International Journal of Progressive Sciences and Technologies (IJPSAT)
ISSN: 2509-0119 Vol. 3 No. 1 June 2016, pp.04-17

http://ijpsat.ijsht-journals.org

De-Bruijn Sequence and Application in Graph theory.

Ashish Kumar

VIT University
Vellore Tamil Nadu, India

ABSTRACT: The goal of this paper is to introduce De Bruijn graphs and discuss their various applications. We will begin by examining N.G. de Bruijn's original paper and the proof of his claim that there are exactly 22^(n-1)-n De Bruijn cycles in the binary De Bruijn graph B(2, n). In order to study this we explore the properties of Hamiltonian and Eulerian cycles that occur on De Bruijn graphs and the type of redundancy that occurs as a result. Lastly, in this paper we seek to provide some guidance into further research on De Bruijn graphs and their potential applications to other areas.

KEYWORDS: De Bruijn cycles, Bruijn graphs, De Bruijn sequence.

1. INTRODUCTION

In graph theory, an *n*-dimensional De Bruijn graph of *m* symbols is a directed graph representing overlaps between sequences of symbols. It has m^n vertices, consisting of all possible length-*n* sequences of the given symbols, the same symbol may appear multiple times in a sequence. If we have the set of *m* symbols $S := \{s_1, \ldots, s_m\}$ then the set of vertices is:

$$V = S^n = \{(s_1, \ldots, s_1, s_1), (s_1, \ldots, s_1, s_2), \ldots, (s_1, \ldots, s_1, s_m), (s_1, \ldots, s_2, s_1), \ldots, (s_m, \ldots, s_m, s_m)\}.$$

If one of the vertices can be expressed as another vertex by shifting all its symbols by one place to the left and adding a new symbol at the end of this vertex, then the latter has a directed edge to the former vertex. Thus the set of arcs (aka directed edges) is

$$E = \{((v_1, v_2, \ldots, v_n), (v_2, \ldots, v_n, s_i)) : i = 1, \ldots, m\}.$$

De Bruijn graphs are an interesting class of graphs that have applications in a variety of different areas. A De Bruijn graph is a directed graph with dn nodes labeled by n-tuples over a d-character alphabet (denoted by juxtaposition). The edges are defined to be ordered pairs of the form $((a_1 \ldots a_n), (a_2 \ldots a_n a_{n+1}))$ where a_{n+1} is any character in the alphabet. We will denote the De Bruijn graph B(d, n). In Figure 1 below we see two examples of these graphs.

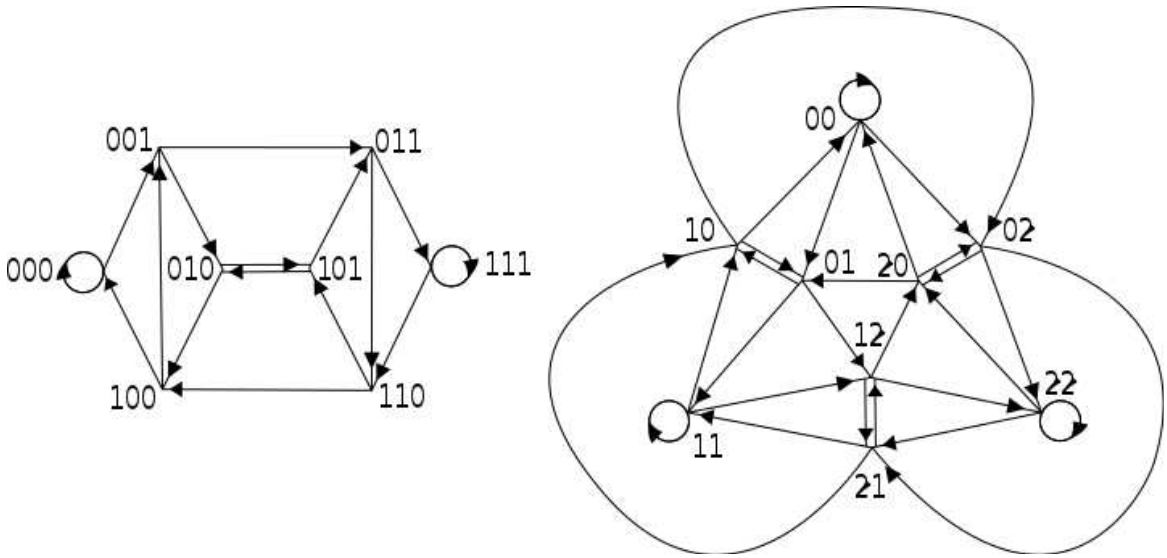


Fig. 1. Two different De Bruijn graphs

2. TERMINOLOGY AND DEFINITIONS

In order to define De Bruijn graphs, we must first begin with the definition of graph.

Definition: 1. A graph is a an ordered pair G = (V,E) where V is a set of vertices called nodes, together with a set E of edges which are pairs of nodes. Unless otherwise stated, the graphs mentioned will be directed graphs, i.e. graphs whose edges are an ordered pair of nodes.

Example: See Figure 2 below for an example of both a directed and undirected graph.

Definition: 2. A subgraph, G' = (V', E'), of a graph G = (V, E), is a graph on a subset V' of V with the property that every edge e ∈ E with endpoints in V' is also in E'.

