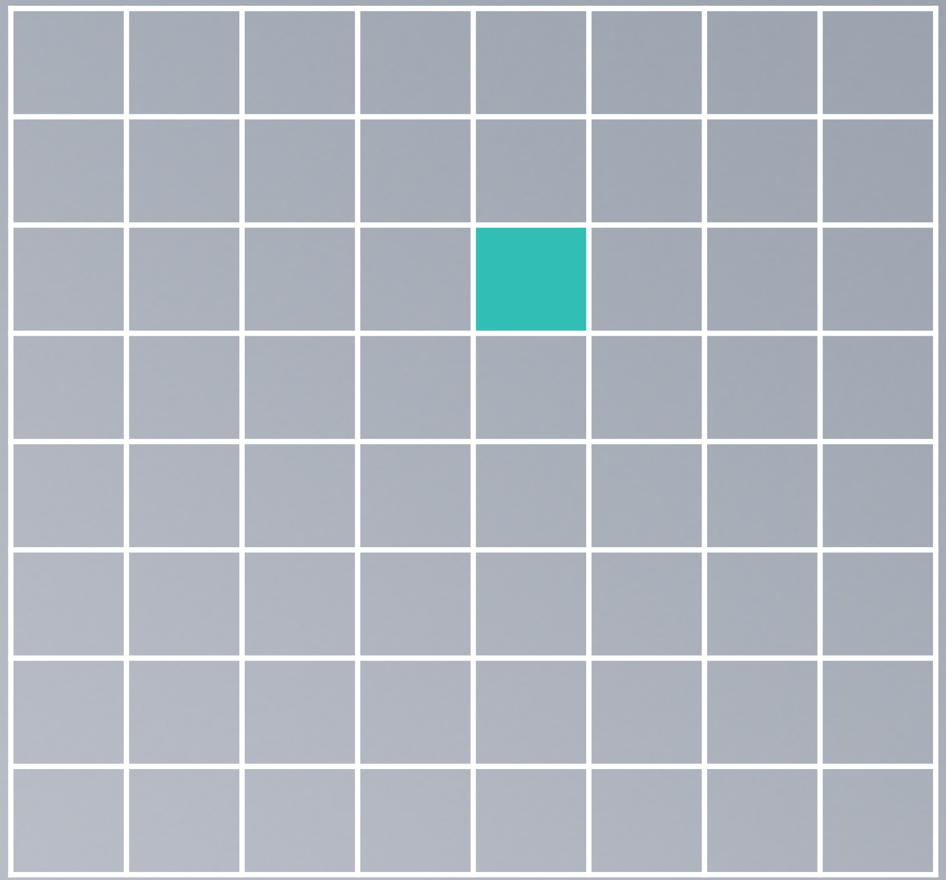


Divisão e Conquista

Um exemplo geométrico: Problema dos Trominós

Como azulejar uma parede n x n, onde n é da forma $n = 2^k$ ou seja, n é potência de 2, de tal forma que sobre um quadradinho 1 x 1, onde será colocada uma placa?

Trominó:
azulejo 2 x 2 do qual foi cortado um canto.



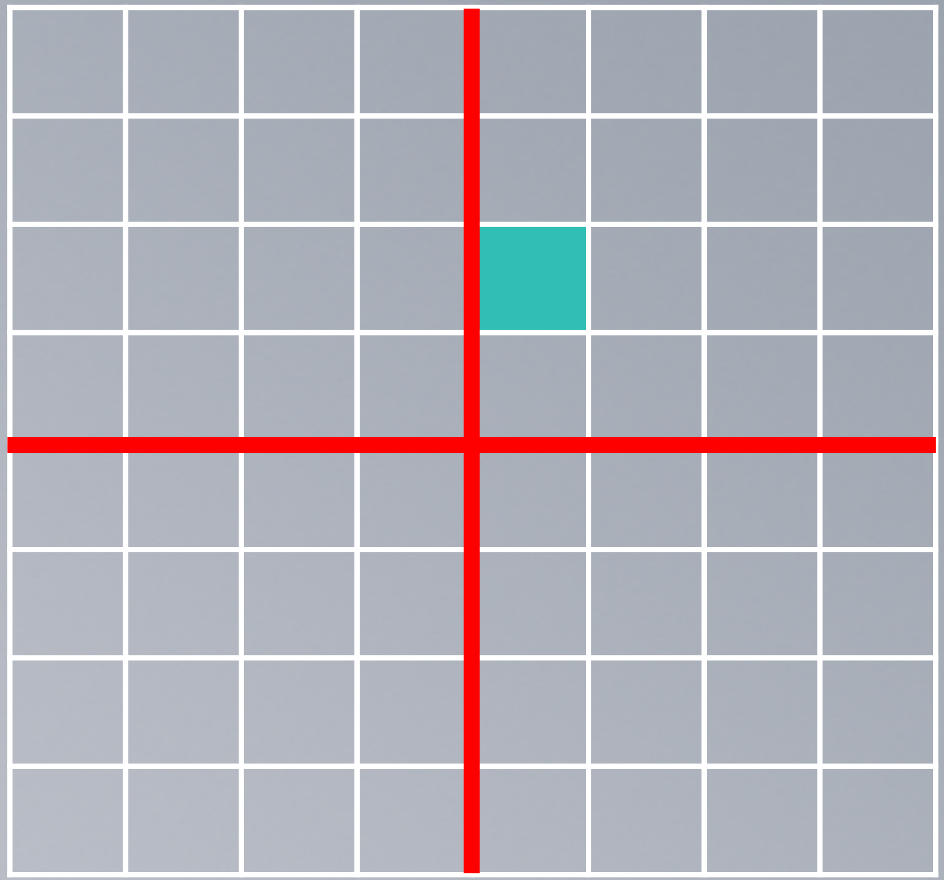
Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Idéia de divisão e conquista:

