Лекция 2
Итерация

Лектор:
Д.Н. Лавров

Краткое содержание

- Математическая модель итерации
- Построение цикла с помощью инварианта

Определение итерации

Постановка задачи

- Пусть M – некоторое множетство
- $P: M \rightarrow \{False, True\}$ – предикат на M
- $M \setminus P = \{x \in M: P(x)=False\}$
- Требуется найти такой x, что $P(x)=True$

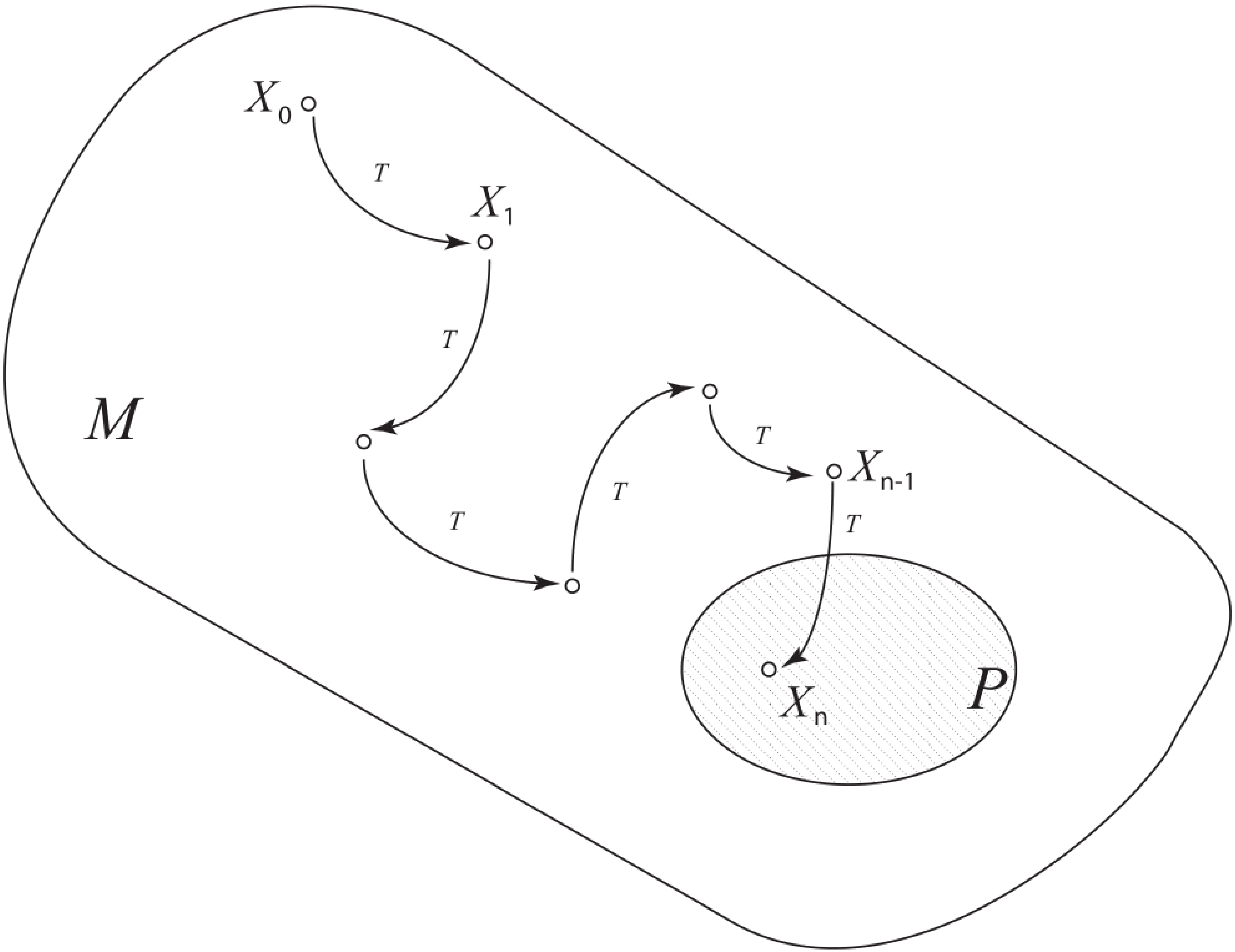
Решение методом итерации (повторения)

- Строится некоторое отображение $T : M \setminus P \rightarrow M$
- T последовательно применяется, начиная с какого-то $x_0 \in$ M:

$$x_1 = T(x_0),$$
$$x_2 = T(x_1),$$
$$\ldots,$$
$$x_n = T(x_{n-1})$$

- до тех пор, пока мы не получим некоторое x_i, для которого $P(x_i) = True$

Математическая модель итерации



Определение итерации

- ***Итерация*** – способ организации обработки данных, при котором определённые действия повторяются многократно, не приводя при этом к рекурсивным вызовам программ.

