УДК 519.6 + 519.7

Н. В. Шилов

Институт систем информатики им. А. П. Ершова СО РАН
пр. Акад. Лаврентьева, 6, Новосибирск, 630090, Россия

Новосибирский государственный университет
ул. Пирогова, 2, Новосибирск, 630090, Россия

Новосибирский государственный технический университет
пр. Карла Маркса, 20, Новосибирск, 630092, Россия

E-mail: shilov@iis.nsk.su

ТРИ ЛИКА ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ [*]

Работа посвящена некоторым методическим аспектам динамического программирования и решению одной олимпиадной задачи с использованием динамического программирования. Новым является интерпретация метода восходящего динамического программирования в терминах вычисления наименьшей неподвижной точки монотонных функционалов. Такая трактовка позволяет единообразно решать как классические оптимизационные задачи, так и задачи, в которых оптимизационная составляющая не просматривается явно (например, задача синтаксического анализа контекстно-свободных языков алгоритмом Коукера – Янгера – Касами). Она же позволяет унифицировать общую схему динамического программирования в виде шаблона проектирования алгоритмов.

Ключевые слова: динамическое программирование, рекурсивное нисходящее динамическое программирование, мемоизация рекурсивных программ, итеративное восходящее динамическое программирование, теорема о неподвижной точке Тарского – Кнастера, решение конечных игр, синтаксический анализ контекстно-свободных языков.

Введение

О бросании кирпичей с высокой башни. Рассказывают, что Галилео Галилей открыл закон равноускоренного движения при свободном падении тел опытным путем, сбрасывая разные предметы со знаменитой Падающей башни в итальянском городе Пиза. Но вот недавно мой коллега по институту за чашкой чая предложил мне следующую головоломку о бросании кирпичей с высокой башни.

1. Марка кирпичей по прочности – это максимальная высота h (в метрах), при падении с которой кирпич не разбивается (но при падении с высоты $(h + 1)$ метров разбивается). Надо определить марку кирпичей опытным путем, сбрасывая их с разных уровней башни высотой в сто метров. Считается, что: уровни в башне следуют через каждый метр; если кирпич не разбивается и при падении с 100-го уровня, марку кирпича принимают за 100; прочность кирпича не изменится, если он не разбился при падении. Сколько бросаний вам для этого

[*] В основу работы положен мастер-класс «Заметки о парадигмах программирования», проведенный автором 5 ноября 2010 г. в рамках школы-семинара «Искусство программирования» на очном туре XI Открытой Всесибирской олимпиады по программированию им. И. В. Поттосина, и одноименный цикл лекций автора на Зимней школе НГУ-Parallels «Теория и практика программирования», которая прошла с 31 января по 5 февраля 2011 г. в Новосибирском государственном университете. Автор выражает свою благодарность Евгению Викторовичу Бодину, регулярно снабжающему его новыми головоломками за чаепитием, и Владиславу Евгеньевичу Кузькову за изящное решение, изложенное в последней части этой статьи.

достаточно, если у вас только два кирпича? Какое число бросаний является минимальным для определения марки кирпича?

Головоломка хороша не только для двух сотрудников лаборатории теоретического программирования. Для ее решения нужно придумать алгоритм (т. е. определить, с каких этажей и в какой последовательности бросать) и доказать, что этот алгоритм оптимален (меньшим числом бросаний обойтись нельзя). Если сделать башню пониже, скажем, высотой только в 10 метров, ее можно предложить школьникам 5–6-х классов для решения, например, подбором. Головоломку со стометровой башней можно предложить старшеклассникам: найти решение и доказать, что оно оптимально. И, наконец, эта головоломка может быть обобщена и представлена в одном из двух следующих вариантов.

2. Каково минимальное число бросаний N_H, достаточное в любом случае для определения марки кирпичей в башне высотой H метров, если у вас есть только два кирпича?

3. Каково минимальное число бросаний $M_{H,B}$, достаточное в любом случае для определения марки кирпичей в башне высотой H метров, если у нас есть только B кирпичей?

Легко видеть, что для решения исходной головоломки надо вычислить значение N_{100}, а $N_H = M_{H,2}$ для любого значения H. Теперь весь вопрос – как научиться вычислять значения $M_{H,B}$ для произвольных натуральных значений H и B.

Задача для самостоятельного решения: решите исходную головоломку 1 и задачу 2.

Мы начнем знакомство с динамическим программированием с решения именно задачи 3 [1], и используем ее функциональное решение [2] как вариант «*gentle introduction*» в область динамического программирования оптимизационных задач. Затем рассмотрим, как повысить эффективность функционального решения при помощи так называемой мемоизации – техники исполнения функциональных алгоритмов с использованием памяти. Далее перейдем от использования памяти в функциональном алгоритме к императивному итеративному алгоритму решения той же задачи и, главное, предложим трактовку динамического программирования как вычисления наименьшей неподвижной точки монотонного функционала. Такая трактовка позволит нам предложить некоторый унифицированный шаблон.

