Fast campus

컴퓨터공학 All in One

C/C++ 문법, 자료구조 및 심화 프로젝트 (나동빈)
제 44강 - 라빈 카프 문자열 매칭

Fast campus

학습 목표

라빈 카프 문자열 매칭

1) 해시(Hash)를 활용하는 알고리즘인 라빈 카프 문자열 매칭 알고리즘에 대해서 이해할 수 있습니다.
2) 평균적으로 $O(N+M)$ 의 시간 복잡도를 가지는 알고리즘입니다.

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭

1) 아스키 코드 기반의 해시 함수(Hash Function)을 이용하여 특정한 문자열에 대한 해시 값을 구합니다.
2) '연속적인 문자열이 이어지는' 상황이므로 해시 함수의 동작에 있어서 연산 속도가 $O(1)$입니다.

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭: 해시 함수

라빈 카프 문자열 매칭 알고리즘에서 해시 함수는 '각 문자의 아스키 코드 값에 2의 제곱 수를 차례대로 곱하여 더한 값'을 구합니다. 일반적으로 서로 다른 문자열의 경우 일반적으로 해시 값이 다르게 나옵니다.

해시 함수에 기반한다는 점에서 해시 충돌에 대한 처리가 필요합니다.

Fast campus

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭: 해시 함수의 눈사태 효과

문자열: “abacdab”

$$= 97 * 2^6 + 98 * 2^5 + 97 * 2^4 + 99 * 2^3 + 100 * 2^2 + 97 * 2^1 + 98 * 2^0$$

= 12,380

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭: 해시 함수의 눈사태 효과

문자열: “**b**bacdab”

$= \mathbf{98} * 2^6 + 98 * 2^5 + 97 * 2^4 + 99 * 2^3 + 100 * 2^2 + 97 * 2^1 + 98 * 2^0$

= 12,444

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭: 해시 함수

라빈 카프 문자열 매칭 알고리즘에서는 해시 충돌을 감안해야 한다는 점에서 부분 문자열의 해시 값이 일치하는 경우에만 문자열을 재검사합니다.

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭

부모 해시 값: 12,398

패턴 해시 값: 12,380

부모 문자열	a	c	a	b	a	c	d	a	b	a	c
패턴 문자열	a	b	a	c	d	a	b				

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭

부모 해시 값: 12,477

패턴 해시 값: 12,380

부모 문자열	a	c	a	b	a	c	d	a	b	a	c
패턴 문자열	a	b	a	c	d	a	b				

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭

부모 해시 값: 12,380 =〉 **문자열 매칭 발생!**

패턴 해시 값: 12,380

부모 문자열	a	c	a	b	a	c	d	a	b	a	c
패턴 문자열	a	b	a	c	d	a	b				

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭

부모 해시 값: 12,441

패턴 해시 값: 12,380

부모 문자열	a	c	a	b	a	c	d	a	b	a	c
패턴 문자열	a	b	a	c	d	a	b				

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭

부모 해시 값: 12,437

패턴 해시 값: 12,380

부모 문자열	a	c	a	b	a	c	d	a	b	a	c
패턴 문자열	a	b	a	c	d	a	b				

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭: 해시 함수의 반복적 계산

해시 함수를 반복적으로 계산할 때는 '문자열이 이어져 있다는 특징'을 이용하여 빠르게 계산합니다.

해시 값: 12,398
다음 해시 값 = 2 * (현재 해시 값 - 가장 앞에 있는 문자의 수치) + 새 문자의 수치

부모 문자열	a	c	a	b	a	c	d	a	b	a	c

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭: 해시 함수의 반복적 계산

해시 함수를 반복적으로 계산할 때는 '문자열이 이어져 있다는 특징'을 이용하여 빠르게 계산합니다.

해시 값: 12,477

부모 문자열	a	c	a	b	a	c	d	a	b	a	c

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭: 문자열 선언 및 확인 함수 구현하기

```c
#include <stdio.h>
#include <string.h>

char *parent = "acabacdabac";
char *pattern = "abacdab";

void check(char* parent, char *pattern, int start) {
  int found = 1;
  int patternSize = strlen(pattern);
  for (int i = 0; i < patternSize; i++) {
    if (parent[start + i] != pattern[i]) {
      found = 0;
      break;
    }
  }
  if (found) {
    printf("[인덱스 %d에서 매칭 발생]\n", start + 1);
  }
}
```

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭: 라빈 카프 알고리즘 구현하기

```
void rabinKarp(char* parent, char *pattern) {
  int parentSize = strlen(parent);
  int patternSize = strlen(pattern);
  int parentHash = 0, patternHash = 0, power = 1;
  for (int i = 0; i <= parentSize - patternSize; i++) {
    if (i == 0) {
      for (int j = 0; j < patternSize; j++) {
        parentHash = parentHash + parent[patternSize - 1 - j] * power;
        patternHash = patternHash + pattern[patternSize - 1 - j] * power;
        if (j < patternSize - 1) power = power * 2;
      }
    }
    else {
      parentHash = 2 * (parentHash - parent[i - 1] * power) + parent[patternSize - 1 + i];
    }
    if (parentHash == patternHash) {
      check(parent, pattern, i);
    }
  }
}
```

라빈 카프 문자열 매칭

라빈 카프 문자열 매칭: 라빈 카프 알고리즘 사용해보기

```
int main(void) {
  rabinKarp(parent, pattern);
  system("pause");
}
```

배운 내용 정리하기

라빈 카프 문자열 매칭

1) 라빈 카프 문자열 매칭 알고리즘은 평균적으로 $O(N + M)$의 시간 복잡도를 가집니다.
2) 라빈 카프 알고리즘은 때에 따라서는 KMP 알고리즘보다 빠르게 동작하기도 합니다.
3) 하지만 라빈 카프 알고리즘은 특정한 상황에서 KMP 알고리즘보다 매우 느리게 동작할 수 있습니다.