Tartu Ülikool
Matemaatika-informaatikateaduskond
Arvutiteaduse Instituut

Kolmnurkade kahendpuu kahendkujul kirjutamise ja lugemise meetod

Kursuse “Algoritmid ja andmestruktuurid” II töö

Autor: Dan Bogdanov
Informaatika II aasta
Juhendaja: Ain Isotamm

Tartu 2002

Sisukord

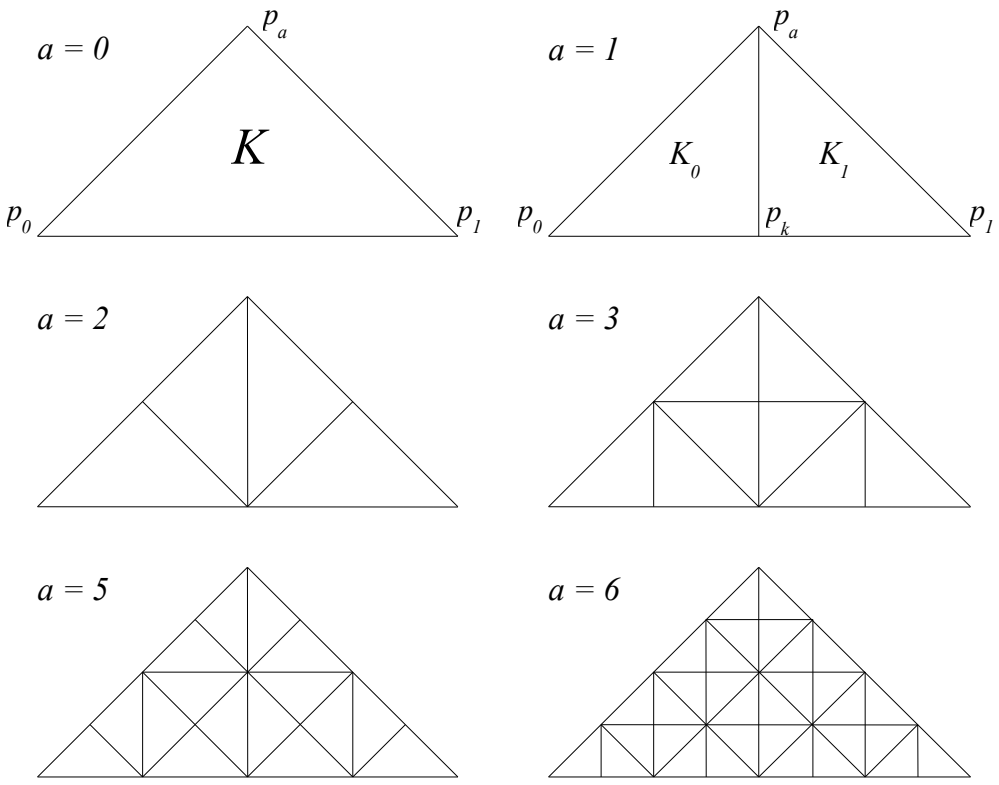
1. Sissejuhatus

Töös uuritakse meetodit, mis võimaldab kolmnurkade kahendpuud teisendada bitijadaks. Saadud jada on üheselt määratud ning sellest saab konstrueerida algse puu. Samuti on bitijada kergesti loetav ja salvestatav. Siinkirjeldatud meetod kasutab jada salvestamiseks bitthaaval salvestamist, et saavutada veelgi kompaktsem kuju.

2. Kolmnurkade kahendpuu esitus bitijadana

2.1 Kolmnurkade kahendpuu struktuur

Kolmnurkade kahendpuu on kahendpuu alamliik, kus tippudeks on täisnurksed võrdhaarsed kolmnurgad. Juurkolmnurga alluvkolmnurgad saame, kui jagame ta kõrgusega pooleks. Joonisel 1 on toodud kolmnurkade kahendpuu esimesed kuus astet. Puu tüvi ehk põhikolmnurk on $K = (p_a ; p_0 ; p_1)$, täisnurkne võrdhaarne kolmnurk kõige madalamal jaotuse astmel ($a = 0$). Järgmisel astmel ($a = 1$) on K poolitatud sirgega kolmnurga tipu p_a ja kolmnurga aluse ($p_0 ; p_1$) keskpunkti p_k vahel. $K_0 = (p_k ; p_a ; p_0)$ ja $K_1 = (p_k ; p_a ; p_1)$ on tüvikolmnurga K alluvkolmnurgad.



