

3장 12절

---

최적화 문제  
결정 문제로 바꿔 풀기

## 최적화 문제와 결정 문제

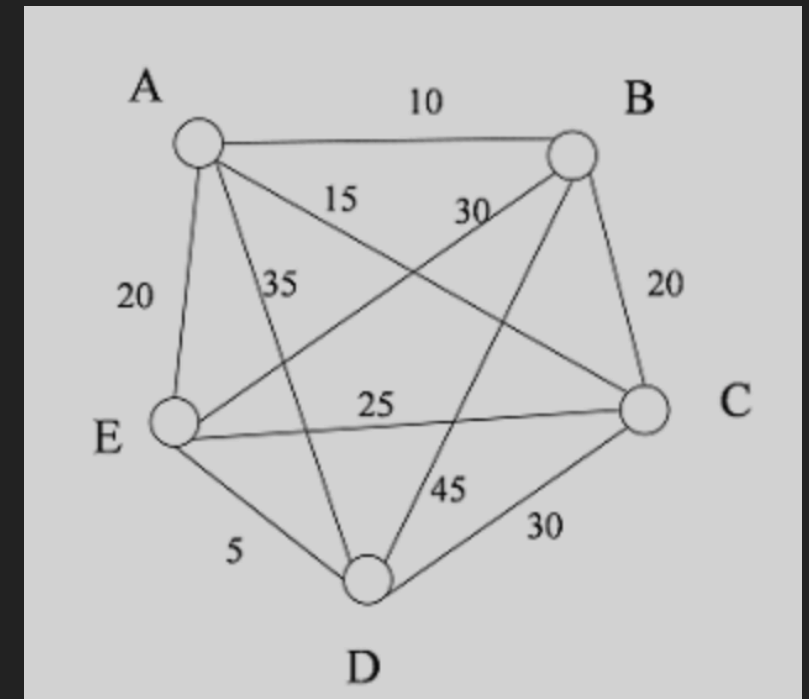
- ▶ 최적화 문제 (optimization problem)- 최소화, 최대화 문제
- ▶ 결정 문제 (decision problem) - yes or no 문제

## 최적화 문제와 결정 문제

- ▶ 여행하는 외판원 문제의 두 가지 ver.

optimize (G): 그래프 G의 모든 정점을 찍고, 시작  
점으로 돌아오는 최단 경로의 길이를 반환해라

decision (G, x): 그래프 G의 모든 정점을 찍고, 시  
작점으로 돌아오면서, 그 길이가 x 이하인  
경로의 존재 여부를 반환



- ▶ optimize() 는 가장 짧은 경로의 길이를 실수로 반환,  
decision()은 최단 경로건 아니건 간에 x보다 짧은 경로가 있는지만 확인

## 최적화 문제와 결정 문제

### ▶ 결정문제가 최적화 문제보다 어려울 수는 없다!

반대의 경우나, 둘이 비슷하게 어려운 경우는 있어도..

```
double optimize(const Graph &g);  
  
bool decision(const Graph &g, double x){  
    return optimize(g) <= x;  
}
```

- decision()이 optimize()보다 시간 복잡도가 클 일은 결코 없다는 뜻

## 최적화 문제와 결정 문제

- ▶ 이번 시간에는 최적화 문제를 결정 문제로 바꿔 풀면 (비교적) 쉽게 풀 수 있는 문제들의 패턴을 연습해 봅시다!

