

자료구조를 구조하자

2019.11.08

32153180 이상민

32162436 신창우 32163006 이건욱

32164420 조정민 32164959 허전진

학습 내용

가중치 그래프

가중치 그래프: 정점에 가중치 부여 가능

가중치 그래프의 특징:

- 1. 가중치 부여 가능
- 2. 가중치 부여 시, 가중치 부여된 정점은 가중치 부여된 정점의 가중치에 따라 정렬된다.
- 3. 가중치 부여 시, 가중치 부여된 정점은 가중치 부여된 정점의 가중치에 따라 정렬된다.

6.2 기본적인 그래프 연산

(1) 그래프의 연산 (Graphs and Graphs)

가중치 그래프의 연산:

- 1. 그래프의 연산
- 2. 그래프의 연산
- 3. 그래프의 연산
- 4. 그래프의 연산
- 5. 그래프의 연산

깊이 우선 탐색의 예

깊이 우선 탐색의 예:

- 1. 그래프의 연산
- 2. 그래프의 연산
- 3. 그래프의 연산
- 4. 그래프의 연산
- 5. 그래프의 연산

깊이 우선 탐색 알고리즘

깊이 우선 탐색 알고리즘:

- 1. 그래프의 연산
- 2. 그래프의 연산
- 3. 그래프의 연산
- 4. 그래프의 연산
- 5. 그래프의 연산

인접 행렬의 성질

인접 행렬의 성질:

- 1. 인접 행렬의 성질
- 2. 인접 행렬의 성질
- 3. 인접 행렬의 성질
- 4. 인접 행렬의 성질
- 5. 인접 행렬의 성질

인접 리스트 (adjacency list)

인접 리스트 (adjacency list):

- 1. 인접 리스트 (adjacency list)
- 2. 인접 리스트 (adjacency list)
- 3. 인접 리스트 (adjacency list)
- 4. 인접 리스트 (adjacency list)
- 5. 인접 리스트 (adjacency list)

인접 리스트의 클래스 선언

인접 리스트의 클래스 선언:

```

class AdjacencyList {
public:
    AdjacencyList(int n): n(n) {}
    ~AdjacencyList() {}
    void AddEdge(int u, int v, int w) {
        // Add edge (u, v) with weight w
    }
    void RemoveEdge(int u, int v) {
        // Remove edge (u, v)
    }
    void GetEdge(int u, int v, int w) {
        // Get edge (u, v) with weight w
    }
    void Print() {
        // Print the graph
    }
private:
    int n;
    vector<vector<int>>> adj;
};
    
```

인접 행렬과 인접 리스트의 특징

인접 행렬과 인접 리스트의 특징:

- 1. 인접 행렬과 인접 리스트의 특징
- 2. 인접 행렬과 인접 리스트의 특징
- 3. 인접 행렬과 인접 리스트의 특징
- 4. 인접 행렬과 인접 리스트의 특징
- 5. 인접 행렬과 인접 리스트의 특징

신창우

학습 내용

용어 및 정의

- 무방향 그래프: 노드와 노드 사이 무방향 간선을 가질 수 있다.

4 3 2 1 0 1 2 3 4

완전 그래프

- n개의 정점을 가진 무방향 그래프에서 모든 노드 쌍 사이에 간선이 존재하는 그래프를 완전 그래프라고 한다.

완전 그래프의 정점 수: n
 간선 수: $\frac{n(n-1)}{2}$
 $\frac{4(4-1)}{2} = 6$
 완전 그래프의 정점 수: n

인접 행렬의 성질

인접 행렬의 성질

- 인접 행렬의 대각선 요소는 0이다.
- 인접 행렬의 행과 열의 합은 정점의 차수이다.
- 인접 행렬의 역행렬은 존재하지 않는다.

인접 리스트(adjacency list)

인접 리스트

- 인접 리스트는 인접 행렬과 달리 정점의 차수에 관계없이 저장할 수 있다.
- 인접 리스트는 인접 행렬보다 메모리 효율적이다.
- 인접 리스트는 인접 행렬보다 검색이 빠르다.

