지하철의 복잡도 확인 알고리즘과 데이터 구조

장재원

Dept. of Software, Sungkyunkwan University, Suwon, South Korea

silverstick3@naver.com

Abstract – 이 논문은 지하철의 복잡도 확인 알고리즘을 만들기 위해서 배열과 linked list를 활용하여 테스트를 해보았다. 테스트를 통해 linked list가 이 알고리즘에는 더욱 뛰어난 형상을 보였다라는 것이 확인되었다. 이 논문을 대략적으로 말하자면, 처음에는 Introduction을 활용하여 이 내용을 고른 이유를 설명하고, Design and Implementation을 부분에서는 어떤 방식으로 배열과 linked list를 썼는지를 설명한다. 그리고 performance evaluation 부분에서는 결과를 제시하고, conclusion을 통해 결론을 적을 것이다.

Introduction

대한민국 직장인들은 대중교통을 통해 대부분 출퇴근을 한다. 특히 대중교통의 비율 중 지하철이 32퍼센트나 차지한다. 또한 이러한 여름에는 사람들이 가득 찬 지하철을 타게 된다면 쪄죽을 듯한 더위가 닥쳐올 것이다. 이를 최대한 피하기 위해 어떠한 방법이 없을까?라고 고민을 하였다. 자료구조를 통해 현재 지하철에 타는 승객의 명수를 안다면 지하철을 타기 전에 다른 교통수단을 이용하거나 다른 지하철역을 타게 된다면 이 무더운 여름을 그나마 덜 덥게 지낼 수 있지 않을까라는 생각을 한다.

Design and Implementation

A. System design

-왜 linked list와 sorted array를 선택했는지

지하철 복잡도 확인 알고리즘을 위해 사용한 자료구조는 배열과 linked list이다. 그 이유는 미리 sorted가 되어 있는 것과 되어 있지 않은 것의 차이점을 확인하기 위해서가 첫번째이다. 그리고 또한 트리구조는 지하철 복잡도 확인 알고리즘이 적절하지가 않다고 생각한다. 그 이유는 이용자가 이용하려는 지하철의 순서를 구하기 위해서는 이미 sorted되어서 각각 몇번째인지를 알아야하는데, 심지어 BST조차도 이 지하철은 다른 지하철보다 큰지 작은지만 알 뿐, 정확히 몇번째이다라는 것을 알기 위해서는 전체를 다 확인해야하기 때문에 시간복잡도가 더욱더 올라갈 것이라는 예상이 든 것이 두번째 이유이다. 따라서 linked list와 정렬이 된 배열을 기준으로 비교를 하는 것이 가장 적절하다고 생각한다.

-각각의 데이터 유형, 데이터 크기, 접근 패턴

Array의 각각은

```
typedef struct{
    char name[30];
    int popularity;
}subway;
```

이러한 구성으로 되어있다. 데이터 유형은 구조체로 되어있고, 데이터의 크기는 하나의 배열마다 34byte를 차지한다. 접근패턴은 array로 되어있어서 *(배열 이름 + 순서) 또는 array이름[순서]로 접근하면 된다.

Linked list 각각은

```
typedef struct pointer{
    int popularity;
    char name[30];
    struct pointer* nodepointer;
} node;
```

이러한 구성으로 되어있다. 데이터 유형은 마찬가지로 구조체로 되어있고, 데이터의 크기는 42byte로 되어있다. 접근 패턴은 앞의 배열이 뒤의 배열을 nodepointer로 가리키고 있어서 접근할 수가 있다.

또한 이 실험에서 사용한 데이터 타입은 “서울 열린데이터 광장”에서 “서울시 지하철호선별 역별 승하차 인원 정보”에 있는 매달 승하차 인원 정보를 .csv확장자 파일로 존재하는 것을 갖고 온 것이다. 그 중에서 역명과 승차총승객만을 갖고 와서 .txt 파일로 바꾼 것을 사용했다.