1. DARPA Grand Challenge
2. 남극 기지
3. 캐나다 여행
4. 수강 취소

## 1. DARPA GRAND CHALLENGE (난이도 중)

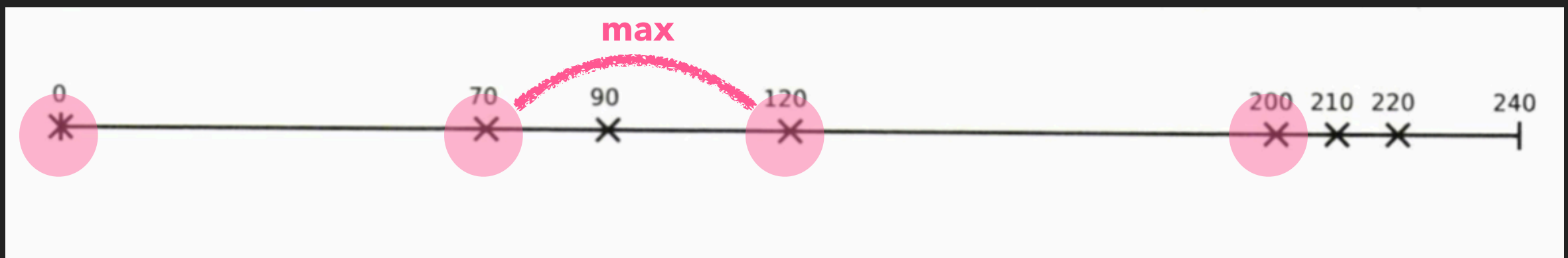
- ▶ DARPA Grand Challenge는 무인 자동차 경주 대회  
사막을 가로지르는 240km의 도로를 완주하는 것이 mission!



## 1. DARPA GRAND CHALLENGE (난이도 중)

- ▶ 우리는 이 경기를  $n$ 개의 카메라로 중계하려고 하는데, 사막을 가로지르는 도로이다 보니 경로 상에 카메라를 설치할 수 있는 곳이  $m$ 군데로 제한이 된다.

문제: 이 중  $n$  군데에 카메라를 설치해서 가장 가까운 두 카메라 사이의 간격을 최대화 해라!



ex) 설치할 수 있는 7 군데 중 카메라 4 대를 설치할 경우

## 1. DARPA GRAND CHALLENGE (난이도 중)

- ▶ 문제: 이 중  $n$  군데에 카메라를 설치해서  
가장 가까운 두 카메라 사이의 간격을 최대화 해라!

$optimize(locations, camera)$  = 카메라 간 최소 간격의 최대치

$decision(locations, cameras, gap)$  = 모든 카메라의 간격이  
**gap 이상** 이 되도록 하는 방법이 있는가?

Q) 왜 gap 이상인 것을 물어볼까?

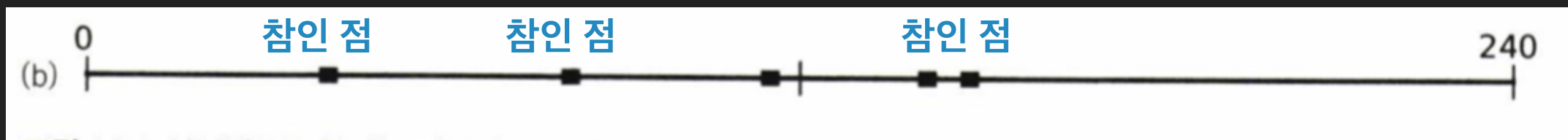


## 1. DARPA GRAND CHALLENGE (난이도 중)

- ▶ 카메라 간의 최소 간격이  $gap$  이상인 방법이 있는가?



- ▶ 카메라 간의 최소 간격이  $gap$ 인 방법이 있는가?



이분법으로 문제를 풀기 위한 레시피

## 1. DARPA GRAND CHALLENGE (난이도 중)

- ▶ 카메라를 설치할 수 있을 때마다 설치하는 탐욕적 알고리즘

Q) 0km (설치할 수 있는 최초)의 위치에 카메라를 설치해야 할까?

A) 귀류법

0km 위치에 카메라를 설치하지 않는다는 답이 존재한다 가정.  
그런데 그 답에서 가장 왼쪽 카메라를 0km의 지점으로 옮기면  
똑같은 답이 나오니, 항상 선택해도 된다.

- ▶ 첫 번째로 만나는 위치에 무조건 카메라를 설치하고, 각 위치를 순회하면서 카메라를 설치할 수 있을 때마다 카메라를 설치하자!