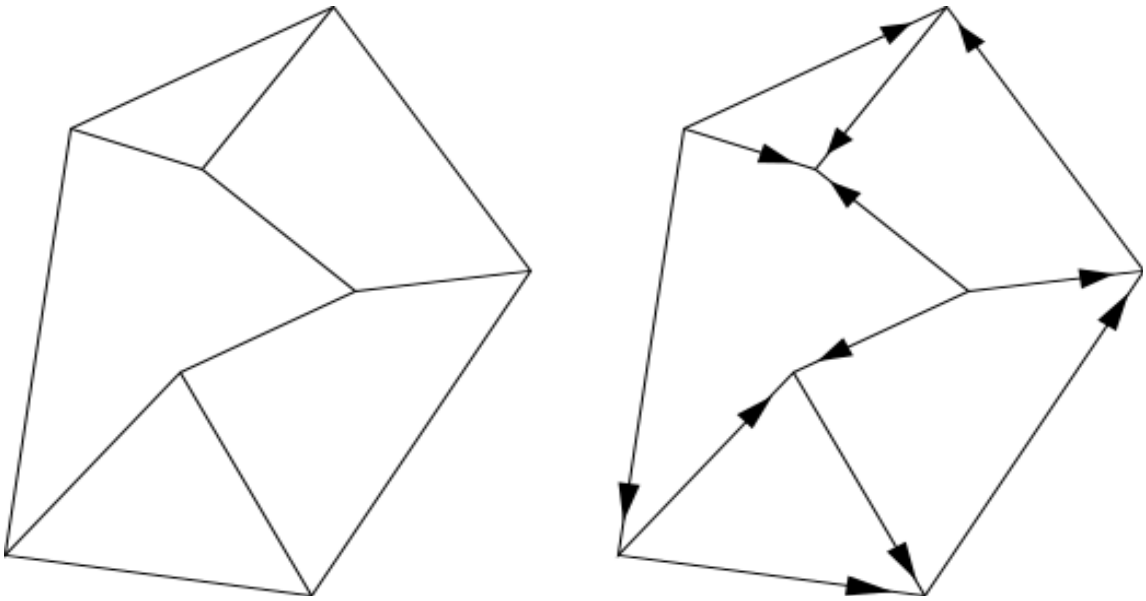


Fig. 2. An undirected graph (left) and a directed version (right).

Definition: 3. A (β1 . . . . . βn) is called a successor of node (β1 . . . . . βn) if there is an edge from (α1 . . . . . α n) to (β1 . . . . . βn). Likewise, (α1 . . . . . α n) is said to be a predecessor of (β1 . . . . . β n). We will say two nodes are adjacent nodes if there is an edge between them and that two edges are adjacent edges if they share a common node.

Definition: 4. A De Bruijn graph is a directed graph with d^n nodes labeled by n-tuples over a d-character alphabet (denoted by juxtaposition). The edges are defined to be ordered pairs of the form ((α1 . . . . . . α n), (α 2 . . . . α n α n+1)) where α n+1 is any character in the alphabet.

We will denote the De Bruijn graph B(d, n).

Example: The De Bruijn graph B(3, 4) has the node (1022) (a 4-tuple with alphabet {0, 1, 2, 3}) which has successors (0220), (0221), (0222) and (0223).

Example: The De Bruijn graph B(2, 2) is given below in Figure 3.

Definition: 5. In [EH85], Esfahanian and Hakimi discuss a modified version of a De Bruijn graph called an undirected De Bruijn graph, denoted UB(d, n). An undirected De Bruijn graph is a De Bruijn graph modified so that:

(1) All edges which are self loops are removed.
(2) If αβ is an edge in B(d, n), then the undirected edge αβ is an edge in UB(d, n).
(3) If αβ and βα are both edges in B(d, n) then there is only the single undirected edge αβ in UB(d, n).

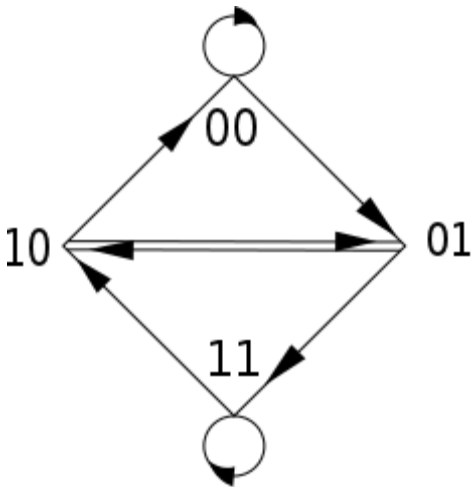


Fig. 3. The De Bruijn graph with alphabet {0, 1} whose nodes are 2-tuples.

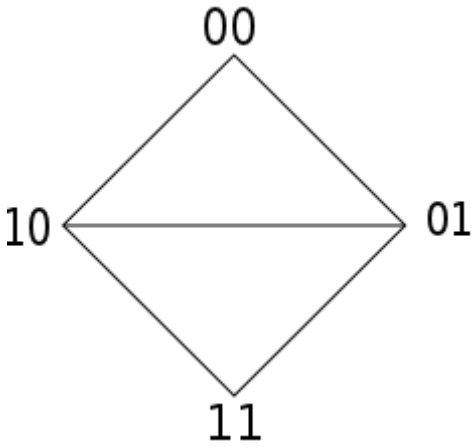


Fig. 4. The undirected De Bruijn graph UB(2, 2).

Definition: 6. The in-degree of a node in a directed graph is the number of edges that end at that node and similarly, the out-degree of a node will be the number of edges that start at that node.

Definition: 7. A path is a sequence of edges for which each edge begins where the previous edge in the sequence ended. We will denote a path.

