Рекурсивные процедуры и функции как инструмент развития алгоритмического мышления школьников.

Морозов В. В.

На наш взгляд, умение создавать грамотные рекурсивные процедуры и функции является необходимой и вместе с тем одной из самых труднопреодолимых степеней на пути к вершинам искусства программирования. И конечно, без четко поставленного умения писать рекурсивные процедуры и функции трудно ученику трудно добиться сколько-нибудь значимого успеха на олимпиадах по программированию.

Рекурсия – метод определения класса объектов или методов предварительным заданием одного или нескольких (обычно простых) его базовых случаев или методов, а затем заданием на их основе правила построения определяемого класса, ссылающегося прямо или косвенно на эти базовые случаи[1].

При подготовке к олимпиаде особую ценность представляют задачи, предполагающие использование рекурсивных процедур или функций для эффективного решения. Одной из таких задач является задача, предложенная на сайте «школа программиста»[2].

Компьютерная игра

(Время: 1 сек. Память: 16 Мб Сложность: 38%)

Вы можете вспомнить хоть одного своего знакомого до двадцатилетнего возраста, который в детстве не играл в компьютерные игры? Если да, то может быть вы и сами не знакомы с этим развлечением? Впрочем, трудностей при решении этой задачи это создать не должно.

Во многих старых играх с двумерной графикой можно столкнуться с подобной ситуацией. Какой-нибудь герой прыгает по платформам (или островкам), которые висят в воздухе. Он должен перебраться от одного края экрана до другого. При этом при прыжке с одной платформы на соседнюю, у героя уходит |y2-y1| единиц энергии, где y1 и y2 – высоты, на которых расположены эти платформы. Кроме того, у героя есть суперприем, который позволяет перескочить через платформу, но на это затрачивается 3*|y3-y1| единиц энергии. Конечно же, энергию следует расходовать максимально экономно.

Предположим, что вам известны координаты всех платформ в порядке от левого края до правого. Сможете ли вы найти, какое минимальное количество энергии потребуется герою, чтобы добраться с первой платформы до последней?

Входные данные

В первой строке входного файла INPUT.TXT записано количество платформ n ($1 \le n \le 30000$). Вторая строка содержит n натуральных чисел, не превосходящих 30000 – высоты, на которых располагаются платформы.

Выходные данные

[1] (Школа программирования, 2014)

[2] (Школа программиста, 2007)

В выходной файл OUTPUT.TXT запишите единственное число – минимальное количество энергии, которую должен потратить игрок на преодоление платформ (конечно же, в предположении, что cheat-коды использовать нельзя).

Пример

Таблица 1

№	INPUT.TXT	OUTPUT.TXT
1	3 1 5 10	9
2	3 1 5 2	3

Обсудим с учащимися стратегию решения поставленной задачи. Информацию о высоте платформ будем хранить в массиве. Для игрока способов пройти все платформы не мало: всякий раз можно делать обычные прыжки, а можно суперпрыжки. Чем больше платформ, тем больше вариантов поведения игрока. Нашей же программе нужно выбрать единственно верный способ движения игрока с наименьшей затратой энергии. Искусство педагога в этот момент в том, чтобы подтолкнуть ученика к самостоятельному выводу о необходимости писать рекурсивную процедуру, чтобы выбрать наиболее оптимальный вариант поведения игрока с целью наименьших затрат энергии.

Таблица 2

uses crt;
  const nn=30000;
  Type
TPlatforms=array[1..nn]of integer;
  var y:TPlatforms;
  n,i,p,E:integer;
	Создадим тип массив для хранения высот платформ. nn=30000 – максимально возможное количество платформ.

procedure move(p:integer; y:TPlatforms; n:integer; var E:integer);	Пишем процедуру move, которая в качестве формальных параметров получает номер текущей платформы, на которой находится игрок, массив y, в котором хранятся высоты всех платформ, n – номер последней платформы, E – изменяемая в процедуре величина – затраченная игроком энергия при выполнении очередного хода.
var e1,e2,p1,p2:integer; begin	Объявляем локальные переменные: e1, e2 – для хранения энергий, потраченных при разных вариантов прыжка. p1, p2 – номера платформ, на которых

