REPORT

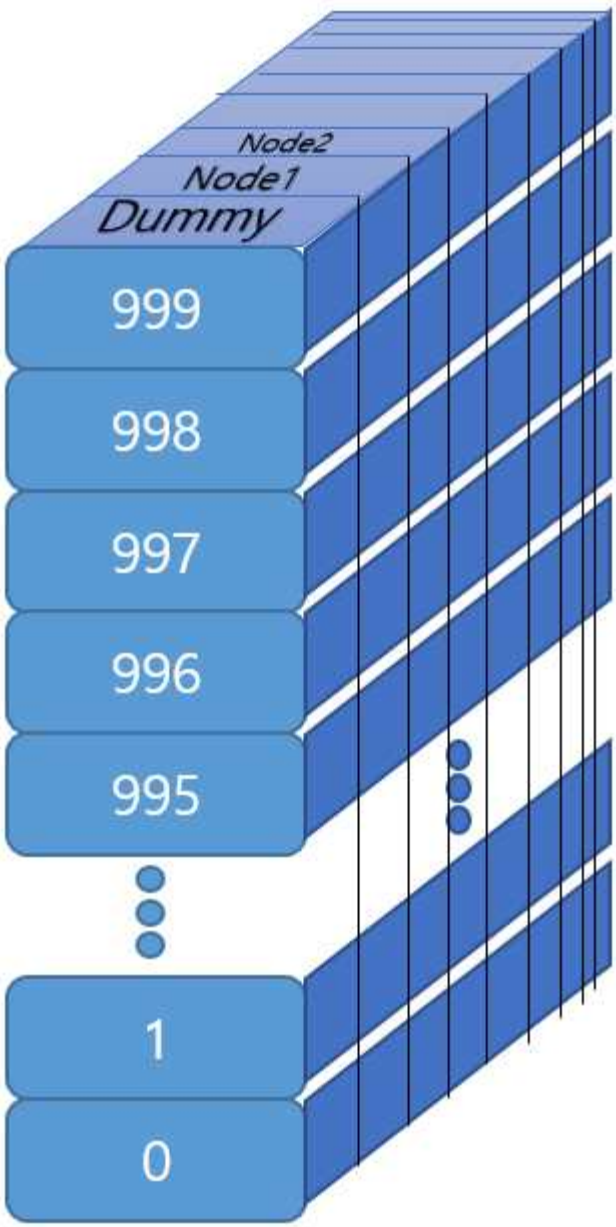
Team Project

Basis and Practice in Programming (Fall 2018)

2018314861 이현석, 2018313582 김동성

2018.12.11

[데이터 구조] - Hash Table



더욱 빠른 데이터 접근을 위해 해시테이블을 사용했습니다.

왼쪽의 그림은 해시 테이블의 구조 도입니다. 그림과 같이 0~999까지의 인덱스가 있고, 각 인덱스마다 연결리스트가 존재합니다.

각 인덱스에는 더미부터 시작, 추가하려는 노드는 각자의 해시값에 일치하는 연결리스트에 큐로 추가됩니다.

그림에 나와있는 자료구조와는 별개로, 알파벳 순서로 정렬된 연결 리스트도 존재합니다.

하나의 노드 안에 해시 테이블에서의 다음 값(next), 알파벳 순서로 정렬되었을 때 다음 값(next_alpha)에 해당하는 두 개의 포인터 값이 있는 것입니다.

이로써 추가적인 검색 알고리즘 없이 만약 찾으려는 key 값을 정확히 알고 있다면 해시값을 이용하여 빠르게 (O(1)에 수렴) 접근할 수 있습니다.

보안에 사용되는 해시값과 달리 해시 테이블에 사용되는 해시값은 여러 개의 key 값이 같은 해시값을 가질 수 있습니다. 이를 해결하기 위해 연결리스트의 배열을 사용하였습니다.
해시테이블이 생성될 때 만들어지는 리스트 배열 1000개를 모두 ListInit() 함수를 통해 초기화 하는데, 만일 사용하지 않는 리스트가 존재할 경우 필요없는 리스트가 생겨 메모리를 불필요하게 차지하지만 거의 없다는 가정 하에 이런 식으로 처리하였습니다.

[알고리즘 설명]

(1) key 값이 완벽한 경우

key 값이 만일 완벽히 주어진 경우 hash 값을 이용해 빠르게 데이터를 찾을 수 있습니다. 먼저 hash 값을 구한 뒤, 알맞은 list로 이동하여 head부터 null이 나올 때까지 모든 노드를 돌아다니며 원하는 노드를 찾습니다. 이 때 10000개의 데이터를 저장했을 때 1000개의 리스트가 있으므로 하나의 리스트에는 평균적으로 10개 정도의 노드가 존재합니다. 10000개를 일일이 찾아다니는 것과 10개만을 찾는 것에는 분명한 차이가 존재할 것입니다.

