1102 - 발전소

난이도	골드1

문제

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	10504	2593	1777	24.279%

문제

은진이는 발전소에서 근무한다. 은진이가 회사에서 잠깐 잘 때마다, 몇몇 발전소가 고장이난다. 게다가, 지금 은진이의 보스 형택이가 은진이의 사무실로 걸어오고 있다. 만약 은진이가 형택이가 들어오기 전까지 발전소를 고쳐놓지 못한다면, 은진이는 해고당할 것이다.

발전소를 고치는 방법은 간단하다. 고장나지 않은 발전소를 이용해서 고장난 발전소를 재시작하면 된다. 하지만, 이때 비용이 발생한다. 이 비용은 어떤 발전소에서 어떤 발전소를 재시작하느냐에 따라 다르다.

적어도 P개의 발전소가 고장나 있지 않도록, 발전소를 고치는 비용의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 발전소의 개수 N이 주어진다. N은 16보다 작거나 같은 자연수이다. 둘째 줄부터 N개의 줄에는 발전소 i를 이용해서 발전소 j를 재시작할 때 드는 비용이 주어진다. i줄의 j번째 값이 그 값이다. 그 다음 줄에는 각 발전소가 켜져있으면 Y, 꺼져있으면 N이 순서대로 주어진다. 마지막 줄에는 P가 주어진다. 비용은 36보다 작거나 같은 음이 아닌 정수이고, P는 0보다 크거나 같고, N보다 작거나 같은 정수이다.

출력

첫째 줄에 문제의 정답을 출력한다. 불가능한 경우에는 -1을 출력한다.

예제 입력 1 복사

```
3
0 10 11
10 0 12
12 13 0
YNN
3
```

예제 출력 1 복사

```
21
```

예제 입력 2 복사

```
3
0 2 4
2 0 3
4 3 0
YNN
3
```

예제 출력 2 복사

```
5
```

예제 입력 3 복사

```
5
1 10 11 12 13
3 5 17 15 8
15 13 22 3 1
10 22 14 9 3
0 1 3 9 0
NYNNN
5
```

예제 출력 3 복사

```
14
```

예제 입력 4 복사

```
3
0 1 2
1 2 3
2 3 4
NNY
2
```

예제 출력 4 복사

```
2
```

예제 입력 5 복사

```
7
1 3 0 9 3 2 8
13 28 2 3 8 9 30
9 2 14 19 15 10 23
9 2 8 15 19 23 28
15 19 28 0 13 19 15
9 15 32 19 32 0 14
2 3 19 15 23 15 28
YYNYYNY
4
```

예제 출력 5 복사

```
0
```

풀이과정

DP와 비트마스킹으로 풀이하는 문제다. 처음엔 비트마스킹을 생각을 못하고 끙끙대다가 결국 검색해보니, 외판원 순회 문제에서 사용했던 비트마스킹을 활용하는 문제였다.

💡 주요 포인트

1. 현재 잘 작동하는 발전기를 비트마스킹을 통해 체크하자.

문제에서 어떻게 활용할 지에 대해 설명하기 전에, 비트마스킹에서 중요한 두 가지 연산을 짚고 넘어가자.

1. num | (1<<i) → 비트마스킹 값을 추가해줄 때 사용
    a. num에 값을 추가하는 것이다.
        i. Ex) 8 | (1<<2) = 12로, 1000 → 1100이 된다.

        발전소로 따지면 4번 on → 4번, 3번 on으로 바뀐 것이다.
2. num & (1<<i) → 비트마스킹 조건문에서 많이 사용
    a. num과 값이 같은 번호만 반환한다.
        i. Ex) 8 & (1<<3) = 8 → 1000 반환: 현재 status 값이 8일 때, 4번 발전소만 켜져있다는 뜻이다.

        12 & (1<<3) = 8 → 1100 중 값이 일치하는 1000만 반환

        12 & (1<<2) = 4 → 1100 중 값이 일치하는 0100만 반환

즉, num & (1<<i) == (1<<i)일 경우, i는 현재 상태와 일치하는 값이다. (2진수로 표현한 num에 2^i에 해당하는 비트가 포함되어 있다)

우리는 비트마스킹을 통하여,

1. 초기의 발전기 상태를 비트필드를 이용한 상태값으로 저장한 뒤,

```
int pos =0;
int cnt = 0;
for(int i=0; i<status.length; i++) {
  if(status[i].equals("Y")) {
    pos = pos | (1<<i);
    cnt++;
  }
}
```

2. 작동하는 발전기의 갯수가 P와 일치할 때까지 **재귀호출** 탐색을 반복할 것이다.