	окажется игрок после совершения обычного прыжка и суперпрыжка соответственно.
`if p=n-1 // Игрок на предпоследней платформе then begin E:=abs(y[n]-y[p]); inc(p) end else`	Обращаем внимание учащихся еще раз на важную деталь реализации рекурсивных методов: всегда задаем граничное условие, то есть условие выхода из рекурсии. Лучше всего его указывать в самом начале[3]. В противном случае не избежать неприятного явления для программиста – зацикливания, и как следствие, зависания программы. В нашем случае граничное условие – ситуация, когда игрок достиг предпоследней или пред-предпоследней платформы. Если игрок на предпоследней платформе, то мы не вызываем процедуру рекурсивно, переменная E (затраченная энергия) для последнего обычного прыжка вычисляется и передается основной программе, номер текущей платформы принимает значение n, – этап игры пройдет полностью.
`if p=n-2 then begin // супер прыжок e1:=3*abs(y[n]-y[p]); e2:=abs(y[n]-y[n-1])+abs(y[n-1]-y[p]);`	Если же игрок находится на пред-предпоследней платформе, то перед ним две возможности – либо совершать суперпрыжок, либо совершать два обычных прыжка. Для каждого из вариантов поведения мы вычисляем необходимые затраты энергии, и…
`if e1<e2 then E:=e1 else E:=e2; p:=n end`	… и выбираем наименьшую затраченную энергию. Номер текущей платформы приравниваем к n – суперпрыжок или два обычных прыжка выполнены, этап игры полностью пройден. Рекурсивно процедуру в этих случаях не вызывали.
`else begin // Игрок далеко от финиша p1:=p+2; // суперпрыжок`	Если же последняя платформа еще далеко, то пробуем сделать и обычный прыжок, и супер-прыжок, при выполнении каждой из этих попыток вычисляем затраченные

[3] (Школа программирования, 2014)

p2:=p+1; // просто прыжок
//e1:=0;
move(p1,y,n,e1);
e1:=e1+3*abs(y[p1]-y[p]);
//e2:=0;
move(p2,y,n,e2);
e2:=e2+abs(y[p2]-y[p]);
//writeln(p1:4, p2:4, e1:4,e2:4);
	энергии в переменных e1 и e2, запоминаем при этом номера платформ p1 и p2, на которых игрок окажется после выполнения суперпрыжка и обычного прыжка соответственно.
	Для каждого варианта поведения пытаемся выполнить следующий ход начиная с платформ p1 и p2

if e1<e2
then
begin p:=p1; E:=E1 end
else
begin p:=p2; E:=E2 end
end
end;
	Ищем наименьшие затраты энергии после выполненных двух попыток. Та платформа, на которой затраты энергии минимальны – запоминается в переменной p.
	Рекурсивная процедура завершена.
	Поскольку после каждого вызова процедуры номер текущей платформы становится больше, обязательно наступит граничная ситуация, когда процедура не вызывает себя рекурсивно, стек не переполнится, зацикливания не произойдет.

begin
read(n);
for i:=1 to n do
read(y[i]);
	Читаем с консоли количество платформ и высоты платформ.

p:=1; E:=0;	Указываем, что текущая платформа – первая. Затраченная энергия пока равна нулю.
Move(p,y,n,E);	Пытаемся сделать оптимальный прыжок с первой платформы. Далее процедура, вызывая себя рекурсивно, находит путь с наименьшими затратами энергии.
Writeln(E) end.	Выводим оптимальную энергию на экран. Задача решена

Предметный указатель

Список таблиц

Список литературы

1. Школа программирования. (24 Март 2014 г.). *Рекурсия. Введение.* Получено 2 11 2014 г., из Школа Программирования: http://www.prog-school.ru/2014/03/rekursiya-vvedenie
2. Школа программиста. (8 1 2007 г.). *Компьютерная игра.* Получено 2 11 2014 г., из Школа программиста - портал для школьников: http://acmp.ru/index.asp?main=task&id_task=29