

#06-2. DFA

나정휘

<https://justicehui.github.io/>

목차

오토마타

결정적 유한 오토마타 (DFA)

동적 계획법 (DP)

오토마타

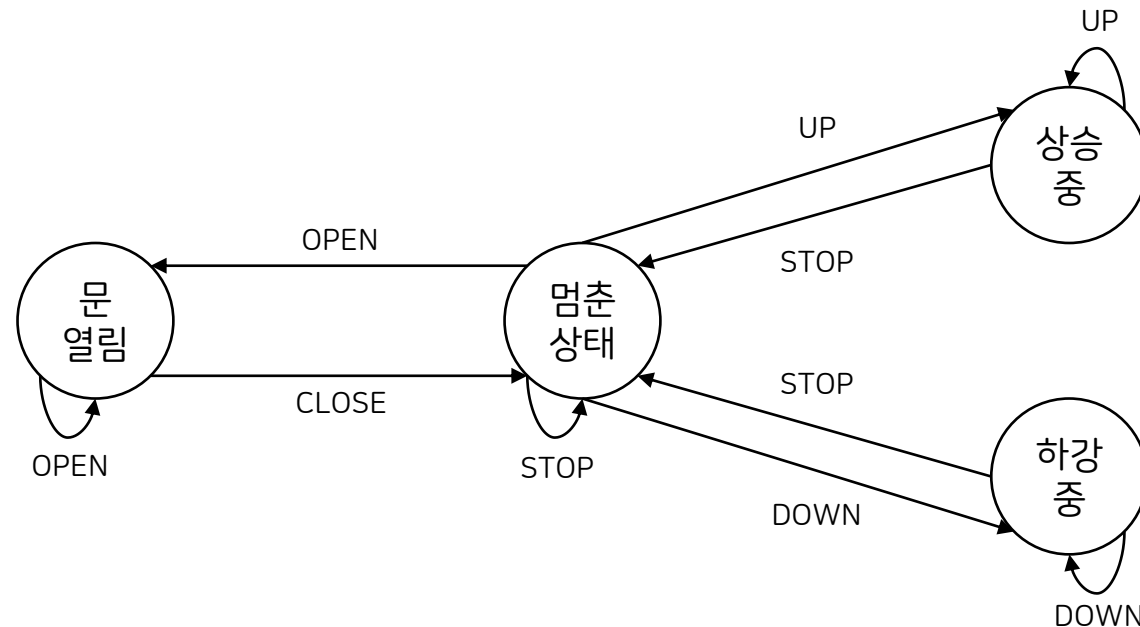
상태 (State)

- 동적 계획법은 문제의 상황을 "상태"로 표현한 다음
- 상태들 간의 전이를 효율적으로 계산하는 방법
- 어떤 작업의 중간 과정을 잘 나타내는 상태들과
- 어떤 이벤트가 발생했을 때 상태의 변화를 관찰하는 것
- ex. 0/1 Knapsack Problem
 - $(n, k) := 1..n$ 번째 물건을 적당히 선택했을 때 무게의 합이 k 인 상태
 - n 번째 원소를 선택하면 $(n-1, k-W_n) \rightarrow (n, k)$
 - n 번째 원소를 선택하지 않으면 $(n-1, k) \rightarrow (n, k)$

오토마타

오토마타

- 유한한 개수의 내부 상태를 갖고 있음
- 유한 또는 무한한 크기의 저장 공간을 갖고 있을 수도 있음
- 입력이 주어지면 미리 정해져 있는 방식으로 상태를 전이해서 출력을 내놓는 장치
- ex. 엘리베이터 (상태 4개, 저장 공간 없음)
 - 가능한 입력: OPEN, CLOSE, UP, DOWN, STOP



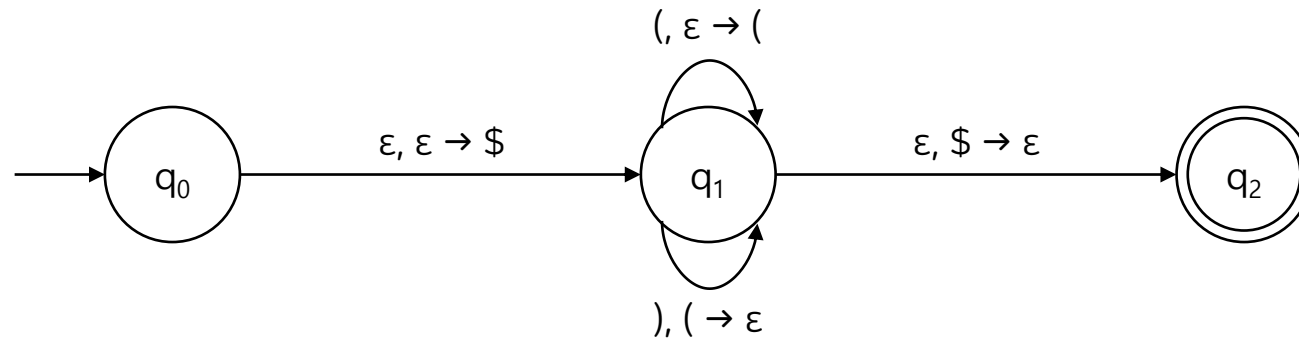
오토마타

오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : $()(())$

- stack :



