確認レポート 12

科目	テーマ	番号	氏名
アルゴリズムとデータ構造 B	線形リストの操作		

問題 1　以下の文章を完成させよ。

線形リストに対する基本処理は、1 次元配列と同様に計算量が O(　　) のアルゴリズムである。

問題 2　リストに対する処理とその計算量のオーダについて考えよ。

リストの初期化：`head=　　;` とする。計算量は、O（　　）。

挿入（新しく確保したノードを r とする。`r = malloc(sizeof(struct Element));`）
リストの先頭にノードを挿入：`r->next=head; head=　　;`。計算量は、O（　　）。
※head からのみデータを出し入れできるようにすることで、　　を実現できる。
リストの途中にノードを挿入：一つ前のノードを記憶するループ `for(p=q=head; p!=NULL;　　,p=p->next)` を利用して挿入する位置を定める。必要ならば、index を保持するループを利用する。挿入する位置が決まったら、`q->next=r; r->next=p;` とする。トータルの計算量は、O（　　）。
リストの末尾にノードを挿入：（基本）末尾まで先頭から辿らなければならない。
（**新たなポインタ tail を用いる場合**）tail が末尾の要素を指しているとして、`tail->next=r;`
`tail=r; tail->next=　　;` とする。計算量は、O（　　）。
※tail からデータを入力し、head からデータを出力することで、　　を実現できる。

削除（※ 領域解放時には free()する）
先頭のノードを削除：`head=head->　　;` とする。計算量は、O（　　）。
途中のノードを削除：一つ前のノードを記憶するループ `for(p=q=head; p!=NULL;　　,p=p->next)` を利用して削除するノードを定める。必要ならば、index を保持するループを利用する。削除するノードが決まったら、`q->next = p->　　;` とする。削除に要する計算量は、O(1)であるが、削除するノードを探索する必要があるので、トータルの計算量は、O（　　）。
末尾のノードを削除：**単純な**単方向リストの場合、末尾のデータは自分の前のノードを知りえないので、先頭から探索する必要がある。トータルの計算量は、O（　　）。tail を用いている場合、この処理が O (1) となる。

リストの探索：リストに対する基本ループ `for(p=head; p!=　　; p=p->next)` を利用。必要ならば、index を保持するループ `for(i=0, p=head; p!=NULL; i++,p=p->next)` を利用する。計算量は、O（　　）。

確認レポート 12

科目	テーマ	番号	氏名
アルゴリズムとデータ構造 B	線形リストの操作		解答例

問題 1　以下の文章を完成させよ。

線形リストに対する基本処理は、1 次元配列と同様に計算量が O(*n*) のアルゴリズムである。

問題 2　リストに対する処理とその計算量のオーダについて考えよ。

リストの初期化：`head=NULL;` とする。計算量は、O（ 1 ）。

挿入（新しく確保したノードを `r` とする。`r = malloc(sizeof(struct Element));`）
リストの先頭にノードを挿入：`r->next=head; head=r;` 。計算量は、O（ 1 ）。
※head からのみデータを出し入れできるようにすることで、スタック を実現できる。
リストの途中にノードを挿入：一つ前のノードを記憶するループ `for(p=q=head; p!=NULL; q=p, p=p->next)` を利用して挿入する位置を定める。必要ならば、index を保持するループを利用する。挿入する位置が決まったら、`q->next=r; r->next=p;` とする。トータルの計算量は、O（ *n* ）。
リストの末尾にノードを挿入：（基本）末尾まで先頭からたどらなければならない。
（**新たなポインタ `tail` を用いる場合**）`tail` が末尾の要素を指しているとして、`tail->next=r; tail=r; tail->next=NULL;` とする。計算量は、O（ 1 ）。
※tail からデータを入力し、head からデータを出力することで、キュー を実現できる。

削除（※領域解放時には `free()` する）
先頭のノードを削除：`head=head->next;` とする。計算量は、O（ 1 ）。
途中のノードを削除：一つ前のノードをを記憶するループ `for(p=q=head; p!=NULL; q=p, p=p->next)` を利用して削除するノードを定める。必要ならば、index を保持するループを利用する。削除するノードが決まったら、`q->next = p->next;` とする。削除に要する計算量は、O(1)であるが、削除するノードを探索する必要があるので、トータルの計算量は、O（ *n* ）。
末尾のノードを削除：**単純な**単方向リストの場合、末尾のデータは自分の前のノードを知りえないので、先頭から探索する必要がある。トータルの計算量は、O（ *n* ）。`tail` を用いている場合、この処理が O (1) となる。

リストの探索：リストに対する基本ループ `for(p=head; p!=NULL; p=p->next)` を利用。必要ならば、index を保持するループ `for(i=0, p=head; p!=NULL; i++, p=p->next)` を利用する。計算量は、O（ *n* ）。