Discrete Mathematics (0034)
Lecture Notes for Final Exam

Yulwon Rhee (202211342)

Department of Computer Science and Engineering, Konkuk University

9 Week 9

9.1 관계

- 서로 다른 두 집합에 속하는 원소들 간의 순서(Order)를 표현
- 순서쌍 집합(곱집합의 부분 집합)에 속하면서 순서쌍을 이루는 원소들은 '관계'가 있다

9.2 이항 관계(Binary Relation)

집합 A에서 집합 B로 가는 관계
R: $A \times B$의 부분 집합
$a \in A, b \in B$일 때, $(a,b) \in R \to {}_aR_b$; $(a,b) \notin R \to {}_a\not{R}_b$

정의역(Domain): $\mathrm{dom}(R) = \{a|a \in A\}$
공변역(Codomain): $\mathrm{codom}(R) = \{b|b \in B\}$
치역(Range): $\mathrm{ran}(R) = \{b|(a,b) \in R\} \subseteq B$

n항 관계(n-ray Relation): $A_1 \times A_2 \times \cdots \times A_n$의 부분 집합, $R \subseteq A_1 \times \cdots \times A_n$

역관계(Inverse Relations): B에서 A로의 관계, $R^{-1} = \{(b,a)|(a,b) \in R\}$, ${}_aR_b$ 존재 $\to {}_bR_a^{-1}$ 존재

9.3 관계의 표현: 서술식 방법

e.g.) $A = 1,2,3$에서 원소 a,b가 $a \geq b$인 관계 R

9.4 관계의 표현: 나열식 방법

- 화살표 도표(Arrow diagram)
- 좌표 도표(Coordinate diagram):
  - 집합 A의 원소를 x축 위의 점으로, B의 원소를 y축 위의 점으로 표시
- 방향 그래프(Directed graph):
  - 관계 R이 하나의 집합 A에 대한 관계 표현일 때
  - A의 각 원소 $\Rightarrow$ 그래프의 정점(Vertex)
  - $(a,b) \in R$이면 a에서 b로 화살표가 있는 연결선(Edge)로 표현

- 관계 행렬(Relation matrix):
  - 부울(Boolean) 행렬 이용
  - $A = \{a_1, a_2, \cdots, a_m\}$에서 $B = \{b_1, b_2, \cdots, b_n\}$로 가는 관계 R에 대한 $m \times n$ 행렬 $M_R = [m_{ij}]$

$$m_{ij} = \begin{cases} 1, (a_i, b_j) \in R \\ 0, (a_i, b_j) \notin R \end{cases}$$

  - e.g.) $A = \{1,2,3\}$과 $B = \{a,b\}$의 이항 관계
    $R = \{(1,b),(2,a),(2,b),(3,a)\}$

$$M_R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} a\ b \\ \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \end{array} \qquad M_{R^{-1}} = \begin{array}{c} \\ a \\ b \end{array}\!\!\begin{array}{c} 1\ 2\ 3 \\ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}$$

9.5 관계의 성질

반사 관계(Reflexive Relation): 모든 $a \in A$에 대해 $(a,a) \in R$인 관계, $\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$

비반사 관계(Irreflexive Relation): 모든 $a \in A$에 대해 $(a,a) \notin R$인 관계, $\begin{bmatrix} 0 & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix}$

반사 관계도 비반사 관계도 아닌 경우: $\begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$

대칭 관계(Symmetric Relation):
$\exists a,b \in A$에 대해 $(a,b) \in R$이면 $(b,a) \in R$, (a,b) 존재 $\to$ (b,a) 존재
관계 행렬에서 대각 성분 기준으로 대칭이면 대칭 관계 성립

반대칭 관계:
$\exists a,b \in A$에 대해 $a = b$이고, $(a,b) \in R$이면 $(b,a) \in R$
$a \neq b$이고, $(a,b) \in R$이면 $(b,a) \notin R$

추이 관계: $\exists a,b,c \in A$에 대해 $(a,b) \in R$이고, $(b,c) \in R$이면 $(a,c) \in R$인 관계

9.6 합성 관계(Composite Relation)

