Surge agora uma outra questão. Viemos buscando algoritmos para resolver problemas. No entanto, será que sempre seria possível achar esses algoritmos? Colocando de outra forma: será que, para todo problema, sempre vai existir um algoritmo que consiga resolvê-lo?

Problemas insolúveis

Algoritmos são programas que rodam em computadores digitais. Então, num primeiro impulso, dada a voz corrente de que computadores são máquinas extremamente poderosas, capazes de realizar qualquer tarefa concebível, poderíamos responder que "sim", ou seja, que para todo problema semprem há de existir um algoritmo que o resolva a contento.

No entanto, surpresa! Não poderíamos estar mais enganados! De fato, existe um vastíssimo elenco de problemas para os quais simplesmente não há algoritmo legítimo que o resolva. Observe que exigimos um algoritmo legítimo, que satisfaça as quatro propriedades básicas já enunciadas.

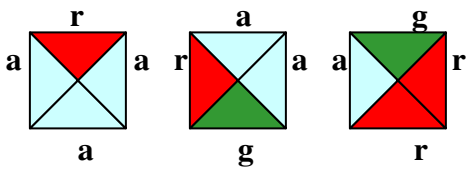
Na verdade, existem problemas de cunho prático, e importantes, para os quais não há solução algorítmica possível. Será que com mais tecnologia, e.g., com computadores mais rápidos e com maior capacidade de memória, e dado tempo suficiente para rodar o programa, poderemos resolver esses problemas, no futuro? A resposta é "não", novamente. Não é uma questão de capacidade de processamento, ou capacidade de memória, que impedem a solução algoritmica para esses problemas insolúveis. O impedimento é intrínseco aos algoritmos. Simplesmente não é possível descrever receitas adequadas para esses problemas.

Computador digital nenhum vai resolver esses problemas, nem hoje, nem amanhã, nem nunca; nem aqui, nem em Marte, nem lugar algum. Nem que o computador seja capaz de rodar programas por quanto tempo quisermos (anos, séculos, ...). Esse é um dos grandes e surpreendentes resultados em Teria da Computação obtidos no século passado, há não mais de 80 anos.

Um exemplo simples e concreto

Vamos exibir um exemplo simples de um problema insolúvel.

Os dados de entrada são um conjunto finito, T, de *tipos de* ladrilhos. Por exemplo, na figura abaixo, o conjunto T seria formado por ladrilhos de três tipos.



As letras próximas das bordas de cada ladrilho indicam as cores daquela borda, caso você esteja lendo uma impressão deste texto que não comporte cores. A seguinte legenda poderia ser usada

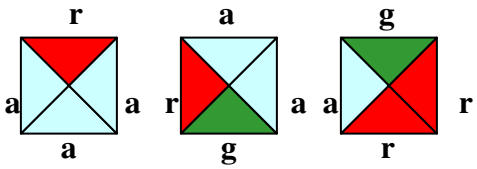
Símbolo	Cor
r	vermelha
g	verde
a	azul

O problema é determinar se podemos ladrilhar *qualquer grade n*x*n*, com ladrilhos de tipos indicados no conjunto T. Uma grade n x n é uma região quadrada do plano subdividida em n x n celas iguais e uniformes.

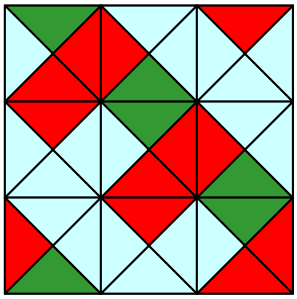
As seguintes condições devem ser respeitadas:

1. Os ladrilhos não podem ser girados, i.e., rotacionados.
2. Podemos usar tantos ladrilhos quantos forem necessários, desde que idênticos a um daqueles contidos no conjunto de tipos T. Ou seja, temos um estoque potencialmente ilimitado de cada tipo de ladrilho.
3. As cores nas faces de ladrilhos que se tocam, ao serem posicionados na grade, devem ser idênticas.
4. Na área quadrada demarcada, não podemos deixar regiões sem ladrilhar.

