Transposição e contrato didático no ensino de algoritmos

André Gustavo Schaeffer
Universidade Federal da Fronteira Sul - Campus Erechim
andre_schaeffer@uffs.edu.br

Resumo: *o presente artigo aborda alguns aspectos teóricos do ensino de ciências e de matemática que podem contribuir para a atuação do professor como mediador no ensino de lógica da programação e algoritmos. Traz à discussão a necessidade do levantamento de preconcepções em relação ao ensino de estruturas de programação e sugere formas de fazê-lo. Também discute o gerenciamento do contrato didático, seus paradoxos e perturbações, e de que forma isso está presente no ensino da programação, posicionando sugestivamente o professor frente a esses desafios.*
Palavras-chave: *transposição didática; contrato didático; estratégias pedagógicas; ensino de algoritmos.*

Abstract: *the present paper discusses some theoretical aspects in the teaching of sciences and mathematics that may contribute to the teacher's role as a mediator in the teaching of logic programming and algorithms. Brings out the necessity of raising misconceptions regarding computer programming structures suggesting ways of making it. Also discusses the management of didactic contract, its paradoxes and perturbations, and the way all of this is present in the teaching of computers programming, positioning the teacher to address these challenges.*
Keywords: *didactic transposition; didactic contract; pedagogical strategies; teaching of algorithms.*

1. Introdução

O ensino de algoritmos representa uma parte considerável do currículo de cursos de ciência da computação e de cursos afins. O amparo fornecido pelos computadores às ciências, principalmente às exatas, aumenta a necessidade de conhecer a forma através da qual as máquinas executam instruções e processam dados. Devido a isso, ganha importância o ensino de algoritmos pois todos os sistemas em suas diferentes aplicações são escritos em linguagem algorítmica e convertidos em um programa executável ou interpretável por um computador. O algoritmo, sendo genérico, pode ser convertido para qualquer linguagem de programação, ficando a cargo do programador a escolha da mesma em função das mais diversas necessidades, como em relação ao sistema operacional no qual o programa será executado, ao dispositivo ou equipamento em que será utilizado, se usará recursos de comunicação ou não, se trata-se de um sistema embarcado ou não, etc.

Há que se fazer uma breve elucidação sobre o significado de algoritmo pois dependendo do contexto, pode acarretar confusão e controvérsia. Em primeiro lugar, um algoritmo pode ser entendido como uma sequência de passos. Se ele existe, deve ter sido criado para resolver algum problema. Um cálculo, por exemplo. E se esse algoritmo se propõe a calcular algo, deve fazer isso em algum momento, ou seja, de forma imediata ou demorando certo tempo, deve terminar e exibir um resultado ou modificar o estado de algo. Porém, no contexto do ensino, um ensino algorítmico não tem o mesmo significado. Levando em conta a forma como as pessoas aprendem,

particularmente as crianças, deve-se tentar ao máximo evitar o ensino unicamente algorítmico pois há risco de se promover um aprendizado sem significado e, portanto, frágil e isolado, diminuindo as chances desse aprendizado ser conectado a outros conhecimentos futuramente. O recurso algorítmico com vistas à realização de operações aritméticas, ensinadas nos anos iniciais do ensino fundamental, foi estudado por Nogueira e Signorini (2010) e sugere que as crianças, naquela pesquisa, parecem apenas ter memorizado as regras mecanicamente, sem entender que os princípios e as propriedades do sistema de numeração decimal estão na base das técnicas operatórias. Ao mesmo tempo, sugerem as autoras que não se abandone o estudo dos algoritmos convencionais para tais operações, desde que o ensino não priorize a memorização dos procedimentos e, sim, promova conexão entre as técnicas e o conceito.