Joonis 1: Kolmnurkade kahendpuu astmed 0 - 5

2.2 Puu kodeerimine bitijadaks

Bitijadaks kodeerimisel kasutame ära kolmnurkade kahendpuu omadust, mille järgi igal alluvaid omaval tipul on kaks täpselt alluvat. Seega on igal tipul kaks olekut – tipul ei ole alluvaid või tipul on kaks alluvat. Vaatleme tippe järgnevalt: iga tipu t puhul $b(t)$ = {1, kui tipul t on kaks alluvat, 0 kui tipul t ei ole alluvaid}.

Koostame kahendpuu tippudest jada t_n . Tähistame positsiooni jadas $i = 0$. Esialgseks tipuks t' on puu juur, t'_v tähistab tipu t' vasakut ja t'_p paremat alluvat. Algoritm on järgnev:

1. Vaatleme antud tippu t'. Märgime $t_i = b(t')$ ja suurendame i väärtust ühe võrra.
2. a) Kui $b(t') = 1$, taaskäivitame protsessi rekursiivselt tipul t'_v ja seejärel tipul t'_p.
   b) Kui $b(t') = 0$, jätkame puu läbimist.

Kokkuvõttes läbitakse puu eesjärjestuses, kuid iga tipu juures märgitaks jada järgmisele positsiooni selle tipu olek. Saadav jada koosneb ühtedest ja nullidest ning ongi soovitud kahendkujul. Paneme tähele, et jada lõppu tekib olenevalt puu suurusest hulk nulle. Seda põhjustavad eesjärjestuse lõppu jäävad puu lehed, millel teatavastu ei ole alluvaid. Seega lisavad nad jadasse vaid nulle. Järelikult saame jadast t_n luua jada t'_m, kui leiame indeksi m nii, et kui $k > m$, siis $t_k = 0$. Jada t'_m ongi juurega t' määratud kahendpuu esitus bitijadana.

2.3 Puu konstrueerimine bitijadast

Puu konstrueerimiseks tuleb rakendada jada koostamisega sarnast algoritmi. Olgu antud jada t_n. Tähistagu $i = 0$ positsiooni jadas. Olgu antud konstrueeritava puu juur t.

Algoritm on järgnev:

1. Kui $i < n$, siis $b = t_i$, kui $i \geq n$, siis $b = 0$.
2. a) Kui b = 1, siis lisame tipule t kaks alluvat, vasaku (t_v) ja parema (t_p). Seejärel suurendame i väärtust ühe võrra ning taaskäivitame protsessi rekursiivselt tipul t_v. Kui see on lõppenud, taaskäivitame protsessi rekursiivselt tipul t_p.
   b) Kui b = 0, siis suurendame i väärtust ühe võrra.
3. Jätkame puu läbimist.

Vastavalt sisendjadale lisame puule tippe. Kui jada lõppeb, tõlgendame jada puuduvaid elemente nullidena. Töö lõpeb, kui puu muutub täielikuks.

3. Bitijada hoidmine kettal

Mälumahu kokkuhoidmiseks kodeerime bitijada kaheksa biti kaupa baitideks ning salvestame need baidid. Olgu antud kolmnurkade kahendpuud esindav bitijada t_n ja selle kaheksa-elemendiline osajada $t_{k...}t_{k+7}$. Seda osajada esindab arvutis ühebaidine märgita arv *r*, mis saadakse valemist

$$r = 2^0 * t_k + 2^1 * t_{k+1}\ 2^2 * t_{k+2} + 2^3 * t_{k+3} + 2^4 * t_{k+4} + 2^5 * t_{k+5} + 2^6 * t_{k+6} + 2^7 * t_{k+7} \quad .$$

Kuna ühebaidise märgita arvu maksimaalne väärtus on 2^8, siis $0 \leq r \leq 255$. Järelikult saab kõikvõimalikud kaheksa-elemendilised bitijadad kodeerida üheks baidiks.