# 1. DARPA GRAND CHALLENGE (난이도 중)

결정문제: 정렬되어 있는 locations 중, cameras를 선택해 모든 카메라 간의 간격이 gap 이상이 되는 방법이 있냐?

```
private boolean decision(double [] location, int cameras, double gap){
    double limit=-1;

    int installed =0;

    for (int i=0; i<location.length; i++){
        if (limit<=location[i]) {

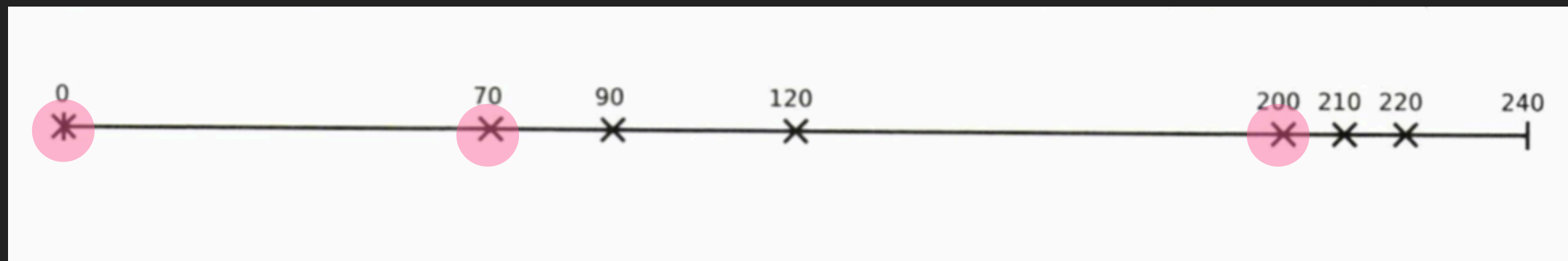
            ++installed;
            limit= location[i]+gap;
        }
    }

    //일단 설치할 수 있는 카메라는 다 설치해놓고 그게 설치해야 되는 카메라 수보다 크거나 같으면 okay!
    return installed>=cameras;
}
```

시간 복잡도는?

# 1. DARPA GRAND CHALLENGE (난이도 중)

decision( [0, 70, 90, 120, 200, 210, 220, 240] , cameras= 4, gap= 60) 이라 가정할때,



0에서 일단 카메라 설치하고 나서,

$\text{limit} = \text{location}[0] + \text{gap} = 0 + 60$

다음에 놓을 수 있는 카메라는 60보다 커야한다.

70? 60 보다 크니까 설치 가능.

새로운  $\text{limit} = \text{location}[1] + 60 = 130$

```
for (int i=0; i<location.length; i++){  
    if (limit<=location[i]) {  
        ++installed;  
        limit= location[i]+gap;  
    }  
}
```

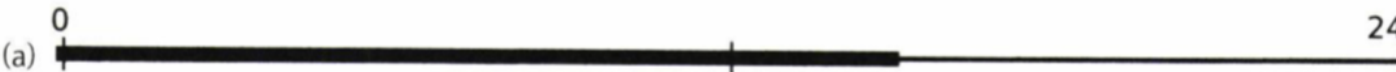
다음에 놓을 수 있는 카메라 위치는 130 보다 커야한다... 이런식으로!

# 1. DARPA GRAND CHALLENGE (난이도 중)

최적화 문제: 정렬되어 있는 locations 중, cameras를 선택해 최소 간격을 최대화 한다!

```
private double optimize(double [] location, int cameras){  
    double lo= 0;  
    double hi= 241;  
    for (int i=0; i<100; i++){  
        double mid= (lo+hi)/2.0;  
        if (decision(location,cameras,mid)) lo=mid;  
        else hi=mid;  
    }  
    return lo;  
}
```

decision(lo) && !decision(hi)



# 1. DARPA GRAND CHALLENGE (난이도 중)

## ▶ 왜 100번의 for loop을 도는가?

있는 것을 확인할 수 있지요. 반복문을 100번 수행하면 우리가 반환하는 답의 절대 오차는 최대  $\frac{|lo - hi|}{2^{101}}$ 이 됩니다.  $2^{101}$ 은 대략 서른한 자리의 수로,  $|hi - lo|$ 가 대략  $10^{20}$  미만의 수라면 이 오차는 항상  $10^{-7}$ 보다 작지요. 따라서 큰 숫자를 다루는 경우에도 충분히 답을 구할 수 있습니다. 또한 이와 같은 방법은 절대로 무

To be continued in the next chapter!

