Алгоритм принятия решения о существовании пути в лабиринте.

Составитель задачи: Яроцкий Дмитрий Александрович, и.о. заведующего сектором 10.1 ИППИ им. А.А. Харкевича РАН
Задачу решил студент второго курса ФУПМ – Чеканов Михаил.

Формулировка задачи

Рассмотрим квадратный лабиринт размера $n \times n$, понимая под ним сетку из ячеек, у каждой из которых некоторые из 4 стенок открыты, а остальные закрыты. Нас будет интересовать такая задача: для заданных двух ячеек A и B определить, существует ли проход от A до B. Будем считать, что проход предъявлять не нужно, предполагается лишь ответ в форме «да/нет».

Такую задачу можно решить простым последовательным алгоритмом обхода всех ячеек, соединенных с данной, однако он может потребовать $O(n^2)$ шагов (если путь заполняет почти весь лабиринт). Вопрос: можно ли решить эту задачу быстрее, например за $O(n)$ или даже $o(n)$ шагов, с помощью клеточного автомата (т. е. параллельных вычислителей, локализованных в ячейках сетки и общающихся с ближайшими соседями)?

Решение с помощью клеточного автомата (КА) подразумевает три составные части:

- инициализация КА: предварительное отображение ячеек лабиринта в начальные состояния вычислителей на основании информации о связях ячейки с ближайшими соседями и принадлежности к ячейкам A, B;
- собственно правила итерации КА, т.е. закон изменения состояний вычислителей в зависимости от собственных состояний и состояний ближайших соседей (одновременное однократное изменение состояний всех вычислителей считается за один шаг алгоритма);
- условия принятия окончательного решения о наличии/отсутствии прохода между A и B — можно считать, что решение принимается, когда либо все вычислители (конъюнктивный вариант), либо хотя

бы один (дизъюнктивный вариант) переходят в некоторые выделенные состояния.

В данной работе будет предъявлен алгоритм, решающий данную задачу за $O(n)$ итераций КА и доказана принципиальная невозможность существования алгоритма, работающего за $o(n)$

Вначале приведём упомянутый алгоритм.

Описание клеточного автомата

Состояния ячейки

В каждой ячейке будем хранить информацию о возможности перейти в соседние клетки, а также является ли стартом(в дальнейшем клетка A), финишем(клетка B) или ни тем, ни другим.

Для кодирования данной информации потребуется 6. Условимся кодировать состояние автомата следующим образом:

- 0-й бит равен 1, если данная клетка – клетка A;
- 1-й бит равен 1, если данная клетка – клетка B;
- 2-й бит равен 1, если данная клетка имеет стенку сверху;
- 3-й бит равен 1, если данная клетка имеет стенку справа;
- 4-й бит равен 1, если данная клетка имеет стенку снизу;
- 5-й бит равен 1, если данная клетка имеет стенку слева;

Схематичное изображение битов и кодируемых им объектов изображено на рис.1. На рис.2 приведён пример кодировки состояний автомата.

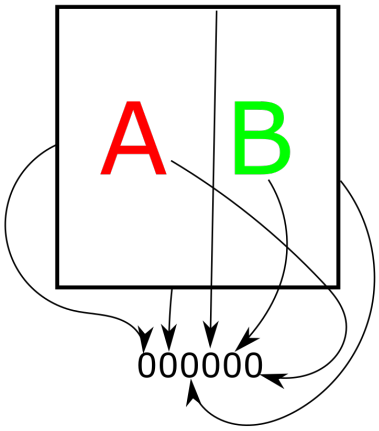

Рис. 1: Связь элементов ячейки с битами

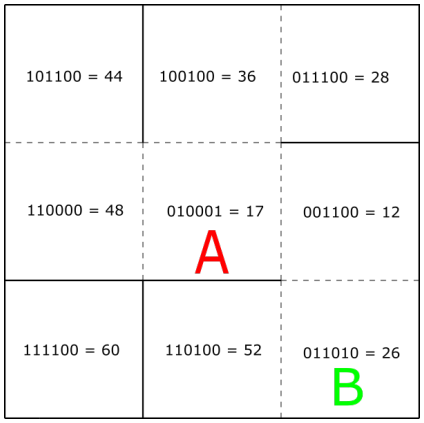

Рис. 2: Пример кодировки ячеек лабиринта

Отметим что, кодировка начальных состояний однозначно определяет стенки в лабиринте, поэтому изменение состояний автоматов можно рассматривать как изменение конфигурации лабиринта.

