백준베스트 컨퍼런스

알고리즘을 이용한 코드 에디터 제작기

김성현(마녀)

코드 에디터 제작기 - 목차

어째서 이런 프로젝트를 했는가?
또 무엇을 사용하여 진행했는가?

코드 에디터에 필요한 기능은 무엇이며
거기서 무엇을 배웠는가?

알고리즘을 사용할 수 있는 부분이나
기능은 무엇이고 어떻게 사용했는가?

코드 에디터 제작 계기

- 2021년, 무난하다는 평을 믿고 양지훈 교수의 자료구조 수업 수강

Ⅱ. 교과목표(Course Objectives)

지식:
1) 자료구조의 종류 및 필요성에 대한 이해
2) linked list, tree, graph 등 대표적인 구조에 대한 이해
3) 자료구조를 이용하는 알고리즘에 대한 이해

- 링크드 리스트, 트리, 그래프 등에 대해 배운다는 강의계획서를 보고 꿀일 거라 생각하며 신청

코드 에디터 제작 계기

- 2021년, 무난하다는 평을 믿고 양지훈 교수의 자료구조 수업 수강

Ⅱ. 교과목표(Course Objectives)

지식:
1) 자료구조의 종류 및 필요성에 대한 이해
2) linked list, tree, graph 등 대표적인 구조에 대한 이해
3) 자료구조를 이용하는 알고리즘에 대한 이해

- 링크드 리스트, 트리, 그래프 등에 대해 배운다는 강의계획서를 보고 꿀일 거라 생각하며 신청

지금까지 배운 알고리즘을 사용해서 ‘자유롭게’ ‘재미있게’ 학기말까지 프로젝트를 하나 만들어 보세요.

코드 에디터 제작 계기

- 2021년, 무난하다는 평을 믿고 양지훈 교수의 자료구조 수업 수강

Ⅱ. 교과목표(Course Objectives)

지식:
1) 자료구조의 종류 및 필요성에 대한 이해
2) linked list, tree, graph 등 대표적인 구조에 대한 이해
3) 자료구조를 이용하는 알고리즘에 대한 이해

- 링크드 리스트, 트리, 그래프 등에 대해 배운다는 강의계획서를 보고 꿀일 거라 생각하며 신청

지금까지 배운 알고리즘을 사용해서 ‘자유롭게’ ‘재미있게’ 학기말까지 프로젝트를 하나 만들어 보세요.

여러분의 선배들은 게임을 만든 사람도 있었고, 간단한 모바일 앱을 만든 사람도 있었습니다. 하지만 핵심은 수업에서 배운 알고리즘을 사용해야 한다는 것을 잊지 마세요.

코드 에디터 제작 계기

- 백준푸는방에서 프로젝트 아이디어를 수집하던 중 '머리좀감고다녀라'(일명 '머리')의 조언에 따라 코드 에디터를 제작하기로 결정

- 머리는 편집/저장 기능과 신택스 하이라이팅 같은 걸 넣어 주면 교수가 오줌을 지리며 만점을 줄 거라고 말했음

- 또 코드 에디터에는 흔히 자동완성이나 텍스트 검색과 같은, 알고리즘이 쓰이거나 알고리즘으로 최적화할 수 있는 기능들이 있기 때문에 자료구조 수업에서 배운 내용을 사용하기도 쉬울 거라고 판단

코드 에디터 제작 계기

- 마침 인터넷에 원시적인 코드 에디터를 스스로 제작하는 튜토리얼이 있었음 (https://viewsourcecode.org/snaptoken/kilo/)

