Hashing + KMP

나정휘
https://justiceHui.github.io/

목차

- Hashing
- KMP

Hashing

Hashing

- 해싱
  - 해시 함수
    - 임의의 길이를 갖는 임의의 데이터를 고정된 길이의 데이터로 매핑하는 단방향 함수
  - 오늘은 해싱을 이용한 문자열 문제 해결 방법에 대해 다룸
    - f : string → int 꼴의 해시 함수 사용

  - 해시 함수의 조건
    - 여러 번 실행해도 같은 결과가 나온다.
    - 데이터가 다르면 결과값이 다를 확률이 높다.
      - 임의의 길이의 데이터를 고정된 길이의 데이터로 매핑하기 때문에 1:1 대응은 안 될 수 있음
      - $S \neq T$이면서 $f(S) = f(T)$인 경우를 해시 충돌이라고 부름

Hashing

- 해싱
  - 해싱으로 할 수 있는 것
    - 문자열 비교 : $O(1)$
    - 문자열 S에서 문자열 P 검색 : $O(|S| + |P|)$
    - 문자열 사전순 비교 : $O(\log |S|)$

    - 기타 등등
    - 뒤에서 자세하게 알아봅시다.

Hashing

- 해싱
  - 10진법
    - 10개의 숫자(문자)를 이용해 수를 표현
    - 10배마다 자릿수 올라감
    - ex. $f(\text{“123”}) = 1\times10^2 + 2\times10^1 + 3\times10^0$
  - 알파벳 소문자
    - 26개의 문자를 이용해 표현
    - $f(\text{“abz”}) = 0\times26^2 + 1\times26^1 + 25\times26^0$
  - 문자열
    - C개의 문자를 이용해 표현
    - $f(\text{“}s_0s_1\ldots s_{L-1}\text{”}) = s_0C^{L-1} + s_1C^{L-2} + \ldots + S_{L-2}C^1 + S_{L-1}C^0$
    - 함수 f는 모든 문자열을 임의의 정수에 1:1 대응시킴
    - “고정된” 길이의 데이터로 만들기 위한 작업을 추가로 수행해야 함

Hashing

- 해싱
  - 라빈 카프 알고리즘
    - 적당한 정수 C, M을 선택
    - $f(\text{“}s_0s_1\ldots s_{L-1}\text{”}) = (s_0C^{L-1} + s_1C^{L-2} + \ldots + S_{L-2}C^1 + S_{L-1}C^0) \bmod M$
    - mod M 을 제외하면 1:1 대응이므로 충돌이 발생하지 않음
      - 단, C ≥ 사용하는 문자의 종류
    - 해시 충돌은 M으로 나눈 나머지를 계산하는 것에 의해 발생
    - C, M을 잘 선택했다면 충돌이 발생할 확률은 1/M
    - M을 크게 잡아야 함
      - 보통 10억 정도의 소수로 잡음
      - C와 M은 서로소, M은 소수일 때 해시 충돌이 적게 발생함

질문?

Hashing

- 해싱
  - 부분 문자열의 해시값
    - H[i] = S[0..i]의 해시값
    - H[i] = (H[i-1] * C + S[i]) % M
      - $H[i] = s_0C^i + s_1C^{i-1} + \dots + s_{i-1}C^1 + s_iC^0$
      - $H[i] = (s_0C^{i-1} + s_1C^{i-2} + \dots + s_{i-1}C^0) * C + s_i$
      - H[i] = H[i-1] * C + S[i]

    - S[l..r]의 해시값 = $(H[r] - H[l-1] * C^{r-l+1})$ % M
    - $H[r] = s_0C^r + s_1C^{r-1} + \dots + s_{l-1}C^{r-l+1} + s_lC^{r-l} + \dots + s_rC^0$
    - $H[l-1] = s_0C^{l-1} + s_1C^{l-2} + \dots + s_{l-1}C^0$
    - H 배열과 C의 거듭제곱은 모두 O(|S|)에 전처리 가능
    - 부분 문자열의 해시값을 O(1)에 구할 수 있음

Hashing

```cpp
template<ll P, ll M> struct Hashing {
    vector<ll> H, B;
    void build(const string &S){
        H.resize(S.size()+1);
        B.resize(S.size()+1);
        B[0] = 1;
        for(int i=1; i<=S.size(); i++) H[i] = (H[i-1] * P + S[i-1]) % M;
        for(int i=1; i<=S.size(); i++) B[i] = B[i-1] * P % M;
    }
    ll get(int s, int e){
        ll res = (H[e] - H[s-1] * B[e-s+1]) % M;
        return res >= 0 ? res : res + M;
    }
};
```

질문?

