```
::::::::::::::
run.sh
::::::::::::::
#!/bin/bash
#-----------------------------------------------------------------------------
#  File Name:
#     run.sh
#
#  Purpose:
#
#     bash run file
#
#  Parameters:
#
#
#  Discussion:
#
#
#  Licensing:
#
#    This code is distributed under the GNU LGPL license.
#
#  Modified:
#
#    2018.11.06 Tue
#
#  Author:
#
#    Young Won Lim
#
#---------------------------------------------------------------------------

# bash -x run.bat


cd ~/Work/CORDIC/1.binary_tree_search

make binary_search N=20

cd ~/

for i in $(seq 1 5 ); do
  ./binary_search 1  |tee binary_search_i_$i.out
done

::::::::::::::
Makefile
::::::::::::::
#-----------------------------------------------------------------------------
#  File Name:
#     Makefile
#
#  Purpose:
#
#     makefile for binary_search
#
#  Parameters:
#
#
#  Discussion:
#
#
#  Licensing:
#
#    This code is distributed under the GNU LGPL license.
#
#  Modified:
#
#    2018.11.06 Tue
#
#  Author:
#
```

```
#    Young Won Lim
#
#---------------------------------------------------------------------------
CC=gcc
CFLAGS=-Wall
MACROS=-DN=$(N)
LIBS=-lm

DEPS = binary1_search_defs.h
SRCS = binary2_search_defs.c \
       binary3_level.c \
       binary4_path.c \
       binary5_traverse.c \
       binary6_subtree.c \
       binary7_cordic.c \
       binary8_main.c


OBJS = $(SRCS:.c=.o)

PRNS = run.sh Makefile $(DEPS) $(SRCS)


.SUFFIXES : .o .c .cpp

.c.o : $(DEPS)
    $(CC) -c $(CFLAGS) $(MACROS) -o $@ $<

binary_search: $(OBJS)
    $(CC) $(CFLAGS) -o ~/binary_search $^  $(LIBS)
    rm -f *.o *~ core

print: run.sh Makefile $(DEPS) $(SRCS)
    /bin/more $(PRNS) > ./print/binary_tree_search.c

clean:
    rm -f *.o *~ core

::::::::::::::
binary1_search_defs.h
::::::::::::::
//-----------------------------------------------------------------------------
//  File Name:
//     binary1_search_defs.h
//
//  Purpose:
//
//     Definitions and macros
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.11.06 Tue
//
//  Author:
//
//    Young Won Lim
//
//-----------------------------------------------------------------------------
// #define N 8     // the number of a tree
#define R 2     // the number of expanding choices = R=2
```

```
//---------------------------------------------------------
// (R)-ary tree node
// 1st R choices -a(i) at the step i  // 0
// 2nd R choices +a(i) at the step i  // 1
//---------------------------------------------------------
// for the file IO in an R script, arrange members
// that leaves no hole in memory
//---------------------------------------------------------
typedef struct node {
  double theta;                 // input angle to the i-th step
  int    branch;                // denotes which child of the parent
  int    depth;                 // denotes the i-th step computation
  int    id;                    // serial number for expand nodes

  int    child[R];              // pointers to the 2 children
  int    parent;                // pointers to the parent
} nodetype;



//---------------------------------------------------------
// queue node type
// used for breadth first search traversal
//---------------------------------------------------------
typedef struct qnode {
  struct  node * node;          // angle tree node
  struct qnode * next;          // queue node
} qnodetype;

//---------------------------------------------------------
// head queue node type
// used for classifying leaf nodes
//---------------------------------------------------------
typedef struct hqnode {
  int cindex;                   // class index
  int cnum;                     // number of classes
  int lnum;                     // number of leaf nodes
  int id;
  struct  qnode * qnode;        // queue node
  struct hqnode * next;         // head queue node
} hqnodetype;

//--- binary2.search_defs.c -------------------------------
nodetype * create_node();
qnodetype * create_qnode();
hqnodetype * create_hqnode();

//--- binary3.level.c -------------------------------------
void print_level_nodes(int depth);
nodetype find_level_min_node(int depth);

//--- binary4.path.c --------------------------------------
