Electronic Colloquium on Computational Complexity, Report No. 158 (2006)

Representing Boolean OR Function by Quadratic Polynomials Modulo 6

Gyula Győr

Eötvös University, Budapest
Address:Pázmány P. stny. 1/C, H-1117, Budapest, HUNGARY
E-mail:angelofd.gy@gmail.com, angelofd@inf.elte.hu

Abstract

We give an answer to the question of Barrington, Beigel and Rudich, asked in 1992, concerning the largest n such that the OR function of n variable can be weakly represented by a quadratic polynomial modulo 6. More specially, we show that no 11-variable quadratic polynomial exists that is congruent to zero modulo 6 in the all-0 input and non-zero modulo 6 anywhere else on the $\{0,1\}$ input.

1 Introduction

In this paper we show a method to decide one of the open questions of Barrington, Beigel and Rudich[BBR92] asked in 1992. The polynomial P weakly represents the N variable OR function over mod m if $P(0,0,\cdots,0) \equiv 0$ (MOD m) and $\forall x \in \{0,1\}^N, x \neq 0 : P(x) \not\equiv 0$ (MOD m). It is known that if the polynomial P weakly represents the N variable Boolean OR function over a p prime modulus then its degree is at least $\left\lceil \frac{N}{p-1} \right\rceil$ [Smo87]. On the other hand Barrington, Beigel and Rudich proved that there exists degree $O(\sqrt[r]{N})$ polynomial that weakly represent the N variable OR function over mod m, where r is the number of distinct prime divisors of m. So polynomials over mod 6 are much powerful than polynomials over a prime power. This construction uses only symmetric polynomials and the bound is matching for this type of polynomials. Barrington, Beigel and Rudich asked that what is the largest N such that the N variable Boolean OR function can be weakly represented by quadratic polynomial over mod 6 ? It is known [BBR92] for symmetric polynomials the answer is 8, but it is not a hard task to construct polynomials showing that $N \geq 10$. As a consequence of their theorem, Tardos and Barrington[TB95] show that $N \leq 18$.

ISSN 1433-8092

We shall see later, that P contains no linear monomials. Deciding this question by a brute force algorithm is hopeless, because we need to check $6^{\binom{11}{2}+11} = 6^{66}$ polynomial if it represents well. To check one polynomial we need $2^{11} = 2048$ operation. So it will take $1.48 \cdot 10^{38}$ years to complete with 10^9 operation/sec speed.

Our algorithm does exhaustive search in approximately 800 2.2GHz-CPU hours.

Ramsey's theorem shows that each graph with 2^n vertices has either a clique or an independent set of $\frac{n}{2}$ vertices. Erdős showed with a probabilistic method[Erd47] that there exist graph with 2^n vertices with no clique and independent set of size $2 \cdot n$. Constructive bounds to date are far from the upper bound. The connection between these problems was shown by Grolmusz [Gro00]. He used polynomial representations of the OR function to construct Ramsey graphs and showed that lower degree representation leads to better Ramsey graph. Because the best lower bound for these type of representations is only $O(\log n)$ it may be possible to construct better graphs construction in this way.

2 The testing algorithm

Definition 1 ([BBR92])
We say, that the n-variable polynomial P over the ring of integers modulo m ***weakly represents*** $f : \{0,1\}^n \to \{0,1\}$ *if and only if*

$$\forall x, y \in \{0,1\}^n : (P(x) \neq P(y)) \textit{ implies } (f(x) \neq f(y))$$

Definition 2
Let

$$M^{*(n,m)} := \left\{ M \in Z_m^{n \times n} : \ M \textit{ is lower triangular matrix} \right\}$$

Let $P^{*(n,m)}$ denote the set of n-variable polynomials over the ring of integers modulo m that does not contain linear and constant monomials.

Instead of working with polynomials usually we work with matrices. The function L makes clear the connection between the polynomials and their matrix representation.

Definition 3
Let $L : M^{*(n,m)} \to P^{*(n,m)}$ be defined as

$$L(M) := \sum_{i=1}^{n} \sum_{j=1}^{n} M_{i,j} \cdot x_i \cdot x_j \tag{1}$$

It is obvious that L is a one-to-one mapping.

