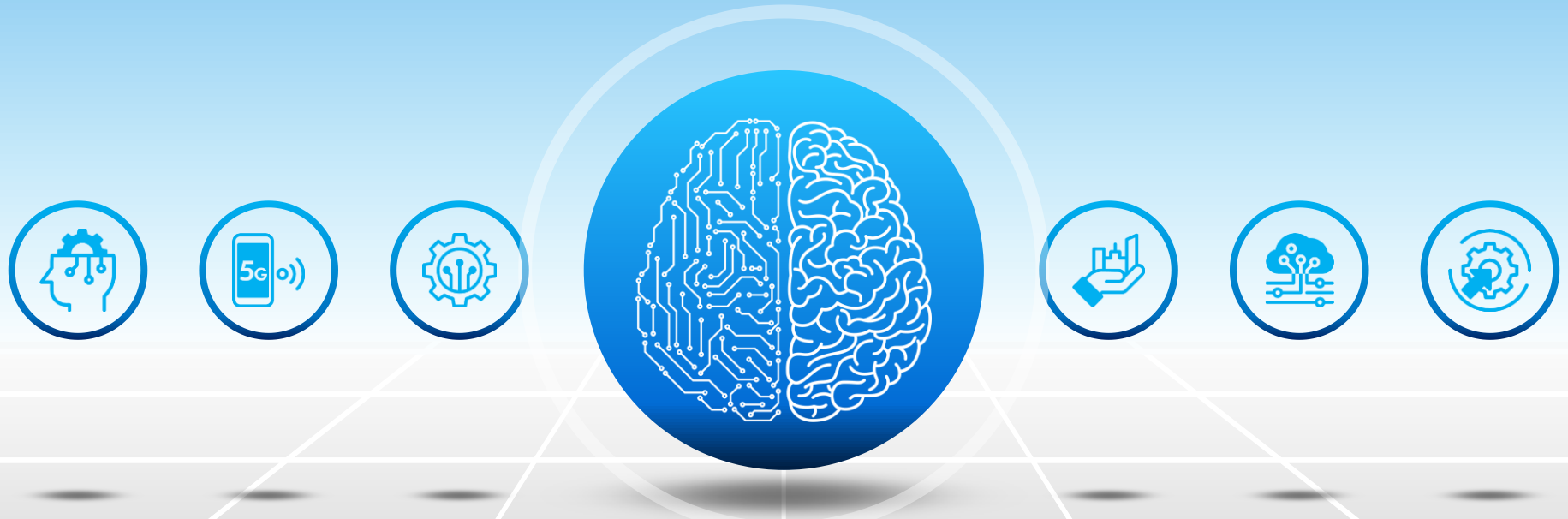
CIS3034 문제해결프로젝트

세트를 활용한 문제해결

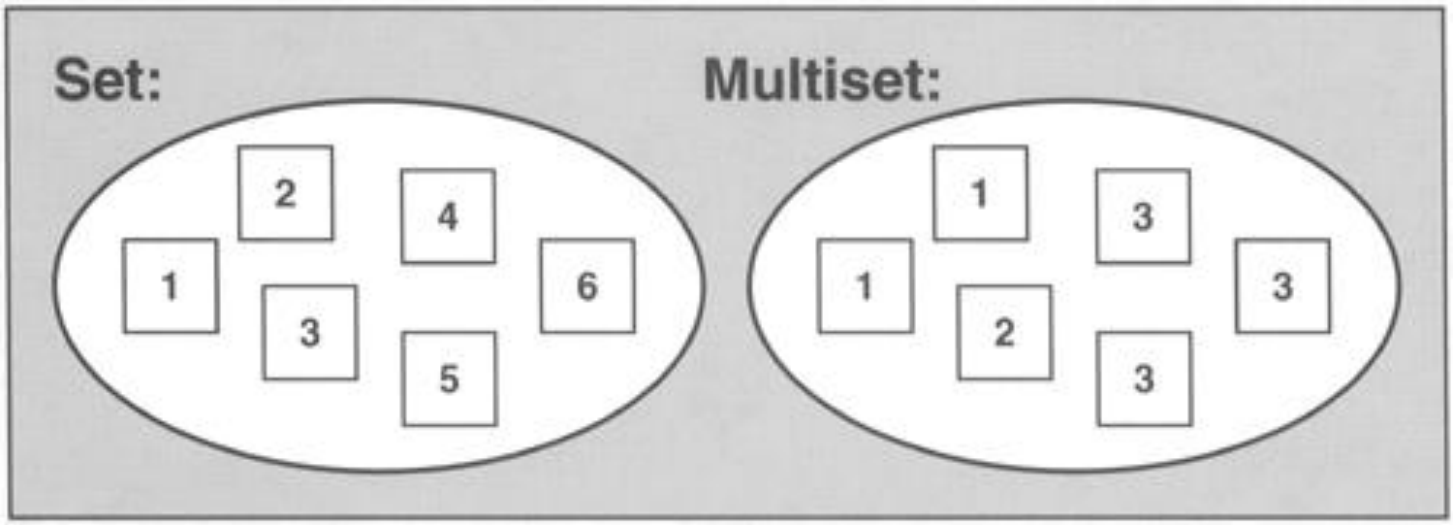


목 차
CONTENTS

세트의 이해

세트

- **세트(Set)**
  - 수학에서 배우는 집합의 개념과 같음
  - 즉 중복되지 않는 항목들이 모인 것 (중복을 허용하려면 Multiset을 사용해야 함)
  - 순서가 필요 없고, 고유 값을 원하는 경우 최선의 자료구조



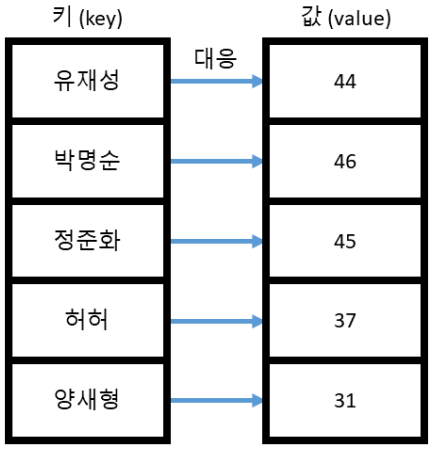
세트

- **세트의 특징**
  - 데이터를 비순차적(unordered)으로 저장할 수 잇는 자료구조
  - 삽입(insert) 순서대로 데이터가 저장되지 않는다.
  - 동일한 값을 여러 번 삽입하는 것이 불가능하다. 여러 번 삽입될 경우, 나중에 들어온 값으로 치환된다.

- **세트의 용도**
  - 중복된 값을 골라내야 할 때 (고유 값을 얻고자 할 때)
  - 빠른 Look Up을 해야 할 때 ($O(\log_2 N)$)
  - 순서는 상관없을 때

세트

- **세트 컨테이너**
  - 연관 컨테이너 중 하나임
  - 연관 컨테이너는 key를 바탕으로 이에 대응되는 값(value)를 얻을 수 있는 구조임.
  - 노드 기반 컨테이너이며 이진 트리로 구현됨
  - key 라고 불리는 원소들의 집합으로 이어진 컨테이너임 (원소 = key, 값 = value)
  - Key 값은 중복이 허용 안 됨
  - 원소가 insert 멤버 함수에 의해 삽입되면, 원소는 자동으로 정렬 됨.
  - Default 정렬 기준은 less(오름차순)



이름(key)을 바탕으로 나이(value)를 얻을 수 있다.

세트

- **세트의 사용 사례**
  - 헤더파일 추가

    #include <set>

  - 표준 네임스페이스 사용하면 편리

    using namespace std;

  - 세트 변수 선언

    set<datatype> 변수명;

    set<int> s;

기본형태

- `set<T>` : 원하는 자료형 및 클래스 T를 통해 생성

iterator(반복자)

- `begin()` : beginning iterator를 반환
- `end()` : end iterator를 반환

추가 및 삭제

- `insert(element)` : 세트에 element를 추가
- `erase(element)` : 세트에서 해당하는 element를 삭제
- `clear()` : 세트에 있는 모든 원소 삭제

조회

- `find(element)` : element에 해당하는 iterator를 반환
- `count(element)` : element에 해당하는 개수를 반환