se k = 0 (lado = 1), nada é feito

senão, dividir a parede em 4 quadrantes



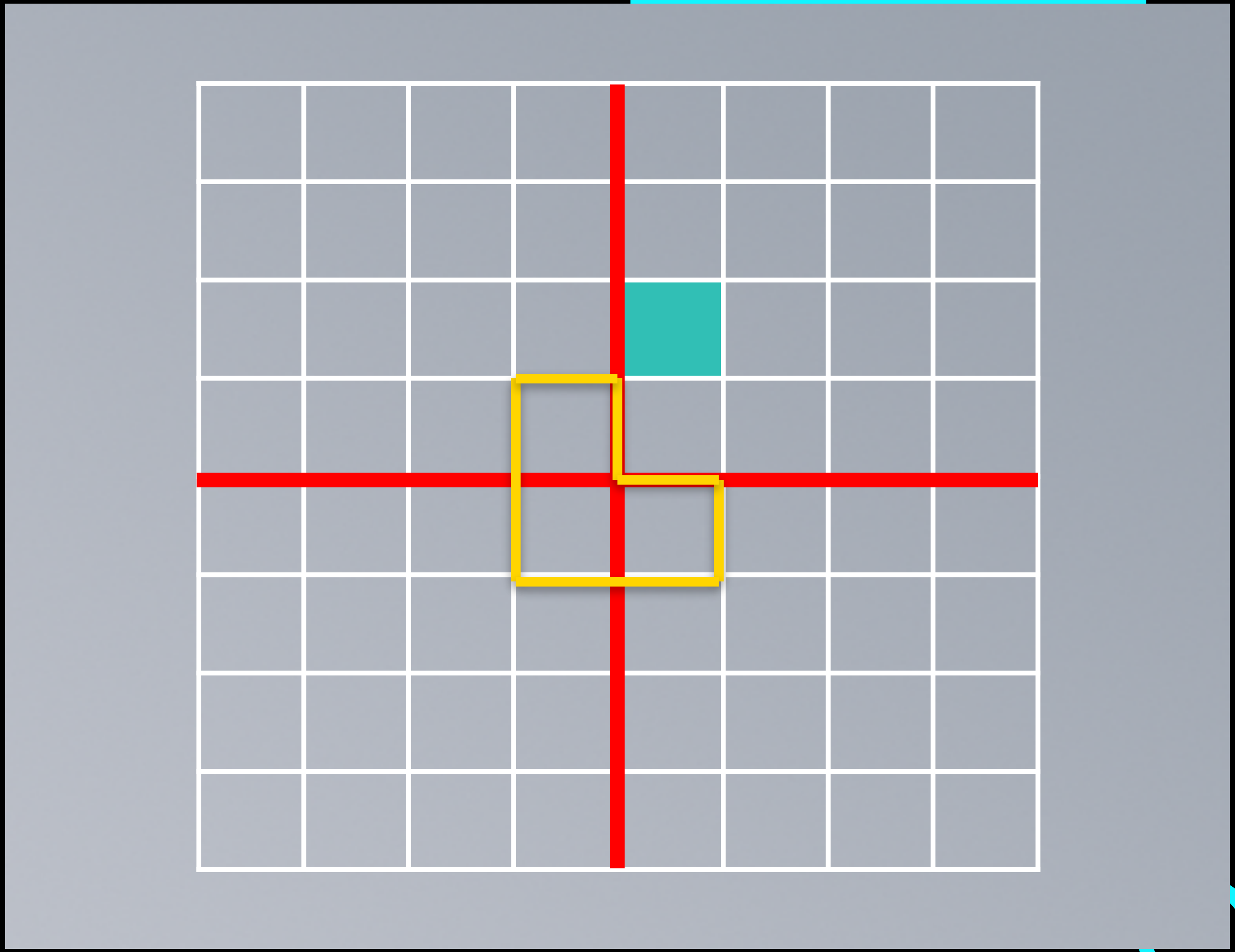
Divisão e Conquista

Abordagem do Problema Trominós para n = 2k

Idéia de divisão e conquista:

se k = 0 (lado = 1), nada é feito

senão, dividir a parede em 4 quadrantes, colocar um trominó nos quadrantes opostos ao "buraco" e resolver os 4 sub-problemas.



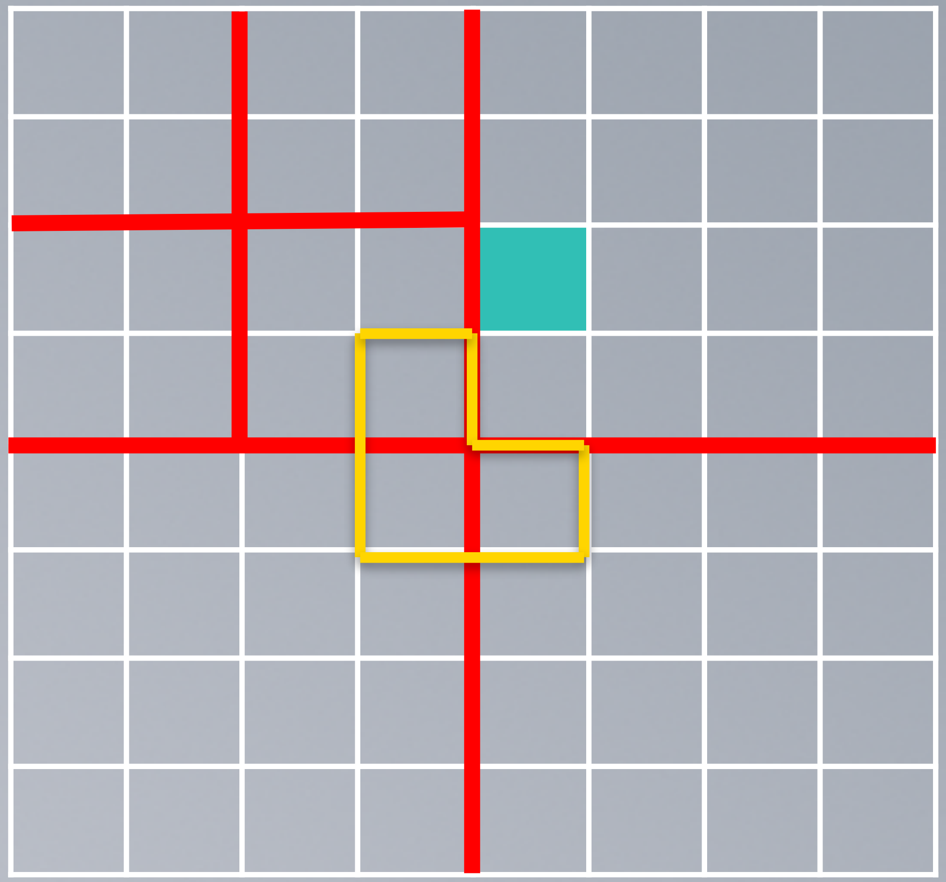
Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Idéia de divisão e conquista:

A seguir resolve o problema para cada um dos quadrantes.

Inicialmente, para o quadrante I.