## 1. DARPA GRAND CHALLENGE (난이도 중)

- ▶ 왜 100번의 for loop을 도는가?
  - 그냥 어차피 값의 가지수는 유한하니까 간격의 종류  $n(n-1)/2$  로 목록을 만들어서 이 안에서 이분 검색을 통해 속도를 높이면 안되는 건가?
- ▶ 고정된 수의 후보만을 탐색하는 이분법의 경우에는 오차가 생겨서 오류가 생기는 경우 아예 그 작은 후보로 반환된 값으로 바뀌기 때문에 위험하다는 것!

## 최적화 문제 결정 문제로 바꾸기 레시피!

- ▶ (Step1) "가장 좋은 답은 무엇인가?" 라는 최적화 문제를  
"x 혹은 그보다 좋은 답이 있는가?" 라는 결정 문제 형태로 바꾸기!
- ▶ (Step2) 결정 문제를 쉽게 풀 수 있는 방법이 있는지 찾아보기
- ▶ (Step3) 결정 문제를 내부적으로 이용하는 이분법 알고리즘을 작성하기!



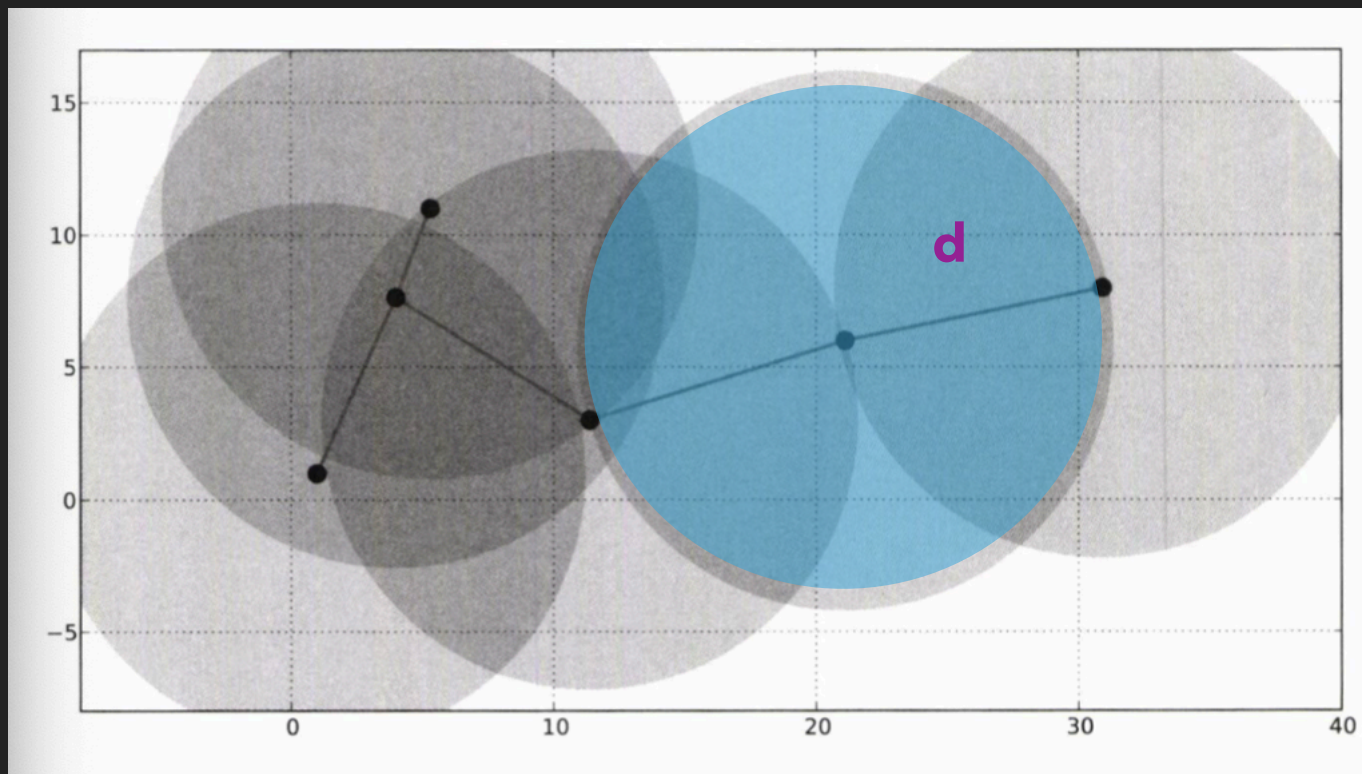
최적화 문제 결정 문제로 바꿔 풀기

---

같이 해봅시다!

## 2. 남극 기지 문제 (난이도 하)

- ▶ 남극에는  $n$ 개의 탐사 기지가 있습니다. 탐사 기지들 간에 서로 통신하기 위해  $n$ 개의 무전기를 구입해서 각 탐사 기지에 배치를 하여 기지 간 연락망을 구축하려고 합니다. 모든 무전기 통신 반경은  $d$  이며, 두 탐사 기지는 사이의 거리가  $d$  이하여만 서로 연락할 수 있습니다.



Input : 기지의 개수  
각 기지들의 좌표 (x,y)

각 기지 간에는 다른 기지를 통해  
간접적으로 연락할 수 있지만  
어쨌든 직접적으로나 간접적으로나  
모든 기지가 연락이 닿아야 한다는 거

