Максим Исаев

Темы 1–4. Теоретическое задание

Корни

3	10	2	9

Куча

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
5	●	7	1	19	●	7	●	13	8	16	●	9	10	16	2	7	25	4	4	22	6	19	4	3	●	1

Рис 1: Пример состояния памяти какой-то программы.

Задание (1): Примените алгоритм обхода в глубину с разворотом указателей [1, Algorithm 13.6] к состоянию памяти представленному на Рис 1 и ответьте на вопросы:

1. Какие блоки памяти (достаточно указать адреса начала блоков) будут *помечены* по итогу работы алгоритма?
2. Каково будет состояние кучи и локальных переменных алгоритма в момент, когда будет помечен блок со значением 7 в первом поле?
   1. Необходимо указать значения во всех ячейках памяти в куче или указать ячейки, которые имеют значение, отличное от исходного.
   2. Необходимо указать значения в массиве `done`.
   3. Необходимо указать значения переменных `t`, `x`, `y`.
3. Сколько операций записи (изменения) памяти в *куче* и массиве `done` требуется для алгоритма на данном примере?
4. Какова амортизированная стоимость сборки мусора (в терминах операции записи/изменения памяти в куче и массиве `done`) на данном примере?

Решение:

Обозначения:

- ...обозначения из задания
- | 8 | 16 | ● | – помеченный блок.
- | 8 | ● | ● | – второе поле временно перезаписано.

Состояния памяти:

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
5	●	7	1	19	●	7	●	13	8	16	●	9	10	16	2	7	25	4	4	22	6	19	4	3	●	1

↓ нашли непомеченный 10 в корнях

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
5	●	7	1	19	●	7	●	13	8	16	●	9	10	16	2	7	25	4	4	22	6	19	4	3	●	1

↓ первое поле не указатель, второе 16 не помечено

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
5	●	7	1	19	●	7	●	13	8	●	●	9	10	16	2	7	25	4	4	22	6	19	4	3	●	1

↓

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
5	●	7	1	19	●	7	●	13	8	●	●	9	10	16	2	10	25	4	4	22	6	19	4	3	●	1

↓

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
5	●	7	1	19	●	7	●	16	8	●	●	9	10	16	2	10	25	4	4	22	6	19	4	3	●	1

↓ у 13 нет непомеченных полей-указателей, выходим вверх по *стеку*

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
5	●	7	1	19	●	7	●	13	8	●	●	9	10	16	2	10	25	4	4	22	6	19	4	3	●	1

↓ переходим к следующему полю-указателю у 16

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
5	●	7	1	19	●	7	●	13	8	●	●	9	10	16	2	7	25	4	4	22	6	19	4	3	●	1

↓

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
5	●	7	1	19	●	7	●	13	8	●	●	9	10	16	2	7	10	4	4	22	6	19	4	3	●	1

↓

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
5	●	7	1	19	●	7	●	13	8	●	●	9	10	16	2	7	10	4	4	22	6	19	4	3	●	16

↓ у 1 нет непомеченных полей-указателей

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
5	●	7	1	19	●	7	●	13	8	●	●	9	10	16	2	7	10	4	4	22	6	19	4	3	●	1

↓

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
5	●	7	1	19	●	7	●	13	8	●	●	9	10	16	2	7	25	4	4	22	6	19	4	3	●	1

↓

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
5	●	7	1	19	●	7	●	13	8	16	●	9	10	16	2	7	25	4	4	22	6	19	4	3	●	1

Ответы:

1. Адреса помеченных блоков: 1, 7, 10, 13, 16, 25
2. Состояние памяти в момент пометки блока со значением 7 в первом поле:
   1.

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
5	●	7	1	19	●	7	●	13	8	●	●	9	10	16	2	10	25	4	4	22	6	19	4	3	●	1

   2. Массив done: [0,0,0,1,0,1,0,0,0]
   3. Переменные: t = 16, x = 7, y = 7
3. 6 (массив `done`) + 10 (изменения в куче) = 16 операций записи
4. $$\frac{c_1 R + c_2 H}{H - R} = \frac{10 \cdot (6 \cdot 3) + 3 \cdot 27}{27 - 6 \cdot 3} = 29$$