(2) key 값이 불완전한 경우

*를 통해 key값이 주어진 경우, hash 값을 이용할 수 없기 때문에 첫 노드부터 모든 노드를 찾아다닙니다. 이 경우 해당하는 모든 노드를 결과적으로 찾아야 하기 때문에 이러한 방법이 비록 효율적이라고 할 수는 없지만 확실한 프로그램을 짜는 데에는 매우 효과적일 수 있습니다.

(3) 알파벳 순으로 나열하기

데이터를 저장할 때, next_alpha라는 추가적인 노드를 통해 테이블을 관통하는 하나의 연결리스트를 추가적으로 구성합니다. 이를 통해 알파벳 순으로 나열하는 과정이 그리 어렵지 않게 이루어질 수 있습니다. 알파벳 순으로 데이터를 저장하는 방법은, strcmp() 함수의 우선순위에 따라 알맞은 데이터 위치로 이동해 노드의 사이에 데이터를 추가합니다. 이를 위해 pre와 cur 라는 2개의 노드 포인터를 사용합니다.

(4) 삭제

삭제를 하는 경우 key 값이 완벽하게 주어지기 때문에 어렵지 않게 노드를 찾을 수 있습니다. 또한 free() 함수를 통해 malloc으로 부여된 영역을 삭제합니다. 테이블의 경우 삭제되는 동시에 프로그램이 끝나기 때문에 특별한 과정을 거치지 않았습니다.

[함수 설명]

HTinit()

0~999의 인덱스를 가지는 해시 테이블을 만듭니다. 매끄러운 프로그램 작동을 위해 각 인덱스 당 하나씩 다음 노드가 저장될 메모리 영역의 포인터를 가진 더미 노드를 미리 추가해 둡니다. 새로운 key와 value가 입력되면 노드 형태로 이 더미 노드 다음부터 연결 리스트 형식으로 연결됩니다. 해시 테이블에 저장된 전체 자료의 개수를 저장하는 해시 테이블의 멤버인 numOfData를 0으로 초기화시킵니다.
위에서 설명한 해시 테이블의 자료구조와는 별개로 연결 리스트를 이용해 알파벳 순서로 다시 한 번 정렬합니다. 그를 위해서 첫 더미 값을 만듭니다. 이 더미 노드 다음부터 알파벳 순서

대로 연결 리스트 형식으로 연결합니다.

putHT()
새로 입력된 key의 해시 값에 해당하는 인덱스를 해시 테이블에서 찾아서, 그 인덱스에 해당하는 연결 리스트에 key와 value의 정보를 담고 있는 노드를 추가합니다. 해시 테이블의 멤버인 numOfData를 증가시킵니다.
또한, 연결 리스트를 이용하여 알파벳 순으로 정렬하는 작업도 합니다. 지금까지 정렬된 리스트 중 새로 입력된 노드가 들어갈 자리를 찾아서 연결합니다.

getHT()
입력된 key 값의 해시 값을 구해서 그에 해당하는 해시 테이블의 인덱스에 접근합니다. 그 인덱스에 있는 리스트에 key 값이 없다면 NULL을, key 값이 있다면 그에 해당하는 Body (value 와 key) 의 주소를 리턴합니다.

deleteHT()
입력된 key 값의 해시 값을 구해서 그에 해당하는 해시 테이블의 인덱스에 접근합니다. 그 인덱스에 있는 리스트에 key 값이 없다면 0을 리턴합니다. 해당하는 key 값이 있다면 그에 해당하는 노드를 free하고, 리스트의 멤버인 numOfData와 해시 테이블의 멤버인 numOfData를 각각 감소시킵니다. 그리고 1을 리턴합니다.

countHT()
해시 테이블의 멤버인 numOfData를 리턴합니다.

hashString()
각 문자열에 있는 캐릭터들의 아스키 값을 모두 더해서, 그것을 1000으로 나눈 나머지에 해당하는 해시 값을 구합니다. 이미 있으면 연결 리스트 형식으로 새로운 노드를 만들어 추가합니다.

searchHT()
해당 key 값에 해당하는 노드를 모두 검색하여 value의 합을 resultPos를 통해 반환합니다. 만일 해당되는 노드가 전혀 존재하지 않는다면 0을, 존재한다면 1을 반환합니다.

decompressRLE()
숫자를 이용해 표현된 문자열의 숫자를 제거하고 형식에 맞게 조절합니다.

alphaCompare()
알파벳 순으로 노드를 추가할 때 사용하는 함수입니다.

powerCompare()
숫자와 *을 이용해 표시된 문자열도 비교할 수 있는 강력한 함수입니다.