Поиск закономерности в случайных данных

Условие задачи.

Дана таблица положительных целых чисел, сформированная случайным образом. Необходимо найти путь из левого верхнего угла таблицы в правый нижний с наибольшим весом. Весом пути будем называть сумму чисел всех элементов таблицы, через которые проходит путь. Путь можно строить только двумя типами смещений: вправо на один шаг или вниз на один шаг.

Данная задача имеет решение в виде полного перебора. Достаточно придумать алгоритм построения всех путей и в процессе построений вес каждого полученного пути сравнивать с уже найденным максимальным. Это элементарный алгоритм поиска наибольшего. Если таблица невелика, то скорость работы алгоритма окажется вполне приемлема. Например, это будет вполне реально для таблицы 10?10, которую обрабатывает наша программа. Если размеры таблицы увеличить, то количество возможных путей начнет стремительно увеличиваться, и соответственно скорость работы программы стремительно падать.

В случае совершенно произвольного построения путей задача становится задачей полного перебора, а проблема быстрого роста вариантов не решается. Однако в нашей задаче есть существенное ограничение на способ построения путей. Попробуем провести анализ дополнительного условия. Может быть это даст возможность упрощения задачи.

Для хорошего метода нужна закономерность. Но наша таблица заполнена случайными числами. Искать закономерность на множестве случайных чисел — дело не благодарное. По определению случайного числа среди такого множества закономерности быть просто не должно.

Но можно попробовать построить дополнительную конструкцию, такую что закономерность будет видна явным образом.

Примечание

Грамотное построение дополнительных структур часто дает дополнительные возможности в организации логики. Этим подходом (или точнее приемом) мы уже пользовались. Например, в задаче о разбиении кучи камней на две было проведено упорядочивание множества камней в порядке возрастания. Введение порядка на множестве — это то же построение дополнительной конструкции. В задаче экономного обхода графа и задаче заливки контура мы применяли флаг, содержащий полезную информацию. Два упомянутых случая использовали дополнительные конструкции различного типа. Введение порядка меняет общую структуру множества, добавление флага создает возможность введения для каждого элемента характеристики, описывающей ход процесса.

Нам нужна структура данных, привязанная к элементу таблицы и в то же время описывающая путь до этого элемента. "Структура" в этом случае звучит, наверное, излишне громко, достаточно будет числовой характеристики, описывающей наибольший вес пути до данного элемента. Если удастся придумать такую характеристику, то, видимо, это будет шаг в верном направлении.

Хорошая зацепка для последующего анализа в предложении, записанном ранее — *достаточно будет числовой характеристики описывающей наибольший вес пути до данного элемента*. В этом предложении скрывается существенная неопределенность. Что есть *"путь до данного элемента"*? Если вы думаете, что имеется в виду путь от верхнего

левого элемента до текущего, то спросите себя, ПОЧЕМУ ТАК? Не кажется ли вам, что такое убеждение происходит лишь от того, что движение осуществляется сверху вниз и справа налево! Характер движения совсем не означает, что поиск пути необходимо осуществлять в том же порядке.

Примечание

Это достаточно обычная ситуация, когда привычный или более естественный ход вещей выдается за логичный. В поисках решения у вас всегда есть серьезный риск попасть в плен к своим психологическим установкам. Поэтому старайтесь к каждому выдвинутому утверждению задавать вопрос ПОЧЕМУ?

Проанализируем обе ситуации. Пусть поиск пути ведется сверху, и на некотором шаге путник, анализирующий различные маршруты, находится в некоторой заданной точке. Сейчас он должен принять решение о дальнейшем движении. В его распоряжении есть информация о весе, накопленном в движении сверху (для удобства рассуждений не будем постоянно повторять, что движение идет не только сверху вниз, но и слева направо). Информации о том, что будет у него дальше, нет, поэтому не видно логических оснований для выбора пути.

Пусть теперь поиск пути ведется снизу вверх. Путник опять находится в определенной точке. Ситуация совершенно иная. Путник предельно точно знает, какой вес он смог набрать, и теперь выбор между направлением вверх и направлением влево теряет смысл, так как ясно что к уже достигнутому весу необходимо добавить большее число, выбранное из левого или верхнего элемента. Такой у нас получился парадокс. Движение должно осуществляться сверху вниз, а поиск пути логичнее вести снизу вверх.

Продолжим анализ. Принципиально идея решения уже есть. Она заключается в следующей фразе "путник предельно точно знает, какой вес он смог набрать". Следовательно, мы должны для каждого элемента выяснить, какой вес можно набрать, двигаясь снизу до данного элемента. Теперь маленький интуитивный рывок и формулируем алгоритм.

Обойдя всю матрицу снизу вверх и справа налево, определим для каждого элемента новый вес, равный сумме собственного и наибольшего из верхнего и левого элемента. Алгоритм такой обработки:

```
Текущий элемент = Правый нижний
Пока текущий элемент не есть левый верхний, делать
  Если вес левого элемента больше веса верхнего
    То Текущий элемент = Левый элемент
    Иначе Текущий элемент = Верхний элемент
```

Искомый путь на построенной вспомогательной конструкции будет найден следующим алгоритмом:

```
Текущий элемент = Верхний левый.
Пока текущий элемент не есть Нижний правый, делать
  Если вес правого элемента больше веса нижнего,
  То Текущий элемент = Правый элемент
  Иначе Текущий элемент = Нижний элемент
```

Далее приведен пример такого построения и жирным шрифтом отмечен искомый путь. Для построения примера мы взяли клетку 5 на 5. Слева исходная таблица, справа преобразованная. Путь отмечен на преобразованной таблице.

1	2	8	3	5
3	2	2	1	45
5	1	1	0	4
2	3	4	3	8
1	1	3	4	8

1480	**816**	**419**	190	70
663	395	**221**	**117**	**65**
265	172	102	51	**20**
88	69	50	31	**16**
17	16	15	12	**8**

Листинг

```
program example;
 uses crt;
 var
  a:array[1..10,1..10] of longint;
  i,j:integer;
begin
 clrscr;
 randomize;
 for i:=1 to 10 do
  for j:=1 to 10 do
   begin
    a[i,j]:=random(9);
    gotoxy(i*7,j*4);write(a[i,j]);
   end;
 a[2,8]:=5360; gotoxy(2*7,8*4);write(a[2,8]);
 readkey;
 for i:=10 downto 1 do
  for j:=10 downto 1 do
    begin
     if i<10 then a[i,j]:=a[i,j]+a[i+1,j];
     if j<10 then a[i,j]:=a[i,j]+a[i,j+1];
     gotoxy(i*7,j*4);write(a[i,j]);
    end;
 readkey;
 i:=1;j:=1;
 textcolor(2);
 repeat
  gotoxy(i*7,j*4);write(a[i,j]);
  if (i=10) then j:=j+1
   else if (j=10) then i:=i+1
    else  if a[i+1,j]>a[i,j+1] then i:=i+1 else j:=j+1;
  delay(64000);
 until (i=10) and (j=10);
 gotoxy(i*7,j*4);write(a[i,j]);
 readkey;
end.
```

В заключение

Решенная задача — редкий пример того, как в переборной задаче удалось полностью избавиться от полного перебора. Такая масштабная удача говорит о том, что оценка данной задачи, как задачи полного перебора, была неверной. И действительно. Необходимость полного перебора следует из совершенного отсутствия информации об исходных данных. В рассмотренной задаче это не совсем так. Числа, которыми заполняется таблица, действительно случайные, но в построении путей полного произвола

нет, именно существующее ограничение на возможные пути и создало возможность построения красивой вспомогательной конструкции.