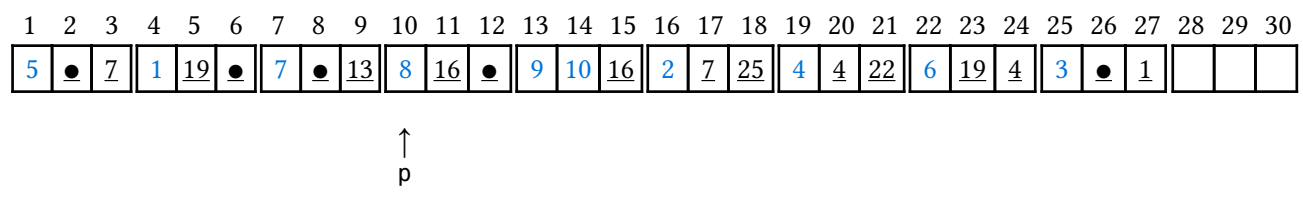
□

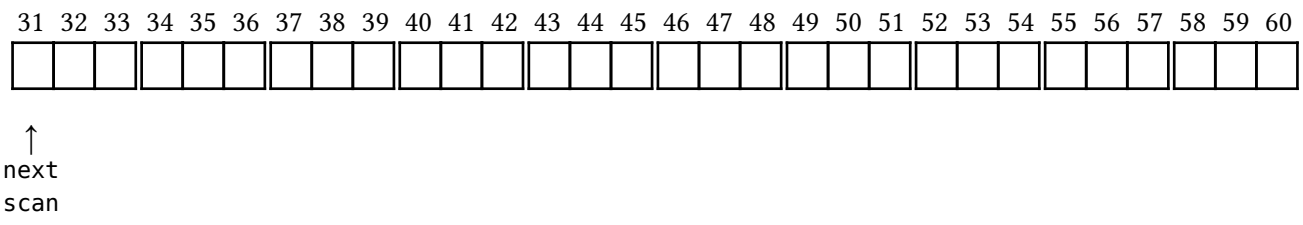
Задание (2): Примените алгоритм сборки мусора копированием [1, Algorithm 13.9] с гибридным перенаправлением указателей [1, Algorithm 13.11], к состоянию памяти представленному на Рис 1 и ответьте на вопросы ниже. В контексте сборки копированием, раздел *from-space* включает адреса с 1 до 30 (включительно), а раздел *to-space* — адреса с 31 до 60 (включительно).

1. Каково состояние кучи после работы алгоритма?
2. Какой адрес p_1 (в *to-space*) соответсвует адресу 1 из *from-space*? То есть, по какому адресу будет находится объект по адресу 1 после копирования?
3. Каково состояние кучи в момент вызова процедуры **`Forward`**(p_1), где p_1 — адрес *копии* данных, которые находились по адресу 4 до сборки?
4. Сколько операций записи (изменения) памяти в *куче* требуется для алгоритма на данном примере? Считайте, что копирование одного машинного слова (из *from-space* в *to-space*) — это одна операция.
5. Какова амортизированная стоимость сборки мусора (то есть количество операций записи/изменения памяти в куче на количество собранного мусора) на данном примере?

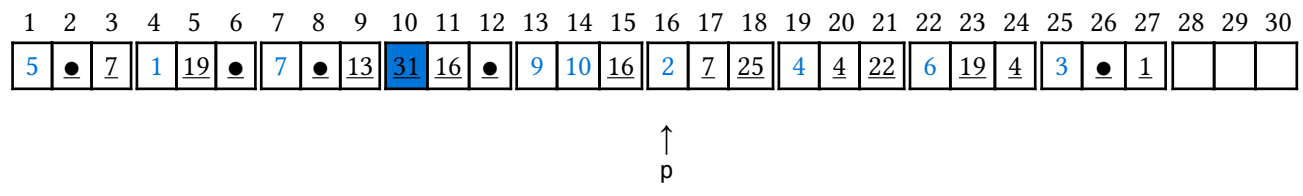
Решение:

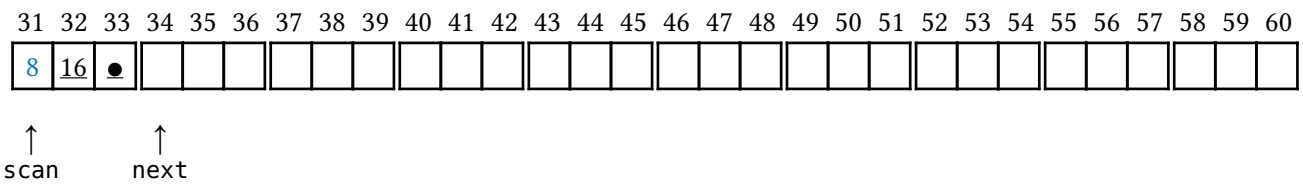
Изменение памяти во время исполнения программы:



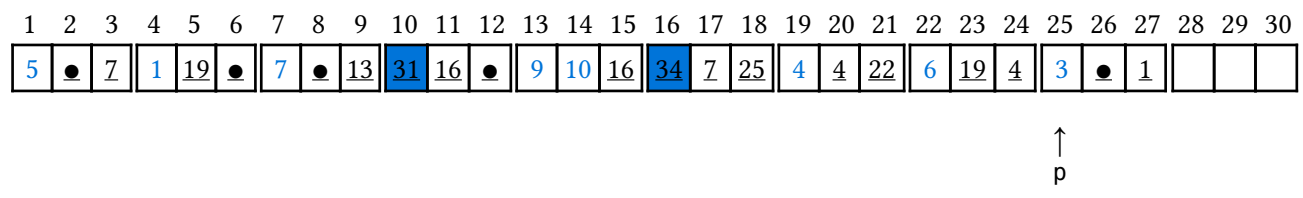


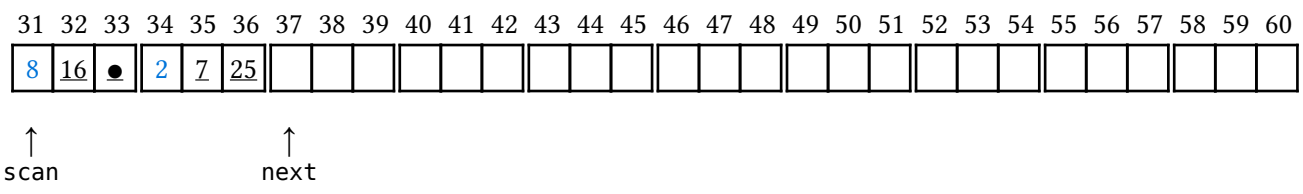
↓ Forward(10) + первая итерация Chase(10)



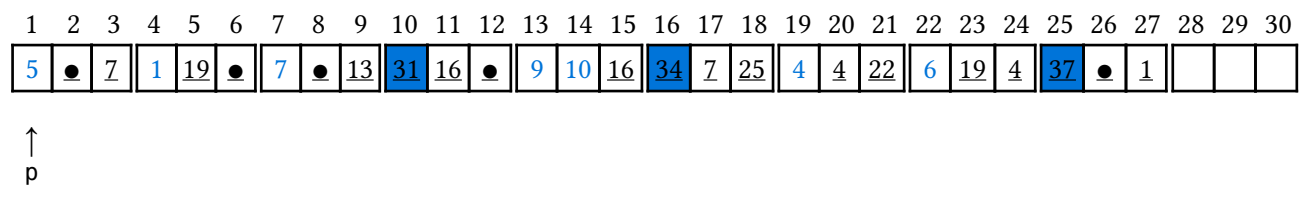


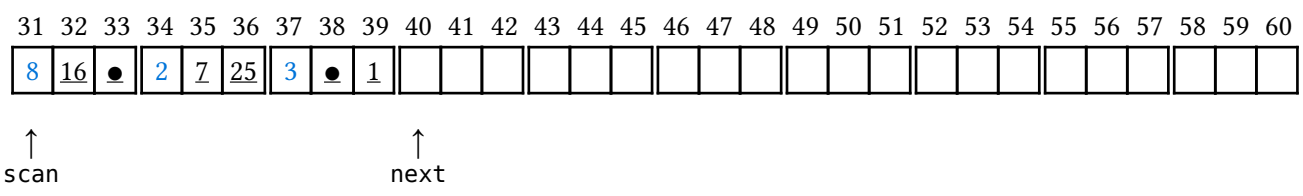
↓ вторая итерация Chase(10)



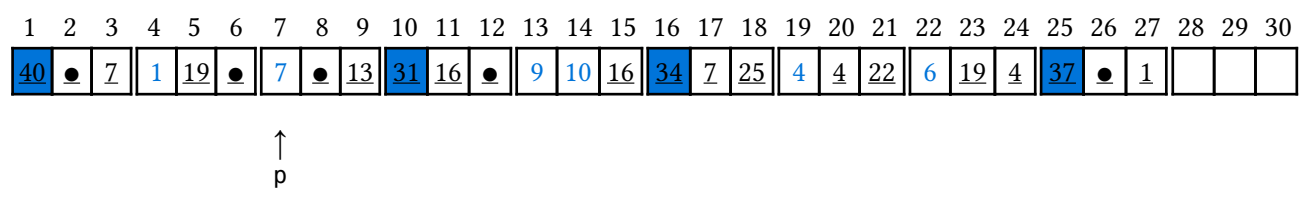


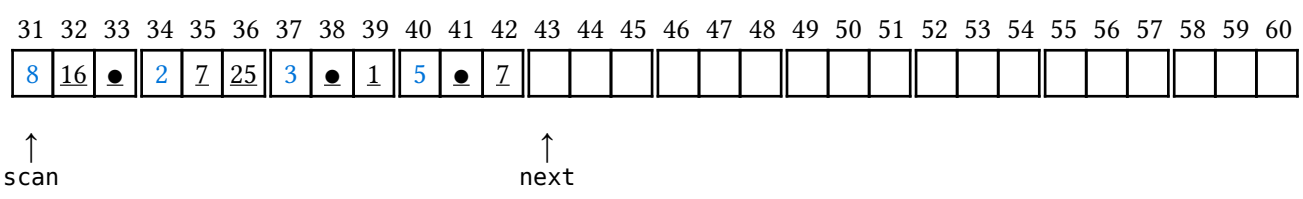
↓ 3-я итерация Chase(10)



