Codeforces ProblemSet 1205F

Beauty of a Permutation

1、题目描述：

Define the beauty of a permutation of numbers from 1 to n $(P_1, P_2, \cdots, P_n)$ as number of pairs (L, R) such that $1 \le L \le R \le n$ and numbers $P_L, P_{L+1}, \cdots, P_R$ are consecutive $R - L + 1$ numbers in some order. For example, the beauty of the permutation $(1, 2, 5, 3, 4)$ equals 9, and segments, corresponding to pairs, are $[1], [2], [5], [4], [3], [1, 2], [3, 4], [5, 3, 4], [1, 2, 5, 3, 4]$.

Answer q independent queries. In each query, you will be given integers n and k. Determine if there exists a permutation of numbers from 1 to n with beauty equal to k, and if there exists, output one of them.

Input

The first line contains a single integer q $(1 \le q \le 10000) -$ the number of queries.

Follow q lines. Each line contains two integers n, k $(1 \le n \le 100, 1 \le k \le \frac{n(n+1)}{2}) -$ the length of permutation and needed beauty respectively,

Output

For a query output "NO", if such a permutation doesn't exist. Otherwise, output "YES", and in the next line output n numbers $-$ elements of permutation in the right order.

Examples

input

```
4
1 1
5 6
5 8
5 10
```

output

```
YES
1
YES
2 4 1 5 3
NO
YES
2 3 1 4 5
```

2、中文翻译

我们定义 1 到 n 的一个排列 $(P_1, P_2, \cdots, P_n)$ 的美丽数字为满足如下条件的数字对 (L, R) 的数量：$1 \le L \le R \le n$ 并且 $P_L, P_{L+1}, \cdots, P_R$ 是连续的 $R-L+1$ 个数字。举个例子，$(1,2,5,3,4)$ 的美丽数字为9，相应满足条件的子段是$[1],[2],[5],[4],[3],[1,2],[3,4],[5,3,4],[1,2,5,3,4]$。

你要回答 q 个查询。在每个查询中，包含两个整数 n 和 k 。判断是否存在一个 1 到 n 的排列满足其美丽数字为 k 。如果存在，输出任意一个满足条件的排列。

输入

第一行只包含一个整数 $q\ (1 \le q \le 10000)-$ 表示查询数量。

接下来 q 行，每一行包含两个整数 $n, k\ (1 \le n \le 100, 1 \le k \le \frac{n(n+1)}{2})-$ 分别表示排列的长度和需要查询的美丽数字。

输出

对于一个查询，如果不存在满足条件的排列，则输出 "NO" 。否则输出 "YES" ，并且在接下来的一行输出n 个数字，为满足条件的排列。

输入输出样例

输入
4 1 1 5 6 5 8 5 10

输出
YES 1 YES 2 4 1 5 3 NO YES 2 3 1 4 5

3、题解

- **为了方便叙述，我们作以下约定：**
  - 美丽数字为 k 的一个 1 到 n 的排列称为 (n,k) 排列。
  - 对于一个特定的排列 A，记子段 A_R^L 为 $(A_L, A_{L+1}, \cdots, A_R)$ ，其中 $L \le R$。

- 我们称子段 $A_{R_1}^{L_1}$ 包含子段 $A_{R_2}^{L_2}$ 当且仅当 $L_1 \le L_2 \le R_2 \le R_1$。子段$A_{R_1}^{L_1}$ 与子段 $A_{R_2}^{L_2}$ 互斥当且仅当两个子段没有公共元素，即 $L_1 \le R_1 < L_2 \le R_2$ 或 $L_2 \le R_2 < L_1 \le R_1$。子段 $A_{R_1}^{L_1}$ 与子段 $A_{R_2}^{L_2}$ 部分重叠当且仅当两个子段没有包含关系却有公共元素，即 $L_1 \le L_2 \le R_1 < R_2$ 或 $L_2 \le L_1 \le R_2 < R_1$。
- 若一个子段是一段连续数字的排列，则我们称这个子段是连续的。若该子段不仅是连续的，还与任何其它连续子段不部分重叠，则我们称这个子段是孤立的。
- 对于 (n_1, k_1) 排列 A $(A_1, A_2, \cdots, A_{n_1})$ 和 (n_2, k_2) 排列 B $(B_1, B_2, \cdots, B_{n_2})$，定义 $A \bigoplus_i B$ 为如下排列 C : $(C_1, C_2, \cdots, C_{n_1+n_2-1})$ 。将排列 A 中大于 A_i 的数字加上 $n_2 - 1$ ，再将 A_i 替换为排列 B，并将其中的数字加上 $A_i - 1$。完整的定义如下：

