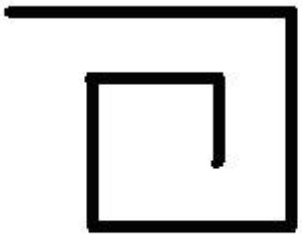
Очень нестандартная задача. Одинокий путник с плохой памятью

Условие задачи

На квадратном поле установлены препятствия произвольной формы, и два пункта ***A*** и ***B***. Перед путником поставлена задача — найти путь из ***A*** в ***B***. Известно, что путник не располагает картой местности, у него очень плохое зрение, и очень плохая память, настолько плохая, что он практически не может запомнить приметы пройденного маршрута. Из средств ориентации путник располагает естественным чувством ориентации в пространстве своего тела, то есть он может сказать, где у него левая рука, а где правая, также у него есть прибор, напоминающий компас. Этот прибор всегда, в любой точке поля показывает направление на пункт ***B***.

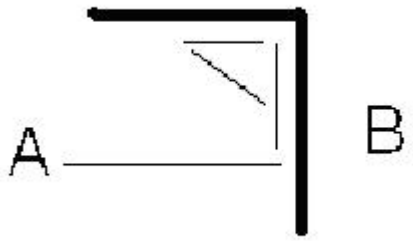
Задача может показаться простой. Действительно рассудим так. У путника есть только две возможности для движения: по пустому пространству и вдоль препятствия. Стратегия его движения в первом случае очевидна. Он идет по компасу (будем это устройство называть компасом). Когда он встретит на своем пути препятствие, его стратегия усложняется не намного. Путник просто идет вдоль препятствия до тех пор пока не обойдет его, а обойдя, опять движется по пустому полю.

Рассуждение вроде бы верное, за исключением одного — неясно, откуда он узнает, что препятствие пройдено. Этот факт легко установить только в том случае, когда препятствие представляет собой тонкую стенку. В примере, приведенном далее, это очень даже непросто. Представьте себе, что путник попал в такой лабиринт и подумайте, откуда, не имея карты, он может взять информацию, что лабиринт пройден.



Можно попробовать выйти из положения, используя следующую стратегию: движемся вдоль стенки препятствия, держась за стенку правой (левой) рукой, до тех пор, пока не появится возможность движения в сторону пункта ***B***. Как только появляется возможность такого движения отрываемся от стенки и движемся по компасу.

Идея неплохая. Заметьте, здесь используется способность путника распознавать, где право и где лево. Это хорошая примета. В решении корректно поставленных задач должны использоваться все условия, это знают те, кто имеет опыт решения математических или физических задач. Но, несмотря на хорошую примету, идея ошибочна. Ее ошибочность показывает следующая картинка:



Путник начинает движение от ***A*** к ***B***. Встречает препятствие. Пользуясь правилом правой руки, он начинает движение вдоль вертикальной стенки и движется, пока не встретит горизонтальную стенку. Пока возможности двигаться по компасу нет, вертикальная стенка этого не позволяет. Далее он начинает движение вдоль горизонтальной стенки и, пройдя совсем немного, обнаруживает, что движение в направлении ***B*** возможно и начинает движение по компасу. Но, пройдя совсем немного путник опять упирается в вертикальную стенку, более того, он упирается в одну из тех точек, в которой он уже был и далее начинает двигаться по кругу.

Два примера, рассмотренные ранее, были нужны для того, чтобы показать сложность задачи, необходимость тщательного анализа.

Еще один важный вопрос связан с проблемой существования решения. Это любимая проблема фундаментальной математики. Программисты в основном «прикладники», и если нам дана задача, то мы априори полагаем, что решение она имеет. Но это совсем неочевидно, огромное количество задач вообще алгоритмически неразрешимо, а условие исследуемой задачи выглядит весьма сложно, и было бы полезно подумать, а для какого типа препятствий можно найти решение гарантированно.

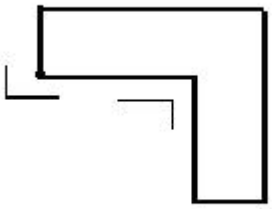
Решение задачи существования может оказаться весьма сложным, тем более, что мы пока даже не решили, как провести классификацию препятствий и что вообще такое «тип препятствия». Но мы сейчас поступим, как самые обычные «прикладники», полагая, что алгоритмическое решение есть, найдем его, а затем, глядя на полученный алгоритм, попробуем вывести условия, для которых он работает. Это вполне разумный подход. Действительно не совсем ясно, как можно искать условия работоспособности алгоритма, не имея алгоритма.