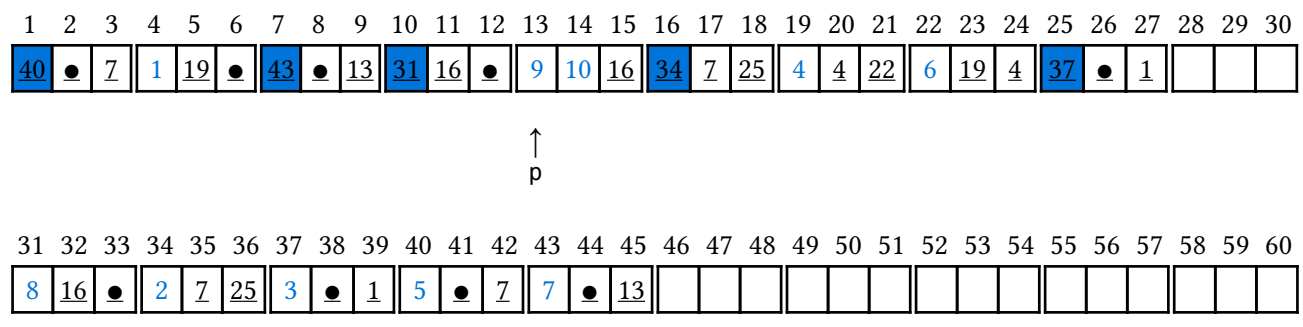


↓ 4-я итерация Chase(10)

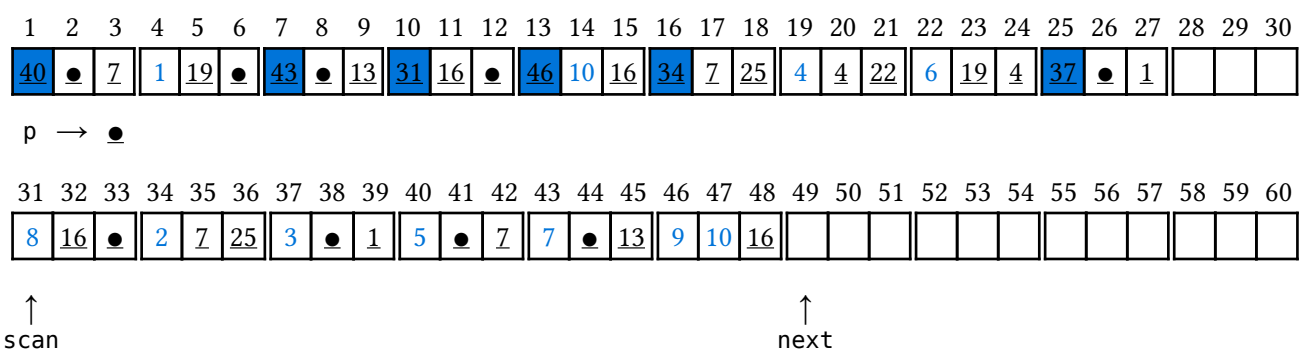




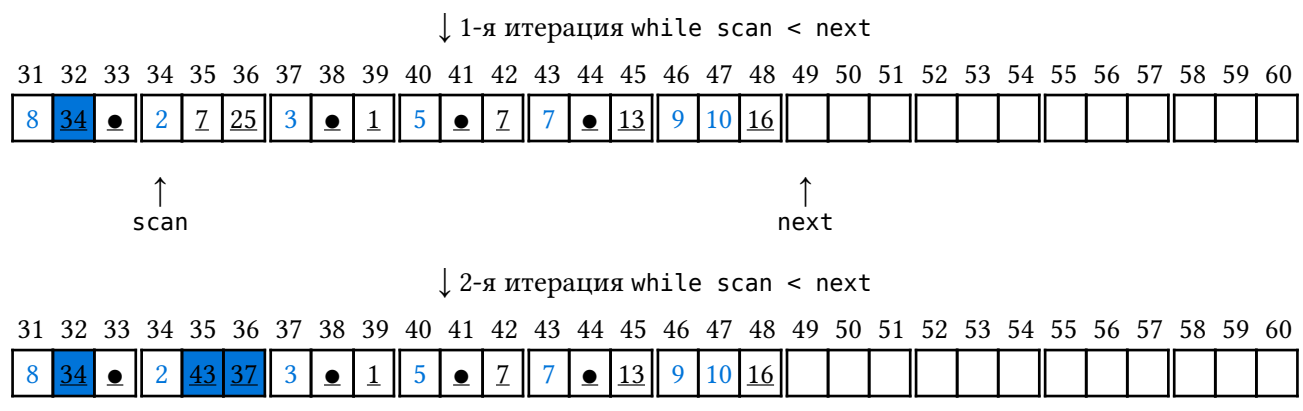
↓ 5-я итерация Chase(10)



