```
::::::::::::::
run.sh
::::::::::::::
#!/bin/bash

# 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
::::::::::::::
CC=gcc
CFLAGS=-Wall
MACROS=-DN=$(N)
LIBS=-lm

DEPS = binary1_search_defs.h
SRCS = binary2_search_defs.c \
       binary3_level_queue.c \
       binary4_path_queue.c \
       binary5_bfs_queue.c \
       binary6_traverse.c \
       binary7_leaves_queue.c \
       binary8_cordic.c \
       binary9_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
::::::::::::::
// #define N 8     // the number of a tree
#define R 1     // the number of expanding choices = 2*R


//---------------------------------------------------------
// (2R)-ary tree node
// 1st R choices -a(i) at the step i  // 2*0+0 =0
// 2nd R choices +a(i) at the step i  // 2*0+1 =1
//---------------------------------------------------------
typedef struct node {
  int    branch;              // denotes which child of the parent
  double theta;               // input angle to the i-th step
  int    depth;               // denotes the i-th step computation
  int    id;                  // serial number for expand nodes

  struct node * child[2*R];   // pointers to the 2 children
  struct node * 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;
  int cnum;
  int lnum;
  int id;
  struct  qnode * qnode;       // queue node
  struct hqnode * next;        // head queue node
} hqnodetype;


nodetype * create_node();
qnodetype * create_qnode();
hqnodetype * create_hqnode();

void insert_level_list(nodetype *np);
void print_level_list(int depth);
void write_level_list(int depth);
nodetype * level_list_min_node(int depth);

void find_minpath(nodetype *p);
void list_path(double a[], qnodetype *q);

void enqueue(qnodetype *q);
qnodetype * dequeue();

void expand_node(double a[], nodetype *p);
void tree_traverse(double a[], nodetype *p);

void init_head_queue(int depth);
int find_ancesstor_id(nodetype *p, int depth);
void insert_leaf_list(nodetype *p, int depth);
void classify_leaf_ancesstor(int depth_root, int depth_leaf);
void write_leaf_ancesstor(int depth);

int cordic_node(double a[], nodetype *p);
void cordic_traverse(double a[], nodetype *p);



::::::::::::::
binary2_search_defs.c
::::::::::::::
//-----------------------------------------------------------------------------
//  Purpose:
//
//     create node and qnode
//
//  Discussion:
//
//
//  Licensing:
//
```

```
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.22 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------
#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_queue.c
::::::::::::::
```

```
//-----------------------------------------------------------------------------
//  Purpose:
//
//     Level Queue
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.22 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "binary1_search_defs.h"

//---------------------------------------------------------------
// queues for each level nodes of an angle tree
//---------------------------------------------------------------
qnodetype *larr[N];                     // Level Queue


//---------------------------------------------------------------
// insert a qnode to larr queues
//---------------------------------------------------------------
void insert_level_list(nodetype *np) {
  int depth = np->depth;
  qnodetype *q;

  q = create_qnode();
  q->node = np;
  q->next = larr[depth];
  larr[depth] = q;
}

//---------------------------------------------------------------
// print all the nodes at the given level
//---------------------------------------------------------------
void print_level_list(int depth) {
  qnodetype *q;

  q = larr[depth];

  while (q) {
    printf(" %d %f\n", (q->node)->id, (q->node)->theta);
    q = q->next;
  }

  printf("\n");

}

//---------------------------------------------------------------
// write all the nodes at the given level
//---------------------------------------------------------------
void write_level_list(int depth) {
  qnodetype *q;
  FILE *fp;
  double d;
  int cnt = 0;

  q = larr[depth];
  while (q) {
    q = q->next;
    cnt++;
  }

  fp = fopen("binary_leaf.bin", "wb");

  fwrite(&cnt, sizeof(cnt), 1, fp);

  q = larr[depth];
  while (q) {
    d = (q->node)->theta;
    fwrite(&d, sizeof(d), 1, fp);
    q = q->next;
  }
  fclose(fp);

  printf("* %d double data write to leaf.bin\n", cnt);
}

//----------------------------------------------------------------
// find the node with the min residue angle at the given level
//----------------------------------------------------------------
nodetype * level_list_min_node(int depth) {
  qnodetype *q;
   nodetype *p;
  double minval = 1e100;
  double residue;

  q = larr[depth];

  while (q) {
    residue = fabs((q->node)->theta);
    if (minval > residue) {
      minval = residue;
      p = q->node;
    }
    q = q->next;
  }

  // printf("%f \n", p->theta);
  return(p);
}


::::::::::::::
binary4_path_queue.c
::::::::::::::
//------------------------------------------------------------------------------
//  Purpose:
//
//     Path Queue
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.22 Mon
//
//
//  Author:
//
//    Young Won Lim
//
```

```
//  Parameters:
//
//----------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "binary1_search_defs.h"