Definition: 8. A graph is connected if for every pair of nodes, α and β ,there exists a path from α to β and a path from β to α.

Definition: 9. A subgraph of a graph G is called a component of G if:

(1) Any two nodes in the subgraph are connected.
(2) Any node in the graph which is connected to a node in the subgraph is also in the subgraph.

Example: In Figure 5, the cycle [000101] is the path 000101000, written in long form as (000)(001)(010)(101)(010)(100)(000).

Definition: 10. A Hamiltonian cycle of a graph G is a cycle of G which visits every node exactly once. An Eulerian cycle of G is a cycle of G which traverses every edge exactly once.

Example: The highlighted cycle in Figure 5 is the Hamiltonian cycle [11010001] which is described by starting at the node (110). We also see that [1111000010011010] is an Eulerian cycle on B(2, 3), starting at the node (111).

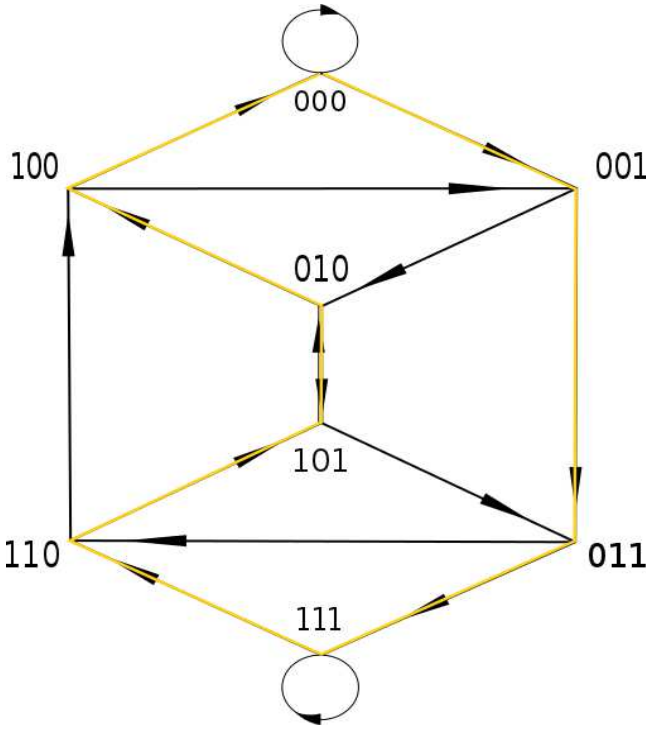


Fig: 5. The graph above is the De Bruijn graph B(2, 3) with alphabet {0,1} and nodes which are 3 tuples. The highlighted cycle is a Hamiltonian cycle on the graph.

Definition: 11. We will call the sequence of characters [α1, α 2 . . . . . αm+n] a De Bruijn sequence when [α 1 . . . . . α n+m] is a Hamiltonian cycle. Since there are d^n nodes in a De Bruijn graph, then a De Bruijn sequence will necessarily be dn characters long.

Example: Using the highlighted Hamiltonian cycle of B (2, 3) in Figure 5 above, we see that [0, 0, 0, 1, 1, 1, 0, 1] is a De Bruijn sequence.

The Doubling of a De Bruijn Graph: An additional interesting property of De Bruijn graphs is that we can expand the De Bruijn graph B(d, n) to the graph B(d, n + 1) with the following procedure.

Procedure 1: Using our standard notation as defined above we can create the De Bruijn graph B (d, n + 1) by the following steps:

1. Construct nodes in B(d, n + 1) from B(d, n) by allowing the edges of B(d, n) to correspond to nodes of B(d, n+1). Specifically, the adjacent nodes (α1 . . α n) and (α 2 . . . . . α n+1) form the edge (α 1 . . . α n) (α 2 . . . . . α n+1) in B(d, n) and correspond to node (α 1 . . . . . α n+1) in B(d, n + 1).

Example: In the doubling from B (3, 3) to B (3, 4) the edge (201)(011) in B(3, 3) gives the node (2011) in B(3, 4)

2. Construct one edge in B(d, n + 1) for each pair of adjacent edges in B(d, n). Here we will say that edges are adjacent if the ending node of one edge is the start node of the next edge.

The node (α 1 . . . α n+1) in B(d, n+1) corresponds to the edge (α 1 . . . α n) (α 2 . . . α n+1) in B(d, n), the edges leaving (α 1 . . . α n+1) are of the form (α 1 . . . α n+1)(α 2 . . . α n+2). These d edges correspond to the d adjacent edges (α 1 . . . α n)(α 2 . . . α n+1) and (α 2 . . . α n+1) (α3 . . . α n+2) in B(d, n + 1).

Example: In the doubling from B(3, 3) to B(3, 4) the edge (1222)(2220) in B(3, 4) came from the adjacent edges (122)(222) and (222)(220) in B(3, 3).

