Место динамического программирования в подготовке школьников к олимпиадам по программированию

Морозов В. В.

Речь сегодня пойдет о переливаниях воды. Математики, посмеиваясь над собой, рассказывают байку об отличии мышления математика и физика. Математику и физику задали вопрос: как приготовить чай. Оба ответили одинаково: «Нужно налить в чайник воды, вскипятить воду в чайнике, заварить чай». Тогда математику и физику задали другой вопрос: а как приготовить чай, если вода в чайнике уже есть. На это раз ответы математика и физика отличались. Физик ответил: «Вот и хорошо, просто ставим чайник на плиту, кипятим воду и завариваем чай». Но математик, удивляя всех, ответил: «Выльем воду из чайника, и задача свелась к предыдущей, которую мы уже решили».

Конечно, в этом случае с решением математика можно поспорить. Действительно, ведь в решении математика появились лишние шаги. Но порой этот метод, свести задачу к более простой, уже решенной оказывается продуктивным, в том числе при решении олимпиадных задач по программированию. Этот метод называется «Динамическое программирование».

Википедия о динамическом программировании: *«Динамическое программирование в теории управления и теории вычислительных систем — способ решения сложных задач путём разбиения их на более простые подзадачи. Он применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной. В этом случае время вычислений, по сравнению с «наивными» методами, можно значительно сократить.*

Ключевая идея в динамическом программировании достаточно проста. Как правило, чтобы решить поставленную задачу, требуется решить отдельные части задачи (подзадачи), после чего объединить решения подзадач в одно общее решение. Часто многие из этих подзадач одинаковы. Подход динамического программирования состоит в том, чтобы решить каждую подзадачу только один раз, сократив тем самым количество вычислений. Это особенно полезно в случаях, когда число повторяющихся подзадач экспоненциально велико.

Метод динамического программирования сверху — это простое запоминание результатов решения тех подзадач, которые могут повторно встретиться в дальнейшем. Динамическое программирование снизу включает в себя переформулирование сложной задачи в виде рекурсивной последовательности более простых подзадач»[1].

Задачи на применение идей динамического программирования – частое явление на олимпиадах по программированию, поэтому одним из важных шагов подготовки учащихся к успешному выступлению на олимпиаде по программированию является обучение приемам динамического программирования. Вопросы применения динамического программирования подробно изложены в книге «Динамическое программирование», Беллман Р[2].

[1] (Динамическое программирование, 2014)
[2] (Беллман, 1960)

При подготовке к олимпиаде высокую ценность представляют собой задачи, предполагающие идеи динамического программирования. Одной из таких задач является задача, предложенная на сайте «школа программиста»[3].

Постановка задачи.

Имеется три ведра, емкости которых известны и не равны. Самое большое ведро полное, остальные пусты. Требуется добиться, чтобы в самом большом ведре был заданный объем воды. За один шаг вода переливается из одного ведра в другое до тех пор, пока либо не закончится вода в ведре-источнике, либо не наполнится доверху вода в ведре-получателе.

Школьник Василий, чтобы занять себя, пытается решать эту задачу с разными входными данными, но не всегда находит решение. И даже если решение найдено, он хочет знать, является ли найденное решение оптимальным, а именно, используется ли минимальное количество шагов. Требуется написать программу, которая поможет Василию проверить его решение.

Входные данные

Во входном файле INPUT.TXT записаны 4 числа: емкости ведер B1, B2, B3 ($1000 \geq B1 > B2 > B3 > 0$) и требуемое количество воды T в первом ведре ($B1 > T > 0$).

Выходные данные

В выходной файл OUTPUT.TXT выведите либо минимальное количество переливаний, либо если задача не имеет решения, то слово IMPOSSIBLE.

Примеры

Таблица 1

№	INPUT.TXT	OUTPUT.TXT
1	10 8 4 4	3
2	10 8 4 5	IMPOSSIBLE

Пояснения к примеру

Входные данные первого примера предлагают следующую конфигурацию сосудов: 10-литровое ведро полное воды, два пустых сосуда емкостью 8 л и 4 л. Последнее число входных данных – 4 л говорит нам, что в первом 10-литровом ведре нужно получить 4л воды. 4 л – это цель наших переливаний. И нужно не просто выяснить, можно ли получить 4 л за несколько переливаний из одного ведра в другое, но еще и сделать это за наименьшее количество переливаний.