$$C_m = \begin{cases} A_m, \quad m \in [1, i) \; and \; A_m < A_i \\ A_m + n_2 - 1, \quad m \in [1, i) \; and \; A_m > A_i \\ A_{m-n_2+1}, \quad m \in [\, n_2 + i, n_1 + n_2 - 1 \,] \; and \; A_{m-n_2+1} < A_i \\ A_{m-n_2+1} + n_2 - 1, \quad m \in [\, n_2 + i, n_1 + n_2 - 1 \,] \; and \; A_{m-n_2+1} > A_i \\ B_{m-i+1} + A_i - 1, \quad m \in [\, i, n_2 + i - 1 \,] \end{cases}$$

  举个例子排列 A 为 $(1, 2, 3, 5, 4)$，排列 B 为 $(3, 1, 4, 2)$，则 $A \bigoplus_3 B$ 为 $(1, 2, 5, 3, 6, 4, 8, 7)$

- **引理1：对于 (n_1, k_1) 排列 A，(n_2, k_2) 排列 B，记 $C = A \bigoplus_i B$，则排列 C 的美丽数大于等于 $k_1 + k_2 - 1$，其中取等当且仅当 C 中连续的子段不与 $C_{i+n_2-1}^i$ 部分重叠。**

  证明：证明的关键在于子段 $C_{i+n_2-1}^i$，即 $(B_1 + A_i - 1, B_2 + A_i - 1, \cdots, B_{n_2} + A_i - 1)$ 。

  - C 中包含 $C_{i+n_2-1}^i$ 或与 $C_{i+n_2-1}^i$ 互斥的子段与 A 中所有子段一一对应。因为对于这些子段来说，我们将 $C_{i+n_2-1}^i$ 替换为 A_i ，将剩下大于 $A_i + n2 - 1$ 的数减去 $n_2 - 1$ 即可得到对应于 A 的子段。易知这种替换不会改变子段的连续性，所以这些子段中连续的子段与 A 中连续的子段也一一对应。
  - C 中包含于 $C_{i+n_2-1}^i$ 的子段，这些子段与 B 中子段一一对应，所有数字加上一个共同的数不会影响子段的连续性，因此这些子段中连续的子段与 B 中连续的子段一一对应。

    综上所述，上述考察的子段只有 $C_{i+n_2-1}^i$ 在 A、B 中都有映射，其它子段要么只与 A 有对应，要么只与 B 对应。所以美丽数大于等于 $k_1 + k_2 - 1$。剩下没有考虑的子段都与 $C_{i+n_2-1}^i$ 部分重叠，因此若这些子段中没有连续的，排列 C 的美丽数即为 $k_1 + k_2 - 1$。

- **定理1：如果存在 (n_1, k_1) 排列 A 和 (n_2, k_2) 排列 B ，则存在一个 $(n_1 + n_2 - 1, k_1 + k_2 - 1)$ 排列 C，且排列 C 可以通过排列 A 和 B 构造出来。**

  证明：我们记排列 A 为 $(A_1, A_2, \cdots, A_{n_1})$ ，而排列 B 为 $(B_1, B_2, \cdots, B_{n_2})$ 。i 为满足如下条件的整数 ，$A_i = n_1$。即 i 表示排列 A 中最大数 n_1 所在的位置。则 $C = A \bigoplus_i B$ 即为满足条件的排列。排列如下：

