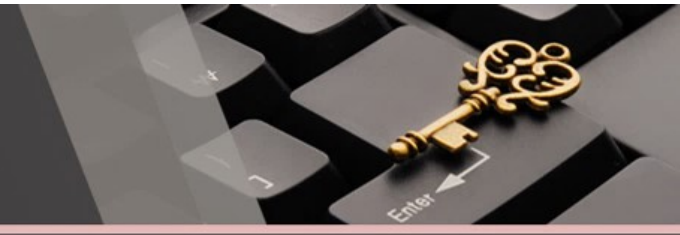
실전 알고리즘 0x04강

연결 리스트

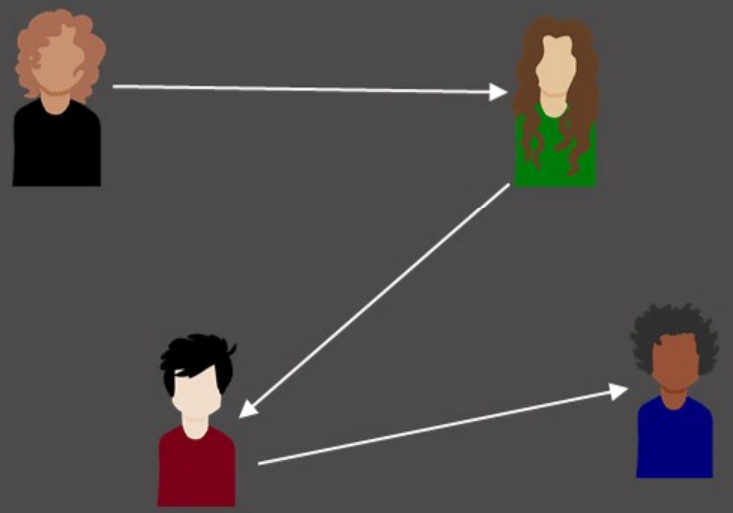
BaaaaaaaaaaaaaaaarkingDog

0x00 정의와 성질

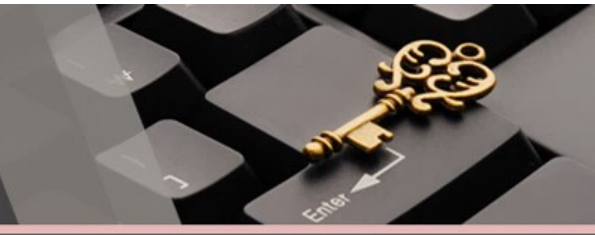


배열

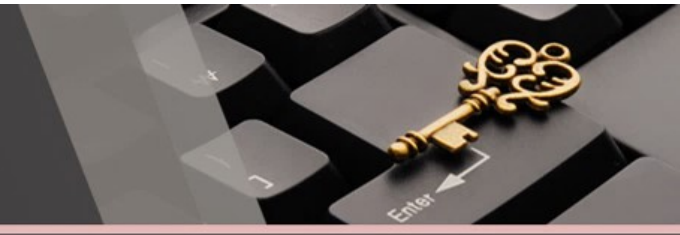
연결 리스트



목차

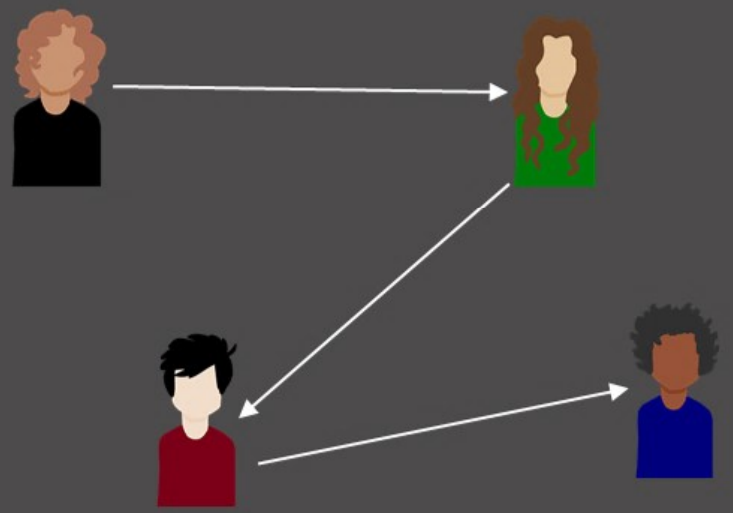


0x00 정의와 성질

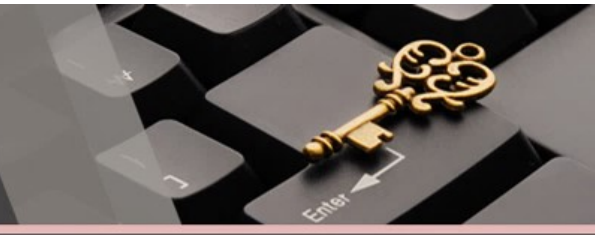


배열

연결 리스트



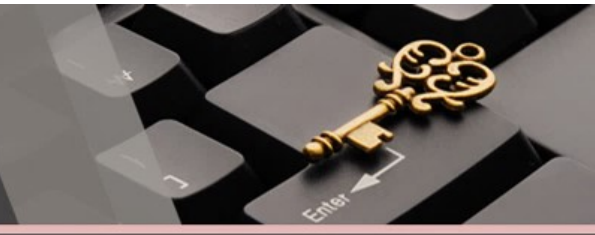
0x00 정의와 성질



연결 리스트의 성질

1. k번째 원소를 확인/변경하기 위해 O(k)가 필요함
2. 임의의 위치에 원소를 추가/임의 위치의 원소 제거는 O(1)
3. 원소들이 메모리 상에 연속해있지 않아 Cache hit rate가 낮지만 할당이 다소 쉬움

