Темы 1-4. Конструирование Компиляторов, Углубленный Курс

Валера Мацкевич, 3 курс ПМИ ВШЭ СПб

на 08.10.2024

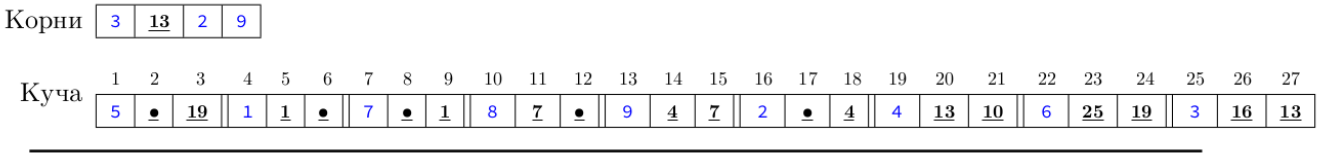

Рис. 1: Куча

1. *Примените алгоритм обхода в глубину с разворотом указателей к состоянию памяти представленному на рис. 1 и ответьте на вопросы*:

   - *Какие блоки памяти будут помечены по итогу работы алгоритма?*

     Только блоки памяти, начинающиеся с адресов $1, 4, 7, 10, 13, 19$ (всего 6 блоков), поскольку лишь они достижимы из корней.

   - *Каково будет состояние кучи и локальных переменных алгоритма в момент, когда будет помечен блок со значением 7 в первом поле?*

     (a) *Необходимо указать значения во всех ячейках памяти в куче или указать ячейки, которые имеют значение, отличное от исходного.*
     Отличные от исходного значения находятся в следующих ячейках памяти: `[7] = 10`, `[11] = 19`, `[21] = 1`, `[3] = 4`, `[5] = 13`, `[14] = nil`.

     (b) *Необходимо указать значения в массиве* **`done`**.
     В массиве `done` нули находятся на всех позициях кроме блоков памяти, начинающихся со следующих позиций: `done[13] = 1`, `done[4] = 1`, `done[1] = 2`, `done[19] = 2`, `done[10] = 1`.

     (c) *Необходимо указать значения переменных* **`t, x, y`**.
     Значения будут следующие: `t = 10`, `x = 7`, `y = 7`

   - *Сколько операций записи (изменения) памяти в куче и массиве* **`done`** *требуется для алгоритма на данном примере?*

     На каждый спуск вниз по дереву поиска в глубину производится 2 записи в память: одну на разворот указателя, одну на его восстановление, а спусков всего 5, поэтому будет 10 записей в память.
     Для массива `done` всё проще: поскольку мы знаем, что будет затронуто 6 блоков, а для покраски блока полностью требуется `BLOCK_SIZE + 1` записей, то в него будет произведено 24 записи.

   - *Какова амортизированная стоимость сборки мусора (в терминах операции записи/изменения памяти в куче и массиве* **`done`***) на данном примере?*

     Ответ: $\frac{10+24}{H-R} = \frac{34}{3} = 11$ записей.

2. *Примените алгоритм сборки мусора копированием с гибридным перенаправлением указателей, к состоянию памяти представленному на рис. 1 и ответьте на вопросы ниже. В контексте сборки копированием, раздел from-space включает адреса с 1 до 30 (включительно), а раздел to-space — адреса с 31 до 60 (включительно).*

- *Каково состояние кучи после работы алгоритма?*
  Корни:

3	31	2	9

  Память (x значит, что значения не отличаются):

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
37	x	x	46	x	x	x	34	x	x	43	x	x	31	x	x	x	x	x	40	x	x	x	x	x	x	x	x	x	x

31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
9	46	34	7	nil	37	5	nil	40	4	31	43	8	34	nil	1	37	nil	x	x	x	x	x	x	x	x	x	x	x	x

- *Какой адрес в to-space соответсвует адресу 4 из from-space?*
  Ответ: 46.
- *Каково состояние кучи в момент вызова процедуры* `Forward(p4)`*, где* `p4` *— адрес копии данных, которые находились по адресу 4 до сборки?*
  В предположении, что в условии задачи имелся ввиду вызов `Forward(4)`, состояние кучи такое же, как и в пункте "после работы алгоритма за исключением: `[4] = x`, `[32] = x`, `[46] = x`, `[47] = x`, `[48] = x`.
- *Сколько операций записи (изменения) памяти в куче требуется для алгоритма на данном примере? Считайте, что копирование каждого машинного слова стоит 1 единицу.*
  Ответ: 18 копирований и 6 записей, итого 24 единицы.
- *Какова амортизированная стоимость сборки мусора (в терминах операции записи/изменения памяти в куче) на данном примере?*
  Ответ: $\frac{24}{H-R} = 8$ записей.

