情報数理科学第３回課題　08-192025 林橘平

SLList.hは別ファイルとして添付している。

Ex 3-2

コード(SLList.hの該当部分のみを抽出)

```
T secondLast() {
Node *u = head;
while (true) {
if (u->next == tail) break;
u = u->next;
}
return u->x;
}
```

実行結果(checkEx3_2.cppは配布されたものと同じ)

環境は、自分のMacBookPro　コンパイラはg++

```
(base) MBP:cpp hayashikippei$ g++ -o checkEx3_2 checkEx3_2.cpp
(base) MBP:cpp hayashikippei$ ./checkEx3_2
8
```

考察

ノードuがsecondLast(末尾から２番目の要素)であるかどうかは、u->next == tailの条件式で判断できる。偽であれば次のノードに移り、真であればbreak;でループを抜け、ノードuのデータu->xを返す。実行結果を見ても正しくsecondLastの値を返している。 実行時間はO(n-1)である。 n = 0,1の場合はSegmentation fault: 11となった。u = u->nextによるリスト外へのアクセスが原因と考えられる。

Ex 3-3

コード(SLList.hの該当部分のみを抽出)

```
T get(int i) {
Node *u = head;
for (int j = 0; j < i; j++) u = u->next;
return u->x;
}
```

```
T set(int i, T x){
        Node *u = head;
        for (int j = 0; j < i; j++) u = u->next;
        u->x = x;
        return x;
}

void add(int i, T x){
        if (i == 0) {
                push(x);
        }
        else {
                Node *u = head;
                for (int j = 0; j < i-1; j++) u = u->next;
                Node *t = new Node(x);
                t->next = u->next;
                u->next = t;
                n++;
        }
}

T remove(int i){
        if (n == 0)     return null;
        if (i == 0) {
                return pop();
        }
        else {
                Node *u = head;
                for (int j = 0; j < i-1; j++) u = u->next;
                T x = (u->next)->x;
                u->next = (u->next)->next;
                n--;
                return x;
        }
}
```

実行結果(checkEx3_3.cppは配布されたものと同じ)

環境は、自分のMacBookPro　コンパイラはg++

```
(base) MBP:cpp hayashikippei$ g++ -o checkEx3_3 checkEx3_3.cpp
(base) MBP:cpp hayashikippei$ ./checkEx3_3
10
[20, 11, 22, 12, 13, 14, 105, 16, 17, 18, 19, 30]
```

考察

get(i)はfor (int j = 0; j < i; j++) u = u->next;でi番目のノードにアクセスし、そのデータu->xを返せば良い。setも同様にi番目のノードにアクセスし、そのノードのデータをu->x = x;で書き換えれば良い。これらの実行時間はいずれも高々O(1+i)である。 add(i,x)について。i=0の場合はpush(x)と同じである。それ以外の場合は、まずi-1番目のノードUi-1にアクセスし、そしてデータにxを持つ新しいノードtを作り、t->next = u->next; u->next = t;でUi-1, t , Uiの順番になるようにして最後に配列のサイズnを 1 増やす。こうしてadd(i, x)の作業が完成する。 remove(i)について。i=0の場合はpop()と同じである。それ以外の場合は、まずi-1番目のノードUi-1にアクセスし、 T x (u->next)->x;でUiのデータxを記録する。 そしてu->next = (u->next)->next;によってUi-1の次のノードがUi+1になるようにする。こうすることでUiが削除される。配列のサイズnを 1 減らし、記録しておいた値xを返せばremove(i)の作業が完成する。いずれも実行時間はO(1+i)である。

Ex 3-4

コード(SLList.hの該当部分のみを抽出)

```
void reverse() {
Node *u = head;
Node *p = NULL;
Node *f = head;


    while (u != NULL) {
        f = u->next;
        u->next = p;
        p = u;
        u = f;
    }
    head = p;
}
```

実行結果(checkEx3_4.cppは配布されたものと同じ)

環境は、自分のMacBookPro　コンパイラはg++

```
(base) MBP:cpp hayashikippei$ g++ -o checkEx3_4 checkEx3_4.cpp
(base) MBP:cpp hayashikippei$ ./checkEx3_4
[19, 18, 17, 16, 15, 14, 13, 12, 11, 10]
```

考察

現在のノードu、その前、次のノードp,fを定める。f = u->next; によってuの次のノードをfに記録する。u->next = pによってuとpの順番を入れ替える。そしてp = u; u = f; とすることでfとuの順番が入れ替わる。これをwhile (u != NULL)で繰り返し、最後にhead = p;でheadに元々の配列の最後の要素を入れることで配列のreverse()ができる。実行時間はO(n)である。