2반 20131532 김승현

추가 구현 함수 및 변수

void list\_swap (struct list\_elem \*a, struct list\_elem \*b);  
 - list 엘리먼트 두 개를 스왑한다. a와 b에 연결되어 있는 링크를 서로 바꿔준다.

void list\_shuffle (struct list \*list);  
 - list의 첫 노드 (list\_begin이용)부터 마지막 노드 (list\_end)까지 난수로 생성된 수만큼 노드를 이동한 후에 둘을 스왑하는 과정을 노드의 개수만큼 반복한다.

bool compare(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux);  
 - a와 b의 데이터를 비교해서 a<b이면 true, 아니면 false를 반환한다.

unsigned hash\_int\_2(int i);  
 - 정수를 받아서 1비트 왼쪽 쉬프트 후에 4만큼 모듈러 연산으로 해쉬값을 반환한다.

unsigned hashswan(const struct hash\_elem \*el, void \*aux);  
 - hash\_init()함수를 실행할떄 필요한 함수 포인터 함수로서 해쉬를 만들기 위한 함수를 정의한다.

bool hashbrown(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux);  
 - hash의 두 엘리먼트의 데이터를 비교하여 a<b이면 true, 아니면 false을 반환한다.

void square(struct hash\_elem \*e, void \*aux);  
 - hash\_action\_func의 함수 포인터의 함수로서 hash의 엘리먼트 데이터를 제곱시켜주는 함수다.

void triple(struct hash\_elem \*e, void \*aux);  
 - hash\_action\_func의 함수 포인터의 함수로서 hash의 엘리먼트 데이터를 세제곱시켜주는 함수다.

void hashdeleter (struct hash\_elem \*e, void \*aux);  
 - hash\_action\_func의 값으로서 e의 hash\_entry를 free시킨다.

struct bitmap\* bitmap\_expand(struct bitmap\* bitmap, int size);  
 - bitmap 크기를 size만큼 확장시켜 준다.

struct list\* find\_list\_arg(char \* str)  
-List의 이름을 저장한 구조체로부터 입력받은 이름과 같은 list의 이름을 찾아 반환한다.

struct bitmap\* find\_bit\_arg(char \* str)  
- bitmap의 이름을 저장한 구조체로부터 입력받은 이름과 같은 bitmap의 이름을 찾아 반환한다.

struct hash\* find\_hash\_arg(char \* str)  
- hash의 이름을 저장한 구조체로부터 입력받은 이름과 같은 hash의 이름을 찾아 반환한다.

struct data\_name{char name[40];struct list list;};  
- list의 이름을 저장하는 구조체다.

struct bit\_name{ char name[40];struct bitmap \* bitmap;};  
 - bitmap 이름을 저장하는 구조체다.

struct hash\_name{char name[40];struct hash hash;};  
 - hash 이름을 저장하는 구조체다.

struct data\_name list\_name[10];

struct bit\_name bit\_name[10];

struct hash\_name hash\_name[10];

* 각 구조체별 이름을 저장할수 있는 구조체를 선언하여 그 type을 10개까지 저장할수 있는 배열이다. 최대 입력갯수가 10개이므로 위와같이 정의한다.

int list\_num=0;

int bit\_num=0;

int hash\_num=0;

* 구조체의 이름을 저장하는 배열의 indexing을 위한 전역변수이다.

테스트할 기존코드의 함수

void list\_init (struct list \*);  
 - create 할 때 사용한다. Parameter의 list의 head와 tail을 연결하고 빈 list를 반환한다.

struct list\_elem \*list\_begin (struct list \*);  
 - list를 인자로 넣으면 그 리스트의 시작 노드를 반환한다.

struct list\_elem \*list\_next (struct list\_elem \*);  
 - list elem에 링크된 다음 노드를 반환한다.

struct list\_elem \*list\_end (struct list \*);

- 리스트의 tail을 반환한다.

void list\_insert (struct list\_elem \*a , struct list\_elem \*b);  
 - a엘리먼트 앞에 b엘리먼트를 넣는다.

void list\_splice (struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last);  
 - before엘리먼트 앞에 first엘리먼트부터 last엘리먼트까지 넣는다. 단, first, last는 같은 list에 연결되어있어야한다.

void list\_push\_front (struct list \*, struct list\_elem \*);  
 - 리스트 맨 앞에 새로 만든 node를 넣는다.

void list\_push\_back (struct list \*, struct list\_elem \*);  
 - 리스트 맨 끝에 새로만든 node를 넣는다.

struct list\_elem \*list\_remove (struct list\_elem \*);  
 - 리스트의 엘리먼트를 지운다. 반환값은 지운 엘리먼트의 다음 엘리먼트다.

struct list\_elem \*list\_pop\_front (struct list \*);  
 - 리스트의 맨 처음 엘리먼트를 없앤다.

struct list\_elem \*list\_pop\_back (struct list \*);  
 - 리스트의 맨 뒤 엘리먼트를 없앤다.

struct list\_elem \*list\_front (struct list \*);  
 - 리스트의 맨 앞 엘리먼트를 반환한다.

struct list\_elem \*list\_back (struct list \*);  
 - 리스트의 맨 뒤 엘리먼트를 반환한다.

size\_t list\_size (struct list \*);  
 - 리스트 사이즈(엘리먼트 개수)를 반환한다.

bool list\_empty (struct list \*);  
 - 리스트가 비어있으면 true, 아니면 false를 반환한다.

void list\_reverse (struct list \*);  
 - 리스트를 뒤집어 준다.

void list\_sort (struct list \*, list\_less\_func \*, void \*aux);  
 - 리스트를 정렬해준다. List\_less\_func은 함수 포인터로서 compare함수를 인자로 사용한다. compare함수는 두 elem의 data값을 list\_entry를 이용하여 참조한 후 비교하여 준다.