A에서 B로의 관계 R_1과, B에서 C로의 관계 R_2에 대해서, A에서 C로의 합성 관계 $= R_1 \cdot R_2$ 또는 R_1R_2

$$R_1 \cdot R_2 = \{(a,c) | a \in A, c \in C, (a,b) \in R_1 \text{이고 } (b,c) \in R_2\}$$

합성 관계의 연산: $R \cdot S = M_{R \cdot S} = M_R \odot M_S$

e.g.) $M_R = \begin{array}{c c} & \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \end{array}$ $\quad M_S = \begin{array}{c c} & \begin{array}{ccc} x & y & z \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}$

$$R \cdot S = M_R \odot M_S = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} (0\wedge 1)\vee(1\wedge 0)\vee(0\wedge 1) & (0\wedge 0)\vee(1\wedge 1)\vee(0\wedge 1) & (0\wedge 1)\vee(1\wedge 1)\vee(0\wedge 0) \\ (0\wedge 1)\vee(1\wedge 0)\vee(1\wedge 1) & (0\wedge 0)\vee(1\wedge 1)\vee(1\wedge 1) & (0\wedge 1)\vee(1\wedge 1)\vee(1\wedge 0) \\ (1\wedge 1)\vee(0\wedge 0)\vee(0\wedge 1) & (1\wedge 0)\vee(0\wedge 1)\vee(0\wedge 1) & (1\wedge 1)\vee(0\wedge 1)\vee(0\wedge 0) \\ (1\wedge 1)\vee(1\wedge 0)\vee(1\wedge 1) & (1\wedge 0)\vee(1\wedge 1)\vee(1\wedge 1) & (1\wedge 1)\vee(1\wedge 1)\vee(1\wedge 0) \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

합성 관계의 거듭제곱 $R^n = \begin{cases} R & (n=1) \\ R^{n-1} \cdot R & (n>1) \end{cases}$

기타 연산

- $R_1 \cap R_2 = M_{R_1 \cap R_2} = M_{R_1} \wedge M_{R_2}$
  $= \{(a,b) \in R_1 \cap R_2 | (a,b) \in R_1 \wedge (a,b) \in R_2\}$
- $R_1 \cup R_2 = M_{R_1 \cup R_2} = M_{R_1} \vee M_{R_2}$
  $= \{(a,b) \in R_1 \cup R_2 | (a,b) \in R_1 \vee (a,b) \in R_2\}$
- $R_1 - R_2 = M_{R_1 - R_2} = \{(a,b) \in R_1 - R_2 | (a,b) \in R_1 \wedge (a,b) \notin R_2\}$

9.7 추이 관계와 합성 관계

[정리] 추이 관계와 거듭제곱의 관계

집합 A에 대한 관계 R이 추이 관계일 필요충분조건은 모든 양의 정수 n에 대하여 $R^n \subseteq R$이다.

9.8 폐포(Closure)

폐포(Closure): A상의 관계 R이 어떤 성질을 만족하지 않을 때, 그 성질을 만족하도록 순서쌍들을 추가하여 R^*(원하는 성질이 만족되는 가장 작은 집합)로 확장

성질 P에 대한 관계 R의 폐포: A에 대한 관계 R에 대해, R^*가 R을 포함하면서 성질 P를 가질 때, R^*는 P에 대한 R의 폐포

9.9 반사 폐포(Reflexive Closure)

A에 대해, R을 포함하면서 반사 관계를 갖는 관계 S
$S = R \cup \{(a,a)|a \in A\}$

9.10 대칭 폐포(Symmetric Closure)

A에 대해, R을 포함하면서 대칭 관계를 갖는 관계 S
$S = R \cup \{(b,a) \in A \times A|(a,b) \in R\} = R \cup R^{-1}$

9.11 추이 폐포(Transitive Closure)

A에 대해, R을 포함하면서 추이 관계를 갖는 관계 S
$S = R \cup \{(a,c) \in A \times A|(a,b) \in R \ \wedge (b,c) \in R\}$

9.12 연결 관계(Connectivity Relation) R^*

$R^* = \bigcup_{n=1}^{\infty} R^n = R^1 \cup R^2 \cup \cdots \cup R^n$
연결 관계 R^*는 R의 추이 폐포