## 2. 남극 기지 문제 (난이도 하)

- ▶ 남극에는  $n$ 개의 탐사 기지가 있습니다. 탐사 기지들 간에 서로 통신하기 위해  $n$ 개의 무전기를 구입해서 각 탐사 기지에 배치를 하여 기지 간 연락망을 구축하려고 합니다. 모든 무전기 통신 반경은  $d$  이며, 두 탐사 기지는 사이의 거리가  $d$  이하여만 서로 연락할 수 있습니다.
- ▶ 무전기의 통신반경이 커지면 더 비싸다. 그래서  $d$ 를 최소화 하는 것이 목표!

optimize (P)= P에 주어진 기지들을 모두 연결하는 연락망을  
구축할 때, 가능한 최소 무전기 반경은 얼마인가?

## 2. 남극 기지 문제 (난이도 하)

- ▶ (Step1) "가장 좋은 답은 무엇인가?" 라는 최적화 문제를  
"x 혹은 그보다 좋은 답이 있는가?" 라는 결정 문제 형태로 바꾸기!

optimize (P)= P에 주어진 기지들을 모두 연결하는 연락망을  
구축할 때, 가능한 최소 무전기 반경은 얼마인가?

decision(P, d) = ?

## 2. 남극 기지 문제 (난이도 하)

- ▶ (Step1) "가장 좋은 답은 무엇인가?" 라는 최적화 문제를  
"x 혹은 그보다 좋은 답이 있는가?" 라는 결정 문제 형태로 바꾸기!

$\text{decision}(P, d)$  = 모든 기지를 하나로 연결하되,  
가장 먼 두 기지 간의 거리가  $d$ 이하인 연락망이 있나?

## 2. 남극 기지 문제 (난이도 하)

- ▶ (Step2) 결정 문제를 쉽게 풀 수 있는 방법이 있는지 찾아보기

$\text{decision}(P, d)$  = 모든 기지를 하나로 연결하되,  
가장 먼 두 기지 간의 거리가  $d$ 이하인 연락망이 있나?