Exemplo 1: Assuma que o conjunto T é formado pelos ladrilhos

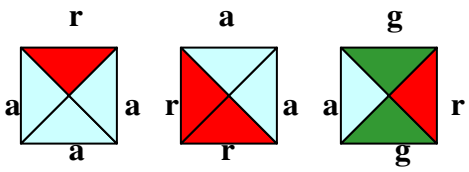


Uma região 3 por 3 do plano poderia ser ladrilhada com este conjunto de ladrilhos, como pode ser visto abaixo:



Será que *toda região quadrada* pode ser ladrilhada com ladrilhos do tipo indicado? A resposta é positiva, embora não demonstremos esse fato aqui.

Exemplo 2: Assuma agora um outro conjunto com três tipos de ladrilhos, como abaixo:



É fácil verificar que uma região 3 por 3 não pode ser ladrilhada com esse conjunto de ladrilhos e obedecendo-se as condições impostas.

Portanto, um algoritmo para resolver o problema deveria emitir uma resposta positiva tendo como entrada a instância do primeiro exemplo e deveria produzir uma resposta negativa quando executado tendo como entrada a instância do segundo exemplo. Uma *instância* para o primeiro caso, por exemplo, poderia ser dada pelo número n (o tamanho do lado da grade) e por uma descrição dos tipos de ladrilhos que podem ser usados. Basicamente, qualquer descrição desses dados serviria, desde que fosse formada por uma sequência finita de símbolos.

O resultado surpreendente é que *não existe* um algoritmo capaz de resolver esse problema de forma geral. Ou seja, o algoritmo deveria dar a resposta correta sempre, não importa qual o conjunto de tipos de ladrilhos e o tamanho da grade que sejam dados como entrada. Podemos *provar* esse fato, *matematicamente*, embora não o façamos aqui.

A inexistência de algoritmos para esse problema, simples e fácil de entender, implica em que NENHUM computador *digital* JAMAIS vai conseguir resolver esse problema, nem agora, nem nunca, nem com QUALQUER melhoria de tecnologia, nem com QUALQUER tamanho de memória, nem que rode POR QUANTO TEMPO QUISERMOS. Isto porque, se algum computador o fizesse, teríamos que tê-lo alimentado com um algoritmo apropriado, o qual já afirmamos que não existe.

O problema, inocente à primeira vista, é simplesmente muito complicado para que um algoritmo legítimo possa resolver *qualquer instância* do problema.

Uma pequena variante do exemplo

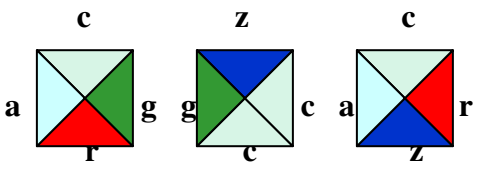
Como uma variante do problema anterior, considere agora o caso onde não estamos interessados na possibilidade de se ladrilhar qualquer região quadrada do plano.

Agora, damos como entrada, além do conjunto de tipos de ladrilhos, uma posição inicial, I, e uma posição final, F, localizadas na grade do plano. Tanto I quanto F são posições marcadas na grade do plano todo, que é infinito nas quatro direções. O problema é

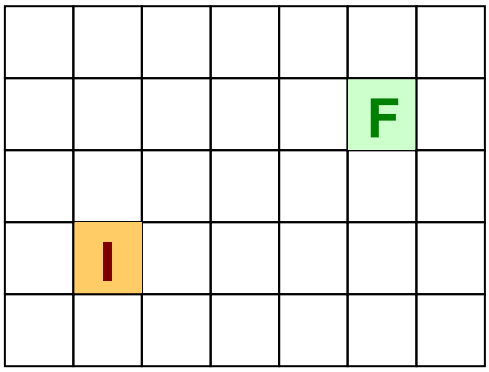
determinar se podemos ladrilhar um caminho partindo de I e chegando até F, podendo o caminho passear por toda a extensão do plano.

É claro que valem as mesmas condições do problema anterior. Em particular, a faces de ladrilhos que se tocam ao longo do caminho devem ser da mesma cor.