  $(A_1, A_2, \cdots, A_{i-1}, B_1 + n_1 - 1, B_2 + n_1 - 1, \cdots, B_{n_2} + n_1 - 1, A_{i+1}, \cdots, A_{n_1})$

  显然 C 的长度为 $n_1 + n_2 - 1$，由引理1可知我们仅需证明 C 中与 $C_{i+n_2-1}^i$ 部分重叠的子段中没有连续的。

  - 若 $n_2 = 1$ 则结论显然，因为 C_i^i 仅含一个元素，不可能有子段与其部分重叠。
  - 若 $n_1 = 1$ 则结论显然，因为所有子段都包含于 $C_{n_2}^1$，不可能有子段与其部分重叠。
  - 若 $n_1 > 1$，$n_2 > 1$ 则我们要求排列 A 中 $n_1 - 1$ 出现在 n_1 之前，排列 B 中 1 出现在 n_2 之后。（这很容易做到，因为若排列不满足条件，则我们将排列倒序后即可得到满足要求的排列。）

考察 C 中与 $C^i_{i+n_2-1}$ 部分重叠的子段，我们证明这些子段没有连续的。因为若某个子段连续，则由其最小值小于 n_1 ，最大值大于等于 n_1 知，该子段肯定包含 n_1-1 和 n_1 ，而由前面的关于排列 A、B 的假设知，C 中 n_1+n_2-1 在 n_1-1 和 n_1 之间，因此该子段肯定包含 n_1 和 n_1+n_2-1 ，可以得出该子段包含 $C^i_{i+n_2-1}$，与该子段与 $C^i_{i+n_2-1}$ 部分重叠矛盾。

所以排列 C 的美丽数为 k_1+k_2-1 。我们将上述构造过程记为 $C=A\bigoplus B$。

- **定理2：如果存在一个 (n,k) 排列 C，且 $n\geq 3, k>n+1, k<\frac{n(n+1)}{2}$ ，则存在两个长度大于1的排列 A、B，满足 $D=A\bigoplus B$ 也是一个 (n,k) 排列。**

  证明的关键是在 C 中找到一个满足条件的长度大于 1 小于 n 的子段，其是孤立的。

  如果存在这样一个子段，我们可以通过逆向引理1的构造过程得到两个排列 A、B，满足美丽数之和减一等于 k，长度之和减一等于 n。排列 A 的长度与该子段相同，所以排列 B 的长度也介于 2 到 $n-1$ 之间。再由定理1可知 $D=A\bigoplus B$ 即是一个 (n,k) 排列。举个例子，若 C 为 $(1,2,5,3,6,4,8,7)$，我们取 $(5,3,6,4)$ 为这样一个子段，取 $A=(1,2,3,5,4), B=(3,1,4,2)$ ，通过验证 $A\bigoplus B$ 满足要求。

  我们令集合 S 为 C 中连续且长度小于 n 的子段集合。集合 S 非空，因为每个单个元素构成的子段均在集合 S 中。而集合 S 是有限的，因为子段数最多为 $\frac{n(n+1)}{2}$。我们令 H 为集合 S 中长度最长的那个子段。（如果存在多个长度最大的子段，则任取一个即可）。记 $H=C^L_R$，由 $k>n+1$ ，知 H 长度肯定在 2 到 $n-1$ 之间。