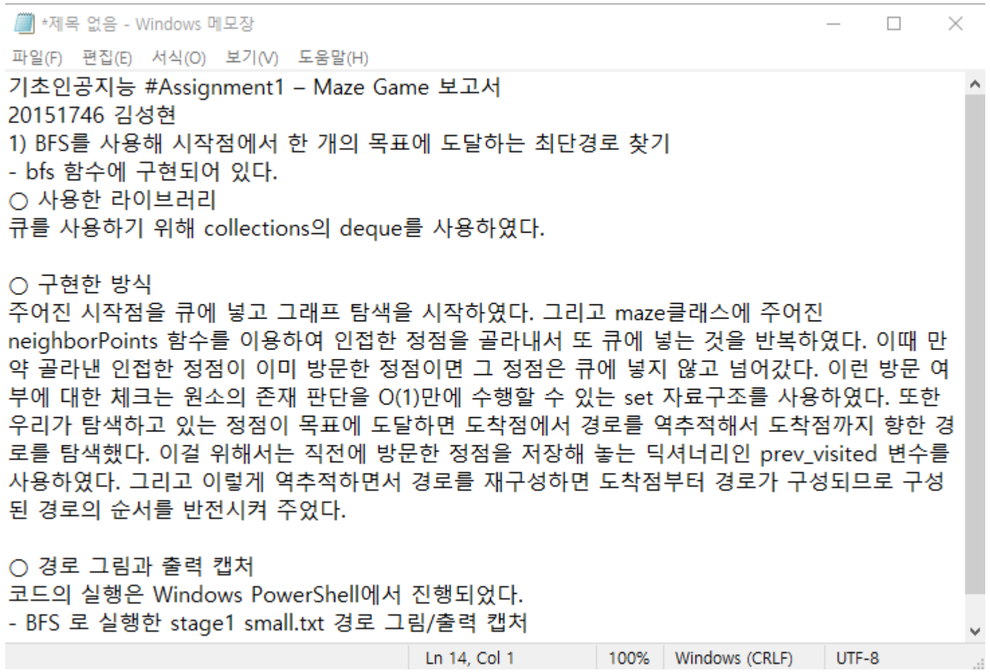
- 정말 원시적인 기능만 갖춘 에디터였고 라이브러리를 전혀 사용하지 않고 짜다 보니 직관적이지 못한 코드도 많았음

- 따라서 ncurses를 이용하여 좀더 코드를 직관적으로 짠 후 알고리즘을 이용한 최적화를 적용하기로 함

코드 에디터 기초 기능

아주 기본적인 텍스트 에디터인 메모장에
무슨 기능이 있는지를 생각해 보자.

- 파일을 열고 닫기
- 키보드를 통한 문서 편집(내용 지우기, 내용 추가 등의 기능 구현이 필요하다)
- 창에 다 들어오지 않는 문서 내용을 스크롤을 통해 아래위로 왔다갔다하며 볼 수 있는 것
- 커서의 이동과 줄번호, 문서 상태 표시
- 내용 검색



코드 에디터 기초 기능

그럼 기본적인 텍스트 에디터가 아니라 코드 에디터에는 무엇이 더 있는가?

- 파일의 타입을 감지하여 적절한 대처
- 파일 타입과 그 언어의 문법에 따른 신택스 하이라이팅
- 변수명/함수명 자동 완성
- 괄호 쌍을 체크하는 기능

⇒늘 당연한 듯이 사용하던 코드 에디터에도 많은 기능이 필요하며 간단한 동작 같아도 그것의 구현을 위해서는 그 동작을 분해해서 볼 수 있는 시각이 필요했다.

예를 들어 파일에 새로운 글자 하나를 입력하는 행위를 구현할 때도 많은 함수가 필요.

