9.1 소개

그래프의 모든 쌍에 대해

1. u로 부터 v까지 경로가 있는가?
2. u로부터 v까지 최단 경로는 무엇인가?

) 를 구하는 방법

9.2 이진 관계의 이행 폐쇄

이행 폐쇄(transitive closure)??

- 정의와 예비 지식

$S = \{S_1, S_2, \cdots, S_n\}$ 에서 S에 대한 binary relation $A \subseteq S \times S$

↳ $(S_i, S_j) \in A$ 이면 S_i는 S_j에 A-관계되어 있다 ⇒ $S_i A s_j$

S가 n개의 원소를 가질 때, 관계 A의 원소 → $n \times n$ 의 boolean 행렬로 표현 가능

$$a_{ij} = \begin{cases} \text{True} & (s_i A s_j) \\ \text{False} & (X) \end{cases}$$

→ O (영행렬) T (항등행렬) 도 정의됨

↳ 그래프의 정점 집합에 대한 인접관계는 이진 관계의 주요 예 (동치관계, 부분 순서 포함)

⇒ 집합 S에 대한 임의의 이진 관계 A를 유향 그래프로 해석 가능 (역) ∴ $G = (S, A)$

관계 A가 S의 모든 원소에 대해 $S_i A s_j$ 이고 $s_j A s_k$ 일 때 $S_i A s_k$ 이면 이행적(transitive)

↳ 동치 관계와 부분 순서는 이행적. 그래프의 인접관계는 보통 이행적이지 않음

〈정의 9.1〉 이행 폐쇄

A가 S에 대한 이진 관계. $G = (S, A)$ A의 반사적 이행 폐쇄 (reflexible transitive closure)는

⇒ G에 S_i로부터 S_j까지 경로가 존재할 때만 $S_i R s_j$ → 도달 가능 관계 (reachability relation)

↳ 반사적 (reflexible) ⇒ 자기 자신으로 가는 길이 0인 경로도 존재

↳ 비반사적 경로는 길이 0을 허용하지 않는 것.

A의 이행 폐쇄는 A 자신 ⇒ 임의의 관계 A의 이행 폐쇄: $A \subseteq R$, R은 이행적이고 반사적

· 깊이 우선 탐색으로 도달 가능 행렬 찾기

$G=(S,A)$ 에서 도달 가능 행렬 R 찾기 ⇒ 깊이 우선 탐색으로 가능

↳ s_i로 부터 깊이 우선 탐색 실시해서 s_j를 만나면 r_{ij} true.

↳ s_i로부터 시작했는데 i행이 아닌 원소도 채울 수 있음 ⇒ 비효율적

$s_i \rightarrow s_k \rightarrow s_j$ 일 때 스택을 이용해서 경로 상의 모든 s_k에 true 가능

↳ 이렇게 변경해도 최악의 경우 작업량은 $\Theta(nm)$ ← 깊이 우선 탐색과 동일

유향 그래프 → 압축 그래프 로 변경 ⇒ 효율적으로 처리 가능

↳ 강연결 성분을 하나의 정점으로 한 그래프

↳ ① G의 강연결 성분을 $\Theta(m+n)$ 시간에 찾음 → G의 압축 그래프 : $G\downarrow$

② $G\downarrow$ 의 도달 가능 관계 탐색

③ $G\downarrow$ 의 각 정점에 대해 $O(n^2)$의 시간에 압축된 G의 모든 정점으로 대치 ⇒ $G\downarrow$의 도달 관계 확장

· 지름길을 이용한 이행 폐쇄

$G=(S,A)$에서 A의 이행 폐쇄 R을 찾는 것 ⇒ 유향 그래프에 에지를 추가하는 것과 대응

↳ s_is_k와 s_ks_j에 대해 s_is_j를 추가하는 것 ⇒ 어떤 k에 대해 s_iRs_k and s_kRs_j ⇒ s_iRs_j

↳ s_is_j는 한 번에 갈 수 있는 지름길 ⇒ 더 이상 추가할 지름길이 없는 R은 이행적

<알고리즘 9.1> 지름길을 이용한 이행 폐쇄

복잡도 : n^3에 for 루프가 반복되는 횟수에 비례

input : A, n (A : n×n boolean 행렬)

