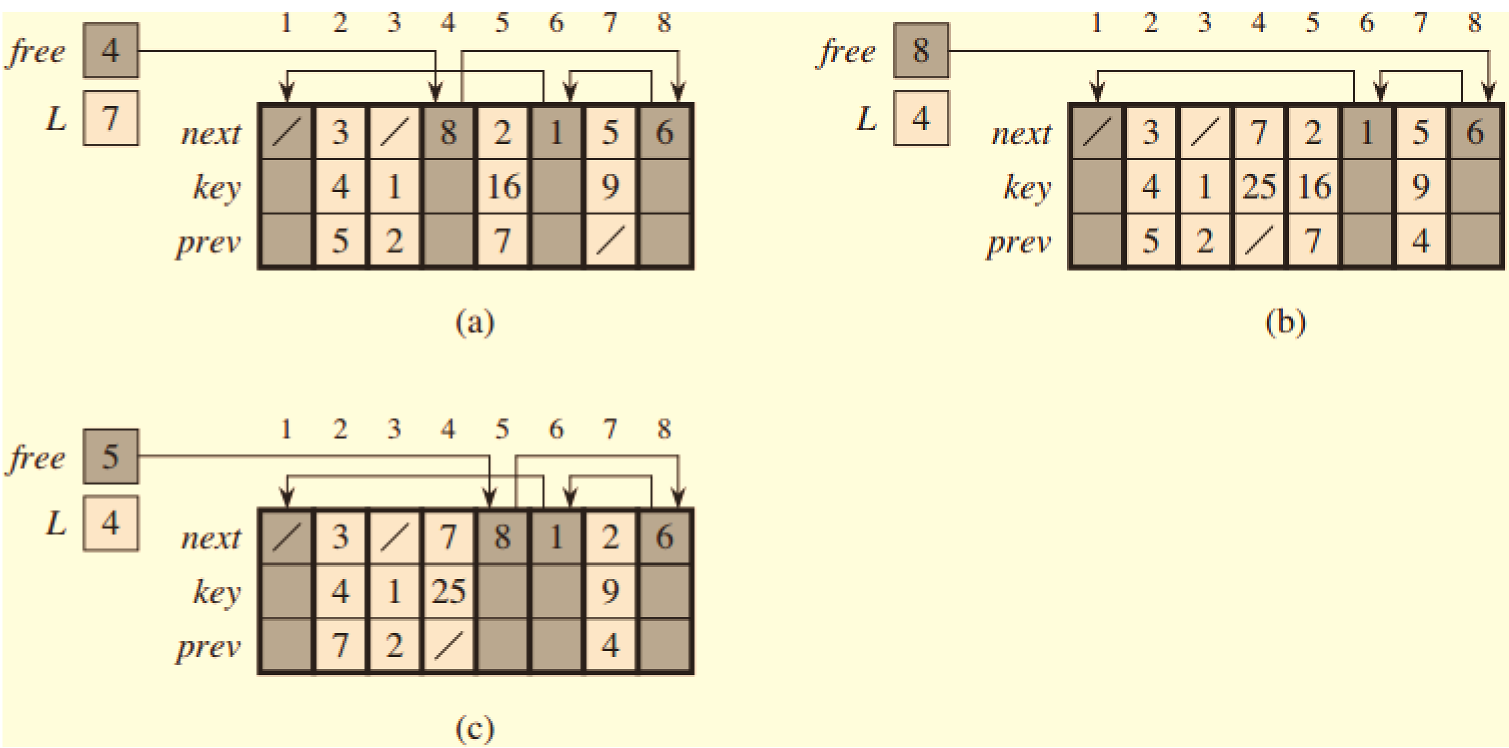
习题2-10

TC-10.3-4，5

Problem 10.3

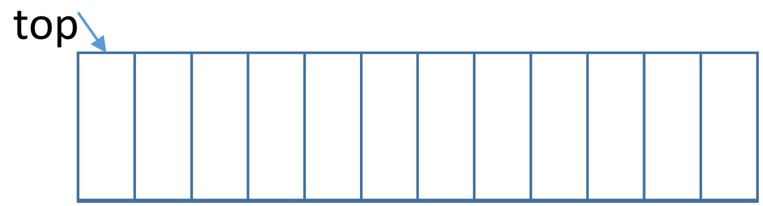
10.3-4

It is often desirable to keep all elements of a doubly linked list compact in storage, using, for example, the first m index locations in the multiple-array representation. (This is the case in a paged, virtual-memory computing environment.) Explain how to implement the procedures ALLOCATE-OBJECT and FREE-OBJECT so that the representation is compact. Assume that there are no pointers to elements of the linked list outside the list itself. (*Hint:* Use the array implementation of a stack.)