Salvestamisel tuleb koostada bittidest järjest baite ning siis neid faili salvestada. Laadimisel jällegi peab sisse lugema baidimassiivi ja sealt järjest baithaaval bitte maha lahutama. Selline andmete salvestamismeetod on võrdlemisi tulus, sest n-tipulise puu puhul on vaja mitterohkem kui [n / 8] baiti. Mitterohkem seepärast, et faili lõppu jäävad nullid lõigatakse maha, säästes veel umbes viiendiku andmete mahust. Veelgi efektiivsema tulemuse annaks andmete kodeerimisel mõne pakkimisalgoritmi kasutamine.

4. Kokkuvõte

Esitatud meetod kolmnurkade kahendpuude salvestamiseks on tähtis just oma rakenduste pärast. Kolmnurkade kahendpuu on küllaltki kasutatav struktuur ruumigeomeetrias. Palju on see rakendust leidnud maastike töötlemisel – nii nende visualiseerimisel ja modelleerimisel kui ka optimaalsete teede otsingul. Kolmnurkade kahendpuu võib olla nii vahe- kui ka tööstruktuuriks. Igal juhul on mahukate kahendpuude talletamise meetodist kasu nendel juhtudel, kui nende puude konstrueerimine lähteandmetest oleks aeganõudvam protseduur.

5. Kasutatud kirjandus

[1] Kiho, J. “Algoritmid ja andmestruktuurid”, Tartu 1997

Lisa 1: Testprogrammi lähtekood keeles C++

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <dir.h>

/*
 *  Klass, mis tegeleb etteantud faili andmete bitthaaval
 *  kirjutamisega, puhverdades need kõigepealt mällu. Lõpus
 *  eemaldatakse faili lõpust ka kõik nullid, sest selle arvelt
 *  saab ruumi kokku hoida. Kui faili lõpp vastu tuleb, hakkab biti
 *  lugemise operatsioon nulle tagastama.
 */
#define BITFILE_BUFFERLENGTH 1000
#define BITFILE_CLOSED 0
#define BITFILE_READ 1
#define BITFILE_WRITE 2
class BitFile {
  public:
    BitFile ();
    ~BitFile ();
    bool OpenForReading (char* filename); // Ava fail lugemiseks
    bool OpenForWriting (char* filename); // Ava fail kirjutamiseks
    void Close ();                        // Sulge fail
    bool EndReached ();                   // Kas fail on otsas?
    bool ReadBit ();                      // Loe bitt
    void WriteBit (bool bit);             // Kirjuta bitt

  protected:
    void Flush ();                        // Puhver kettale
    void Truncate ();                     // Eemaldab lõpust nullid
    char* tempname;                       // Ajutise faili nimi
    char* name;                           // Faili nimi
    FILE* bitfile;                        // Failisang
    unsigned char* bytedata;              // Mälupuhver
    int bytes;                            // Puhvri indeks
    int bytesread;                        // Loetud baite kettalt
    int bitnum;                           // Aktiivne bitt
    int mode;                             // Faili olek
    int bitmask;                          // Bitimask
    int zerooffset;                       // Nullide algusaadress
    int offset;                           // Jooksev failiaadress
};

BitFile::BitFile ()
  : bitfile (NULL), tempname (NULL), name (NULL),
    bytedata (NULL), bytes (0), bitnum (0), bytesread (0),
    bitmask (1), zerooffset (-1), offset (0),
    mode (BITFILE_CLOSED)
{
  bytedata = (unsigned char*)malloc (BITFILE_BUFFERLENGTH);
  if (bytedata == NULL) {
    printf ("BitFile: viga mälu haaramisel\n");
  } else {
    memset (bytedata, 0, BITFILE_BUFFERLENGTH);
  }
}
```

```
BitFile::~BitFile () {
  Close ();
  if (bytedata != NULL)
    free (bytedata);
  if (name != NULL)
    free (name);
  if (tempname != NULL)
    free (tempname);
  bytedata = 0;
}

bool BitFile::OpenForReading (char* filename) {
  Close ();
  bitfile = fopen (filename, "rb");
  if (bitfile != NULL) {
    mode = BITFILE_READ;
    return 1;
  } else return 0;
}

bool BitFile::OpenForWriting (char* filename) {
  Close ();
  name = strdup (filename);
  tempname = tmpnam (NULL);
  bitfile = fopen (tempname, "wb");
  if (bitfile != NULL) {
    mode = BITFILE_WRITE;
    return 1;
  } else return 0;
}