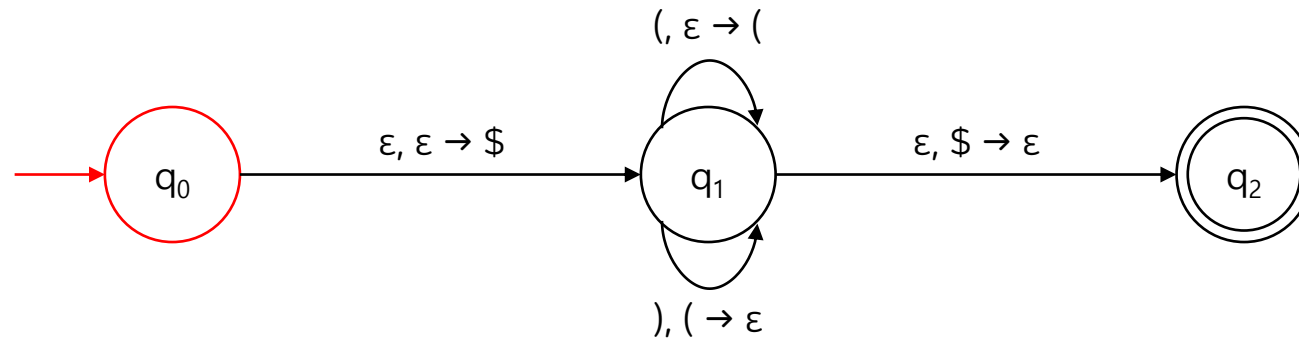
오토마타

오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : `()(())`

- stack :



오토마타

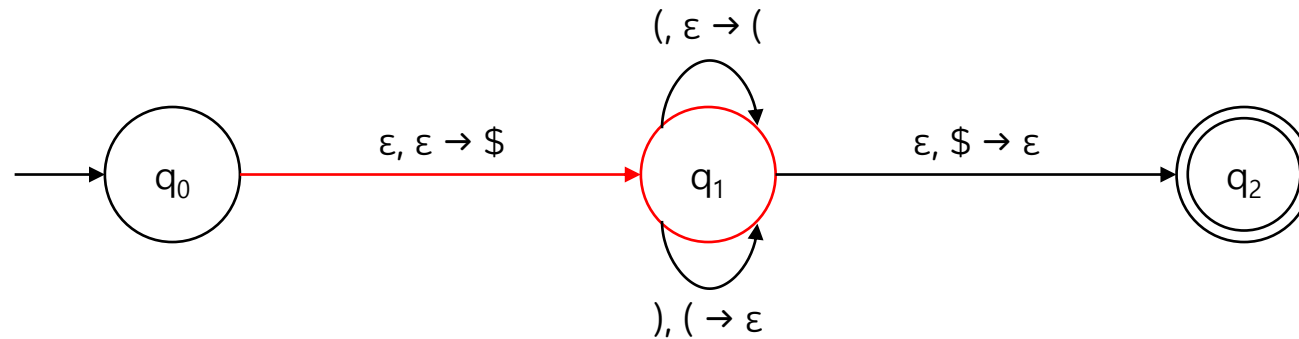
오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : $()(())$

- stack :

\$



오토마타

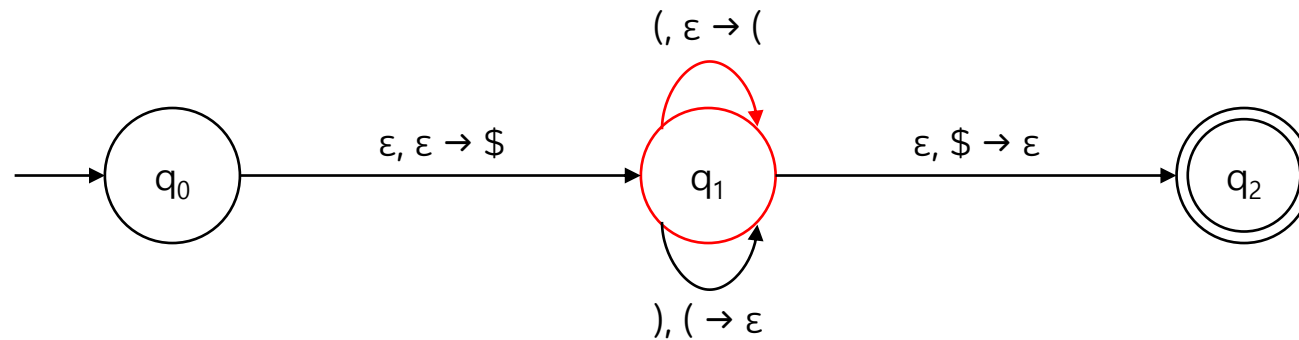
오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : **(**) (())