- ▶ 서로 거리가  $d$  이하인 기지들을 우선 전부 연결해 연락망을 만든 뒤,  
이들이 하나로 연결되어 있는지 확인한다!

## 2. 남극 기지 문제 (난이도 하)

- ▶ (Step2) 결정 문제를 쉽게 풀 수 있는 방법이 있는지 찾아보기

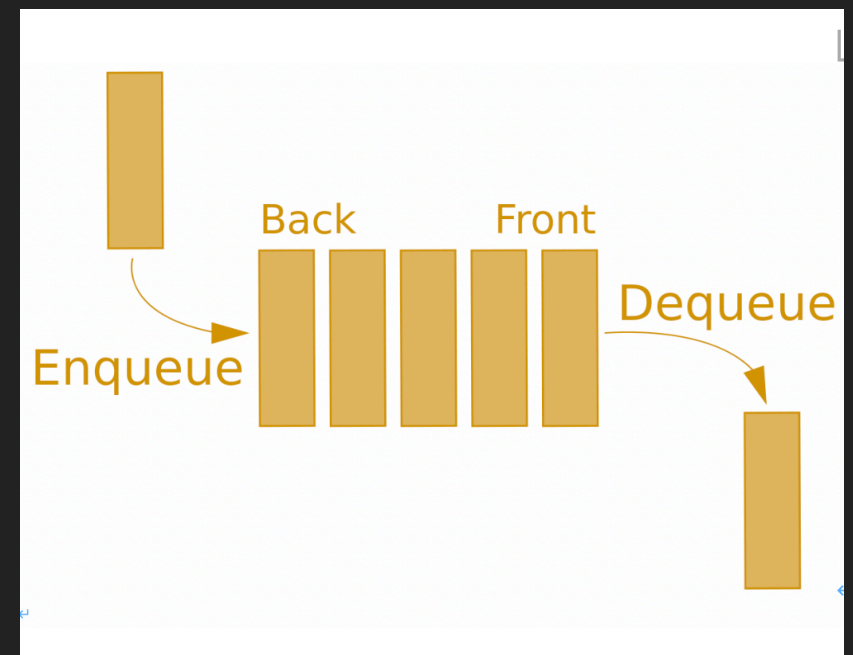
```
int n;  
double dist[][]= new double[101][101];  
  
private boolean decision(double d){  
    boolean [] visited= new boolean[n];  
    for (int i=0; i<n; i++){  
        visited[i]=false;  
    }
```

기지 n을 방문했는지 안했는지  
저장하는 vector 생성  
모두 방문하지 않은 것으로 초기화

## 2. 남극 기지 문제 (난이도 하)

- ▶ (Step2) 결정 문제를 쉽게 풀 수 있는 방법이 있는지 찾아보기

```
visited[0]= true;
Queue <Integer> queue= new LinkedList<>();
queue.add(0);
int seen=0;
while(!queue.isEmpty()){
    int here= queue.peek();
    queue.poll();
    ++seen;
    for (int there=0; there<n; there++){
        if(!visited[there] && dist[here][there] <=d){
            visited[there]= true;
            queue.add(there);
        }
    }
}
return seen==n;
}
```