В качестве инициализации клеточного автомата понимаем задание состояния автоматов в ячейках в соответствии с заданной схемой лабиринта.

Отметим, что мы несколько модифируем данный нам алгоритм. А именно, заключим его в "кожух"из пустых клеток. Это необходимо, для того, чтобы правила, которые мы приведём ниже работали и для граничных клеток лабиринта(такая конструкция гарантирует наличие всех соседей у каждой клетки исходного лабиринта)

Правила изменений состояний автоматов

Как отмечалось в конце предыдущего пункта изменения состояний автоматов можно рассматривать как преобразование лабиринта, поэтому далее будем говорить о добавлении/удалении стенок и перемещение точек старта/финиша, подразумевая соответствующие изменения состояний автоматов.
Список правил следующий:

1 Если клетка содержит одиночную стенку (то есть такая стенка, что

хотя бы с одного конца, она не соединена ни с одной стенкой)и при этом клетки содержащие , удалить эту стенку.

2 Если клетка содержит левый-верхний уголок, добавить в эту клетку нижний правый уголок. Если помимо этого из левой клетки можно перейти в левую верхнюю, то удаляем левую стенку, аналогично, если из верхней можно перейти в левую верхнюю удаляем верхнюю стенку.

3 Если клетка содержит старт/финиш, и из неё есть путь вниз, сместить вниз – в противном случае, если возможно, сместить вправо(если нельзя и этого, оставить без изменений)

Для наглялности на(рис. 3, 4, 5) продемонстрированы применения этих правил.

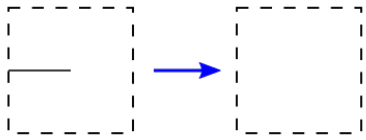

Рис. 3: Правило 1

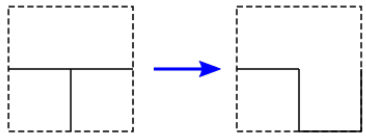

Рис. 4: Правило 2

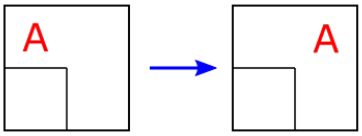

Рис. 5: Правило 3

Условия окончания работы и принятия решения

Автомат прекращает работу, если в одной из ячеек 0-й и 1-й биты равны 1. В этом случае старт совпадёт с финишем, что означает существование пути в исходном лабиринте.

Автомат также прекратит работу, если старт/финиш окажется в изолированной клетке. В этом случае пути нет.

Далее будет показано, что такой клеточный автомат позволяет "свернуть"лабиринт, так, что если старт и финиш располагались в одной компоненте связности, то в преобразованном лабиринте они окажутся в одной клетке. В противном случае старт или финиш окажется в изолированной клетке.

Пример работы алгоритма

Для наглядности приведём примеры работы клеточного автомат в виде схем лабиринтов, соответствующих состояниям клеток на каждом такте.

Первый пример(рис. 6)) иллюстрирует решение на лабиринте, на котором алгоритм обхода в глубину работает за $O(n^2)$.