Первый лик динамического программирования: рекурсивный метод решения оптимизационных задач

Задачи 2 и 3 – это оптимизационные задачи, т. е. задачи на вычисление оптимальных значений. Метод динамического программирования был разработан как раз для решения оптимизационных задач посредством рекурсивного сведения поиска оптимального решения к поэтапному поиску оптимальных решений: план или программа действий, приводящая к оптимальному результату и состоящая из нескольких шагов, остается оптимальной на любом своем шаге.

Для примера разберем задачу 2. Переформулируем ее следующим эквивалентным образом: каково минимальное число бросаний N_H, достаточное в любом случае для определения марки кирпичей в башне, с которой сбрасывать кирпичи можно с H уровней, если у нас есть только два кирпича?

Оптимальный (по количеству бросаний) метод начинается с некоторого шага: «сбросить первый кирпич с уровня h...». Предположим, мы как-то выбрали h, тогда верно равенство

$$N_H = 1 + \max\{(h-1), N_{(H-h)}\},$$

правая часть которого имеет следующий смысл:

1) «плюс 1» соответствует первому бросанию;

---

[1] Когда эта статья уже была готова, Теодор Ясонович Заркуа, профессор Грузинского университета им. Святого Андрея Первозванного Патриаршества Грузии, сообщил автору, что эта задача хорошо известна и активно используется в олимпиадном программировании (например, http://acm.timus.ru/problem.aspx?space=1&num=1223).

[2] Именно «функциональное», а не «рекурсивное» решение, так как здесь речь идет о решении в функциональной парадигме, а не императивной. Подробно разница между парадигмами программирования обсуждена в [1] и научно-популярной статье для школьников [2].

2) $(h - 1)$ соответствует случаю, когда после первого бросания кирпич разбился и у нас остался единственный кирпич, который надо последовательно бросать с уровней 1, 2, … $(h - 1)$;

3) $N_{(H-h)}$ соответствует случаю, когда после первого бросания кирпич остался цел, и нам осталось определить марку этой пары кирпичей сбрасыванием их с $(H - h)$ уровней $[(h + 1) \ldots H]$;

4) «max» выбирает худший из случаев, описанных в 2 и 3.

Как оптимальным образом выбрать h, мы пока не знаем. Но нас интересует оптимальный метод, дающий минимальное число бросаний, и поэтому мы можем выписать следующее равенство:

$$N_H = \min_{1 \le h \le H} (1 + \max\{(h-1), N_{(H-h)}\}) = 1 + \min_{1 \le h \le H} \max\{(h-1), N_{(H-h)}\}.$$

Кроме того, можно выписать еще одно очевидное равенство $N_0 = 0$.

Заметим, что последовательность натуральных чисел $N_0, N_1, \ldots N_H, \ldots$, которые удовлетворяют этим двум равенствам, единственна. Действительно, N_0 определено, N_1 определяется только через N_0; N_2, – только через N_0 и N_1; N_H – только через $N_0, N_1, \ldots N_{(H-1)}$.

Следующий шаг – переход от последовательности чисел $N_0, N_1, \ldots N_H, \ldots$, которые удовлетворяют этим двум равенствам, к функции N: $N \to N$, которая является решением системы функциональных уравнений

$$N(0) = 0,$$
$$N(H) = 1 + \min_{1 \le h \le H} \max\{(h-1), N(H-h)\}.$$

Как следует из рассуждения о единственности последовательности чисел $N_0, N_1, \ldots N_H, \ldots$, данная система уравнений имеет единственное решение N: $H \mid\to N_H$.

Осталось сделать последний шаг – заметить, что эта система функциональных уравнений является определением рекурсивной функции, т. е. рекурсивным алгоритмом. Это и есть (исторически) первый лик динамического программирования: рекурсивный метод решения оптимизационных задач.

В такой форме динамическое программирование было введено Ричардом Беллманом в 1950-х гг., он же предложил сам термин «динамическое программирование» [3; 4]. Но этот термин имеет мало общего с программированием для ЭВМ, которое только зарождалось 1950-е гг. Существительное «программирование» тогда имело значение «планирование» (например, в названии «линейное программирование»). А прилагательное «динамическое» указывает на смену состояний, как, например, в названиях «динамические системы» или «динамическая логика».