0x00 정의와 성질

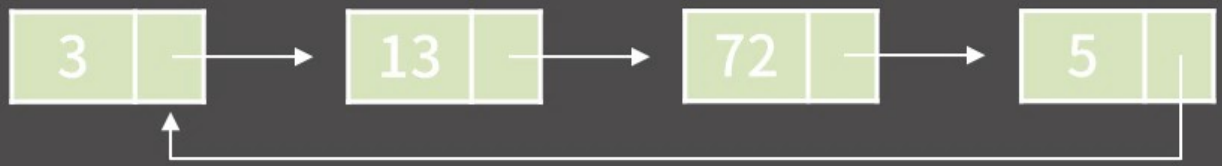


연결 리스트의 종류

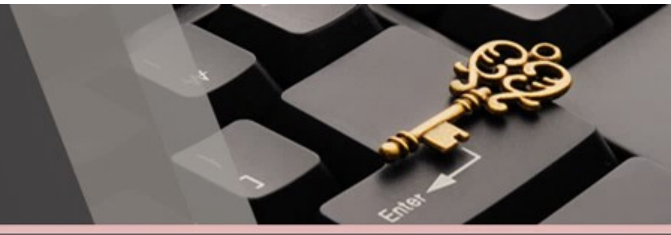
단일 연결 리스트
(Singly Linked List)

이중 연결 리스트
(Doubly Linked List)

원형 연결 리스트
(Circular Linked List)



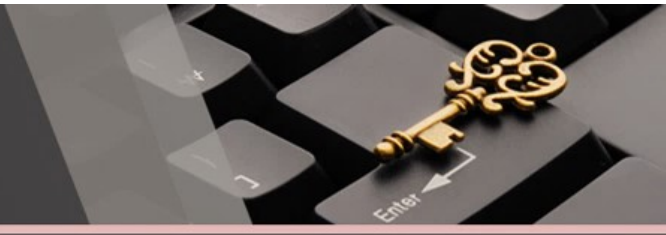
0x00 정의와 성질



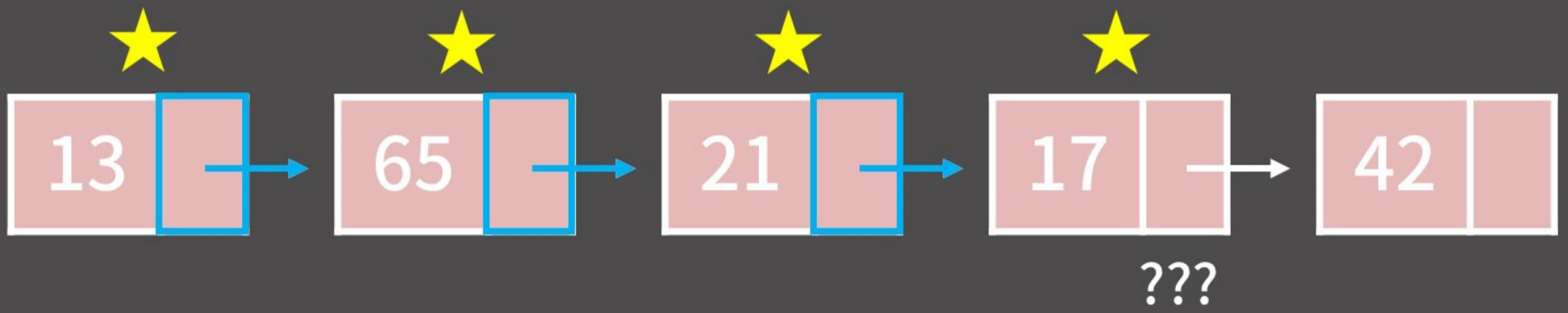
배열 vs 연결 리스트

	배열	연결 리스트
k번째 원소의 접근	O(1)	O(k)
임의 위치에 원소 추가/제거	O(N)	O(1)
메모리 상의 배치	연속	불연속
추가적으로 필요한 공간 (Overhead)	-	O(N)

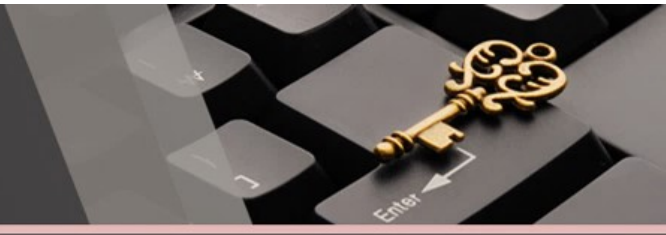
0x01 기능과 구현



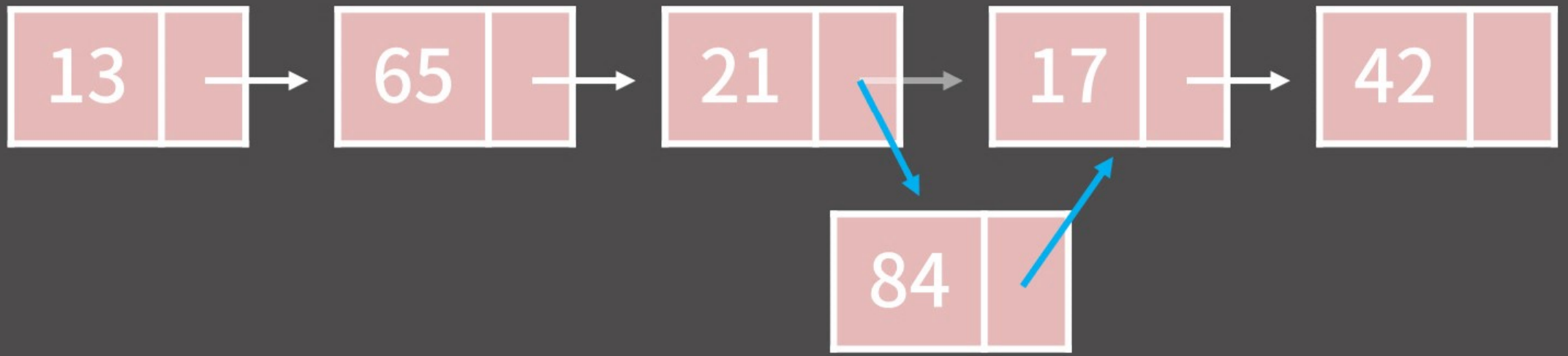
임의의 위치에 있는 원소를 확인/변경, O(N)