For any polynomial that weakly represents the n variable OR, there exists a polynomial that weakly represents the n variable OR and does not contain linear and constant monomials.
The proof is immediate from the fact that $\left|OR^{(-1)}(0)\right| = 1$ and $x_i = x_i^2$ on the $\{0,1\}$ input.

Definition 4
Let $H^{*(n,m)} \subseteq M^{*(n,m)}$ be defined as

$$H^{*(n,m)} := \left\{ h \in M^{*(n,m)} : \ L(h) \textit{ weakly represents the n variable OR function.} \right\} \quad (2)$$

Let us call the elements of $H^{*(n,m)}$ well representing.

Example 5

$$H^{*(1,6)} = \{[1], [2], \cdots, [5]\}$$

We would like to build all the well representing $(n+1) \times (n+1)$ matrices from smaller parts.This is motivated we for the next definition.

Definition 6
Let $C : Z_m^{n \times n} \to Z_m^{n-1}$

$$C(M)_i := M_{i+1,1}$$

Let $Q : Z_m^{n \times n} \to Z_m^{(n-1)\times(n-1)}$

$$Q(M)_{i,j} := M_{i+1,j+1}$$

So $C(M)$ is the first column of M without the first item, and $Q(M)$ is the M matrix truncated the first row and column.

Definition 7 ([TB95]) *We call a Boolean function g a strict restriction of the Boolean function f if g is what we get by setting some variables of f to 0, while setting some of variables to be equal. The number of variables of g is therefore the number of equivalent classes of then nonzero variable of f. We call a polynomial Q a strict restriction of the polynomial P if we can obtain Q from P via this kind of restriction.*

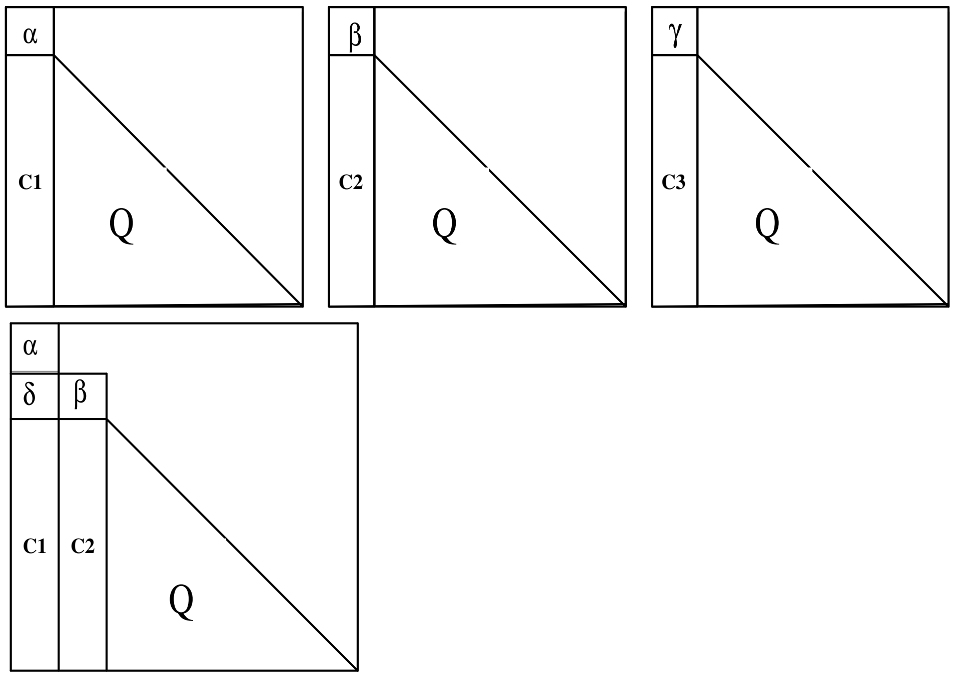

Figure 1: $\delta = (\gamma - \beta - \alpha)$MOD m

The following main lemma shows a method to build all $(n+1) \times (n+1)$ well representing matrices from the $n \times n$ ones.
The main idea is to split the first variable.