Имя Р. Беллмана сохранено в названиях «принцип оптимальности Беллмана» и «уравнение Беллмана». Этот принцип фактически уже был сформулирован в начале этого раздела: план, программа действий, приводящая к оптимальному результату и состоящая из нескольких шагов, остается оптимальной на любом своем шаге, в частности после первого шага. А уравнение Беллмана – это рекурсивное уравнение для целевой функции (значения которой оптимизируются), выражающее значение целевой функции перед исполнением очередного шага через ее значения после исполнения этого шага. Например, в случае задачи об оптимальном числе бросаний двух кирпичей это уже знакомое нам уравнение

$$N(H) = 1 + \min_{1 \le h \le H} \max\{(h-1), N(H-h)\}.$$

Задача для самостоятельного решения: выпишите уравнение Беллмана для функции M: $N \times N \to N$, M: $(H, B) \mid\to M_{H,B}$, которая вычисляет числа $M_{0,0}, M_{1,0}, \ldots M_{H,B}, \ldots$

Второй лик динамического программирования: рекурсия + мемоизация

Давайте попробуем вычислить одно значение функции N, например, $N(4)$, в полном соответствии с ее рекурсивным определением в леворекурсивном порядке (т. е. всегда первым вычисляется самый левый самый внутренний вызов):

$$N(4) = 1 + \min_{1 \le h \le 4} \max\{(h-1), N(4-h)\} =$$
$$= 1 + \min\{\max\{0, N(3)\}, \max\{1, N(2)\}, \max\{2, N(1)\}, \max\{3, N(0)\}\} =$$
$$= 1 + \min\{\max\{0, 1 + \min\{\max\{0, N(2)\}, \max\{1, N(1)\}, \max\{2, N(0)\}\}\}\},$$

$$\max\{1, N(2)\}, \max\{2, N(1)\}, \max\{3, N(0)\}\} =$$
$$= 1 + \min\{\max\{0, 1 + \min\{\max\{0, 1 + \min\{\max\{0, N(1)\}, \max\{1, N(0)\}\}\},$$
$$\max\{1, N(1)\}, \max\{2, N(0)\}\}\},$$
$$\max\{1, N(2)\}, \max\{2, N(1)\}, \max\{3, N(0)\}\} = \ldots = 3.$$

Уже на представленной части вычислений можно заметить, что многократно приходится вычислять значения функции N в одних и тех же «точках» (т. е. при одних и тех же значениях аргументов): в частности $N(2)$ и $N(1)$ вычислялись многократно.

Возможный вариант оптимизации состоит в том, чтобы вычислять только новые [3] вызовы функции N, сохранять их значения в памяти (в виде списка или хеш-таблицы), а значения повторных [4] вызовов функции N уже брать из памяти. Эта техника хорошо известна в функциональном программировании как мемоизация (*memoization*) [5].

Предположим, можно без исполнения вызова функции (т. е. статически) предсказать (или «угадать») конечное множество всех «нужных точек», которые могут возникнуть во время вычислений целевой функции в интересующей нас «целевой точке». Тогда можно применить метод «восходящего динамического программирования», состоящий в следующем.

1. Вычислить и сохранить множество значений функции во всех нужных «тривиальных» точках. Например, в случае функции N интересующее нас значение – это $N(H)$, а значение в тривиальной точке – это $N(0)$ в точке 0, его можно сохранить в ячейке $N[0]$ массива int array $N\,[0..H]$, где H – заданная высота башни.

2. Пополнить множество уже вычисленных значений функции значениями в новых «промежуточных» точках, которые можно вычислить непосредственно по уже сохраненным значениям функции в точках. Например, в случае функции N, если множество уже вычисленных значений $N(0)$ в точке 0, … $N(K)$ в точке K ($0 \le K < H$) записано в элементах массива $N[0]$, … $N[K]$, то множество вычисленных значений функции N можно пополнить значением $N(K + 1)$ в точке $(K + 1)$ посредством простого присваивания

$$N[K + 1] := 1 + \min_{1 \le k \le K} \max\{(k - 1), N[K - k]\}.$$

3. Повторять шаг 2, пока не будет вычислено значение целевой функции в нужной «целевой» точке. В случае задачи 1, которую мы обсуждаем, предыдущий шаг надо выполнить H раз, когда в элемент массива $N[H]$ будет записано значение $N(H)$ в точке H.

Заметим, что метод восходящего динамического программирования уже не рекурсивный, а итеративный.

Задача для самостоятельного решения: запрограммируйте метод восходящего динамического программирования для вычисления числа $M_{H,B}$ для заданных значений H и B.

Третий лик динамического программирования: вычисление неподвижной точки монотонного функционала

Попробуем формализовать описание итеративного метода восходящего динамического программирования. Для этого понадобятся операции на множествах, итеративный псевдокод для представления алгоритма и так называемые «условия частичной корректности» для спецификации алгоритма.