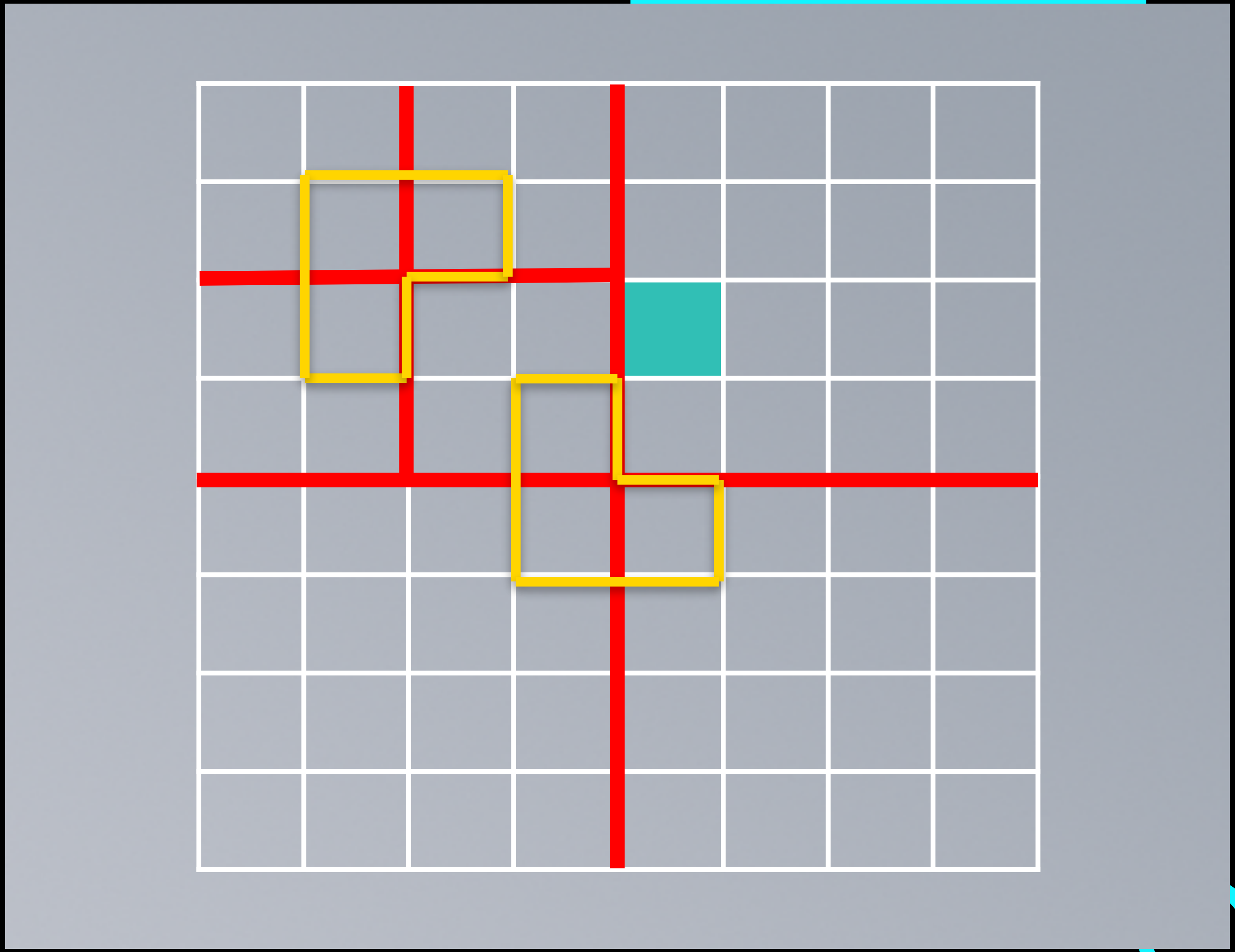
Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Idéia de divisão e conquista:

A seguir resolve o problema para cada um dos quadrantes.

Inicialmente, para o quadrante I.



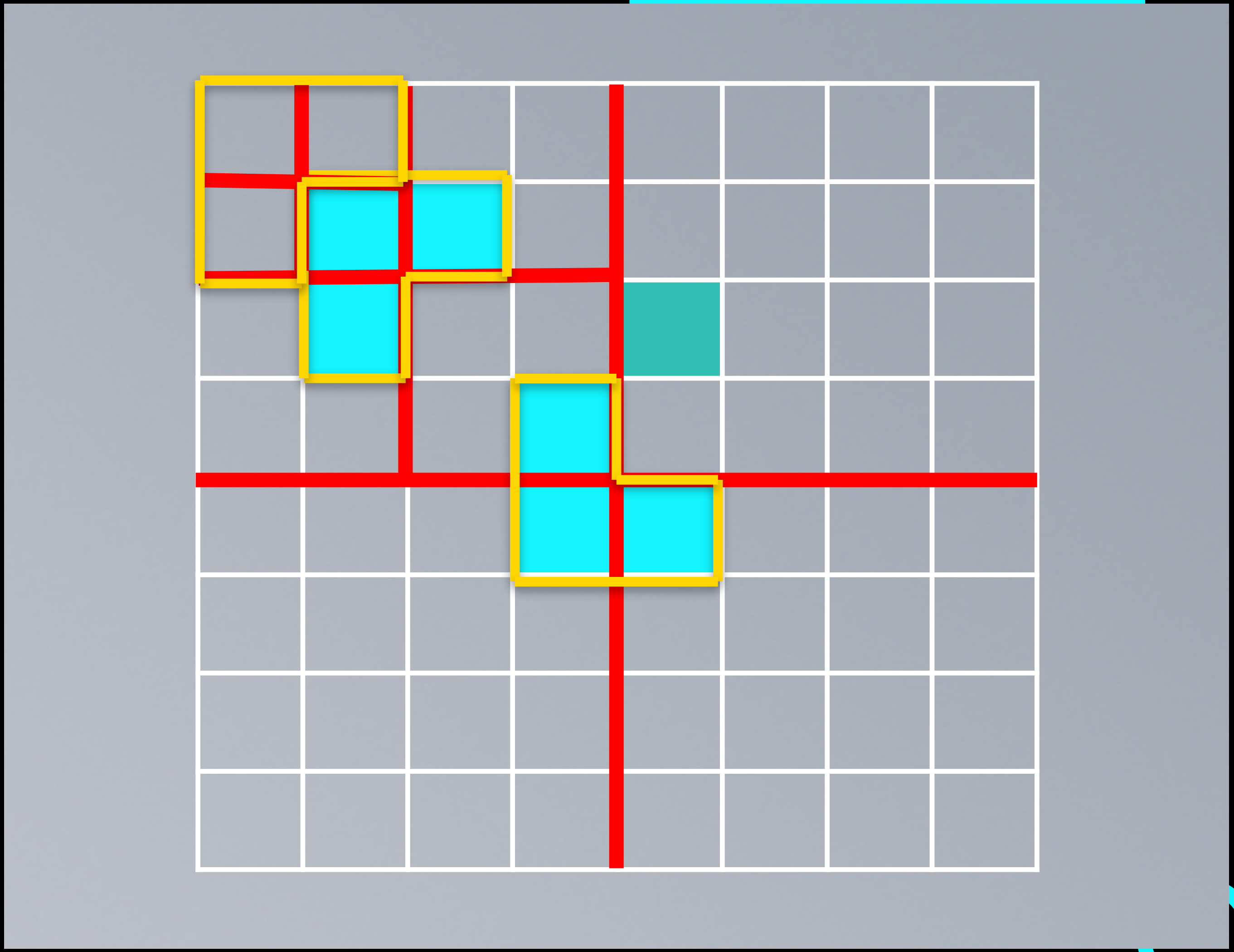
Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Idéia de divisão e conquista:

A seguir resolve o problema para cada um dos quadrantes.