We will define the function Φ that maps three well representing $n \times n$ matrices to one $(n+1) \times (n+1)$ matrix as Figure 1 shows. For formal definition see Lemma 8.

Lemma 8

$$H^{*(n+1,m)} = \Big\{\Phi(H_1, H_2, H_3) : \exists H_1, H_2, H_3 \in H^{*(n,m)} : C(H_1) + C(H_2) \equiv$$

$$\equiv C(H_3) \ (\text{MOD } m) \ \ Q(H_1) = Q(H_2) = Q(H_3)\}$$

where $\Phi : \left(H^{*(n,m)}\right)^3 \to M^{*(n+1,m)}$
$(\Phi(H_1, H_2, H_3))_{1,1} = (H_1)_{1,1}$
$(\Phi(H_1, H_2, H_3))_{1,2} = ((H_3)_{1,1} - (H_2)_{1,1} - (H_1)_{1,1}) \ (\text{MOD } m)$
$\forall i \in \{3, 4, \cdots, n+1\} : (\Phi(H_1, H_2, H_3))_{1,i} = (H_1)_{1,i-1}$
$\forall i, j \in \{2, 3, \cdots, n+1\} : (\Phi(H_1, H_2, H_3))_{i,j} = (H_2)_{i-1,j-1}$

Proof: Let H' denote the set on the RHS of the equation.
Case: $\subseteq$
Let $h \in H^{*(n+1,m)}$.We will show, that $h \in H'$.
Let

$$H_1 := L^{-1}\left(L(h)|_{x_2=0}\right),\ H_2 := L^{-1}\left(L(h)|_{x_1=0}\right),\ H_3 := L^{-1}\left(L(h)|_{x_1=x_2}\right)$$

Because any strict restriction of OR is also an OR function, $H_1, H_2, H_3 \in H^{*(n,m)}$, and these satisfy $C(H_1) + C(H_2) \equiv C(H_3)$ (MOD m) $Q(H_1) = Q(H_2) = Q(H_3)$.
Case: $\supseteq$
Let $h = \Phi(H_1, H_2, H_3) \in H'$.
It is enough to show, that L(h) represents well.
If $x_1 = 0$ then L(h) behaves exactly like $L(H_2)$.
If $x_2 = 0$ then L(h) behaves exactly like $L(H_1)$.
If $x_1 = x_2$ then L(h) behaves exactly like $L(H_3)$.
Because of $L(H_1), L(H_2), L(H_3)$ represent well $L(h)$ must do the same.
□

Algorithm 1 *Basic*

The algorithm constructs $H^{*(n,6)}$ for $n = \{2, 3, \cdots, 11\}$ by induction. It decides about all triplets in $\left(H^{*(n-1,6)}\right)^3$ if they satisfy the conditions in Lemma 8. In this case we add $\Phi(h1, h2, h3)$ to `Hnew`*. At the end of each level* `Hnew` *equals to $H^{*(n,6)}$.*

```
H:={[1],[2],[3],[4],[5]}
for level:=2 to 11 do
begin
  Hnew:=empty;
  forall h1,h2,h3 in H do
  begin
    if (Q(h1)=Q(h2)=Q(h3)) and
        (C(h1)+C(h2)-C(h3) mod 6=0) then
    begin
      Hnew:=Hnew+[PHI(h1,h2,h3)];
    end;
  end;
  H:=Hnew;
end;
if H=empty then
  print('There is no quadratic polynomial representing the 11 variables OR.')
```

```
else
  print('There is quadratic polynomial representing the 11 variables OR.');
```

Remark 9
It is easy to show that if $H_1, H_2 \in H^{*(n,m)}$ for some n and $Q(H_1) \neq Q(H_2)$ then $\forall H_1^*$ descendant of H_1 and $\forall H_2^*$ descendant of $H_2 : Q(H_1^*) \neq Q(H_2^*)$
So function Q partitionates H and we should search separately in these classes.
This is an opportunity to segment the problem to disjoint subproblems and solve these subproblems with reduced computational resources.

