cst(x,d,k) tests whether or not an integer x lies between d and k.

cst := proc(x,d,k)
if x<d then
    RETURN(0):
elif x>k then
    RETURN(0):
else
    RETURN(1):
fi:
end:

to compute C*(m,d,k), we'll use a Maple function
combinat[composition](n,k)
which produces a list of all ordered k-tuples of integers that add to n. For example,
combinat[composition](5,3) = [ [3, 1, 1], [2, 2, 1], [1, 3, 1], [1, 2, 2], [2, 1, 2], [1, 1, 3] ]

Our idea for computing C*(m,d,k) is as follows. For a given string in S*(m,d,k) the number of 1's that can occur ranges from 2 to m/(d+1). The first extreme corresponds to a string that starts and ends with 1's, the second extreme to when 1's are separated by the substrings consisting of d consecutive 0's.

Knowing number n1s of 1's in a string, we can deduce the number n0s of 0's. It'll be n0s = m-n1s.

Next, we can use the combinat[composition] function to figure out how the 0's can be split up into contiguous substrings of 0's with each each contiguous substring containing at least d and at most k substrings.

Explicitly, we'll use combinat[composition] to figure out all the ways positive integers can sum to n0s and then test the individual integers i in such a combination satisfy the requirement d <= i <= k. Each time we find such an admissible combination, we count it. The routine below implements this idea.
for p in cp do
    t := convert(map(x->ctst(x,d,k),p),`*`):
    if t = 1 then
        res := res+1:
    fi:
    od:
od:
RETURN(res):
end:

Now let's assume a suitable value of k to be 24 and compute log_2(C(m,d,k) for various values of m and d. For small values of m, we will base the computations on the explicit count given by the function Cstar0(m,d,k) defined above. However, by the time we reach m > k = 24, we should have enough initial data to compute C*(m,d,k) from the recursive formula defined in class.

Cstar := proc(m,d,k) option remember:
local res,i:
if m < k+3 then
    RETURN(Cstar0(m,d,k)):
else
    res := add(Cstar(m-i-1,d,k),i=d..k):
    RETURN(res):
fi:
end:
Cstar0(10,2,5);
5
Cstar(10,2,5);
5
for m from 4 to 20 do printf("m = %a , C*0(m,2,5) = %a , C*(m,2,5) = %a\n",m,Cstar0(m,2,5),Cstar(m,2,5)): od:

m = 4 , C*0(m,2,5) = 1 , C*(m,2,5) = 1
m = 5 , C*0(m,2,5) = 1 , C*(m,2,5) = 1
m = 6 , C*0(m,2,5) = 1 , C*(m,2,5) = 1
m = 7 , C*0(m,2,5) = 2 , C*(m,2,5) = 2
m = 8 , C*0(m,2,5) = 2 , C*(m,2,5) = 2
m = 9 , C*0(m,2,5) = 3 , C*(m,2,5) = 3
m = 10 , C*0(m,2,5) = 5 , C*(m,2,5) = 5
m = 11 , C*0(m,2,5) = 6 , C*(m,2,5) = 6
m = 12 , C*0(m,2,5) = 8 , C*(m,2,5) = 8
m = 13 , C*0(m,2,5) = 12 , C*(m,2,5) = 12
m = 14 , C*0(m,2,5) = 16 , C*(m,2,5) = 16
m = 15 , C*0(m,2,5) = 22 , C*(m,2,5) = 22
\[
\begin{align*}
\text{m} &= 16, \quad \text{C}^0(\text{m}, 2, 5) = 31, \quad \text{C}^*(\text{m}, 2, 5) = 31 \\
\text{m} &= 17, \quad \text{C}^0(\text{m}, 2, 5) = 42, \quad \text{C}^*(\text{m}, 2, 5) = 42 \\
\text{m} &= 18, \quad \text{C}^0(\text{m}, 2, 5) = 58, \quad \text{C}^*(\text{m}, 2, 5) = 58 \\
\text{m} &= 19, \quad \text{C}^0(\text{m}, 2, 5) = 81, \quad \text{C}^*(\text{m}, 2, 5) = 81 \\
\text{m} &= 20, \quad \text{C}^0(\text{m}, 2, 5) = 111, \quad \text{C}^*(\text{m}, 2, 5) = 111
\end{align*}
\]

\begin{verbatim}
C := proc(m,d,k)  option remember:
local i,j,res:
if m < 50 then
  res := add(add(Cstar(m-i-j,d,k),i=0..k),j=0..k):
else
  res := add(C(m-i-1,d,k),i=d..k):
fi:
RETURN(res):
end:

C(96,2,10);

C2n := proc(m,d,k)
local res:
res := floor(evalf(log[2](C(m,d,k)),5));
RETURN(res):
end:

C2n(96,3,10);

EfficiencyFactor := proc(m,d,k)
local spc,n:
spc := m/(d+1);
n := C2n(m,d,k);
RETURN(evalf(n/spc,2)):
end:

EfficiencyFactor(96,2,10); EfficiencyFactor(96,3,10);

k := 25: m := 96:
for d from 1 to k do
  n := C2n(m,d,k):
  eff := EfficiencyFactor(m,d,k):
  printf("C2n(%a,%a,%a) = %a , Eff = %a\n",m,d,k,n,eff):
od:
C2n(96,1,25) = 66 , Eff = 1.4
C2n(96,2,25) = 53 , Eff = 1.7
C2n(96,3,25) = 45 , Eff = 1.9
C2n(96,4,25) = 39 , Eff = 2.0
\end{verbatim}
\[
\begin{align*}
C_{2n}(96,5,25) &= 35, \quad \text{Eff} = 2.2 \\
C_{2n}(96,6,25) &= 32, \quad \text{Eff} = 2.3 \\
C_{2n}(96,7,25) &= 29, \quad \text{Eff} = 2.4 \\
C_{2n}(96,8,25) &= 27, \quad \text{Eff} = 2.5 \\
C_{2n}(96,9,25) &= 25, \quad \text{Eff} = 2.6 \\
C_{2n}(96,10,25) &= 24, \quad \text{Eff} = 2.8 \\
C_{2n}(96,11,25) &= 22, \quad \text{Eff} = 2.8 \\
C_{2n}(96,12,25) &= 21, \quad \text{Eff} = 2.8 \\
C_{2n}(96,13,25) &= 20, \quad \text{Eff} = 2.9 \\
C_{2n}(96,14,25) &= 18, \quad \text{Eff} = 2.8 \\
C_{2n}(96,15,25) &= 17, \quad \text{Eff} = 2.8 \\
C_{2n}(96,16,25) &= 16, \quad \text{Eff} = 2.8 \\
C_{2n}(96,17,25) &= 15, \quad \text{Eff} = 2.8 \\
C_{2n}(96,18,25) &= 14, \quad \text{Eff} = 2.8 \\
C_{2n}(96,19,25) &= 13, \quad \text{Eff} = 2.7 \\
C_{2n}(96,20,25) &= 12, \quad \text{Eff} = 2.6 \\
C_{2n}(96,21,25) &= 11, \quad \text{Eff} = 2.5 \\
C_{2n}(96,22,25) &= 10, \quad \text{Eff} = 2.4 \\
C_{2n}(96,23,25) &= 9, \quad \text{Eff} = 2.2 \\
C_{2n}(96,24,25) &= 8, \quad \text{Eff} = 2.2 \\
C_{2n}(96,25,25) &= 7, \quad \text{Eff} = 2.1 \\
\end{align*}
\]

```plaintext
> k := 25: m := 256:
> for d from 1 to k do
>   n := C2n(m,d,k):
>   eff := EfficiencyFactor(m,d,k):
>   printf("C2n(%a,%a,%a) = %a , Eff = %a\n",m,d,k,n,eff):
> od:
C2n(256,1,25) = 177 , Eff = 1.4
C2n(256,2,25) = 141 , Eff = 1.7
C2n(256,3,25) = 119 , Eff = 1.9
C2n(256,4,25) = 104 , Eff = 2.0
C2n(256,5,25) = 93 , Eff = 2.2
C2n(256,6,25) = 84 , Eff = 2.3
C2n(256,7,25) = 77 , Eff = 2.4
C2n(256,8,25) = 71 , Eff = 2.5
C2n(256,9,25) = 66 , Eff = 2.6
C2n(256,10,25) = 62 , Eff = 2.7
C2n(256,11,25) = 58 , Eff = 2.7
C2n(256,12,25) = 54 , Eff = 2.7
C2n(256,13,25) = 51 , Eff = 2.8
C2n(256,14,25) = 47 , Eff = 2.8
C2n(256,15,25) = 44 , Eff = 2.8
C2n(256,16,25) = 42 , Eff = 2.8
C2n(256,17,25) = 39 , Eff = 2.7
C2n(256,18,25) = 36 , Eff = 2.7
C2n(256,19,25) = 33 , Eff = 2.6
C2n(256,20,25) = 30 , Eff = 2.5
C2n(256,21,25) = 27 , Eff = 2.3
C2n(256,22,25) = 23 , Eff = 2.1
C2n(256,23,25) = 18 , Eff = 1.7
C2n(256,24,25) = 10 , Eff = .98
C2n(256,25,25) = 4 , Eff = .41
```
> for d from 1 to k do
>   n := C2n(m,d,k):
>   eff := EfficiencyFactor(m,d,k):
>   printf("C2n(%a,%a,%a) = %a , Eff = %a\n",m,d,k,n,eff):
> od:
C2n(1024,1,25) = 711 , Eff = 1.4
C2n(1024,2,25) = 565 , Eff = 1.7
C2n(1024,3,25) = 476 , Eff = 1.9
C2n(1024,4,25) = 415 , Eff = 2.0
C2n(1024,5,25) = 370 , Eff = 2.2
C2n(1024,6,25) = 335 , Eff = 2.3
C2n(1024,7,25) = 306 , Eff = 2.4
C2n(1024,8,25) = 282 , Eff = 2.5
C2n(1024,9,25) = 262 , Eff = 2.6
C2n(1024,10,25) = 244 , Eff = 2.6
C2n(1024,11,25) = 228 , Eff = 2.7
C2n(1024,12,25) = 213 , Eff = 2.7
C2n(1024,13,25) = 200 , Eff = 2.7
C2n(1024,14,25) = 187 , Eff = 2.7
C2n(1024,15,25) = 175 , Eff = 2.7
C2n(1024,16,25) = 163 , Eff = 2.7
C2n(1024,17,25) = 151 , Eff = 2.7
C2n(1024,18,25) = 139 , Eff = 2.6
C2n(1024,19,25) = 127 , Eff = 2.5
C2n(1024,20,25) = 115 , Eff = 2.4
C2n(1024,21,25) = 101 , Eff = 2.2
C2n(1024,22,25) = 86 , Eff = 1.9
C2n(1024,23,25) = 67 , Eff = 1.6
C2n(1024,24,25) = 42 , Eff = 1.0
C2n(1024,25,25) = 3 , Eff = .76e-1