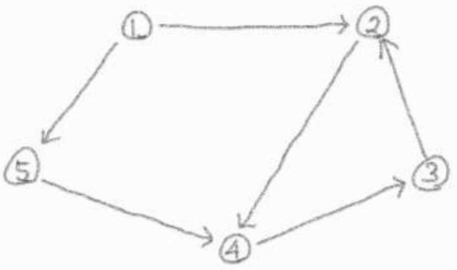
output : A의 이행 폐쇄를 나타내는 boolean 행렬 R

```
void simpleTransitiveClosure(boolean[][] A, int n, boolean[][] R)
    int i, j, k
    A를 R에 복사
    모든 주 대각 원소 rij를 true
    while ( loop를 다 돌았을 때 R의 한 원소라도 변경되면)
        for (i=1, i≤n, i++)
            for (j=1, j≤n, j++)
```

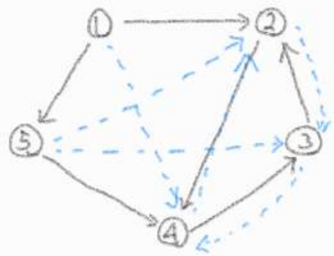
for($k=1$, $k \le n$, k++)

$$r_{ij} = r_{ij} \vee (r_{ik} \wedge r_{kj})$$

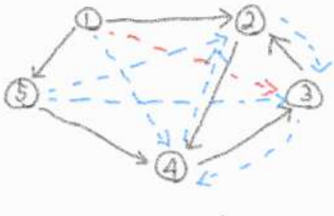
<예시>



관계 A (에지)



loop 1st.



loop 2nd

↳ while loop 시작 전에 자기 자신으로 가는 에지 추가

처음 loop 에서 (5, 2)는 추가 가능 ← (4, 2)가 먼저 추가되기 때문에

(1, 3)은 처음에 추가 불가능

while loop 생략 불가능 → 어떤 s_i, s_j가 고려될 때 연결된 s_k가 존재하지 않을수도 있음

↳ 나중에 에지가 추가됨에 따라 $s_i s_j$가 삽입될 수도 있음. (여러 번 고려 필요)

$r_{ij} = r_{ij} \vee (r_{ik} \wedge r_{kj})$: 삼중쌍 (triple pair) i, j, k의 처리

위의 예시에서 삼중쌍이 역순으로 처리되면 (5, 4, 3)이 먼저 처리 ⇒ 어떤 삼중쌍도 두 번 고려 X

↳ 어떤 삼중쌍도 두 번 이상 고려할 필요가 없는 순서가 있을까?

↳ 순서를 어떻게 두더라도 반복이 필요한 예를 찾을 수 있는가?

9.3 Warshall의 이행 폐쇄 알고리즘

삼중쌍을 적절한 순서로 (가장 바깥 루프에서 k를 변화시키면서) 처리하는 알고리즘.

<알고리즘 9.2> Warshall 이행 폐쇄

input, output은 9.1 과 동일

void transitiveClosure(boolean[][] A, int n, boolean[][] R)

int i, j, k — ↳ 복잡도: n^3

A를 R에 복사 — R 초기화: $\Theta(n^2)$

모든 주 대각 원소를 true — 변경 혹은 검사: $\Theta(n^3)$

```
for (k=1, k≤n, k++)
    for (i=1, i≤n, i++)
        for (j=1, j≤n, j++)
            r_ij = r_ij ∨ (r_ik ∧ r_kj)
```

〈정의 9.2〉 최상위 중간 정점

G의 정점을 $(s_1, s_2, \cdots, s_n)$인 순서열로 간주 (not set) G에서 길이가 1 이상인 경로에 대해

경로의 최상위 중간 정점(highest-numbered intermediate vertex)은 출발 정점이나 도착 정점을

제외하고 인덱스가 가장 큰 정점. 경로가 에지 1개면 0으로 간주

· 비트 행렬에 대한 Warshall의 알고리즘

행렬 A와 R의 한 원소가 한 비트에 저장된다면 비트 연산을 이용해 빠르게 구현 가능

〈정의 9.3〉 비트 스트링, 비트 행렬, bitwise OR

길이 n인 bit string : 컴퓨터 워드-경계(word-boundary)에서 시작하여 끝이 워드 경계까지 채워져 있는

연속적인 공간을 차지하고 있는 n 비트의 열.