Então, há que se diferenciar *ensino de algoritmos* de *ensino algorítmico*. O primeiro diz respeito ao ensino da disciplina de algoritmos que ensina as pessoas a programar um computador, enquanto que o segundo refere-se a uma forma de ensino de conteúdos em qualquer área do saber, objeto de estudo das teorias de aprendizagem. Curiosamente, se didaticamente deve-se tentar evitar o ensino somente algorítmico, o ensino de algoritmos está repleto de situações onde conhecimentos didáticos tornam-se necessários, e podem promover uma melhor compreensão do assunto.

Este artigo pretende, portanto, analisar aspectos teóricos que possam servir como aporte didático para o ensino de algoritmos. Mais especificamente, a título de contextualização, serão usados como base algoritmos para geração de números primos, muito comuns em planos de aula de disciplinas de ensino de lógica da programação, agregando elementos da teoria da transposição didática de Yves Chevallard e da teoria dos contratos didáticos de Guy Brousseau.

2. Transposição didática e preconcepções

Com vistas a promover o entendimento do que consiste um algoritmo e de como podem ser construídos, professores de lógica da programação lançam mão de diversas estratégias. É comum tomarem por base para os primeiros contatos de estudantes com o tópico da programação de computadores, situações familiares a eles, como solicitar a realização de uma lista com a sequência de passos necessários para realizar a troca de uma lâmpada ou a troca de um pneu de carro. As diferentes soluções dos alunos são formas algorítmicas de realizar tais tarefas. Muito provavelmente são todas corretas, uma vez que, na ausência de regras é difícil considerar errada uma das soluções apresentadas. Mas por que isso é importante? O professor, naquele momento, está trabalhando com questões relativas a linguagem e abstração, presentes nas interações cotidianas de qualquer aluno. A linguagem e a abstração de que necessitamos para nos comunicar representam uma herança que carregamos até aquele momento desde que nascemos e que vão se moldando na medida em que interagimos com outras pessoas. Isto denota um caráter idiossincrático preconceitual, e, portanto, difícil de ser resolvido. O professor assim o faz porque uma linguagem de programação possui regras léxicas e sintáticas que nos remetem à nossa língua materna, que também as possui. De forma abstrata, entendemos o que nosso interlocutor quer nos dizer em um diálogo ou através de um texto escrito mesmo que ele tenha cometido erros de grafia ou erros sintáticos. Uma linguagem de programação, que também possui uma gramática e, portanto, um

conjunto léxico e regras sintáticas, não aceita a abstração e os vícios de linguagem com os quais estamos acostumados. Vejamos um exemplo. Se dissermos a alguém: *"Vá ao mercado e, se tiver maçãs, compre uma. Se tiver laranja, compre duas."* Provavelmente todas as pessoas comprariam uma maçã (se tivesse) e comprariam duas laranjas (se tivesse). As duas frases geram dúvidas na escrita de um algoritmo que tome por base operadores lógicos *booleanos*, presentes em qualquer linguagem de programação. Haveria, neste caso, a possibilidade de se entender que se tivesse laranja à venda, o correto seria comprar duas maçãs em vez de uma, e não comprar nenhuma laranja. Em outro exemplo, tomemos por base o texto referente aos requisitos específicos para concorrer a uma vaga de professor em uma universidade pública federal: *"curso superior em informática; ou estatística; ou matemática; ou engenharia(s) e mestrado e/ou doutorado em informática; ou estatística; ou matemática; ou engenharia(s)".* Uma das interpretações erradas que se pode fazer após a análise do texto é de que somente o graduado em engenharia precisa ter mestrado ou doutorado para concorrer. Como interpretar os pontos vírgula presentes no texto? E os conectores "*e", "ou" e "e/ou"* ? Professores de física, por exemplo, já se preocupam com as preconcepções dos alunos há tempo. Precisam levantar os conhecimentos prévios alternativos que os alunos têm acerca de conceitos como o de calor, peso, energia e aceleração para avançarem de forma construtiva nos conteúdos ministrados. Ao professor de lógica da programação para qualquer nível de escolaridade cabe reservar um bom espaço em seu plano de ensino para tais desconstruções e reconstruções, a fim de preparar o aluno para a rigorosa interação que se apresentará quando do contato dele com a máquina.