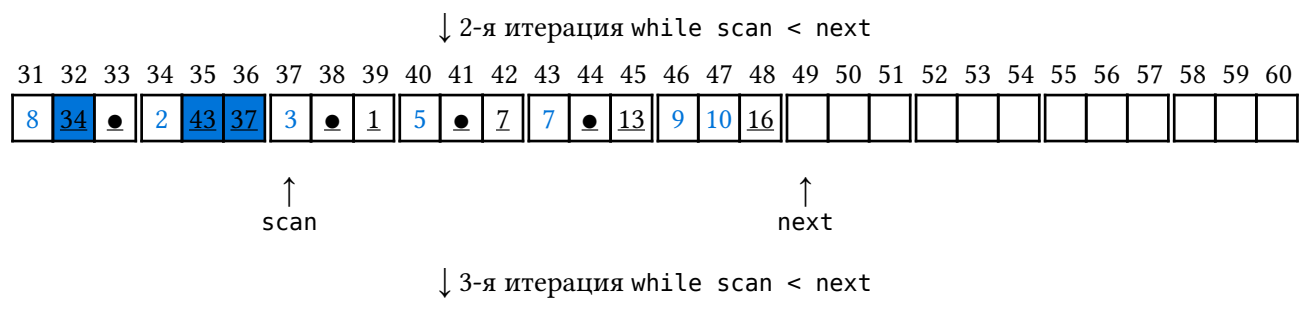
↓ 6-я итерация Chase(10)