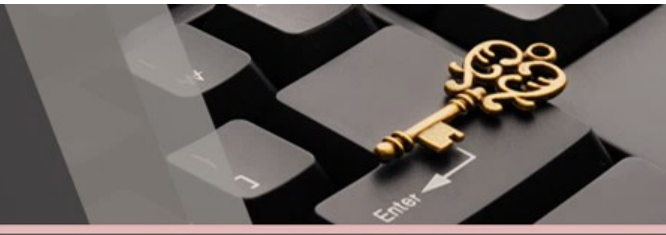
0x01 기능과 구현



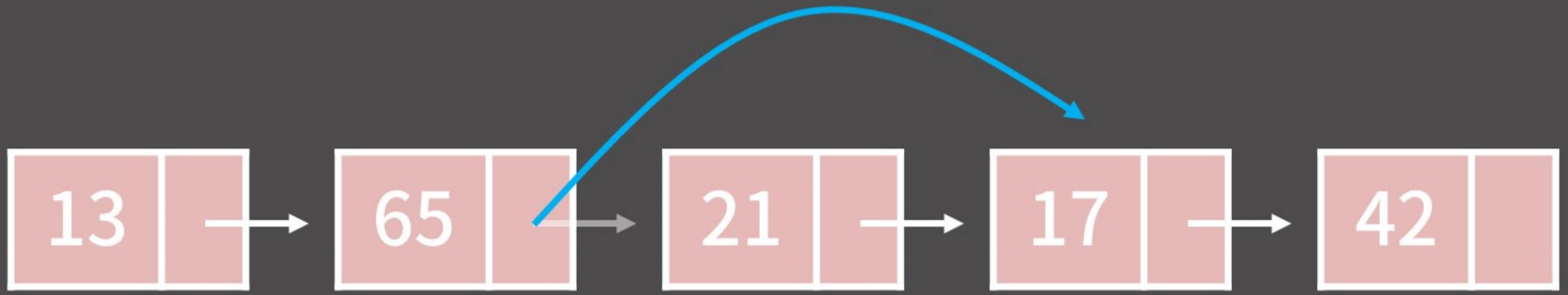
임의의 위치에 원소를 추가, O(1)



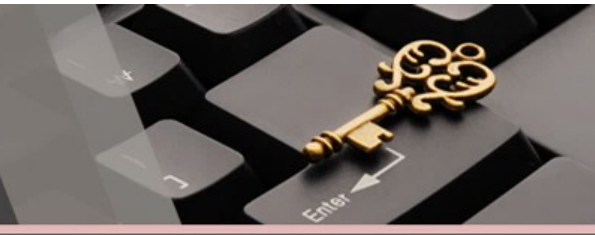
0x01 기능과 구현



임의 위치의 원소를 제거, O(1)



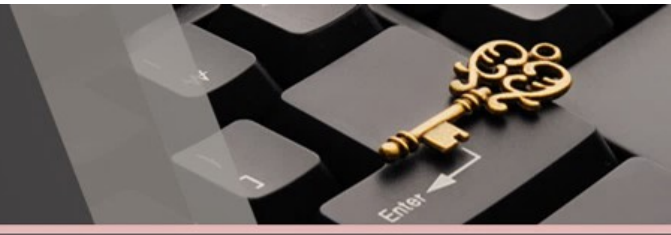
0x01 기능과 구현



임의의 위치에 있는 원소를 확인/변경 = O(N)

임의의 위치에 원소를 추가/임의 위치의 원소 제거 = O(1)

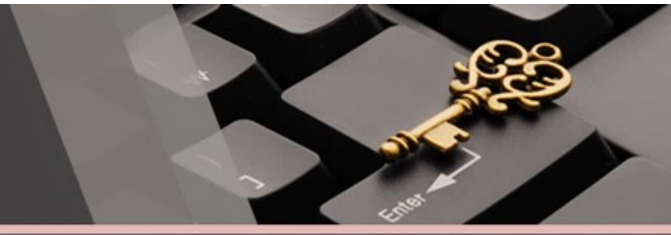
0x01 기능과 구현



연결 리스트의 구현

```
01  struct NODE {
02    struct NODE *prev, *next;
03    int data;
04  };
```

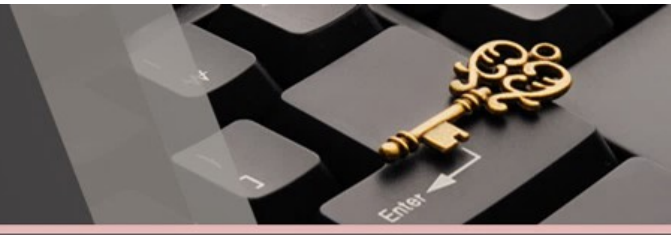
0x01 기능과 구현



야매 연결 리스트

```
01  const int MX = 1000005;
02  int dat[MX], pre[MX], nxt[MX];
03  int unused = 1;
04
05  fill(pre, pre+MX, -1);
06  fill(nxt, nxt+MX, -1);
```

0x01 기능과 구현

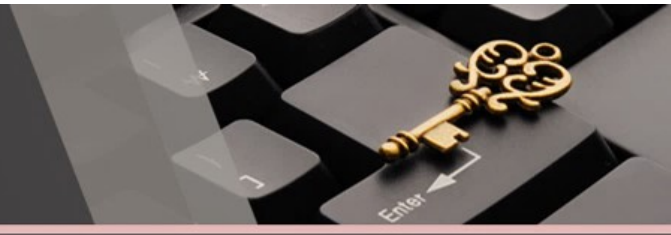


야매 연결 리스트

unused ↓

	0	1	2	3	4	5	6	7	8	9
dat	-1	65	13		21	17				
pre	-1	2	0		1	4				
nxt	2	4	1		5	-1				

0x01 기능과 구현



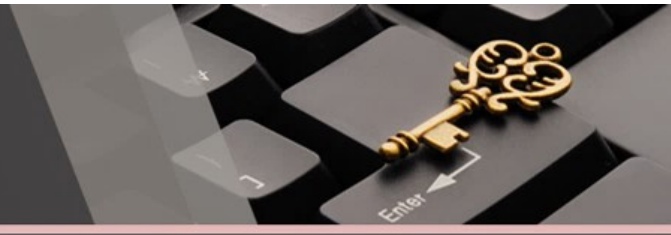
traverse 함수


```
01  void traverse(){
02    int cur = nxt[0];
03    while(cur != -1){
04      cout << dat[cur] << ' ';
05      cur = nxt[cur];
06    }
07    cout << "\n\n";
08  }
```