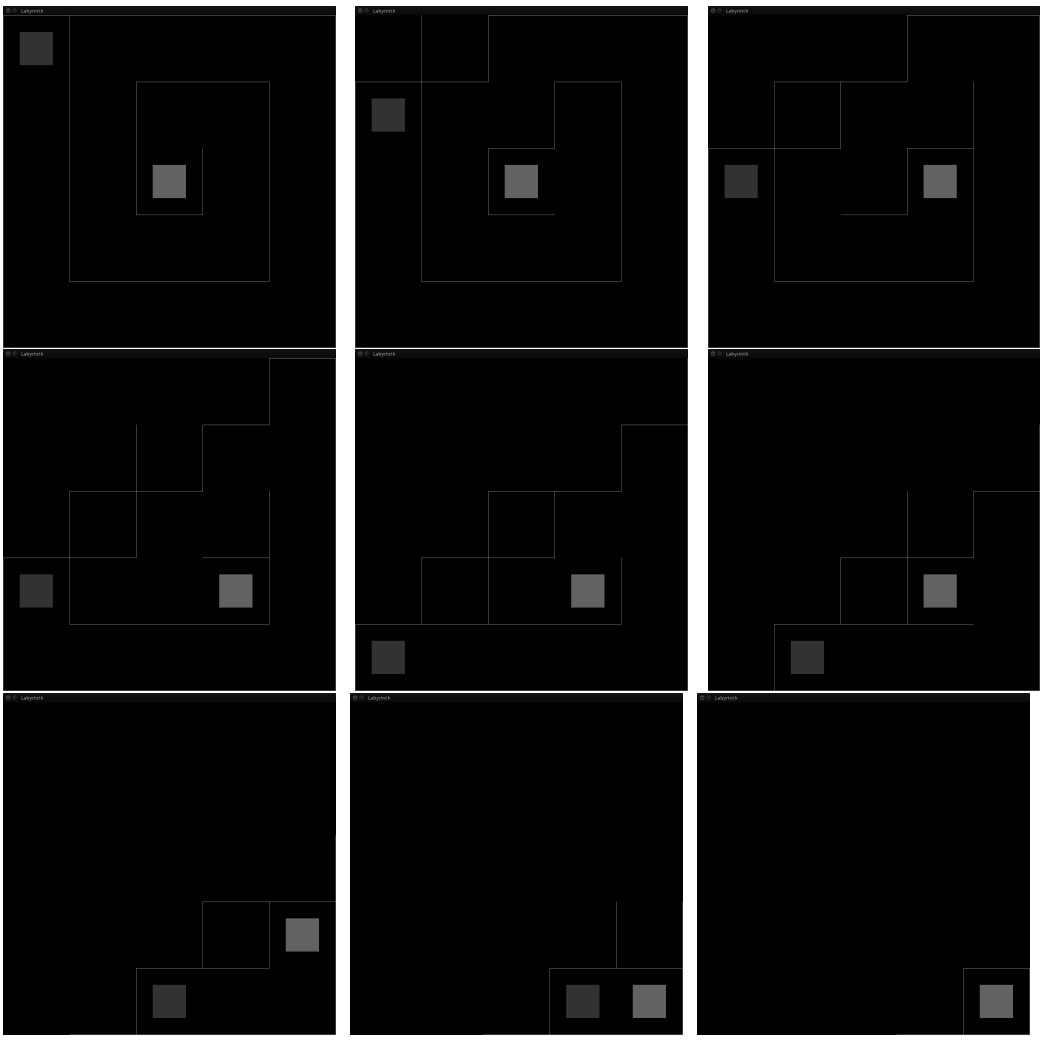

Рис. 6: Пример работы алгоритма(Путь есть)

Второй пример(рис. 7) иллюстрирует работу на лабиринте, где не существует пути из A в B