Одним из способов показать шаги переливаний – таблица.

[3] (Школа программиста, 2007)

Таблица 2

Шаг	10-литровый сосуд	8-литровый сосуд	4-литровый сосуд
0	10	0	0
1	6	0	4
2	0	6	4
3	4	6	0

Действительно, на 3 шаге в 10-литровом сосуде удается получить 4 л воды.

Что касается второго примера входных данных, заметим, что все сосуды имеют четное количество литров, и после любого переливания в каждом сосуде может оказаться только четное количество литров воды, поэтому 5 л – нечетное число литров – недостижимый результат, в этом случае наша программа должна выдать сообщение «Impossible».

Обратим внимание на важную деталь, которая отличает эту задачу на переливание от других: воду переливаем только в имеющиеся сосуды, в раковину, в реку или на землю воду не выливаем. Отсюда вывод: сумма объемов всех сосудов всегда неизменна и равна объему первого сосуда.

Прежде чем начать обсуждение методов решения поставленной задачи, следует решить несколько задач на переливание. Такие задачи можно часто встретить на олимпиадах по математике и информатике в 5-7 классах, а на просторах Интернета имеется масса имитаторов для решения задач не переливание. Впрочем, высока вероятность того, что задачи на переливание уже хорошо знакомы учащимся и решать задачи на переливания не потребуется. Но все-таки решение нескольких задач на переливание может подтолкнуть учащихся к самостоятельному выводу о том, что для реализации алгоритма на компьютере следует применить рекурсивные процедуры, то есть процедуры, которые будут вызвать сами себя, но только для решения уже немного более простой задачи, которая на один шаг ближе к финальному результату. Таким образом, учащиеся делают самостоятельный выбор метода решения поставленной задачи, – это динамическое программирование.

Теперь обсудим идею решения задачи. Напишем процедуру, которая на вход в качестве формальных параметров получает некоторое состояние сосудов – их объемы и количество воды в них, пусть эта процедура пытается перелить воду из одного сосуда в другой. После этого нужно проверить новое состояние сосудов, не было ли получено такое состояние сосудов на одном из предыдущих этапов. Если такое состояние сосудов уже встречалось, то от такой попытки переливания отказываемся. Если мы этого не сделаем, то рискуем переливать одну и ту же воду из сосуда А в сосуд Б и обратно, процесс зациклится, и мы не сможем получить нужный результат. Следовательно, нам следует продумать механизм запоминания всех сделанных ранее состояний.

Выходить из этой процедуры будем в случае, если в первом сосуде окажется нужное количество воды или если любое переливание приводит сосуды в состояние, которое уже было раньше. В этом случае делаем вывод, что задача неразрешима.

Приступаем теперь к реализации описанной идеи.

Таблица 3

uses crt; Type TBaskets=array[1..3] of integer; TState=array[1..360,1..2] of integer;	Создадим тип TBaskets – для хранения объемов сосудов и количества воды в них. Тип TState – для хранения всех состояний сосудов, учитываются только первый и второй сосуды, поскольку количество воды в третьем сосуде запоминать не нужно, оно вычисляется как объем первого сосуда минус сумма объемов воды, находящихся в первом и втором сосудах
var b,w: TBaskets; LLL:Boolean; s:TState; kk,t,step,min:integer;	Объявляем глобальные переменные: b – массив для хранения объемов сосудов w – массив для хранения количества воды в сосудах LLL – логическая переменная – признак успешного решения задачи s – массив состояний, которые уже были получены на текущий момент kk – количество состояний, записанных в массиве s t – количество воды, которое нужно получить в первом сосуде step – переменная запоминает номер текущего переливания min – минимальное количество шагов, приводящих к результату
Procedure CheckState(w:TBaskets;s:TState; kk:integer; var LL:boolean);	Напишем вспомогательную процедуру CheckState, которая в качестве формальных параметров получает массив объемов воды, массив записанных ранее состояний, количество записанных состояний. Процедура возвращает в качестве результата