unused ↓

	0	1	2	3	4	5	6	7	8	9
dat	-1	65	13		21	17				
pre	-1	2	0		1	4				
nxt	2	4	1		5	-1				

0x01 기능과 구현



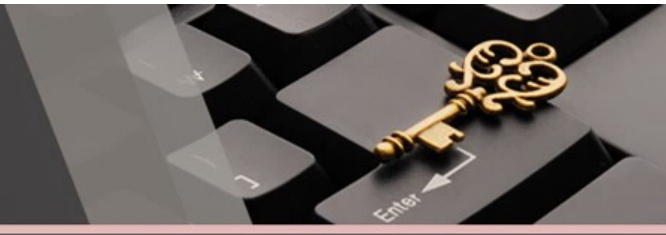
구현

https://github.com/blisstoner/basic-algo-lecture-metarial/blob/master/0x04/linked_list_test.cpp

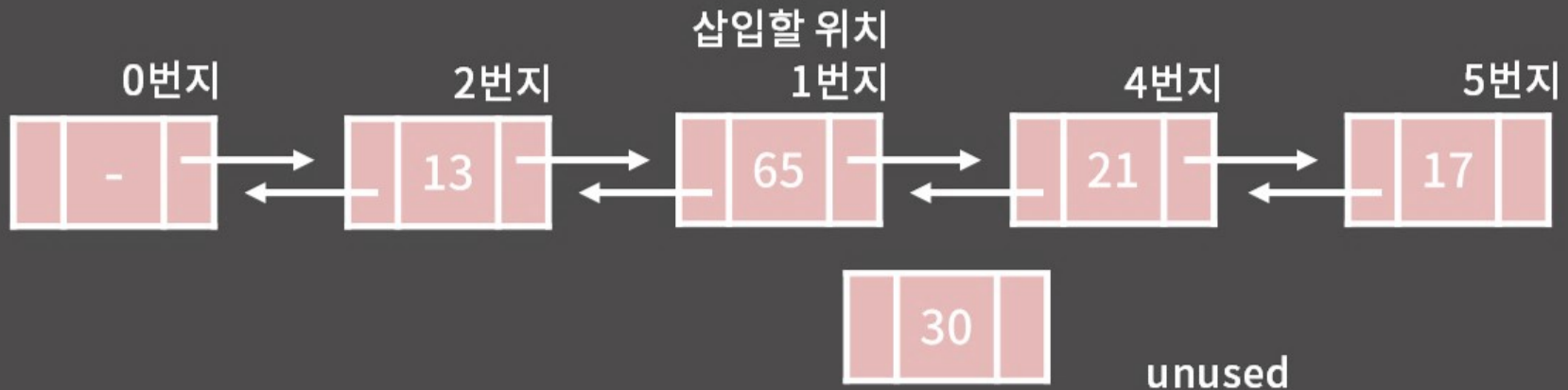
```cpp
#include <bits/stdc++.h>
using namespace std;

const int MX = 1000005;
int dat[MX], pre[MX], nxt[MX];
int unused = 1;

void insert(int addr, int num){

}

void erase(int addr){

}
```

```cpp
void traverse(){
  ...
}

void insert_test(){
  ...
}

void erase_test(){
  ...
}

int main(void) {
  fill(pre, pre+MX, -1);
  fill(nxt, nxt+MX, -1);
  insert_test();
  erase_test();
}
```

0x01 기능과 구현



insert 함수

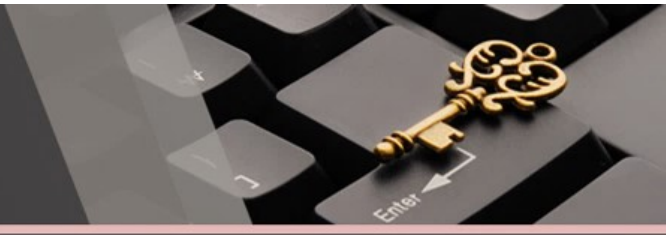


1. 새로운 원소를 생성

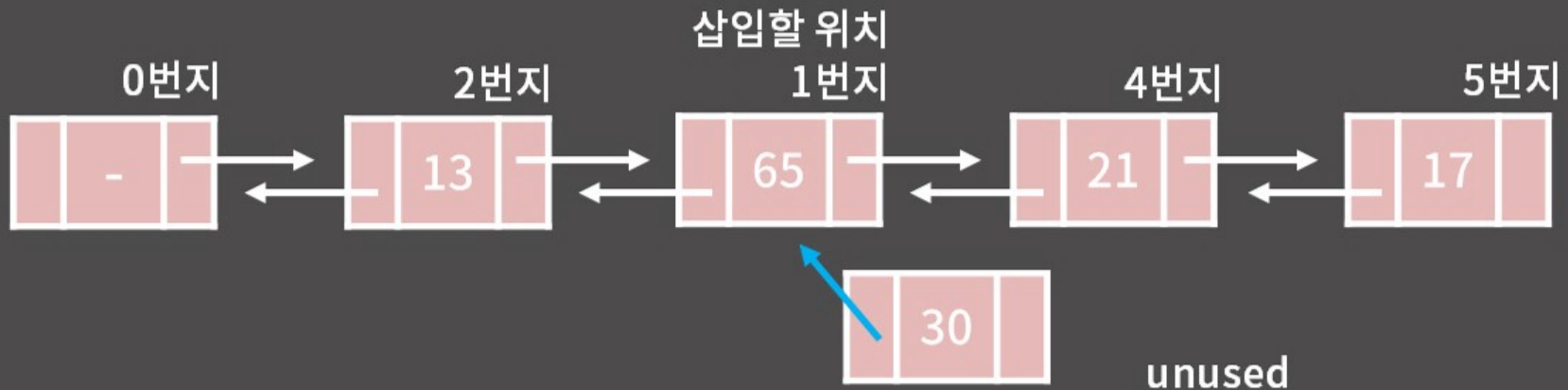
unused

	0	1	2	3	4	5	6	7	8	9
dat	-1	65	13		21	17	30			
pre	-1	2	0		1	4				
nxt	2	4	1		5	-1				