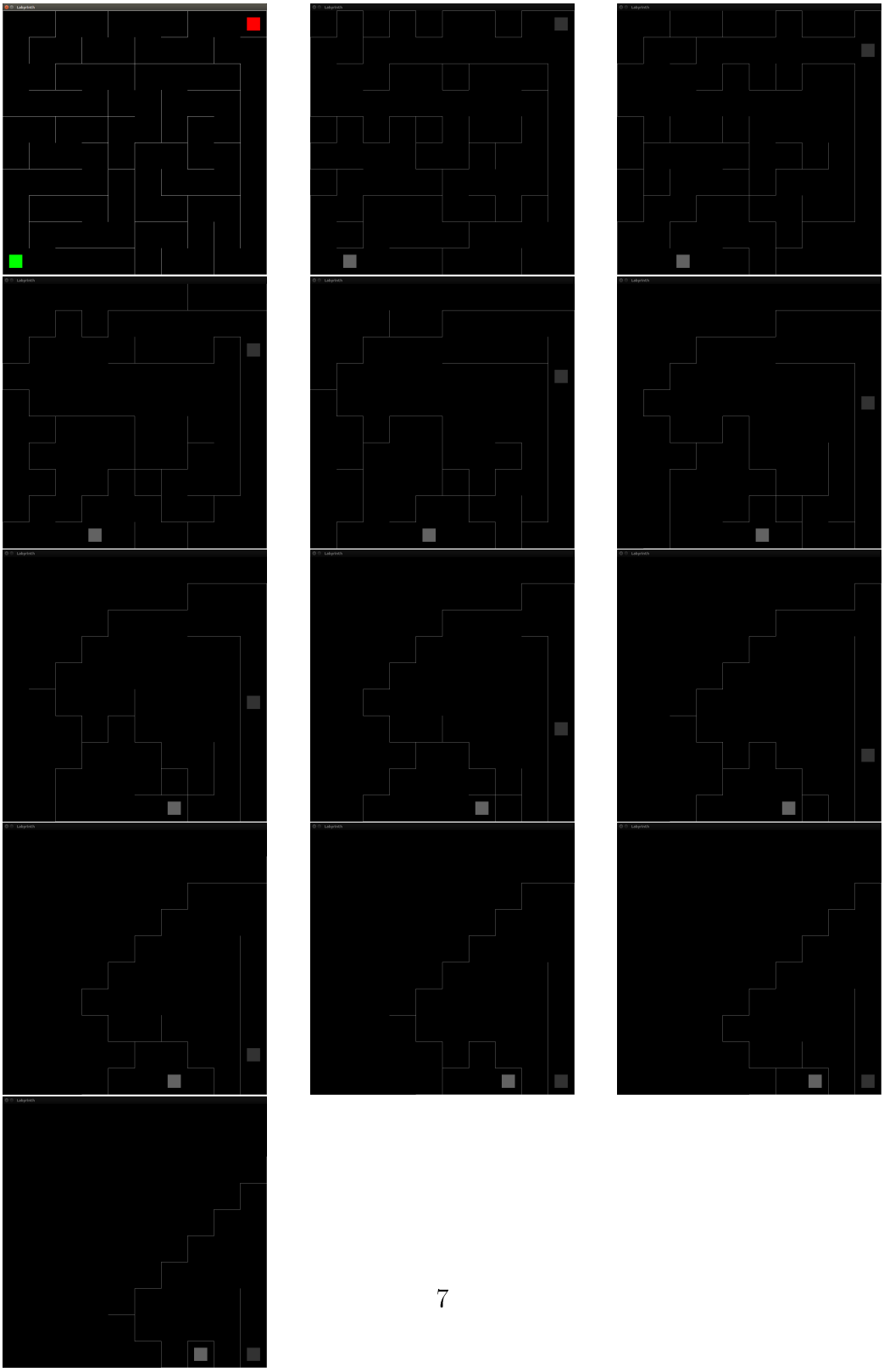

Рис. 7: Пример работы алгоритма(Пути нет)

Корректность алгоритма

Покажем, что если существует путь из A в B, то он не исчезает при работе алгоритма.