Обсуждение

- Модель и определение объясняют, что происходит, но не помогают построить алгоритм
- Если найдено или задано *T*, то легко строиться алгоритм

```
x=x0
while not P(x):
    x=T(x)
```

- Найти *T* и есть главная творческая задача программиста

Понятие инварианта

- Основная идея заключается в том, что необходимо сформулировать логическое утверждение (предикат), истинное для всех изменяемых объектов (переменных) на каждом шаге цикла, связывающее отношения между объектами.
- Такой предикат, так как он не меняется в процессе выполнения цикла, называется ***инвариантом***.
- Описание с помощью инварианта статическое, поэтому его легко понять и спроектировать.

Проектирования цикла с помощью инварианта

Обозначения:

M – некоторое множество;

$P : M \rightarrow \{Flase, True\}$ – предикат, задающий искомое свойство объекта;

$M \setminus P = \{x \in M : P(x) = True\}$ – множество тех элементов M, для которых $P(x) = False$;

$T : M \setminus P \rightarrow M$ – некоторое преобразование (из метода итераций).

Пусть существуют предикаты:

$I: M \rightarrow \{False, True\}$ – инвариант, условие истинное для всех переменных на каждом шаге цикла (см. свойства ниже);

$Q: M \rightarrow \{False, True\}$ – условие окончания итерации.

Введённые предикаты должны обладать следующими ***свойствами***:

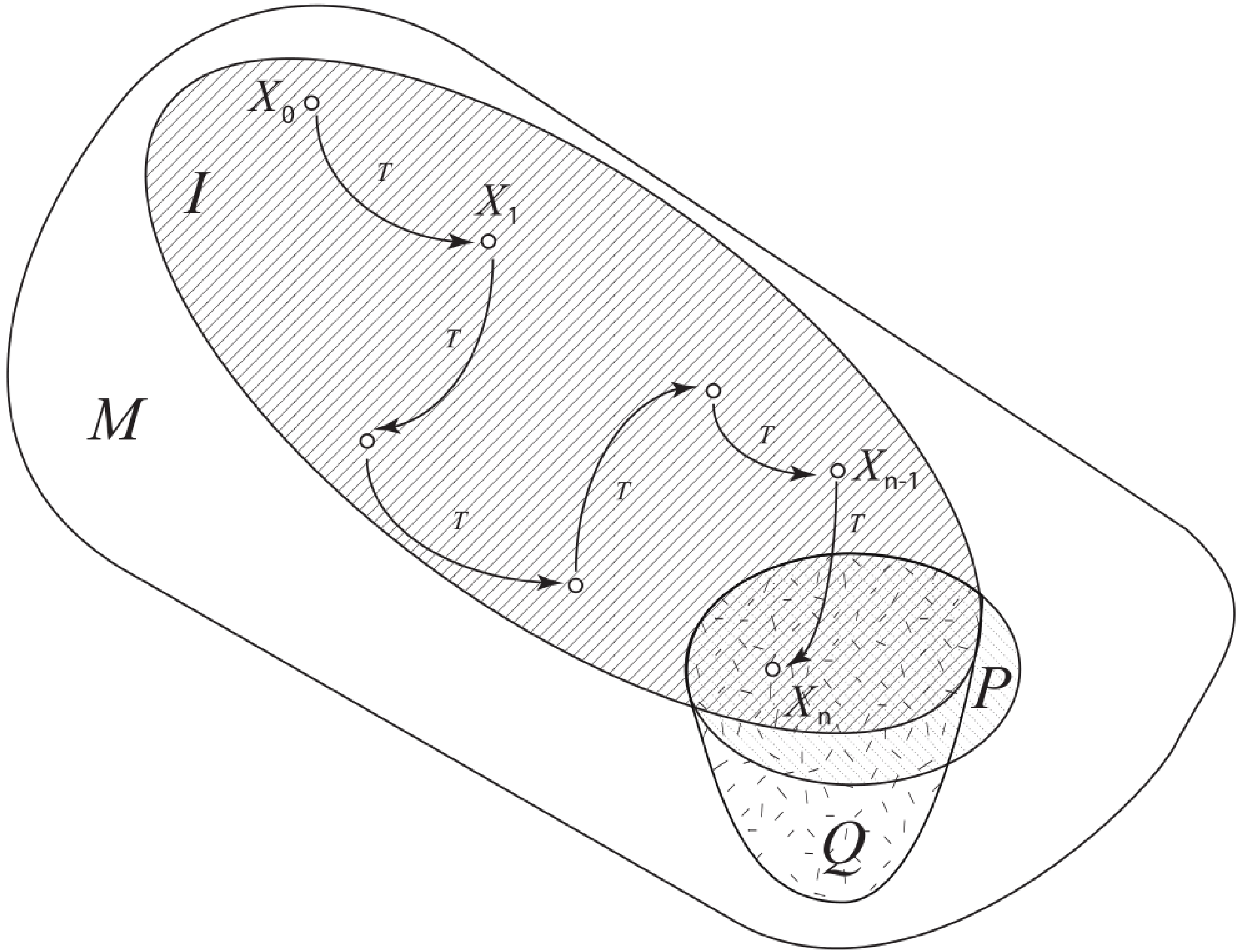
1) $I(x) \& Q(x) \Rightarrow P(x)$ – при окончании цикла выполняется предикат наличия искомого свойства (постусловие);