Hashing

- 해싱
  - 문자열 A, B의 사전순 비교
    - 문자열 A가 B보다 사전순으로 앞선다.
      - A[i] ≠ B[i]인 가장 작은 i에서 A[i] < B[i]이다.
    - A와 B의 접두어가 처음으로 다른 지점을 이분 탐색으로 찾을 수 있음
    - O(log min(|A|, |B|))

Hashing

- BOJ 21162 뒤집기 K
  - 문제 요약
    - 길이가 N인 수열과 정수 K가 주어짐
    - 수열을 길이가 0이 아닌 두 부분으로 나눠서 각각을 뒤집은 다음 다시 붙임
      - A[1..i]와 A[i+1..N]으로 나눴다면 A[i]A[i-1]…A[1]A[N]A[N-1]…A[i+1]이 됨
    - 이렇게 해서 생성되는 모든 N-1개의 수열을 사전순으로 나열했을 때 K번째로 오는 수열을 찾는 문제
  - 풀이
    - 위에서 언급한 방법으로 생성되는 수열은
    - 주어진 수열을 뒤집은 수열의 Cyclic Shift 형태
    - 주어진 수열을 뒤집은 다음 2번 이어 붙인 수열을 A라고 하자.
    - A[2..N+1], A[3..N+2], … , A[N..2N-1]을 정렬하면 됨
    - 사전순 비교를 O(log N)에 할 수 있으니 정렬은 $O(N \log^2 N)$에 할 수 있음

Hashing

- BOJ 21162 뒤집기

```cpp
int N, K;
vector<int> V;
Hashing<524287, 998244353> H1;

bool cmp(int a, int b){
    int l = 0, r = N-1;
    while(l < r){
        int m = (l + r + 1) / 2;
        auto t1 = H1.get(a, a+m-1), t2 = H1.get(b, b+m-1);
        if(t1 == t2) l = m;
        else r = m - 1;
    }
    if(l == N-1) return false;
    return V[a+l] < V[b+l];
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K; V.resize(N);
    for(auto &i : V) cin >> i;
    reverse(V.begin(), V.end());
    for(int i=0; i+1<N; i++) V.push_back(V[i]);
    H1.build(V);

    vector<int> O(N-1);
    iota(O.begin(), O.end(), 1);
    stable_sort(O.begin(), O.end(), cmp);
    for(int i=0; i<N; i++) cout << V[O[K-1]+i] << " ";
}
```

질문?

Hashing

- BOJ 17228 아름다운 만영로
  - 문제 요약
    - rooted tree가 주어짐
    - 부모 정점에서 자식 정점으로만 이동할 수 있음
    - 트리의 간선에 각각 한 개의 알파벳 소문자가 적혀 있음
    - 경로에 포함된 간선의 알파벳을 차례대로 연결한 문자열이 주어진 문자열 P가 되는 경우의 수
  - 풀이
    - DFS를 하면서
    - 루트에서 현재 정점으로 가는 경로의 해시를 관리하면 됨
    - 스택으로 쉽게 할 수 있음

Hashing

- BOJ 17228 아름다운 만영로

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr ll P = 917, M = 998244353;

int N, R;
string S; ll H, B = 1;
vector<pair<int,char>> G[505050];
vector<ll> Stack;