As estruturas de programação também são passíveis de possuírem preconcepções alternativas por parte dos alunos e, assim sendo, devem ser objeto de análise cautelosa na hora de serem ensinadas. Tomemos por base o conceito de *loop* em linguagens de programação. Usa-se *linguagens de programação*, no plural, pois todas elas agregam tal conceito uma vez que os *loops*, também conhecidos como laços de repetição ou iterações, são estruturas fundamentais para a implementação de algoritmos em computador. Chevallard (1991), no âmbito da matemática, debruçou-se sobre a forma de apresentação do saber sabido pela comunidade científica para o saber ensinado praticado nas escolas, criando a teoria da transposição didática, que, em sentido restrito, designa a passagem, portanto, do saber sábio para o saber ensinado. Ora, podemos perceber semelhanças entre a matemática, enquanto ciência mas também enquanto linguagem, e o ensino de algoritmos. Algumas ferramentas de apoio ao ensino de programação como o Alice[1] e o Scratch[2] oferecem formas de transposição de conceitos de estruturas de programação que permitem ao professor trabalhar o conceito e a preconcepção do aluno de maneira conjunta. Em alusão ao exemplo do conceito de *loop*, citado no início deste parágrafo, pode-se programar o giro da hélice de um helicóptero em um laço de repetição em vez de se ensinar iterações com base em conceitos matemáticos. O levantamento das preconcepções permitirá ao professor saber isso, mas provavelmente o conhecimento prévio pelo aluno do giro contínuo das hélices de um helicóptero pode servir de subsunçor para o novo aprendizado sendo proposto. O que se propõe, em outras palavras, é que se inicie o contato com um novo conceito ou estrutura de programação com um problema de domínio prévio por parte do aluno. Neste exemplo, sugere-se abordar **inicialmente** o conceito de *loop* pela implementação do giro de hélice de um helicóptero e não calculando o número de divisores inteiros de um número natural[3]. O estudo de Souza et al. (2013) corrobora o exposto ao apresentar

que 40% dos 80 alunos que participaram da pesquisa, estudantes de disciplina de ensino de lógica da programação, reportam dificuldades no entendimento do problema a ser resolvido. O entendimento do problema a ser resolvido é uma etapa anterior ao entendimento do conceito ou estrutura de programação que se está querendo ensinar.

Estas propostas pedagógicas sendo apresentadas são independentes das estratégias para o ensino de algoritmos (RAPKIEWICZ et al., 2006). Não questionam a linguagem de programação usada (NOSHANG et al., 2014), o uso intenso ou reservado de computadores para implementação de algoritmos[4], nem mesmo o uso de ambientes virtuais de ensino e aprendizagem - AVEAs - ou de objetos de aprendizagem (AMARAL et al., 2015). Também não questionam o estudo de fatores afetivos e suas relações com a aprendizagem de algoritmos (BERCHT et al., 2005). Busca-se, por outro lado, reduzir o caráter tecnicista empregado por muitos professores no ensino de lógica da programação. A seção seguinte do artigo complementa a sugestão de que o ensino de algoritmos pode sim espelhar-se em estratégias usadas no ensino de ciências e matemática garantindo certo grau de liberdade ao professor, que, no papel de mediador, não precisa e não deve ensinar tudo ao aluno, e, construtivamente, deve valorizar a presença do erro dentro do processo. Neste particular, cabe uma referência à epistemologia de Gaston Bachelard quanto à valorização do erro no processo de aprendizagem, e conforme explica Lopes (1996),

> … Bachelard nos coloca o desafio de repensar como interpretamos o erro no processo de ensino-aprendizagem. Se o erro possui uma função positiva na gênese do saber, cabe procurarmos pensar sobre a necessidade dos estudantes errarem no processo de ensino-aprendizagem. O erro deveria, então, deixar de ser encarado como o oposto do conhecimento verdadeiro. O erro é constitutivo do processo de construção do conhecimento. (LOPES, 1996).