[정리] R이 n개의 원소를 갖는 집합에 대한 관계이고, M_R을 관계 R에 대한 부울 행렬이라고 했을 때, R의 추이 폐포 R^*는

$$M_{R^*} = M_R \vee M_{R^2} \vee M_{R^3} \vee \cdots \vee M_{R^n}$$

9.13 예제 풀이

양의 정수(Positive Integer) 집합에서 두 원소 a,b에 대해서 'a가 b를 나눈다'라는 관계는 어떤 성질을 만족하는가? (관계(1) 강의 참조)

10 Week 10

10.1 동치 관계(Equivalence relation)

동치 관계: 반사 관계, 대칭 관계, 추이 관계가 모두 성립하는 경우

10.2 동치류(Equivalence Class) $[a]$

A에 대한 관계 R이 동치 관계일 때, R에 대한 a의 동치류: a와 순서쌍을 이루는 원소들의 집합
$[a] = \{x|(a,x) \in R\}$

DM-06-관계(2)_2 (2022) 29:57부터...

함수 파트는 나중에 따로 봐야지..

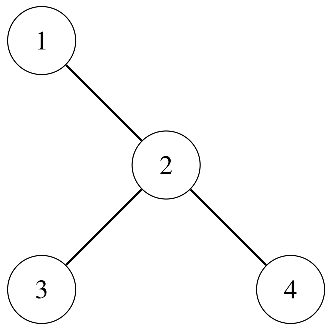
11 Week 11

11.1 그래프(Graph)

공집합이 아닌 정점(Vertex or Node)의 집합 V와 서로 다른 정점의 쌍 (v_i, v_j)를 연결하는 변 또는 연결선(Edge)의 집합 E로 구성되는 구조 G

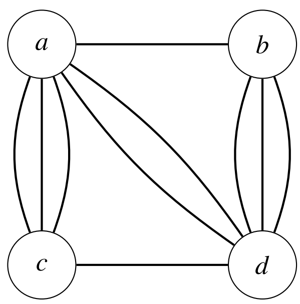
$G = (V, E)$
$V = \{v_1, v_2, \cdots, v_n\}$
$E = \{e_1, e_2, \cdots, e_m\} = \{(v_i, v_j), \cdots\}$

e.g.)



$V = \{1, 2, 3, 4\},\ E = \{(1,2), (2,3), (2,4)\}$

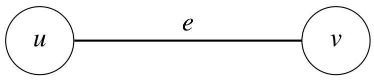
무방향 그래프(Undirected Graph): 특별한 언급 없으면 무방향 그래프
방향 그래프(Directed Graph, Digraph): 선행자? 후속자?
단순 그래프(Simple Graph): 루프가 없는 그래프
멀티 그래프(Multigraph):



연결 그래프(Connected Graph): 모든 Vertex가 연결된 그래프, 모든 Vertex간 경로 존재
강한 연결 그래프(Strongly Connected Graph): 방향 그래프에서만, 모든 두 Vertex v_1, v_2에 대해 $v_1 \leftrightarrow v_2$
연결 요소(Connectivity Component): 그래프에서 모든 Vertex들이 연결되어 있는 부분 그래프
연결 수(Connectivity Number): G에서 연결 요소 개수

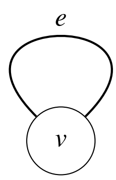
11.2 그래프 용어

인접(Adjacent)과 근접(Incident):



u,v는 서로 Adjacent, e는 u,v에 Incident

루프(Loop):



근접하는 점이 같은 e

경로(Path): Vertex들의 열(Sequence) $v_1, v_2, \cdots, v_n$에서 $(v_{k-1}, v_k) \in E, 1 \le k < n$, 경로의 길이는 $k-1$
 단순 경로(Simple Path): 같은 Edge를 두 번 포함하지 않는 경로
 기본 경로(Elementary Path): 같은 Node를 두 번 포함하지 않는 경로

사이클(Cycle) 또는 순회(Circuit): $v_1 = v_k (k \neq 1)$인 경로, 종점 == 시점
 단순 사이클(Simple Cycle): 같은 Edge를 반복해 방문하지 않는 사이클
 기본 사이클(Elementary Cycle): 시점 제외 어떤 Node도 반복해 방문하지 않는 사이클