기타

- `empty()` : 비어있으면 true 아니면 false를 반환
- `size()` : 세트에 포함되어 있는 원소들의 수를 반환

세트

- **세트의 사용 사례**

test) insert, insert(중복값), find(존재하는 값), find(존재하지 않는 값)

```cpp
#include<iostream>
#include<set>
using namespace std;

int main(void){
    set<int> s;

    s.insert(40);
    s.insert(10);
    s.insert(80);
    s.insert(30);
    s.insert(70);
    s.insert(60);
    s.insert(20);
    s.insert(50);

    set<int>::iterator iter;
    for(iter = s.begin(); iter != s.end(); iter++){
        cout << *iter << " " ;
    }
    cout << endl;

    //중복 값 넣어보기.
    s.insert(20);
    for(iter = s.begin(); iter != s.end(); iter++){
        cout << *iter << " " ;
    }
    cout << endl;

    //존재 하는 원소 찾기
    iter = s.find(30);
    if(iter != s.end()){
        cout << *iter << " : 존재 " << endl;
    }else{
        cout << "존재하지 않음 " << endl;
    }

    //존재 하지 않는 원소 찾기
    iter = s.find(333);
    if(iter != s.end()){
        cout << *iter << " : 존재 " << endl;
    }else{
        cout << "존재하지 않음 " << endl;
    }


    return 0;
}
```

기본형태

- `set<T>` : 원하는 자료형 및 클래스 T를 통해 생성

iterator(반복자)

- `begin()` : beginning iterator를 반환
- `end()` : end iterator를 반환

추가 및 삭제

- `insert(element)` : 세트에 element를 추가
- `erase(element)` : 세트에서 해당하는 element를 삭제
- `clear()` : 세트에 있는 모든 원소 삭제

조회

- `find(element)` : element에 해당하는 iterator를 반환
- `count(element)` : element에 해당하는 개수를 반환

결과

```
10 20 30 40 50 60 70 80
10 20 30 40 50 60 70 80
30 : 존재
존재하지 않음
```

세트

- 세트의 사용 사례

```cpp
#include <iostream>
#include <set>
#include <string>

using namespace std;

int main(){

    // set
    set<string> s;


    // insert(element)
    s.insert("abc");
    s.insert("def");
    s.insert("ghi");
    s.insert("jkl");


    // erase(element)
    s.erase("jkl");
```

```cpp
    // empty(), size()
    if(!s.empty()) cout << "s size : " << s.size() << '\n';


    // find(element)
    cout << *s.find("abc") << '\n';
    cout << *s.find("def") << '\n';


    // count(element)
    cout << "abc count : " << s.count("abc") << '\n';


    // begin(), end()
    cout << "traverse" << '\n';
    for(auto it = s.begin(); it != s.end(); it++){
        cout << "value : " << *it << '\n';
    }

    return 0;

}
```

```
s size:3
abc
def
abc count:1
traverse
value:abc
value:def
value:ghi
```

기본형태

- `set<T>` : 원하는 자료형 및 클래스 T를 통해 생성

iterator(반복자)

- `begin()` : beginning iterator를 반환
- `end()` : end iterator를 반환

추가 및 삭제

- `insert(element)` : 세트에 element를 추가
- `erase(element)` : 세트에서 해당하는 element를 삭제
- `clear()` : 세트에 있는 모든 원소 삭제

조회

- `find(element)` : element에 해당하는 iterator를 반환
- `count(element)` : element에 해당하는 개수를 반환

기타

- `empty()` : 비어있으면 true 아니면 false를 반환
- `size()` : 세트에 포함되어 있는 원소들의 수를 반환

세트 예제 풀이

문제6H. 중복 제거하기

N개의 정수가 차례로 입력으로 주어진다.

각 입력에 대하여 아래 둘 중 하나의 결과를 출력하시오.

- 이미 앞서 등장한 적 있는 숫자의 경우 한 줄에 `DUPLICATED`를 출력한다.
- 처음 등장한 숫자의 경우 한 줄에 `OK`를 출력한다.

예를 들어서 차례대로 $\{5, 3, 5, 3, 2\}$가 입력으로 주어진 경우 출력 결과는 $\{OK, OK, DUPLICATED, DUPLICATED, OK\}$가 된다.

입력 형식

첫 줄에는 차례로 입력으로 주어질 정수들의 총 개수를 나타내는 20만이하의 자연수 N이 주어진다.

이후 총 N줄에 걸쳐서 한 줄에 하나 씩 입력이 주어진다.

- 고려해야 할 순서대로 입력이 주어진다.
- 모든 숫자는 32비트 정수형임이 보장된다.

출력 형식

각 N개의 숫자에 대한 처리 결과를 한 줄마다 출력한다.

- 이미 앞서 등장한 적 있는 숫자의 경우 한 줄에 `DUPLICATED`를 출력한다.
- 처음 등장한 숫자의 경우 한 줄에 `OK`를 출력한다.