- stack :

(
\$



오토마타

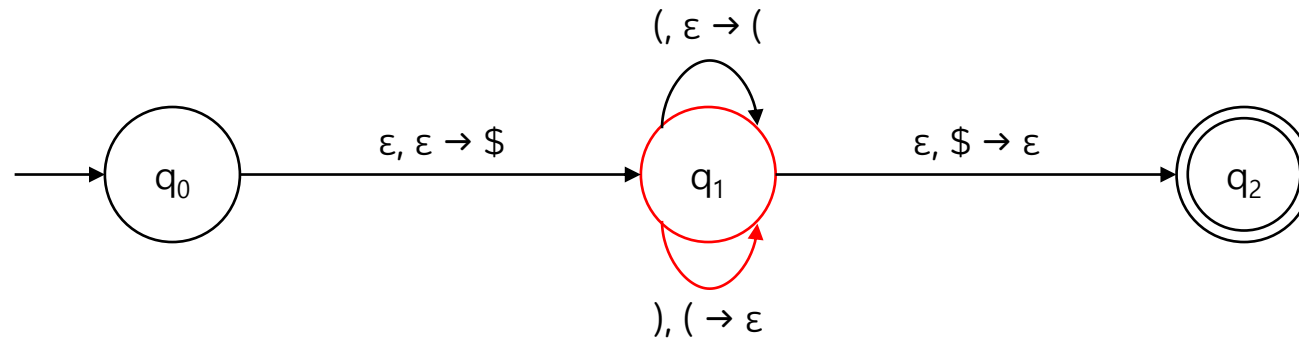
오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : (**)** (())

- stack :

(
\$



오토마타

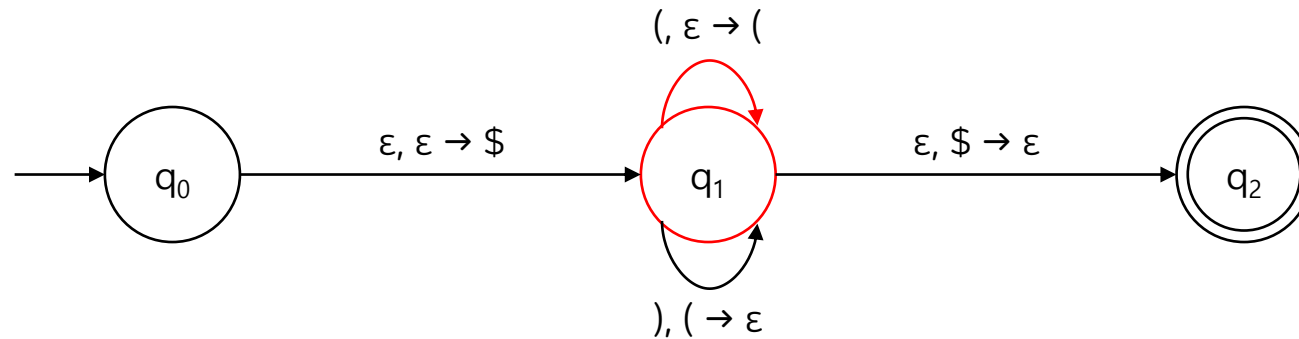
오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : () (())

- stack :

(
\$



오토마타

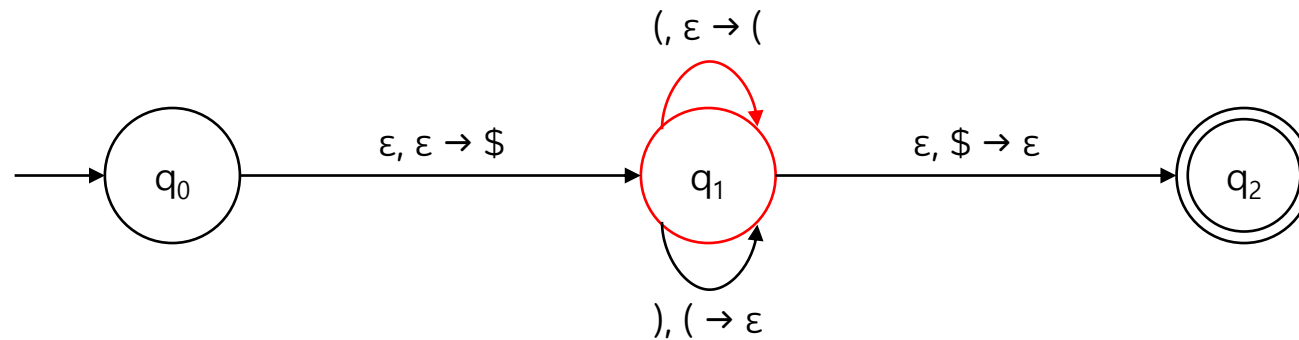
오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : () (())