> for k from 5 to 25 do
>   maxeff := 0:
>   dmax := 0:
>   for d from 1 to k do
>     eff := EfficiencyFactor(256,d,k):
>     if eff > maxeff then
>       dmax := d:
>       maxeff := eff:
>     fi:
>   od:
\begin{verbatim}
> n := C2n(256,dmax,k):
> printf("k = %a , dmax = %a , maxeff = %a , n = %a\n",k,dmax,maxeff,n):
> od:
>
k = 5 , dmax = 2 , maxeff = 1.4 , n = 119
k = 6 , dmax = 2 , maxeff = 1.5 , n = 128
k = 7 , dmax = 2 , maxeff = 1.6 , n = 133
k = 8 , dmax = 3 , maxeff = 1.7 , n = 109
k = 9 , dmax = 3 , maxeff = 1.8 , n = 112
k = 10 , dmax = 4 , maxeff = 1.9 , n = 96
k = 11 , dmax = 5 , maxeff = 2.0 , n = 85
k = 12 , dmax = 6 , maxeff = 2.1 , n = 76
k = 13 , dmax = 5 , maxeff = 2.1 , n = 88
k = 14 , dmax = 6 , maxeff = 2.2 , n = 79
k = 15 , dmax = 8 , maxeff = 2.3 , n = 65
k = 16 , dmax = 7 , maxeff = 2.3 , n = 73
k = 17 , dmax = 8 , maxeff = 2.4 , n = 67
k = 18 , dmax = 8 , maxeff = 2.4 , n = 68
k = 19 , dmax = 9 , maxeff = 2.5 , n = 63
k = 20 , dmax = 8 , maxeff = 2.5 , n = 70
k = 21 , dmax = 10 , maxeff = 2.6 , n = 60
k = 22 , dmax = 10 , maxeff = 2.6 , n = 60
k = 23 , dmax = 11 , maxeff = 2.7 , n = 57
k = 24 , dmax = 14 , maxeff = 2.8 , n = 47
k = 25 , dmax = 13 , maxeff = 2.8 , n = 51

> maxk := 25:
> for m from 5 to 160 do
> if m < (maxk+3) then
>   k := m-3:
> else
>   k := maxk:
> fi:
> d := floor(k/2):  # presumably the most efficient choice of d for a given k
> n := C2n(m,d,k):
> eff := EfficiencyFactor(m,d,k):
> if n > 31 then
>   printf("m = %a , d = %a , k = %a , n = %a , eff = %a\n",m,d,k,n,eff):
> fi:
> od:
> \end{verbatim}
m = 148, d = 12, k = 25, n = 32, eff = 2.8
m = 149, d = 12, k = 25, n = 32, eff = 2.8
m = 150, d = 12, k = 25, n = 32, eff = 2.8
m = 151, d = 12, k = 25, n = 32, eff = 2.8
m = 152, d = 12, k = 25, n = 32, eff = 2.7
m = 153, d = 12, k = 25, n = 33, eff = 2.8
m = 154, d = 12, k = 25, n = 33, eff = 2.8
m = 155, d = 12, k = 25, n = 33, eff = 2.8
m = 156, d = 12, k = 25, n = 33, eff = 2.8
m = 157, d = 12, k = 25, n = 33, eff = 2.7
m = 158, d = 12, k = 25, n = 34, eff = 2.8
m = 159, d = 12, k = 25, n = 34, eff = 2.8
m = 160, d = 12, k = 25, n = 34, eff = 2.8

Looks like m=148, d=12 is an optimal choice for (m,d,k) if we want to encode 32 bit data using instruction strings with k=25. For instruction strings in S(148,12,25) we get an efficiency factor of 2.8.