Puud

Lineaarsed struktuurid ei ole alati sobivad andmestruktuurid tegelike andmete ja nende omavaheliste seoste modelleerimiseks (esitamiseks). Andmete omavahelised seosed võivad tegelikkuses olla keerulisemad, kui lihtsalt eelnev-järgnev seos. Kõige üldisem struktuur erinevate seoste modelleerimiseks on **graaf** *(ingl graph),* mida vaatleme edaspidi. Graafi erivormiks võib pidada lineaarset jada, aga samuti ka **puud** *(ingl tree)*.

Seega siis puu on üldisem andmestruktuur kui loend, puu on mittelineaarne ja hierarhiline. Puu koosneb elementidest, mida nimetatakse **tippudeks** ehk **sõlmedeks** *(ingl node)*, ja seostest tippude vahel, mida nimetatakse **kaarteks** *(ingl edge)*. Andmed paigutatakse tippudesse. Andmete paigutamisel võib olla mitmeid erinevaid põhimõtteid. Kaks puud ei ole ühendatud, kui neil puuduvad ühised sõlmed ja kaared. **Triviaalsel puul** *(ingl trivial tree)* puuduvad tipud ja seega ka andmed. Ka üksik sõlm moodustab puu.

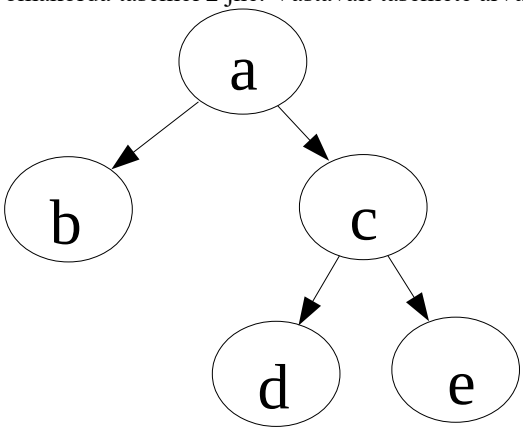
Puu definitsioon Donald Knuth'i järgi on järgmine:

> **Puu** *(ingl tree)* on lõplik hulk T, mis koosneb ühest või mitmest sõlmest. Sõlmed rahuldavad järgmisi tingimusi:
>
> a) eksisteerib üks, teistest erinev sõlm, mis on selle puu **juur** *(ingl root)*
>
> b) teised sõlmed (väljaarvatud juur) jagunevad `m (m >= 0)` mittelõikuvaks alamhulgaks $T_1 \ldots T_m$ ja iga alamhulk on omakorda puu. Hulki $T_1 \ldots T_m$ nimetatakse antud puu **alampuudeks** *(ingl subtree)*

NB! Antud definitsioon on rekursiivne: mõiste **puu** kirjeldamiseks kasutatakse mõistet **puu**. Ja puu ongi rekursiivne struktuur – iga puu sees on puu või puud. Eriolukorraks saab pidada ühest sõlmest koosnevat puud - puu juurt. Teatud liiki puudel võivad esineda ka tühjad alampuud.

Mõisted

Iga puu sõlm on juureks mõnele alampuule. Sõlme kõigi alampuude arvu nimetatakse **sõlme järguks** *(ingl degree)*. Sõlm, mille järk on 0, on **leht** *(ingl leaf)*. Ülejäänud sõlmed on **hargnevad sõlmed** *(ingl branch node)*. Puu sõlmed jagunevad paiknemishierarhia järgi **tasemetesse** *(ingl level).* Juur on tasemel 0, juure lapsed on tasemel 1, nende lapsed omakorda tasemel 2 jne. Vastavalt tasemete arvule mõõdetakse ka **puu kõrgust**. (vt Joonis 1)



Joonis 1: Puu (ka kahendpuu, teist järku puu - pildi järgi ei ole võimalik eristada). Puu kõrgus on 2.

Puu sõlmedesse paigutatakse informatsioon. Seosed sõlmede vahel (sõlmede hierarhia) näitab seoseid sõlmedes oleva informatsiooni vahel.

Puu on **järjestamata** *(ingl unordered)*, kui ühe tipu laste omavaheline järjestus ei ole määratud. Puu on **järjestatud** *(ingl ordered)*, kui ühe tipu laste järjestus on mingil alusel määratud ja võib rääkida esimesest, teisest jne pojast/lapsest.

Enamasti, kui räägitakse puust, siis peetakse silmas **orienteeritud puud** *(ingl oriented tree)*. Orienteeritus tähendab, et puus on hierarhilised seosed, et seostel on suund. Orientatsioon on suunaga juurest lehtede poole. Veel kasutatakse mõistet on **juurega puu** *(ingl rooted tree),* mis tähendab seda, et puu tippude hulgast on välja toodud üks tipp, mis on kogu puu juureks. Sellest tipust (puu juurest) lähtub kogu puu hierarhia, tasemete lugemine ja muu. Valdavalt on järgnevalt juttu juurega puust.