길이(Length): 경로 또는 사이클을 구성하는 Edge의 수

차수(Degree) $d(v)$: Vertex v에 근접하는 Edge의 수, Loop는 두 개로 Count
 홀수점(Odd Vertex): 차수가 홀수인 Vertex
 짝수점(Even Vertex): 차수가 짝수인 Vertex

외차수(Out-degree) out$-d(v)$: 방향 그래프에서 Vertex v에서 시작하는 화살표 수

내차수(In-degree) in$--d(v)$: 방향 그래프에서 Vertex v에서 끝나는 화살표 수

[정리] 차수에 대한 정리
- $G = (V, E)$에서, 모든 Vertex의 차수의 합은 Edge의 수의 두 배

$$\sum_{v \in V} d(v) = 2|E|$$

- $G = (V, E)$에서, 차수가 홀수인 정점의 수는 짝수

11.3 오일러 경로(Eulerian Path) 및 오일러 회로(Eulerian Circuit)

오일러 경로: 멀티 그래프에서 모든 Edge들을 한 번씩만 통과하는 경로를 찾는 문제
오일러 회로: Node는 여러 번 통과할 수 있지만, Edge는 한 번씩만 통과하는 사이클

어떤 그래프 G가 오일러 경로를 가지기 위한 필요충분조건은 G가 연결 그래프이고, 홀수 차수의 개수가 0 또는 2인 경우이다.

어떤 그래프 G가 오일러 회로를 가지기 위한 필요충분조건은 G가 연결 그래프이고, 모든 Node들이 짝수 개의 차수를 가지는 경우이다.

11.4 해밀턴 경로(Hamiltonian Path) 및 해밀턴 회로(Hamiltonian Circuit)

해밀턴 경로: 그래프에서 모든 Node를 오직 한 번씩만 지나지만 시점으로 돌아오지 않는 경로
해밀턴 회로: 그래프에서 모든 Node를 오직 한 번씩만 지나는 순회
해밀턴 회로에 대한 충분 조건:

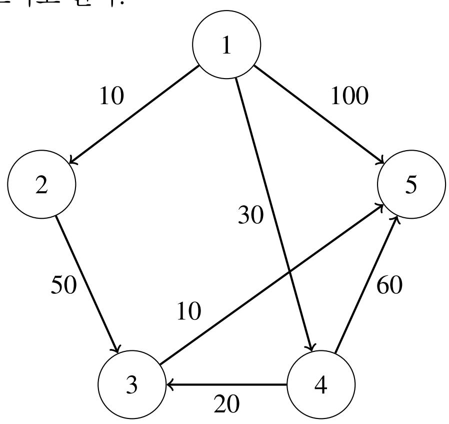
- 차수 1을 갖는 Node를 가진 그래프는 해밀턴 순환을 가질 수 없다.
- 차수가 2인 Node에 근접하는 두 Vertex는 해밀턴 순환에 포함된다.
- 한 Node에 근접하는 두 Vertex가 해밀턴 순환에 포함되면, 그 Node에 근접한 다른 Vertex는 해밀턴 순환에 포함될 수 없다.

Ore's Theorem: $n \geq 3$일 때, n개의 Node를 갖는 단순 연결 그래프 G에서 인접하지 않은 임의의 정점 u, v에 대해 $d(u) + d(v) \geq n$이면 G는 해밀턴 그래프이다.

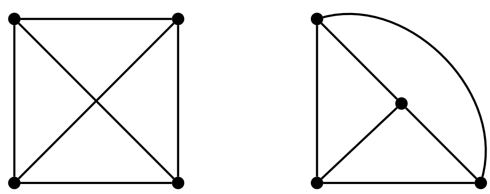
Dirac's Theorem: $n \geq 3$일 때, n개의 Node를 갖는 단순 연결 그래프 G에서 임의의 정점 v에 대해 $2d(v) \geq n$이면 G는 해밀턴 그래프이다.

11.5 특수 형태의 그래프

가중 그래프(Weight Graph): G의 각 Edge에 0보다 큰 수가 할당되었을 때, 이 값을 가중값(Weight)이라고 하며, 이를 가중 그래프라고 한다.