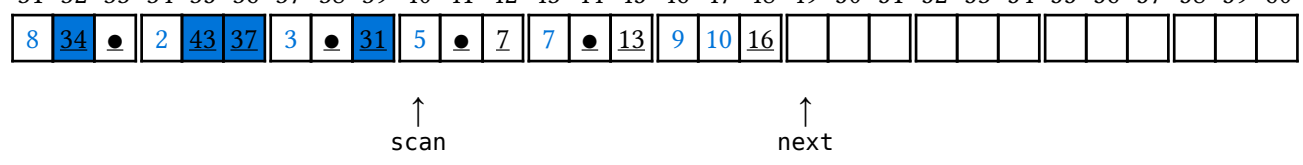
↓ 1-я итерация `while scan < next`



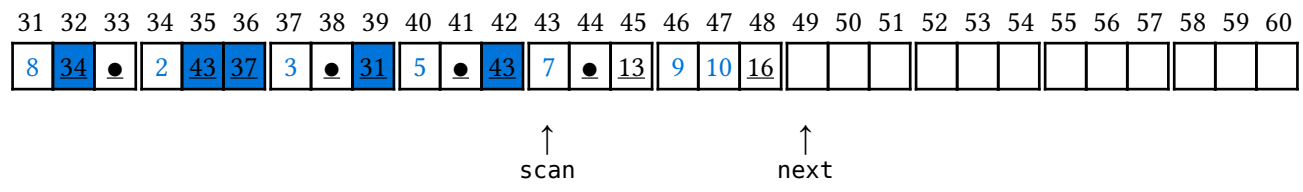
↓ 2-я итерация `while scan < next`



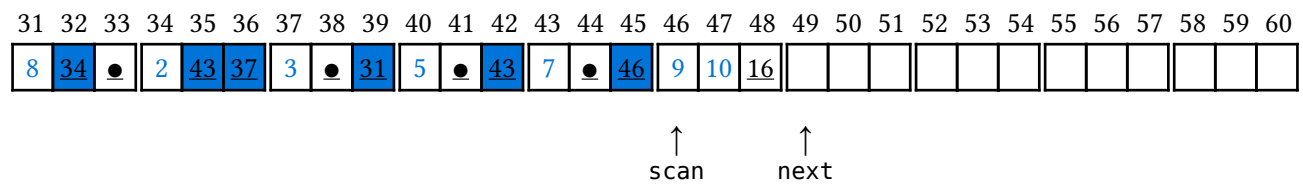
↓ 3-я итерация `while scan < next`



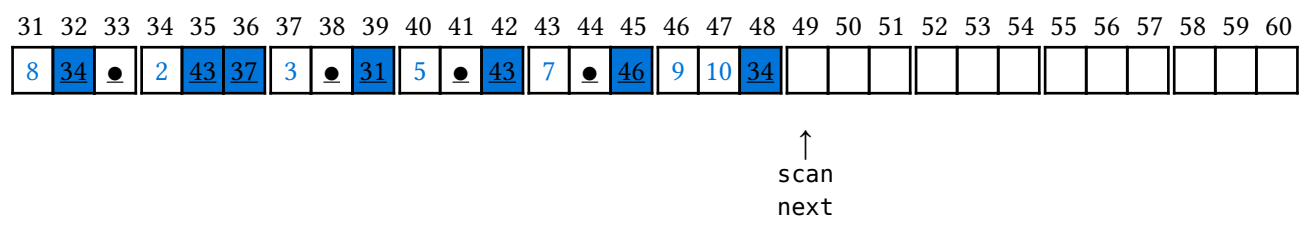
↓ 4-я итерация `while scan < next`



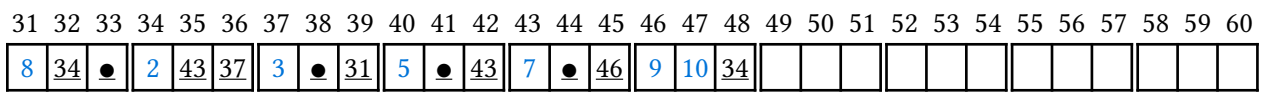
↓ 5-я итерация `while scan < next`



↓ 6-я итерация `while scan < next`



Ответы:

1. 

2. 40
3. Объект по адресу 4 недостежим и поэтому у него не будет адреса в *to-space* и **`Forward`** вызван не будет.
4. $R \cdot (<\text{кол-во полей}> + 1\ (\text{перезапись } p.f_1)) + <\text{кол-во указателей в достижимых объектах}> = 6 \cdot (3 + 1) + 7 = 31$
5. $$\frac{c_3 R}{\frac{H}{2} - R} = \frac{10 \cdot (6 \cdot 3)}{\frac{60}{2} - 6 \cdot 3} = 15$$

□

Задание (3):

Продемонстрируйте работу алгоритма Бейкера [1, §13.6] (инкрементальная сборка мусора), полагаясь на гибридный (semi-depth-first) алгоритм обхода и перенаправления указателей [1, Algorithm 13.11], на следующей программе. Учтите, что общая доступная память (*from-space* + *to-space*) — 60 машинных слов.

```python
def insert(t, x):
  if t is None:
    return [x, None, None]
  elif x < t[0]:
    r = [t[0], None, t[2]]
    lt = insert(t[1], x)
    r[1] = lt
    return r
  else:
    r = [t[0], t[1], None]
    rt = insert(t[2], x)
    r[2] = rt
    return r

n = 12
t = None
while n > 1:
  t = insert(t, n)
  if n % 2 == 0:
    n = n / 2
  else:
    n = 3 * n + 1
print(t)
```

1. В какой момент работы программы происходит инициализация сборки мусора? Происходит ли инициализация второй раз? Если да, то в какой момент?
2. Сколько мусора (кол-во машинных слов) находится на куче в момент вызова сборщика мусора (первый раз)?
3. Сколько мусора (кол-во машинных слов) находится на куче в момент завершения сборки мусора (первый раз)?
4. Каково состояние кучи после вызова insert(t, 5) в основной программе?
   (a) Покажите состояние ячеек памяти в from-space и to-space.
   (b) Покажите значения корней основной программы.
   (c) Покажите значения (куда указывают) переменные scan, next, limit.

Решение:

Изменение памяти во время исполнения программы:

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
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
																													↑ limit

n = 12, t = ●

↓ insert(●, 12)

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
12	●	●																											

n = 6, t = [12, ●, ●]

↓ insert(1, 6)

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
12	●	●	12	●	●																								

↓ insert(●, 6)

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
12	●	●	12	7	●	6	●	●																					

n = 6, t = [12, [6, ●, ●], ●]

↓ insert(4, 3)

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
12	●	●	12	7	●	6	●	●	12	●	●																		

↓ insert(7, 3)

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
12	●	●	12	7	●	6	●	●	12	●	●	6	●	●															

↓ insert(●, 3)

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	3	●	●												

n = 10, t = [12, [6, [3, ●, ●], ●], ●]

↓ insert(10, 10)

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	3	●	●	12	●	●									

↓ insert(13, 10)

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	3	●	●	12	●	●	6	16	●						

↓ insert(●, 10)

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	3	●	●	12	22	●	6	16	25	10	●	●			

n = 5, t = [12, [6.0, [3.0, ●, ●], [10.0, ●, ●]], ●]

↓ insert(19, 5)

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	3	●	●	12	22	●	6	16	25	10	●	●	12	●	●
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
↑ scan next																													↑ limit

Попытка аллоцировать память для нового объекта [6, None, ptr(25)] вызывает сборку мусора.