Inicialmente, para o quadrante I.

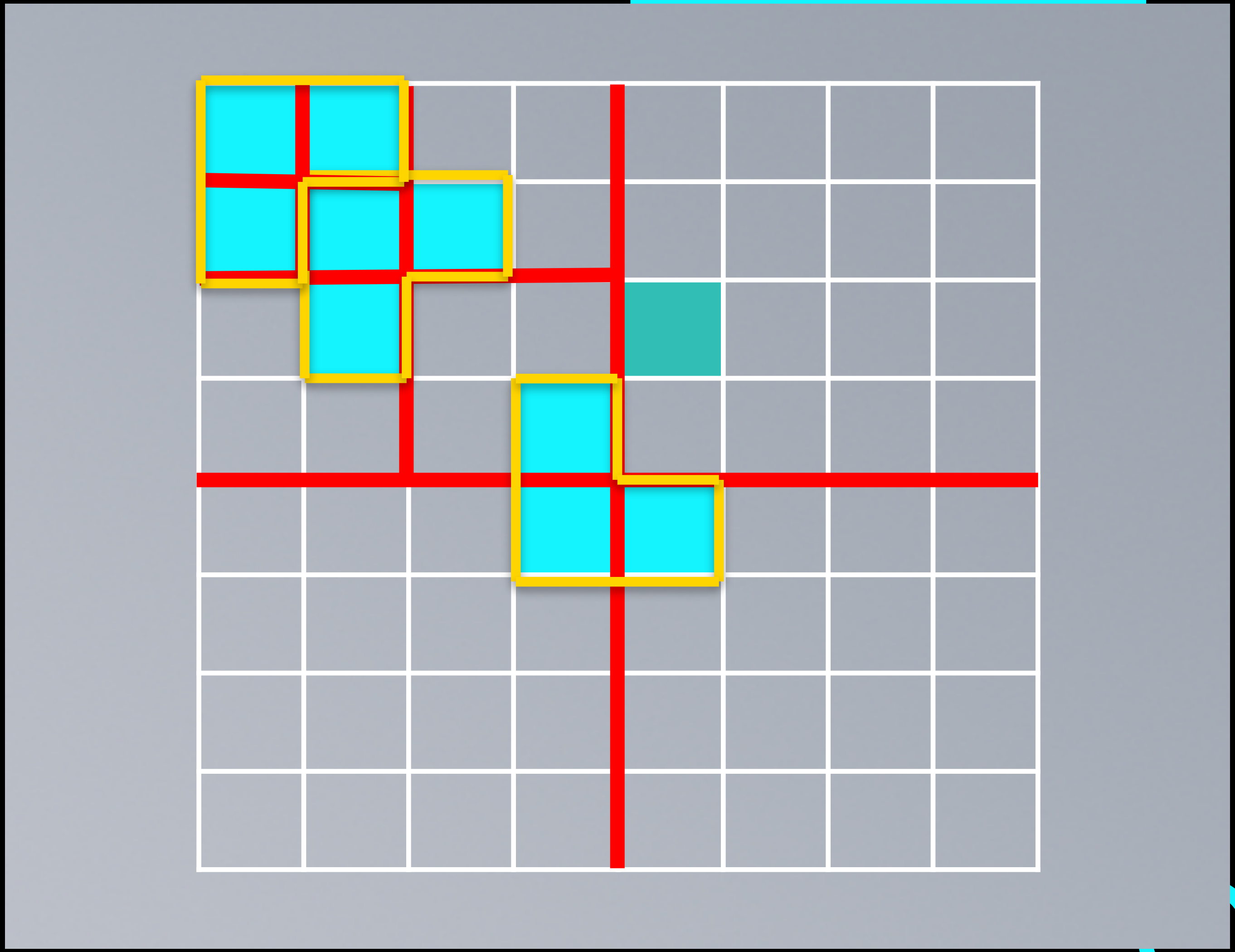


Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Idéia de divisão e conquista:

Quando o lado = 2, coloca-se o último trominó do quadrante e resolvem-se quatro problemas infantis, para lado = 1, quando nada é feito.

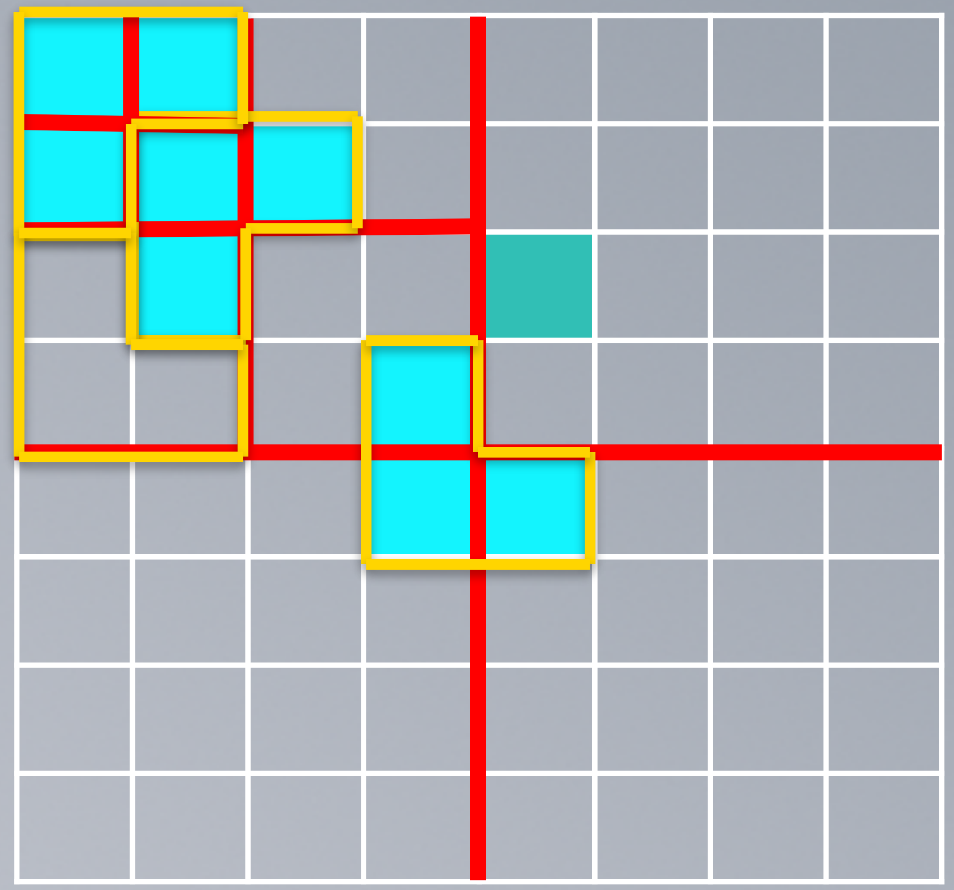


Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Idéia de divisão e conquista:

Segue resolvendo o problema…

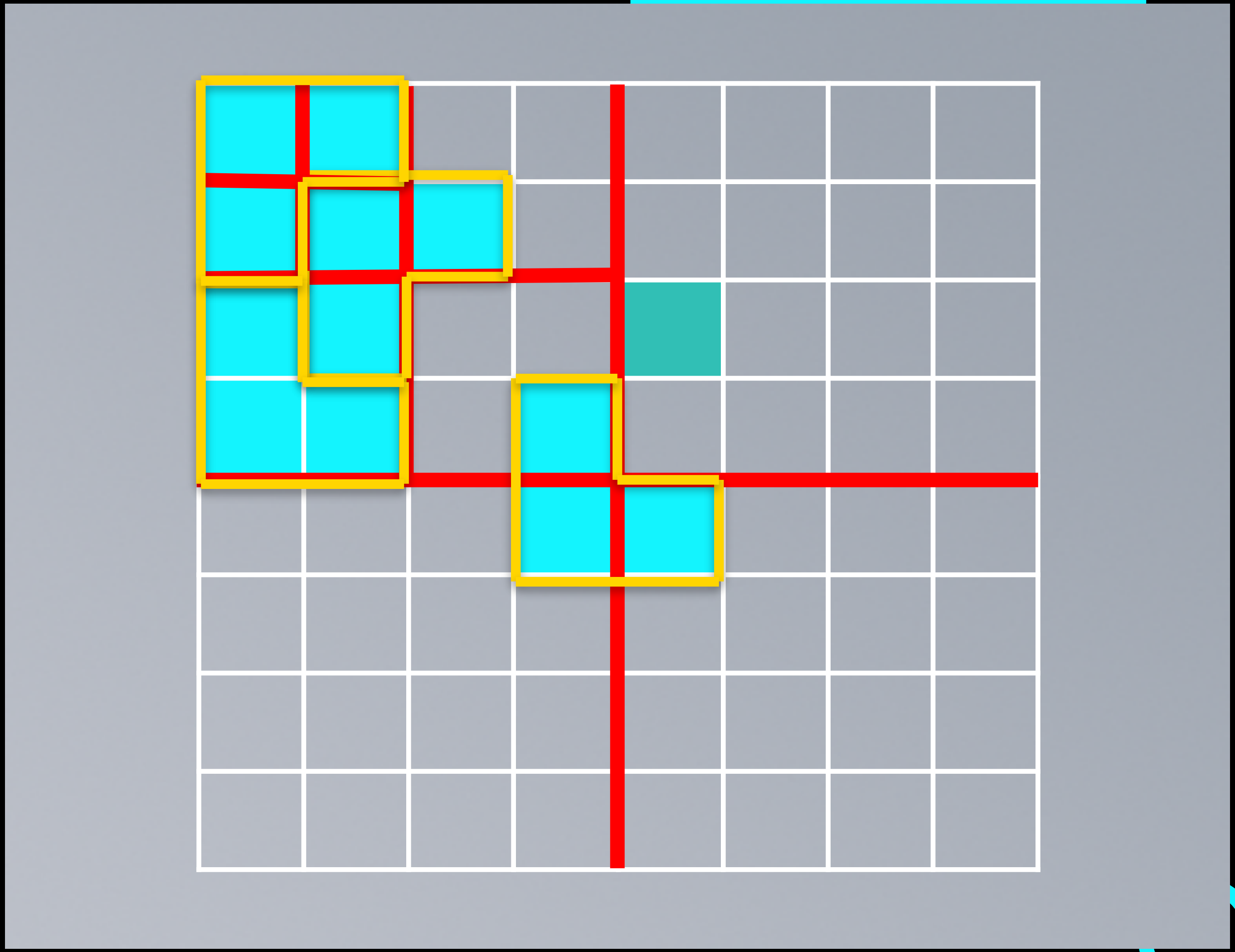


Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Idéia de divisão e conquista:

Segue resolvendo o problema...