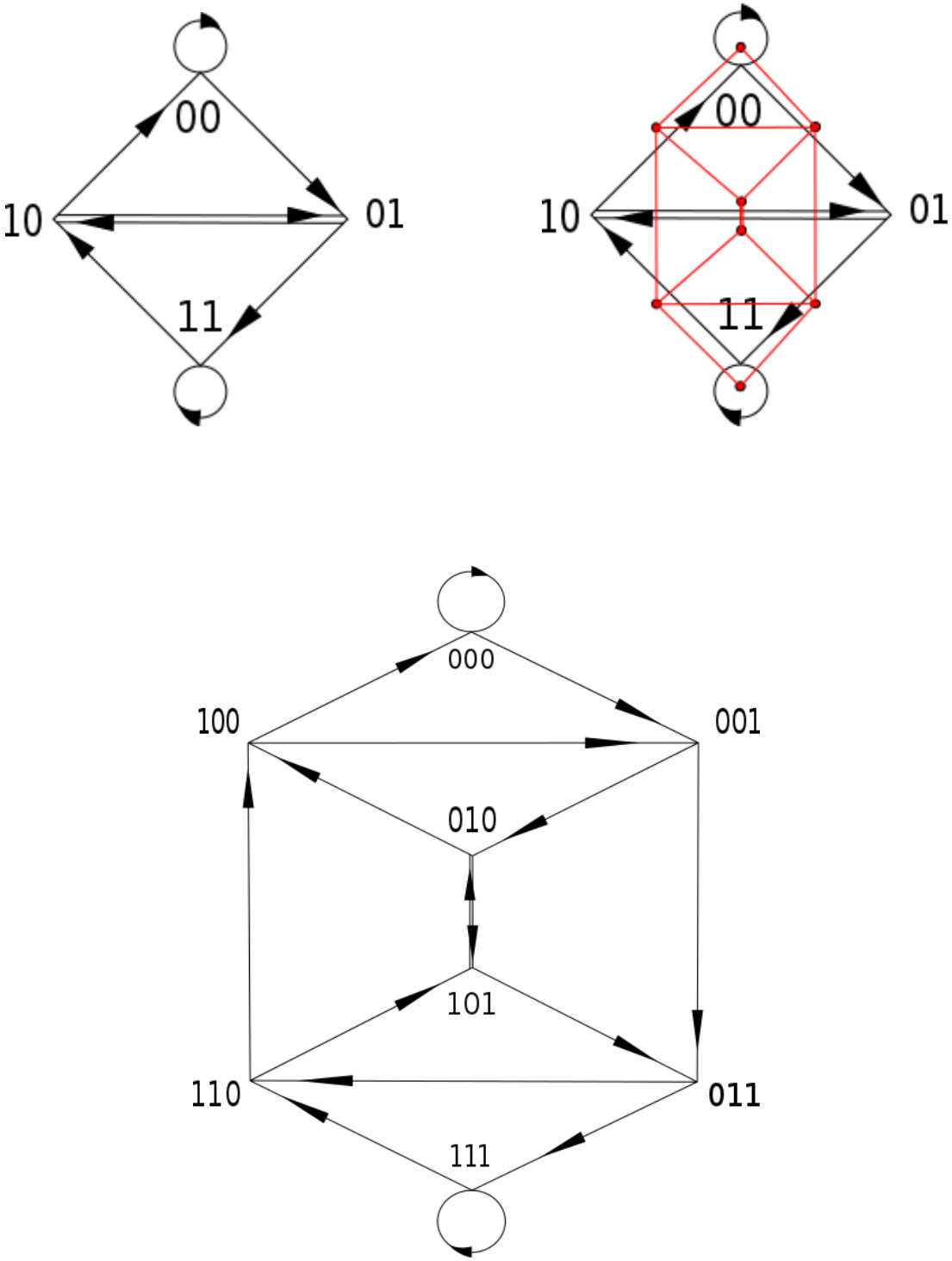


Fig. 6. The graph B(2; 2) (left) getting doubled with intermediate step (middle) to B(2; 3) (right).

3. DE BRUIJN GRAPHS

A De Bruijn graph is a directed graph representing overlaps between sequences of symbols.

1. Each vertex has exactly m incoming and m outgoing edges
2. Each n-dimensional de Bruijn graph is the line digraph of the (n-1)-dimensional de Bruijn graph
3. Each de Bruijn graph is Eulerian and Hamiltonian.

The Euler cycles and Hamiltonian cycles of these graphs are de Bruijn sequences.

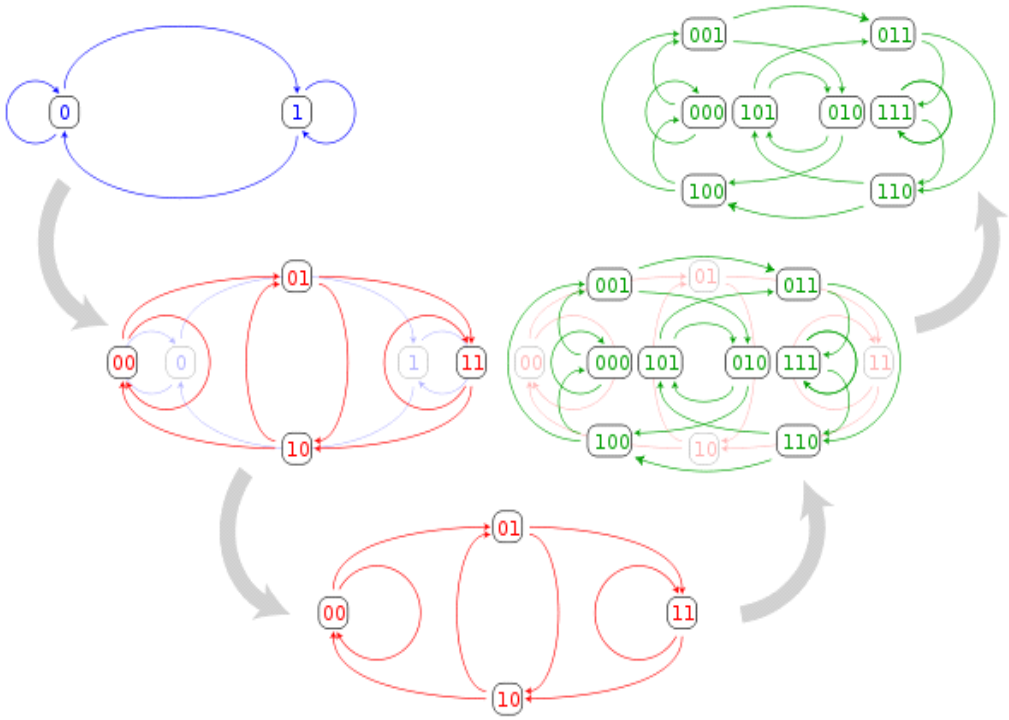


Fig. 76. F0

4. DE BRUIJN SEQUENCE

What is special about the following cyclic binary word (cycle of 0's and 1's)?

