2021 여름학기 동국대학교 SW역량강화캠프

3일차. 완전탐색 2

● 완전탐색

▶ 가능한 모든 경우의 수를 고려하는 가장 기본적인 탐색 알고리즘

▶ 가장 무식하지만 가장 확실한 방법. 답을 찾을 확률 100%.

▶ 주로 반복문, DFS(깊이 우선 탐색), BFS(너비 우선 탐색)을 사용

● 완전 탐색 문제 유형 – 기준 잡아 비교하기

▶ 주어진 값들 중 하나를 잡아, 나머지와 비교하여 계산하는 형태의 문제

● 덩치 (767)

문제

우리는 사람의 덩치를 키와 몸무게, 이 두 개의 값으로 표현하여 그 등수를 매겨보려고 한다. 어떤 사람의 몸무게가 x kg이고 키가 y cm라면 이 사람의 덩치는 (x,y)로 표시된다. 두 사람 A 와 B의 덩치가 각각 (x,y), (p,q)라고 할 때 x>p 그리고 y>q 이라면 우리는 A의 덩치가 B의 덩치보다 "더 크다"고 말한다. 예를 들어 어떤 A, B 두 사람의 덩치가 각각 (56,177), (45,165) 라고 한다면 A의 덩치가 B보다 큰 셈이 된다. 그런데 서로 다른 덩치끼리 크기를 정할 수 없는 경우도 있다. 예를 들어 두 사람 C와 D의 덩치가 각각 (45, 181), (55,173)이라면 몸무게는 D가 C보다 더 무겁고, 키는 C가 더 크므로, "덩치"로만 볼 때 C와 D는 누구도 상대방보다 더 크다고 말할 수 없다.

N명의 집단에서 각 사람의 덩치 등수는 자신보다 더 "큰 덩치"의 사람의 수로 정해진다. 만일 자신보다 더 큰 덩치의 사람이 k명이라면 그 사람의 덩치 등수는 k+1이 된다. 이렇게 등수를 결정하면 같은 덩치 등수를 가진 사람은 여러 명도 가능하다. 아래는 5명으로 이루어진 집단에서 각 사람의 덩치와 그 등수가 표시된 표이다

이름	<몸무게, 키>	덩치 등수
A	<55, 185>	2
B	<58, 183>	2
C	<88, 186>	1
D	<60, 175>	2
E	<46, 155>	5

위 표에서 C보다 더 큰 덩치의 사람이 없으므로 C는 1등이 된다. 그리고 A, B, D 각각의 덩치보다 큰 사람은 C뿐이므로 이들은 모두 2등이 된다. 그리고 E보다 큰 덩치는 A, B, C, D 이렇게 4명이므로 E의 덩치는 5등이 된다. 위 경우에 3등과 4등은 존재하지 않는다. 여러분은 학생 N명의 몸무게와 키가 담긴 입력을 읽어서 각 사람의 덩치 등수를 계산하여 출력해야 한다.

● 완전탐색을 통한 해결

▶ 모든 사람에 대해서, 자신보다 덩치가 큰 사람이 몇 명인지 계산하면 된다.