Условия частичной корректности [6; 7] схематически записываются в виде $\{B\}A\{C\}$, где A – это алгоритм; B – «предусловие» на входные данные; C – «постусловие» на результаты работы алгоритма; другое известное название для условий частичной корректности – тройки Хоара. Говорят, что тройка $\{B\}A\{C\}$ истинна (или что алгоритм частично корректен по отношению к предусловию и постусловию), если на любых входных данных, которые удовлетворяют свойству B, алгоритм A или не останавливается (зацикливается, зависает и т. п.), или останавливается с выходными результатами, которые удовлетворяют свойству C.

[3] Для тех значений аргументов, для которых ранее функция не вычислялась.
[4] Для тех значений аргументов, для которых функция уже была вычислена ранее.

Наш вариант формализации метода восходящего динамического программирования:

{D – непустое множество, 2^D – множество всех подмножеств D;
S и P – «тривиальное» и «целевое» подмножества D;
F: $2^D \to 2^D$ – всюду определенный [5] монотонный [6] функционал [7]}\\\ Предусловие.
var X, Y : subsets of D;
X:= S; repeat Y:= X; X:= $F(X)$ until ($P \cap X \neq \varnothing$ or $X = Y$).
{$P \cap X \neq \varnothing \Leftrightarrow P \cap T \neq \varnothing$,
где T – это наименьшее [8] подмножество D такое, что $S \subseteq T$ и $F(T) = T$}\\\ Постусловие.

В этой формализации присваивание «X:= S» соответствует шагу 1 метода, присваивание «X:= $F(X)$» в теле цикла repeat – until соответствует шагу 2, а часть условия выхода из цикла «$P \cap X \neq \varnothing$» – проверке условия на шаге 3 метода. Дополнительная переменная Y, присваивание «Y:= X» и проверка «$X = Y$» нужны для завершения работы алгоритма в случае, если мы не угадали множество нужных «тривиальных» точек.

В случае задачи об оптимальном числе бросаний двух кирпичей в башне высоты H имеем:

- D – это множество всех пар натуральных чисел (m, n), где m соответствует числу уровней, среди которых надо определить марку кирпичей, а n – возможному числу бросаний;
- S = {(0, 0)} – множество, состоящее из единственной «тривиальной» точки, а P = {(H, n) : n – оптимальное число бросаний в башне с H уровнями} – целевое множество, состоящее из единственной «целевой» точки;
- F – отображение, которое сопоставляет множеству пар $X \subseteq D$ новое множество пар $F(X)$ = {(m, n)$\in D$ | существуют m таких чисел n_0, … n_{m-1}, что (0, n_0), … (m – 1, n_{m-1}) $\in X$ и $n = 1 + \min_{1 \le k \le m} \max\{(k-1), n_{m-k}\}$}.

Может показаться, что при такой интерпретации невозможно проверить условие $P \cap X \neq \varnothing$ в силу неконструктивности определения множества P. Однако это не так: условие $P \cap X \neq \varnothing$ эквивалентно проверяемому условию $\exists n \in [0..H]$: (H, m) $\in X$.

В этой части мы докажем частичную корректность специфицированного алгоритма и его завершаемость для конечных множеств, а в следующей обсудим примеры использования такой интерпретации динамического программирования.

Частичная корректность алгоритма следует из теоремы Кнастера – Тарского о неподвижной точке [9]. Мы не будем приводить ни полную формулировку этой теоремы, ни ее обобщение (принадлежащее Тарскому); ограничимся формулировкой следствия из этой теоремы без доказательства.

Следствие (из теоремы Кнастера – Тарского о неподвижной точке [8]). Пусть D – непустое множество, а G: $2^D \to 2^D$ – всюду определенный монотонный функционал. Тогда существует наименьшее подмножество $T \subseteq D$ такое, что $G(T) = T$.

Утверждение 1. Пусть D – непустое множество, G: $2^D \to 2^D$ – всюду определенный монотонный функционал, $T \subseteq D$ – его наименьшая неподвижная точка, а R_0, R_1, … – последовательность подмножеств D, определенная по следующему правилу: $R_0 = \varnothing$ и $R_{k+1} = G(R_k)$ для любого $k \ge 0$. Тогда эта последовательность – монотонно неубывающая и содержится в T: $R_0 \subseteq R_1 \subseteq R_2 \subseteq \ldots R_k \subseteq R_{k+1} \subseteq \ldots \subseteq$ T.