void DFS(int v){
    if(Stack.size() > S.size()){
        ll now = Stack.back() - Stack[Stack.size()-S.size()-1] * B;
        now = (now % M + M) % M;
        R += now == H;
    }
    for(auto [i,c] : G[v]){
        Stack.push_back((Stack.back() * P + c) % M);
        DFS(i);
        Stack.pop_back();
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<N; i++){
        int s, e; char c; cin >> s >> e >> c;
        G[s].emplace_back(e, c);
    }
    cin >> S;
    for(auto i : S) H = (H * P + i) % M, B = B * P % M;
    Stack.push_back(0);
    DFS(1);
    cout << R;
}
```

질문?

KMP

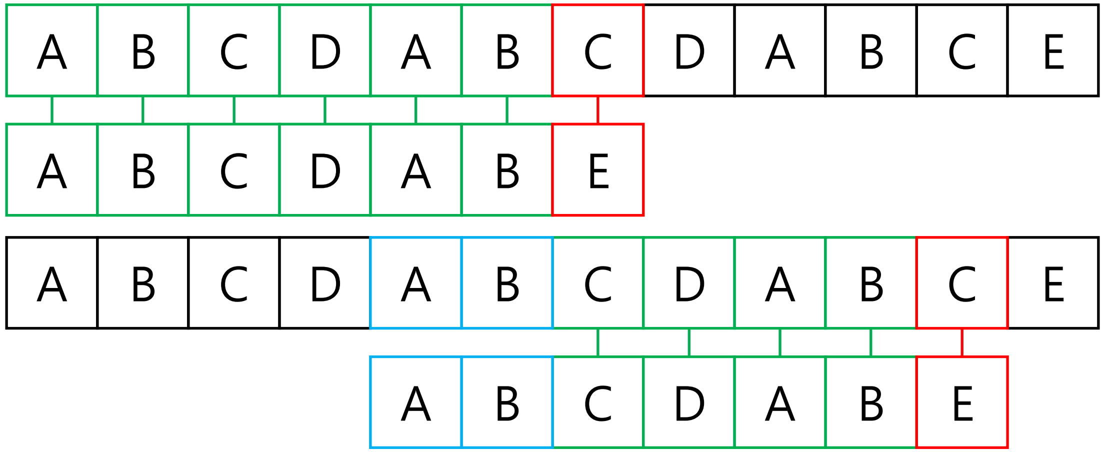
KMP

- KMP
  - 해결해야 하는 문제
    - 문자열 S, P가 주어지면
    - S에서 P가 부분 문자열로 등장하는 위치를 모두 찾는 문제

    - 브루트 포스하면 $O(|S||P|)$
    - KMP를 이용하면 $O(|S| + |P|)$에 할 수 있음

KMP

- KMP
  - 브루트 포스 풀이의 비효율적인 부분
    - S[x+i]와 P[i]가 일치하지 않는다는 것을 통해 알 수 있는 정보
      - S[x..x+|P|-1]은 P와 일치하지 않음
      - S[x..x+i-1]과 P[0..i-1]은 일치함
        - 브루트 포스 풀이는 두 번째 정보를 사용하지 않음
    - 두 번째 정보를 통해 알 수 있는 것
      - $1 \le p < i$, P[0] = P[p]인 p에 대응되는 S[x+p]에서 시작하는 부분 문자열만 봐도 됨
      - P[0..t] = P[p..p+t]라면 첫 t+1개의 문자는 비교하지 않아도 됨



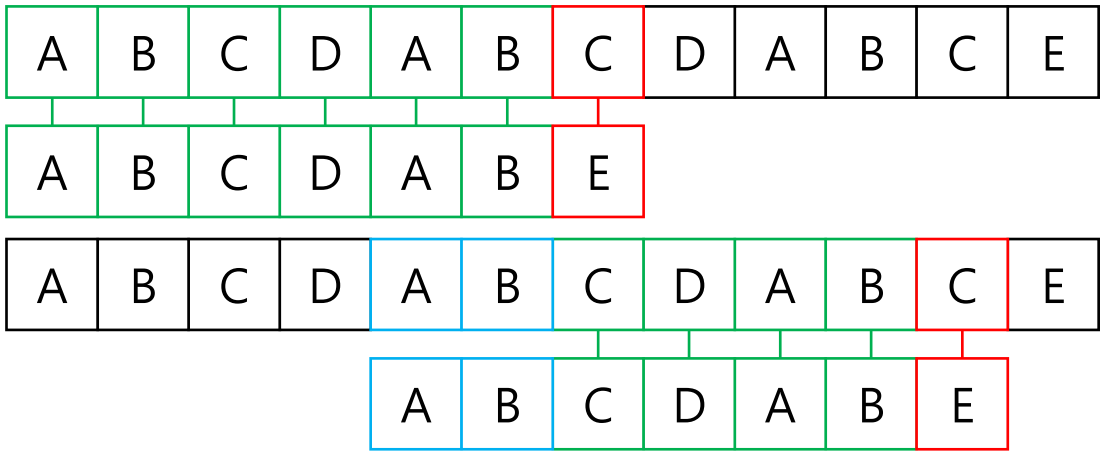
KMP

- KMP
  - 실패 함수 F[i] := P[0..k-1] = P[i-k+1..i]를 만족하는 가장 큰 k
    - P의 Prefix와 Suffix가 일치하는 가장 긴 길이

  - S[x+i]와 P[i]가 일치하지 않음
    - S[x+i-F[i]]부터 다시 시작
    - P[F[i]]부터 비교하면 됨

A	B	C	D	A	B	E
0	0	0	0	1	2	0

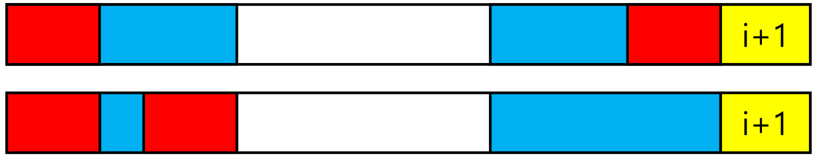
A	B	C	A	B	C	A	C	A	B
0	0	0	1	2	3	1	0	1	2



KMP

- KMP
  - 실패 함수를 구하는 방법 (알고리즘)
    - F[i]를 구했고, F[i+1]을 구해야 하는 상황
    - 만약 P[F[i]] = P[i+1]이라면 F[i+1] = F[i] + 1

    - 그렇지 않다면
      - P[0..F[i-1]]의 Suffix와 일치하는 P의 Prefix를 찾아야 함
      - P[0..F[i]-1]과 P[i-F[i]+1..i]는 일치하므로
        P[0..F[i]-1]의 Suffix를 찾아도 됨
      - P[F[i]] = P[i+1] 를 만족할 때까지 F[i], F[F[i]], F[F[F[i]]], ... 를 따라가면 됨



KMP

- KMP
  - 실패 함수를 구하는 방법 (코드, 시간 복잡도)
    - i, j의 증감 횟수를 구해야 함

    - i-j는 항상 증가하고, j는 항상 감소함
    - 반복문의 각 단계마다 아래 두 가지 중 하나 시행
      - i, j 1씩 증가
      - j 감소
    - $0 \le i, j < |P|$이므로 $O(|P|)$