↳ 컴퓨터 워드 길이가 C 비트라면, n bit의 bit string은 $\lceil n/c \rceil$개로 된 컴퓨터 워드의 배열에 저장

bit matrix : 각 비트 스트링이 한 행렬의 행이 되는 bit string의 배열

만약 A가 비트 행렬이면, A[i]는 A의 i번째 행을 나타내며 bit string. a_{ij}는 A[i]의 j번째 bit

a, b가 bit string, n이 정수이면 프로시저 bitwiseOR(a, b, n)은 n 비트에 대해 $a \vee b$를 비트별로 계산 후 a에 저장

〈알고리즘 9.3〉 비트 행렬에 대한 이행 폐쇄

Input, output은 알고리즘 9.1과 동일

```
void warshallBitMatrix(Bitmatrix A, int n, Bitmatrix R)
    int i, k
    A를 R에 복사
    모든 r_ii를 1로 초기화
    for (k=1, k≤n, k++)
        for (i=1, i≤n, i++)
            for (j=1, j≤n, j++)
```

```
if (r_ik = 1)
    bitwiseOR (R[i], R(k), n)
```

9.4 그래프의 모든 쌍 최단 경로

dijkstra 알고리즘 → 한 소스에서 다른 모든 정점까지의 최단 경로

ㄴ 인접 리스트 구조. 최대 $\Theta(n^2)$

$V = \{v_1, v_2, \cdots, v_n\}$ $G = (V, E, w)$ 에서 w가

$$w_{ij} = \begin{cases} w(v_i v_j) & v_{ij} \in E \\ \infty & v_{ij} \notin E, \quad i \neq j \\ \min(0, w(v_i v_j)) & v_{ii} \in E \\ 0 & v_{ii} \notin E \end{cases}$$

일 때, d_{ij}가 v_i로부터 v_j까지 최단 경로라고 정의되는 $n \times n$ 행렬 D 구하기

쉽게 생각해서 모든 정점에 대해 dijkstra 알고리즘 돌릴수도 있음

⇒ warshall의 알고리즘 이용하면 더 효과적으로 구현 가능

최단경로는 임의의 다른 정점을 임의순으로 방문 ⇒ 최상위 중간 정점에 따라 분류

ㄴ 최단 경로의 성질: v_i로부터 v_j까지 최단경로가 v_k를 지나면 v_i와 v_k, v_k와 v_j의 부분경로도 최단

ㄴ k가 v_i, v_j 사이에서 가장 큰 인덱스라면 두 부분경로의 최상위 정점은 k보다 작다.

$$D^0(i)(j) = w_{ij}$$

$$D^k(i)(j) = \min(D^{k-1}(i)(j),\ D^{k-1}(i)(k) + D^{k-1}(k)(j))$$

〈보조정리 9.3〉

$0, \cdots, n$에 속한 각 k에 대해 $d_{ij}^{(k)}$를 최상위 중간정점이 v_k인 $v_i \to v_j$의 최단 경로 가중치.

$D^{(k)}(i)(j)$가 위에서 정의된 식일 때, $D^{(k)}(i)(j) \leq d_{ij}^{(k)}$

위 식을 이용하여 $D^{(0)}, \cdots, D^{(n)}$ 까지의 행렬 계산. $D^{(k)}$ 계산에는 $D^{(k-1)}$만 필요하므로 이전 것 저장 X.