  - 若子段 H 是孤立的，则 H 就是要寻找的子段。
  - 若子段 H 不是孤立的，则存在连续的子段与子段 H 部分重叠，可将它们合并后可以得到一个更大的连续子段 P。若 P 的长度小于 n，则与 H 的定义相矛盾。因此 P 的长度为 n ，从而（1）$H=C^1_R$ 或 C^L_n。我们不妨设 $H=C^1_R$。（另一种情况同理可证）(2) 与H 部分重叠的连续子段均包含 C^{R+1}_n。因此不存在连续且与 C^{R+1}_n 部分重叠的子段。若 C^{R+1}_n 的长度大于 1，则 C^{R+1}_n 即为要寻找的子段。
  - 若 C^{R+1}_n 的长度等于 1，则排列中最后一个数要么是 1，要么是 n。我们不妨设为 n（为 1 时同理可证）。取满足下列条件的最长子段 Q：$C^j_n=(j,j+1,\cdots,n)$。这是良定义的，因为 C^n_n 满足要求。由 $k\neq\frac{n(n+1)}{2}$ 知，$j>2$。再取满足下列条件的最短连续子段 T：(1) 型式为 C^i_{j-1} (2) 包含数字 $j-1$。T 是良定义的，因为 C^1_{j-1} 即满足要求。又由 $C_{j-1}\neq j-1$ 知 T 的长度大于 1。下证子段 T 是孤立的：
    - 若存在连续子段 C^L_R 满足 $R\geq j$ 且与 T 部分重叠，则重叠部分构成的子段长度必定小于 T、满足连续性且包含数字 $j-1$。这样的子段与 T 的定义相矛盾。
    - 若存在连续子段 C^L_R 满足 $L<i$ 且与 T 部分重叠，易知 T 的数字范围为 $[i,j-1]$。C^L_R 中的数字都小于等于 $j-1$，因此该子段要与 T 部分重叠，其必不能包含数字 $j-1$ 。从而取 T 中不与 C^L_R 重叠的数字构成的子段，也满足连续，型式为 C^i_{j-1} 且包含数字 $j-1$。从而与 T 的最短性矛盾。
    - 从而 T 是孤立的。也是要寻找的子段。

    综上所述，每个排列 C 均存在我们要找的子段。所以定理成立。
- **一个 (n,k) 排列，如果 $n\geq 3, k>n+1, k<\frac{n(n+1)}{2}$，则称该排列是可分割排列。反之称为不可分割排列。**
- **定理3：对于每个n，均存在一个 $(n,\frac{n(n+1)}{2})$ 排列。对于 $n\neq 1$、3，均存在一个 $(n,n+1)$ 排列。**

- 排列 $(1,2,\cdots,n)$ 即为一个 $(n,\frac{n(n+1)}{2})$ 排列。
- 对于 $n=1$，不可能存在 $(1,2)$ 排列。
- 对于 $n=2$，$(n,\frac{n(n+1)}{2})$ 排列即是一个$(n,n+1)$ 排列。
- 对于 $n=3$，通过枚举可以验证不存在 $(3,4)$ 排列。
- 对于 $n>4$ 来说我们可以构造如右排列 $(3,5,7,\cdots,1,n,2,4,\cdots)$ 。易验证该排列满足要求。

- **定理4：对于任意的 (n,k) 来说，若存在一个 (n,k) 排列，则存在一个排列 $A=B_1\bigoplus B_2\bigoplus\cdots\bigoplus B_k$。其中排列 A 是一个 (n,k) 排列，而 B_1、B_2、$\cdots$、B_k 均是不可分割的排列。**

  证明：若 (n,k) 排列是不可分割的，则直接取 B_1 为该排列即可。若 (n,k) 排列是可分割的，则根据定理2，可以将该排列分割为两个排列。重复运用定理2即可得到一系列不可分割排列 B_1、B_2、$\cdots$、B_n。因为对于 $\bigoplus$ 操作来说，交换计算顺序不会导致最终排列的长度和美丽数发生变化，因此可以得出 $B_1\bigoplus B_2\bigoplus\cdots\bigoplus B_k$ 是一个 (n,k) 排列。

- **最终我们可以得到如下的动态转移方程，记 E_n^k 是一个布尔变量，表示是否存在一个 (n,k) 排列。通过计算动态转移方程的过程，我们也可以很轻易的计算出一个 (n,k) 排列。**

$$E_n^k=\begin{cases} true, & k=\frac{n(n+1)}{2} \\ true, & k=n+1\ and\ n\neq 1、3 \\ (\bigvee_{i=2\ to\ n-1} E_{n+1-i}^{k+1-\frac{i(i+1)}{2}})\vee(\bigvee_{i=4\ to\ n-1} E_{n+1-i}^{k-i}), & n>=3\ and\ k>n+1\ and\ k<\frac{n(n+1)}{2} \end{cases}$$