	логическую переменную LL. Если проверяемое состояние w уже встречалось, то переменная LL равна false, если такого состояние еще не было, то true.
`  var i:integer;`	Объявляем локальную переменную i в качестве индексной переменной

`begin`
`  LL:=false;`
`  for i:=1 to kk do`
`LL:=((w[1]=s[i,1])and(w[2]=s[i,2]))or (LL);`
`  LL:=not(LL);`
`end;`
	Тело процедуры.
	Перед просмотром массива состояний в переменную LL предварительно записываем false.
	Затем просматриваем массив состояний, и если хотя бы одно состояние, записанное ранее, совпадает с текущим, то LL принимает значение true. Перед выходом из процедуры делаем инверсию логической переменной LL.

`procedure spill(b:TBaskets; var w:TBaskets; var s:TState;`
`  var kk:integer;`
`  t:integer;`
`i,j:integer; var L:boolean; var step:integer);`
	Напишем вспомогательную процедуру Spill, которая будет пытаться перелить воду из одного сосуда в другой.
	В качестве формальных параметров процедура получает массив объемов сосудов b, массив w объемов воды в сосудах (этот параметр при выполнении процедуры может измениться, поэтому он идет со словом var), массив записанных ранее состояний s (этот параметр при выполнении процедуры может измениться, поэтому он идет со словом var), kk – количество записанных ранее состояний, t – количество воды, которое нужно получить в первом сосуде, i – номер сосуда, откуда переливается вода, j – номер сосуда, в который переливается вода, L – логическая переменная, которая

	будет принимать значение true, если переливание прошло успешно и нового состояния пока не было в списке прошлых состояний и значение false в противном случае, step – номер шага текущего шага.
`var v:integer;` `LL:boolean;`	Объявляем локальные переменные: v – количество воды, которое переливается LL – признак того, что переливание удачно

`begin`
`if i<>j`
`then`
`begin`
	Запрещаем переливать воду из сосуда в себя самого – ненужная операция

`if (w[i]>0)and(b[j]>w[j])`
`then`
`begin`
	Переливание допустимо только в случае, если сосуд, откуда переливаем воду, не пустой, а также если сосуд, куда переливаем воду, не заполнен до краев.

`v:=b[j]-w[j];`	Определяем количество пустого места в сосуде, куда пытаемся перелить воду.

`if w[i]<=v`
`then`
`begin`
`w[j]:=w[j]+w[i];`
`w[i]:=0;`
`inc(step);`
`L:=true`
`end`
	Если количество воды в исходном сосуде меньше или равно, чем количество пустого места в принимающем сосуде, то вода переливается полностью, исходный сосуд становится пустым, номер шага увеличивается на 1, флаг успешного переливания поднимаем.

`else`
`begin`
`w[i]:=w[i]-v;`
`w[j]:=b[j];`
`inc(step);`
`L:=true;`
`end;`
`end;`
	Если количество воды в исходном сосуде больше, чем количество пустого места в принимающем сосуде, то из исходного сосуда выливается только количество воды, которое может поместиться в пустом месте принимающего сосуда,

	принимающий сосуд заполняется до краев, номер шага увеличивается на 1, флаг успешного переливания поднимаем.
`CheckState(w,s,kk,LL);`	Проверяем, было ли текущее состояние в списке ранее имеющихся состояний.

`if not(LL)`
`then`
`L:=false`
	Если текущее состояние уже было в списке ранее записанных состояний, значит текущее переливание неудачное, и рассматривать его больше не нужно.

`else`
`begin`
`inc(kk);`
`s[kk,1]:=w[1];`
`s[kk,2]:=w[2];`
	Если текущее еще не появлялось в списке ранее записанных состояний, то увеличиваем количество уникальных состояний на 1, записываем удачное текущее состояние в массив состояний.

`if w[1]=t then` `begin`	Проверяем, привело ли текущее состояние к нужному количеству воды в первом сосуде. Если привело, то…

`if min=0`
`then`
`min:=step`
`else`
`if step<min`
`then`
`min:=step;`
`end;`
`end`
`end`
	Если глобальная переменная min=0 (это будет, если найденная удачная конфигурация сосудов обнаружена первый раз), то в переменную min записываем количество шагов, которые привели к нужному количеству воды в первом сосуде.
	Если же min отлично от нуля, значит найденная удачная конфигурация уже не первая, в этом случае применяем классический алгоритм поиска минимума среди количества шагов, приводящих к нужному количеству воды в первом сосуде.