3. Números primos e o paradoxo do contrato didático

O conceito de número primo é aplicável ao conjunto dos números inteiros, mais comumente voltado ao conjunto dos números naturais, sendo qualquer número natural *maior que o número "um"* considerado primo se possuir somente dois divisores inteiros naturais: o número "*um"* e ele próprio. Números primos e a fatoração dos mesmos constituem importantes mecanismos na geração de informações criptografadas processadas por computadores para codificação de dados digitais armazenados ou transmitidos por canais de comunicação. A atividade didática sendo proposta para análise, de geração de números primos, não tem relação direta com a aplicação ou utilidade de números primos grandes[5] para fins de criptografia descrita anteriormente, ficando restrita ao método utilizado para tanto através da criação de algoritmos executados por computadores[6].

Para Brousseau (1996), todos os procedimentos em que o professor não dá a resposta são aceitáveis para levar o aluno a dar à luz esse saber. No aprendizado de algoritmos, o aluno, prestes a realizar a construção de um algoritmo para geração de números primos, já detém (ou já deveria deter) conhecimentos acerca de estruturas de programação que o permitam obter sucesso na atividade. Tais estruturas permitem a realização de testes condicionais lógicos com variáveis (se-então-senão), iteração através de laços de repetição e uso de diferentes operadores lógicos, relacionais e

aritméticos. Explicitado o conceito de número primo ao aluno, pode-se experimentar esta proposta de Brousseau na espera de resultados, que podem variar já que soluções algorítmicas são distintas, mesmo que com o propósito de resolverem o mesmo problema. Busca-se evitar, assim, induzir o aluno a *uma resposta correta.* A figura 1 mostra um programa transcrito a partir de um algoritmo simples de geração de números primos em linguagem de programação Pascal.

```
var D,N,I,R: integer;

begin
N:=2;

 while(true)do
 begin
   D:=0;
   for I:=1 to N do
     begin
       R:=N mod I;
       if(R=0)then D:=D+1;
     end;
   if(D=2)then writeln(N);
   N:=N+1;
 end;

end.
```

Figura 1 – Um programa em linguagem Pascal para geração de números primos

Porém, no gerenciamento de um contrato didático[7] feito por regras de responsabilidade na relação entre professor, aluno e saber, figura um paradoxo pois, "de um lado, o professor precisa orientar sua prática de modo a não deixar tudo explícito ao aluno, para não colocar em risco a aprendizagem; de outro lado, se o professor não faz a necessária mediação, rompe com o contrato na relação didática" (RICARDO, SLONGO e PIETROCOLA, 2003). Existe, em tais atividades, pelo menos duas questões importantes passíveis da mediação do professor: uma diz respeito ao tempo de execução do algoritmo, e outra refere-se à capacidade do mesmo em lidar com números grandes, que por vezes excedem a capacidade de representação dos tipos de variáveis sendo usadas. O professor pode querer mais um aprimoramento para que o algoritmo respeite o conceito matemático de número primo ignorando os números "*zero"* e "*um"*, pois não se tratam de números primos por definição, e com vistas a simplificar o trabalho, pode sugerir que considerem-se apenas os números naturais como passíveis de processamento. Tais exceções mediadas pelo professor caracterizam-se como perturbações no contrato estável, conforme Ricardo, Slongo e Pietrocola (2003), e que visam um melhor gerenciamento deste paradoxo presente no contrato didático. Se desafiado por intermédio do professor, cabe agora ao aluno aprimorar o algoritmo contemplando as exceções: permitindo que o mesmo consiga trabalhar com números grandes e que execute mais rapidamente. Abre-se, assim, possibilidade de avanços em estudos sobre o problema, que neste caso específico dizem respeito a questionamentos sobre o limite de representação numérica dos tipos de dados envolvidos e sobre o