void find_optimal_path(nodetype *p);
void print_path(double a[], qnodetype *q);
void plot_path(qnodetype *q);

//--- binary5.traverse.c ----------------------------------
void expand_node(double a[], nodetype *p);
void tree_traverse(double a[], nodetype *p);

//--- binary6.leaves.c ------------------------------------
void write_subtree_leaves(int depth_leaf, int depth_root);
void read_subtree_leaves(int depth_leaf, int depth_root);
void write_subtree_nodes(int depth_root, int class, int depth_leaf);
void read_subtree_nodes(int depth_root, int class, int depth_leaf);

//--- binary7.cordic.c ------------------------------------
nodetype* find_cordic_nodes(double a[], nodetype *p);
void find_cordic_path(double a[], nodetype *p);
```

```
::::::::::::::
binary2_search_defs.c
::::::::::::::
//-----------------------------------------------------------------------------
//  File Name:
//     binary2_search_defs.c
//
//  Purpose:
//
//     create node and qnode
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.11.06 Tue
//
//  Author:
//
//    Young Won Lim
//
//---------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "binary1_search_defs.h"


//---------------------------------------------------------------
// create a node for an angle tree
//---------------------------------------------------------------
nodetype * create_node() {
  nodetype * p = (nodetype *) malloc (sizeof(nodetype));

  if (p == NULL) {
    perror("node creation error \n");
    exit(1);
  }
  else {
    return p;
  }
}

//---------------------------------------------------------------
// create a node for a queue
//---------------------------------------------------------------
qnodetype * create_qnode() {

  qnodetype * q = (qnodetype *) malloc (sizeof(qnodetype));

  if (q == NULL) {
    perror("qnode creation error \n");
    exit(1);
  }
  else {
    return q;
  }
}

//---------------------------------------------------------------
// create a node for a head queue
//---------------------------------------------------------------
hqnodetype * create_hqnode() {
  hqnodetype * hq = (hqnodetype *) malloc (sizeof(hqnodetype));

  if (hq == NULL) {
    perror("qnode creation error \n");
    exit(1);
  }
  else {
    return hq;
  }
}


::::::::::::::
binary3_level.c
::::::::::::::
//-----------------------------------------------------------------------------
//  File Name:
//     binary3_level.c
//
//  Purpose:
//
//     find the minimum cost leaf node
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.11.12 Mon
//
//  Author:
//
//    Young Won Lim
//
//-----------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "binary1_search_defs.h"


//---------------------------------------------------------------
// print all the nodes at the given level
//---------------------------------------------------------------
void print_level_nodes(int depth) {
  FILE *fp;
  char fname[64];
  nodetype p;

  sprintf(fname, "binary_tree_L%02d.dat", depth);

  fp = fopen(fname, "rb");

  while (fread(&p, sizeof(p), 1, fp) != 0) {
    printf(" %-5d %+f (Level %2d) ", p.id, p.theta, depth);
    printf("child: %2d %2d ", p.child[0], p.child[1]);
    printf("parent: %2d ", p.parent);
    printf("\n");
  }
  printf("------------------------------\n");

  fclose(fp);

}
```

```
//---------------------------------------------------------------
// find the node with the min residue angle at the given level
//---------------------------------------------------------------
nodetype find_level_min_node(int depth) {
  static nodetype p, p_min;
  double minval = 1e100;
  double residue;

  FILE *fp;
  char fname[64];

  // printf("* find level min node \n");

  sprintf(fname, "binary_tree_L%02d.dat", depth);

  fp = fopen(fname, "rb");

  while (fread(&p, sizeof(p), 1, fp) != 0) {
    // printf(" %d %f\n", p.id, p.theta);
    residue = fabs(p.theta);
    if (minval > residue) {
      minval = residue;
      p_min = p;
    }
  }

  fclose(fp);

  return(p_min);
}

//---------------------------------------------------------------
// sorting residue angles at the given level
//---------------------------------------------------------------
// void sort_level_nodes(int depth) { T.B.D.