3 Permutation filtering

We say that two polynomials are equivalent if and only if there exists a permutation of their variables that makes them equal. Clearly, this relation partitionates the polynomials.

Lemma 10 *Let polynomials P_1 and P_2 be in the same class. Then P_1 represents the OR function if and only if P_2 does.*

Proof:
Since OR is a symmetric function the proof is obvious. □

The previous algorithm does not filter such permutations.

Our goal is check one representing item in each class. This is a hard task, but we can define the following property of our matrices, that help us to check only a small number of polynomials in each class.

$\tau(M)$ is true exactly when ignoring the first column the following conditions are satisfied.

- The diagonal items are nondecreasing.
- When the i^{th} and $(i+1)^{th}$ elements are equal in the diagonal then $(M_{n,i}, M_{n-1,i}, \cdots, M_{i+2,i})$ is lexicographically not greater than $(M_{n,i+1}, M_{n-1,i+1}, \cdots, M_{i+2,i+1})$.

Definition 11
Let $M \in M^{*(n,m)}$

$$\tau(M) := \bigwedge_{i=2}^{n-1} (M_{i,i} \leq M_{i+1,i+1} \wedge ((M_{i,i} = M_{i+1,i+1}) \textit{ implies } \exists j \in \{i+1, \cdots, n\} :$$
$$\forall k \in \{j+1, \cdots, n\} : M_{k,i} = M_{k,i+1} \wedge (M_{j,i} < M_{j,i+1} \vee j = i+1)))$$

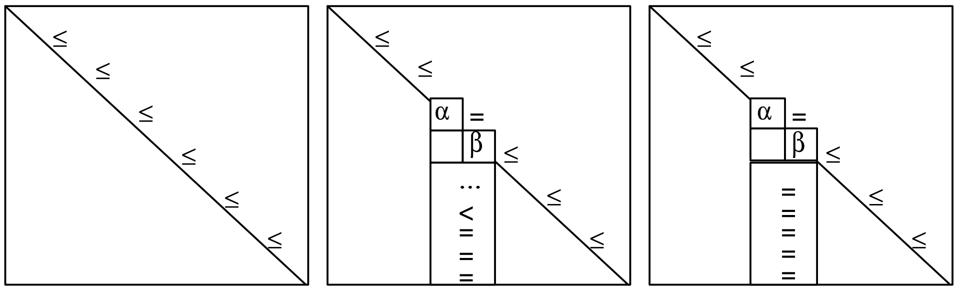

Figure 2: Matrices satisfying τ ($\beta = \alpha$)

Remark 12 *There is no condition on the first variable.*

Lemma 13
For all polynomial classes there is at least one polynomial whose corresponding matrix satisfies τ.

Proof: We will show that there exists at least one permutation of variables for any polynomial in $P^{*(n,m)}$ such that its corresponding matrix satisfies τ.
Let $P \in P^{*(n,m)}$ and $M := L^{-1}(P)$. Let $N \in Z_m^{n\times n}, N_{i,j} := M_{\min\{i,j\},\max\{i,j\}}$. So N behaves like M in the lower triangular and behaves like M^T in the upper. Where M^T denotes the transpose of M.
In this section the representing items of the MOD m ring are $\{0,1,\cdots,m-1\}$ and we use the natural ordering on these numbers. Let $\pi : \{1,2,\cdots,n\} \to \{1,2,\cdots,n\}$. We construct the inverse of the promised permutation.
Let π_n be the index of the one of the greatest diagonal item.
We assume that $\pi_{s+1},\cdots,\pi_n$ are already chosen. We chose now π_s.
Let relation R be defined on the set of not-chosen variables. More precisely we call x_i to be not chosen if $\pi^{(-1)}(j) = \emptyset$.
Let $x_i R x_j$ (where i,j are not chosen) if and only if

$$(N_{i,i} < N_{j,j})$$

or

$$(N_{i,i} = N_{j,j}$$

and

$$\exists t \in \{s+1,s+2,\cdots,n\}\,\forall t' \in \{t+1,t+2,\cdots,n\} : N_{i,\pi_{t'}} = N_{j,\pi_{t'}} \wedge N_{i,\pi_t} < N_{j,\pi_t}\Big)$$

Relation R is antisymmetric and transitive. So this is a partial order on the set of not chosen variables.