aprofundamento do conceito matemático de número primo. A saber, algumas linguagens de programação como o Pascal fazem diferenciação entre variáveis numéricas a fim de diminuir o consumo de memória e agilizar o processamento das mesmas. Se são dadas somente 10 opções de valores a serem armazenados em uma variável, por que utilizar um tipo de dado que reserve espaço para 65536 números e não uma variável de um tipo de dado que suporte até no máximo 256 valores, por exemplo? Então, para que se possa trabalhar com números grandes[8], é necessário escolher o tipo de dado correto para as variáveis em questão, ou seja, que suporte a representação dos números com os quais se deseja trabalhar. Também é possível ao aluno, na medida em que analisa os resultados das primeiras versões de seu algoritmo, ou mesmo ao fazer um aprofundamento conceitual, perceber que o único número primo que é par é o número 2. Isto permite a construção de uma solução algorítmica que ignore todos os números pares com exceção do número 2, o que repercute na performance de geração dos mesmos pois o algoritmo tende a ser executado com o dobro da velocidade daquele anteriormente aceito. A figura 2 mostra uma alteração no programa que ignora números pares maiores que o número 2. Encaminha-se, dessa forma, a uma atividade cercada por descobertas e desafios, devendo assumir o professor um papel de mediação que permita aos alunos trabalharem de forma ativa e

> ...movidos por situações-problema que desafiam e geram interesse. Nesse enfoque, a criatividade, a tomada de decisão e o exercício da autonomia têm lugar garantido e se desenvolvem no decorrer do processo de aprendizagem. Essa perspectiva epistemológica e pedagógica pauta o Contrato Didático na interação professor - aluno - saber, dialetizando seus papéis. Desse modo, o aluno é colocado permanentemente diante de situações novas que o levam a propor hipóteses, discutir suas próprias ideias, refletir criticamente, enfim, põe o aluno em ação, em permanente auto-reflexão. (RICARDO, SLONGO e PIETROCOLA, 2003).

```
var D,N,I,R: integer;

begin
writeln('2');
N:=3;

 while(true)do
 begin
   D:=0;
   for I:=1 to N do
     begin
       R:=N mod I;
       if(R=0)then D:=D+1;
     end;
   if(D=2)then writeln(N);
   readln;
   N:=N+2;
 end;

end.
```

Figura 2 – Programa alterado para analisar somente números ímpares, tratando o número 2 como exceção.

A mediação pode incentivar o aluno a buscar cada vez mais performance e aprofundamento. Não há divisores de um número que sejam maiores que a metade dele, com exceção dele próprio. Assim, se o algoritmo não mais analisar possíveis divisores maiores que a metade do número sendo verificado, novo ganho de performance poderá ser observado.

Agora vejamos a riqueza do exemplo sendo analisado. Neste ponto, outro paradoxo previsto por Brousseau toma forma. Se o aluno pôde satisfazer-se em dominar o conceito de número primo, que possui somente dois divisores inteiros entre o número "*um"* e ele próprio, terá que desconstruir de imediato tal conceito para poder implementar um algoritmo que execute mais rapidamente. Brousseau chamou isso de *Inadaptação a uma Adaptação Posterior.* Tal paradoxo de razão epistemológica, segundo Brousseau (1996), reside no fato de que a sobreadaptação de um saber ou de um entendimento à solução de uma situação particular não é, necessariamente, um fator favorável à solução de uma situação nova. Isto porque

> … se os alunos se adaptaram bem às situações que lhes foram propostas, compreenderam melhor as razões das suas respostas e as relações do seu saber com os problemas, pelo que será mais difícil, em seguida, alterar esse saber a fim de torná-lo correto e de o completar. (BROUSSEAU, 1996).