Кое-что про будущий алгоритм уже известно. Очевидно, что единственная разумная стратегия поведения путника на пустом пространстве, это движение по компасу. Других причин для выбора направления у него на пустом пространстве просто нет. Единственный способ движения при обнаружении препятствия — это, конечно, движение вдоль стенок, как уже предлагалось. Использование правила левой или правой руки также вполне целесообразно. Действительно, пользуясь одним из этих правил, можно гарантированно обойти препятствие и вернуться в ту точку, с которой движение вдоль препятствия началось, а следовательно, и та точка, в которой надо продолжить движение по компасу, также будет достигнута. Вопрос остается только в том, как определить точку отрыва от стенки?

Прежде всего, зададим себе вопрос, на основании какой информации, мы можем сделать заключение о такой точке. У этого вопроса есть две формы. Можно спросить, что необходимо знать для определения точки отрыва, а можно спросить, как использовать то, что уже известно. Ответ на любой из этих вопросов даст решение, но вопрос в первой форме выглядит как-то неопределенно. Может получиться так, что мы придумаем

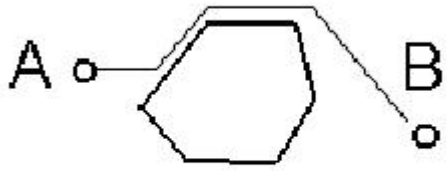
необходимые условия, а их в наличии не окажется. Все-таки более реально исходить из того, что есть.

Что же мы знаем о движении вдоль препятствия? Путь вдоль стены информации в себе не несет никакой. Путник не может определить положение этой стенки в пространстве. Он, наверное, может определить длину пройденного участка в шагах, но это вряд ли ему поможет. Еще одно важное событие, с которым он сталкивается, это поворот. Типов поворота может быть два. Они показаны на рисунке:



Различие между этими двумя поворотами в типе угла. Для одного из них угол вогнутый, для другого выпуклый. Мы так их и назовем: выпуклый и вогнутый повороты. Можно назвать их еще левым поворотом и правым, но первое название нагляднее. Теперь путник может собрать содержательную информацию, а именно, двигаясь вдоль препятствия, он может считать выпуклые и вогнутые повороты. Ясно, что последовательность этих поворотов как-то характеризует форму препятствия, а знание формы как раз и требуется для определения точки отрыва.

А сейчас обратите внимание на следующий интересный факт. Ранее были приведены примеры проблемных препятствий. Их особенность в том, что они имеют вогнутые углы, для обхода которых необходимы вогнутые повороты. Рассмотрим полностью выпуклое препятствие такое, как на картинке далее.



К этому препятствию вполне применим простейший алгоритм отрыва, предложенный в самом начале анализа. Действительно возможно идти вдоль стенок, пока не появится свободное пространство в направлении, показываемом компасом, после чего можно отойти от стенки и пойти к ***B***. Действительно проблемы начинаются, с появлением вогнутых углов.

Проблема озвучена, попробуем применить к ней новый исследовательский прием. Предположим, есть хорошая структура данных. Хорошая в том смысле, что мы умеем с ней работать. В нашей задаче такой хорошей структурой будет полностью выпуклая фигура. И есть плохие структуры, с которыми мы пока не знаем как работать, в нашей

задаче плохая структура — это фигура с вогнутыми углами. Новый прием заключается в следующих действиях:

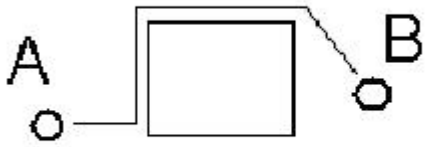
1. Построим пример хорошей структуры.

2. Найдем минимальное изменение, такое что оно превращает хорошую структуру в плохую.

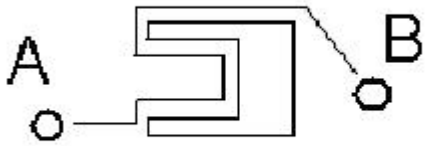
3. Посмотрим, что изменилось и как эти изменения можно компенсировать.