2) $I(x_0) = True$ – инвариант выполнен для начального значения (предусловие);

3) $I(x) = True \Rightarrow I(T(x)) = True$ – инвариант сохраняется при преобразовании T.

Тогда в методе итераций в качестве условия окончания **вместо** $P(x)$ **можно взять** $Q(x)$.

К понятию инварианта цикла



Как решать задачи с помощью инварианта

- На практике поступают следующим образом. Вначале придумывают общую стратегию решения задачи: определяют какие объекты и как будут меняться на каждом шаге цикла. Фактически этим мы задам отображение *T(x)*.
- формулируется условие окончания цикла *Q(x)*;
- формулируется инвариант *I(x)* – условие которое связывает наши объекты и их изменения с шагом цикла и между собой;
- необходимо, чтобы одновременное выполнение условий окончания цикла *Q(x)* и инварианта *I(X)* означало обладание заданным свойством искомого объекта (выполнение условия *P(x)*).

- Если удалось сформулировать такое утверждение *I(x)*, то цикл описывается следующим кодом:

```
x=x0
while not Q(x):
        x=T(x)
```

- В силу того, что *T(x)* не меняет инвариант, то в конце *I(x)* остаётся истинным. Кроме того, после окончания цикла истинно и *Q(x)*,
  а из *P(x)&Q(X)* = *True* следует *P(x)* = *True*.
- Таким образом, найден *x* обладающий искомым свойством *P*.

Пример применения теории

- **Задача**. Напишите программу, перемножающую два целых неотрицательных числа a и b без использования операции умножения.
- **Решение 1**. Идея: умножение заменить повторным сложением.
- Введём обозначение $x = (s,y)$ – вектор переменных, где предполагаем, что в s будет исходный результат, а $(b - y)$ хранит число текущее число повторений a.
- Условие окончания цикла также становится очевидным $Q(x) : b - y = b$

  или после упрощения
  $Q(x) : y = 0$.

Решения 1
Формулировка инварианта

- Теперь необходимо сформулировать высказывание инварианта и проверить выполнение предусловий.
- Предлагается такое высказывание *I(x)*:
«На каждом шаге цикла в переменной s лежит текущее произведение ***a*** на ***(b − y)***».
- Или на формальном языке
$I(x) : s = a \cdot (b - y)$.
- В данном контексте =

эквивалентно Python-скому ==

Продолжение решения 1

- Шаг алгоритма будет выглядеть так:

  $x_{k+1} = T(x_k)$

- или

  $(s_k + a,\ y_k - 1) = T(s_k, y_k)$

Решение 1. Предусловие

- Перед началом работы цикла положим
  $y = b$ и $s = 0$,
- тогда $I(x) : s = a \cdot 0$ – верное высказывание.
- Что означает, что предусловия выполнены.