void BitFile::Flush () {
  if (mode == BITFILE_WRITE) {
    if (bitnum == 0) fwrite(bytedata, 1, bytes, bitfile);
    else fwrite(bytedata, 1, bytes + 1, bitfile);
    bytes = 0;
  }
}

void BitFile::Truncate () {
  if (mode == BITFILE_WRITE) {
    if (zerooffset == -1)
      zerooffset = offset + 1;
    FILE* i = fopen (tempname, "rb");
    FILE* o = fopen (name, "wb");
    if (i == NULL || o == NULL) {
      printf ("BitFile: Truncation failed!");
      return;
    }
    int bytesleft = zerooffset;
    do {
      bytesread = fread(bytedata, 1, BITFILE_BUFFERLENGTH, i);
      if (bytesread <= bytesleft) {
        bytesleft -= bytesread;
      } else {
        bytesread = bytesleft;
        bytesleft = 0;
      }
      if (bytesread > 0)
        fwrite(bytedata, 1, bytesread, o);
    } while (bytesleft != 0);
    fclose (i);
    fclose (o);
```

```
    unlink (tempname);
    free (tempname);
    free (name);
  }
}

void BitFile::Close () {
  Flush ();
  if (bitfile != NULL) {
    fclose (bitfile);
  }
  Truncate ();
  bitnum = 0;
  bitmask = 1;
}

bool BitFile::EndReached () {
  if (feof (bitfile) && bytes == bytesread) return 1;
  else return 0;
}

bool BitFile::ReadBit () {
  if (EndReached ())
    return 0;
  if (bytes == bytesread) {
    memset (bytedata, 0, BITFILE_BUFFERLENGTH);
    bytesread = fread(bytedata, 1, BITFILE_BUFFERLENGTH, bitfile);
    if (bytesread == 0) {
      return 0;
    }
    bitnum = 0;
    bytes = 0;
    bitmask = 1;
  }
  bool bit = bytedata[bytes] & bitmask;
  bitnum++;
  bitmask <<= 1;
  if (bitnum == 8) {
    bytes++;
    bitnum = 0;
    bitmask = 1;
  }
  return bit;
}

void BitFile::WriteBit (bool bit) {
  if (bytes == BITFILE_BUFFERLENGTH) {
    if (fwrite(bytedata, 1, bytes, bitfile) == bytes) {
      memset (bytedata, 0, BITFILE_BUFFERLENGTH);
      bytes = 0;
      bitnum = 0;
      bitmask = 1;
    } else {
      printf ("BitFile: viga faili kirjutamisel!\n");
    }
  }
  if (bit == 1)
    bytedata[bytes] += bitmask;
  bitnum++;
  bitmask <<= 1;
  if (bitnum == 8) {
    if (bytedata[bytes] == 0) {
      if (zerooffset <= 0)
          zerooffset = offset;
```

```
    } else {
      zerooffset = -1;
    }
    bytes++;
    offset++;
    bitmask = 1;
    bitnum = 0;
  }
}

/* Kahendpuu tippu esindav klass */
class Node {
  public:
    Node ();
    Node (Node* l, Node* r);
    Node* leftchild;         // Viit vasakule alluvale
    Node* rightchild;        // Viit paremale alluvale
};

Node::Node () {
  leftchild = NULL;
  rightchild = NULL;
}

Node::Node (Node* l, Node* r) {
  leftchild = l;
  rightchild = r;
}

/*
 * Puuoperatsioone (failist lugemist ja faili kirjutamist ning
 * väljastust) sisaldav klass. Samuti on lisatud meetod pseudo-
 * juhusliku kolmnurkade kahendpuu genereerimiseks.
 */
class TreeFile {
  public:
    TreeFile ();
    ~TreeFile ();
    Node* ReadTree (char* filename);           // Failist lugemine
    void WriteTree (char* filename, Node* root);  // Salvestamine
    void BitPrint (Node* node); // Puu bittkuju väljastamine
    void GenTree (Node* root);  // Puu genereerimine
  protected:
    void WriteNode (Node* root);
    Node* ReadNode ();
    BitFile* file;
    int limit;             // Genereeritava puu tippude piirarv
};

