```
#define N 7     // the number of a tree
#define R N     // the number of expanding choices = 2*R+1



//-----------------------------------------------------------
// (2R+1)-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
// last choice   zero  at the step i  // 2*1+0 =2
//-----------------------------------------------------------
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+1]; // pointers to the 3 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;



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

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

void find_minpath(nodetype *p);
void list_minpath(double a[]);

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

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

```
//-----------------------------------------------------------------------
//  Purpose:
//
//     create node and qnode
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.08 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "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;
  }
}
```

```c
//-----------------------------------------------------------------------
//  Purpose:
//
//     Level Queue
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.08 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "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");

}

//---------------------------------------------------------------
// 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);
}
```

```
//-----------------------------------------------------------------------
//  Purpose:
//
//     Path Queue
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.08 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "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 the min path from root to node
//---------------------------------------------------------------
void list_minpath(double a[]) {
  qnodetype *q;
  int u, i;

  while (q) {
    printf("depth=%d ", (q->node)->depth);
    printf("theta=%f ", (q->node)->theta);
    i = (q->node)->depth;

    q = q->next;

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

```
    else if ((q->node)->branch < 2*R)   u = -1;  // ==1
    else if ((q->node)->branch == 2*R)  u = 0;   // ==2
    printf("u=%d ", u);
    printf("a[%d]=%f ", i, a[i]);
    printf("\n");


  }

}
```

```
//-------------------------------------------------------------------------
//  Purpose:
//
//     BFS Queue
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.08 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "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;
  }
```

```
}
```

```
//-----------------------------------------------------------------------
//  Purpose:
//
//     Tree Traverse
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.11 Thr
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "search_defs.h"

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



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

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

  theta = p->theta;

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

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


    if (fabs(ntheta) > fabs(theta)) {
       p->child[i]   = NULL;
       continue;
    }

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

    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);
```

```
  }

}




//-------------------------------------------------------------
// 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+1; ++i) {

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

  }

}
```

```c
//------------------------------------------------------------------------
//  Purpose:
//
//     Ternary Angle Tree Search
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2018.10.08 Mon
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "search_defs.h"



//---------------------------------------------------------------
// main - Ternary Angle Tree Search
//---------------------------------------------------------------
int main(void) {
  double a[N];
  // double angle[N+1];
  // double theta = 0.63761;
  // double theta = 0.27623;
  double theta; // = 4*atan(pow(2,-5));

  int i, s;

  nodetype *p;
  nodetype *leaf;

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



for (s=2; s<3; ++s) {

  theta = s * atan(pow(2, -5));
  // angle[0] = theta;

  printf("theta= %2d * atan(pow(2,-6) = %10g \n", s, theta);

  // conventional_cordic(a, angle);

  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);
  find_minpath(leaf);
  list_minpath(a);


}

}
```