ㄴ 행렬 D 1개만 있어도 계산 가능

<알고리즘 9.4> (Floyd) 모든 쌍 최단 경로

input: 정점이 $v_1, v_2, \cdots, v_n$인 그래프의 가중치 행렬 W, n

output: 그래프 내에 가중치 음수인 사이클 X ⇒ $D[i][j]$는 $v_i \to v_j$가 최단 경로인 행렬

```
void allPairsShortestPaths(float[][] W, int n, float[][] D)
    int i,j,k
    W를 D에 복사
    for (k=1, k≤n, k++)
        for (i=1, i≤n, i++)
            for (j=1, j≤n, j++)
                D[i][j] = min( D[i][j] , D[i][k] + D[k][j])
```

$\theta(n^3)$ 연산 수행

모든 쌍 최단 경로 문제 → 도달 가능 행렬 R을 찾는 것보다 더 일반적인 문제

↳ Floyd 알고리즘은 warshall 알고리즘을 일반화한 것.

↳ D에서 삼중쌍 처리 ⇒ $\min(D[i][j],\ D[i][k] + D[k][j])$

→ 삼중쌍 처리 순서는 반복 처리를 하지 않으면서 옳은 결과를 얻는데 매우 중요한 역할.

9.5 행렬 연산으로 이행 폐쇄 계산하기

행렬 A : $S = \{s_1, s_2, \cdots, s_n\}$의 이진 관계 행렬, $G = (S, A)$에서 인접 관계

↳ 길이 1인 경로 ⇒ 에지

$$A^{(p)} : a_{ij}^{(p)} = \begin{cases} \text{true} & s_i \to s_j \text{ 길이가 } p\text{인 경로 존재} \\ \text{false} & \times \end{cases}$$

$A^{(0)} = I$ (항등행렬) $A^{(1)} = A$ 이때 $A^{(2)}$는?

$s_i \to s_j$인 길이 2인 경로가 $a_{ij}^{(2)}$ ⇒ $a_{ik} = \text{true}$, $a_{kj} = \text{true}$인 k가 존재해야함.

⇒ $a_{ij}^{(2)} = \bigvee_{k=1}^{n} (a_{ik} \wedge a_{kj})$ ← bool 행렬곱 $A^2(AA)$와 동일

<정의 9.4> boolean 행렬 연산

$n \times n$ 행렬 A, B의 bool 행렬 곱 $C = AB$: $1 \le i, j \le n$에 대해 $c_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj})$

〃 bool 행렬 합 $D = A + B$: $1 \le i, j \le n$에 대해 $d_{ij} = a_{ij} \vee b_{ij}$

↳ 결국 $A^{(P)}$의 특정 원소 a_{ij} = true 이면 $s_i \to s_j$인 길이 P의 경로가 존재

⇒ n개의 정점을 가진 유향 그래프 v로부터 w까지의 경로가 존재 ⇒ $v \to w$인 단순 경로 존재 (최대 길이 : $n-1$)

⇒ 이행 폐쇄를 구하기 위해 길이 $n-1$까지인 경로만 찾으면 됨.

임의의 P, q에 대해 $A^{(P)} + A^{(q)}$의 (i, j)번째 원소 : $s_i \to s_j$에서 길이가 P인 경로 혹은 q인 경로가 있을때만 true.

$R = \sum_{p=0}^{n-1} A^p$ 로 계산 가능 ($\theta(n^4)$ 소요)

↳ $s \geq n-1$인 s로 대치하면?

1. 지수가 2의 거듭제곱인 경우 ex) A^{32}

→ A^{32} → A^2, A^4, A^8 .. 이런 식으로 다섯번만에 가능

↳ 지수가 높아도 유용

2. 대수적 조작을 가해 더 효율적인 계산 형태로 변경

↳ +의 흡수 : $A+A=A$

+의 교환 : $A+B=B+A$

+와 ×의 결합 : $A+(B+C)=(A+B)+C$ $\quad A(BC)=(AB)C$

+의 ×에 대한 분배 : $A(B+C)=AB+AC$ $\quad (B+C)A=BA+CA$

곱셈에 대한 항등원 : $IA=AI=A$

s가 $s \geq n-1$를 만족하는 가장 작은 거듭제곱일 때,

$$R = \sum_{p=0}^{s} A^p = I + A + A^2 + \cdots + A^s$$

〈정리 9.7〉

A : 이진관계를 나타내는 $n \times n$ 행렬. A의 이행 폐쇄를 나타내는 행렬 R : 임의의 $s \geq n-1$에 대해 $(I+A)^s$

↳ $\theta(n^3)$보다 작은 시간에 가능