Mathematical Methods for Computer Science 2 Fall 2017

Series 5

Sylvain Julmy

1

Exercice 1 is in appendix.

2

a)

$$(f(x))^2 = \frac{f(x)-1}{x}$$

$$\begin{aligned} x \cdot (f(x))^2 &= x \cdot \sum_{n=0}^{\infty} (C_0C_n + C_1C_{n-1} + \cdots + C_nC_0)x^n \\ &= \cdot \sum_{n=0}^{\infty} (C_0C_n + C_1C_{n-1} + \cdots + C_nC_0)x^{n+1} \\ &= \sum_{n=0}^{\infty} C_{n+1}x^{n+1} \\ &= f(x) - 1 \end{aligned}$$

Then

$$(f(x))^2 = \frac{f(x)-1}{x}$$

b)

$$f(x) = \frac{1-\sqrt{1-4x}}{2x}$$

From $(f(x))^2$ we get that $x \cdot (f(x))^2 - f(x) + 1 = 0$, then

$$\Delta = (-1)^2 - 4 \cdot x \cdot 1 = 1 - 4x$$

$$f(x)_{1,2} = \frac{1 \pm \sqrt{1-4x}}{2x}$$

We consider only the "-" sign :

$$\lim_{x \to 0} \frac{1+\sqrt{1-4x}}{2x} = \lim_{x \to 0} \frac{2(1-4x)^{-\frac{1}{2}}}{2} = \frac{2 \cdot (1)^{-\frac{1}{2}}}{2} = \frac{2}{2} = 1$$

Finally, we have

$$f(x) = \frac{1 - \sqrt{1 - 4x}}{2x}$$

c)

$$f(x) = \frac{1 - \sqrt{1 - 4x}}{2x}$$

$$\sqrt{1 - 4x} = [y/ - 4x](1 + y)^{\frac{1}{2}}$$
$$(1 + y)^{\frac{1}{2}} = \sum_{k=0}^{n} \binom{\frac{1}{2}}{k} y^k$$

Then we have

$$\begin{aligned}
\binom{\frac{1}{2}}{k} &= \frac{\frac{1}{2}(\frac{1}{2} - 1)(\frac{1}{2} - 2) \cdots (\frac{1}{2} - k + 1)}{k!} \\
&= \frac{1(-1)(-3) \ldots (3 - 2k)}{2^k k!} \\
&= \frac{(-1)^{k-1}}{2^k k!}(2k - 3)!! \\
&= \frac{(-1)^{k-1}}{2^k k!(2k - 1)}(2k - 1)!! \\
&= \frac{(-1)^{k-1}}{2^k k!(2k - 1)} \cdot \frac{(2k)!}{(2k)!!} \\
&= \frac{(-1)^{k-1}}{2^k k!(2k - 1)} \cdot \frac{(2k)!}{2^k k!} \\
&= \frac{(-1)^{k-1}}{4^k (2k - 1)} \cdot \frac{(2k)!}{k!k!} \\
&= \frac{(-1)^{k-1}}{4^k (2k - 1)} \binom{2n}{n}
\end{aligned}$$

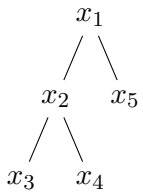
So we have

$$\begin{aligned}
\frac{1+\sqrt{1-4x}}{2x} &= \frac{1+\sum_{n=0}^{\infty}\binom{2n}{n}\frac{x^n}{2n-1}}{2x} \\
&= \frac{1+(-1)+\sum_{n=1}^{\infty}\binom{2n}{n}\frac{x^n}{2n-1}}{2x} \\
&= \sum_{n=1}^{\infty}\binom{2n}{n}\frac{x^{n-1}}{(2n-1)2} \\
&= \sum_{n=0}^{\infty}\binom{2n+2}{n+1}\frac{x^n}{2(2n-1)} \\
&= \sum_{n=0}^{\infty}\frac{(2n+2)!}{(n+1)!(n+1)!}\frac{x^n}{2(2n-1)} \\
&= \sum_{n=0}^{\infty}\frac{(2n+2)(2n+1)(2n)!x^n}{(n+1)(n+1)(n!)(n!)(2n+1)2} \\
&= \sum_{n=0}^{\infty}\binom{2n}{n}\frac{x^n}{n+1}
\end{aligned}$$

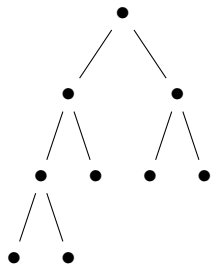
Which is the generating function for Catalan number.

