```
 1 package problem4;import java.util.ArrayDeque;import
    java.util.Deque;/* 슬라이딩 윈도우를 이용하는O(n)문제입니다.
 2 Deque를 이용하면 슬라이딩 윈도우를 쉽게 구현할 수 있습니다. 한편,
   우리가 계산해야 하는 목적 함수인
 3 y[i] + y[j] + |x[i]-x[j]|-(1)에서,x[i]<x[j]이므로 (i<
   j이고,엄격한 증가수열이므로)|x[i]-x[j]|=x[j]-x[i]-(2)이 되고,
 4 원식(1)을 다시 정리하면 (y[j] + x[j]) + (y[i] - x[i])-(3
   ) 가 됩니다. 따라서 각 점에서 x좌표와 y좌표의 합과 차를 구해서
 5 두 값의 합을 계산하면 더 편리하게 계산이 가능합니다. (1)처럼 두
   좌표가 주어져야 계산을 할 수 있는 것보다는,(2)처럼 각 좌표에서
 6 계산할 수 있는걸 미리 계산해둘 수 있기 때문에 이러한 수학적 표현을
   익혀두면 구현이 매우 간단해집니다!
 7  * 코드 참조 : https://leetcode.com/problems/max-value
   -of-equation/discuss/1985474/Java-O(N)-Deque.-
   Similar-Sliding-Window-Maximum*/
 8 class Solution { public int solution(int[] x, int
   [] y, int k) {Deque<int[]> q = new ArrayDeque<>();
 9     int ans = Integer.MIN_VALUE;for (int i = 0; i
    < x.length; i++) {// k보다 차이가 커지면 제거한다.
10         while (!q.isEmpty() && q.peek()[0] + k < x[
   i]) {q.remove();}// (3) 수식 참조
11         int diff = y[i] - x[i];int plus = y[i] + x[
   i];if (!q.isEmpty()) {
12             int eq=plus+q.peek()[1];//수식(3)
13             ans = Math.max(ans, eq);// diff가 이미
   계산된 것보다 작으면 필요가 없으니 모두 뺀다.
14         }while (!q.isEmpty() && q.getLast()[1] <
   diff) q.removeLast();
15         q.add(new int[] {x[i], diff}); // 지나온
   인덱스는 diff만 필요}return ans;}}
16         class Test{public static void main(String
   [] args){
17             int[] x = {1, 2, 5, 6};int[] y = {3, 1,
   10,-9};
18             int k = 2;System.out.println(new
   Solution().solution(x, y, k));}}
19 package problem5;import java.util.ArrayList;
20 import java.util.Arrays;import java.util.List;
21 //이분 매칭 문제입니다! 강의에서 다루지 않은 내용이라 생소하시겠지만,
   배운 것만 풀 수는 없기 때문에 출제하였습니다.
22 //백준 온라인 저지의 원본 문제를 참고하시면 이해에 도움이 되실것
```

```
22 같습니다:)원본문제:https://www.acmicpc.net/problem/1867
23 class Solution { List<List<Integer>> adjNode;int[]
   bipartiteMatch;boolean[] visited;
24     public int solution(int N, int[][] stones){
   adjNode = new ArrayList<>();for (int i = 0; i<N;i
   ++){
25             adjNode.add(new ArrayList<>());}for(int
   []stone:stones)adjNode.get(stone[0]).add(stone[1]);
26         bipartiteMatch = new int[N];Arrays.fill(
   bipartiteMatch, -1);int count = 0;
27         for (int i = 0; i < N; i++) {visited = new
   boolean[N];count += dfs(i);}return count;}
28     int dfs(int node) {if (visited[node]) {return 0
   ;}visited[node]=true;for(int i:adjNode.get(node)){
29             if (bipartiteMatch[i] == -1 || dfs(
   bipartiteMatch[i]) == 1) {bipartiteMatch[i] = node;
30                 return 1;}}return 0;}}
31 class Test{public static void main(String[] args){
32         int N = 6;int[][] stones = {{0, 1}, {0, 2
   }, {5, 3}, {4, 3}};System.out.println
33             (new Solution().solution(N,stones));}}
```