- stack :

(
(
\$



오토마타

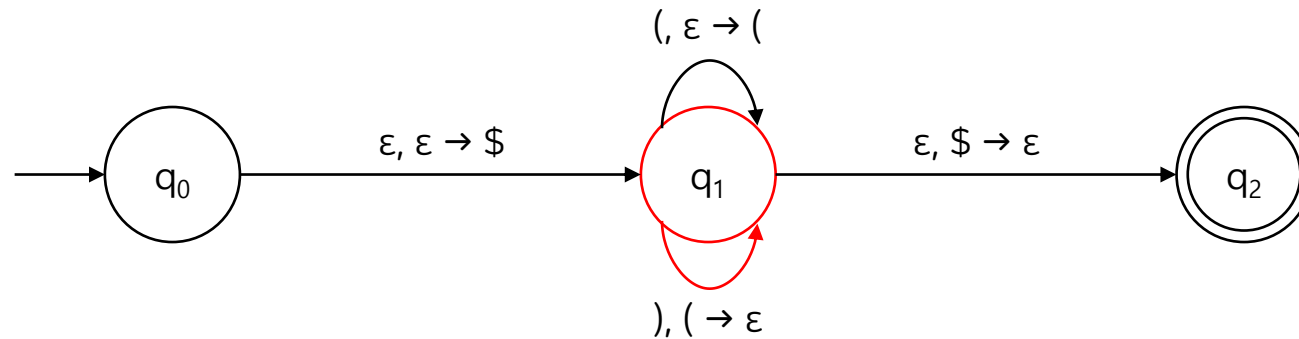
오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : () (())

- stack :

(
(
\$



오토마타

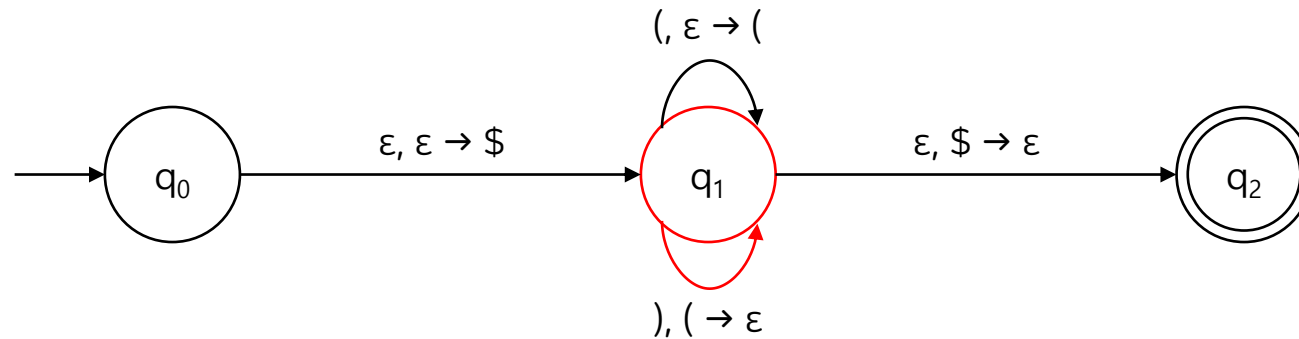
오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : () (())

- stack :