3

We show a bijection between a rooted tree of $n+1$ vertices to a Dyck path. We already know that the number of Dyck path from $(0,0)$ to (n,n). So showing the bijection show that the number of rooted trees on $n+1$ vertices is equal to C_n.
To encode a rooted tree into a Dyck path, we start from the root of the tree and we traverse the tree in depth first and coming back to the root at the end. Each time we increase the depth (go to a child from its parent), we note a 0 for the Dyck path and each time we decrease the depth (go to the parent of a child), we note a 1. Clearly, the number of 1 can't be greater than then number of 0 at any point in the sequence because we can't go back to a parent without going to its children.
For example, for the following tree



We have the walk $x_1x_2x_3x_2x_4x_2x_1x_5x_1$ which is the following encoding for a Dyck path : 00101101.
We use the inverse algorithm to construct a rooted tree from a Dyck path : we start from the root of the rooted tree and use the symbol of the Dyck path. When there is a 0, we create a left child (or a right child if a left child already exist) and go to it. When there is a 1, we simply go back to the parent.
For example, the following Dyck path 0001011011001011 produce the following rooted tree :



Clearly, such a walk for a rooted tree is unique for a tree, so the encoding of the Dyck path is unique too.

4

We show a bijection between a table of handshake without crossing arms and a bracket-expression. Counting the number of bracket-expression is the same as C_n so the number of table of handshake with $2n$ personn is C_n too.
For any ways of arranging handshake between people, we can construct a bracket-expression. We start from any person across the table and turn counter-clockwise to visit every point across the table.
When visiting a point, we maintain a list of visited points. There are two cases :

- If the point is handshaked by a point which is already present in the list, we add a closed bracket to our construction.
- Otherwise, add an open bracket to our construction.

In order to encode a bracket-expression into a table of handshake, we use a table of $2n$ points and we pick a starting point. We visit each point in the counter-clockwise order. We maintain a stack (we remember the point in the order of their visit time, the closest visited point in time is the first one) of the point we visit. When visiting the points and iterating over the bracket-expressions, there are two cases :

- If the current item is an opening bracket, we just push it on the stack.
- If the current item is a closing bracket, we match it with the one we pop out of the stack.

Any bracket-expression can be converted to a table of handshake and the invert too.

5

The proof is by induction on the structure. There is two distinct base case :

- x_0 and no variables :
  1. we don't add any dot operator but add the bracket, so we have (x_0).
  2. we remove everything except the closing bracket, so we have).
  3. we replace nothing by an opening bracket.
  4. finally we obtain).

  That's corresponds to $C_1 = 1$ and $C_0 = 1$.

- (x_0x_1) : we apply the transformation and obtain :

$$(x_0x_1) \mapsto (x_0 \cdot x_1) \mapsto \cdot) \mapsto ()$$

  Which corresponds to $C_2 = 2$.

For the inductive step, we consider a bracket-variable expression e which is of one of the following form :

- One of the base case : no variable, x_i or x_ix_{i+1}.
- $e = ((x_ix_{i+1})e_k)$
- $e = (e_k(x_ix_{i+1}))$
- $e = (x_ix_{i+1})(x_{i+2}x_{i+3})$

Where i and k are any possible indices. We denote by $enc(e)$ the encoding of the bracket-variable expression e using the presented algorithm. Then, for each of the cases mentionned :

$e = ((x_ix_{i+1})e_k)$ **:**

$$((x_ix_{i+1})e_k) \mapsto ((x_i \cdot x_{i+1}) \cdot enc(e_k)) \mapsto \cdot) \cdot enc(e_k)) \mapsto ()(enc(e_k))$$

$e = (e_k(x_ix_{i+1}))$ **:**

$$(e_k(x_ix_{i+1})) \mapsto (enc(e_k) \cdot (x_i \cdot x_{i+1})) \mapsto enc(e_k) \cdot \cdot)) \mapsto enc(e_k)(())$$

$e = (e_k)(e_{k+1})$ **:**

$$(enc(e_k))(enc(e_{k+1})) \mapsto enc(e_k))enc(e_{k+1}))$$

In this case, the encoding of both expression will produce an opening bracket.
In the inductive step, each encoding of the expression produces different results so the encoding of a bracket-variable expression is uniquely defined.
We use the reverse algorithm to encode bracket expression into bracket-variable expression :

Base case : $() \mapsto \cdot) \mapsto (x_i \cdot x_{i+1}) \mapsto (x_ix_{i+1})$
The inductive step is the same as before but in the other direction, each encoding is unique.