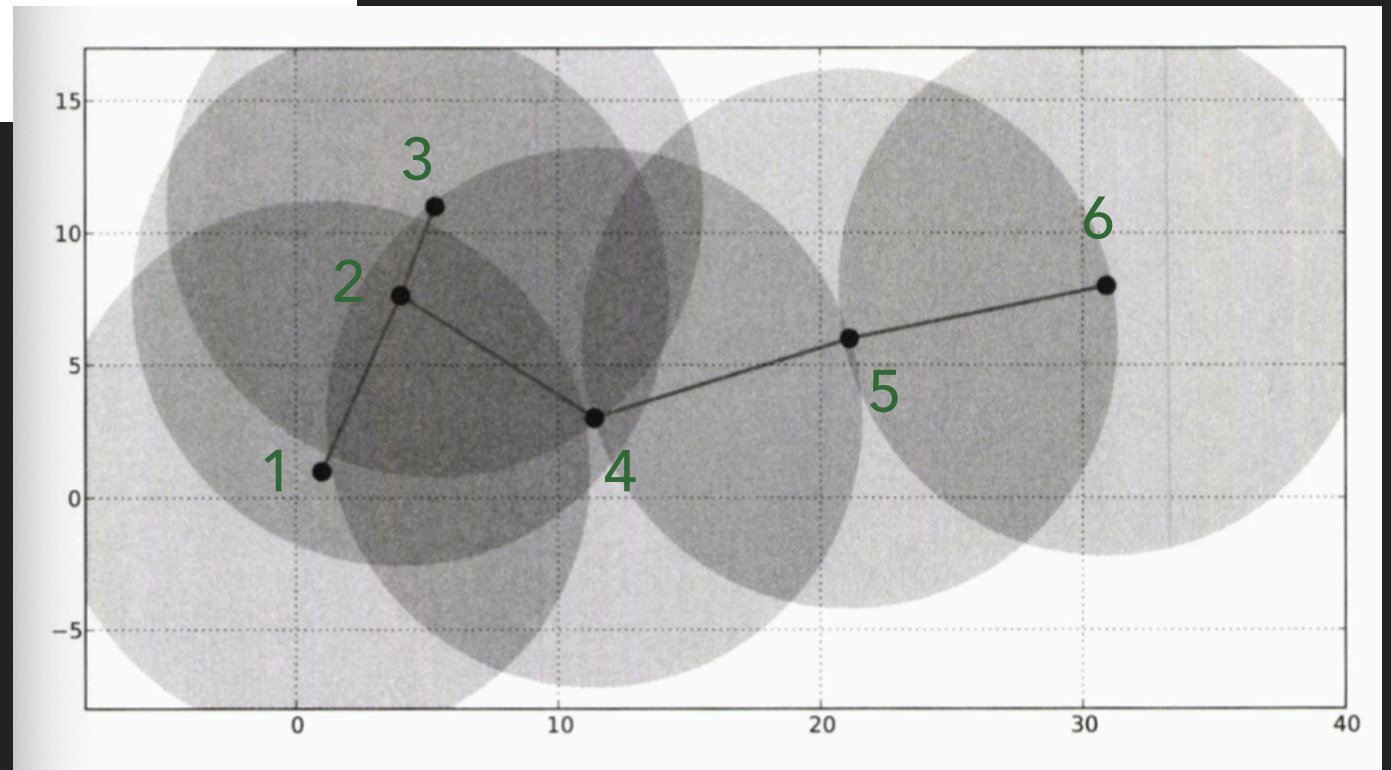


## 2. 남극 기지 문제 (난이도 하)

- ▶ (Step2) 결정 문제를 쉽게 풀 수 있는 방법이 있는지 찾아보기

```
while(!queue.isEmpty()){  
    int here= queue.peek();  
    queue.poll();  
    ++seen;  
    for (int there=0; there<n; there++){  
        if(!visited[there] && dist[here][there] <=d){  
            visited[there]= true;  
            queue.add(there);  
        }  
    }  
}
```

시간 복잡도는?



## 2. 남극 기지 문제 (난이도 하)

- ▶ (Step3) 결정 문제를 내부적으로 이용하는 이분법 알고리즘을 작성

```
private double optimize(){
    double lo=0;
    double hi= 1416.00; 가능한 최대 거리 (1000,0) (0,1000)= 1000* 루트2 = 1415
    for (int it=0; it < 100; it++){
        double mid= (lo+hi)/2.0;
        if (decision(mid)) hi=mid;
        else lo=mid;
    }
    return hi; !decision(lo) && decision(hi)
}
```

### 3. 수강취소 (난이도 상)

- ▶ 이번 학기에 욕심을 부려 학점 초과신청을 한 재은이는 중간고사 성적을 보고 한숨을 토할 수 밖에 없었습니다. 다음 학기 장학금을 받을 만큼 성적이 잘 나오지 않았기 때문입니다. 이제 재은이에게 남은 희망은 다음 주의 수강철회 기간 뿐입니다.