(
\$



오토마타

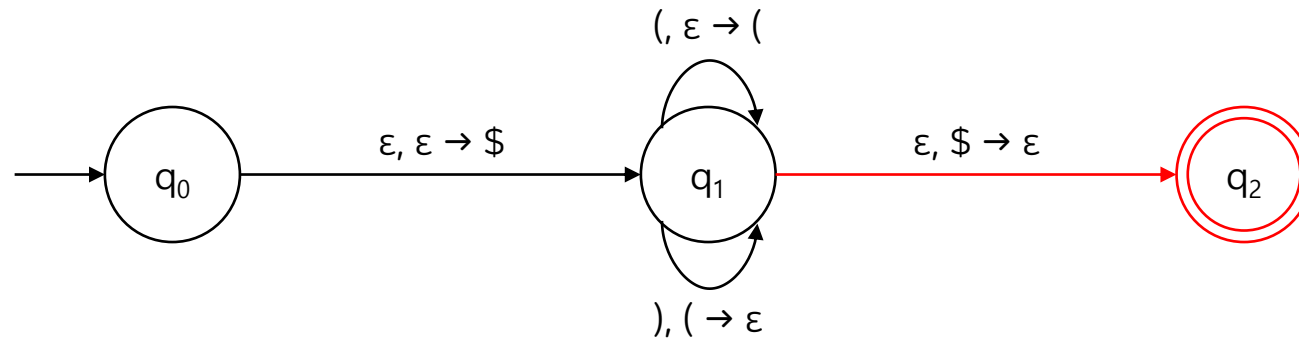
오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : $()(())$

- stack :

\$



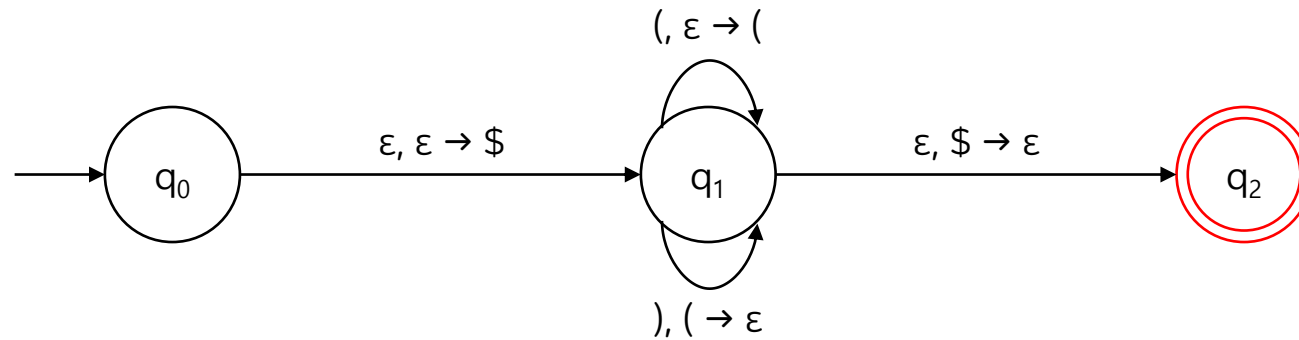
오토마타

오토마타

- ex. 올바른 괄호 문자열 판별 (상태 3개, 스택 형태의 메모리)
 - 가능한 입력: '(', ')'

- input : `()(())`

- stack :



질문?

용어 정의

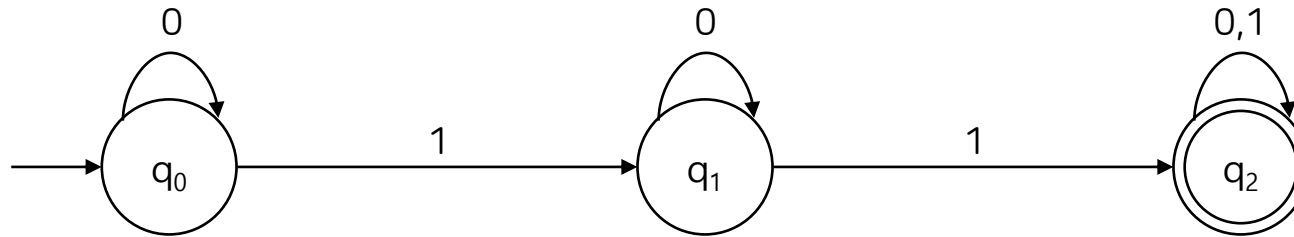
용어 정의

- 알파벳(alphabet): symbol들로 구성된 유한 집합
- 문자열(string): 길이가 유한한 symbol들의 나열
 - 알파벳 Σ 에서 정의된 모든 문자열을 포함하는 집합을 Σ^* 라고 부름
- 언어(language): Σ^* 의 부분 집합, 문자열들의 집합
- 예시
 - 알파벳이 $\Sigma_1 = \{a, b, c, \dots, y, z\}$ 이면 abc, sccc, jhnah와 같은 문자열을 만들 수 있음
 - 알파벳이 $\Sigma_2 = \{0, 1\}$ 이면 $L_2 = \{w \mid w \text{ starts with } 010\}$ 과 같은 언어를 정의할 수 있음
 - L_2 에는 010, 0101, 010010과 같은 문자열을 포함함

DFA

결정적 유한 오토마타 (Deterministic Finite Automata)

