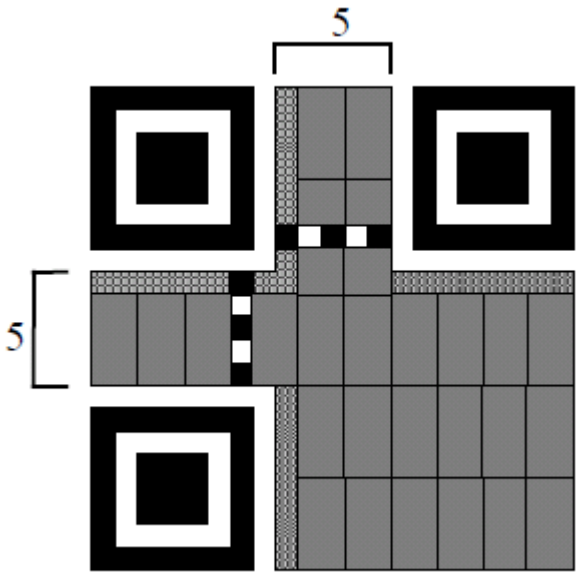
# F • QR

*QR Codes* (the smallest, which is 21 pixels by 21 pixels, is shown below) are square arrays of black or white pixels (modules) which include *Position Detection Patterns* (the square bull's-eye patterns), *Timing Patterns* (the alternating black and white lines), *Alignment Patterns* in larger *QR Codes*, *Format Information* (the stippled pixels), *Version information* in larger *QR Codes* and *Data and Error Correction Codewords* (gray 8 pixel blocks).




The 21-by-21 *QR Code* has *26 data and error correction codewords*. At the lowest error correction level for this code, 19 are *data codewords* and 7 are *error correction codewords*. Data may be encoded as numeric at 3 numbers per 10 bits, as alphanumeric at 2 characters per 11 bits, as 8 bit bytes or as Kanji at 13 bits per character. Data is encoded in groups of (mode, character count, character data bits). The mode can change within the data stream. The mode is specified by a 4 bit code and the character count by a varying number of bits depending on the mode and *QR Code* size. For the 21-by-21 code, the character count bits are:

| Mode Name | Mode Bits | Count Bits |
|---|---|---|
| Numeric | 0001 | 10 |
| Alphanumeric | 0010 | 9 |
| 8 bit byte | 0100 | 8 |
| Kanji | 1000 | 8 |
| Termination | 0000 | 0 |

The entire data stream ends in the *termination code* which may be truncated if there is not enough room. Any partially filled codeword after the termination code is filled with 0 bits. Any remaining codewords are set to `11101100` followed by `00010001` alternating.

Numeric strings are encoded 3 digits at a time. If there are remaining digits, 2 digits are encoded in 7 bits or 1 digit in 4 bits. For example:

`12345678` → `123 456 78` → `0001111011 0111001000 1001110`

Prefix with mode (`0001`) and count (`8` → `0000001000`) is (`4 + 10 + 10 + 10 + 7`) bits:

`0001 0000001000 0001111011 0111001000 1001110`

Alphanumeric strings encode the characters (<SP> represents the space character):

`0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ<SP>$%*+-./:`

as numbers from `0` to `44`, then two characters are encoded in `11` bits:

*<first char code>* `* 45 +` *<second char code>*

if the number of characters is odd, the last character is encoded in 6 bits. For example:

`AC-42` → `(10, 12, 41, 4, 2)` → `10*45 + 12 = 462, 41*45 + 4 = 1849, 2` →
`00111001110 11100111001 000010`

Prefix with mode and count is $(4 + 9 + 11 + 11 + 6)$ bits:

`0010 000000101 00111001110 11100111001 000010`

The 8 bit binary and Kanji modes will be straightforward for the purposes of this problem. Kanji codes will just be opaque 13 bit codes; you need not decode the characters they represent, just the hexadecimal values. For example:

`8 bit 0x45 0x92 0xa3` → `01000101 10010010 10100011`

Prefix with mode and count is (`4 + 8 + 8 + 8 + 8`) bits:

`0100 00000011 01000101 10010010 10100011`

Kanji `0x1ABC 0x0345` → `1101010111100 0001101000101`

Prefix with mode and count is (`4 + 8 + 13 + 13`) bits:

`1000 00000010 1101010111100 0001101000101`

To illustrate forming the 19 codeword content of a *QR Code*, combine the first 3 sequences above (for numeric, alphanumeric and bytes). Concatenate the bits, split into 8 bit code words add the termination codeword, any fill bits and fill bytes (`41 + 41 + 36` *data bits* + 4 bit *termination code* = `122` → `6` fill bits are needed to get `16` bytes, and to fill out the 19 bytes, 3 fill bytes are needed):

```
0001 0000001000 0001111011 0111001000 1001110
0010 000000101 001110011110 111001111001 000010
0100 00000011 01000101 10010010 10100011
0000 000000 11101100 00010001 11101100
```

split into 8 bit codewords:

```
00010000 00100000 01111011 01110010 00100111 00010000 00010100 11100111
01110011 10010000 10010000 00001101 00010110 01001010 10001100 00000000
11101100 00010001 11101100 → HEX 10207B72271014E77390900D164A8C0EC11EC
```

Write a program to read 19 codewords and print the corresponding data.

**Input**

The first line of input contains a single integer ***P***, (`1 ≤ P ≤ 1000`), which is the number of data sets that follow. Each data set is a single line of input consisting of the data set number, ***N***, followed by a single space and 38 hexadecimal digits giving the 19 bytes of *QR Code* data. The valid hexadecimal digits are `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E` and `F`.

**Output**

For each data set there is one line of output. It contains the data set number (***N***) followed by a single space, the number of *QR decoded* characters in the result, a single space and the character string corresponding to the *QR Code* data. In the output string, printable ASCII characters (in the range `0x20` to `0x7e`) are printed as the ASCII character *EXCEPT* that backslash (****) is printed as **** and pound sign (**#**) is printed as **\#**. *Non-printable* 8 bit data is output as **\xx**, where **x** is a hexadecimal digit (e.g. **\AE**). *Non-printable* 8 bit data is any value that is less than the ASCII value of a space (`0x20`) or greater than `0x76`. 13 bit Kanji values are printed as **#bxxx**, where **b** is **0** or **1** and **x** is a hexadecimal digit (e.g. **#13AC**).

| Sample Input |
| --- |
| 4<br>1 10207B72271014E77390900D164A8C00EC11EC<br>2 802D5E0D1400EC11EC11EC11EC11EC11EC11EC<br>3 20BB1AA65F9FD7DC0ED88C973E15EF533EB0EC<br>4 2010B110888D9428D937193B9CEA0D7F45DF68 |

| Sample Output |
| --- |
| 1 16 12345678AC-42E\92\A3<br>2 2 #1ABC#0345<br>3 23 HTTP://WWW.ACMGNYR.ORG/<br>4 36 3.14159265358979323846264338327950 28 |