Talking about FP with Hamming Codes Problem

This post was originally written in Chinese and published on CSDN.

Problem Link

The problem asks to find the lexicographically smallest `n` numbers such that the hamming distance between any two numbers is at least `d`.

Hamming distance can be calculated using XOR. `1^0=1`, `0^1=1`, `0^0=0`, `1^1=0`. So, XORing two numbers will result in a number where the set bits represent the differing bits. We can then count the number of set bits in the result.

I made a mistake once because the output requires 10 numbers per line, with the last line potentially having fewer than 10. My initial output had a trailing space after the last number on the last line, followed by a newline.

I think this is a pretty good Functional Programming style code. The benefit is that it's more structured, making `main` act like a top-level in Lisp or other functional languages.

This way, I don't need to create a new cpp file to test unfamiliar functions or debug individual functions. I can just comment out `deal()` and use `main` as a top-level REPL (read-print-eval-loop).

Lisp also taught me to program as functionally as possible, FP! This way, each function can be extracted and debugged separately. The semantics are also clearer. For example:

`hamming(0, 7, 2)` means to check if the binary representations of 0 and 7 differ by at least 2 bits. 7 is `111`, so they differ by 3 bits, and the function returns true.

So, I can comment out `deal()` and add `hamming(0, 7, 2)` to test this function independently.

AC Code:

```
/*
{
ID: lzwjava1
PROG: hamming
LANG: C++
}
*/
#include<cstdio>
#include<cstring>
#include<math.h>
#include<stdlib.h>
#include<algorithm>
```

```
#include<ctime>
using namespace std;
const int maxn=1000;

bool hamming(int a,int b,int d)
{
    int c=a^b;
    int cnt=0;
    for(int i=0;i<=30;i++)
    {
        if((1<<i) & c)
        {
            cnt++;
            if(cnt>=d) return true;
        }
    }
    return false;
}

void printArr(int *A,int n)
{
    for(int i=0;i<n;i++)
    {
        printf("%d",A[i]);
        if((i+1)%10==0 || (i==n-1)) printf("\n");
        else printf(" ");
    }
}

bool atLesat(int *A,int cur,int i,int d)
{
    for(int j=0;j<cur;j++)
        if(!hamming(A[j],i,d))
            return false;
    return true;
}

void dfs(int *A,int cur,int n,int d)
{
    if(cur==n)
    {
        printArr(A,n);
        return;
    }
    int st=(cur==0? 0: A[cur-1]+1);
    for(int i=st;;i++)
    {
        if(atLesat(A,cur,i,d))
        {
            A[cur]=i;
            dfs(A,cur+1,n,d);
            return;
        }
    }
}

void deal()
{
    int n,b,d;
    scanf("%d%d%d",&n,&b,&d);
    int A[n];
    dfs(A,0,n,d);
}

int main()
{
  freopen("hamming.in","r",stdin);
  freopen("hamming.out","w",stdout);
    deal();
    //printf("%.2lf\n",(double)clock()/CLOCKS_PER_SEC);
  return 0;
}

/*
*/
```