동형 그래프(Isomorphic Graph): $G_1 = (V_1, E_1)$과 $G_2 = (V_2, E_2)$가 주어졌을 때, 전단사 함수 $f : V_1 \to V_2$가 존재하여 $\{u, v\} \in E_1 \Leftrightarrow \{f(u), f(v)\} \in E_2$이면 f를 동형(Isomorphism)이라고 하고, G_1과 G_2를 동형 그래프라 한다.

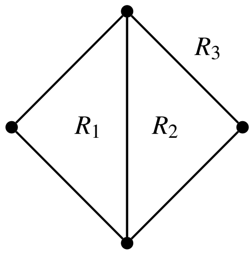


11.6 오일러의 정리

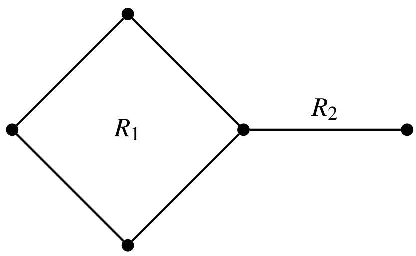
연결된 평면 그래프 G에서 Vertex의 수를 v, Edge의 수를 e, 면(Space)의 수를 s라고 할 때, $v - e + s = 2$

11.7 평면 그래프

평면 그래프(Planar Graph): $G = (V, E)$를 평면에 그릴 때, 교차하지 않는 그래프
면 f의 차수 $d(f)$: 평면 그래프의 면 f의 경계를 이루는 변의 수
e.g.)



$$d(R_1) = 3,\ d(R_2) = 3,\ d(R_3) = 4$$
$$d(R_1) + d(R_2) + d(R_3) = 2e$$



$$d(R_1) = 4,\ d(R_2) = 6$$
$$d(R_1) + d(R_2) = 2e$$

평면 그래프의 면의 차수의 총 합은 변의 수의 두 배이다.

$$2e = \sum_{f=1}^{n} d(f)$$

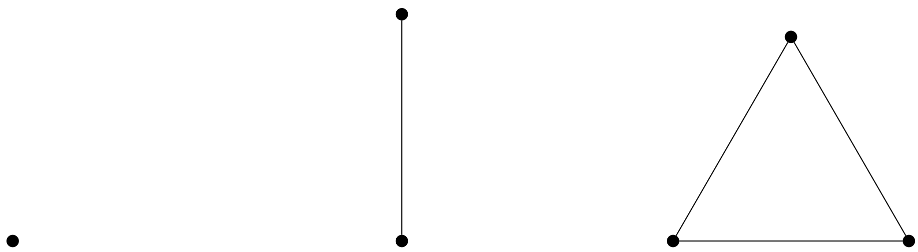
연결된 평면 단순 그래프의 Vertex의 수를 v, Edge의 수를 e라 할 때, $v \geq 3$이면 $e \leq 3(v-2)$
모든 면의 차수는 3이상이므로, $2e = \sum_{f=1}^{n} d(f) \geq 3f,\ 2 = v - e + f \leq c - e + \frac{2}{3}e \leq v - \frac{1}{3}e$

12 Week 12

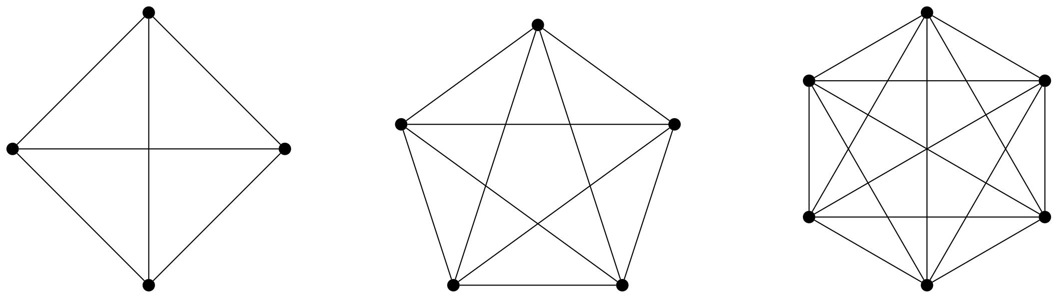
12.1 특수 형태의 그래프

완전 그래프(Complete Graph): 모든 n개의 Vertex들의 쌍 사이에 Edge가 존재하는 $G = K_n$

e.g.)