Применим описанный прием к нашей задаче.

Шаг первый. Очень хорошая ситуация.



Шаг второй. Добавляем к квадрату только одну вогнутость.



Что изменилось в траектории движения одинокого путника? Ответ очевиден. Если его траектория, как уже говорилось, характеризуется последовательностью выпуклых и вогнутых углов, то можно заметить, что в данной траектории появилось два вогнутых и два выпуклых угла. Прием дал продуктивную идею: если на препятствии появляется некоторое количество вогнутых углов, то они компенсируются таким же количеством выпуклых. Мы больше не будем приводить примеров, а вы можете попробовать проверить это утверждение на более сложных фигурах.

Теперь можно сформулировать и стратегию поведения путника. Путник движется, исходя из того, что препятствие перед ним хорошее, то есть полностью выпуклое. Из этого соображения путник после каждого выпуклого поворота пытается начать движение по компасу, если это не получается, то он движется вдоль стенки до следующего

выпуклого поворота. Если же ему встречается некоторое количество вогнутых поворотов, то он не делает никаких попыток движения по компасу, пока не пройдет столько же выпуклых поворотов, сколько он прошел вогнутых.

Мы создали работающую стратегию, опираясь на специальный прием построения препятствий, и при этом утверждаем, что наша стратегия будет работать для любого типа препятствий. Это, конечно, не очевидно, для полной уверенности нужно еще доказать следующее утверждение: любое цельное препятствие можно построить из прямоугольника последовательными вырезами из него прямоугольников, но справедливость этого утверждения лежит на поверхности и тратить время на его доказательство не будем.

Еще одна фундаментальная проблема — необходимо выяснить, в каких ситуациях можно найти путь из ***A*** в ***B***, а в каких нельзя. Этим надо бы заняться сейчас, но мы поступим наоборот. Интуиция подсказывает, что ситуаций, в которых можно найти путь, очень много. Поэтому проблему существования отложим на потом. А сейчас займемся техническими деталями.

Главный алгоритм оформим в виде цикла, работающего до тех пор, пока не достигнут пункт B. Тело цикла составим из двух процедур: путь по компасу и обход препятствия. Естественное условие завершение пути по компасу — это встреча препятствия, а условие завершения обхода препятствия — это отход от стенки. Оба условия можно проверять внутри самих процедур, это даже удобно, тогда текст главного алгоритма очень короток:

```
Пока не достигнут пункт назначения, делать
  Выполнить процедуру движения по компасу.
  Выполнить процедуру обхода препятствия.
```

Процедура движения по компасу заключается в прорисовке линии в направлении пункта назначения. Для этого в начале процедуры рассчитывается вектор, начальная точка которого — это текущая точка путника, а конечная — пункт назначения. Затем полученный вектор преобразовывается в единичный и далее шаг движения путника — это смещение его на величину полученного единичного вектора до тех пор, пока не достигнут пункт назначения или не встретится препятствие.

Процедура обхода препятствия технически существенно сложнее. Ключевая сложность в моделировании пространственной ориентировки путника. Не вполне ясно, как обозначить где у него левая рука, а где правая. В процессе обхода путник выполняет различные повороты, в результате которых меняется его ориентировка, и это надо как-то учитывать. Договоримся путника считать точкой на экране монитора с координатами x, y. Тогда его соприкосновение со стенкой препятствия возможно точками, имеющими следующие координаты:(x+1, y), (x, y+1), (x–1, y), (x, y–1). Если эти четыре точки пронумеровать произвольным образом, то номер точки, которой путник соприкоснулся с препятствием, можно считать числовым обозначением его ориентации. Расчет такой исходной ориентации в программе ведет функция ***stenka***. Затем в процессе обхода, каждый раз, когда путник совершает поворот, номер его ориентации в пространстве пересчитывается в соответствии с тем, какой поворот он совершает.

Движение вдоль препятствия — это последовательность шагов, но шагов не одинаковых. Существует несколько ситуаций.