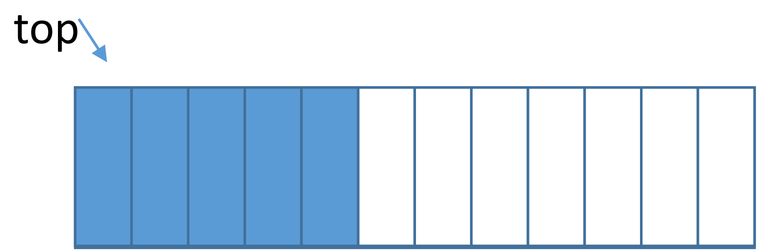


10.3-4

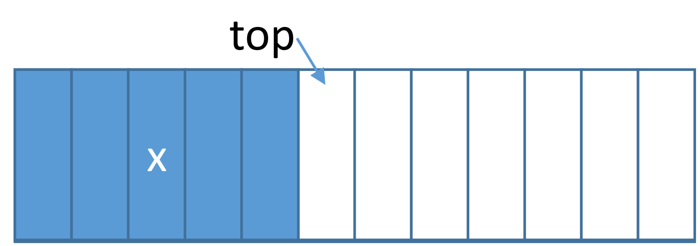
It is often desirable to keep all elements of a doubly linked list compact in storage, using, for example, the first m index locations in the multiple-array representation. (This is the case in a paged, virtual-memory computing environment.) Explain how to implement the procedures ALLOCATE-OBJECT and FREE-OBJECT so that the representation is compact. Assume that there are no pointers to elements of the linked list outside the list itself. (*Hint:* Use the array implementation of a stack.)



ALLOCATE-OBJECT()

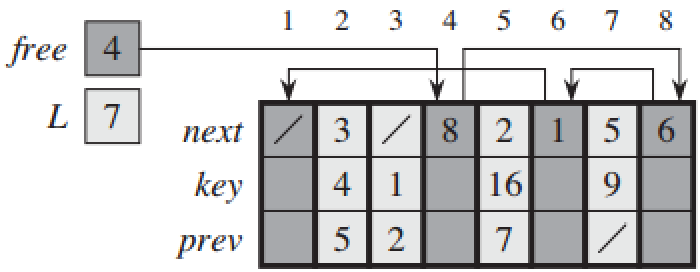


FREE-OBJECT(x)



10.3-5

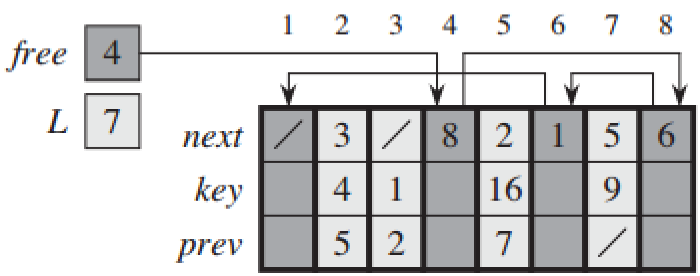
Let L be a doubly linked list of length n stored in arrays *key*, *prev*, and *next* of length m. Suppose that these arrays are managed by ALLOCATE-OBJECT and FREE-OBJECT procedures that keep a doubly linked free list F. Suppose further that of the m items, exactly n are on list L and $m - n$ are on the free list. Write a procedure COMPACTIFY-LIST(L, F) that, given the list L and the free list F, moves the items in L so that they occupy array positions $1, 2, \ldots, n$ and adjusts the free list F so that it remains correct, occupying array positions $n+1, n+2, \ldots, m$. The running time of your procedure should be $\boxed{\Theta(n),}$ and it should use only a constant amount of extra space. Argue that your procedure is correct.




```
COMPACTIFY-LIST(L, F)
if L is empty return
count = 1
while count < m
        do while next[L] <= m
                do L ← next[L]
                    count++
            while next[F] > m
                do F ← next[F]
            swap(key[next[F]], key[next[L]])
            swap(prev[next[F]], prev[next[L]])
            swap(next[F], next[L])
```

10.3-5

Let L be a doubly linked list of length n stored in arrays *key*, *prev*, and *next* of length m. Suppose that these arrays are managed by ALLOCATE-OBJECT and FREE-OBJECT procedures that keep a doubly linked free list F. Suppose further that of the m items, exactly n are on list L and $m - n$ are on the free list. Write a procedure COMPACTIFY-LIST(L, F) that, given the list L and the free list F, moves the items in L so that they occupy array positions $1, 2, \ldots, n$ and adjusts the free list F so that it remains correct, occupying array positions $n+1, n+2, \ldots, m$. The running time of your procedure should be $\Theta(n)$, and it should use only a constant amount of extra space. Argue that your procedure is correct.