TreeFile::TreeFile () : file (NULL), limit (50) {}
TreeFile::~TreeFile () {}

void TreeFile::WriteTree (char* filename, Node* root) {
  file = new BitFile ();
  if (!file->OpenForWriting (filename)) {
    printf ("TreeFile: viga faili '%s' avamisel!\n", filename);
    return;
  }
  WriteNode (root);
  file->Close ();
}

void TreeFile::WriteNode (Node* node) {
  if (node->leftchild != NULL && node->rightchild != NULL) {
```

```
    file->WriteBit (1);
    WriteNode (node->leftchild);
    WriteNode (node->rightchild);
  } else {
    file->WriteBit (0);
  }
}

Node* TreeFile::ReadTree (char* filename) {
  file = new BitFile ();
  if (!file->OpenForReading (filename)) {
    printf ("TreeFile: viga faili '%s' avamisel!\n", filename);
    return NULL;
  }
  Node* root = ReadNode ();
  file->Close ();
  return root;
}

void TreeFile::GenTree (Node* node) {
  if (limit <= 0)
    return;
  int a = rand () % 100;
  if (a < 80) {
    limit -= 2;
    node->leftchild = new Node ();
    GenTree (node->leftchild);
    node->rightchild = new Node ();
    GenTree (node->rightchild);
  }
}

Node* TreeFile::ReadNode () {
  Node* node = new Node ();
  bool split = file->ReadBit ();
  if (split) {
    node->leftchild = ReadNode ();
    node->rightchild = ReadNode ();
  }
  return node;
}

void TreeFile::BitPrint (Node* node) {
  if (node->leftchild != NULL && node->rightchild != NULL) {
    printf ("1");
    BitPrint (node->leftchild);
    BitPrint (node->rightchild);
  } else {
    printf ("0");
  }
}

/* Põhifunktsioon */
int main(int argc, char *argv[])
{
  srand (time (NULL));
  printf ("Koostan kolmnurkade kahendpuu puu...\n");
  Node* root = new Node (new Node (new Node (new Node (), new Node
()), new Node ()), new Node (new Node (), new Node (new Node (), new
Node ())));
  TreeFile* tf = new TreeFile ();
  printf ("Loodud puu bittkuju: ");
  tf->BitPrint (root);
  printf ("\n");
```

```
  printf ("Salvestan puu faili 'test.tre'...\n");
  tf->WriteTree ("test.tre", root);
  printf ("Loen puu failist 'test.tre'...\n");
  Node* reload = tf->ReadTree ("test.tre");
  printf ("Loetud puu bittkuju: ");
  tf->BitPrint (reload);
  printf ("\n");
  printf ("Genereerin suvalise kolmnurkade kahendpuu...\n");
  Node* random = new Node ();
  tf->GenTree (random);
  printf ("Saadud puu bittkuju: ");
  tf->BitPrint (random);
  printf ("\n");
  printf ("Salvestan puu faili 'suvaline.tre'...\n");
  tf->WriteTree ("suvaline.tre", random);
  printf ("Loen puu failist 'suvaline.tre'...\n");
  Node* randomreload = tf->ReadTree ("suvaline.tre");
  printf ("Loetud puu bittkuju: ");
  tf->BitPrint (randomreload);
  printf ("\n");
  delete tf;
}
```

Lisa 2: Testprogrammi väljund

Test 1

```
Koostan kolmnurkade kahendpuu puu...

Loodud puu bittkuju: 11100010100

Salvestan puu faili 'test.tre'...

Loen puu failist 'test.tre'...

Loetud puu bittkuju: 11100010100

Genereerin suvalise kolmnurkade kahendpuu...

Saadud puu bittkuju:
11110101110101011111111111011101000000000000000000000

Salvestan puu faili 'suvaline.tre'...

Loen puu failist 'suvaline.tre'...

Loetud puu bittkuju:
11110101110101011111111111011101000000000000000000000
```

Test 2

```
Koostan kolmnurkade kahendpuu puu...

Loodud puu bittkuju: 11100010100

Salvestan puu faili 'test.tre'...

Loen puu failist 'test.tre'...

Loetud puu bittkuju: 11100010100

Genereerin suvalise kolmnurkade kahendpuu...

Saadud puu bittkuju:
11111110111011100111111111110100000000000000000000000

Salvestan puu faili 'suvaline.tre'...

Loen puu failist 'suvaline.tre'...

Loetud puu bittkuju:
11111110111011100111111111110100000000000000000000000
```