Утверждение. *Если клетки A и B принадлежат одной компоненте связности, то после каждого шага алгоритма, они будут оставаться связными.*

Доказательство. Воспользуемся доказательством от противного. Пусть на некотором шаге алгоритма, старт и финиш располагались в одной компоненте связности, а на следующем шаге, путь из A в B перестал существовать. Заметим, что путь может исчезнуть только при добавлении стенок. Из указанных нами правил добавление стенок происходит только по правилу 2, которое изменяет клетки с левым верхним уголком. Таким образом, если между A и B существовал путь, не проходящий через такие клетки, то путь сохранится и в лабиринте на следующем шаге. Значит, из нашего предположения следует, что исходные пути из A в B содержат левый верхний уголок. Докажем, что правила преобразования изменяют лабиринт, таким образом, что появляется обходной путь из A в B.

Вначале рассмотрим случай, когда уголок не является стартом/ финишем. Поскольку, предполагается, что клетка с уголком является частью пути, делаем вывод, что правая и нижняя стенки открыты. На рис. 8 показаны все возможные, ситуации в окрестности и уголка и конфигурации этой окрестности после применения правил. Легко видеть, что хоть мы и исключили уголок из пути, но при этом появился "обход"через клетку справа-снизу относительно уголка. Таким образом, добавление новых стенок не нарушила связности старта и финиша.

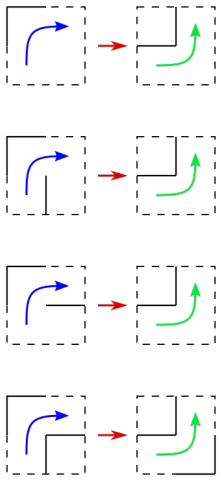

Рис. 8: Изменение схемы лабиринта, с возникающим обходом(штрихом указаны отрезки состояния, которых неизвестны)

Рассмотрим теперь случай, когда уголок является стартом или финишем. Поскольку существует некоторый путь, то либо правая, либо нижняя стенка открыты. По 3-му правилу старт/финиш переместится по возможности вниз, а если это невозможно – вправо. Если при этом старт/финиш окажется в следующей/предыдущей клетке "старого"пути, то в силу доказанного ранее, путь из A в B по-прежнему будет существовать. Если же, старт/финиш переместились в клетку, которая не была частью пути(то есть сместились вниз, когда старый путь шёл вправо/справа, заметим, что при этом и правая и нижняя стенка должны быть открыты, как и в первой части доказательства), то по 8 видно, что существует возможность добраться в клетку "старого"пути.

Таким образом, мы показали, что правила лабиринта изменяют его так, что расположние клеток A, B относительно компонент связности не изменяется. □

Далее покажем, что если пути между A и B не существовало, то изменения лабиринта не приведут к его появлению.

Утверждение. *Если клетки A и B принадлежат разным компонентам связности, то на каждом шаге алгоритма, эти клетки будут принадлежать разным компонентам.*

Доказательство. Появление пути означает, что две клетки лабиринта ранее принадлежащие разным компонентам свяязности теперь принадлежат одной. Мы покажем, что наши правила исключают возможность присоединения клетки с сохранением связности со своей предыдущей компонентой.

Заметим, что первое правило удаляет стенку между клетками одной компоненты(так как одиночную стенку, очевидно, можно обойти). То есть как и в предыдущем доказательстве, мы рассматриваем клетки, к которым применяется второе правило. Будем вначале предполагать, что старт/ финиш не меняют своих клеток. На рис. 9 показаны, все возможные конфигурации отщепляемой клетки от компоненты связности и их состояния при преобразовании. Как видно из рисунка, клетка присоединяясь к одной компоненте связности отгараживается от ранее смежных с ней клеток. Таким образом, через эту клетку не может проходить путь из компоненты финиша в компоненту старта. Поэтому единственный вариант, появления пути, что сама клетка финиша или старта была перемещена в другую компоненту связности. Покажем, что и этого не могло произойти.