- В направлении точки соприкосновения путника со стенкой нет больше препятствия. В этом случае путник совершает выпуклый поворот и продолжает движение, соприкасаясь со стенкой уже другой точкой, номер которой перевычисляется. В программе такое движение реализуется процедурой ***move1***

❑ В направлении касания со стенкой препятствия двигаться по-прежнему нельзя, тогда путник продолжает движение вдоль стены в соответствии со своей ориентацией. Такой тип движения в программе осуществляется процедурой ***move2***

❑ Путник зашел в угол и должен совершить поворот. Поворот сводится к смене ориентации, или, что то же самое, к смене точки соприкосновения с препятствием. Наш путник совершает повороты против часовой стрелки, точки касания пронумерованы также против часовой стрелки, поэтому смена номера точки сводится к его увеличению, если он меньше 4 (максимальный номер). Если же текущий номер = 4, то он становится равен 1.

В первых двух случаях соответствующим образом изменяются координаты путника, в третьем изменяется ориентация, то есть перевычисляется номер точки соприкосновения путника со стенкой препятствия.

Для определения типа новой ситуации в программе написана специальная функция situation, использующая как текущее положение одинокого путника, так и номер текущей ситуации.

Еще раз напомним, что ключевой момент для обработки любого движения — это ориентация путника, то есть номер точки, которой он в данный момент касается стенки препятствия. Все операции движения жестко привязаны к этой точке, поэтому они организованы в виде выбора из четырех альтернатив.

Полный текст программы приведен в листинге

Листинг

```
program example;
  uses crt,graph;
  const
    n=35;
    ax=10;ay=10;
    bx=230;by=430;
  var
   dr,md,i,x,y,dx,dy,r:integer;
procedure move;
 var
  l,dx,dy,x1,y1:real;
  q:boolean;
begin
  x1:=0;
  y1:=0;
  q:=false;
  l:=sqrt(sqr(bx-x+0.0)+sqr(by-y+0.0));
  repeat
   dx:=(bx-x)/l;dy:=(by-y)/l;
   x1:=x1+dx;y1:=y1+dy;
   if abs(x1)>=1 then
    if getpixel(x+round(x1),y)=0 then
      begin
       x:=x+round(x1);
       putpixel(x,y,15);delay(25);putpixel(x,y,0);
       x1:=0;
      end
    else q:=true;
   if abs(y1)>=1 then
    if getpixel(x,y+round(y1))=0 then
      begin
        y:=y+round(y1);
        putpixel(x,y,15);delay(25);putpixel(x,y,0);
        y1:=0;
```

```
      end
    else q:=true;
  until q
end;
procedure soround;
 var
  k,n,p:integer;
function stenka:integer;
begin
 if getpixel(x,y+1)<>0 then stenka:=1
   else if getpixel(x+1,y)<>0 then stenka:=2
     else if getpixel(x,y-1)<>0 then stenka:=3
       else stenka:=4;
end;
function situation(n:integer):integer;
begin
 case n of
  1: if getpixel(x,y+1)=0 then situation:=1
      else if getpixel(x+1,y)=0 then situation:=2
       else situation:=3;
  2: if getpixel(x+1,y)=0 then situation:=1
      else if getpixel(x,y-1)=0 then situation:=2
       else situation:=3;
  3: if getpixel(x,y-1)=0 then situation:=1
      else if getpixel(x-1,y)=0 then situation:=2
       else situation:=3;
  4: if getpixel(x-1,y)=0 then situation:=1
      else if getpixel(x,y+1)=0 then situation:=2
       else situation:=3;
 end;
end;
procedure move1(n:integer);
{to N}
begin
 putpixel(x,y,0);
 case n of
  1: y:=y+1;
  2: x:=x+1;
  3: y:=y-1;
  4: x:=x-1;
 end;
 delay(25);putpixel(x,y,15);
end;
procedure move2(n:integer);
{to n+1}
begin
 putpixel(x,y,0);
 case n of
  1: x:=x+1;
  2: y:=y-1;
  3: x:=x-1;
  4: y:=y+1;
 end;
 putpixel(x,y,15);delay(25);
end;
function turn90(n:integer):integer;
begin
 if n<4 then turn90:=n+1 else turn90:=1;
end;
begin
  n:=stenka;
  p:=1;
  repeat
   k:=situation(n);
```

```
   case k of
    1: begin
        move1(n);
        p:=p-1;
        n:=stenka;
       end;
    2: move2(n);
    3: begin
        n:=turn90(n);
        p:=p+1;
       end;
   end;
  until p=0;
  delay(25);putpixel(x,y,0);
end;
begin
 randomize;
 dr:=detect;initgraph(dr,md,'');
 for i:=1 to n do
  if i<>15 then
   begin
    setcolor(i);
    setfillstyle(1,i);
    x:=50+random(400);
    y:=50+random(300);
    dx:=2+random(100);
    dy:=2+random(100);
    r:=5+random(30);
    rectangle(x,y,x+dx,y+dy);
    floodfill(x+dx div 2,y+dy div 2,i);
{    circle(x,y,round(r));floodfill(x,y,i);}
   end;
 {setcolor(2);
 rectangle(10,80,400,85);
 setfillstyle(1,2);floodfill(12,82,2);}
 setcolor(15);
 setfillstyle(1,15);
 circle(bx,by,2);floodfill(bx,by,15);
 x:=ax;y:=ay;
 repeat
   move;
   soround;
 until (abs(x-bx)<3)and(abs(y-by)<3);
 delay(1000);
end.
```