인접과 부족

인접 그래프: 두 정점 사이에 간선이 존재하는 그래프를 인접 그래프라고 한다.

부족 그래프: 두 정점 사이에 간선이 존재하지 않는 그래프를 부족 그래프라고 한다.

부분 그래프(subgraph)

부분 그래프: 원래 그래프의 일부 정점과 간선만을 취한 그래프를 부분 그래프라고 한다.

인접 리스트의 클래스 선언

```

class AdjacencyList {
public:
    AdjacencyList(int n) {
        // 초기화
    }
    void AddEdge(int u, int v) {
        // 간선 추가
    }
    void Print() {
        // 인접 리스트 출력
    }
};
    
```

인접 행렬과 인접 리스트의 특징

인접 행렬: 모든 정점 쌍에 대해 간선의 존재 여부를 나타내는 행렬이다.

인접 리스트: 각 정점에 대해 인접한 정점들의 리스트를 저장하는 구조이다.

가중치 그래프

가중치 그래프: 각 간선에 가중치가 부여된 그래프를 가중치 그래프라고 한다.

6.2 기본적인 그래프 연산

그래프의 기본적인 연산은 다음과 같다.

1. 그래프의 정점 수와 간선 수를 구하기
2. 그래프의 인접 행렬을 구하기
3. 그래프의 인접 리스트를 구하기
4. 그래프의 최단 경로를 구하기
5. 그래프의 연결성을 구하기
6. 그래프의 색칠을 구하기

깊이 우선 탐색 알고리즘의 분석

깊이 우선 탐색 알고리즘의 시간 복잡도는 $O(V+E)$ 이다.

깊이 우선 탐색 알고리즘의 공간 복잡도는 $O(V)$ 이다.

너비 우선 탐색

너비 우선 탐색 알고리즘의 시간 복잡도는 $O(V+E)$ 이다.

너비 우선 탐색 알고리즘의 공간 복잡도는 $O(V)$ 이다.

깊이 우선 탐색의 예

깊이 우선 탐색 알고리즘의 예시이다.

깊이 우선 탐색 알고리즘

```

void DFS(int v) {
    visited[v] = true;
    for (int i = 0; i < adj[v].size(); i++) {
        int u = adj[v][i];
        if (!visited[u]) {
            DFS(u);
        }
    }
}
    
```

너비 우선 탐색의 예

너비 우선 탐색 알고리즘의 예시이다.

너비 우선 탐색 알고리즘

```

void BFS(int s) {
    queue<int> q;
    visited[s] = true;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = 0; i < adj[u].size(); i++) {
            int v = adj[u][i];
            if (!visited[v]) {
                visited[v] = true;
                q.push(v);
            }
        }
    }
}
    
```

이건 목록

학습 내용

```
bool Bst::Insert(const int &n)
{
    BstNode *p = root;
    BstNode *q = 0;

    while (p)
    {
        q = p;
        if (x == p->data)
        {
            cout << x << "가 이미 존재합니다" << endl;
            return false;
        }
        if (x < p->data)
            p = p->LeftChild;
        else
            p = p->RightChild;
    }

    p = new BstNode;
    p->LeftChild = p->RightChild = 0;
    p->data = x;

    if (!root)
        root = p;
    else if (x < q->data)
        q->LeftChild = p;
    else
        q->RightChild = p;
    return true;
}

bool Bst::Delete(const int &n)
{
    BstNode *p = root;
    BstNode *q = 0;

    while (p)
    {
        if (x < p->data)
        {
            q = p;
            p = p->LeftChild;
        }
        else if (x > p->data)
        {
            q = p;
            p = p->RightChild;
        }
        else
            break;
    }

    if (!p)
    {
        cout << "삭제할 노드가 없습니다." << endl;
        return false;
    }

    if (p->LeftChild == 0 && p->RightChild == 0)
    {
        if (x < q->data)
        {

