```
::::::::::::::
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:
#
#    2018.12.07 Fri
#
#  Author:
#
#    Young Won Lim
#
#----------------------------------------------------------------------------

# bash -x run.sh

NT=12

rm ~/binary_*
rm ~/ternary_*
rm ~/quaternary_*
rm ~/tc1_*
rm ~/Data/tc1_*

#----------------------------------------------------------
cd ~/Work/CORDIC/1.binary_tree_search
make binary_library N=$NT DISP=0

cd ~/Work/CORDIC/2.ternary_tree_search
make ternary_library N=$NT DISP=0

cd ~/Work/CORDIC/3.quaternary_tree_search
make quaternary_library N=$NT DISP=0
#----------------------------------------------------------

dname=~/Work/CORDIC/5.testbench/testcase1

set -x

cd $dname

make tc1 N=$NT

cd ~/


#----------------------------------------------------------
function m_ary_run {
  printf "\n\n\n\n\n"
  bname=tc1_power2
  fname="$bname"_"$1"

  ./$fname |tee $fname.log


  awk -f $dname/table.awk $fname.log > $fname.tab

  enscript -o - $fname.log | ps2pdf - $fname.log.pdf
  enscript -o - $fname.tab | ps2pdf - $fname.tab.pdf

  # pdfunite binary_tree_*.pdf $fname.log.pdf $fname.out.pdf

  cp $fname.log.pdf $dname/output
  cp $fname.tab.pdf $dname/output

}
#----------------------------------------------------------


m_ary_run 2ary
m_ary_run 3ary
m_ary_run 4ary



exit



::::::::::::::
table.awk
::::::::::::::
#-------------------------------------------------------------------------
# tc1_power2_2ary_i3 R=2 i=3 i=3
# * leaf min node : depth= 9 theta= -0.002643 minval=  0.002643 id=728
# * global min node : depth= 7 theta= -0.001263 minval=  0.001263 id=192
# * cordic min node : depth= 9 theta=  0.124355 minval=  0.002643 id=0
#-------------------------------------------------------------------------


# /^tc1_power2/   { print $3;  split($3, a, "="); i=a[2]; print i}
# /\* leaf min/   { print $9; leaf[i]=$9; print i, leaf[i] }
# /\* global min/ { print $9; glob[i]=$9; print i, glob[i] }
# /\* cordic min/ { print $9; cord[i]=$9; print i, cord[i] }

/^tc1_power2/   { split($3, a, "="); i=a[2]; }
/\* leaf min/   { leaf[i]=$9; }
/\* global min/ { glob[i]=$9; }
/\* cordic min/ { cord[i]=$9; }

END {
  printf("%4s %+14s %+14s %+14s \n", "i", "leaf", "global", "cordic");
  for (i in leaf) {
    printf("%4d %+14.6e %+14.6e %+14.6e \n", i, leaf[i], glob[i], cord[i]);
  }
}
::::::::::::::
plot.R
::::::::::::::
#------------------------------------------------------------------------------
#  File Name:
#     plot.R
#
#  Purpose:
#
#     R script file for search tree bar plot
#
```