0 0

1 0

0 1

1 1

As we travel around the cycle (either clockwise or counterclockwise), we will encounter each of the 23 = 8 three-digit patterns 000, 001, 010, 011, 100, 101, 110, 111 exactly once. In a sense, this cycle is a very efficient encoding of 24 digits of information into only 8 digits. The formal name for this kind of pattern is a De Bruijn sequence.

Definition 1. A De Bruijn sequence of rank n on an alphabet of size k is a cyclic word in which each of the k^n words of length n appears exactly once as we travel around the cycle.

For example, the above cycle is a De Bruijn sequence with n = 3 and k = 2 (the alphabet is {0, 1}). De Bruijn sequences have applications to computers (electronic memory and coding theory), crime (efficiently cracking a combination lock), and of course magic tricks (as we saw in class). One thing that is not immediately obvious, however, is whether these things actually exist. We have seen one example, but how do we know that there are others, and how do we find them?

5. ALGORITHM FOR CONSTRUCTION

The De Bruijn sequences can be constructed by taking a Hamiltonian path of an n-dimensional De Bruijn graph over k symbols (or equivalently, a Eulerian cycle of a (n − 1)-dimensional De Bruijn graph).

E.g. Let n=2 A={0,1}

We have 4 possible subsequences

10, 01, 11 and 00

The following Python code calculates a De Bruijn sequence, given k and n, based on an algorithm from Frank Ruskey's Combinatorial Generation.

```
def de_bruijn(k, n):
    De Bruijn sequence for alphabet k
    and subsequences of length n.
    """
    try:
        # let's see if k can be cast to an integer;
        # if so, make our alphabet a list
        _ = int(k)
        alphabet = list(map(str, range(k)))

    except (ValueError, TypeError):
        alphabet = k
        k = len(k)

    a = [0] * k * n
    sequence = []

    def db(t, p):
        if t > n:
            if n % p == 0:
                sequence.extend(a[1:p + 1])
        else:
            a[t] = a[t - p]
            db(t + 1, p)
            for j in range(a[t - p] + 1, k):
                a[t] = j
                db(t + 1, t)
    db(1, 1)
```

```
    return "".join(alphabet[i] for i in sequence)

print(de_bruijn(2, 3))
print(de_bruijn("abcd", 2))
```

which prints

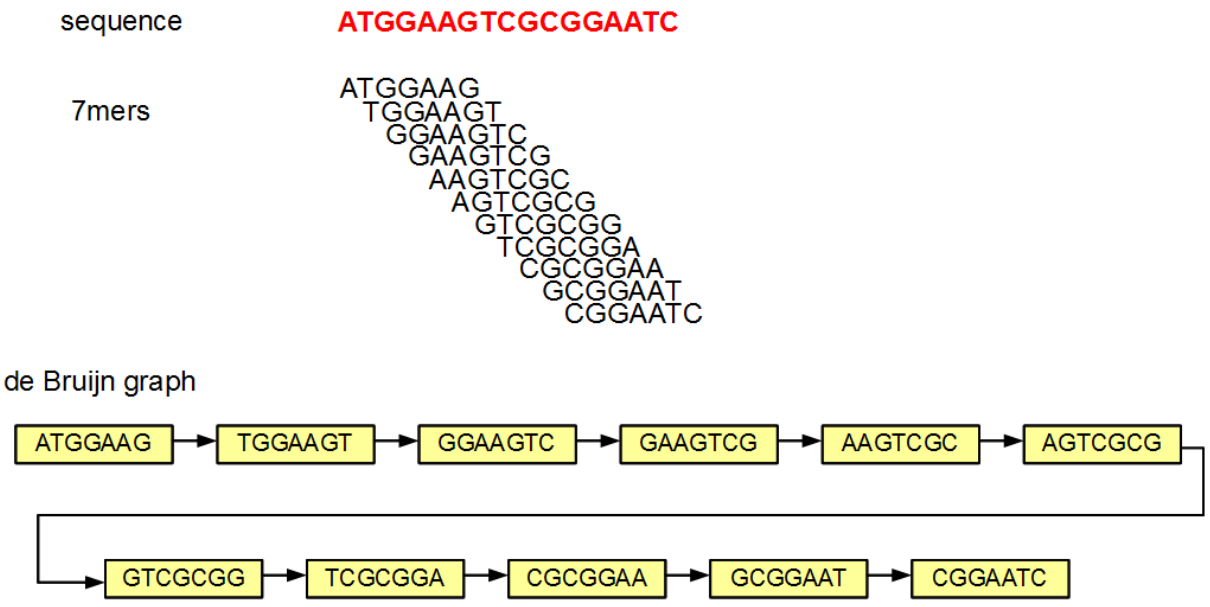
```
00010111
aabacadbbcbdccdd
```

6. APPLICATIONS

6.1. De Bruijn Graph of a Small Sequence

De Bruijn graph is an efficient way to represent a sequence in terms of its k-mer components. In an assembly problem, a de Bruijn graph is constructed from an NGS library, and then the genome is derived from the de Bruijn graph. To ease our understanding, in this section, we will check what the de Bruijn graph of an already assembled genome looks like.