코드 에디터 알고리즘 최적화

기존 코드에도 검색 기능은 있었다. 그러나 naïve한 O(nm) 방식의 검색이었다.
따라서 O(n+m)에 문자열 탐색을 할 수 있는 KMP 알고리즘을 활용한 최적화를 적용했다.

```
void fail_func(char* pat) {
	int i, k;
	int n = strlen(pat);
	for (i = 0; i < 2; i++) {
		fail[i] = realloc(fail[i], sizeof(char)*n);
	}

	/* 왼쪽-> 오른쪽 방향의 실패함수 */
	fail[0][0] = -1;
	for (k = 1; k < n; k++) {
		i = fail[0][k - 1];
		while (pat[k] != pat[i + 1] && i >= 0) {
			i = fail[0][i];
		}
		if (pat[k] == pat[i + 1]) { fail[0][k] = i + 1; }
		else { fail[0][k] = -1; }
	}

	/* 오른쪽 -> 왼쪽 방향의 실패함수 */
	fail[1][n - 1] = n;
	for (k = n - 2; k >= 0; k--) {
		i = fail[1][k + 1];
		while (pat[k] != pat[i - 1] && i < n) {
			i = fail[1][i];
		}
		if (pat[k] == pat[i - 1]) { fail[1][k] = i - 1; }
		else { fail[1][k] = n; }
	}
}
```

```
char* KMP_match(char* string, char* pat, int* i, int direction) {
	/* string의 (*i)번째 인덱스에서 탐색 시작 */
	int k;
	int lens = strlen(string);
	int lenp = strlen(pat);

	if (direction == 1) {
		/* 오른쪽 방향 탐색 */
		k = 0;
		while ((*i) < lens && k < lenp) {
			if (string[*i] == pat[k]) {
				(*i)++; k++;
			}
			else if (k == 0) {
				(*i)++;
			}
			else {
				k = fail[0][k - 1] + 1;
			}
		}

		return(k == lenp) ? (string + (*i) - lenp) : NULL;
	}

	else {
		/* 왼쪽 방향 탐색 */
		k = lenp - 1;
		while ((*i) >= 0 && k >= 0) {
			if (string[*i] == pat[k]) {
				(*i)--; k--;
			}
			else if (k == lenp - 1) {
				(*i)--;
			}
			else {
				k = fail[1][k + 1] - 1;
			}
		}
		return (k == -1) ? (string + (*i) + 1) : NULL;
	}
}
```

코드 에디터 기능에 맞게 수정한 사항

- 순방향 검색과 역방향 검색이 모두 가능하게 하기 위해서 두 방향의 실패함수를 만들었다.

- 그리고 KMP 탐색을 하는 함수에서 어떤 방향으로 탐색할지를 인자로 받아 탐색을 진행하도록 했다.

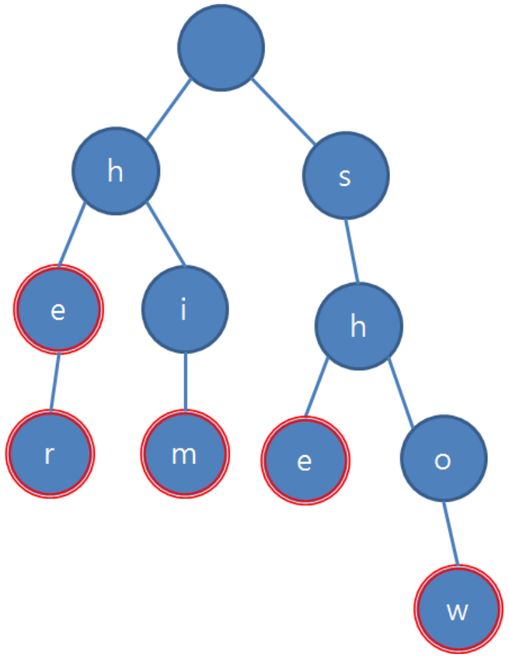
코드 에디터 알고리즘 최적화

코드 에디터에는 흔히 함수명이나 변수명을 자동완성해주는 기능이 포함되어 있다.
이 자동완성 기능을 위해서 트라이(trie) 자료구조를 사용하였다.

트라이는 문자열을 한 글자씩 쪼개서 첫 글자부터 다음 글자로 간선이 이어지는 트리의 형태로 문자열을 저장하는 자료구조이다. 저장하고 있는 단어의 끝이 되는 글자의 노드마다 특정한 표시를 해서 단어의 끝을 나타낸다.