Let π_s the index of one of the maximal variables by R.
Now π is a one-to-one mapping and $\tau\left(L^{-1}(P(x_{\pi_1^{-1}}, x_{\pi_2^{-1}}, \cdots, x_{\pi_n^{-1}})\right)$ is true. □

Definition 14

$$H^{**(n,m)} := \left\{h \in H^{*(n,m)} : \tau(h)\right\}$$

The following lemma corresponds to Lemma 8, but these sets contain only matrices satisfying τ.

Lemma 15

$$H^{**(n+1,m)} = \left\{\Phi(H_1, H_2, H_3) : \tau(\Phi(H_1, H_2, H_3)) \wedge \exists H_1, H_2, H_3 \in H^{**(n,m)} : C(H_1) + C(H_2) \equiv\right.$$

$$\equiv C(H_3) \ (\text{MOD} m) \ \ Q(H_1) = Q(H_2) = Q(H_3)\}$$

Case: $\subseteq$
Obvious. According to the proof of Lemma 8, Remark 12.

Case: $\supseteq$
Obvious. According to Lemma 8 and definition of H^{**}.

Remark 16
$\tau(\Phi(H_1, H_2, H_3)) \Leftrightarrow \tau_{ext}(H_2)$ *where* τ_{ext} *is* τ *extended to the first column.*
$\tau(\Phi(H_1, H_2, H_3))$ *depends only on the first and second column of* H_2.

Algorithm 2 *Final*

```
H:={[1],[2],[3],[4],[5]}
for level:=2 to 11 do
begin
  Hnew:=empty;
  forall H' Q-class of H do
  begin
    forall h2 in H' do
    begin
      if isGoodPermutation(h2) then
      begin
```

```
        forall h1,h3 in H' do
        begin
          if (C(h1)+C(h2)-C(h3) mod 6=0) then
          begin
            Hnew:=Hnew+[PHI(h1,h2,h3)];
          end;
        end;
      end;
    end;
  end;
  H:=Hnew;
end;
if H=empty then
  print('There is no quadratic polynomial representing the 11 variables OR.')
else
  print('There is quadratic polynomial representing the 11 variables OR.');
```

We are enclosing the code implemented in C++ in the Appendix.

4 The Result

We have used the above algorithm with minor technical modifications and implemented in C++. It uses STL data structures (sets) to speed up the search.
It did not found any 11 variable polynomials, that means the largest n such that the OR function of n variable can be weakly represented by a quadratic polynomial modulo 6 is 10.
The computation took approximately 800 2.2GHz-CPU hours on 6 computer.

5 Acknowledgments

The author wishes to thank Vince Grolmusz for his help in this work. The author also acknowledge the partial support of OTKA T046236 grant.

References

[BBR92] David A. Mix Barrington, Richard Beigel, and Steven Rudich. Representing Boolean functions as polynomials modulo composite numbers. In *STOC*

'92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing, pages 455–461, New York, NY, USA, 1992. ACM Press.

[Erd47] P. Erdős. Some remarks on the theory of graph. In *Bull. Amer. Math. Soc.*, pages 292–294, 1947.

[Gro00] Vince Grolmusz. Superpolynomial size set-systems with restricted intersections mod 6 and explicit Ramsey graphs. *Combinatorica*, 20(1):71–86, 2000.

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *STOC '87: Proceedings of the Nineteenth Annual ACM Conference on Theory of Computing*, pages 77–82, New York, NY, USA, 1987. ACM Press.

[TB95] Gabor Tardos and David A. Mix Barrington. A lower bound on the mod 6 degree of the OR function. In *Israel Symposium on Theory of Computing Systems*, pages 52–56, 1995.