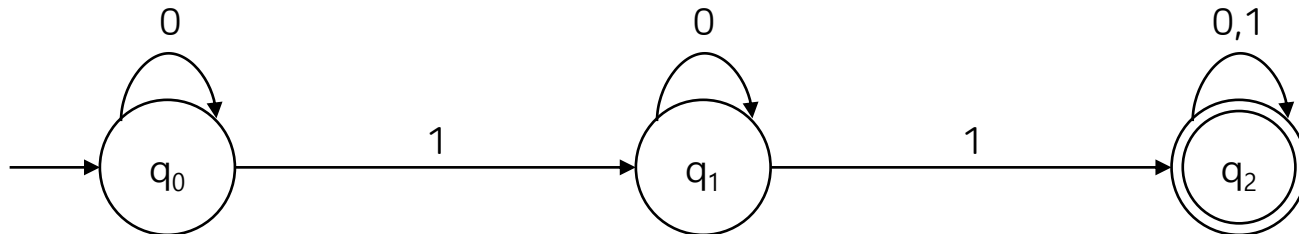
- 문자열을 읽어서 accept 또는 reject 시키는 장치
- 유한한 개수의 내부 상태가 있음
- 문자를 앞에서부터 한 글자씩 읽으면서, 읽은 문자에 따라 정해진 대로 상태를 전이함
- ex. 1이 2개 이상 있는 문자열만 accept하는 DFA
 - 시작점이 없는 화살표는 정확히 1개 존재 → 시작 상태
 - 두 겹으로 그려진 원은 종료 상태, 여러 개 존재할 수 있음
 - 문자열을 모두 읽었을 때 종료 상태 중 한 곳에 도달하면 accept, 그렇지 않으면 reject



DFA

결정적 유한 오토마타

- DFA는 $M = (Q, \Sigma, \delta, q_0, F)$ 와 같이 5개의 원소로 이루어진 튜플로 표현함
 - Q : 상태들의 집합
 - Σ : 알파벳
 - $\delta: Q \times \Sigma \rightarrow Q$: 전이 함수, $\delta(p, a) \rightarrow q$ 는 현재 상태가 p 인 상황에서 a 를 읽으면 q 로 간다는 뜻
 - $q_0 \in Q$: 시작 상태
 - $F \subseteq Q$: 종료 상태
- M 이 accept 시키는 모든 문자열의 집합을 $L(M)$ 이라고 부름
- 아래의 DFA는 $M_1 = (\{q_0, q_1, q_2\}, \{0,1\}, \delta_1, q_0, \{q_2\})$ 로 나타낼 수 있음
 - 1이 2개 이상인 문자열만 accept 시키는 DFA
 - $L(M_1) = \{s \mid n_1(s) \geq 2\}$



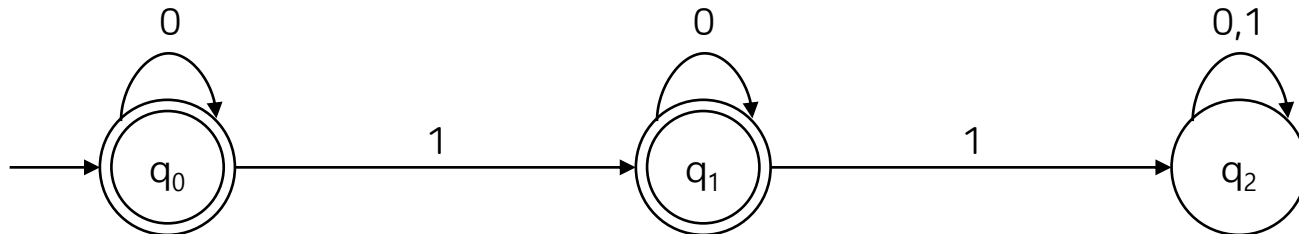
δ_1	0	1
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_2	q_2

DFA

결정적 유한 오토마타

- DFA는 $M = (Q, \Sigma, \delta, q_0, F)$ 와 같이 5개의 원소로 이루어진 튜플로 표현함
 - Q : 상태들의 집합
 - Σ : 알파벳
 - $\delta: Q \times \Sigma \rightarrow Q$: 전이 함수, $\delta(p, a) \rightarrow q$ 는 현재 상태가 p 인 상황에서 a 를 읽으면 q 로 간다는 뜻
 - $q_0 \in Q$: 시작 상태
 - $F \subseteq Q$: 종료 상태
- M 이 accept 시키는 모든 문자열의 집합을 $L(M)$ 이라고 부름

- 아래의 DFA는 $M_2 = (\{q_0, q_1, q_2\}, \{0,1\}, \delta_2, q_0, \{q_0, q_1\})$ 로 나타낼 수 있음
 - 1이 2개 미만인 문자열만 accept 시키는 DFA
 - $L(M_2) = \{s \mid n_1(s) < 2\}$