4、代码

```
#include<iostream>
#include<string>
using namespace std;
bool exist[101][51 * 101];  //exist[n][k]表示是否存在一个满足美丽数字为k的1到n的排列
int last[101][51 * 101];
bool order[101][51 * 101];
void permutation(int n, int k, int base, bool coutOrder) {
    int number = last[n][k];
    if (n == 1) {
        cout << base + 1;
        return;
    }
    if (coutOrder && order[n][k]) {
        for (int i = 1; i < number; i++) cout << base + i << " ";
        permutation(n+1-number,k+1-number*(number+1)/2, base+number-1,
!coutOrder);
    }
    else if (!coutOrder && order[n][k]) {
        permutation(n+1-number,k+1-number*(number+1)/2, base+number-1,
!coutOrder);
        for (int i = number - 1; i >= 1; i--) cout << " " << base + i;
    }
    else if (coutOrder && !order[n][k]) {
        if (n % 2 == 0) {
            for (int i = n - 1; i >= 1; i = i - 2) cout << base + i << " ";
```

```
            permutation(n + 1 - number, k - number, base + number - 1,
!coutOrder);
            for (int i = n - 2; i >= 2; i = i - 2) cout << " " << base + i;
        }
        else {
            for (int i = n - 1; i >= 2; i = i - 2) cout << base + i << " ";
            permutation(n + 1 - number, k - number, base + number - 1,
!coutOrder);
            for (int i = 1; i <= n - 2; i = i + 2) cout << " " << base + i;
        }
    }
    else if (!coutOrder && !order[n][k]) {
        if (n % 2 == 0) {
            for (int i = 2; i <= n - 2; i = i + 2) cout << base + i << " ";
            permutation(n + 1 - number, k - number, base + number - 1,
!coutOrder);
            for (int i = 1; i <= n - 1; i = i + 2)cout << " " << base + i;
        }
        else {
            for (int i = n - 2; i >= 1; i = i - 2) cout << base + i << " ";
            permutation(n + 1 - number, k - number, base + number - 1,
!coutOrder);
            for (int i = 2; i <= n - 1; i = i + 2) cout << " " << base + i;
        }
    }
}
int main()
{
    for (int i = 1; i <= 100; i++)
        for (int j = 1; j <= i * (i + 1) / 2; j++)
            exist[i][j] = false;
    for (int i = 1; i <= 100; i++) {
        exist[i][i*(i + 1) / 2] = true;
        last[i][i*(i + 1) / 2] = i;
        order[i][i*(i + 1) / 2] = true;
        if (i >= 4) {
            exist[i][i + 1] = true;
            last[i][i + 1] = i;
            order[i][i + 1] = false;
        }
        for (int j = i + 2; j < i * (i + 1) / 2; j++) {
            for (int k = 2; k <= i - 1; k++)
                if (j - k*(k+1)/2 > i - k && exist[i + 1 - k][j + 1 - k*
(k+1)/2]) {
                    exist[i][j] = true;
                    last[i][j] = k;
                    order[i][j] = true;
                    break;
                }
```

```
            if (!exist[i][j])
                for (int k = 4; k <= i - 1; k++)
                    if (j > i + 1 && exist[i + 1 - k][j - k]) {
                        exist[i][j] = true;
                        last[i][j] = k;
                        order[i][j] = false;
                        break;
                    }
        }
    }
    int q, n, k;
    cin >> q;
    for (int i = 1; i <= q; i++) {
        cin >> n >> k;
        if (exist[n][k]) {
            cout << "YES" << endl;
            permutation(n, k, 0, true);
            cout << endl;
        }
        else cout << "NO" << endl;
    }
    return 0;
}
```