Решение 1. Постусловие

- $P(x) : s = a \cdot b$ – верно, если $Q(x) : y = 0$ и $I(x) : s = a(b - y) = \text{True}$ – верны одновременно.
- Это означает, что постусловие выполнено
- Нам осталось проверить только сохранения инварианта $I(x) = True \rightarrow I(T(x)) = True$

Решение 1
Сохранение инварианта

- Пусть на каком-то шаге $I(x) = True$, это означает, что $(s = a(b - y)) = True$.
- Тогда $I(T(x)) = I((s+a, y-1))$. И далее

  $I((s+a, y-1)) : s+a = a(b-(y-1))$,

  а после упрощения

  $I((s+a, y-1)) : s = a(b-y+1)-a$

  или

  $I((s+a, y-1)) : s = a(b-y)$ – что является верным высказыванием в силу начального предположения
  $I(x) : s = a(b - y) = True$

Решение 1
Сохранение инварианта

```
# Первое решение "в лоб"
a=int(input("a="))
b=int(input("b="))
y=b
s=0
while not y==0: # или проще y>0
    s+=a
    y-=1
print(s)
```

Решение 1
Обсуждение

- Все эти рассуждения очень похожи на доказательство методом математической индукции.
- Мы не только построили алгоритм, мы сделали больше – мы доказали, что этот алгоритм корректный и даёт правильный результат при корректных входных данных
- От выбора T зависит эффективность алгоритма. Сколько операций сложения требуется для решения задачи предложенным алгоритмом? Оценка в O-нотации очевидна O(b) .
- Можно ли выполнить эту операцию ещё быстрее? Ответ утвердительный – ДА.

Решение 2. Идея

- Алгоритм будет повторять идею быстрого алгоритма возведения в степень путём повторного возведения а квадрат. Но операции умножения будут заменены на сложение и возведение в квадрат будет заменено умножением на 2.
- Трудоемкость $O(\log_2 b)$
- *Идея алгоритма* следующая если число повторений y – чётное, то удваиваем слагаемое x, а число повторений уменьшаем вдвое.
- Иначе, к результату добавляем слагаемое, а число повторов уменьшаем на 1.

Решение 2. Преобразование T()

- Исходя из вышесказанного, T(x,y,s) определяется следующим правилом:
- $T(x_k, y_k, s_k) =$

  $=(x_k, y_k-1, s_k+x_k)$, если y_k – нечётное;

  $=(x_k+x_k, y_k/2, s_k)$, если y_k – чётное.

Решение 2. Проверка условий

- Условие окончания итерации (цикла) Q(x,y,s) : y = 0.
- Инвариант $I(x,y,s) : s + x \quad y = a \cdot b$.
- Предусловие: $I(a,b,0) : 0 + a \cdot b = a \cdot b$ – верное высказывание.
- Постусловие. Из истинности по окончанию итераций $Q(x,y,s)$ следует, что

  $y = 0, \quad I(x,0,s) : s + x \cdot 0 = a \cdot b$ – должно быть верным высказыванием и переменная s будет содержать искомый результат, если только T(x,y,s) сохраняет инвариант:
- Пусть $I(x_k ,y_k ,s_k) = True$, что эквивалентно истинности равенства $s_k + x_k \cdot y_k = a \cdot b$.
- Если y_k нечётное, то $I(T(x_k ,y_k ,s_k)) = I(x_k ,y_k -1,s_k +x_k)$. Но тогда $(s_k+x_k) + x_k \cdot (y_k -1) = s_k+x_k \cdot y_k = a \cdot b$.
- Если y_k чётное, то $I(T(x_k ,y_k ,s_k)) = I(x_k+x_k, y_k/2, s_k)$, то есть

  $s_k + 2x_k \cdot y_k /2 = s_k + x_k \cdot y_k = a \cdot b$.
- Утверждение о корректности алгоритма доказано.

Решение 2. Вариант 1

```
# В точности по сформулированным Q(x), T(x) и I(x).
a=int(input("a="))
b=int(input("b="))
x=a; y=b; s=0
while not y==0:
    if y%2==1:
        s+=x
        y-=1
    else:
        x+=x
        y//=2
print(s)
```

Решение 2. Вариант 2

```
# С заменой сложений на битовые операции
# и упрощением логических выражений
a=int(input("a="))
b=int(input("b="))
x=a; y=b; s=0
while y:
    if y&1:          # тоже самое, что и y%2==1
        s += x
        y -= 1
    else:
        x<<=1  # тоже самое, что и x+=x или x=x*2
        y>>=1  # тоже самое, что и y//=2 или y=y//2
print(s)
```

- Можно заметить, что блок битовых сдвигов можно выполнять на каждом шаге потому, что если y нечётное, то на следующем шаге цикла y обязано уже быть чётным, а, следовательно, блок битовых сдвигов можно выполнить сразу на этой же итерации.

Решение 2. Вариант 3

```
# Оптимальное итоговое решение
a=int(input("a="))
b=int(input("b="))
x=a; y=b; s=0
while y:
    if y&1:
        s += x
        y -= 1
    x<<=1
    y>>=1
print(s)
```

Вопросы