`Else L:=false;` `end;`	В случае, если i=j, и поэтому переливать воду из сосуда в себя

	самого, то попытка переливания неудачна, и в переменную L записываем false. Процедура завершена.
`Procedure check(b,w:TBaskets; s: TState;kk:integer; t:integer; LLL:boolean; step:integer);`	Напишем рекурсивную процедуру, которая как раз и реализует идею динамического программирования. Процедура Check будет пытаться сделать все возможные переливания, принимать во внимание будет только удачные переливания. И если переливание удачно, то к новому состоянию процедура будет применять себя рекурсивно. В качестве формальных параметров процедура получает на вход объемы сосудов b, текущее количество w воды в них , массив предыдущих состояний s, количество состояний kk, записанных в массиве состояний , количество воды t, которое нужно получить в первом сосуде, LLL – логическая переменная, признак того, что задача пока не решена и не все состояния изучены, step – номер текущего шага.
`var SStep:integer; ww:TBaskets; L1, L2, L3, L4, L5, L6:boolean;`	Объявляем локальные переменные: Sstep – резервная копия переменной Step. Необходима, чтобы откатить ситуацию к предыдущему удачному состоянию. ww – резервная копия переменной w. Необходима, чтобы откатить ситуацию к предыдущему удачному состоянию. L1, …, L6 – логические переменные, показывающие, насколько удачным было очередное переливание.

begin
  ww:=w;
  SStep:=step;
	Тело процедуры.
	Запоминаем резервные копии состояния воды в сосудах и номер текущего шага.

if (w[1]<>t)
  then
  begin
	Необходимо донести до учащихся важную мысль, что при написании рекурсивной процедуры необходимо предусмотреть случай, когда процедура не будет себя вызывать рекурсивно. Этот случай – ситуация, когда достигнуто нужное количество воды в первом сосуде.
	Если в первом сосуде пока не достигнуто нужное количество воды, то…

spill(b,w,s,kk,t,1,2,L1,step);
  if L1 then
check(b,w,s,kk,t,L1,step);
  w:=ww; Step:=SStep;
	Пробуем перелить воду из 1 сосуда во 2. Если переливание удачно, то пробуем совершать переливания дальше, вызывая процедуру Check рекурсивно. Результат проверки записываем в логическую переменную L1.
	Возвращаем сохраненные ранее резервные копии обратно, чтобы попробовать сделать другое переливание.

spill(b,w,s,kk,t,2,1,L2,step);
  if L2 then
check(b,w,s,kk,t,L2,step);
  w:=ww; Step:=SStep;
	Пробуем перелить воду из 2 сосуда в 1. Если переливание удачно, то пробуем совершать переливания дальше, вызывая процедуру Check рекурсивно. Результат проверки записываем в логическую переменную L2.
	Возвращаем сохраненные ранее резервные копии обратно, чтобы попробовать сделать другое переливание.

spill(b,w,s,kk,t,2,3,L3,step);
  if L3 then
check(b,w,s,kk,t,L3,step);
  w:=ww; Step:=SStep;
	Пробуем перелить воду из 2 сосуда в 3. Если переливание удачно, то пробуем совершать переливания дальше, вызывая процедуру Check рекурсивно.

	Результат проверки записываем в логическую переменную L3. Возвращаем сохраненные ранее резервные копии обратно, чтобы попробовать сделать другое переливание.

spill(b,w,s,kk,t,3,2,L4,step);
if L4 then
check(b,w,s,kk,t,L4,step);
w:=ww; Step:=SStep;
	Пробуем перелить воду из 3 сосуда во 2. Если переливание удачно, то пробуем совершать переливания дальше, вызывая процедуру Check рекурсивно. Результат проверки записываем в логическую переменную L4.
	Возвращаем сохраненные ранее резервные копии обратно, чтобы попробовать сделать другое переливание.