문제6H. 중복 제거하기

입/출력 예시

␣ : 공백 ↲ : 줄바꿈 ↹ : 탭

예시 1

입력

```
5↲
5↲
3↲
5↲
3↲
2↲
```

출력

```
OK↲
OK↲
DUPLICATED↲
DUPLICATED↲
OK↲
```

문제6H. 중복 제거하기

```cpp
#include <iostream>
#include <set>

using namespace std;

int main() {
  int N;
  cin >> N;

  // integers := 이전까지 등장한 모든 정수를 저장한 집합
  set<int> integers;

  for (int i = 0; i < N; ++i) {
    int X;
    cin >> X;
  }
}
```

문제6I. 정사각형

N개의 점에 대한 좌표가 주어진다. 이 좌표들 중 4개를 골라 만들 수 있는 정사각형들 중 가장 큰 넓이는 몇 인지 계산하는 프로그램을 작성하시오.

입력 형식

첫 줄에는 테스트케이스의 수를 나타내는 10이하의 자연수 T가 주어진다. 이후 T개의 테스트케이스에 대한 입력이 차례로 주어진다.

각 테스트케이스의 첫 줄에는 좌표의 수를 나타내는 500이하의 자연수 N이 주어진다.

이후 총 N줄에 걸쳐서 한 줄에 하나씩 이 테스트케이스에서 고려해야 할 점들의 좌표가 x y형식으로 주어진다.

- X와 Y는 모두 절대값이 1억 이하인 정수다.

출력 형식

각 테스트케이스에 대하여 한 줄에 정답을 출력한다.

- N개의 점 중 네 개로 만들 수 있는 정사각형의 넓이 중 최대값을 출력한다.

문제6I. 정사각형

입/출력 예시

␣ : 공백 ↵ : 줄바꿈 ⇆ : 탭

예시 1

입력

```
2↵
5↵
0␣2↵
2␣0↵
2␣2↵
0␣0↵
1␣1↵
10↵
0␣0↵
10␣10↵
89␣76↵
97␣79↵
86␣84↵
94␣87↵
53␣14↵
54␣27↵
40␣15↵
41␣28↵
```

출력

```
4.00↵
170.00↵
```

문제6I. 정사각형

```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
#include <set>

using namespace std;


class Point2D  {
public:
  int x;
  int y;
  int index;

  Point2D(int index, int x, int y) {
    this->index = index;
    this->x = x;
    this->y = y;
  }

  Point2D(int x, int y) {
    this->index = 1;
    this->x = x;
    this->y = y;
  }

  long long getSquaredDistanceTo(Point2D target) {
    // 두 좌표간의 제곱거리를 계산
    long long dx = abs(this->x - target.x);
    long long dy = abs(this->y - target.y);
    return dx * dx + dy * dy;
  }

  double getDistanceTo(Point2D target) {
    // 두 좌표간의 실수 거리를 계산
    long long sqd = this->getSquaredDistanceTo(target);
    return sqrt(sqd);
  }

  bool operator < (const Point2D& other) const {
    // 각 좌표의 우선순위를 비교하기 위한 비교 연산자

    // x좌표가 다르다면 x좌표를 기준으로 비교한다.
    if (this->x != other.x) {
      return this->x < other.x;
    }

    // x좌표가 같다면 y좌표를 기준으로 비교한다.
    return this->y < other.y;
  }
};
```

문제6I. 정사각형

```
long long getMaximumSquareArea(int n, const vector<Point2D>& points) {
  long long answer = 0;

  // 모든 점을 Set에 저장한다
  set<Point2D> pSet;
  for (int i = 0; i < n; i += 1) {
    pSet.insert(points[i]);
  }

  for (int i = 0; i < n; i += 1) {
    Point2D pa = points[i];
    for (int j = 0; j < n; j += 1) {
      Point2D pb = points[j];
      // 두 기준점 pa와 pb를 지정한다.
      // 선분 pa-pb가 정사각형의 한 변이라고 하자.


    }
  }


  return answer;
}
```

문제6I. 정사각형

```
void process(int caseIndex) {
  int n;
  cin >> n;

  vector<Point2D> points;

  for (int i = 0; i < n; i += 1) {
    int x, y;
    cin >> x >> y;
    points.push_back(Point2D(i, x, y));
  }

  long long answer = getMaximumSquareArea(n, points);

  cout << fixed << setprecision(2) << answer << endl;
}

int main() {
  int caseSize;
  cin >> caseSize;

  for (int caseIndex = 1; caseIndex <= caseSize; caseIndex += 1) {
    process(caseIndex);
  }
}
```