↓ insert(22, 5)

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	3	●	●	31	22	●	34	16	25	37	●	●	40	●	●
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
12	34	●	6	16	25	10	●	●	12	●	●																6	●	37
			↑ scan									↑ next															↑ limit		

↓ insert(16, 5) read barier 16 -> 43

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	43	●	●	31	22	●	34	16	25	37	●	●	40	●	●
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
12	34	●	6	43	37	10	●	●	12	●	●	3	●	●										3	●	●	6	●	37
						↑ scan									↑ next									↑ limit					

↓ insert(●, 5)

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	43	●	●	31	22	●	34	16	25	37	●	●	40	●	●
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
12	34	●	6	43	37	10	●	●	12	58	●	3	●	●							5	●	●	3	52	●	6	55	37
									↑ scan						↑ next						↑ limit								

↓ Конец сборки мусора insert(49, 8)

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	43	●	●	31	22	●	34	16	25	37	●	●	40	●	●
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
12	34	●	6	43	37	10	●	●	12	58	●	3	●	●	16	●	●	12	●	46	5	●	●	3	52	●	6	55	37
															↑ scan next limit														

Ответы:

1. Инициализация сборки мусора (помечено красным) произойдёт на 5 итерации при попытке аллоцировать память для нового объекта [6, None, ptr(25)].

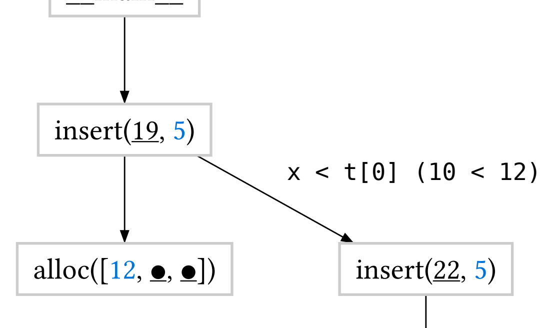


Figure 1: Дерево вызовов на 5-й итерации

   Инициализация сборки мусора произойдёт второй раз, т.к. всего будет 28 аллокаций (легко проверить запустив программу и посчитав создания списков), что равно $28 * 3 = 84$ машинным словам, а за одну итерацию можно собрать не более половины памяти, т.е. 30 машинных слов: $84 - 30 = 54 > 30$.

2. Рассмотрим состояние памяти в момент начала сборки мусора:

Корни	
t	r
19	28

   Достижимые объекты:

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	3	●	●	**12**	**22**	**●**	6	16	25	10	●	●	**12**	**●**	**●**

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	3	●	●	**12**	**22**	**●**	**6**	**16**	**25**	10	●	●	**12**	**●**	**●**

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	**3**	**●**	**●**	**12**	**22**	**●**	**6**	**16**	**25**	**10**	**●**	**●**	**12**	**●**	**●**

   Итого на куче 5 недостижимых объектов: **15 мусорных машинных слов**.

3. Рассмотрим состояние памяти в момент начала сборки мусора:

Корни	
t	r
40	49

   Достижимые объекты:

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
12	34	●	6	43	37	10	●	●	**12**	**58**	**●**	3	●	●	16	●	●	**12**	**●**	**46**	5	●	●	3	52	●	6	55	37

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
12	34	●	6	43	37	10	●	●	**12**	**58**	**●**	3	●	●	**16**	**●**	**●**	**12**	**●**	**46**	5	●	●	3	52	●	**6**	**55**	**37**

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
12	34	●	6	43	37	**10**	**●**	**●**	**12**	**58**	**●**	3	●	●	**16**	**●**	**●**	**12**	**●**	**46**	5	●	●	**3**	**52**	**●**	**6**	**55**	**37**

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
12	34	●	6	43	37	**10**	**●**	**●**	**12**	**58**	**●**	3	●	●	**16**	**●**	**●**	**12**	**●**	**46**	**5**	**●**	**●**	**3**	**52**	**●**	**6**	**55**	**37**

   Итого на куче 3 недостижимых объектов: **9 мусорных машинных слов**.