δ_2	0	1
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_2	q_2

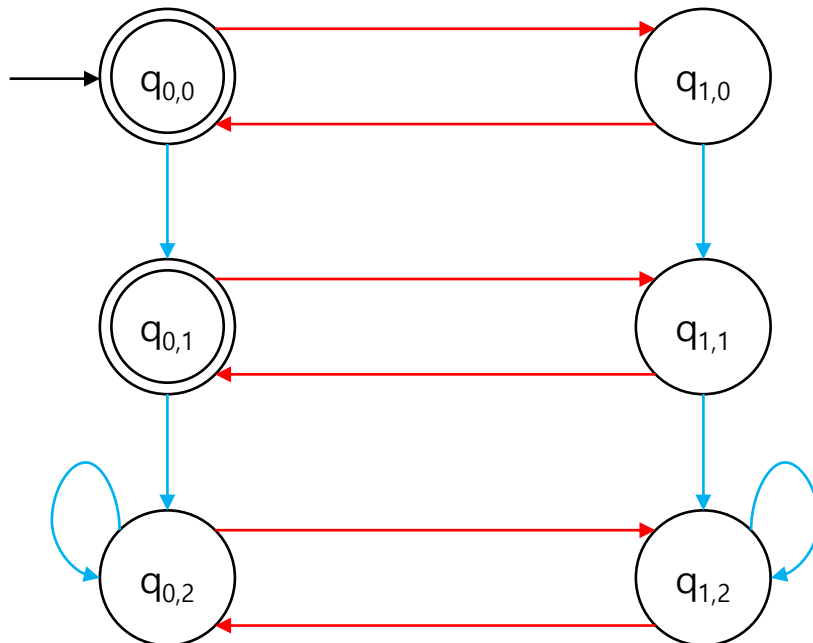
DFA

결정적 유한 오토마타

- 0이 짝수 개 있는 문자열만 accept하는 DFA



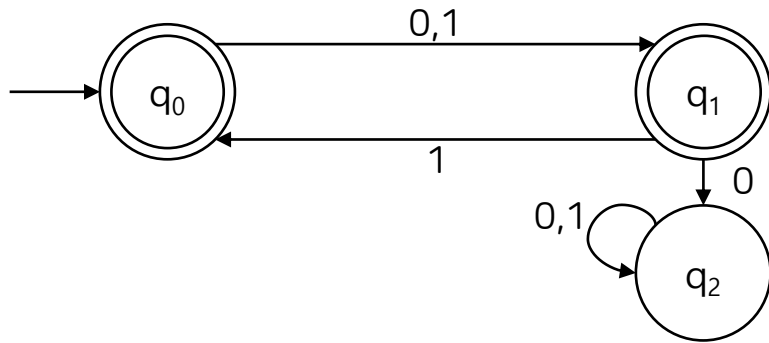
- 0이 짝수 개, 1이 2개 미만 있는 문자열만 accept하는 DFA



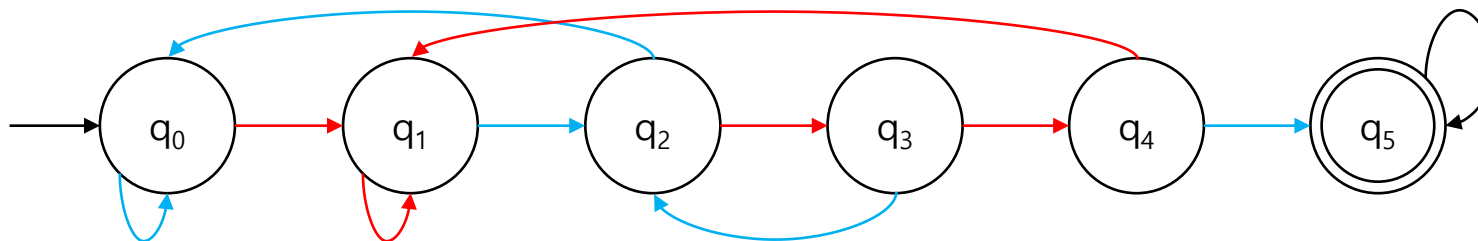
DFA

결정적 유한 오토마타

- 모든 짝수 번째 자리가 1인 문자열만 accept하는 DFA




- 01001을 포함하는 문자열만 accept하는 DFA
 - q_i = 01001과 앞에서부터 i 글자 매칭된 상태
 - 이 방법을 잘 활용한 알고리즘이 KMP



DFA

결정적 유한 오토마타

- 십진법으로 해석했을 때 K 의 배수인 문자열만 accept하는 DFA
 - q_x = 현재까지 읽은 수를 K 로 나눈 나머지가 x 인 상태
 - $M = (\{q_0, q_1, \dots, q_{K-1}\}, \{0, 1, 2, \dots, 9\}, \delta, q_0, \{q_0\})$
 - $\delta(q_x, y) = q_{(10x+y) \% K}$



```
int string_modulo_k(const string &s, int k){
    int res = 0;
    for(auto i : s){
        res = (res * 10 + i - '0') % k;
    }
    return res;
}
```

질문?