::::::::::::::
binary4_path.c
::::::::::::::
//-----------------------------------------------------------------------------
//  File Name:
//     binary4_path.c
//
//  Purpose:
//
//     find and print the optimal path
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.11.13 Tue
//
//  Author:
//
//    Young Won Lim
//
//---------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "binary1_search_defs.h"

//-----------------------------------------------------------------
// a queue for a path from the root to a leaf
//-----------------------------------------------------------------
qnodetype *optimal_path;                     // Path Queue


//-----------------------------------------------------------------
// find min path (min residue angles)
//-----------------------------------------------------------------
void find_optimal_path(nodetype *p) {
  qnodetype *q;
  int depth, pid;
  FILE *fp;
  char fname[64];


  optimal_path = NULL;

  depth = p->depth;

  while (depth >= 0) {

    // printf("* optimal path : depth= %d \n", depth);

    q = create_qnode();
    q->next = optimal_path;
    q->node = p;
    optimal_path = q;

    pid = p->parent;

    p = create_node();

    depth--;

    if (depth < 0) break;

    sprintf(fname, "binary_tree_L%02d.dat", depth);

    fp = fopen(fname, "rb");

    fread(p, sizeof(*p), 1, fp);

    if (p->id == pid) continue;

    fseek(fp, (pid - p->id)*sizeof(*p), SEEK_CUR);
    fread(p, sizeof(*p), 1, fp);

    fclose(fp);

    depth = p->depth;
  }

  // printf("* end of find optimal path \n");
}

//-----------------------------------------------------------------
// print nodes in a path from root to node
//-----------------------------------------------------------------
void print_path(double a[], qnodetype* q) {
  int u, i;

  while (q) {
    printf("depth=%2d ", (q->node)->depth);
    printf("theta=%10.6f ", (q->node)->theta);
    printf("%+16.10e ", (q->node)->theta);
    i = (q->node)->depth;

    q = q->next;
```

```c
    if (q == NULL) {
      printf("\n");
      break;
    }
    printf("branch=%2d ", (q->node)->branch);

    if      ((q->node)->branch <  (R-1))    u = +1;  // ==0
    else if ((q->node)->branch == (R-1))    u = -1;  // ==1

    printf("u=%+2d ", u);
    printf("a[%2d]=%10.6f ", i, a[i]);
    printf("\n");


  }

}

//-----------------------------------------------------------------
// latex plot a path from root to node
//-----------------------------------------------------------------

char *tree(char * path) {
  char *t, *s;
  int br;
  static int i = 0;

  // printf("path=%s \n", path);

  s = malloc(256);
  t = strtok(path, " ");

  if (t == NULL) {
    sprintf(s, "%d", i++);
    // printf("s=%s \n", s);
    return(s);
  } else {
    br = atoi(t);
    switch (br) {
      case 0 : sprintf(s, "[.%d %s  x ] ", i++, tree(path+2)); break;
      case 1 : sprintf(s, "[.%d  x %s ] ", i++, tree(path+2)); break;
    }
    // printf("s=%s \n", s);
    return(s);
  }
}


void write_tree_file(char *tree_string) {
  FILE *fp;

  fp = fopen("tree.tex", "w");

  fprintf(fp,"\\documentclass{article}\n");
  fprintf(fp,"\\usepackage[margin=1in]{geometry}\n");
  fprintf(fp,"\\usepackage{graphicx}\n");
  fprintf(fp,"\\usepackage{tikz-qtree}\n");
  fprintf(fp,"\\begin{document}\n");
  fprintf(fp,"\\begin{tikzpicture}[scale=1]\n");
  fprintf(fp,"\\Tree %s\n", tree_string);
  fprintf(fp,"\\end{tikzpicture}\n");
  fprintf(fp,"\\end{document}\n");

  fclose(fp);
}


void plot_path(qnodetype* q) {
  char path[256]="", p[256]="";

  while (q) {
    q = q->next;
    if (q == NULL) {
      printf("\n");
      break;
    }
    sprintf(p, "%d ", (q->node)->branch);
    strcat(path, p);
  }

  printf("path=%s\n", path);