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


//----------------------------------------------------------------
// find min path (min residue angles)
//----------------------------------------------------------------
void find_minpath(nodetype *p) {
  qnodetype *q;
  minpath = NULL;

  while (p) {
    q = create_qnode();
    q->next = minpath;
    q->node = p;
    minpath = q;
    p = p->parent;
  }

}

//----------------------------------------------------------------
// print nodes in a path from root to node
//----------------------------------------------------------------
void list_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;

    if (q == NULL) {
      printf("\n");
      break;
    }
    printf("branch=%2d ", (q->node)->branch);
    if ((q->node)->branch < R)          u = +1;  // ==0
    else if ((q->node)->branch == R)    u = -1;  // ==1

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


  }

}



::::::::::::::
binary5_bfs_queue.c
::::::::::::::
//----------------------------------------------------------------------------
//  Purpose:
//
//     BFS Queue
```

```
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.22 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "binary1_search_defs.h"


//-----------------------------------------------------------------
// A queue for BFS (Breadth First Search) Tree Traversal
//-----------------------------------------------------------------
qnodetype *head =NULL;                  // BFS Queue Head
qnodetype *tail =NULL;                  // BFS Queue Tail



//-----------------------------------------------------------------
// insert a qnode into the BFS queue
//-----------------------------------------------------------------
void enqueue(qnodetype *q) {
  // printf("* enqueue ... \n");
  if (head == NULL && tail == NULL) head = q;
  if (tail != NULL) tail->next = q;
  tail = q;
}

//-----------------------------------------------------------------
// delete a qnode from the BFS queue
//-----------------------------------------------------------------
qnodetype * dequeue() {
  // printf("* dequeue ... \n");
  qnodetype *q;
  static int depth = 0;

  if (head != NULL)  {
    q = head;

    if (head != tail) head = head->next;
    else head = tail = NULL;

    if (depth != (q->node)->depth) {
      // printf("level %d \n", depth);
      depth = (q->node)->depth;
    }

    return q;
  }
  else {
    return NULL;
  }
}
```

```
::::::::::::::
binary6_traverse.c
::::::::::::::
//------------------------------------------------------------------------------
//  Purpose:
//
//     Tree Traverse
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.22 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//------------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "binary1_search_defs.h"

extern qnodetype *head;                  // BFS Queue Head
extern qnodetype *tail;                  // BFS Queue Tail

//-----------------------------------------------------------------
// create (2R) children node to the current node pointed by p
//-----------------------------------------------------------------
void expand_node(double a[], nodetype *p) {
  nodetype *np;
  int i, depth;
  double ntheta, theta;
  static int id = 1;

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

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

  if (p->depth == 0) insert_level_list(p);
  //-- if (p->branch < 0) return;

  for (i=0; i<2*R; ++i) {
    if (i < R)        ntheta = theta + 1 * a[depth];
    else if (i == R)  ntheta = theta - 1 * a[depth];

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


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

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

  }
}



//---------------------------------------------------------------
// BFS Tree Traversal
//---------------------------------------------------------------
void tree_traverse(double a[], nodetype *p) {
  qnodetype *q, *nq;
  int i, k =0;

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

  q = create_qnode();
  q->node = p;
  enqueue(q);

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

    q = dequeue();
    k++;

    if ((q->node)->depth >= N) break;

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

    for (i=0; i<2*R; ++i) {
      //-- if ((q->node)->theta < ((q->node)->child[i])->theta) continue;
      //-- if (((q->node)->child[i])->branch <0) continue;

      nq = create_qnode();
      nq->node = (q->node)->child[i];
      enqueue(nq);
    }
  }

}

::::::::::::::
binary7_leaves_queue.c
::::::::::::::
//------------------------------------------------------------------------------
//  Purpose:
//
//     Level Queue
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.22 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//------------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "binary1_search_defs.h"
```

```
#define CL 2        // Class Level
#define CN 4        // Clsss Number 2^CL number of classes

extern qnodetype *larr[N];                     // Level Queue

//---------------------------------------------------------------
// head queues for classified leaf nodes of an angle tree
//---------------------------------------------------------------