DP

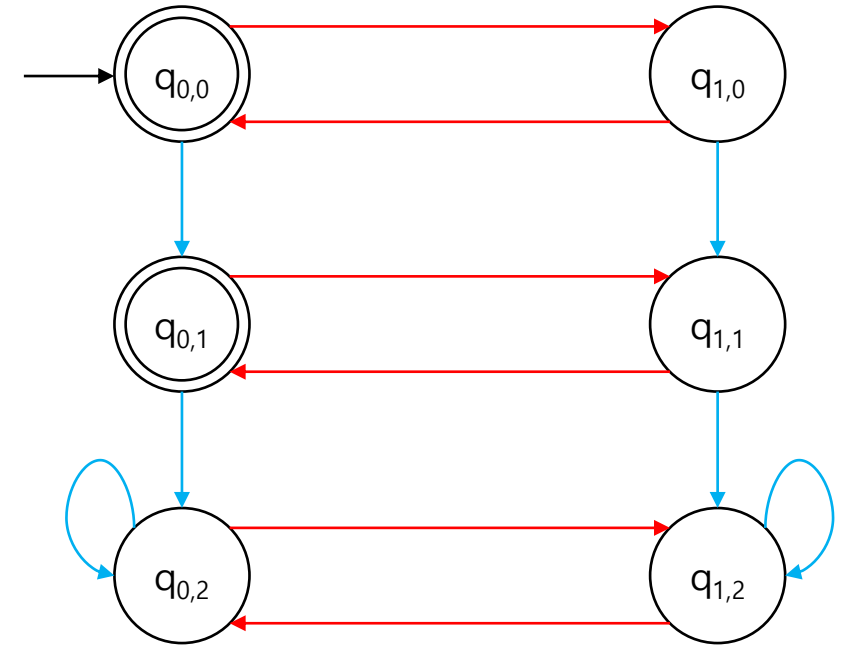
동적 계획법에서의 활용

- 길이가 n 이면서 0이 짝수 번, 1이 최대 1번 등장하는 binary string의 개수를 세는 문제
 - $D(n, x, y) =$ 길이가 n 이면서 $q_{x,y}$ 에서 종료하는 binary string의 개수



```
int N, D[SZ][2][3];

int main(){
    cin >> N;
    D[0][0][0] = 1;
    for(int i=0; i<N; i++){
        for(int j=0; j<2; j++){
            for(int k=0; k<3; k++){
                D[i+1][(j+1)%2][k] += D[i][j][k];
                D[i+1][j][min(k+1,2)] += D[i][j][k];
            }
        }
    }
    return D[N][0][0] + D[N][0][1];
}
```



DP

동적 계획법에서의 활용

- 길이가 n 이고 이진법으로 해석했을 때 9의 배수인 binary string의 개수
 - $D(n, m)$ = 길이가 n 이고 9로 나눈 나머지가 m 인 binary string의 개수
 - 사실 $((1 \ll n) - 1) / 9 + 1$ 출력해도 됨



```
int N, D[101010][9];

int main(){
    cin >> N;
    D[0][0] = 1;
    for(int i=0; i<N; i++){
        for(int j=0; j<9; j++){
            for(int k=0; k<2; k++) D[i+1][(j*2+k)%9] += D[i][j];
        }
    }
    return D[N][0];
}
```

DP

동적 계획법에서의 활용

- 길이가 n 인 binary string이 주어졌을 때, 최소 개수로 수정해서 9의 배수가 되도록 수정
 - $D(n, m) = 1..n$ 번째 글자를 적당히 수정해서 9로 나눈 나머지가 m 이 되도록 만드는 최소 수정 횟수



```
int N, D[101010][9];
string S;

int main(){
    cin >> N >> S;
    for(int i=0; i<=N; i++) for(int j=0; j<9; j++) D[i][j] = 1e9;
    D[0][0] = 0;
    for(int i=0; i<N; i++){
        for(int j=0; j<9; j++){
            int c = S[i] - '0';
            D[i+1][(j*2+c)%9] = min(D[i+1][(j*2+c)%9], D[i][j]);
            D[i+1][(j*2+1-c)%9] = min(D[i+1][(j*2+1-c)%9], D[i][j] + 1);
        }
    }
    return D[N][0];
}
```

질문?