핵심 코드

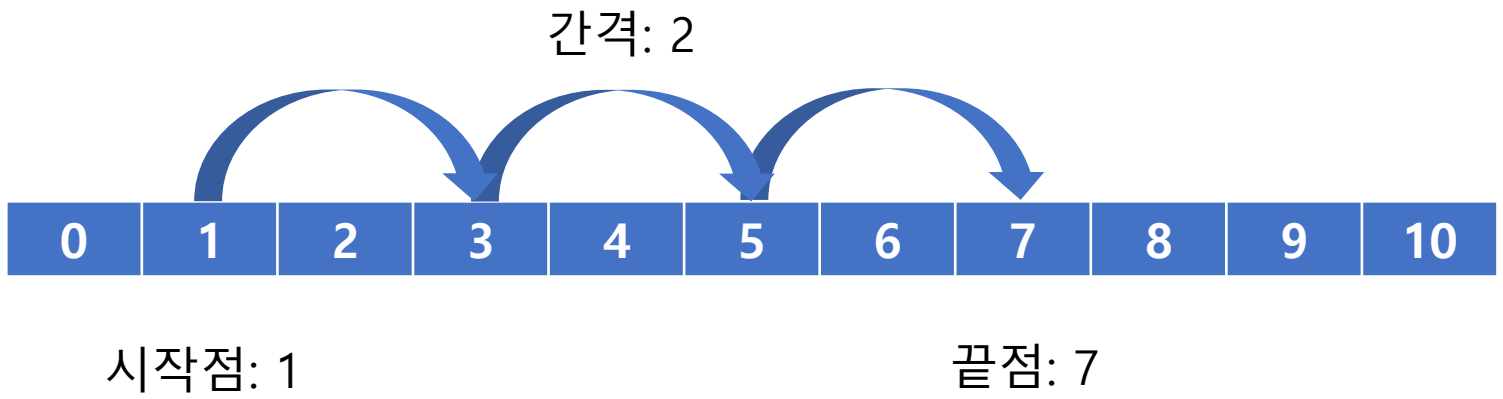
```
for (int i = 0; i < N; i++) {
    int bigger = 0; // i보다 덩치가 큰 사람을 세어준다.
    for (int j = 0; j < N; j++) {
        if (i == j)
            continue; // 자기 자신은 제외
        if (arr[i][0] < arr[j][0] && arr[i][1] < arr[j][1]) { // j가 i보다 키와 몸무게 둘 다 크다면
            bigger++;
        }
    }
    sb.append((bigger + 1) + " "); // 자신보다 덩치가 큰 사람의 수 + 1 을 출력
}
```

● 완전 탐색 문제 유형 – 구간 탐색

▶ 배열에서 특정 조건을 만족하는 구간을 찾는 형태의 문제

▶ 3가지 변수를 조절하면서 반복문을 돌리며, 구간을 찾는다.

▶ 구간의 시작점, 구간의 끝점, 고르는 수들 사이의 간격



● 구간의 합들(2123)

문제

자연수가 N개 주어질 때, 아래 조건을 만족하는 구간의 수를 출력하자.

A[i] + A[i + 1] + ... + A[j - 1] + A[j] = M

첫 번째 예제에서는 [1, 2], [2, 3], [3, 4] 구간이 조건을 만족하므로 답이 3이다.

입력

첫 줄에는 수의 개수를 나타내는 정수 N과 구간의 합을 의미하는 정수 M이 주어진다. $1 \leq N \leq 500, 1 \leq M \leq 10^8$
두 번째 줄에는 배열을 구성하는 $3 * 10^4$ 이하의 자연수가 N개 주어진다.

출력

문제의 조건을 만족하는 구간의 수를 출력하자.

●반복문을 이용한 완전탐색 풀이

▶ 구간의 시작점과 끝점을 반복문으로 완전 탐색

▶ 시작점 ≤ 끝점

▶ 반복문으로 시작점과 끝점 사이 구간의 합을 구한다.

▶ 시작점 고르는 반복문, 끝점 고르는 반복문, 구간의 합 구하는 반복문 → $O(N^3)$

핵심 코드

```
int ans = 0;
for (int i = 0; i < N; i++) { // 구간의 시작점 i
    for (int j = i; j < N; j++) { // 구간의 끝점 j
        int sum = 0;
        for (int k = i; k <= j; k++) { // sum에 구간의 원소들의 합을 저장
            sum += arr[k];
        }
        if (sum == M) {
            ans++;
        }
    }
}
```

● 표지 (4103)

문제

윌리는 새로운 표지를 제작하려고 한다.
새 표지를 만드는 일은 굉장히 비용이 많이들어 기존에 있는 표지를 재활용하려고 한다.
표지들은 알파벳 소문자로 구성되어 있다.

기존 표지로 새 표지를 만들기 위해서는, 기존 표지에서 같은 간격으로 있는 문자를 선택해서 새로운 표지 단어와 정확히 일치해야한다.
예를 들어, 만들고자 하는 새로운 표지가 abc 이고, 기존 표지가 aabc 라면, 2, 3, 4번째 문자를 선택하여 새로운 표지를 만들 수 있다.
허지만, 기존 표지가 abdc 라면, 1, 2, 4번째 문자가 같은 간격으로 존재하지 않으므로 새로운 표지로 만들 수 없다.

N개의 기존 표지와 새로운 표지가 주어졌을 때, 새로운 표지로 만들 수 있는 기존 표지의 개수를 구하자.