```

```
bool Bst::Insert(const int &n) // 삽입 함수
{
    BstNode *p = root;
    BstNode *q = 0; // p를 뒤따라오는 노드

    while (p)
    {
        q = p;
        if (x == p->data) // 삽입하려는 x가 이미 존재할 경우
        {
            cout << x << "가 이미 존재합니다" << endl;
            return false;
        }
        if (x < p->data)
            p = p->LeftChild;
        else
            p = p->RightChild;
    }

    p = new BstNode;
    p->LeftChild = p->RightChild = 0; // p의 왼쪽 자식, 오른쪽 자식 NULL
    p->data = x;

    if (!root) // 빈 리스트
        root = p;
    else if (x < q->data)
        q->LeftChild = p;
    else
        q->RightChild = p;
    return true;
}

bool Bst::Delete(const int &n) // 삭제 함수
{
    BstNode *p = root;
    BstNode *q = 0; // p를 뒤따라오는 노드

    while (p) // 빈 리스트가 아닌 경우
    {
        if (x < p->data) // x가 p의 data보다 작은 경우
        {
            q = p;
            p = p->LeftChild; // 왼쪽 자식으로 이동
        }
        else if (x > p->data) // x가 p의 data보다 큰 경우
        {
            q = p;
            p = p->RightChild; // 오른쪽 자식으로 이동
        }
        else // x가 p의 data와 같은 경우
            break;
    }

    if (!p) // 일치하는 값이 없는 경우
    {
        cout << "삭제할 노드가 없습니다." << endl;
        return false;
    }

    if (p->LeftChild == 0 && p->RightChild == 0) // 삭제할 노드의 자식이 없는 경우(단말노드)
    {
        if (x < q->data)
        {

```

```
C:\Windows\system32\cmd.exe
이진탐색트리
<1> 삽입 <2> 삭제 <3> 탐색 <4> 중위우선순회 <0> 종료
1
입력할 키 개수 : 10
1번째 : 5
2번째 : 80
3번째 : 25
4번째 : 67
5번째 : 16
6번째 : 24
7번째 : 1
8번째 : 99
9번째 : 86
10번째 : 45
4
중위우선순회 출력
1 5 16 24 25 45 67 80 86 99
3
탐색할 값 입력 : 45
성공!
3
탐색할 값 입력 : 68
실패!
2
삭제할 값 입력 : 24
2
삭제할 값 입력 : 99
4
중위우선순회 출력
1 5 16 25 45 67 80 86
0
계속하려면 아무 키나 누르십시오 . . .
```