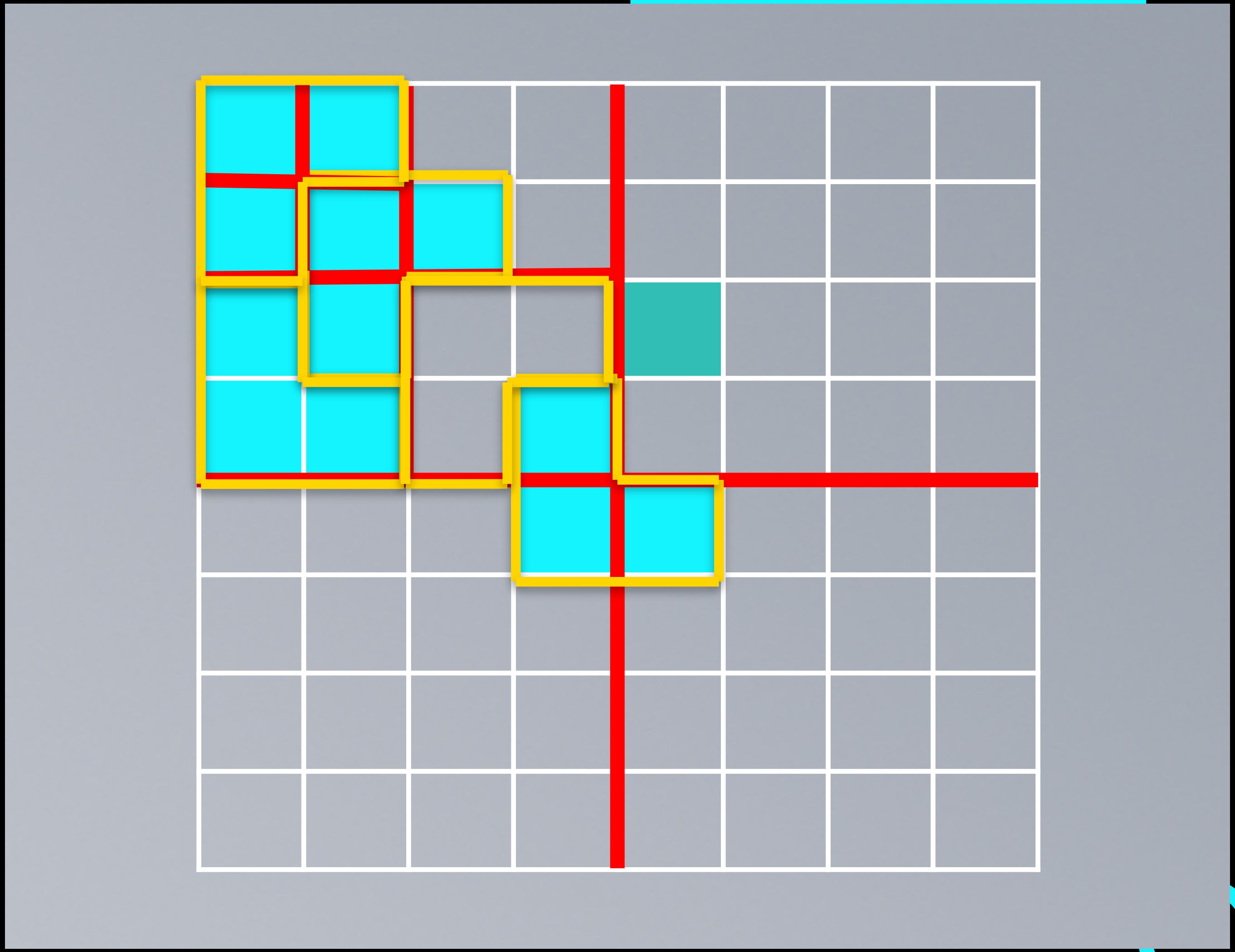


Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Idéia de divisão e conquista:

Segue resolvendo o problema...

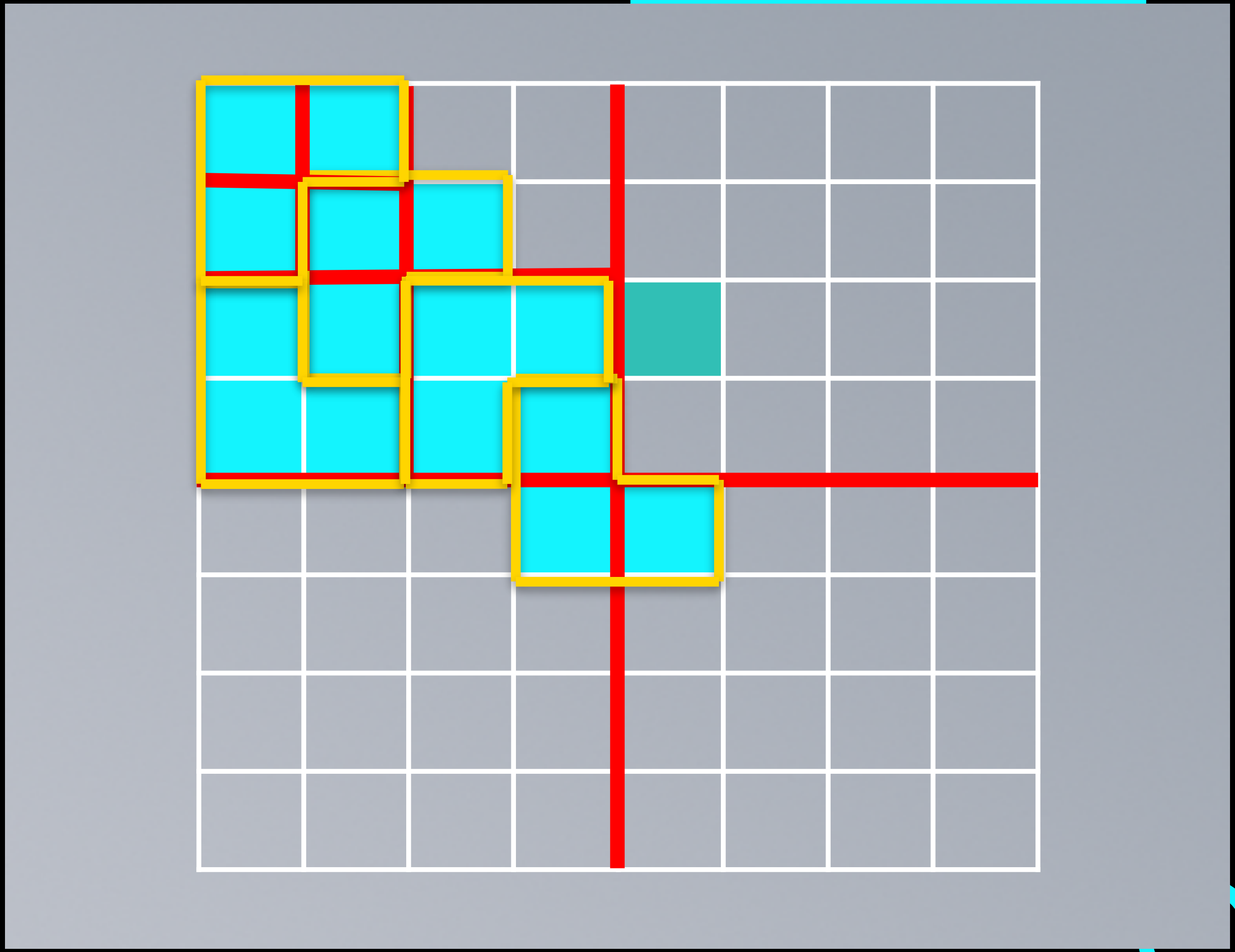


Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Idéia de divisão e conquista:

Segue resolvendo o problema...