Доказательство. Так как $R_0 = \varnothing$, то $R_0 \subseteq R_1$. В силу монотонности F, $R_1 = G(R_0) \subseteq G(R_1) = R_2$; продолжая этот процесс, получаем $R_k = G(R_{k-1}) \subseteq G(R_k) = R_{k+1}$ для любого $k > 0$. По аналогичным соображениям для любой неподвижной точки $W \subseteq D$ функции G имеют место соотношения $R_0 \subseteq W$ и $R_k = G(R_{k-1}) \subseteq G(W) = W$ для любого $k > 0$. В частности, для наименьшей неподвижной точки $T \subseteq D$ функции G имеют место соотношения $R_k \subseteq T$ для любого $k \ge 0$. ■

[5] Определенный на всех подмножествах D.
[6] Такой, что для любых $S_1, S_2 \subseteq D$, если $S_1 \subseteq S_2$, то $F(S_1) \subseteq F(S_2)$.
[7] Функция или отображение, у которой аргументами являются множества.
[8] Содержащееся в любом другом $R \subseteq D$ таком, что $F(R) = R$.
[9] Неподвижной точкой какой-либо функции f: $U \to U$ называется любое решение уравнения $f(x) = x$.

Утверждение 2. Формализованный алгоритм восходящего динамического программирования частично корректен по отношению к своим предусловию и постусловиию.

Доказательство. Предположим, что предусловие алгоритма выполнено и алгоритм завершает работу. Тогда в силу следствия из теоремы Кнастера – Тарского о неподвижной точке множество T – это наименьшая неподвижная точка отображения $G: X \mid\to S \cup F(X)$. Теперь заметим, что для любого $k > 0$ значение переменной X после k-й итерации цикла равно R_{k+1}, значение Y равно R_k, и, в силу утверждения 1, $R_k \subseteq T$. Поэтому если завершение цикла repeat – until произошло по условию $P \cap X \neq \varnothing$, то $P \cap T \neq \varnothing$. В противном случае (если $P \cap X = \varnothing$) завершение цикла произошло по условию $X = Y$, т. е. значение переменной X равно наименьшей неподвижной точке T, и поэтому $P \cap T = \varnothing$. Таким образом, постусловие выполнено. ■

Утверждение 3. Если выполнено предусловие формализованного алгоритма восходящего динамического программирования и, кроме того, множество D конечно, то алгоритм завершает свою работу не более чем за $|D|$ итераций цикла repeat – until.

Доказательство. Будем использовать обозначения, введенные в утверждениях 1 и 2. Как было показано в доказательстве утверждения 1, множества

$$R_0 \subseteq R_1 \subseteq R_2 \subseteq \ldots R_k \subseteq R_{k+1} \subseteq \ldots \subseteq \mathrm{T} \subseteq D$$

образуют неубывающую цепь. Если множество D конечно, то различными в ней могут быть только первых $|D|$ множеств, т. е. $R_k = \mathrm{T}$ начиная с $k = |D|$ (возможно раньше). Но, как было замечено в доказательстве утверждения 2, для любого $k > 0$ значение переменной X после k-й итерации цикла равно R_{k+1}, значение Y равно R_k. Следовательно, цикл repeat – until выполнит не более чем $|D|$ итераций. ■

Примеры неподвижной точки монотонного функционала

Кратко рассмотрим два алгоритма, которые получаются в результате конкретизации формализации метода восходящего динамического программирования.

Решение конечных позиционных игр. Теория игр чрезвычайно разнообразна [9]; мы же займемся только конечными позиционными играми двух игроков «A» (Алисы) и «B» (Боба). Такая игра – это шестерка $G = (P_A, P_B, M_A, M_B, F_A, F_B)$, где

- P_A и P_B – непересекающиеся конечные множества «позиций» для Алисы и Боба;
- $M_A \subseteq P_A \times (P_A \cup P_B)$ и $M_B \subseteq P_B \times (P_A \cup P_B)$ – множества «ходов» для Алисы и Боба;
- $F_A \subseteq (P_A \cup P_B)$ и $F_B \subseteq (P_A \cup P_B)$ – непересекающиеся множества «выигрышных позиций» для Алисы и Боба.

Можно сказать, что G – это ориентированный граф, все вершины которого «поделены» между Алисой и Бобом, и, кроме того, для Алисы и Боба выделены выигрышные вершины. Сессия игры G – это конечная или бесконечная последовательность позиций $p_0, \ldots p_k, \ldots$, в которой каждая соседняя пара позиций (p_k, p_{k+1}) – ход (Алисы или Боба). Партия – это или бесконечная сессия G, в которой не встречается выигрышных позиций (ни Алисы, ни Боба), или конечная сессия, содержащая только одну выигрышную позицию в качестве своего последнего члена. Конечная партия выиграна игроком $X \in \{A, B\}$, если партия заканчивается выигрышной позицией этого игрока (из F_A для Алисы и из F_B для Боба). Стратегия игрока $X \in \{A, B\}$ – это произвольное подмножество его ходов (т. е. стратегия Алисы $S \subseteq M_A$, стратегия Боба $S \subseteq M_B$). Стратегия игрока $X \in \{A, B\}$ называется выигрышной, если любая партия, в которой игрок использует только эту стратегию, выиграна этим игроком. Решить игру G – значит вычислить множества позиций партий W_A и W_B, в которых Алиса и Боб соответственно имеют выигрышные стратегии.