  strcpy(p, tree(path));
  printf("tree=%s\n", p);

  write_tree_file(p);
}


::::::::::::::
binary5_traverse.c
::::::::::::::
//-------------------------------------------------------------------------------
//  File Name:
//     binary5_traverse.c
//
//  Purpose:
//
//     tree traverse and expanding a node
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.11.12 Mon
//
//  Author:
//
//    Young Won Lim
//
//------------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "binary1_search_defs.h"


FILE *fp_r;  // read file pointer
FILE *fp_w;  // write file pointer


//-----------------------------------------------------------------
// create (R) children node to the current node pointed by p
//-----------------------------------------------------------------
void expand_node(double a[], nodetype *p) {
  nodetype c;
  int i, depth;
  double ntheta, theta;
  static int id = 1;

  // printf("* expanding a node... \n");

  theta = p->theta;
  depth = p->depth;
  for (i=0; i<R; ++i) {
    if      (i < (R-1))   ntheta = theta + 1 * a[depth];
    else if (i == (R-1))  ntheta = theta - 1 * a[depth];

    // printf("%d %f =(%f %f) \n", i, ntheta, theta, a[i]);

    c.parent     = p->id;
    c.theta      = ntheta;
    c.depth      = p->depth +1;
    c.branch     = i;
    c.id         = id++;
    p->child[i]  = c.id;

    fwrite(&c, sizeof(c), 1, fp_w);
  }

  // printf("* -sizeof(*p)=%ld \n", -sizeof(*p));

  fseek(fp_r, -sizeof(*p), SEEK_CUR);
  fwrite(p, sizeof(*p), 1, fp_r);

  // printf("* end of expand\n");
}


//---------------------------------------------------------------
// BFS Tree Traversal - level by level
//---------------------------------------------------------------
void tree_traverse(double a[], nodetype *r) {
  nodetype p;
  int depth;

  char fname_r[64];
  char fname_w[64];


  printf("* tree traversing ... \n");

  sprintf(fname_w, "binary_tree_L%02d.dat", 0);
  fp_w = fopen(fname_w, "w");
  fwrite(r, sizeof(*r), 1, fp_w);
  fclose(fp_w);


  for (depth=0; depth<N; ++depth) {


    // printf("* depth= %d \n", depth);

    sprintf(fname_r, "binary_tree_L%02d.dat", depth);
    sprintf(fname_w, "binary_tree_L%02d.dat", depth+1);

    // printf("* reading %s\n", fname_r);
    // printf("* writing %s\n", fname_w);

    fp_r = fopen(fname_r, "r+");
    fp_w = fopen(fname_w, "w");

    while (fread(&p, sizeof(p), 1, fp_r) != 0) {
      expand_node(a, &p);
    }

    fclose(fp_r);
    fclose(fp_w);
  }

  // printf("* end of tree traversing ... \n");
}

::::::::::::::
binary6_subtree.c
```

```
::::::::::::::
//-----------------------------------------------------------------------------
//  File Name:
//     binary6_subtee.c
//
//  Purpose:
//
//     read / write subtrees and their leaf nodes
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.11.06 Tue
//
//  Author:
//
//    Young Won Lim
//
//---------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "binary1_search_defs.h"



//---------------------------------------------------------------
// write all classified leaf nodes
//---------------------------------------------------------------
void write_subtree_leaves(int depth_root, int depth_leaf) {
  nodetype p;
  int cnum;   // the number of classes at depth_root
  int lnum;   // the number of leaves per each class at depth_leaf
  int i, j, cnt;

  FILE *fp1;  // read file pointer
  FILE *fp2;  // write file pointer

  char fname1[64];
  char fname2[64];


  cnum = (int) pow(R, depth_root);          // no of classes
  lnum = (int) pow(R, depth_leaf) / cnum;   // no of leaves per class


  sprintf(fname1, "binary_tree_L%02d.dat", depth_leaf);
  fp1 = fopen(fname1, "r");

  for (i=0; i<cnum; i++) {