6 Appendix

6.1 item.h

```
#ifndef ITEM_H
#define ITEM_H

class item
{
  public:
    int n;
    unsigned char *v;
    unsigned char *parent;
    item(int in);
    item(const item& in);
    ~item();
    bool __fastcall operator <(const item& rhs) const;
    bool __fastcall operator ==(const item& rhs) const;
    item& __fastcall operator =(const item& rhs);
    item __fastcall operator +(const item& rhs) const;
    item combinate(const item& item1,const item& item2) const;
    bool SameParent(const item& rhs) const;
    bool isGoodPermutation() const;
};
#endif
```

6.2 item.cpp

```
#include "item.h"
item::item(int in)
{
  n=in;
  v=new unsigned char[++in];
  int ca=n+1;
  while (ca--)
    v[ca]=0;
  parent=new unsigned char[(n*(n-1))/2+1];
  ca=(n*(n-1))/2+1;
  while (ca--)
    parent[ca]=0;
```

```
}

item::~item()
{
  delete[] v;
  delete[] parent;
}

bool __fastcall item::operator <(const item& rhs) const
{
  if (rhs.n != n)
    throw (*this);

  int ca=(n*(n-1))/2;
  while ((ca) && (parent[ca-1]==rhs.parent[ca-1]))
    ca--;
  if (!(ca))
  {
    ca=n;
    while ((ca) && (v[ca-1]==rhs.v[ca-1]))
      ca--;
    return ((ca) && (v[ca-1]<rhs.v[ca-1]));
  } else
    return ((ca) && (parent[ca-1]<rhs.parent[ca-1]));
}

bool __fastcall item::operator ==(const item& rhs) const
{
  return ((!((*this)<rhs)) && (!(rhs<(*this))));
}

item::item(const item& in)
{
  n=in.n;
  v=new unsigned char[n+1];
  int ca=n+1;
  while (ca--)
    v[ca]=in.v[ca];
  parent=new unsigned char[(n*(n-1))/2+1];
  ca=(n*(n-1))/2+1;
  while (ca--)
    parent[ca]=in.parent[ca];
}

bool item::SameParent(const item& rhs) const
{
  int ca=(n*(n-1))/2;
  while ((ca) && (parent[ca-1]==rhs.parent[ca-1]))
    ca--;
  return (!(ca));
}

item& __fastcall item::operator =(const item& rhs)
{
  if ((&rhs != this) && (n==rhs.n))
  {
    int ca=n;
    while (ca--)
      v[ca]=rhs.v[ca];
    ca=(n*(n-1))/2+1;
    while (ca--)
      parent[ca]=rhs.parent[ca];
  }
  return (*this);
}

item __fastcall item::operator +(const item& rhs) const
{
  item ret(*this);
  if (n==rhs.n)
  {
    int ca=n;
    while (ca--)
      ret.v[ca]=(ret.v[ca]+rhs.v[ca]) % 6;
    ca=(n*(n-1))/2;
    while (ca--)
      ret.parent[ca]=rhs.parent[ca];
  }
  return ret;
}
```

```
item item::combinate(const item& item1,const item& item2) const
{
  item ret(n+1);
  if ((n==item1.n) && (n==item2.n))
  {
    int ca=n;
    while ((ca--)>1)
      ret.v[ca+1]=item1.v[ca];
    ret.v[0]=item1.v[0];
    ret.v[1]=(v[0]+(6-item2.v[0])+(6-item1.v[0]))%6;
    ca=ret.n*(ret.n-1)/2;
    while (ca--)
      if (ca<n)
        ret.parent[ca]=item2.v[ca];
      else
        ret.parent[ca]=item2.parent[ca-n];
  }
  return ret;
}

bool item::isGoodPermutation() const
{
  bool ret=0;
  if (n<2)
    ret=1;
  else if (parent[0]==v[0])
  {
    int i=n-2;
    while ((i>=1) && (parent[i]==v[i+1]))
      i--;
    ret=((i<1) || (parent[i]>v[i+1]));
  }
  else
    ret=(parent[0]>v[0]);
  return ret;
}
```