(a) K_1 (b) K_2 (c) K_3

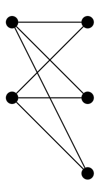


(d) K_4 (e) K_5 (f) K_6

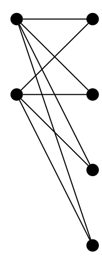
이분 그래프(Bipartite Graph): V가 X와 $Y = V - X$로 나누어져, 각 Edge가 X 내의 Vertex와 Y 내의 Vertex의 쌍으로 연결될 때, $G = (V, E)$

완전 이분 그래프(Complete Bipartite Graph): X 내의 모든 Vertex와 Y 내의 모든 Vertex 사이에 Edge가 존재할 때, 그래프 G

e.g.)



(a) $K_{2,3}$



(b) $K_{2,4}$

12.2 그래프의 표현 방법

인접 행렬(Adjacency Matrix): $G = (V, E)$에서, $|V| = n$일 때, G의 인접 행렬은 $n \times n$ 행렬 A

$$a_{ij} = \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

인접 리스트(Adjacency List): 각 Vertex에 대해 포인터가 주어지고, 그 Vertex로 부터 인접한 Vertex들을 모두 Linked List에 담음 (List 내에서는 순서에 관계가 없음)

Linked List는 Node의 연결로 표현

- Head: Linked List의 시작 Node (그래프의 각 Vertex들)
- Node: 데이터 필드 + 포인트 필드 (다음에 연결된 Node의 주소 저장)
- 마지막 Node의 포인트 필드는 `null`

12.3 그래프의 응용 및 활용

최단 경로 찾기(Shortest Path Problem): 다익스트라(Dijkstra) 알고리즘

해밀턴 순회의 응용: 일반적인 해결 알고리즘 존재 X $\rightarrow$ 최근접 이웃 방법(Nearest Neighbour Method, Greedy 알고리즘)

12.4 그래프의 탐색(Traversal)

깊이 우선 탐색(Depth First Search; DFS):
- 시작 Vertex v에서 인접 Vertex 중 방문하지 않은 Vertex w 방문
- w에서 다시 인접 Vertex 중 방문하지 않은 Vertex u 방문 반복
- 어떤 Vertex $\boldsymbol{v}$ 방문 후 $\boldsymbol{v}$에 인접한 모든 Vertex 방문한 경우, 바로 이전 Vertex로 돌아가 위 반복
- 모든 Vertex 방문 후 탐색 종료

Stack 사용 or 재귀 알고리즘으로 구현

너비 우선 탐색(Breath First Search; BFS): 처음 방문한 Vertex와 인접한 Vertex들을 차례로 방문
- 시작 Vertex v에서 인접한 Vertex 모두 차례로 방문
- 더 이상 방문할 Vertex가 없을 때, 다시 v에 인접한 Vertex중 처음 방문한 Vertex와 인접한 Vertex 방문
- v에 인접한 Vertex중 두 번째 방문한 Vertex와 인접한 Vertex 방문 반복
- 모든 Vertex 방문 후 탐색 종료

Queue 사용

12.5 트리

트리(Tree):
- A connected undirected graph with no circuits
- An undirected graph is a tree iff there is a unique simple path between any two of its vertices
- 특별히 지정된 노드인 루트가 있고, 나머지 노드들은 다시 각각 트리이면서 연결되지 않는(disjoint) $T_1, T_2, \cdots, T_n (N \geq 0)$으로 나누어 진다.
- 이 때 $T_1, T_2, \cdots, T_n$을 루트의 서브 트리(Subtree)라고 한다.

루트(Root): 주어진 트리의 시작 노드, 트리의 가장 높은 곳에 위치
차수(Degree): 각 노드의 서브 트리의 개수
잎 노드 or 단말 노드(Leaf Node): 차수가 0인 노드
자식 노드(Children Node): 어떤 노드의 서브 트리의 루트 노드
부모 노드(Parent Node): 자식 노드의 반대
형제 노드(Sibling Node): 동일한 부모를 가지는 노드
중간 노드(Internal Node): 루트도 아니고 잎 노드도 아닌 노드
조상(Ancestor): 루트로부터 각 노드에 이르는 경로 상에 나타난 모든 노드들
자손(Descendant): 각 노드부터 잎 노드에 이르는 경로 상에 나타난 모든 노드들
레벨(Level): 루트의 레벨 $= 0$, 자손 노드로 내려가며 ++
트리의 높이 or 깊이(Height or Depth): 트리에서 노드가 가질 수 있는 맥스 레벨
숲(Forest): 서로 연결되지 않는 트리들의 집합, 트리에서 루트를 제거 $\rightarrow$ 숲 생성