Мы потратили достаточно много времени на разработку алгоритма и программы, полученное решение выглядит красиво, и если окажется, что оно применимо к небольшому количеству задач, работа будет напрасной. Поэтому сейчас проделаем небольшую, но важную работу по выявлению условий, при которых одинокий путник, пользуясь разработанным алгоритмом, сможет найти путь из точки ***A*** в точку ***B***.

Критерий будем искать, анализируя алгоритм движения путника. Обратим внимание на два утверждения:

- любое цельное препятствие можно построить из большого прямоугольника, последовательно вырезая из него меньшие прямоугольники;

- путник сможет корректно обойти любое препятствие, построенное по только что сформулированному правилу, пользуясь созданным алгоритмом.

Эти два утверждения позволяют нам сформулировать очень серьезную **теорему**: если вся система препятствий, которую путнику предстоит преодолеть, находится внутри прямоугольника, и путник находится за его пределами, то, пользуясь построенным алгоритмом, путник сможет преодолеть эту систему препятствий. Теорема выглядит правдоподобной, но не очевидной. Попробуем ее хотя бы нестрого доказать.

Обоснование. Если система препятствий представляет собой цельное препятствие, то теорема очевидна из сказанного ранее. Предположим теперь, что система состоит из нескольких несвязанных между собой препятствий. Для доказательства достаточно показать, что в процессе движения путник никогда не приблизится к той стенке прямоугольника, охватывающего систему препятствий, с которой он начал свой путь. Очевидно, что если путник никогда не приблизится, то, следовательно, он будет удаляться и, следовательно, рано или поздно должен достичь дальней стенки охватывающего прямоугольника.

Весь маршрут путника внутри охватывающего прямоугольника состоит из отрезков, вдоль которых он движется по компасу, и в ходе такого движения он, очевидно, приближается к дальней стенке, и из ломаных линий обхода препятствий. Следовательно, необходимо показать, что в ходе движения по линии обхода (вдоль стенок) он также приближается к удаленной стенке охватывающего прямоугольника.

Рассмотрим текущее препятствие, с которым столкнулся путник. Для него существует охватывающий прямоугольник, ориентированный так же, как охватывающий прямоугольник для всей системы препятствий. Это означает, что путник сталкивается со стенкой, дальней по отношению к удаленной стенке охватывающего прямоугольника, и это же означает, что он пойдет в отрыв от удаленной стенки текущего охватывающего прямоугольника, а это в свою очередь и означает, что, двигаясь по ломанной линии, он также приближается к точке назначения.

Рассуждения, записанные ранее, нельзя считать строгим доказательством, скорее это более менее убедительное обоснование, но если к этому обоснованию добавить некоторое количество сложных тестовых примеров, то можно быть уверенным, что полученное решение, если и не охватывает все возможные ситуации, то во всяком случае область его применения велика.

В заключение

Очень и очень часто неопытные программисты опускают обоснование своего алгоритма. Необходимость доказательства игнорируется по причинам, далеким от логики. Разработчик, придумавший красивый алгоритм, как правило, не готов согласиться с возможностью ошибки. Действительно красивый алгоритм должен быть верен, особенно если на него потрачено много сил. В иное не хочется верить, в этом все дело, но такие рассуждения далеки от логики. Сколько бы разработчик не потратил времени и сил, его работа вполне может оказаться ошибочной. Это нужно помнить, и если алгоритм сложен, его дополнительное обоснование, если не доказательство, никогда не будет лишним.