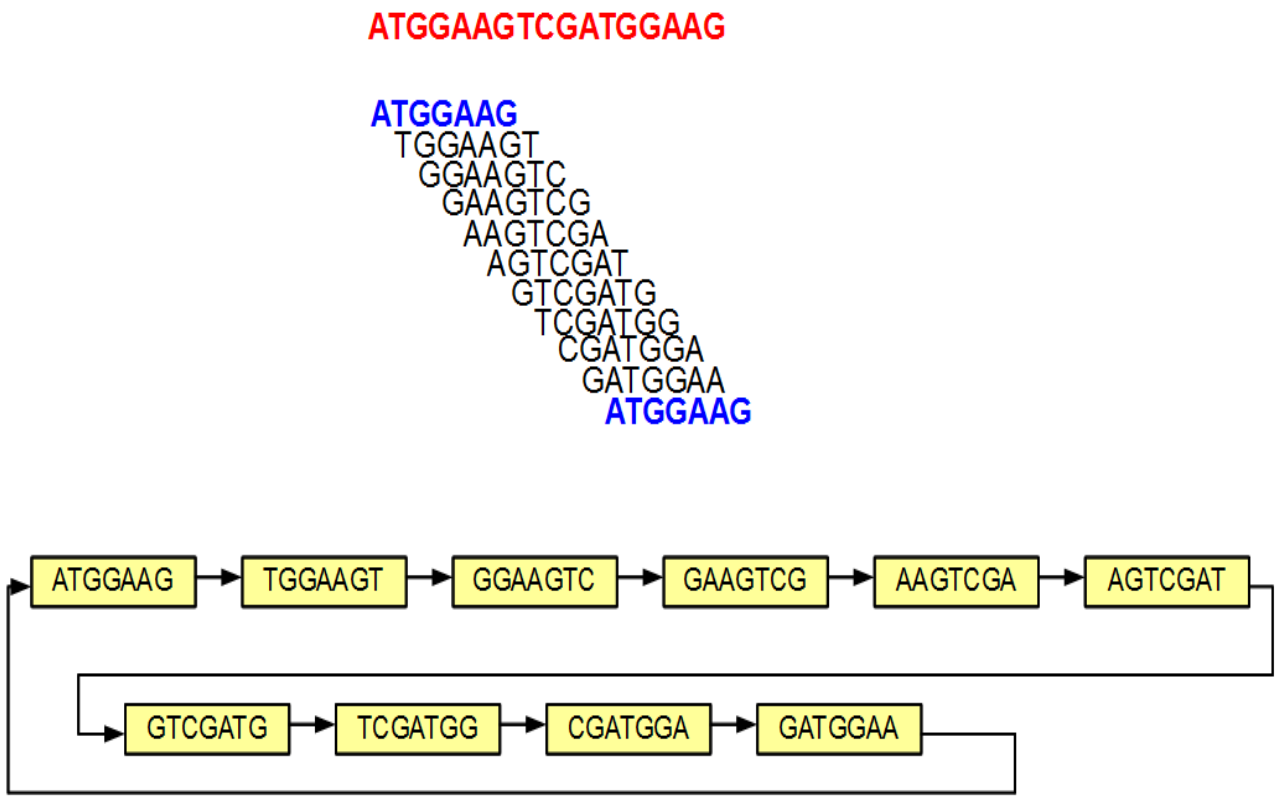
A de Bruijn graph can be constructed for any sequence, short or long. The first step is to choose a k-mer size, and split the original sequence into its k-mer components. Then a directed graph is constructed by connecting pairs of k-mers with overlaps between the first k-1 nucleotides and the last k-1 nucleotides. The direction of arrow goes from the k-mer, whose last k-1 nucleotides are overlapping, to the k-mer, whose first k-1 nucleotides are overlapping.



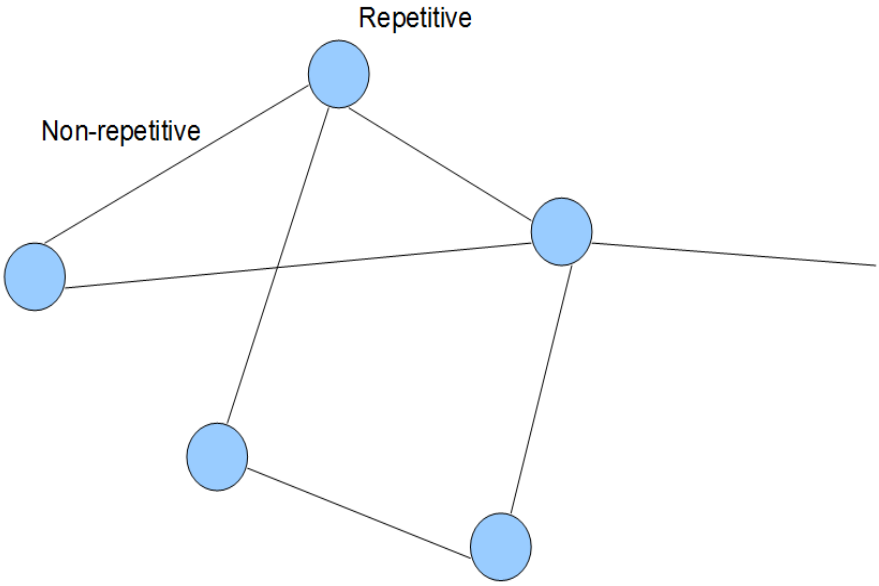
The method is best explained by an example. For sake of simplicity, we ignore the issue of double-strandedness of nucleotides here. In the above figure, we constructed the de Bruijn graph of ATGGAAGTCGCGGAATC by splitting it into all k-mers (k=7) and then by constructing a directed graph with those 7-mers as nodes. Edges are drawn between node pairs in such a way that the connected nodes have overlaps of 6 (=k-1) nucleotides. In our example, only the adjacent 7-mers from the original sequence got connected in the graph.