3. *Аргументированно ответьте на вопросы связанные с работой следующей программы:*

```
1  n = new Counter(value: 0)              # создание нового счётчика со значением 0
2
3  def inc(c):
4    return new Counter(value: c.value + 1)
5
6  def next():
7    n = inc(n)
8
9  def append(lst, x):
10   if lst == null:
11     return new Cons(head: x, tail: null)
12   else:
13     lst.tail = append(lst.tail, x)
14     return lst
15
16 def r():
17   a = null            # создание пустого связного списка
18   while n.value < 6:
19     next()
20     i = new Counter(value: 1)
21     j = i
22     while j.value < n.value:
23       j = inc(j)
24       i = new Counter(value: i.value + j.value)
25     a = append(a, i)  # добавление элемента в конец списка
26   return a
27
28 s = 0
29 for k in r():
30   s += k
31 print(s)
```

Рис. 2: Программа с циклом

- *Каково общее количество памяти (кол-во машинных слов), которое выделяет эта программа на куче на протяжении своей работы?*

7 раз будет создана переменная `n`, по 6 раз во внешнем цикле в функции `r` будут вызов `append` (в хвостовом вызове которого всегда есть аллокация 2-ух слов) и аллокация `i`, также внутренний цикл содержит $0+1+2+3+4+5 = 15$ аллокаций переменной `j` и `i`, итого – $7+3\cdot 6+2\cdot 15 = 55$ машинных слов.

Ответ: 55 машинных слов.

- *Какое максимальное количество живой памяти (достижимых машинных слов) находится на куче в течение работы этой программы?*

  Максимум одновременно могут жить слово `j`, слово `i`, слово `n`, и $2 \cdot 6$ слов `a`, итого $3 + 2 \cdot 6 = 15$ машинных слов.

  Ответ: 15 машинных слов.

- *При использовании копирующего сборщика мусора без поколений, достаточно ли будет* 20 *машинных слов на from-space (и столько же на to-space)? Достаточно ли* 15 *машинных слов?* 30 *машинных слов?*

  20 слов достаточно, если запускать наш stop-the-world копирующий сборщих мусора после каждой аллокации, переполняющей from-space. 30 слов хватит с запасом. 15 слов – нет, поскольку при пиковом потреблении памяти, увеличение любого из счётчиков создаёт новый "временный"объект, который некуда положить.

- *При использовании сборки по поколениям (на основе копирующего сборщика мусора) с двумя поколениями (G_0 и G_1) общим размером в* 30 *машинных слов, как бы вы разделили память по поколениям (сколько машинных слов будет относится к G_0, а сколько — к G_1)?*

  Память которой имеет смысл жить долго для данной программы – лишь переменные `n`, и список `a`, итого получая $1 + 2 \cdot 6 = 13$ слов на G_1.

4. *Продемонстрируйте работу алгоритма Бейкера (инкрементальная сборка мусора), полагаясь на гибридный (semi-depth-first) алгоритм обхода и перенаправления указателей, на следующей программе. Учтите, что общая доступная память (from-space + to-space) –* 60 *машинных слов.*

```
1  def fib(n):
2    if (n < 3):
3      return new Node(value: 1, left: null, right: null)
4    else:
5      x = new Node(value: n, left: null, right: null)
6      x.right = fib(n - 2)
7      x.left = fib(n - 1)
8      x.value = x.left.value + x.right.value
9      fib[n - 1] = x
10     return x
11
12 z = fib(5)
13 print(z.value)
14 print(fib(3).value)
15 z = z.left.left
16 print(z.left.value)
17 print(fib(4).value)
```

Рис. 3: Программа с Фибоначчи

Примечание: мне всегда казалось, что алгоритм Бейкера не имеет как такового разделения на from-space и to-space, что он на самом деле treadmill алгоритм с раскраской в три цвета. Судя по вопросам в этом задании – я не прав, поэтому это задание я решал на интуицию...

- *В какой момент работы программы происходит инициализация сборки мусора? Происходит ли инициализация второй раз? Если да, то в какой момент?*

Вычисление `fib(5)` генерирует $3 \cdot 9 = 27$ машинных слов. После, при вычислении `fib(3)`, во время аллокации `fib(1)` на строчке 3, память в from-space заканчивается, **отчего впервые происходит инициализация сборщика мусора**.

Во второй раз инициализация не произойдёт, потому что всего памяти выделяется недостаточно: каждая аллокация очищает по одному элементу (по крайней мере, пока не дойдёт до `fib(5).left.left`).

- *Сколько мусора (кол-во машинных слов) находится на куче в момент вызова сборщика мусора (первый раз)?*

  0, мусора не появится вплоть до конца жизни `fib(3)` на 14-й строчке.

- *Сколько мусора (кол-во машинных слов) находится на куче в момент завершения сборки мусора (первый раз)?*

  Мы очистим всё, кроме `fib(3)` с 14-й строчки. То есть $3 \cdot 3 = 9$ машинных слов.

- *Каково состояние кучи после выполнения строчки 16 в основной программе?*

  (a) *Покажите состояние ячеек памяти в from-space и to-space.*

  (b) *Покажите значения (куда указывают) переменные `scan`, `next`, `limit`.*