    sprintf(fname2, "binary_tree_L%02d.G%02d.dat", depth_leaf, i);
    fp2 = fopen(fname2, "w");

    for (j=0; j<lnum; j++) {
       cnt = fread(&p, sizeof(p), 1, fp1);
       if (cnt == 0) {
         perror("* error in reading file ...\n");
         exit(1);
       }
       fwrite(&p, sizeof(p), 1, fp2);
    }
    fclose(fp2);
  }

  fclose(fp1);

}

//---------------------------------------------------------------
// read all classified leaf nodes
//---------------------------------------------------------------
void read_subtree_leaves(int depth_root, int depth_leaf) {
  nodetype p;
  int cnum;   // the number of classes at depth_root
  int lnum;   // the number of leaves per each class at depth_leaf
  int i, j;

  FILE *fp2;  // write file pointer

  char fname2[64];


  cnum = (int) pow(R, depth_root);          // no of classes
  lnum = (int) pow(R, depth_leaf) / cnum;   // no of leaves per class


  for (i=0; i<cnum; i++) {
    sprintf(fname2, "binary_tree_L%02d.G%02d.dat", depth_leaf, i);
    fp2 = fopen(fname2, "r");

    for (j=0; j<lnum; j++) {
       fread(&p, sizeof(p), 1, fp2);
       printf(" %d", p.id);
    }
    printf(" * Group %02d\n", i);

    fclose(fp2);
  }

}

//---------------------------------------------------------------
// write subtree nodes
//---------------------------------------------------------------
void write_subtree_nodes(int depth_root, int class, int depth_leaf) {
  nodetype p;
  int cnum;   // the number of clases
  int lnum;   // the number of leaves per each class at depth_leaf
  int i, j, cnt;

  FILE *fp1;  // read file pointer
  FILE *fp2;  // write file pointer

  char fname1[64];
  char fname2[64];


  for (i=depth_root; i<=depth_leaf; i++) {
    cnum = (int) pow(R, depth_root);     // no of classes
    lnum = (int) pow(R, i) / cnum ;      // no of leaves per class

    sprintf(fname1, "binary_tree_L%02d.dat", i);
    fp1 = fopen(fname1, "r");

    sprintf(fname2, "binary_tree_L%02d.G%02d", i, class);
    sprintf(fname2, "%s.L%02d.dat", fname2, i - depth_root);
    fp2 = fopen(fname2, "w");

    fseek(fp1, class*lnum*sizeof(p), SEEK_CUR);
    for (j=0; j<lnum; j++) {
       cnt = fread(&p, sizeof(p), 1, fp1);
       if (cnt == 0) {
         perror("* error in reading file ...\n");
         exit(1);
       }
       fwrite(&p, sizeof(p), 1, fp2);
    }

    fclose(fp2);
    fclose(fp1);
  }

}

//------------------------------------------------------------------
// read subtree nodes
//------------------------------------------------------------------
void read_subtree_nodes(int depth_root, int class, int depth_leaf) {
  nodetype p;
  int cnum;   // the number of clases
  int lnum;   // the number of leaves per each class at depth_leaf
  int i, j;

  FILE *fp2;  // write file pointer

  char fname2[64];


  for (i=depth_root; i<=depth_leaf; i++) {
    cnum = (int) pow(R, depth_root);     // no of classes
    lnum = (int) pow(R, i) / cnum ;      // no of leaves per class

    sprintf(fname2, "binary_tree_L%02d.G%02d", i, class);
    sprintf(fname2, "%s.L%02d.dat", fname2, i - depth_root);
    fp2 = fopen(fname2, "r");

    for (j=0; j<lnum; j++) {
       fread(&p, sizeof(p), 1, fp2);
       printf(" %d", p.id);
    }
    printf(" * Level %02d (%02d)\n", i, i-depth_root);

    fclose(fp2);
  }

}




::::::::::::::
binary7_cordic.c
::::::::::::::
//--------------------------------------------------------------------------------
//  File Name:
//     binary7_cordic.c
//
//  Purpose:
//
//     finding the cordic path
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.11.06 Tue
//
```

```c
//  Author:
//
//    Young Won Lim
//
//---------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "binary1_search_defs.h"

qnodetype *cordic_path=NULL;             // CORDIC Queue Head