Com isso, a questão epistemológica que deve vir à tona tem relação com o fato de que o conceito inicial aprendido, no decorrer do processo de sintonia daquele algoritmo, deve ser desconstruído. Mais do que isso: tal conceito pode, sim, ser alterado. Ora, dizer que um número primo é um número inteiro que possui apenas dois divisores inteiros entre o número *"um"* e ele próprio não é o mesmo que dizer que trata-se de um número inteiro que não possui nenhum divisor inteiro maior que o número *"um"* e menor que ele próprio? E quantos outros conceitos corretos ainda podem ser formulados?

A figura 3 demonstra uma possível solução para tal alteração, bem como apresenta variáveis de um tipo[9] que permite trabalhar com números maiores.

```
var D,N,I,R: qword;
begin
writeln('2');
N:=3;
 while(true)do
 begin
   D:=0;
   I:=1;
   while(true)do
     begin
       R:=N mod I;
       if(R=0)then D:=D+1;
       if(I>trunc(N/2))then break;
       I:=I+1;
     end;
   if(D=1)then writeln(N);
   readln;
   N:=N+2;
 end;
end.
```

Figura 3 – Programa otimizado para não considerar divisores maiores que a metade do próprio número e permitindo o processamento de números grandes

Outras possíveis otimizações poderiam ser feitas, como por exemplo a quebra de processamento caso o algoritmo encontre um divisor qualquer maior que um, pois nesse caso, como a validação está sendo feita até a metade do próprio número analisado, para que seja primo não pode haver nenhum divisor maior que o número 1 até o número que corresponde à metade do próprio número analisado. De forma ainda mais aprimorada, poder-se-ia limitar a busca por divisores inteiros até o número que corresponde à raiz quadrada do número analisado, resultando em um algoritmo bastante otimizado, permitindo ao aluno perceber mais uma nova forma de conceituar um número primo.

4. Reflexões e considerações

Muitas são as contribuições das outras áreas do saber à área da computação. Em particular as vindas do ensino de ciências com uma aproximação ainda maior com o ensino de matemática, talvez devido à própria simbiose genética entre matemática e computação. O levantamento prévio de preconcepções sobre as estruturas de programação sendo estudadas pode dar novo rumo à forma de mediação a ser utilizada pelo professor. O uso de diferentes ferramentas para ensino da programação que possibilitem trabalhar conhecimentos prévios e construir novos entendimentos também pode facilitar o avanço do estudante. Estudos atuais sobre novas abordagens de ensino de algoritmos, alguns deles citados nas referências bibliográficas deste artigo, devem ser experimentados e questionados. Mas a grande contribuição sendo proposta tem inclinação epistemológica. O professor de programação precisa saber que tem liberdade para lidar com os paradoxos e perturbações existentes no contrato didático. Em seu papel de mediador, precisa fazer perceber que o erro faz parte do processo de aprendizagem e tirar proveito disso. Enfim, precisa posicionar-se contra um comportamento positivista-tecnicista dando liberdade para novas descobertas em vez de privar o estudante de fazê-las.

Notas

[1] Disponível em http://www.alice.org

[2] Disponível em http://scratch.mit.edu

[3] Diferentes conceitos matemáticos são comumente implementados ao se ensinar o conceito de loop pela primeira vez. O número de divisores inteiros de um número natural foi apenas um exemplo.

[4] O projeto Computer Science Unplugged visa desenvolver o interesse pela ciência da computação através de atividades que não requerem computadores, softwares ou equipamentos especiais. É gerenciado pelo Computer Science Education Research Group da University of Canterbury, Nova Zelândia. Disponível em http://csunplugged.org

[5] O maior número primo atualmente conhecido foi descoberto em 2013 como fruto do projeto GIMPS (Great Internet Mersenne Prime Search), coordenado pelo professor Curtis Cooper da Universidade Central do Missouri (EUA). É o número $(2^{57.885.161} - 1)$.
Fonte: http://www.mersenne.org/primes/?press=M57885161

[6] Em uma atividade regular, utilizando-se um computador pessoal moderno, um estudante deveria conseguir gerar números primos próximos ao número $(2^{64} - 1)$ na velocidade de 1 número encontrado por

minuto, em média, o que evidencia a provável complexidade do algoritmo que os matemáticos usaram para encontrar o maior número primo atual.