Решить игру G можно следующим образом. Легко видеть, что W_A и W_B – это наименьшие множества позиций, для которых верны следующие равенства:

- $W_A = F_A \cup \{p \in P_A \backslash F_B \mid \exists p' : (p, p') \in M_A \,\&\, p' \in W_A\} \cup \{p \in P_B \backslash F_B \mid \forall p' : (p, p') \in M_B \,\&\, p' \in W_A\}$,
- $W_B = F_B \cup \{p \in P_B \backslash F_A \mid \exists p' : (p, p') \in M_B \,\&\, p' \in W_B\} \cup \{p \in P_A \backslash F_A \mid \forall p' : (p, p') \in M_A \,\&\, p' \in W_B\}$.

Заметим, что следующие две функции

- $\mathrm{Win}_A : X \mid\to \{p \in P_A \backslash F_B \mid \exists p' : (p, p') \in M_A \,\&\, p' \in X\} \cup \{p \in P_B \backslash F_B \mid \forall p' : (p, p') \in M_B \,\&\, p' \in X\}$,
- $\mathrm{Win}_B : X \mid\to \{p \in P_B \backslash F_A \mid \exists p' : (p, p') \in M_B \,\&\, p' \in X\} \cup \{p \in P_A \backslash F_A \mid \forall p' : (p, p') \in M_A \,\&\, p' \in X\}$

являются всюду определенными монотонными функционалами на $(P_A \cup P_B)$. Поэтому множества W_A и W_B можно вычислить с использованием формализованного метода восходящего динамического программирования, если принять соответственно

- F_A в качестве S, $W\text{in}_A$ в качестве F, и $\varnothing$ – в качестве P;
- F_B в качестве S, $W\text{in}_B$ в качестве F, и $\varnothing$ – в качестве P.

Согласно утверждению 3, множества W_A и W_B будут вычислены как заключительные значения переменной X, причем число итераций цикла repeat – until будет не более $|P_A \cup P_B|$.

Задача для самостоятельного решения: запрограммируйте метод восходящего динамического программирования для решения игры в числа и игры в даты.

- Игра в числа. Позициями в этой игре являются натуральные числа от 1 до 109. Игроки ходят строго по очереди: Алиса – Боб – … Ходы для обоих игроков совпадают: из позиции p можно походить в позицию $(p + 1)$ или в позицию $(p + 10)$. Проигрывает тот игрок, который первым назовет число 100 или большее.
- Игра в даты. Позициями в этой игре являются даты 2011 и начала 2012 г. – от 1 января 2011 г. до 31 января 2012 г. Игроки ходят строго по очереди. Ходы для обоих игроков совпадают: из даты p можно походить в следующую календарную дату или в то же число следующего календарного месяца; например, из 1 января 2011 г. можно походить как во 2 января 2011 г., так и в 1 февраля 2012 г., но из 30 января 2011 г. можно походить только в 31 января 2011, поскольку даты «30 февраля 2011 г.» не существует. Проигрывает тот игрок, который первым назовет дату 2012 г.

Синтаксический анализ контекстно-свободных языков. Теория синтаксического анализа – хорошо развитое направление программирования [10; 11]. Особое значение имеет синтаксический анализ контекстно-свободных языков (к-с языков). Первым формально обоснованным методом синтаксического анализа к-с языков стал алгоритм Коука – Янгера – Касами, который мы приведем (с обоснованием) далее.

Алфавит – это произвольное конечное множество A, состоящее из «символов». Слово в алфавите A – это любая конечная последовательность символов из A. Язык в алфавите A – это любое множество слов в этом алфавите A. В частности, A^* – стандартное обозначение для языка всех слов [10] в алфавите A. Для двух слов w' и w'' пишут $w' \equiv w''$ и говорят, что слова синтаксически совпадают, если w' и w'' – две одинаковые последовательности символов; говорят, что w' является подсловом w'', если $w'' \equiv uw'v$, где u и v – произвольные слова (пустые в том числе).

Контекстно-свободная грамматика (к-с грамматика) – это четверка $G = (N, T, P, S)$, где

- N и T непересекающиеся алфавиты нетерминальных и терминальных символов,
- $P \subseteq N \times (N \cup T)^*$ – конечное множество «продукций», каждая из которых имеет вид «$n \to w$», где $n \in N$, $w \in (N \cup T)^*$,
- $S \in N$ – «стартовый символ».