```
#  Parameters:
#
#
#  Discussion:
#
#
#  Licensing:
#
#    This code is distributed under the GNU LGPL license.
#
#  Modified:
#
#    2018.12.24 Mon
#
#  Author:
#
#    Young Won Lim
#
#----------------------------------------------------------------------------


#----------------------------------------------------------------------

mary_plot <- function(fname) {
  df <- read.table(fname, header=T)

  c0<-df["i"]
  c1<-df["leaf"] / 2^(-df["i"])*100
  c2<-df["global"] / 2^(-df["i"])*100
  c3<-df["cordic"] / 2^(-df["i"])*100

  df2 <- data.frame("i"=c0, "leaf"=c1, "global"=c2, "cordic"=c3)
  df3 <- abs(df2)
  df4 <- t(df3[2:4])



  local({
  ## Print result
  x <- df3[2:4]
  # barplot is a bit picky about attributes, so we need to convert to vector explicitly
  if(!is.matrix(x)) x <- as.vector(x)
  if(!is.matrix(x) && is.data.frame(x)) x <- data.matrix(x)
  rk.header (paste(fname, "(1)"), parameters=list ("Variable"="percentage error", "colors"=
"default", "Type"="juxtaposed", "Legend"="TRUE"))

  rk.graph.on ()
  try ({
          barplot(x, beside=TRUE)
  })
  rk.graph.off ()
  })


  local({
  ## Print result
  x <- df4
  n <- c(0:9);
  # barplot is a bit picky about attributes, so we need to convert to vector explicitly
  if(!is.matrix(x)) x <- as.vector(x)
  if(!is.matrix(x) && is.data.frame(x)) x <- data.matrix(x)
  rk.header (paste(fname, "(2)"), parameters=list ("Variable"="percentage error", "colors"=
"default", "Type"="juxtaposed", "Legend"="TRUE"))

  rk.graph.on ()
  try ({
          barplot(x, beside=TRUE, names.arg=n)
  })
  rk.graph.off ()
  })

}


#--------------------------------------------------------------------------------
compare_plot <- function() {
  df2ary <- read.table("tc1_power2_2ary.tab", header=T)
  df3ary <- read.table("tc1_power2_3ary.tab", header=T)
  df4ary <- read.table("tc1_power2_4ary.tab", header=T)

  c0<-df2ary["i"]
  c1<-df2ary["cordic"] / 2^(-df2ary["i"])*100
  c2<-df3ary["cordic"] / 2^(-df3ary["i"])*100
  c3<-df4ary["cordic"] / 2^(-df4ary["i"])*100

  df2 <- data.frame("i"=c0, "2ary"=c1, "3ary"=c2, "4ary"=c3)
  df3 <- abs(df2)
  df4 <- t(df3[2:4])



  local({
  ## Print result
  x <- df3[2:4]
  n <- c("binary cordic", "ternary cordic", "quaternary cordic")
  # barplot is a bit picky about attributes, so we need to convert to vector explicitly
  if(!is.matrix(x)) x <- as.vector(x)
  if(!is.matrix(x) && is.data.frame(x)) x <- data.matrix(x)
  rk.header (paste("comparison", "(1)"), parameters=list ("Variable"="percentage error", "c
olors"="default", "Type"="juxtaposed", "Legend"="TRUE"))

  rk.graph.on ()
  try ({
          barplot(x, beside=TRUE, names.arg=n)
  })
  rk.graph.off ()
  })


  local({
  ## Print result
  x <- df4
  n <- c(0:9);
  # barplot is a bit picky about attributes, so we need to convert to vector explicitly
  if(!is.matrix(x)) x <- as.vector(x)
  if(!is.matrix(x) && is.data.frame(x)) x <- data.matrix(x)
  rk.header (paste("comparison", "(2)"), parameters=list ("Variable"="percentage error", "c
olors"="default", "Type"="juxtaposed", "Legend"="TRUE"))

  rk.graph.on ()
  try ({
          barplot(x, beside=TRUE, names.arg=n)
  })
  rk.graph.off ()
  })

}


mary_plot("tc1_power2_2ary.tab")
mary_plot("tc1_power2_3ary.tab")
mary_plot("tc1_power2_4ary.tab")



compare_plot()
```

```
::::::::::::::
Makefile
::::::::::::::