需要维护两个链表！！

基本思路：

- i=1；
- x=L;
- $While(x! = nil)$
  - if$(x > n)$
    - **SWAP**（$item_x$，$item_i$）
    - i++;
    - Continue;
  - x=next(x);

10-3 Searching a sorted compact list

Exercise 10.3-4 asked how we might maintain an n-element list compactly in the first n positions of an array. We shall assume that all keys are distinct and that the compact list is also sorted, that is, $key[i] < key[next[i]]$ for all $i = 1, 2, \ldots, n$ such that $next[i] \neq$ NIL. We will also assume that we have a variable L that contains the index of the first element on the list. Under these assumptions, you will show that we can use the following randomized algorithm to search the list in $O(\sqrt{n})$ expected time.

```
COMPACT-LIST-SEARCH(L, n, k)
 1  i = L
 2  while i ≠ NIL and key[i] < k
 3      j = RANDOM(1, n)
 4      if key[i] < key[j] and key[j] ≤ k
 5          i = j
 6          if key[i] == k
 7              return i
 8      i = next[i]
 9  if i == NIL or key[i] > k
10      return NIL
11  else return i
```

```
COMPACT-LIST-SEARCH′(L, n, k, t)
i = L
for q = 1 to t
    j = RANDOM(1, n)
    if key[i] < key[j] and key[j] ≤ k
        i = j
        if key[i] == k
            return i
while i ≠ NIL and key[i] < k
    i = next[i]
if i == NIL or key[i] > k
    return NIL
else return i
```

a. Suppose that COMPACT-LIST-SEARCH(L, n, k) takes t iterations of the **while** loop of lines 2–8. Argue that COMPACT-LIST-SEARCH$'(L, n, k, t)$ returns the same answer and that the total number of iterations of both the **for** and **while** loops within COMPACT-LIST-SEARCH$'$ is at least t.

```
COMPACT-LIST-SEARCH(L, n, k)
1  i = L
2  while i ≠ NIL and key[i] < k
3      j = RANDOM(1, n)
4      if key[i] < key[j] and key[j] ≤ k
5          i = j
6          if key[i] == k
7              return i
8      i = next[i]
9  if i == NIL or key[i] > k
10     return NIL
11 else return i
```

```
COMPACT-LIST-SEARCH′(L, n, k, t)
1  i = L
2  for q = 1 to t
3      j = RANDOM(1, n)
4      if key[i] < key[j] and key[j] ≤ k
5          i = j
6          if key[i] == k
7              return i
8  while i ≠ NIL and key[i] < k
9      i = next[i]
10 if i == NIL or key[i] > k
11     return NIL
12 else return i
```

a. Suppose that COMPACT-LIST-SEARCH(L, n, k) takes t iterations of the **while** loop of lines 2–8. Argue that COMPACT-LIST-SEARCH$'(L, n, k, t)$ returns the same answer and that the total number of iterations of both the **for** and **while** loops within COMPACT-LIST-SEARCH$'$ is at least t.

Case 1: $CLS(L, n, k)$ returns at 7
Case 2: $CLS(L, n, k)$ returns at 10
Case 3: $CLS(L, n, k)$ returns at 11

$CLS(L, n, k, t')$
Case a: $t' < t$
Case b: $t' = t$
Case c: $t' > t$

```
COMPACT-LIST-SEARCH(L, n, k)
1   i = L
2   while i ≠ NIL and key[i] < k
3       j = RANDOM(1, n)
4       if key[i] < key[j] and key[j] ≤ k
5           i = j
6           if key[i] == k
7               return i
8       i = next[i]
9   if i == NIL or key[i] > k
10      return NIL
11  else return i
```

```
COMPACT-LIST-SEARCH'(L, n, k, t)
1   i = L
2   for q = 1 to t
3       j = RANDOM(1, n)
4       if key[i] < key[j] and key[j] ≤ k
5           i = j
6           if key[i] == k
7               return i
8   while i ≠ NIL and key[i] < k
9       i = next[i]
10  if i == NIL or key[i] > k
11      return NIL
12  else return i
```

a. Suppose that COMPACT-LIST-SEARCH(L, n, k) takes t iterations of the **while** loop of lines 2–8. Argue that COMPACT-LIST-SEARCH$'(L, n, k, t)$ returns the same answer and that the total number of iterations of both the **for** and **while** loops within COMPACT-LIST-SEARCH$'$ is at least t.