spill(b,w,s,kk,t,1,3,L5,step);
if L5 then
check(b,w,s,kk,t,L5,step);
w:=ww; Step:=SStep;
	Пробуем перелить воду из 1 сосуда в 3. Если переливание удачно, то пробуем совершать переливания дальше, вызывая процедуру Check рекурсивно. Результат проверки записываем в логическую переменную L5.
	Возвращаем сохраненные ранее резервные копии обратно, чтобы попробовать сделать другое переливание.

spill(b,w,s,kk,t,3,1,L6,step);
if L6 then
check(b,w,s,kk,t,L6,step);
	Пробуем перелить воду из 3 сосуда в 1. Если переливание удачно, то пробуем совершать переливания дальше, вызывая процедуру Check рекурсивно. Результат проверки записываем в логическую переменную L6.
	Возвращаем сохраненные ранее резервные копии обратно, чтобы попробовать сделать другое переливание.

LLL:=L1 or L2 or L3 or L4 or L5 or L6; end end;	Если все 6 попыток переливания были неудачны, то переменная LLL становится равной False. В этом случае задача неразрешима. Процедура закончена.

begin
  b[1]:=100; b[2]:=99; b[3]:=1;
t:=73;
	Тело программы.
	Задаем входные данные. Делаем это «в ручном режиме», поскольку чтение (и запись) данных их файла не является предметом рассмотрения настоящей статьи.
	Первый сосуд – 100 л, второй сосуд – 99 л. Третий сосуд – 1 л.
	Необходимо получить в первом сосуде 73 л.

w[1]:=b[1]; w[2]:=0; w[3]:=0;
  kk:=0; LLL:=true;
  step:=0; min:=0;
	Первый сосуд наполнен до краев, второй и третий сосуды пусты. Флаг LLL – задача пока не решена.
	Step=0 – пока не сделано ни одного переливания.
	Min=0 – нужное количество воды в первом сосуде пока не достигнуто.

Check(b,w,s,kk,t,LLL,step);	Запускаем процедуру поиска удачных переливаний. Именно она реализует динамическое программирование.

if min=0
  then
  writeln('Impossible')
  else writeln(min);
end.
	Если min=0, то нужное количество воды в первом сосуде не достигнуто, в то время как все состояния сосудов уже имели место, и никакие переливания больше не приводят к новым состояниям сосудов.
	Если же, напротив, min отличен от нуля, значит значение переменной min как раз и равно наименьшему количеству переливаний, которые приводят к нужному количеству воды в первом сосуде.
	Задача решена.

Возможные проблемы при выполнении алгоритма: если объем сосудов достаточно велик, и переливаний приходится делать много, то из-за большого количества вложенных рекурсивных процедур иногда наблюдается переполнение стека. Изменить эту неприятную ситуацию можно, предложив учащимся самостоятельно изменить

алгоритм, передавая в качестве параметра не массив состояний, а указатель на него. Это значительно уменьшит риск переполнения стека.

К сожалению, программа всего лишь определяет, возможно ли в принципе получить нужное количество воды в первом сосуде, и если да, то находит минимальное количество переливаний для этого. В ворохе всех шагов, конечно, есть шаги, которые привели к нужному количеству воды в первом сосуде, но программа пока совершенно не в состоянии напечатать именно те переливания, которые наиболее коротким путем приводят к нужному результату. Поэтому предлагаем учащимся доработать эту программу так, чтобы она выводила еще и все шаги, которые приводят к успешному решению задачи наиболее коротким способом.

Список Таблиц

Ключевые слова

Список литературы

1. Беллман, Р. (1960). *Динамическое программирование.* Москва: Издательство иностранной литературы.

2. *Динамическое программирование.* (20 09 2014 г.). Получено 9 11 2014 г., из Википедия: https://ru.wikipedia.org/wiki/%C4%E8%ED%E0%EC%E8%F7%E5%F1%EA%EE%E5_%EF%F0%EE%E3%F0%E0%EC%EC%E8%F0%EE%E2%E0%ED%E8%E5

3. Школа программиста. (8 1 2007 г.). *Вода.* Получено 8 11 2014 г., из Школа программиста: http://acmp.ru/index.asp?main=task&id_task=308