0x01 기능과 구현



insert 함수

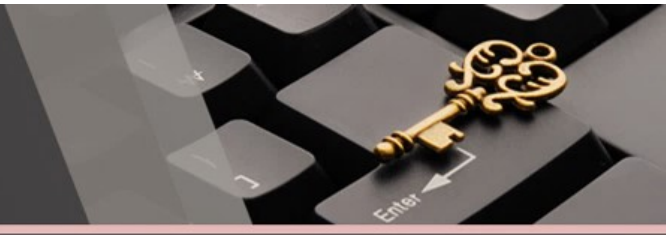


1. 새로운 원소를 생성
2. 새 원소의 pre 값에 삽입할 위치의 주소를 대입

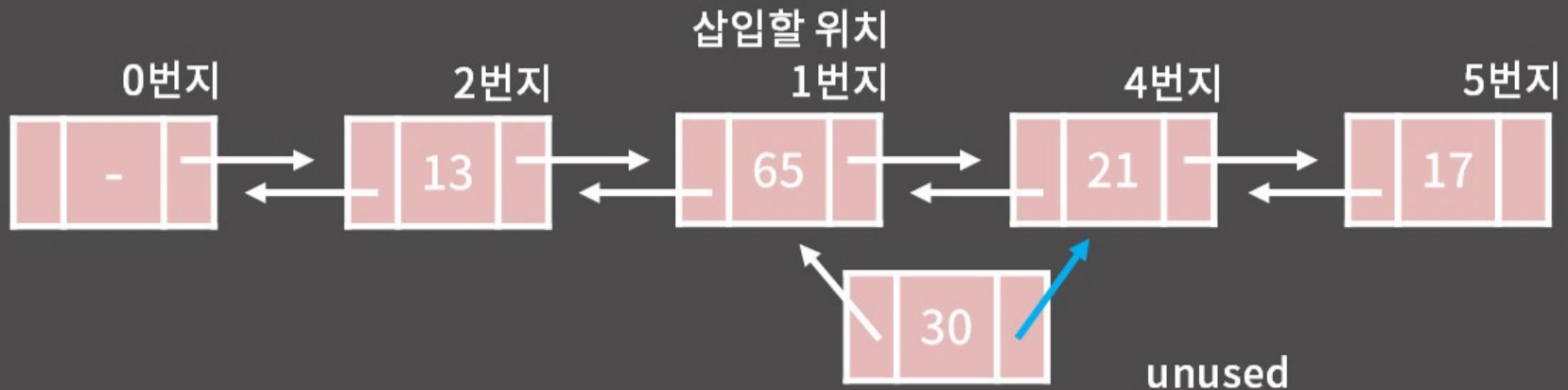
unused

	0	1	2	3	4	5	6	7	8	9
dat	-1	65	13		21	17	30			
pre	-1	2	0		1	4	1			
nxt	2	4	1		5	-1				

0x01 기능과 구현



insert 함수

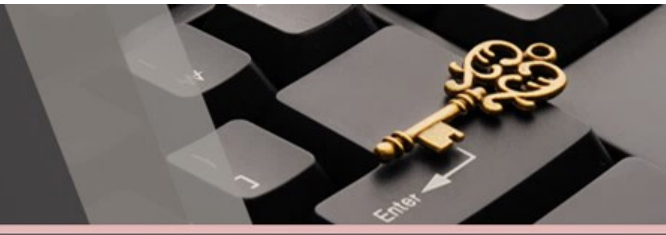


1. 새로운 원소를 생성
2. 새 원소의 pre 값에 삽입할 위치의 주소를 대입
3. 새 원소의 nxt 값에 삽입할 위치의 nxt 값을 대입

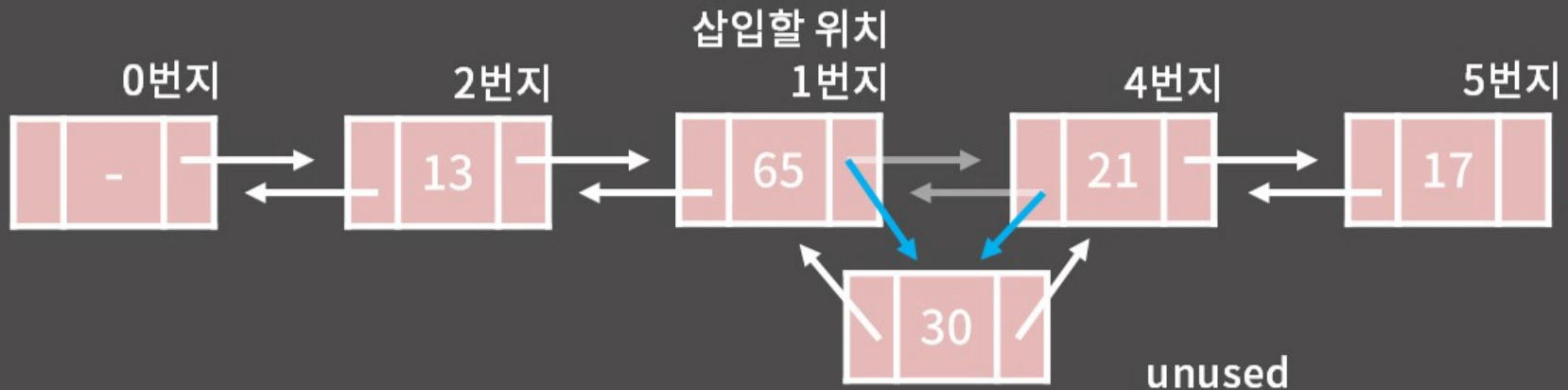
unused

	0	1	2	3	4	5	6	7	8	9
dat	-1	65	13		21	17	30			
pre	-1	2	0		1	4	1			
nxt	2	4	1		5	-1	4			

0x01 기능과 구현



insert 함수

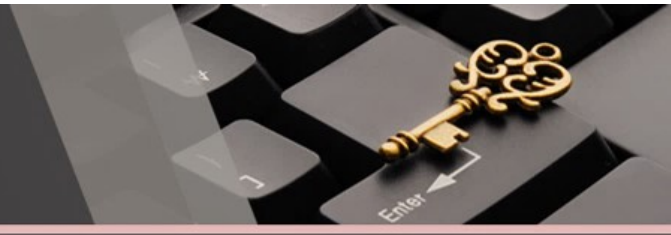


1. 새로운 원소를 생성
2. 새 원소의 pre 값에 삽입할 위치의 주소를 대입
3. 새 원소의 nxt 값에 삽입할 위치의 nxt 값을 대입
4. 삽입할 위치의 nxt 값과 삽입할 위치의 다음 원소의 pre 값을 새 원소로 변경