입력

첫째 줄에 기존 표지의 수 N이 주어진다. ($1 \leq N \leq 100$) 둘째 줄에는 윌리의 새 표지 이름이 주어진다. 이름은 알파벳 소문자로만 이루어져 있고, 길이는 3자 이상, 25자 이하이다. 다음 N개 줄에는 기존 표지에 쓰여 있는 문자가 주어진다. 이 표지에 쓰여 있는 문자는 알파벳 소문자이고, 길이는 1자 이상 100자 이하이다.

출력

첫째 줄에 상근이가 만들 수 있는 표지의 수를 출력한다.

●반복문을 이용한 완전탐색 풀이

▶ 문자열 str에서 조건대로 문자를 골라 board 문자열을 완성할 수 있는지 탐색

▶ str에서 시작점과 고르는 문자들 사이의 간격을 결정해서 문자들을 고른다.

▶ str에서 시작점 j, 간격 d로 len개의 문자를 고르면

str[j], str[j+d], str[j+2*d], ……, str[j+(len-1)*d]

핵심 코드

```
int ans = 0;
for(int i=0; i<N; i++) {
    String Str = br.readLine();
    char[] str = Str.toCharArray();
    boolean pos = false;
    for(int j=0; j<str.length; j++){ // str에서 j를 시작점으로 하고
        for(int d=1; d<str.length; d++){ // d칸씩 건너뛰면서 길이 board.length 의 문자열을 골라보자.
        //str[j], str[j+d], str[j+2*d], ....., str[j+(board.length - 1)*d] 의 board.length개의 문자를 검사하면 됩니다.
            if(j + (board.length - 1) * d >= str.length) continue;
            boolean same = true;
            for(int k=0; k<board.length; k++){
                if(board[k] != str[j + k * d]) same = false;
            }
            if(same == true){
                pos = true;
            }
        }
    }
    if(pos == true) {
        ans++;
    }
}
```

● 마라톤 (3370)

문제

농장에 있는 젖소들이 건강하지 못하다고 생각한 농부 존은 젖소들을 위한 마라톤 대회를 열었고, 농부 존의 총애를 받는 젖소 박승원 역시 이 대회에 참가할 예정이다.

마라톤 코스는 N (3 ≤ N ≤ 100000) 개의 체크포인트로 구성되어 있으며, 1번 체크포인트에서 시작해서 모든 체크 포인트를 순서대로 방문한 후 N번 체크포인트에서 끝나야지 마라톤이 끝난다. 게으른 젖소 박승원은 막상 대회에 참가하려 하니 귀찮아져서 중간에 있는 체크포인트 한개를 몰래 건너뛰려 한다. 단, 1번 체크포인트와 N번 체크포인트를 건너뛰면 너무 눈치가 보이니 두 체크포인트는 건너뛰지 않을 생각이다.

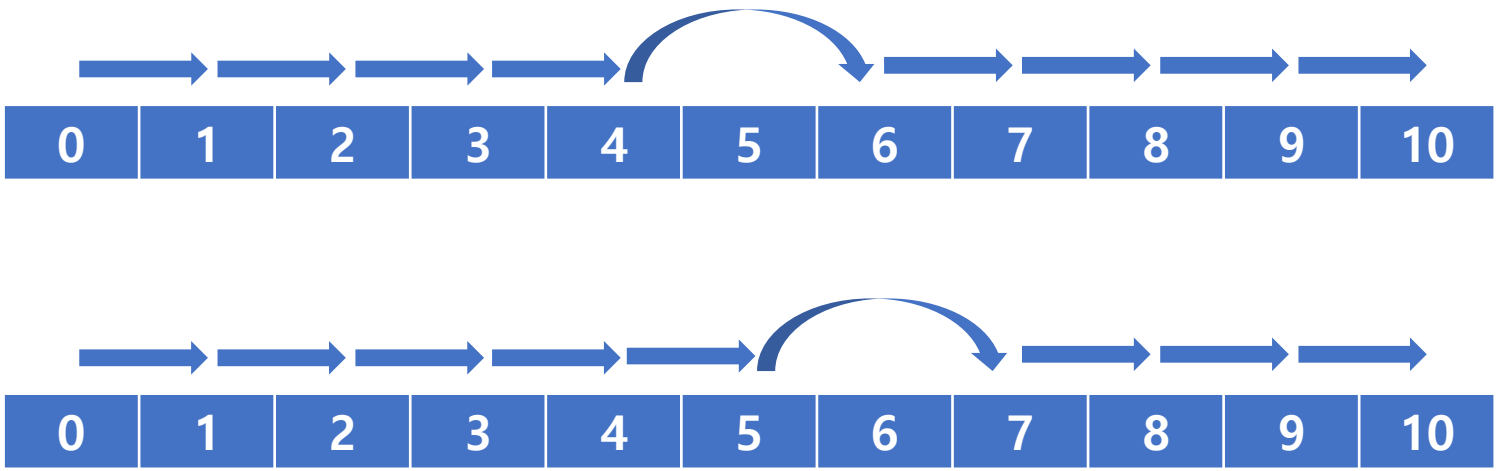
젖소 박승원이 체크포인트 한개를 건너뛰면서 달릴 수 있다면, 과연 승원이가 달려야 하는 최소 거리는 얼마일까?