4. (a) Память на момент **после** вызова `insert(t, 5)`:

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
12	●	●	12	7	●	6	●	●	12	13	●	6	16	●	43	●	●	31	22	●	34	16	25	37	●	●	40	●	●
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
12	34	●	6	43	37	10	●	●	12	58	●	3	●	●							5	●	●	3	52	●	6	55	37
									↑ scan						↑ next						↑ limit								

   (b)

Корни	
t	n
40	5

   (c)

Переменные сборщика мусора		
scan	next	limit
40	46	51

□

Задание (4):

```
def f(x):
  n = x[0]
  while x[1] is not None:
    x = x[1]
  while n > 1:
    x[1] = [n - 1, None]
    x = x[1]
    n = n - 1

s = 0
def g(x):
  global s
  if x[1] is not None:
    n = x[0]
    x = x[1]
    s = s + n
    return x
  else:
    return None

x = [7, None]
while x is not None:
  f(x)
  x = g(x)
  print(x)
```

1. Каково общее количество памяти (кол-во машинных слов), которое выделяет эта программа на куче на протяжении своей работы?
2. Какое максимальное количество *живой* памяти (достижимых машинных слов) находится на куче в течение работы этой программы?
3. При использовании копирующего сборщика мусора без поколений [1, §13.3], достаточно ли будет 30 машинных слов на *from-space* (и столько же на *to-space*)? Достаточно ли 20 машинных слов? 25 машинных слов?
4. При использовании сборки по поколениям [1, §13.4] (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 относиться к G_0, а сколько – к G_1)? Обоснуйте свой ответ.

Решение: Проанализируем поведение функций:

- `f` добавляет в конце списка `x` (`x` – список в смысле (`голова : хвост`)) числа от `x[0]-1` до `1`.

```
x = [3, None]
f(x)
# x == [3, [2, [1, None]]]
```

- `g` удаляет первый элемент списка и добавляет его значение к глобальной переменной `s`.

```
x = [3, [2, [1, None]]]
x = g(x)
# x == [2, [1, None]]
```

Рассмотрим композицию:

```
def h(x):
  f(x)
  return g(x)
```

`h` добавляет в конец списка числа от `x[0]-1` до `1`, а затем удаляет первый элемент списка и возвращает хвост.

```
x = [3, None]
x = h(x)
# x == [2, [1, None]]
```

Основная программа вызывает `h` пока `x` не станет `None`, т.е. пока не удалит все элементы списка. Начальное значение `x` – список из одного элемента `7`, значит `h` будет вызвана 7 раз.

Ответы:

1. Один вызов `h` создаёт `x[0]` объектов на куче. Эти числа в будущем тоже станет `x[0]` и породят другие объекты. Получаем рекуренту:

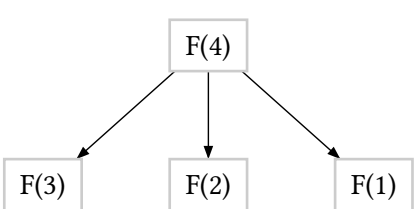


Figure 2: Пример для `F(4)`

$$F(n) = \sum_{i=1}^{n-1} F(i)$$

Угадывается паттрен $2^{n-1} - 1$ при $n \geq 1$.

Итого получаем, что при вызове `h([7, None])` будет создано $F(7)$ объектов на куче, т.е 63.

Всего программа выделяет 63 объекта по 2 машинных слова = **126 машинных слов**.

2. В каждый момент исполнения программы есть только один корень указывающий на кучу – `x`, поэтому вся достижимая память может быть только в списке `x`. Запустив программу получим, что максимальная длина списка `x` за всё время исполнения: 23. Т.к вывод случается после вызова `g`, то нужно добавить ещё 1.

   Итого 24 объекта по 2 машинных слова = **48 машинных слов**.

3. Т.к есть момент когда живо 48 машшиных слов, то ни одного из этих размеров не хватит, но видимо тут имелось ввиду по 60, 50 и 40 машинных слов на *from-space* и *to-space*.

   - 40 машинных слов **не хватит**, т.к это не вместит 48 доступных.
   - 50 машинных слов **хватит**, т.к объекты представляют односвязный список и ссылки не меняются и все объекты одинакового размера, так что не будет дефрагментации памяти и 48 слов, смогут поместиться в *from-space* размера 50.
   - 60 машинных слов **хватит**, т.к хватит и 50 и поведение от этого не меняется.

4. Аналогично предыдущему заданию буду считать, что общий размер $G_0 + G_1 = 60$ машшиных слов.

   Сборка по поколениям спроектирована с гипотизой о том, что объекты умирают молодыми. Однако в этой программе новые объекты всегда добавляются в конец списка и каждый раз умирает самый старый (FIFO). При сборке по поколениям, объекты в G_1 собираются в n раза реже чем в G_0, где n отношение их размеров. Когда в G_0 аллоцируется новый объект он гарантировано переживёт все объекты в G_1. От сюда слудует, что чем меньше n тем эффективнее, т.к мы меньше раз будет просматривать гарантировано живой объект.

   Если взять $n = 0$ (т.е G_0 = 60, G_1 = 0), тогда мы сделаем минимум бесполезной работы. В этой задаче сборка по поколениям никогда не будет эффективной т.к не выполняется гипотиза на который она основана.

□