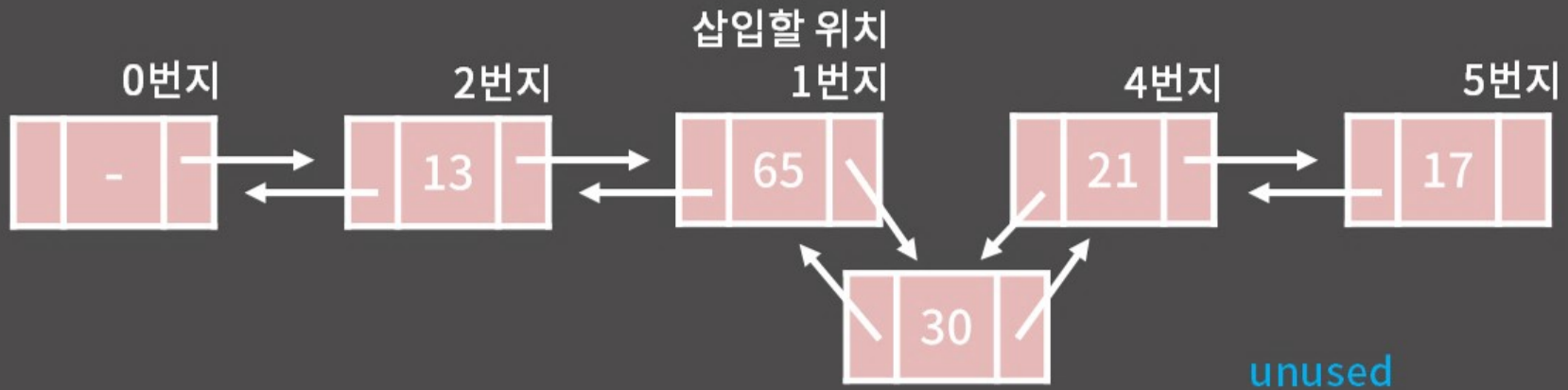
unused

	0	1	2	3	4	5	6	7	8	9
dat	-1	65	13		21	17	30			
pre	-1	2	0		6	4	1			
nxt	2	6	1		5	-1	4			

0x01 기능과 구현



insert 함수

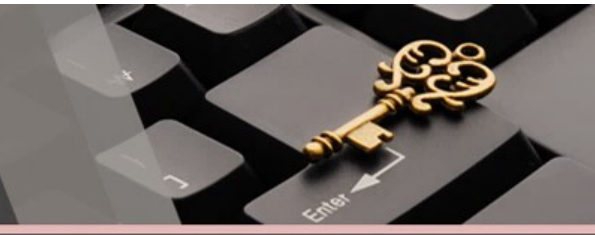


1. 새로운 원소를 생성
2. 새 원소의 pre 값에 삽입할 위치의 주소를 대입
3. 새 원소의 nxt 값에 삽입할 위치의 nxt 값을 대입
4. 삽입할 위치의 nxt 값과 삽입할 위치의 다음 원소의 pre 값을 새 원소로 변경
5. unused 1 증가

unused

	0	1	2	3	4	5	6	7	8	9
dat	-1	65	13		21	17	30			
pre	-1	2	0		6	4	1			
nxt	2	6	1		5	-1	4			

0x01 기능과 구현

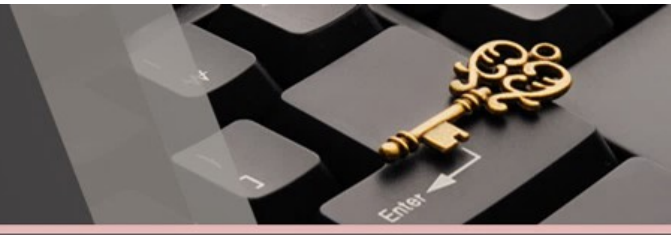


insert 함수

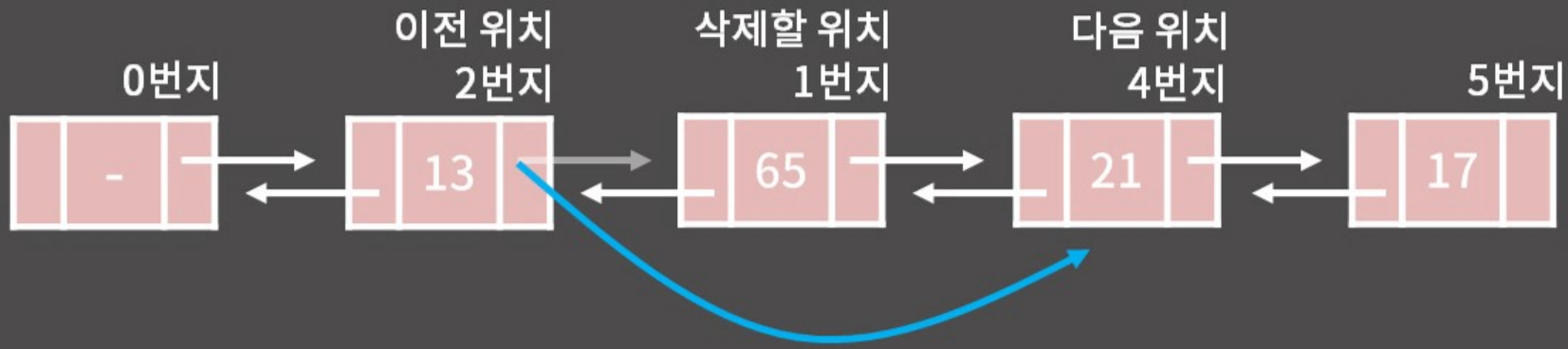
1. 새로운 원소를 생성
2. 새 원소의 pre 값에 삽입할 위치의 주소를 대입
3. 새 원소의 nxt 값에 삽입할 위치의 nxt 값을 대입
4. 삽입할 위치의 nxt 값과 삽입할 위치의 다음 원소의 pre 값을 새 원소로 변경
5. unused 1 증가

```
01  void insert(int addr, int num){
02    dat[unused] = num;
03    pre[unused] = addr;
04    nxt[unused] = nxt[addr];
05    if(nxt[addr] != -1) pre[nxt[addr]] = unused;
06    nxt[addr] = unused;
07    unused++;
08  }
```