```cpp
vector<int> GetFail(const string &p){
    int m = p.size();
    vector<int> fail(m);
    for(int i=1, j=0; i<m; i++){
        while(j > 0 && p[i] != p[j]) j = fail[j-1];
        if(p[i] == p[j]) fail[i] = ++j;
    }
    return fail;
}
```

질문?

KMP

- KMP
  - 문자열을 매칭하는 방법 (알고리즘)
    - S[i-1] = P[j-1]이 대응됐고, S[i]와 대응시킬 P[k]를 찾아야 하는 상황
    - 만약 S[i] = P[j]이라면 k = j
      - 문자열의 끝까지 대응시켰다면(k+1=|P|) 정답 배열에 i-|P|+1을 추가하고
      - j = F[|P|-1]로 바꾸고 탐색 계속 진행
    - 그렇지 않다면
      - S[i-j..i-1]의 Suffix와 일치하는 P의 Prefix를 찾아야 함
      - P[0..j-1]의 Suffix와 일치하는 P의 Prefix를 찾아야 함
      - S[i] = P[j]를 만족할 때까지 F[j-1], F[F[j-1]], … 를 따라가면 됨

KMP

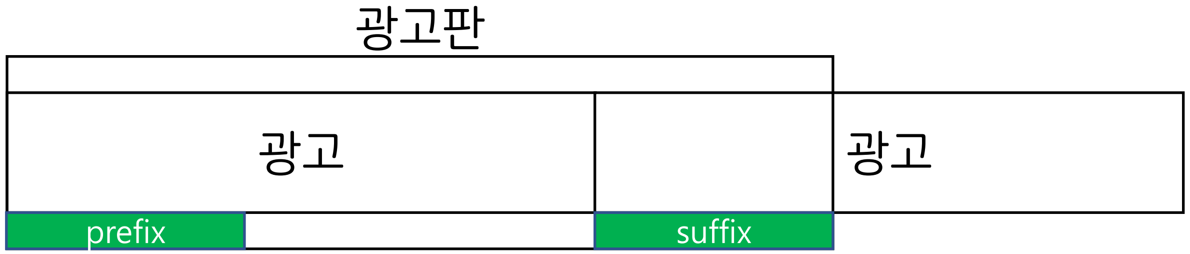
- KMP
  - 문자열을 매칭하는 방법 (코드)
    - GetFail 함수와 똑같음

```
vector<int> KMP(const string &s, const string &p){
    int n = s.size(), m = p.size();
    vector<int> fail = GetFail(p), res;
    for(int i=0, j=0; i<n; i++){
        while(j > 0 && s[i] != p[j]) j = fail[j-1];
        if(s[i] == p[j]){
            if(j + 1 == m) res.push_back(i-m+1), j = fail[j];
            else j++;
        }
    }
    return res;
}
```

KMP

- BOJ 1305 광고
  - 문제 요약
    - 광고 문구의 길이 N, 광고판의 길이 L
    - L = 6, 광고 문구 = “aaba”면 광고판에는 “aabaaa” → “abaaab” → “baaaba” → …
    - 광고판에 보이는 문자열이 주어지면 가능한 N의 최솟값을 구하는 문제

  - 풀이
    - 광고판에 보이는 문자열 = 광고 문구의 cyclic shift를 2번 붙인 것의 prefix
    - 정답 = L – Fail[L-1]



KMP

- BOJ 4354 문자열 제곱
  - 문제 요약
    - 문자열 S가 주어짐
    - S = A^k를 만족하는 가장 큰 k를 구하는 문제
      - A^k = A를 k번 붙인 문자열
  - 풀이
    - A는 S의 prefix
    - 0 ≤ i < N-C 인 모든 i에 대해 S[i] = S[i+C]이면 C는 문자열 S의 주기
    - N/k는 S의 주기, k를 최대화해야 하므로 C의 최솟값을 구해야 함
    - 문자열의 가장 작은 주기 = N – F[N-1]
    - N – F[N-1]이 N의 약수면 N/(N-F[N-1]), 아니면 1

질문?