### 3. 수강취소 (난이도 상)

- ▶ 재은이네 학교에서는 장학금을 학생의 중간고사 등수와 기말고사 등수에 따라 배정합니다. 어떤 학생이 듣는  $i$  번째 과목의 수강생 수가  $C_i$  라고 합시다. 이 학생의  $i$  번째 과목 중간고사 등수가  $r_i$  라고 하면, 이 학생의 중간고사 등수는 다음과 같습니다.

150명, 200명, 15명 듣는 강의에서 각각 100등, 10등, 5등을 하면:

$$cumulativeRank = \frac{\sum r_i}{\sum C_i}$$

$$\frac{100 + 10 + 5}{150 + 200 + 15} = \frac{115}{365} \approx 0.315$$

### 3. 수강취소 (난이도 상)

- ▶ 수강철회를 하면 철회한 과목은 중간 고사의 누적 등수 계산에 들어가지 않습니다. 재은이네 학교에서는 수강 철회를 해도 남은 과목이 k개 이상이면 장학금을 받을 수 있습니다. 재은이가 적절히 드랍을 했을 때 얻을 수 있는 **최소 누적 등수**를 계산합니다.
- ▶ (Step1) "가장 좋은 답은 무엇인가?" 라는 최적화 문제를 "x 혹은 그보다 좋은 답이 있는가?" 라는 결정 문제 형태로 바꾸기!

decision (x)= 적절히 과목들을 드랍해서 누적 등수가  
x 이하가 될 수 있나?

과목들의 집합  $[0, 1, \dots, n-1]$ 의 모든 부분 집합 중  
크기가 k이상이며 다음 조건을 만족하는 S를 고르기!

$$\frac{\sum_{j \in S} r_j}{\sum_{j \in S} c_j} \leq x$$

### 3. 수강취소 (난이도 상)

- ▶ (Step2) 결정 문제를 쉽게 풀 수 있는 방법이 있는지 찾아보기
- ▶  $x \cdot c(j) - r(j) = v(j)$  라고 두었을 때,  $\text{decision}(x)$ 는 실수의 배열  $v(j)$ 가 주어질 때, 이 중  $k$ 개 이상을 선택해서 그 합을 0이상으로 만들 수 있는가 하는 비교적 쉬운 문제로 바뀐다!

$$0 \leq x \sum_{j \in S} c_j - \sum_{j \in S} r_j = \sum_{j \in S} (x \cdot c_j - r_j)$$

그러면,  $v(j)$ 를 정렬해서 가장 큰  $k$ 개의 원소를 더해서 풀 수 있다! (탐욕법)

### 3. 수강취소 (난이도 상)

시간 복잡도는?

```
int n; //재은이가 수강하는 과목의 수=n
int k; //서울대학교 최소 수강 과목 개수
int []c= new int[1000]; //수업 당 전체 학생 수 저장
int []r= new int[1000]; //수업 당 재은이의 점수 저장

//누적 등수가 average가 되도록 할 수 있나?
private boolean decision(double average){
    double [] v= new double[n];
    for (int i=0; i<n; i++){
        v[i]= average*c[i]-r[i];
    }
    Arrays.sort(v);

    //v 중에서 k개의 합이 0 이상이 될 수 있는가? - 탐욕법
    double sum=0;
    for (int i=n-k; i<n; i++){
        sum += v[i];
    }

    return sum>=0;
}
```

### 3. 수렴최소 (난이도 상)

- ▶ (Step3) 결정 문제를 내부적으로 이용하는 이분법 알고리즘을 작성

```
private double optimize(){
    double lo= -1e-9;    10-9
    double hi=1;
    for (int it=0; it < 100; it++){
        double mid= (lo+hi)/2.0;
        if (decision(mid)) hi=mid;
        else lo=mid;
    }
    return hi;    !decision(lo) && decision(hi)
}
```