Como um exemplo, escolhemos, de novo, um conjunto de ladrilhos de três tipos, como abaixo:

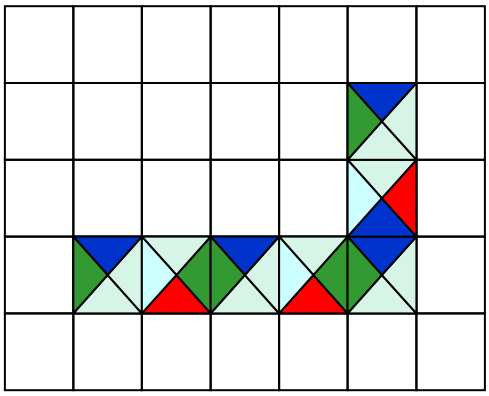


e, como marcações de início e fim, escolhemos como indicado abaixo:



A questão, então, é se podemos sair ladrilhando da posição indicada por I até alcançar a posição indicada por F.

Nesse caso, e com esses ladrilhos, a resposta é positiva:



Então, será que existe uma receita infalível que, recebendo como entrada o conjunto de tipos de ladrilhos e as posições inicial e final, determina se existe o caminho ladrilhado? Mais, essa receita deve ser um algoritmo legítimo.

Note que essa é uma pequena variante do problema anterior, para o qual já sabemos que não existe um algoritmo apropriado. Pois para o caso presente, *existe* um algoritmo que decide a questão, quaisquer que sejam o conjunto de ladrilhos e as posições marcadas! Podemos exibir o algoritmo e provar matematicamente sua correção parcial e terminação. Nessa variante, portanto, o problema se torna decidível algoritimicamente.

Outra pequena variante

O problema anterior permitia que o caminho vagasse por todo o plano, em princípio. Vamos considerar o mesmo problema de determinar se existe um caminho de I até F que podemos ladrilhar. Mas, agora, vamos impor uma condição extra bem simples: não permitimos que o caminho cruze uma dada linha horizontal que divide o plano em dois semi-planos.

É claro que se I e F estiverem em semi-planos distintos, então o caminho não existirá. Mas e quando estão no mesmo semi-plano: será que existe um algoritmo que sempre determina se o caminho existe, dadas as posições de I e de F?

A resposta, surpreendentemente, é *negativa*. Isto é, *não existe um algoritmo* que resolva essa questão, determinando se existe ou não o caminho.

Pelo simples fato de permitirmos, ou não, que um caminho cruze uma linha horizontal fixa, o problema passa de *decidível* para *não decidível*! No primeiro caso, existe um algoritmo para resolvê-lo; no segundo caso, não. No primeiro caso, podemos programar um computador para nos dar a resposta, para qualquer instância de entrada. No segundo caso, não há programa de computador capaz de sempre nos dar a resposta correta! E podemos demonstrar esses fatos matematicamente.

Um exemplo envolvendo programas

Já sabemos que podemos codificar receitas em linguagens de programação, produzindo o texto fonte do programa. Fixe uma linguagem de programação de cunho geral, P. Ou seja, P deve ser ampla o suficiente tal qual uma linguagem de programação moderna, como C ou Pascal. Com isso queremos dizer que não vale tomar P como uma linguagem trivial. E um programa escrito em P só será considerado um algoritmo legítimo se, para todas as suas entradas válidas, o programa sempre pára.

Isso nos leva à seguinte consideração: será que não podemos escrever um algoritmo, L, que, tendo como entrada o código de outros programas escritos na linguagem P, o programa L determina se o programa dado sempre vai parar quando alimentado com dados de entrada válidos? Ou seja, gostaríamos de um algoritmo L capaz de detectar *ausência de loops* em códigos fontes de outros programas escritos na linguagem P. Esse algoritmo L seria muito útil, pois evitaria que tivéssemos que demonstrar correção total dos algoritmos que codificássemos na linguagem de programação P. Ou seja, poderíamos automatizar as provas de terminação para programas escritos na linguagem P.

Pois bem. Podemos demonstrar que um tal algoritmo L *não existe*. Ou seja, *é impossível* escrevermos um algoritmo que detecte ausência de loops em programas fonte escritos na linguagem P. A dificuldade não está na linguagem P, especificamente. Está na natureza do problema que queremos resolver, qual seja, a detecção de loops. Esse problema é simplesmente muito complicado para ser resolvido em um computador digital,

algoritmicamente. Então, **NENHUM** computador **JAMAIS** vai conseguir testar se, dado um programa **PROG** (escrito em **C**, por exemplo), e dada a especificação das entradas válidas para **PROG**, EXISTE OU NÃO alguma entrada válida que force **PROG** a *não parar* (i.e., que force **PROG a** "entrar em loop", como se diz) quando ele executa com aquela entrada.