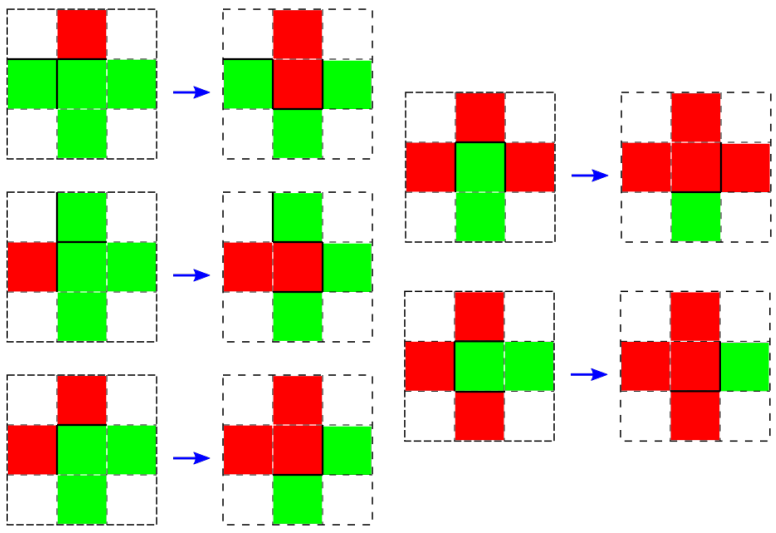

Рис. 9: Клетка, меняющая компоненту связности. Одинаковым цветом выделены клетки которые могут лежать в одной компоненте свзяности, разными – точно разные компоненты связности

1-й случай: старт/финиш лежит в клетке с правым нижним уголком.

В этом случае старт/финиш не смещается. Такая клетка не содержит левый верхний уголок(так как иначе это было бы состояние принятия решения) и, следовательно, не открывает стенки(кроме одиночных, но как уже упоминалось их удаление не меняет компоненты связности).

2 - случай – клетка не содержит правый нижний. В этом случае, старт/финиш смещается в клетку(заметим, не содержащую левый верхний уголок). Поэтому компонента связности сохраняет старые клетки, за исключением, возможно, некоторых клеток, содержащих левый верхний уголок. Последние же как было показано раннее, при смене компоненты изолируются от бывших соседей, поэтому слияния компонент не происходит. □

Итак, мы доказали, что если между A и B существовал путь, то в преобразованном лабиринте путь тоже будет существовать. Если же пути не было, то он не появится. Следовательно, в силу уменьшения объёма лабиринта(данный факт, будет доказан в следующем пункте) одно из состояний принятия решения будет достигнуто, а доказанные утверждения означают, что полученный результат – корректен.

В следующем пункте, мы покажем, что алгоритм работает за $O(n)$ тактов.

Время работы алгоритма

Напомним, что в конце первого пункта мы условились "вкладывать"данный нам лабиринт в "кожух"из слоя пустых клеток.

Утверждение. *На каждом шаге алгоритма количество клеток, принадлежащих лабиринту уменьшается, остальные присоединяются к "кожуху. Алгоритм завершится не более чем за* $2n$ *итераций клеточного автомата.*

Доказательство. Рассмотрим левую верхнюю клетку лабиринта. Она содержит левый верхний уголок. После применения 2-го правила эта клетка становится частью кожуха(появляются правая и нижняя стенки, старые – удаляются). При этом появляются два левых верхних уголка. На следующем шаге эти два уголка добавятся к кожуху, появтся новые три. Продолжая эти действия (см.10) видим, что за n шагов, лбиринт будет заключён в правом нижнем треугольнике исходного лабиринта(с каждым шагом из верхнего ряда исключается одна клетка лабиринта).

Ещё через n шагов, лабиринт свернётся в клетку (если до этого не произошло событие принятия решения). Так как все правила сохраняют старт и финиш внутри лабиринта, в этот момент они окажутся в одной клетке, что является событием принятия решения.