참고로, 젖소 마라톤 대회는 서울시내 한복판에서 진행될 예정이기 때문에 거리는 택시 거리(Manhattan Distance)로 계산하려고 한다. 즉, (x1,y1)과 (x2,y2) 점 간의 거리는 |x1-x2| + |y1-y2| 로 표시할 수 있다. (|x|는 절댓값 기호다.)

예를 들어 예시에서 젖소 박승원은 2번째 혹은 3번째 체크포인트를 건너뛸 수 있는데, 여기서 두 번째 체크포인트를 건너뛸 경우 경로는 (0,0) -> (11,-1) -> (10,0) 이 되며 거리는 14이다. 박승원은 이것보다 더 짧게 달릴 수 없다.

●반복문을 이용한 완전탐색 풀이 $O(N^2)$

▶ 어느 체크포인트를 건너뛸 지를 완전탐색



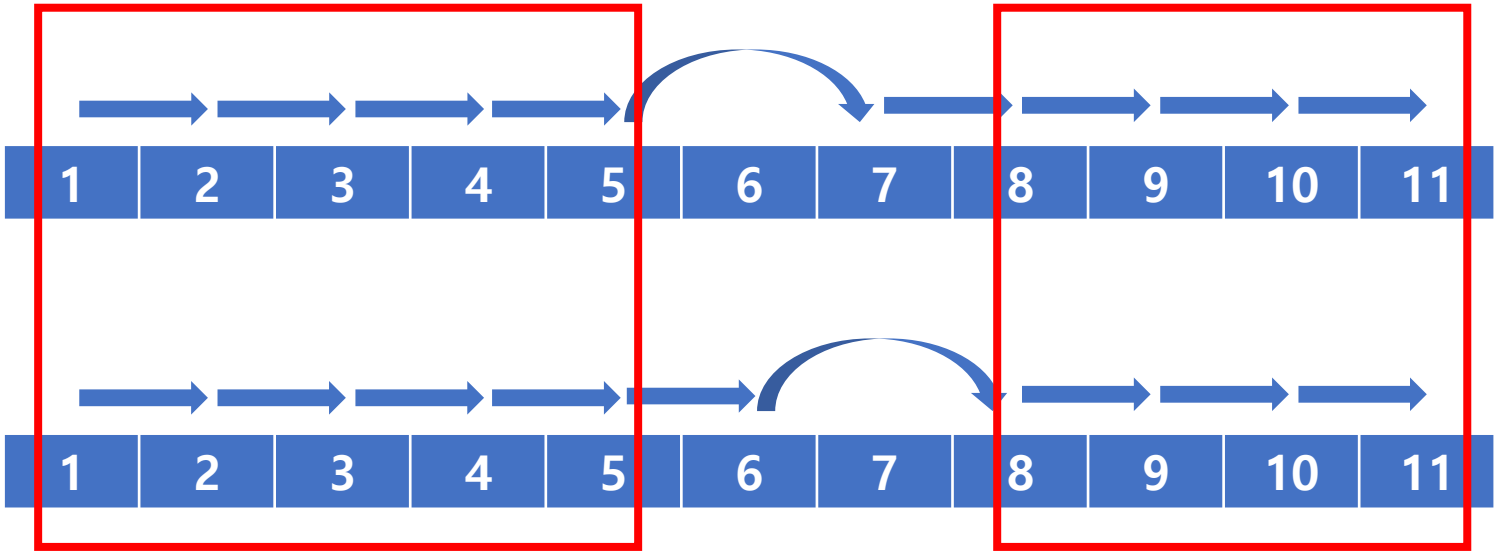
핵심 코드

```
int ans =  1987654321;
for(int i=2; i<N; i++) { // i번 체크포인트를 건너뛰어보자
    int totaldist = 0;
    for(int j=1; j<N; j++) { // j ~ (j+1) 거리를 더해준다.
        if(j == i - 1) { // 만약 j+1번을 건너뛰어야 한다면
            totaldist += dist(j, j+2); // j번과 j+2번 사이의 거리를 더해준다.
        }
        else if(j == i) { // j번을 건너뛰어야 한다면
            continue; // 어떠한 값도 더해주지 않는다.
        }
        else { // j ~ (j+1) 거리를 더해준다.
            totaldist += dist(j, j+1);
        }
    }
    if(totaldist < ans) ans = totaldist;
}
```