#------------------------------------------------------------------------------
#  File Name:
#     Makefile
#
#  Purpose:
#
#     makefile for testbenches
#
#  Parameters:
#
#
#  Discussion:
#
#
#  Licensing:
#
#    This code is distributed under the GNU LGPL license.
#
#  Modified:
#
#    2018.12.07 Fri
#
#  Author:
#
#    Young Won Lim
#
#----------------------------------------------------------------------------
CC=gcc
CFLAGS=-Wall
MACROS1=-DN=$(N) -DR=2
MACROS2=-DN=$(N) -DR=3
MACROS3=-DN=$(N) -DR=4

TC1=tc1_power2

LIBS=-lm

DEPS = tc0_testcase_defs.h
SRCS = tc1_main.c \
       tc1_power2.c \

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

PRNS = run.sh table.awk plot.R Makefile $(DEPS) $(SRCS)

# FNAME = ./print/binary_search.$(shell date +%Y%m%d).c
FNAME = ./print/testcase1.c


# .SUFFIXES : .o .c .cpp

tc1_bin : $(SRCS) $(DEPS)
        $(CC) -c $(CFLAGS) $(MACROS1) $(SRCS)
        $(CC) $(CFLAGS) $(MACROS1) -o ˜/$(TC1)_2ary $(OBJS) -L. -lm -lbinary
        rm -f *.o *˜ core


tc1 : $(SRCS) $(DEPS)
        $(CC) $(CFLAGS) $(MACROS1) -o ˜/$(TC1)_2ary $(SRCS) -L.. -lm -lbinary
        $(CC) $(CFLAGS) $(MACROS2) -o ˜/$(TC1)_3ary $(SRCS) -L.. -lm -lternary
        $(CC) $(CFLAGS) $(MACROS3) -o ˜/$(TC1)_4ary $(SRCS) -L.. -lm -lquaternary
        rm -f *.o *˜ core

print: $(PRNS)
```

```
        /bin/more $(PRNS) > $(FNAME)
        enscript -o - --highlight=c $(FNAME) | ps2pdf - $(FNAME).pdf


clean:
        rm -f *.o *~ core
        rm -f ~/binary* ~/ternary* ~/quaternary* ~/tc1*

run:
        bash -x run.sh


::::::::::::::
tc0_testcase_defs.h
::::::::::::::
//------------------------------------------------------------------------------
//  File Name:
//     tc0_testcase_defs.h
//
//  Purpose:
//
//     Definitions and macros
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.12.05 Wed
//
//  Author:
//
//    Young Won Lim
//
//------------------------------------------------------------------------------
// #define N 8     // the depth of a binary tree
// #define R 2     // the number of expanding choices = R=2
// #define TREE "binary_tree"

#if R == 2
  #define M_ARY "binary"
  #define TREE "binary_tree"
#elif R == 3
  #define M_ARY "ternary"
  #define TREE "ternary_tree"
#elif R == 4
  #define M_ARY "quaternary"
  #define TREE "quaternary_tree"
#endif




//----------------------------------------------------------
// (R)-ary tree node
//----------------------------------------------------------
// 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;


//--------- 2.search_defs.c -------------------------------
nodetype * create_node();
qnodetype * create_qnode();

//--------- 3.traverse.c ----------------------------------
void pr_node(nodetype *p);
void copy_node(nodetype *p, nodetype *q);
void expand_node(nodetype *p, int rid);
void tree_traverse(nodetype *p);

//--------- 4.level.c -------------------------------------
void print_level_nodes(int depth);
nodetype find_level_min_node(int depth);
nodetype find_global_min_node();

//--------- 5.path.c --------------------------------------
qnodetype* find_path(nodetype *p);
void print_path(qnodetype *q, char *str);
void delete_path(qnodetype* q, char *str);

//--------- 6.cordic.c ------------------------------------
nodetype* cordic_expand(nodetype *p, int rid);
qnodetype* cordic_traverse(nodetype *p);
qnodetype *find_cordic_path(nodetype *p);
nodetype find_cordic_node(nodetype *p);

//--------- 7.subtree.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);

//--------- 8.plot.c --------------------------------------
void plot_path(qnodetype *q, char *str);


//---------------------------------------------------------
// Global Variables
//---------------------------------------------------------
typedef struct param {
  int NN;  // the depth/height of a tree
  int RR;  // R
  double theta;

  char tstring[256];

} paramtype;
```

```
paramtype Param;

double a[2*N];  // because of quaternary search tree






::::::::::::::
tc1_main.c
::::::::::::::
//----------------------------------------------------------------------------
//  File Name:
//     tc1_main.c
//