2. dp 배열은 어떻게 선언하고, 점화식은 어떻게 세울까?

1. dp 배열
   a. 일단 dp 배열은 2차원 배열로 선언할 것이다.

      dp[현재 정상작동하는 발전소 갯수][현재 정상작동하는 발전소 상태 비트필드]

      - 현재 정상작동하는 발전소의 갯수가 이렇고 상태는 이럴 때, 조건을 만족할 시의 비용의 최솟값
      - 즉, cnt가 P개 이상을 만족할 때까지 재귀호출을 하며 내려가다가 cnt ≥ P를 만족하는 순간, 그때의 상태에서 조건을 만족할 시의 최소 비용들을 계산해나가며, 최종적으로는 처음 주어진 상태에서 조건을 만족할 때의 비용의 최솟값이 저장되는 것
2. 점화식
   a. dp[cnt][posNum] = min(dp[cnt][posNum], solution(cnt+1, posNum | (1<<offPos) + cost[onPos][offPos])
3. 불가능한 경우
   a. 발전소가 모두 꺼져있고 P가 1 이상일 경우
      i. 모두 꺼져있더라도 P가 0이면 불가능이 아니다!

소스코드

```
package DP.BOJ1102;

import java.io.*;
import java.util.StringTokenizer;

/*
* Ref: https://loosie.tistory.com/230
*
* 비트마스킹
* */
```

```
public class Main {

    static int N, P, curOn;
    static int[][] cost, dp;
    static int isOn;
    static int answer;
    static final int init = Integer.MAX_VALUE;
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
        N = Integer.parseInt(br.readLine());

        cost = new int[N][N];
        dp = new int[N+1][1<<16];
        for(int i=0 ; i<=N ; i++){
            Arrays.fill(dp[i], -1);
        }

        StringTokenizer st;
        for(int i=0 ; i<N ; i++){
            st = new StringTokenizer(br.readLine());
            for(int j=0; j<N ; j++){
                cost[i][j] = Integer.parseInt(st.nextToken());
            }
        }

        String status = br.readLine();
        curOn = 0;
        // 비트마스킹
        for(int i=0 ; i<status.length() ; i++){
            if(status.charAt(i) == 'Y'){
                isOn = isOn | (1<<i);
                curOn++;
            }
        }

        P = Integer.parseInt(br.readLine());

        answer = solution(curOn, isOn);
        answer = (answer == init ? -1 : answer);
        bw.write(answer +"\n");
        bw.flush();
        bw.close();
        br.close();
    }

    static int solution(int cnt, int posNum) {
        // 현재 정상작동되는 기기가 P개 이상일 경우
        if(cnt >= P) return 0;

        // 이미 계산한 경우
        if(dp[cnt][posNum] != -1) return dp[cnt][posNum];

        dp[cnt][posNum] = init;
        for(int i=0 ; i<N ; i++){
            // posNum의 발전소가 작동 중인 경우
            if((posNum & (1<<i)) == (1<<i)) {
                for(int j=0 ; j<N ; j++){
                    // 같은 발전소일 경우, 혹은 j번째 발전소도 켜져있는 경우 생략
                    if((i==j) || (posNum & (1<<j)) == (1<<j)) continue;

                    // 최솟값 - 점화식
                    dp[cnt][posNum] = Math.min(dp[cnt][posNum], solution(cnt+1, posNum|(1<<j)) + cost[i][j]);
                }
            }
        }

        return dp[cnt][posNum];
    }
```

```
}
```

소감

백준 1102 문제 밑에 비트마스킹이라고 쓰여있는 걸 문제 다 풀고 봤다. 물론 미리 알고리즘 분류를 봐서 알고 푸는 것보다 모르는 상태로 접근하는 게 더 좋지만, 모를 땐 저기서라도 힌트를 좀 얻어야겠다.

그리고 비트마스킹이 아직 익숙하지 않다. 다음 문제도 비트마스킹을 쓰는 문제로 풀어봐야겠다.

그리고...어렵다...이해는 했어도 어렵다...이 문제는 더 복습해야겠다.