hqnodetype *headq;                       // Head Queue Global Var

//---------------------------------------------------------------
// initializes the head queue
//---------------------------------------------------------------
void init_head_queue(int depth) {
  hqnodetype *hq;
  qnodetype *q;
  int cindex=0;

  // traverse a given depth level queue
  q = larr[depth];

  while (q != NULL) {
    hq = create_hqnode();

    hq->cindex = cindex;
    hq->id     = (q->node)->id;

    hq->qnode = NULL;
    hq->next = headq;
    headq = hq;

    cindex++;

    q = q->next;
  }

  // tarverse headq filling cnum
  hq = headq;
  while (hq != NULL) {
    hq->cnum = cindex;
    hq->lnum = 0;
    hq = hq->next;
  }

}

//---------------------------------------------------------------
// find out the ancesstor's id of a leaf node
//---------------------------------------------------------------
int find_ancesstor_id(nodetype *p, int depth) {

  while (p) {
    // printf(" %d %f\n", p->depth, p->id);
    p = p->parent;

    if (p->depth <= depth) break;
  }

  return (p->id);
}

//---------------------------------------------------------------
// insert a qnode to larr queues
//---------------------------------------------------------------
void insert_leaf_list(nodetype *p, int depth) {
  hqnodetype *hq;
  qnodetype *nq;
  int id;

  id = find_ancesstor_id(p, depth);
  // find out the place in headq
  hq = headq;
  while (hq != NULL) {
    if (hq->id == id) break;
    hq = hq->next;
  }

  (hq->lnum)++;

  nq = create_qnode();

  nq->node = p;

  nq->next = hq->qnode;
  hq->qnode = nq;
}



//-----------------------------------------------------------------
// classify all leaf node (depth_leaf) as descendants
// of subtrees rooted at (depth_root)
//-----------------------------------------------------------------
void classify_leaf_ancesstor(int depth_root, int depth_leaf) {
  qnodetype *q;

  init_head_queue(depth_root);

  q = larr[depth_leaf];

  while (q) {
    // printf(" %d %f\n", (q->node)->id, (q->node)->theta);

    insert_leaf_list(q->node, depth_root);

    q = q->next;
  }

  printf("\n");

}

//-----------------------------------------------------------------
// write all classified leaf nodes
//-----------------------------------------------------------------
void write_leaf_ancesstor(int depth) {
  qnodetype *q;
  hqnodetype *hq;
  int cnum, lnum;
  double d;

  // int i;

  FILE *fp;

  fp = fopen("binary_leaf_class.bin", "wb");

  hq = headq;
  cnum = hq->cnum;

  fwrite(&cnum, sizeof(cnum), 1, fp);

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

  while (hq != NULL) {

    q = hq->qnode;
    lnum = hq->lnum;

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

    fwrite(&lnum, sizeof(lnum), 1, fp);
    // i=0;
    while (q != NULL) {
      d = (q->node)->theta;

      // printf("i=%d d=%f\n", i++, d);

      fwrite(&d, sizeof(d), 1, fp);

      q = q->next;
    }

    // printf("move next hq\n");

    hq = hq->next;
  }


}

::::::::::::::
binary8_cordic.c
::::::::::::::
//-----------------------------------------------------------------------------
//  Purpose:
//
//     CORDIC Traverse
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.22 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------
#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 (2R) children node to the current node pointed by p
//-----------------------------------------------------------------
int cordic_node(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<2*R; ++i) {
    if (i < R)        ntheta[i] = theta + 1 * a[depth];
```

```c
    else if (i == R)  ntheta[i] = theta - 1 * a[depth];
  }

  for (i=0; i<2*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] = np;
  np->parent        = p;
  np->theta         = ntheta[mindex];
  np->depth         = p->depth +1;
  np->branch        = mindex;
  np->id            = id++;


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

   return mindex;
}




//------------------------------------------------------------------
// CORDIC Traversal
//------------------------------------------------------------------
void cordic_traverse(double a[], nodetype *p) {
  qnodetype *q, *nq;
  int i, 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) i = cordic_node(a, q->node);

    nq = create_qnode();
    nq->node = (q->node)->child[i];

    cordic_tail->next = nq;
    cordic_tail = nq;

    q = nq;
  }

}
```

```
::::::::::::::
binary9_main.c
::::::::::::::
//-------------------------------------------------------------------------------
//  Purpose:
//
//     Ternary Angle Tree Search
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.22 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-------------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "binary1_search_defs.h"

extern qnodetype *minpath;
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 = create_node();
  p->theta = theta;
  p->depth = 0;
  tree_traverse(a, p);

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

  leaf = level_list_min_node(N-1);
  write_level_list(N-1);

  printf("* the optimal min path \n");
  find_minpath(leaf);
  list_path(a, minpath);

  printf("* the cordic path \n");
  cordic_traverse(a, p);
  list_path(a, cordic_path);

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


}
```