0x01 기능과 구현



erase 함수

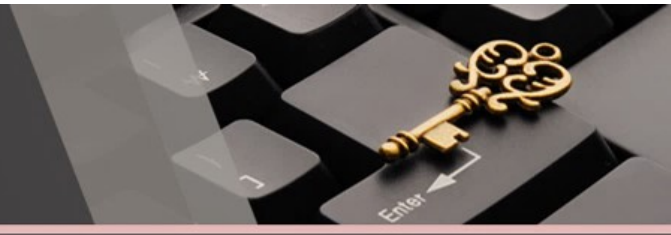


1. 이전 위치의 nxt를 삭제할 위치의 nxt로 변경

unused

	0	1	2	3	4	5	6	7	8	9
dat	-1	65	13		21	17				
pre	-1	2	0		1	4				
nxt	2	4	4		5	-1				

0x01 기능과 구현



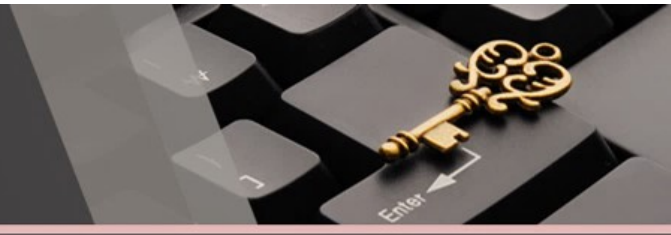
erase 함수

1. 이전 위치의 nxt를 삭제할 위치의 nxt로 변경
2. 다음 위치의 pre를 삭제할 위치의 pre로 변경

unused ↓

	0	1	2	3	4	5	6	7	8	9
dat	-1	65	13		21	17				
pre	-1	2	0		2	4				
nxt	2	4	4		5	-1				

0x01 기능과 구현



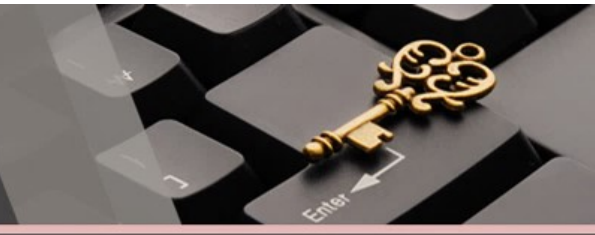
erase 함수

1. 이전 위치의 nxt를 삭제할 위치의 nxt로 변경
2. 다음 위치의 pre를 삭제할 위치의 pre로 변경

```
01  void erase(int addr){
02    nxt[pre[addr]] = nxt[addr];
03    if(nxt[addr] != -1) pre[nxt[addr]] = pre[addr];
04  }
```

https://github.com/blisstoner/basic-algo-lecture-metarial/blob/master/0x04/linked_list_test_ans.cpp

0x02 STL list

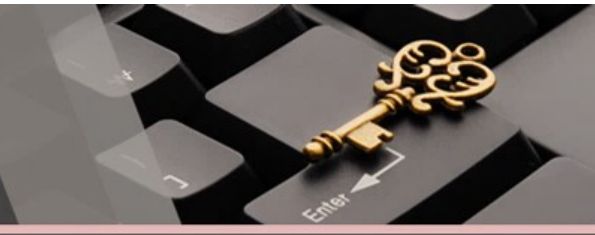


reference : http://www.cplusplus.com/reference/list/list/

https://github.com/blisstoner/basic-algo-lecture-metarial/blob/master/0x04/list_example.cpp

```
01  int main(void) {
02    list<int> L = {1,2}; // 1 2
03    list<int>::iterator t = L.begin(); // t는 1을 가리키는 중
04    L.push_front(10); // 10 1 2
05    cout << *t << '\n'; // t가 가리키는 값 = 1을 출력
06    L.push_back(5); // 10 1 2 5
07    L.insert(t, 6); // t가 가리키는 곳 앞에 6을 삽입, 10 6 1 2 5
08    t++; // t를 1칸 앞으로 전진, 현재 t가 가리키는 값은 2
09    t = L.erase(t); // t가 가리키는 값을 제거, 그 다음 원소인 5의 위치를 반환
10                    // 10 6 1 5, t가 가리키는 값은 5
11    cout << *t << '\n'; // 5
12    for(auto i : L) cout << i << ' ';
13    cout << '\n';
14    for(list<int>::iterator it = L.begin(); it != L.end(); it++)
15      cout << *it << ' ';
    }
```