B. Implementation in detail

Linked list에서 각각 리스트들을 createNode라는 함수를 통해 만들었다. 해당 함수는 재귀적인 방법으로 리스트들을 생성하고 서로를 연결을 해준다. 그리고 연결이 되어있는 노드들을 readfile이라는 함수를 통해서 노드에 있는 popularity와 name에다가 각각 대입을 할 수 있다. 이는 파일 입출력 함수 중 하나인 fscanf라는 함수를 통해서 이루어진다.

```
fscanf(fp, "%s %d", list -> name, &(list -> popularity));
```

그리고 linked list는 sort를 하지 않고 원하는 지하철의 바쁜 정도를 찾는 것이 핵심이기 때문에,

```
int findNode(nodepointer list, int num, char* nickname){
    if(num <= 0 || list == NULL){
        return -1;}
    else{
        if(strcmp(nickname, list->name) == 0){
            return list -> popularity;
        }
        findNode(list -> nodepointer,num - 1, nickname);
    }}
```

findNode라는 함수를 통해서 원하는 지하철에 탑승하는 인원수를 파악하고, 그 인원수보다 적다면 popular라는 변수가 1씩 커져서 해당 지하철의 바쁜 순위를 popular로 표현해주는

```
int rank(nodepointer list, int num, int key){
    if(num <= 0 || list == NULL){
        return popular;
    }
    else{
        if(key > list->popularity)
            popular++;
    }
    rank(list -> nodepointer, num - 1, key);}
```

rank라는 함수를 통해 확인이 가능하다.

Array는 각각은 앞서 말한 구조체로 구성이 되어져 있고, 전체구성은 동적할당을 통해 이루어졌다. 그리고 배열들은 똑같이 readfile이라는 함수를 통해서 값을 popularity와 name에 받았는데 이는 파일 입출력 함수 중 하나인 fscanf를 통해 이루어졌다. 그리고 이 배열은 sorted함수를 통해 미리 정렬을 한 것이 핵심이다.

```
void sort(subway* list, int n){
    int i, j, min;
    subway temp;
    for(i = 0; i < n-1; i++) {
        min = i;
        for (j = i+1; j<n; j++){
            if(list[j].popularity <list[min].popularity){
                min = j;
            }
        }
        SWAP(list + i, list + min, temp);
    }}
```

따라서 이러한 함수를 통해서 미리 정렬을 했고 따라서 단순히 각각 배열의 이름만 확인하고, 해당하는 배열의 index에다가 1을 더하기만 하면 원하는 지하철의 복잡한 순서를 쉽게 확인이 가능해진다.

Performance Evaluation

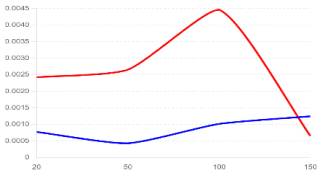
A. Experiment setup

해당 연구는 Samsung galaxy book3 Ultra라는 기종에서 진행하였다. 해당 기종은 13th Gen Intel(R) Core(TM) i9-13900H 2.60 GHz라는 프로세서를 갖고 있고, 램은 32기가가 탑재되어있다. 또한 그래픽은 RTX 4070 LAPTOP을 사용하였다. 운영체제는 리눅스를 사용하였고, 컴파일러는 gcc를 사용하였다.

B. Analysis

	Linked list		Sorted Array	
개수	입력시간	찾는 시간	입력&분류시간	찾는시간
20	0.000699	0.000072	0.002412	0.000011
50	0.000347	0.000082	0.002641	0.000013
100	0.000998	0.000016	0.004379	0.000085
150	0.001188	0.000056	0.000637	0.000012

지하철 개수와 각 실행시간



지하철의 개수와 실행시간(파란색 – linked list/ 빨간색 – Array)

표랑 그래프를 보면 지하철의 수가 600개 정도가 전부이고, 우리 주변의 지하철 개수만 생각한다면 최대 100개이기 때문에 linked list가 이 알고리즘을 실행하는데 더 적은 시간이 걸린다. 그 이유는 sorted를 통해 미리 정렬을 하는 것이 적은 데이터의 경우에는 오히려 시간을 더 키우는 요인이 되기 때문이다.

Conclusion

발전 방향 - 이 연구를 더 발전시킬 수 있는 점은 실시간으로 정보를 받을 수 있다면 더욱 더 현실성이 있을 것 같다.

결론 - 이 연구를 통해 지하철의 복잡도 확인 알고리즘을 구현할 때는 데이터 베이스가 애초에 600개정도가 최대이고, 실사용에서는 사람들이 자신의 주위에 있는 지하철을 이용하기 때문에 50개이상을 넘지 않을 것이라고 확인이 된다. 그러면 오히려 sorted를 통해 정렬을 하는 것이 더욱 시간을 길게 하는 요인이 될 수 있다는 것을 확인을 했다. 따라서 linked list와 sorted array 중 linked list를 사용하는 것이 더 낫다는 것을 확인 것에 대해 의의가 있다.