G는 트리
$\equiv G$는 연결되어 있고, $M = n - 1$
$\equiv G$는 연결되어 있고, 어느 한 연결선만을 제거하더라도 G는 연결되지 않음
$\equiv G$는 사이클을 가지지 않고, $m = n - 1$
$\equiv G$는 어느 한 연결선만 첨가하더라도 사이클 형성

12.6 이진 트리

이진 트리(Binary Tree): 노드들의 유한 집합, 공집합이거나, 루트와 왼쪽 서브 트리, 오른쪽 서브 트리로 이루어짐
사향 이진 트리(Skewed Binary Tree): 왼쪽 or 오른쪽으로 편향된 트리
완전 이진 트리(Complete Binary Tree): 높이가 k일 때 레벨 1부터 $k-1$까지는 모두 차있고 레벨 k에서는 왼쪽 노드부터 차례로 차있는 이진 트리
포화 이진 트리(Full Binary Tree): 잎 노드가 아닌 것들은 모두 2개씩 자식 노드를 가지며 트리의 높이가 일정할 때

이진 트리가 레벨 i에서 가질 수 있는 최대 노드 수 $= 2^i$
높이가 k인 이진 트리가 가질 수 있는 최대 전체 노드 수 $= 2^{k+1} - 1$
잎 노드 개수 $= n_0$, 차수가 2인 노드 개수 $= n_2$ 일 때, $n_0 = n_2 + 1$ 항상 성립

12.7 이진 트리의 표현

배열:
- 트리의 중간에 새로운 노드를 삽입하거나 기존의 노드를 지울 때 비효율적
- 높이가 h인 이진 트리는 각 노드 번호를 인덱스로 하여 1차원 배열로 구현 가능 (인덱스는 1부터 시작)
- 노드 인덱스 n의 부모 인덱스 $= \left\lfloor \frac{n}{2} \right\rfloor$
- 노드 인덱스 n의 왼쪽 자식 인덱스 $= 2n$, 오른쪽 자식 인덱스 $= 2n + 1$

연결 리스트: 일반적으로 가장 많이 사용. 중간 데이터, 왼쪽 자식 포인터, 오른쪽 자식 포인터 저장

13 Week 13

13.1 이진 트리의 탐방

트리의 각 노드를 꼭 한 번씩만 방문(Traversal)하는 방법

- 각 노드와 그 노드의 서브 트리를 같은 방법으로 탐방
- 전순위, 중순위, 후순위 탐방 기법
- D: 노드, L: 노드의 왼쪽 서브 트리, R:노드의 오른쪽 서브 트리
- 왼쪽을 오른쪽보다 항상 먼저 방문한다고 가정
- 중순위: LDR
- 전순위: DLR
- 후순위: LRD
- 수식 표현에서 중순위 표기(Infix), 전순위 표기(Prefix), 후순위 표기(Postfix)와 각각 대응

탐방의 결과, 각 노드에 들어있는 데이터를 차례로 나열

중순위:

```
void inOrder(TREE* currentNode) {
    if (currentNode != NULL) {
        inOrder(currentNode->leftChild);
        std::cout << currentNode->data;
        inOrder(currentNode->rightChild);
    }
}
```

전순위:

```
void preOrder(TREE* currentNode) {
    if (currentNode != NULL) {
        std::cout << currentNode->data;
        preOrder(currentNode->leftChild);
        preOrder(currentNode->rightChild);
    }
}
```

후순위:

```
void postOrder(TREE* currentNode) {
    if (currentNode != NULL) {
        postOrder(currentNode->leftChild);
        postOrder(currentNode->rightChild);
        std::cout << currentNode->data;
    }
}
```