0x03 연습문제

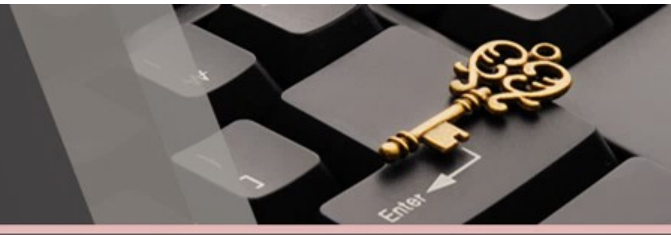


BOJ 1406번: 에디터

abcd

abcdx

0x03 연습문제

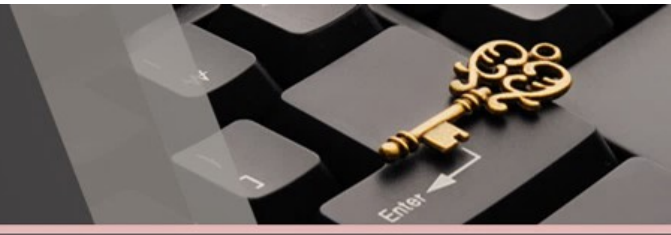


BOJ 1406번: 에디터

https://github.com/blisstoner/basic-algo-lecture-metarial/blob/master/0x04/1406_1.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(void) {
  ios::sync_with_stdio(0);
  cin.tie(0);
  string init;
  cin >> init;
  list<char> L;
  for (auto c : init) L.push_back(c);
  auto cursor = L.end();
  int q;
  cin >> q;
  while (q--) {
    char op;
    cin >> op;
    if (op == 'P') {
      char add;
      cin >> add;
      L.insert(cursor, add);
    }
    else if (op == 'L') {
      if (cursor != L.begin()) cursor--;
    }
    else if (op == 'D') {
      if (cursor != L.end()) cursor++;
    }
    else { // 'B'
      if (cursor != L.begin()) {
        cursor--;
        cursor = L.erase(cursor);
      }
    }
  }
  for (auto c : L) cout << c;
}
```

0x03 연습문제



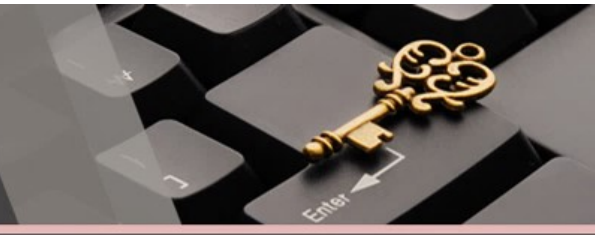
BOJ 1406번: 에디터

https://github.com/blisstoner/basic-algo-lecture-metarial/blob/master/0x04/1406_2.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MX = 1000005;
char dat[MX];
int pre[MX];
int nxt[MX];
int unused = 1;

void insert(int addr, int num){
  ...
}

void erase(int addr){
  ...
}
```

```cpp
void traversal(){
  ...
}

int main(void) {
  ios::sync_with_stdio(0);
  cin.tie(0);
  fill(pre,pre+MX,-1);
  fill(nxt,nxt+MX,-1);
  string init;
  cin >> init;
  int cursor = 0;
  for(auto c : init){
    insert(cursor, c);
    cursor++;
  }
  int q;
  cin >> q;
  while (q--) {
    char op;
    cin >> op;
    if (op == 'P') {
      char add;
      cin >> add;
      insert(cursor, add);
      cursor = nxt[cursor];
    }
    else if (op == 'L') {
      if (pre[cursor] != -1)
        cursor = pre[cursor];
    }
    else if (op == 'D') {
      if (nxt[cursor] != -1)
        cursor = nxt[cursor];
    }
    else { // 'B'
      if (cursor != 0) {
        erase(cursor);
        cursor = pre[cursor];
      }
    }
  }
  traversal();
}
```

0x03 연습문제



손코딩 문제 1

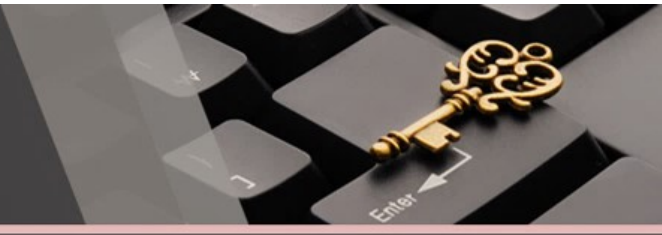
문제

원형 연결 리스트 내의 임의의 노드 하나가 주어졌을 때 해당 List의 길이를 효율적으로 구하는 방법?

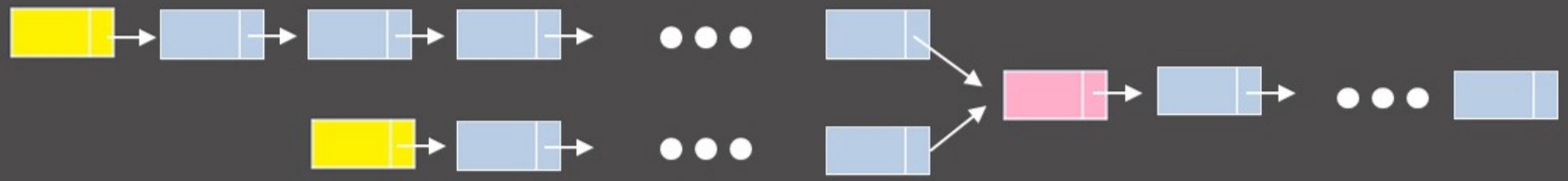
정답

동일한 노드가 나올 때 까지 계속 다음 노드로 가면 됨, 공간복잡도 O(1), 시간복잡도 O(N)

0x03 연습문제



손코딩 문제 2



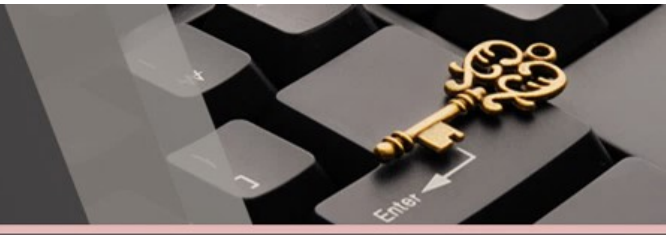
문제

중간에 만나는 두 연결 리스트의 시작점들이 주어졌을 때 만나는 지점을 구하는 방법?

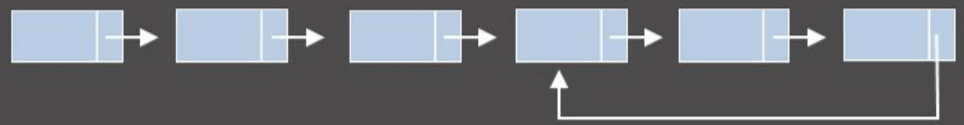
정답

일단 두 시작점 각각에 대해 끝까지 진행시켜서 각각의 길이를 구함.
그 후 다시 두 시작점으로 돌아와서 더 긴 쪽을 둘의 차이만큼 앞으로 먼저 이동시켜놓고 두 시작점이 만날 때 까지 두 시작점을 동시에 한 칸씩 전진시키면 됨. 공간복잡도 O(1), 시간복잡도 O(A+B)

0x03 연습문제



손코딩 문제 3



문제

주어진 연결 리스트 안에 사이클이 있는지 판단하라

정답

Floyd's cycle-finding algorithm, 공간복잡도 O(1), 시간복잡도 O(N)

강의 정리