The above example is simple, because none of the 7-mers appeared more than once in the original sequence. In the next example, the 5'-most and 3'-most 7-mers are identical (both marked in blue). The de Bruijn graph has one less node due to

merger of those two identical nodes. Like before, the adjacent nodes in the original sequence are still connected in the graph, but in addition the graph forms a loop by connecting its two ends.



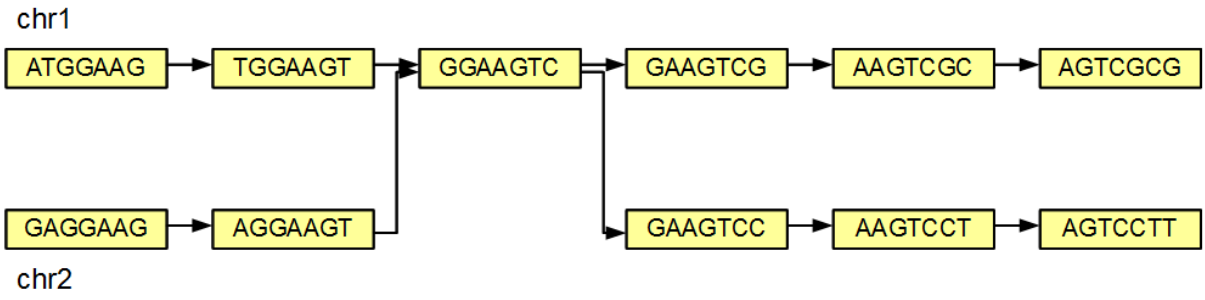
The steps shown above can be repeated to construct de Bruijn graph of any large genome with k-mer of any size. First, the genome is split into its k-mer components. Then various k-mers are connected based on whether they have k-1 common nucleotides. For prudently chosen k-mer size, topology of de Bruijn graph of a large genome will not look too different from the above figures. Vast majority of links will connect only the adjacent k-mers from the genome, whereas distant parts of the genome will get connected mostly in repetitive regions. The general topological structure is shown in the following figure, where each line and circle represents collection of many k-mer nodes. Keeping this visual picture in mind can help you understand many different applications of de Bruijn graphs, including genome assembly (Section 3), transcriptome assembly (Section 6) and metagenome assembly (Section 6).



We note that even if a genome has multiple chromosomes, the de Bruijn graph of the genome may not remain disjoint. This point is best illustrated with a simple example (see figure below).

chr1 **ATGGAAGTCGCG**

chr2 **GAGGAAGTCCTT**



6.2. De Bruijn decoding

6.2.1. Sequencing by hybridization

Few people remember that the idea of DNA sequencing using short reads dates back to the late 1980s. In fact, the very first short-read sequencing approach, sequencing by hybridization23, 24, 25, aimed to achieve genome assembly in this fashion. Sequencing by hybridization, proposed in 1988, is based on building a microarray containing every possible oligonucleotide of length k. After hybridization of such an array with an unknown genome, one would get information about all k-mers present in the genome. De Bruijn graphs were first brought to bioinformatics in 1989 as a method to assemble k-mers generated by sequencing by hybridization26; this method is very similar to the key algorithmic step of today's short-read assemblers.

DNA arrays ultimately failed to realize the dream (DNA sequencing) that motivated their inventors because the fidelity of DNA hybridization with the array turned out to be too low and the value of k was too small. Yet the failure of DNA arrays was short-lived. Although their original goal (DNA sequencing) was still out of reach, two new unexpected applications emerged: measuring gene expression and analyzing genetic variations. Today, these applications have become so ubiquitous that most people have forgotten that the original goal of the inventors of DNA arrays was to sequence the human genome.

6.2.2. De Bruijn Graph on a Chess Board:

A directed De Bruijn Graph of B(2, 6) sequences with Little-Endian Rank-File Mapping board coordinates (a1 = 0, b1 = 1, h8 = 63). For topology reasons, almost each node (except a1 and h8) of the graph is deconcentrated and appears twice in the form of two reversed binary trees. The leaf outputs join the respective reversed tree. Between c6 and f3 is a direct cycle, since 42 is 2*21 and 21 is (2*42 + 1) % 64, with both six-bit pattern reversed - 010101 (21) versus 101010 (42). The challenge is to traverse the graph in any way to visit each of the 64 nodes aka squares exactly once.