Говорят, что к-с грамматика находится в нормальной форме Хомского, если стартовый символ не используется в правых частях продукций, и все ее продукции имеют вид $n \to n'n''$ или $n \to t$, где $n, n', n'' \in N$ – нетерминальные символы, а $t \in T$ – терминальный символ.

Вывод в G – это конечная последовательность слов $w_0, \ldots w_k, w_{k+1}, \ldots w_m$, $m \geq 0$, в объединенном алфавите $(N \cup T)$, в которой каждое следующее слово w_{k+1} получается из предыдущего w_k в результате однократного применения какой-либо продукции, т. е. w_k можно представить в виде $w'nw''$, а w_{k+1} – в виде $w'ww''$, где $(n \to w)$ – продукция G. Для слов w', w'' в объединенном алфавите $(N \cup T)$ принято писать $w' \Rightarrow w''$ и говорить, что w'' выводится из w', если существует вывод $w_0, \ldots w_k, w_{k+1}, \ldots w_m$, $m \geq 0$, который начинается со слова w', а заканчивается словом w''. Язык, порождаемый грамматикой G, есть множество $L(G)$ всех слов терминального алфавита, которые выводятся из стартового символа: $L(G) = \{w \in T^* \mid S \Rightarrow w\}$.

Язык называется контекстно-свободным (к-с языком), если он порождается некоторой к-с грамматикой. Синтаксический анализа для к-с языка – это сопоставление слов в терми-

[10] Включая пустое слово ε.

нальном алфавите с его к-с грамматикой с целью распознать слова языка и отвергнуть слова, не входящие в язык. Произвести синтаксический анализ какого-либо слова w в соответствии с к-с грамматикой G – значит построить множество всех пар (n, u), где n – нетерминал из N, а u – подслово w такое, что $n \Rightarrow u$.

Две к-с грамматики называются эквивалентными по языку, если они порождают один и тот же к-с язык. Известно (см. [10. С. 352–358), что для каждой к-с грамматики, которая не порождает пустого слова, можно построить эквивалентную по языку к-с грамматику в нормальной форме Хомского [11] (в случае, если пустое слово порождается, достаточно добавить только одну продукцию $S \to \varepsilon$, где S – стартовый символ, а ε – пустое слово). Поэтому, не теряя общности, мы можем считать, перед нами стоит задача синтаксического разбора для не содержащего пустого слова к-с языка, порожденого к-с грамматикой G в нормальной форме Хомского.

Итак, пусть $G = (N, T, P, S)$ – к-с грамматика в нормальной форме Хомского, $L = L(G)$ – порожденный ею язык, а $w \in T^*$ – произвольное слово в терминальном алфавите, синтаксический разбор которого надо произвести. Рассмотрим множество D всех пар вида (n, u), где n – нетерминал из N, а u – непустое подслово w, и его подмножество $SA = \{(n, u) \in D \mid n \Rightarrow u\}$.

Легко видеть, что SA – это наименьшее подмножество D, для которого верно следующее равенство:

- $SA = \{(n, t) \in D \mid t \in T, (n \to t) \in P\} \cup \{(n, u) \in D \mid \exists(n', u'),(n'', u'') \in SA: u \equiv u'u''$ и $(n \to n'n'') \in P\}$.

Заметим, что функция

- deriv$e$: $X \mid\to \{(n, u) \in D \mid \exists(n', u'),(n'', u'') \in X : u \equiv u'u''$ и $(n \to n'n'') \in P\}$

является всюду определенным монотонным функционалом на D. Поэтому множество SA можно вычислить с использованием формализованного метода восходящего динамического программирования, если принять $\{(n, t) \in D \mid t \in T, (n \to t) \in P\}$ в качестве S, deriv$e$ – в качестве F, и $\{(S, w)\}$ – в качестве P. Это и есть алгоритм Коука – Янгера – Касами синтаксического анализа к-с языков в «теоретико-множественной» форме. Согласно утверждению 3, при построении множества SA число итераций цикла repeat – until будет не более $|D| = |N| \times |w|$ или $O(|w|)$, если считать грамматику фиксированной. Знаменитая оценка сложности $O(|w|^3)$ алгоритма Коука – Янгера – Касами в классической «матричной» форме получается за счет того, что на каждой итерации этого цикла реализуется проверка условия $(\exists(n', u'),(n'', u'') \in X : u \equiv u'u''$ и $(n \to n'n'') \in P)$ перебором уже полученных пар (n, u).

А ларчик просто открывался...

Эта статья началась с головоломки (про оптимальное число бросаний двух кирпичей из башни высотой в сто метров) и двух ее задач-обобщений (об оптимальном бросании двух кирпичей в башне заданной высоты и об оптимальном бросании данного числа кирпичей в башне заданной высоты). Головоломка пока так и не решена, вторая задача решена теоретически, третья предложена для самостоятельного решения на компьютере.