6.3 main.cpp

```
#include <cstdlib>
#include <iostream>
#include <set>
#include <algorithm>
#include <fstream>
#include "item.h"
#define MAXFS (200000000)
#define INITMAXLEVEL (4)
#define TOTALMAXLEVEL (11)

using namespace std;

bool fileExists(const char* fileName)
{
  std::fstream fin;
  fin.open(fileName,std::ios::in);
  if( fin.is_open() )
  {
    fin.close();
    return true;
  }
  fin.close();
  return false;
}

void InsertNum(int num, char* buf)
{
  do {
    *buf=(num % 10)+'0';
    num/=10;
    buf--;
  } while (num);
}

char * FileName(int level,int filecounter, char* buf)
{
  strcpy(buf,"out      .txt");
  InsertNum(level,buf+4);
  InsertNum(filecounter,buf+8);
  return buf;
}

int main(int argc, char *argv[])
{
//  SetPriorityClass(GetCurrentProcess(),IDLE_PRIORITY_CLASS);

  bool initing=(argc>1) && (strcmp(argv[1],"-i")==0);

  set<item> *v=new set<item>;
  set<item> *vn=new set<item>;
  set<item>::iterator vi1,vi2,vi3;
  item *i,*ri=0;
  char filename[255];

  int maxlevel=(initing?INITMAXLEVEL:TOTALMAXLEVEL);

  if (initing)
  {
    int ca=6;
    while (--ca)
    {
      i=new item(1);
      i->v[0]=ca;
      v->insert(*i);
    }

    {
      ofstream f;
      f.open(FileName(1,0,filename), ios::out|ios::binary|ios::trunc);

      for (vi1 = v->begin(); vi1 != v->end(); vi1++)
      {
        f.write((const char *)vi1->v,1);
      }
      f.close();
    }
  }
```

```
int h;
int ifc;
int ofc;
int maxfiles;
for (h=(initing?2:INITMAXLEVEL+1); (h<=maxlevel) ; h++)
{
  ifc=0;
  ofc=0;
  maxfiles=((h==4)?30000:MAXFS);
  while (fileExists(FileName(h-1,ifc,filename)))
  {
    ifstream fi;
    fi.open(FileName(h-1,ifc,filename), ios::binary);
    bool nowOpened=1;
    while (nowOpened || (!fi.eof()))
    {
      nowOpened=0;
      ofstream fo;
      fo.open(FileName(h,ofc,filename), ios::out|ios::binary|ios::trunc);
      ofc++;

      i=new item(h-1);
      fi.read((char *)i->v,h-1);
      fi.read((char *)i->parent,((h-1)*(h-2))/2);
      while ((!fi.eof()) && (fo.tellp()<maxfiles))
      {
        v->clear();
        vn->clear();
        int co=0;
        do
        {
          v->insert(*i);
          delete ri;
          ri=i;
          i=new item(h-1);
          fi.read((char *)i->v,h-1);
          fi.read((char *)i->parent,((h-1)*(h-2))/2);
          co++;
        } while ((!fi.eof()) && ri->SameParent(*i));
    int coc=0;
    for (vi2 = v->begin(); vi2 != v->end(); vi2++)
    {
      coc++;
      if (vi2->isGoodPermutation())
      {
        for (vi1 = v->begin(); vi1 != v->end(); vi1++)
        {
          item wh((*vi1)+(*vi2));
          wh.v[0]=6;//(h==2?4:6);
          while (--wh.v[0])
          {
            if (((vi3=v->find(wh))!=v->end()))
            {
              vn->insert(vi3->combinate(*vi1,*vi2));
            }
          }
        }
        printf("_");
      }
    }

    for (vi1 = vn->begin(); vi1 != vn->end(); vi1++)
    {
      fo.write((const char *)vi1->v,h);
      fo.write((const char *)vi1->parent,(h*(h-1))/2);
    }
    printf(".");
  }
  fo.flush();
  fo.close();
  printf("#");
}
fi.close();
if (h<maxlevel)
  remove(FileName(h-1,ifc,filename));

ifc++;
printf("#");
```

```
    }
    printf("\n");
  }

  printf("Done\n");

  return EXIT_SUCCESS;
}
```