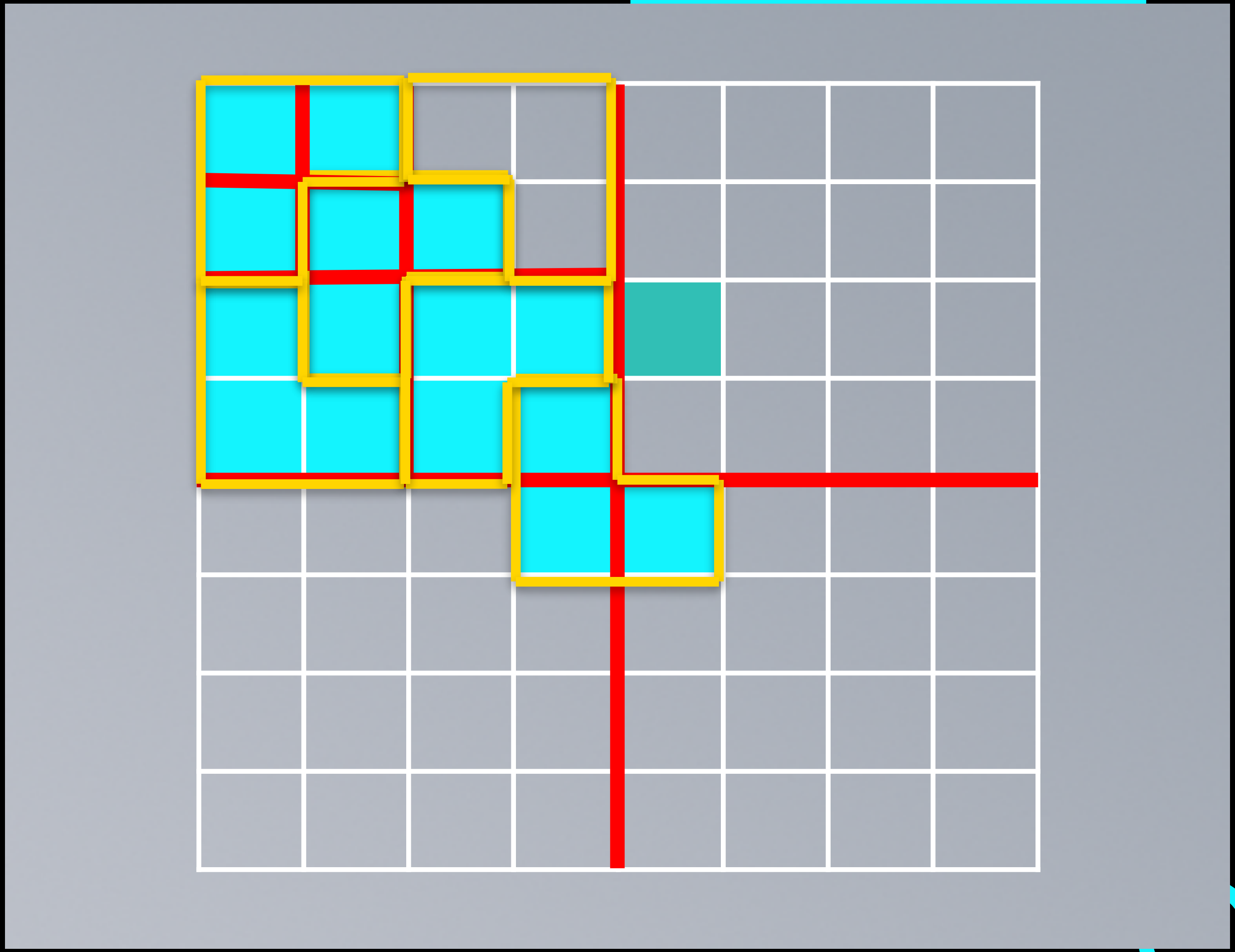


Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Idéia de divisão e conquista:

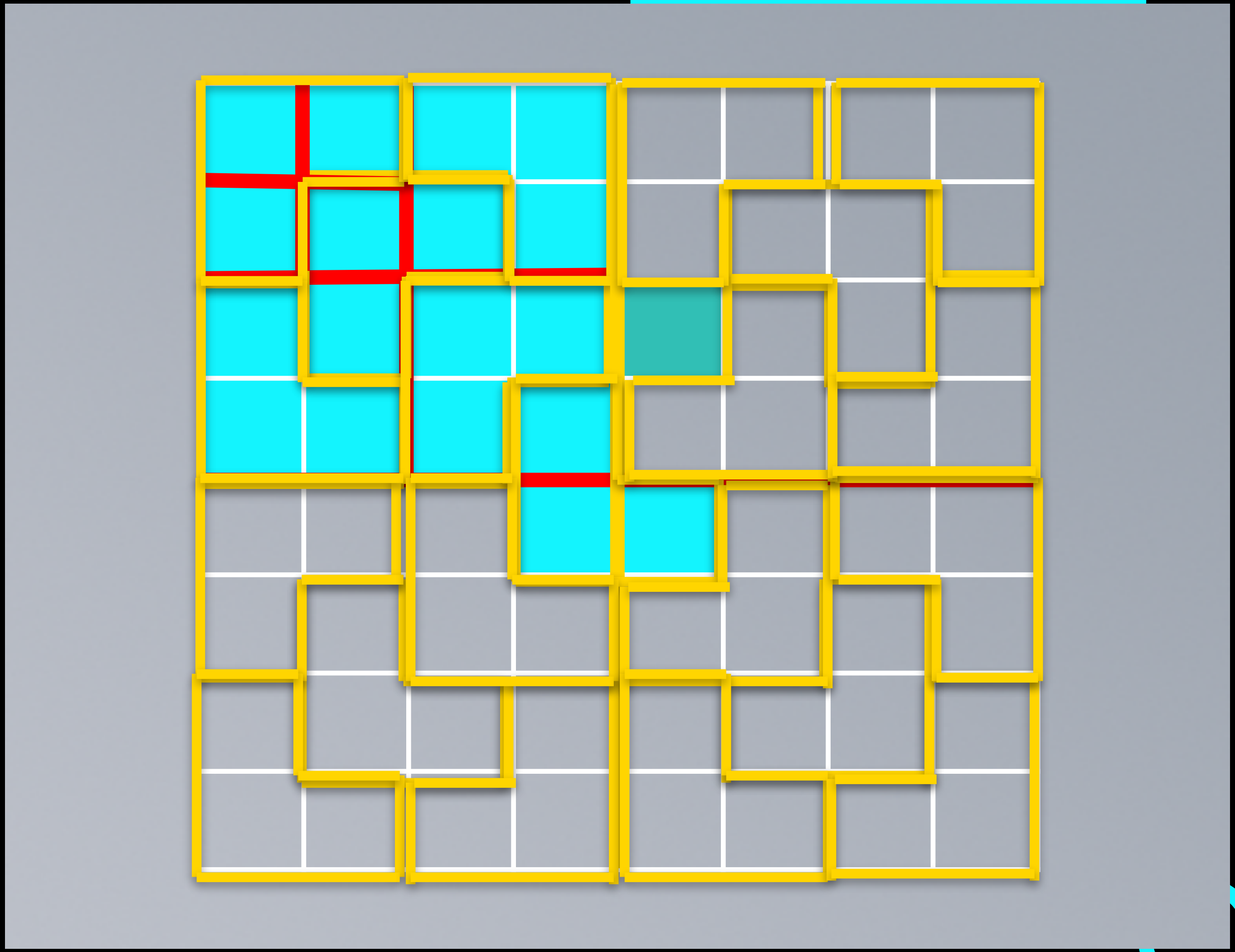
Segue resolvendo o problema...



Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Solução final

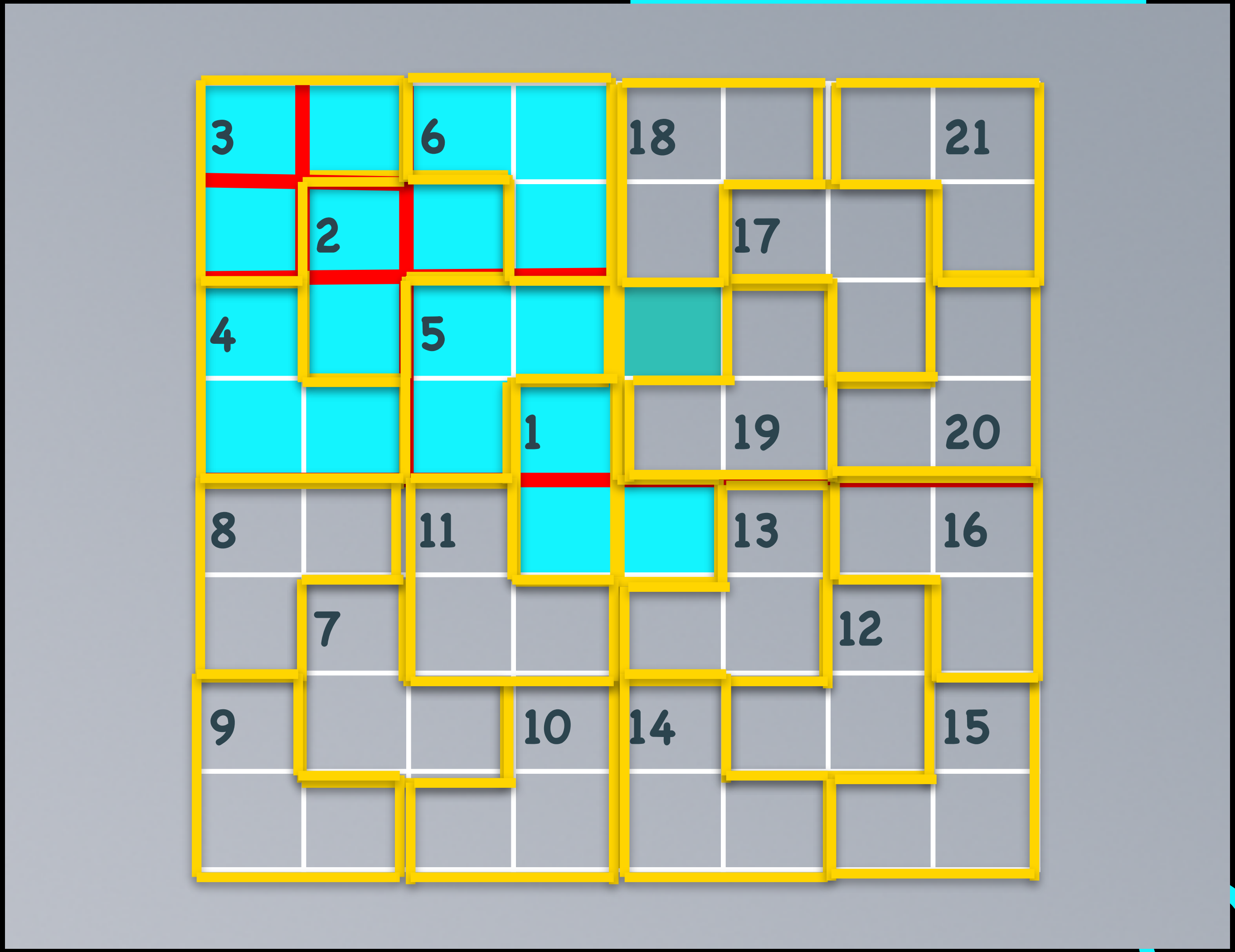


Divisão e Conquista

Abordagem do Problema
Trominós para n = 2k

Solução final

Ordem de preenchimento

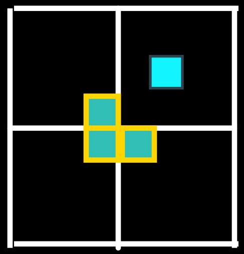


Abordagem do Problema Trominós para n = 2k

```
Azulejo(r, c, l, rb, cb);
      se  l = 1:
           nada é feito   #neste caso o quadrado é o próprio buraco
      senão:
           Divide o quadrado em quatro quadrantes
           Coloca 1 trominó na área central, de forma que fique 1 pedaço do
           trominó em cada quadrante onde não se encontra o burac
           Resolve o problema Azulejo para cada 1 dos 4 quadrantes, pois
           cada 1 deles é do mesmo tipo do original (tem 1 "buraco",
           sendo em 1 o buraco original; e 1 novo buraco nos outros 3,
           já preenchido devido à colocação do trominó atual).
```

Obs: r e c são as coordenadas do topo superior direito do quadrado; l é o tamanho do lado; rb e cb são as coordenadas do buraco.

A chamada externa é: Azulejo(1,1,n,rb,cb);



Abordagem do Problema Trominós para n = 2k

```
Azulejo(r, c, l, rb, cb);
    se (l > 1):
        nt ← nt+1;  l ← l/2;
        se (rb < (r+l)) e (cb < (c+l)):
             rnb ← rb;  cnb ← cb;
        senão:
             rnb ← r+l-1; cnb ← c+l-1;  P[rnb,cnb] ← nt;
             Azulejo(r,c,l,rnb,cnb)
        se (rb ≥ (r+l)) e (cb < (c+l)):
             rnb ← rb;  cnb ← cb;
        senão:
             rnb ← r+l; cnb ← c+l-1;  P[rnb,cnb] ← nt;
             Azulejo(r+l,c,l,rnb,cnb);
        se (rb ≥ (r+l)) e (cb ≥ (c+l)):
             rnb ← rb;  cnb ← cb;
        senão:
             rnb ← r+l; cnb ← c+l;  P[rnb,cnb] ← nt;
             Azulejo(r+l,c+l,l,rnb,cnb);
        se (rb < (r+l)) e (cb ≥  (c+l)):
             rnb ← rb;  cnb ← cb;
        senão:
             rnb ← r+l-1; cnb ← c+l;  P[rnb,cnb] ← nt;
             Azulejo(r,c+l,l,rnb,cnb);
nt ← 0;   P[rb,cb] ← 0;  Azulejo(1,1,n,rb,cb);
```

Dúvidas?

lucila.bento [at] ime.uerj.br