[7] Para melhor aprofundamento sobre Contratos Didáticos, ver a obra de Guy Brousseau Fondements et méthodes de la didactique des mathématiques. Recherches en Didactique des Mathématiques. La Pensée Sauvage, v. 7, n. 2, Grenoble, 1986.

[8] Um número "grande" em matemática é bastante relativo. Nestes exemplos didáticos, está assumindo-se como um número "grande" um número natural com mais de 15 dígitos.

[9] Neste caso, o tipo de dado usado qword permite trabalhar com números até o número 18.446.744.073.709.551.615 (= 1,884 x 10^{19} ou 2^{64})

Referências bibliográficas

AMARAL, E.; HUEGEL C.; GOMES M.; BECKER L.; CAMARGO A.; TAROUCO L. Proposta de um portal web alinhado a teorias de aprendizagem para o apoio ao ensino de programadores iniciantes. **RENOTE: Revista Novas Tecnologias na Educação**, Porto Alegre, v. 13, N. 1, jul. 2015.

BERCHT M.; FERRREIRA L.; SILVEIRA S. Aprendendo a construir algoritmos através da mediação digital. **RENOTE: Revista Novas Tecnologias na Educação**, Porto Alegre, v. 3, N. 1, mai. 2005.

BROUSSEAU, G. Fundamentos e Métodos da Didáctica da Matemática. In: BRUN, J. **Didática das Matemáticas**. Lisboa: Instituto Piaget, 1996.

CHEVALLARD, Y. **La transposition didactique***:* du savoir savant au savoir enseigné. Grenoble: La pensée Sauvage, 1991.

CONWAY, M. **Alice***:* Easy to Learn 3D Scripting for Novices. 1997. 242 f. Tese (Doutorado em Ciência da Computação) - Faculty of the School of Engineering and Applied Science, University of Virginia, EUA. 1997.

LOPES, A. Bachelard: o filósofo da desilusão. **Cadernos catarinenses de ensino de física**, v. 13, n. 3, p. 248-273, 1996.

NOGUEIRA, C.M.I.; SIGNORINI, M.B. Crianças, algoritmos e o sistema de numeração decimal. **Investigações em Ensino de Ciências**, v. 15, n. 2, p. 259-274, 2010.

NOSCHANG, L.; PELZ F.; JESUS E.; RAABE A. Portugol Studio: uma IDE para iniciantes em programação. **Anais do XXXIV Congresso da Sociedade Brasileira de Computação**. Brasília: Universidade de Brasília, 2014.

RAPKIEWICZ, C.; FALKEMBACH, G.; SEIXAS, L.; ROSA, N.; CUNHA, V.; KLEMANN M. Estratégias pedagógicas no ensino de algoritmos e programação associadas ao uso de jogos educacionais. **RENOTE: Revista Novas Tecnologias na Educação**, Porto Alegre, v. 4, N. 2, dez. 2006.

RESNICK, M.; MALONEY, J.; BURD, L.; KAFAI, Y.; RUSK, N.; SILVERMAN, B. Scratch: a sneak preview. **Second International Conference on Creating, Connecting, and Collaborating through Computing**, p. 104-109. Kyoto, Japão, 2004.

RICARDO, E.; SLONGO, I.; PIETROCOLA, M. A perturbação do contrato didático e o gerenciamento dos paradoxos. **Investigações em Ensino de Ciências**, v. 8, n. 2, p. 153-163, 2003.

SOUZA, M. B.; MOREIRA, J. L.; LOBO, F.; ALENCAR, M. A. Uma Abordagem Metodológica voltada para o Ensino-Aprendizagem de Algoritmos. **RENOTE: Revista Novas Tecnologias na Educação**, Porto Alegre, v. 11, N. 1, jul. 2013.