Case 1: $CLS(L, n, k)$ returns at 7
Case 2: $CLS(L, n, k)$ returns at 10
Case 3: $CLS(L, n, k)$ returns at 11

$CLS(L, n, k, t')$
Case b: $t' = t$
Case c: $t' > t$

- i的取值由j确定，而j的取值则由Random()函数以及执行次数 t 确定
- 由于两者Random()函数一致所以，CLS’(L,n,k,t’)必然在执行t次for循环后在第7行返回相同的结果

```
COMPACT-LIST-SEARCH(L, n, k)
1   i = L
2   while i ≠ NIL and key[i] < k
3       j = RANDOM(1, n)
4       if key[i] < key[j] and key[j] ≤ k
5           i = j
6           if key[i] == k
7               return i
8       i = next[i]
9   if i == NIL or key[i] > k
10      return NIL
11  else return i
```

```
COMPACT-LIST-SEARCH′(L, n, k, t)
1   i = L
2   for q = 1 to t
3       j = RANDOM(1, n)
4       if key[i] < key[j] and key[j] ≤ k
5           i = j
6           if key[i] == k
7               return i
8   while i ≠ NIL and key[i] < k
9       i = next[i]
10  if i == NIL or key[i] > k
11      return NIL
12  else return i
```

注意：此处i ,i‘ 是指具体在链表中的位次，而不是程序中的含义

Case 1-A: $CLS(L,n,k)$ returns at 7, $t' < t$

- i的取值由j确定，而j的取值则由Random()函数以及执行次数 t 确定
- 由于两者Random()函数一致，
- 所以，CLS’(L,n,k,t’)必然在执行t’次for循环后退出循环；此时，$i' < i$, 且$\boldsymbol{i - i' \geq t - t'}$
- **while循环至少需要执行$\boldsymbol{i - i'}$次，且一定能找到$\boldsymbol{i}$;所以，总for+while循环数>=t**

Case 1-bc: $CLS(L,n,k)$ returns at 7, $t' \geq t$

- i的取值由j确定，而j的取值则由Random()函数以及执行次数 t 确定
- 由于两者Random()函数一致
- 所以，CLS’(L,n,k,t’)必然在执行t次for循环后在第7行返回相同的结果

```
COMPACT-LIST-SEARCH(L, n, k)
1   i = L
2   while i ≠ NIL and key[i] < k
3       j = RANDOM(1, n)
4       if key[i] < key[j] and key[j] ≤ k
5           i = j
6           if key[i] == k
7               return i
8       i = next[i]
9   if i == NIL or key[i] > k
10      return NIL
11  else return i
```

```
COMPACT-LIST-SEARCH'(L, n, k, t)
1   i = L
2   for q = 1 to t
3       j = RANDOM(1, n)
4       if key[i] < key[j] and key[j] ≤ k
5           i = j
6           if key[i] == k
7               return i
8   while i ≠ NIL and key[i] < k
9       i = next[i]
10  if i == NIL or key[i] > k
11      return NIL
12  else return i
```

Case 2-A: $CLS(L,n,k)$ returns at 10, $t' < t$

- CLS'(L,n,k,t')必然在执行t'次for循环后退出循环；此时，$i' < i$, 且$\max(\boldsymbol{i}, \boldsymbol{L.length}) - \boldsymbol{i'} \geq \boldsymbol{t} - \boldsymbol{t'}$
- while循环至少需要执行$max(i, L.length) - i'$次，所以，总for+whle循环数>=t，最终在第11行返回

Case 2-bc: $CLS(L,n,k)$ returns at 10, $t' \geq t$

- i的取值由j确定，而j的取值则由Random()函数以及执行次数 确定
- 由于两者Random()函数一致
- 所以，CLS'(L,n,k,t')至少需要执行t'次for循环，并继续执行>=1次while循环，并在第11行返回

```
COMPACT-LIST-SEARCH(L, n, k)
1  i = L
2  while i ≠ NIL and key[i] < k
3      j = RANDOM(1, n)
4      if key[i] < key[j] and key[j] ≤ k
5          i = j
6          if key[i] == k
7              return i
8      i = next[i]
9  if i == NIL or key[i] > k
10     return NIL
11 else return i
```

```
COMPACT-LIST-SEARCH'(L, n, k, t)
1  i = L
2  for q = 1 to t
3      j = RANDOM(1, n)
4      if key[i] < key[j] and key[j] ≤ k
5          i = j
6          if key[i] == k
7              return i
8  while i ≠ NIL and key[i] < k
9      i = next[i]
10 if i == NIL or key[i] > k
11     return NIL
12 else return i
```

Case 3-A: $CLS(L, n, k)$ returns at 11, $t' < t$

- CLS'(L,n,k,t')必然在执行t'次for循环后退出循环；此时，$i' < i$, 且$\max(\boldsymbol{i}, \boldsymbol{L.length}) - \boldsymbol{i'} \geq \boldsymbol{t} - \boldsymbol{t'}$
- while循环至少需要执行$max(i, L.length) - i'$次，所以，总for+whle循环数>=t，最终在12行返回

Case 3-bc: $CLS(L, n, k)$ returns at 11, $t' \geq t$

- i的取值由j确定，而j的取值则由Random()函数以及执行次数 确定
- 由于两者Random()函数一致
- 所以，CLS'(L,n,k,t')至少需要执行t'次for循环，并继续执行>=1次while循环，并在第12行返回