void list\_insert\_ordered (struct list \*, struct list\_elem \*, list\_less\_func \*, void \*aux);  
 - 리스트를 list\_less\_func(compare함수)에 따라 정렬해서 삽입한다.

void list\_unique (struct list \*, struct list \*duplicates, list\_less\_func \*, void \*aux);  
 - 리스트 안에 중복된 데이터를 제거한다. List\_less\_func은 두 값이 같은지 판단하기 위해 사용한다.

struct list\_elem \*list\_max (struct list \*, list\_less\_func \*, void \*aux);  
 - 리스트 엘리먼트 중에 가장 큰 데이터를 가진 엘리먼트를 반환한다.

struct list\_elem \*list\_min (struct list \*, list\_less\_func \*, void \*aux);  
 - 리스트 엘리먼트 중에 가장 작은 데이터를 가진 엘리먼트를 반환한다.

bool hash\_init (struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux);  
 - 해쉬를 초기화 한다. hash\_hash\_func에 따라 해쉬인덱스가 결정된다.

void hash\_clear (struct hash \*, hash\_action\_func \*);

* 해쉬를 clear 한다. hash\_action\_func은 hashdeleter를 사용한다.

void hash\_destroy (struct hash \*, hash\_action\_func \*);  
 - 해쉬를 삭제한다.

struct hash\_elem \*hash\_insert (struct hash \*, struct hash\_elem \*);  
 - 해쉬 엘리먼트를 삽입한다. 중복된 데이터는 넣지 않는다.

struct hash\_elem \*hash\_replace (struct hash \*, struct hash\_elem \*);  
 - 해쉬 엘리먼트를 삽입한다.

struct hash\_elem \*hash\_find (struct hash \*, struct hash\_elem \*);  
 - 데이터가 일치하는 해쉬 엘리먼트를 찾아준다.

struct hash\_elem \*hash\_delete (struct hash \*, struct hash\_elem \*);  
 - 데이터가 일치하는 해쉬 엘리먼트를 삭제한다

void hash\_apply (struct hash \*, hash\_action\_func \*);  
 - 해쉬에 있는 데이터들을 제곱하거나 세제곱한다.(square,triple함수 참조)

void hash\_first (struct hash\_iterator \*, struct hash \*);  
 - 해쉬의 시작을 hash\_iterator 변수에 얻어온다.

struct hash\_elem \*hash\_next (struct hash\_iterator \*);  
 - hash\_iterator 변수로 다음 해쉬 노드를 찾아갈 때 사용한다

struct hash\_elem \*hash\_cur (struct hash\_iterator \*);  
 - hash\_iterator 변수로 현재 해쉬 노드를 찾을 때 사용한다.

size\_t hash\_size (struct hash \*);  
 - 해쉬 사이즈(엘리먼트 개수)를 반환한다

bool hash\_empty (struct hash \*);  
 - 해쉬가 비었으면 true, 아니면 false을 반환한다.

unsigned hash\_int (int);  
 - 정수를 받아서 해쉬인덱스를 반환한다.

struct bitmap \*bitmap\_create (size\_t bit\_cnt);  
 - bit\_cnt만큼 비트맵을 할당해서 반환한다.

void bitmap\_destroy (struct bitmap \*);  
 - 비트맵을 제거한다. 할당해제

size\_t bitmap\_size (const struct bitmap \*);  
 - 비트맵 크기를 반환한다.

void bitmap\_set (struct bitmap \*, size\_t idx, bool);  
 - 해당 인덱스의 값을 bool 값으로 바꾼다.

void bitmap\_mark (struct bitmap \*, size\_t idx);  
 - 해당 인덱스의 값을 1로 바꾼다.

void bitmap\_reset (struct bitmap \*, size\_t idx);  
 - 해당 인덱스의 값을 0으로 바꾼다.

void bitmap\_flip (struct bitmap \*, size\_t idx);  
 - 해당 인덱스의 비트를 반전한다.

bool bitmap\_test (const struct bitmap \*, size\_t idx);  
 - 해당 인덱스의 비트를 반환한다.

void bitmap\_set\_all (struct bitmap \*, bool);  
 - 비트맵 전체를 bool값으로 만든다.

void bitmap\_set\_multiple (struct bitmap \*, size\_t start, size\_t cnt, bool);  
 - 비트맵 일부를 bool값으로 만든다.

size\_t bitmap\_count (const struct bitmap \*, size\_t start, size\_t cnt, bool);  
 - 비트맵 일부에서 bool개수를 카운트해서 반환한다.

bool bitmap\_contains (const struct bitmap \*, size\_t start, size\_t cnt, bool);  
 - 비트맵 일부에서 bool값이 존재하면 true 아니면 false를 반환한다.

bool bitmap\_any (const struct bitmap \*, size\_t start, size\_t cnt);  
 - 비트맵 일부에서 하나의 비트라도 1이면 true 아니면 false를 반환한다.

bool bitmap\_none (const struct bitmap \*, size\_t start, size\_t cnt);  
 - 비트맵 일부에서 일부 전체가 0이면 true 아니면 false를 반환한다.

bool bitmap\_all (const struct bitmap \*, size\_t start, size\_t cnt);  
 - 비트맵 일부에서 start에서부터 cnt까지가 1이면 true 아니면 false을 반환한다.

size\_t bitmap\_scan (const struct bitmap \*, size\_t start, size\_t cnt, bool);  
 - 비트맵 일부에서 bool값의 연속의 개수를 반환한다.

size\_t bitmap\_scan\_and\_flip (struct bitmap \*, size\_t start, size\_t cnt, bool);  
 - 비트맵 일부에서 bool값의 연속의 개수만큼 비트를 반전한다.