조정민

학습 내용

```

else // 찾는 값의 노드가 존재할 경우.
{
    if (p->LeftChild == 0 && p->RightChild == 0) // 단말노드일 경우.
    {
        if (p == root) // 루트노드일 경우.
            root = 0; // 루트삭제.
        else if (p->key < q->key) // p가 q의 왼쪽자식일 경우.
            q->LeftChild = 0; // q의 왼쪽자식 삭제.
        else // p가 q의 오른쪽자식일 경우.
            q->RightChild = 0; // q의 오른쪽자식 삭제.
    }

    else if (p->LeftChild && p->RightChild == 0)
        // 왼쪽 자식만을 갖는 비단말노드일 경우.
    {
        if (p == root) // 루트노드일 경우.
            root = p->LeftChild; // 루트에 p의 왼쪽자식 대입.
        else if (x < q->key) // p가 q의 왼쪽자식일 경우.
            q->LeftChild = p->LeftChild;
            // q의 왼쪽자식에 p의 왼쪽자식 연결.
        else // p가 q의 오른쪽자식일 경우.
            q->RightChild = p->LeftChild;
            // q의 오른쪽자식에 p의 왼쪽자식 연결.
    }

    else if (p->LeftChild == 0 && p->RightChild)
        // 오른쪽 자식만을 갖는 비단말노드일 경우.
    {
        if (p == root) // 루트노드일 경우.
            root = p->RightChild; // 루트에 p의 오른쪽자식 대입.
        else if (x < q->key) // p가 q의 왼쪽자식일 경우.
            q->LeftChild = p->RightChild;
            // q의 왼쪽자식에 p의 오른쪽자식 연결.
        else // p가 q의 오른쪽자식일 경우.
            q->RightChild = p->RightChild;
            // q의 오른쪽자식에 p의 오른쪽자식 연결.
    }

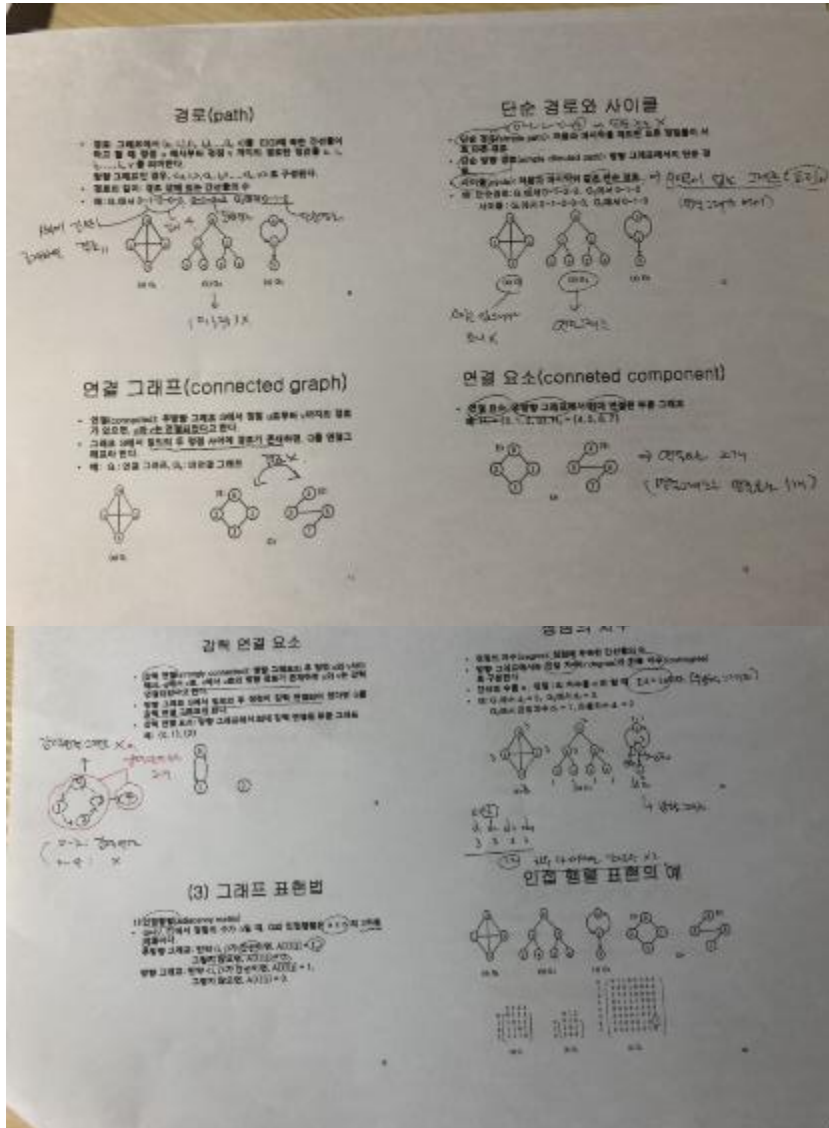
    else // 두 개의 자식을 갖는 노드일 경우.
    {
        r = p; // r에 p를 저장해둠.
        q = p; // q는 p의 선행 노드를 가리킴.
        p = p->RightChild; // p는 p의 오른쪽자식으로 이동.

        while (p->LeftChild)
        {
            // p는 가장 작은 값으로 이동하고 q는 p의 선행노드를 가리킴.
            q = p;
            p = p->LeftChild;
        }

        r->key = p->key; // key값 저장.
    }
}

```

히전진



참고자료

<https://www.youtube.com/watch?v=fVcKN42YXXI>

https://www.youtube.com/watch?v=_hxFgg7TLZQ