13.2 순회 표기 & 수식 트리

중순위: $(a+b)\times(c-d)$
전순위: $\times + ab - cd$
후순위: $ab + cd - \times$

전순위 수식 $+ - \times\ 2\ 3\ 5 \div \wedge\ 2\ 3\ 4 \Longrightarrow$ 중순위 수식 $2\times 3-5+2^3\div 4$
후순위 수식 $7\ 2\ 3\times -4\wedge 9\ 3\div + \Longrightarrow$ 중순위 수식 $(7-2\times 3)^4+9\div 3$

후순위 표기식과 스택을 활용하여 수식 트리 생성:
후순위 표기식이 주어지면 스택에 피연산자 저장
연산자를 만나면 스택에서 두 개의 피연산자 pop() 후 연산 결과(트리) 다시 저장

13.3 생성 트리와 최소 비용 생성 트리

생성 트리(Spanning Tree): $\exists G$에서 모든 노드들을 포함하는 트리
비용(Cost): 트리 연결선의 값의 합
최소 비용 생성 트리(Minimum Spanning Tree: MST): 생성 트리 중 최소 비용

Prim's Algorithm:
- $G=(V,E)$에서 $V=\{1,2,\cdots,n\}$
- 노드의 집합 U를 1로 시작
- $u\in U,\ v\in V-U$일 때, U와 $V-U$를 연결하는 사이클 형성 X인 가장 짧은 연결선 (u,v)를 찾아 v를 U에 포함시킴
- 위를 $U-V$까지 반복

```
void prim(graph G: set_of_edges T) {
    set_of_vertices U;
    vertex u, v;
    T = NULL;
    U = {1};
    while (U != V) {
        let (u, v) be a lowest cost edge such that u is in U and v is in
↪ V - U;
        if ((u, v) does not create a cycle) {
            T = T ∪ {(u, v)};
            U = U ∪ {v};
        }
    }
}
```

Kruskal Algorithm:

- $G = (V,E)$에서 $V = \{1,2,\cdots,n\}$, $T =$ (연결선의 집합)
- Let $T = \varnothing$
- E를 비용이 적은 순서로 정렬
- 가장 최솟값 가진 연결선 (u,v) 차례로 찾아 사이클 형성 X이면 T에 포함
- 위를 $|T| = |V| - 1$까지 반복

```
void kruskal(graph G: set_of_edges T) {
    T = NULL;
    while (T contains less than n - 1 edges and E is not empty) {
        choose an edge (v, w) from E of lowest cost;
        delete (v, w) from E;
        if ((v, w) does not create a cycle in T)
            add (v, w) to T;
        else discard (v, w);
    }
    if (T contains fewer than n - 1 edges)
        std::cout << "No Spanning Tree";
}
```

13.4 트리의 활용

문법의 파싱(Parsing)

허프만 코드(Huffman Code):

- 알파벳 문자를 0과 1의 비트 코드로 Encoding
- 문자의 발생 빈도에 따라 코드의 길이를 다르게 → 통신의 효율성
- 접두어 성질: 어떤 문자 코드도 다른 문자 코드의 접두어 코드가 아님

Huffman Algorithm:

- 발생 빈도가 가장 낮은 두 문자를 선택해 하나의 이진 트리로 연결
  - 왼쪽 노드에는 빈도수 낮은 문자, 오른쪽 노드에는 빈도수 높은 문자
  - 그 두 문자의 루트 노드는 두 문자의 빈도의 합
  - 문자들을 이진 트리로 연결
  - 그 후에 이진 트리들을 연결
- 위 과정을 모든 문자가 하나의 이진 트리로 묶일 때까지 반복
- 생성된 이진 트리의 왼쪽 노드는 0, 오른쪽 노드에는 1 부여
- 루트부터 해당 문자까지 0 또는 1을 순서대로 나열한 것이 해당 문자의 허프만 코드

14 Week 14

14.1 이진 탐색 트리

이진 탐색 트리(Binary Search Tree):

- 모든 노드 x에 대하여 노드 y가 노드 x의 왼쪽 서브 트리에 있을 때 $y < x$이고, 노드 z가 노드 x의 오른쪽 서브 트리에 있으면 $x < z$인 이진 트리
- $<$는 순서 관계를 의미
- 높이 균형 이진 트리인 경우 탐색 시간은 $\log n$

14.2 결정 트리

결정 트리(Decision Tree): 8개의 동전 문제

14.3 부울 대수