●반복문을 이용한 완전탐색 풀이 O(N)

▶ 어느 체크포인트를 건너뛸 지를 완전탐색

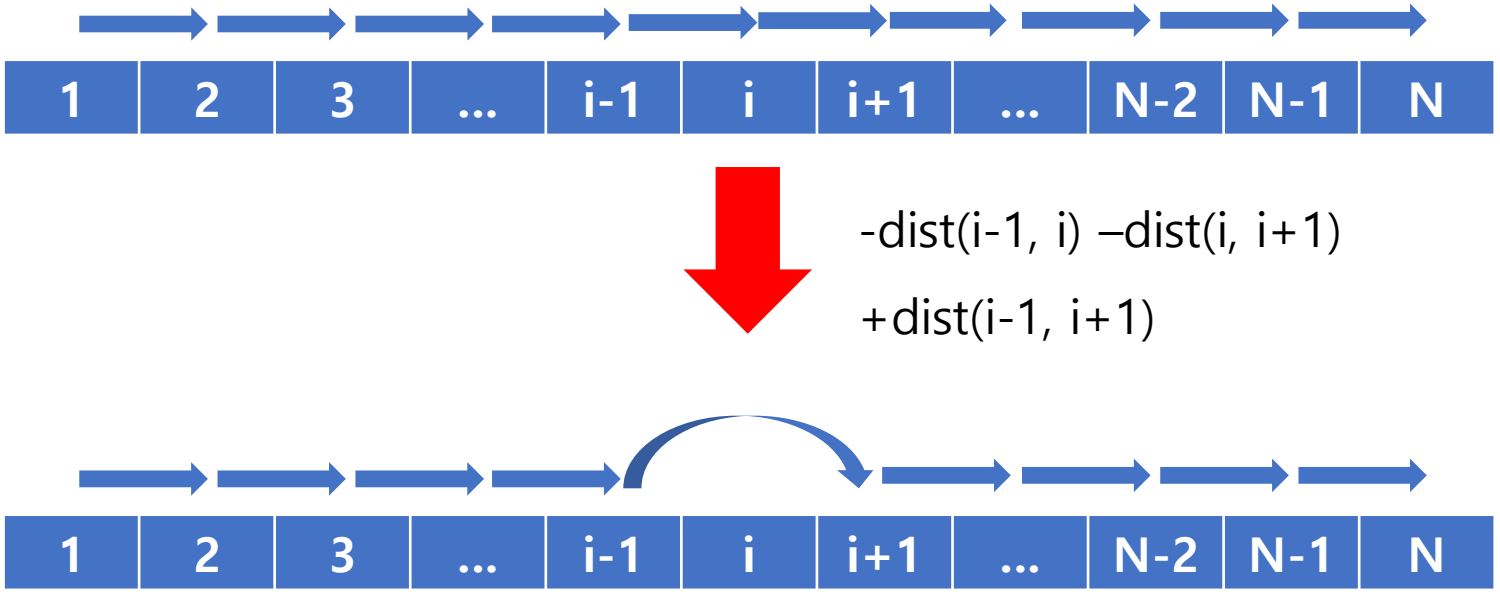
▶ 시간복잡도를 줄일 방법은?



●반복문을 이용한 완전탐색 풀이 O(N)

▶ 중복되는 계산을 줄이자.

▶ 체크포인트를 건너뛰지 않은 상황에서의 변화값만을 계산



핵심 코드

```
int basedist = 0;
for(int i=2; i<=N; i++) {
    basedist += dist(i-1, i);
}
int ans =  1987654321;
for(int i=2; i<N; i++) { // i번 체크포인트를 건너뛰어보자
    int totaldist = basedist - dist(i-1, i) - dist(i, i+1) + dist(i-1, i+1);
    if(totaldist < ans) ans = totaldist;
}
```