C++ code for chess-boarding problem:

Program Output:

```
const unsigned int magictable[64] =
{
   0,  1,  2, 53,  3,  7, 54, 27,
   4, 38, 41,  8, 34, 55, 48, 28,
  62,  5, 39, 46, 44, 42, 22,  9,
  24, 35, 59, 56, 49, 18, 29, 11,
  63, 52,  6, 26, 37, 40, 33, 47,
  61, 45, 43, 21, 23, 58, 17, 10,
  51, 25, 36, 32, 60, 20, 57, 16,
  50, 31, 19, 15, 30, 14, 13, 12,
};

unsigned int bitScanForward (BitBoard b) {
   return magictable[((b&-b)*magic) >> 58];
}
```

Source code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef unsigned __int64 U64; // long

class CGenBitScan
{
public:
  //========================================
  // constructor immediately starts the search
  //========================================
  CGenBitScan(int match4nth) {
    clock_t start, stop;
    m_dBCount = 0;
    m_Match4nth  = match4nth;
    initPow2();
    start = clock();
```

```
    m_Lock = pow2[32]; // optimization to exclude 32, see remarks
    try {findDeBruijn(0, 64-6, 0, 6);} catch(int){}
    stop = clock();
    printf("\n%.3f Seconds for %d De Bruijn sequences found\n", (float)(stop - start) / CLOCKS_PER_SEC, m_dBCount);
  }

private:
  U64 pow2[64];    // single bits
  U64 m_Lock;      // locks each bit used
  int m_dBCount;   // counter
  int m_Match4nth; // to match

  //========================================
  // on the fly initialization of pow2
  //========================================
  void initPow2()  {
    pow2[0] = 1;
    for (int i=1; i < 64; i++)
      pow2[i] = 2*pow2[i-1];
  }

  //========================================
  // print the bitscan routine and throw
  //========================================
  void bitScanRoutineFound(U64 deBruijn) {
    int index[64], i;
    for (i=0; i<64; i++) // init magic array
      index[ (deBruijn<<i) >> (64-6) ] = i;
    printf("\nconst U64 magic = 0x%08x%08x; // the %d.\n\n",
          (int)(deBruijn>>32), (int)(deBruijn), m_dBCount);
    printf("const unsigned int magictable[64] =\n{\n");
    for (i=0; i<64; i++) {
      if ( (i & 7) == 0 ) printf("\n  ");
      printf(" %2d,", index[i]);
    }
    printf("\n};\n\nunsigned int bitScanForward (U64 b) {\n");
    printf("    return magictable[((b&-b)*magic) >> 58];\n}\n");
    // throw 0; // unwind the stack until catched
  }

  //========================================
  // recursive search
  //========================================
  void findDeBruijn(U64 seq, int depth, int vtx, int nz) {
    if ( (m_Lock & pow2[vtx]) == 0 && nz <= 32 ) { // only if vertex is not locked
      if ( depth == 0 ) { // depth zero, De Bruijn sequence found, see remarks
        if ( ++m_dBCount == m_Match4nth )
          bitScanRoutineFound(seq);
      } else {
        m_Lock ^= pow2[vtx]; // set bit, lock the vertex to don't appear multiple
        if ( vtx == 31 && depth > 2 ) { // optimization, see remarks
          findDeBruijn( seq | pow2[depth-1], depth-2, 62, nz+1);
        } else {
          findDeBruijn( seq, depth-1, (2*vtx)&63, nz+1); // even successor
          findDeBruijn( seq | pow2[depth-1], depth-1, (2*vtx+1)&63, nz); // odd successor
        }
        m_Lock ^= pow2[vtx]; // reset bit, unlock
      }
```

```
      }
    }
  };

  int main(int argc, char* argv[])
  {
    if (argc < 2)
      printf("usage: genBitScan 1 .. %d\n", 1<<26);
    else
      CGenBitScan(atoi(argv[1]));
    return 0;
  }
```

REFERENCES

[1] [F.S. Annexstein, Generating de Bruijn sequences: An efficient implementation, IEEE Transactions on Computers 46 (2) (1997) 198–200.

[2] Nicolaas Gover de Bruijn, A combinatorial problem, Koninklijke Nederlandse Akademie v. Wetenschappen 49 (1946) 758–764, Indagationes Mathematicae 8 (1946) 461–467.

[3] Jean Berstel, Dominique Perrin, The origins of combinatorics on words, European Journal of Combinatorics 28 (2007) 996–1022.

[4] Taejoo Chang, Bongjoo Park, Yun Hee Kim, Iickho Song, An efficient Implementation of the D-homomorphism for generation of de Bruijn Sequences, IEEE Transactions on Information Theory 45 (4) (1999)

[5] 1280–1283.

[6] Joshua Cooper, Christine Heitsch, The discrepancy of the lex-least de Bruijn sequence, Discrete Mathematics 310 (2010) 1152–1159.

[7] Harold Fredricksen, A survey of full length nonlinear shift register Cycle algorithms, SIAM Review 24 (2) (1982) 195–221.

[8] Abraham Flaxman, Aram Harrow, Gregory Sorkin, Strings with maximally Many distinct subsequences and substrings, Electronic Journal Of Combinatorics 11 (2004), #R8.

[9] M. Fleury, Deux problems de géométrie de situation, Journal de Mathématiques Élémentaires (1883) 257–261.

[10] I.J. Good, Normal recurring decimals, Journal of the London Mathematical Society 21 (3) (1946) 167–169.

[11] Abraham Lempel, On a homomorphism of the de Bruijn graph and its applications to the design of feedback shift registers, IEEE Transactions on Computers C-19 (12) (1970) 1204–1209.

[12] E.B. Leach, Regular sequences and frequency distributions, Proceedings of American Mathematical Society 11 (1960) 566–574.

[13] Camille Flye Sainte-Marie, Question 48, L'intermédiaire des mathématiciens