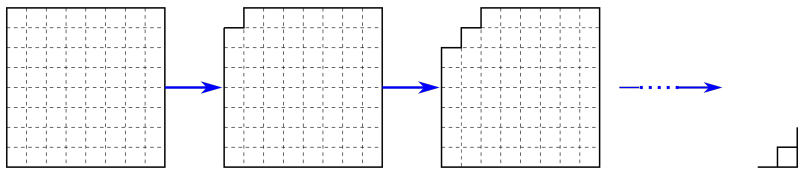

Рис. 10: "Сворачивание"лабиринта

Отметим, что сворачивание лабиринта происходит назависимо от его содержимого, то есть конфигурация лабиринта не может задерживать свёртку и не более чем через $2n$ операций перейдёт в состояние принятия решения. □

Итак, мы доказали корректность алгоритма и сложность его работы. Тем самым мы подтвердили, что алгоритм удовлетворяет условию задачи.

Наконец, покажем принципиальную невозможность существования алгоритма, решающий поставленную задачу за $o(n)$ тактов.

Несуществование корректного алгоритма o(n)

Утверждение. *Не существует корректного алгоритма, решающего поставленную задачу, сложности* $o(n)$

Доказательство. Рассмотрим произвольный корректный алгоритм, решающий задачу. Запустим его на трёх схожих лабиринтах, отличающихся наличием отсутствием двух стенок(см. рис 11)

Как видно из рисунка в случаях a) и b) путь из A в B существует. Клетками $1, 2$ помеченны клетки состояние которых может отличаться от состояний тех же клеток в других лабиринтах (если клетка в разных лабиринтах обозначена одной цифрой – их состояния совпадают). Так как пути существуют, ни одна из клеток, помеченных цифрами не может быть состоянием принятия отрицательного решения(то есть состояние которое означает, что в лабиринте нет пути).

Обратимся теперь к лабиринту c). Очевидно, в нём пути не существует. При этом клетки инициализированы так же как и в лабиринтах, где пути нет. Это означает, что пока "возмущение"от клеток расположенных у т-образных соединений не наложатся(при этом состояние клетки, в которой произошло наложение неопределенно и может быть состоянием, соответствующим отсутствию пути в лабиринте), алгоритм не завершит работу. Но наложение произойдёт через $n/2$ тактов. То есть количество тактов алгоритма ограниченно снизу $n/2$, откуда следует $T(n) \neq o(n)$

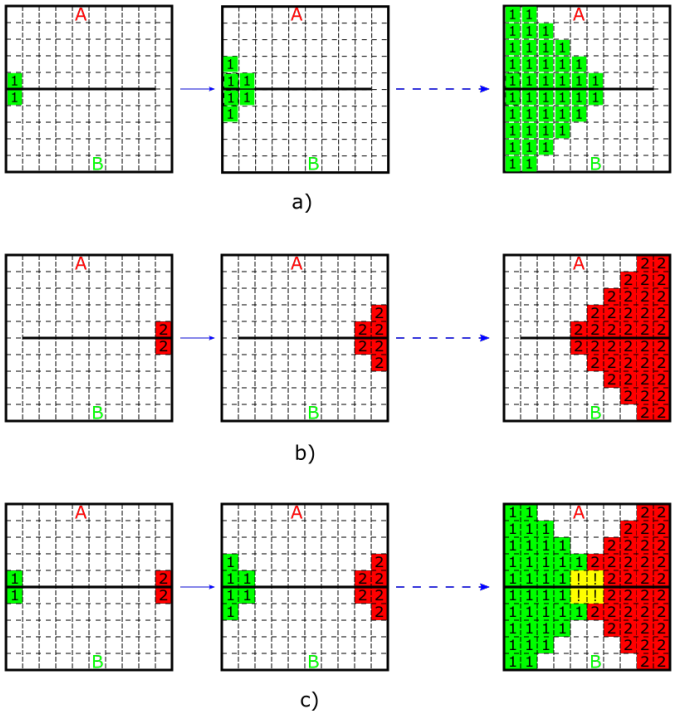

Рис. 11: Изменение состояний клеток в процессе выполнения алгоритма. Цифрами и пустыми клетками отмечены клетки, состояние которых не означают отсутствие пути в лабиринте, состояние жёлтых клеток неопределенно и может приводить к завершению алгоритма

□