트라이 자료구조를 사용하면 많은 문자열들이 저장되어 있는 상태에서 특정 문자열을 탐색하기가 용이하다.

he
she
her
him
show



그림으로 표현한 트라이. 출처는 kks227 블로그

이 자료구조를 응용하면 자동완성 기능도 구현 가능하다.

특정 접두사를 트라이에서 탐색한 후 접두사의 마지막 문자를 저장하고 있는 노드에서 DFS를 진행하여 완성할 수 있는 단어들을 모두 찾으면 그것이 그 접두사에서 자동완성될 수 있는 후보 단어들이다.

이때 트라이의 구성은 신택스 하이라이팅 관련 코드에서 키워드를 탐색하는 부분이 있는데 그때 함수명과 변수명에 해당하는 문자열들을 파싱하여 트라이에 삽입하는 방식으로 이뤄진다.

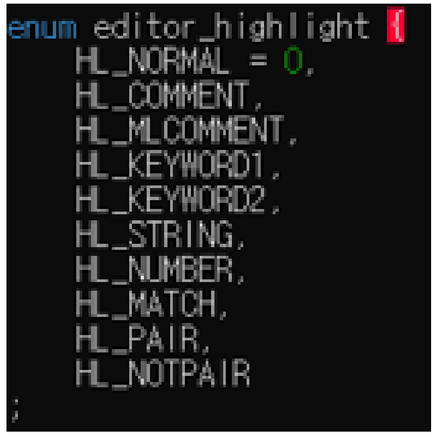
```
#define COLOR_SKYBLUE 14
#define COLOR_IVORY 15


/* trie structure for autocomplete */

#define CHAR_SIZE 96
/* 32~127 ASCII code is used. */
#define CHAR_TO_INDEX(c) ((int)c - (int)' ')

typedef struct trie {
    int is_leaf;
    int branch;
    int words;
```

```
/*** autocomplete functions ***/

void suggestion_by_prefix(trie* root, char* prefix) {
    //find the words starts with prefix string, with length len
    int i;

    if (root->is_leaf) {
        insert_list(prefix, strlen(prefix));
    }

    if (has_children(root) == 0) {
        //no child. That is, no word in trie starts with this prefix
        return;
    }

    for (i = 0; i < CHAR_SIZE; i++) {
        if (root->ch[i]) {
            prefix = string_append(prefix, ' ' + i);
            suggestion_by_prefix(root->ch[i], prefix);
            prefix = string_pop_back(prefix);
        }
    }
}
```

그리고 C 등의 언어를 사용할 때를 위해, 특정 괄호의 쌍이 되는 괄호가
어디 있는지를 판별할 수 있는 기능을 만들었다.

이러한 괄호 쌍 검사는 스택을 이용해서 간단히 진행할 수 있음이 알려져 있다.

유의할 점은 여는 괄호 { 의 짝을 찾는 경우만 있는 것이 아니라는 것이다.
닫는 괄호 } 의 짝을 찾아야 할 경우도 있으므로 경우에 따라서는 텍스트의
역방향으로 괄호 쌍 검사를 진행해야 하는 경우도 있음에 주의해서 구현했다.



코드 에디터 정리

코드 에디터에 있을 법한 기능들을 C언어와 ncurses 라이브러리를 이용해서 구현하였다.

기본적인 문서 편집과 저장, 스크롤 기능, 현재 줄번호, 신택스 하이라이팅 기능 구현

자료구조 수업에 흔히 배울 법한 내용들을 사용해서 기능 구현 혹은 최적화를 했다.
- KMP 알고리즘을 이용한 검색 최적화
- 스택 자료구조를 이용한 괄호쌍 검사 기능
- 트라이 자료구조, DFS를 이용한 자동완성 기능

배운 내용들을 그대로 사용하는 것이 아니라 상황에 맞게 응용하여 사용하는 경험을 했다.