//  Purpose:
//
//     binary angle tree search main
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.12.07 Fri
//
//  Author:
//
//    Young Won Lim
//
//--------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#include "tc0_testcase_defs.h"


void tc1_power2(int exp, double theta);

qnodetype *leafmin_path;
qnodetype *globalmin_path;
qnodetype *cordic_path;




//----------------------------------------------------------------
// main - Ternary Angle Tree Search
//----------------------------------------------------------------
int main(void) {
  double theta; // = 4*atan(pow(2,-5));
  int i;

  char fname1[64];


  for (i=0; i<10; ++i) {
    switch (R) {
      case 2: sprintf(fname1, "tc1_power2_2ary"); break;
      case 3: sprintf(fname1, "tc1_power2_3ary"); break;
      case 4: sprintf(fname1, "tc1_power2_4ary"); break;
    }


    printf(";;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;\n");
    printf("%s_i%d R=%d i=%d \n", fname1, i, R, i);
    printf(";;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;\n");



    theta = atan(pow(2, -1*i));

    tc1_power2(i, theta);

  }

}







::::::::::::::
tc1_power2.c
::::::::::::::
//-----------------------------------------------------------------------------
//  File Name:
//     tc1_power2.c
//
//  Purpose:
//
//     testcase 1: power of 2 angles
//
//  Parameters:
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.12.07 Fri
//
//  Author:
//
//    Young Won Lim
//
//---------------------------------------------------------------------------
//  m_ary_search.log
//  m_ary_search.log.pdf
//  m_ary_search.out.pdf
//
//  m_ary_tree_1_leafmin.pdf   (.aux, .dvi, .log, .tex)
//  m_ary_tree_2_globalmin.pdf (.aux, .dvi, .log, .tex)
//  m_ary_tree_3_cordic.pdf    (.aux, .dvi, .log, .tex)
//
//  m_ary_tree_L01.dat
//  m_ary_tree_L02.dat
//  ...
//  m_ary_tree_LNN.dat
//---------------------------------------------------------------------------
#include <stdio.h>
```

```c
#include <math.h>
#include <stdlib.h>
#include <string.h>

#include "tc0_testcase_defs.h"



extern qnodetype *leafmin_path;
extern qnodetype *globalmin_path;
extern qnodetype *cordic_path;

//---------------------------------------------------------------
// main - Ternary Angle Tree Search
//---------------------------------------------------------------
void tc1_power2(int exp, double theta) {

  nodetype p;
  nodetype min_leaf;
  nodetype min_global;
  nodetype cordic_node;

  int i;

  char tstring[64];


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

  sprintf(tstring, "%s angle tree", M_ARY);

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


  Param.NN    = N;
  Param.RR    = R;
  Param.theta = theta;
  strcpy(Param.tstring, tstring);

  p.theta = theta;
  p.depth = 0;
  p.id = 0;
  p.branch = 0;
  for (i=0; i<R; ++i) p.child[i]= i+1;


  //--------------------------------------------------
  tree_traverse(&p);
  //--------------------------------------------------


  printf("\n......................................\n");
  printf("* A: the leaf optimal path  R=%d i=%d\n", R, exp);
  printf("......................................\n");
  min_leaf = find_level_min_node(N-1);
  leafmin_path = find_path(&min_leaf);
  print_path(leafmin_path, "leafmin");
  // plot_path(leafmin_path, "leafmin");


  printf("\n......................................\n");
  printf("* B: the global optimal path  R=%d i=%d\n", R, exp);
  printf("......................................\n");
  min_global = find_global_min_node();
  globalmin_path = find_path(&min_global);
  print_path(globalmin_path, "globalmin");
  // plot_path(globalmin_path, "globalmin");


  printf("\n......................................\n");
  printf("* C: the cordic path R=%d i=%d\n", R, exp);
  printf("......................................\n");
  cordic_node = find_cordic_node(&p); // method 3
  cordic_path = find_path(&cordic_node);
  print_path(cordic_path, "cordic");
  // plot_path(cordic_path, "cordic");



  delete_path(leafmin_path, "leafmin");
  delete_path(globalmin_path, "globalmin");
  delete_path(cordic_path, "cordic");

}
```