Сейчас, когда рассказ о трех ликах динамического программирования завершен, можно решить и головоломку, и ее обобщения. Только для этого понадобится следующий новый вариант обобщения головоломки (нумерация продолжает нумерацию из части 1).

4. У вас есть башня неограниченной высоты. Какова максимальная высота $H(N, B)$, которой достаточно для определения марки кирпичей, если у вас B кирпичей, а общее число разрешенных испытаний (бросаний) N?

Для решения этой задачи попробуем составить уравнение Беллмана, которому должна удовлетворять искомая функция H: $N \times N \to N$. Выберем произвольные $n, b \geq 0$ и пусть $h = H(n, b)$. Выберем оптимальный по числу бросаний алгоритм для высоты h и b кирпичей и пусть $m = M(h, b)$.

$$H(n, k) = (H(n - 1, k - 1) + 1) + H(n - 1, k).$$

[11] Мы не обсуждаем сложность перевода грамматики в эквивалентную по языку к-с грамматику в нормальной форме Хомского. Подробнее об этом можно прочитать в работе [12].

Смысл этого уравнения прост: первое слагаемое в правой части соответствует случаю, когда первый кирпич разбился при первом бросании, а второе – тому, что не разбился. Остается дополнить это уравнение очевидными соотношением $H(1, 1) = 1$, $H(n, 0) = H(0, k) = 0$ и… получить биномиальные коэффициенты в качестве решения

$$H(N, B) = N! / (B! \times (N - B)!),$$

а треугольник Паскаля – как эффективный метод вычисления значений функции $H(M, B)$. Отсюда получаем

- $M(H, B) = \min\{m \in N \mid m! / (B! \times (m - B)!) \geq H\}$,
- $N(H) = \min\{n \in N \mid n \times (n + 1)/2 \geq H\}$,
- минимальное число бросаний двух кирпичей с башни высотой сто метров равно 14.

Список литературы

1. *Шилов Н. В.* Заметки о трех парадигмах программирования // Компьютерные инструменты в образовании. 2010. № 2. С. 24–37.

2. *Шилов Н. В.* Заметки о парадигмах программирования // Потенциал. 2010. № 4. С. 33–38.

3. *Беллман Р.* Динамическое программирование. М.: Иностр. лит., 1960.

4. *Щербина О. А.* Методологические аспекты динамического программирования // Динамические системы. 2007. Вып. 22. С. 21–36.

5. *Астапов Д.* Рекурсия + мемоизация = динамическое программирование // Практика функционального программирования. 2009, № 3. С. 17–33. URL: http://fprog.ru/2009/issue3/dmitry-astapov-recursion-memoization-dynamic-programming (дата обращения: 18.02.2011).

6. *Грис Д.* Наука программирования. М.: Мир, 1984.

7. *Шилов Н. В.* Основы синтаксиса, семантики, трансляции и верификации программ: Учеб. пособие. Новосибирск, 2011.

8. *Knaster B., Tarski A.* Un théorème sur les fonctions d'ensembles // Ann. Soc. Polon. Math. 1928. Vol. 6. P. 133–134.

9. *Оуэн Г.* Теория игр. М.: Мир, 1979.

10. *Ахо А. В., Ульман Дж. Д.* Теория синтаксического анализа, перевода и компиляции: В 2 т. М.: Мир, 1978. Т. 1.

11. *Ахо А. В., Лам М. С., Сети Р., Ульман Дж. Д.* Компиляторы: принципы, технологии и инструментарий. 2-е изд. М.: Вильямс, 2008.

12. *Lange M., Leiß H.* To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm // Informatica Didactica. 2009. Vol. 8. URL: http://www.informatica-didactica.de/cmsmadesimple/index.php?page=LangeLeiss2009_en (дата обращения: 18.02.2011).

Материал поступил в редколлегию 23.04.2012

N. V. Shilov

THREE FACES OF DYNAMIC PROGRAMMING

The paper discusses some methodological issues of Dynamic Programming by study of some programming contest problem A methodological novelty consists in treatment (interpretation) of ascending Dynamic Programming as least fixpoint computation (according to Knaster – Tarski fix-point theorem). This interpretation leads to an uniform approach to classical optimization problems as well as to problems where optimality is not explicit (Cocke – Younger – Kasami parsing algorithm for example). This interpretation leads also to an opportunity to design a unified template for imperative Dynamic Programming in a form of algorithm design template.

Keywords: Dynamic programming, recursive descending dynamic programming, memoization of recursive programs, iterative ascending dynamic programming, Knaster – Tarski fixpoint theorem, solution of a finite game, context-free parsing.