qnodetype *cordic_tail=NULL;             // CORDIC Queue Tail

//---------------------------------------------------------------
// create (R) children node to the current node pointed by p
//---------------------------------------------------------------
nodetype* find_cordic_nodes(double a[], nodetype *p) {
  nodetype *np;
  int i, depth, mindex=0;
  double ntheta[2], theta, minval=1E+10;
  static int id = 1;

  // printf("* cordic node... \n");

  theta = p->theta;
  depth = p->depth;

  for (i=0; i<R; ++i) {
    if      (i < (R-1))   ntheta[i] = theta + 1 * a[depth];
    else if (i == (R-1))  ntheta[i] = theta - 1 * a[depth];
  }

  for (i=0; i<R; ++i) {
    if (minval > fabs(ntheta[i])) {
      minval = ntheta[i];
      mindex = i;
    }
  }

  // printf("%d %f =(%f %f) \n", mindex, ntheta[mindex], theta, a[depth]);


  np = create_node ();
   p->child[mindex] = id;
  np->parent        = p->id;
  np->theta         = ntheta[mindex];
  np->depth         = p->depth +1;
  np->branch        = mindex;
  np->id            = id++;


  //-- if (ntheta > theta) np->branch = -1;

   return np;
}




//---------------------------------------------------------------
// CORDIC Traversal
//---------------------------------------------------------------
void find_cordic_path(double a[], nodetype *p) {
  qnodetype *q, *nq;
  nodetype *np;
  int k =0;

  // printf("* cordic traversing ... \n");

  q = create_qnode();
  q->node = p;
  cordic_path = q;
  cordic_tail = q;

  while (cordic_tail != NULL) {
    // printf("* node %d to be expanded \n", k);

    k++;

    if ((q->node)->depth >= (N-1) ) {
      cordic_tail->next = NULL;
      break;
    }

    if (q != NULL) np = find_cordic_nodes(a, q->node);

    nq = create_qnode();
    nq->node = np;

    cordic_tail->next = nq;
    cordic_tail = nq;

    q = nq;
  }

}





::::::::::::::
binary8_main.c
::::::::::::::
//-------------------------------------------------------------------------------
//  File Name:
//     binary8_main.c
//
//  Purpose:
//
//     binary angle tree search main
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.11.06 Tue
//
//  Author:
//
//    Young Won Lim
//
//-----------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "binary1_search_defs.h"

extern qnodetype *optimal_path;
extern qnodetype *cordic_path;

//-----------------------------------------------------------------
// main - Ternary Angle Tree Search
```

```
//---------------------------------------------------------------
int main(int argc, char *argv[]) {
  double a[N];
  double theta; // = 4*atan(pow(2,-5));
  int i;

  nodetype p;
  nodetype leaf;

  if (argc != 2) {
    printf("binary_search i (theta=2^(-i)) \n");
    return 0;
  }

  i = atoi(argv[1]);
  theta = atan(pow(2, -1*i));

  printf("binary angle tree search (N=%d) \n", N);
  printf("theta= atan(pow(2,%d) = %10g \n", -1*i, theta);


  for (i=0; i<N; ++i) {
    a[i] = atan(1./pow(2, i));
  }

  p.theta = theta;
  p.depth = 0;
  p.id = 0;

  tree_traverse(a, &p);

  for (i=0; i<N; ++i) {
     // printf("level %d\n", i);
     // print_level_list(i);
     find_level_min_node(i);
  }

  leaf = find_level_min_node(N-1);

  printf("* the optimal min path \n");
  find_optimal_path(&leaf);
  print_path(a, optimal_path);
  plot_path(optimal_path);

  return 0;

  printf("* the cordic path \n");
  find_cordic_path(a, &p);
  print_path(a, cordic_path);


  printf("* classify leaf nodes \n");
  write_subtree_leaves(2, N-1);
  read_subtree_leaves(2, N-1);

  printf("* subtree nodes \n");
  write_subtree_nodes(2, 3, 5);
  read_subtree_nodes(2, 3, 5);

  printf("* print level nodes \n");
  for (i=0; i<N; ++i) {
     print_level_nodes(i);
  }

}
```