Põhimõtteliselt võib olla puustruktuur ka selline, kus otseselt puu juurt ei fikseerita. Sel juhul kehtib reegel, et andmestruktuuris ei ole tsüklit ja kahe puu tipu vahel on maksimaalselt üks tee. Sellised puud seostuvad pigem graafidega (näiteks graafi toesepuu ehk aluspuu). Üldiselt kui räägitakse puust, siis peetaksegi silmas juurega puud.

Mets *(ingl forest)* on järjestatud hulk, mis koosneb nullist või mitmest mittelõikuvast puust. Seega kui eemaldada puu juur, tekib tema alampuudest mets. Lisades metsa uue sõlme ja ühendades tema külge kõik puud, saame ühe puu.

> Puud, mille kõigil sõlmedel on maksimaalne laste arv piiratud arvuga n, nimetatakse **n-järku puuks**.

Puu sõlmede nimetamiseks ja nende omavaheliste suhete kirjeldamiseks on palju mõisteid, enamus neist seotud sugupuuga. Osa autoreid kasutab ka matriarhaalseid väljendeid (emad, tütred, tädid, jms). Et oleks ikka poliitiliselt korrektne. Lisaks tuleb arvestada, et kuna puud kujutatakse reeglina juur ülal ja lehed alla, siis võib üsna julgelt rääkida paiknemisest ülalpool ja/või allpool..

- Puul on üks **sõlm** *(ingl node)*, mida kutsutakse **juureks** *(ingl root).* Juurel ei ole ühtegi eellast.
- Kõiki sõlmi, millel on järglased või lapsed, nimetatakse **vanemateks** (*ingl parent nodes, nonterminal nodes)*
- Igal sõlmel (va juur) on täpselt **üks vanem.**
- Sõlmi, millel on vanemad, nimetatakse **lasteks** *(ingl child nodes, siblings*)
- Sõlmi, millel järglased puuduvad, nimetatakse **lehtedeks** *(ingl terminal node, leaf).*
- Sõlmi, millel on sama vanem, nimetatakse **vendadeks** *(ingl brother)*
- Kõik antud sõlmest kõrgemal (juure pool) olevad sõlmed on sõlme **eellased** *(ingl ancestor)*.
- Kõik antud sõlmest allpoololevad sõlmed on sõlme **järglased** *(ingl descendant)*.

> **Tee** *(path)* on ainus, lühim kaarte järgnevus, mis viib puu juurest leheni. Puu juure ja konkreetse lehe vahel on alati ainult üks tee. Teena võib vaadelda ka kaarte järgnevust juurest suvalise sõlmeni.

Puude ülesmärkimine

Kõige piltlikum on puu sõlmede ja kaartena välja joonistada (nagu joonisel 1). Sealjuures on tavaks, et juur joonistatakse üles ja lehed alla. See on oluline, sest tihti kasutatakse tippudest rääkides mõisteid ülal ja all, vasakul ja paremal.

Teksti kujul ülesmärkimiseks on järgnevad moodused:

a) **sulgavaldisena** – puu erinevad tasemed sulustatakse ja joonisel 1 oleva puu saaks kirja panna järgmiselt: `(a(b)(c(d)(e)))`. Selles kirjaviisis kajastub ka ühe vanema laste järjestatus, st sulgavaldis on sobiv nii järjestatud kui järjestamata puu kirjeldamiseks.

b) **Dewey kümnendesitusena** *(ingl Dewey decimal notation)* – süsteem on iseenesest tuttav, sarnanedes raamatu peatükkide nummerdusele. Ka selles süsteemis kajastub nii järjestus kui hierarhia.
Joonisel 1 olev puu näeb antud esituses välja järgmine: `1 a; 1.1 b; 1.2 c; 1.2.1 d; 1.2.2 e`.

Tekstikujuline kirjaviis võib aidata näiteks puud salvestada.

Kahendpuu

Kahendpuu *(ingl binary tree)* on üks enim kasutatavaid "puuliike". Kahendpuu igal sõlmel on maksimaalselt kaks alampuud (olles sel viisil teist järku puu). Kuid erinevalt teist järku puust tehakse iga alampuu puhul ranget vahet, kas ta on vasakpoolne või parempoolne. Seega ei saa võrdsustada teist järku puud (igal tipul maksimaalselt kaks järglast) kahendpuuga.

Kahendpuu definitsioon:

> **Kahendpuu** *(ingl binary tree)* on tippude lõplik hulk, mis on tühi või mis koosneb juurest ja kahest mittelõikuvast alampuust, mida nimetatakse antud juure **vasakuks** ja **paremaks alampuuks** *(ingl left and right subtree)*.

Oluline erinevus võrreldes tavalise puuga on veel see, et kahendpuu puhul peetakse ka tühja alampuud puuks. Tavalise puu puhul tühjast alampuust ei räägita. Kahendpuus tehakse vahet vasaku ja parema alampuu vahel, st pole tähtis laste

omavaheline järjestus, vaid vasak võib puududa ja parem olemas olla. Seega ei saa väita, et kahendpuu oleks järjestatud.

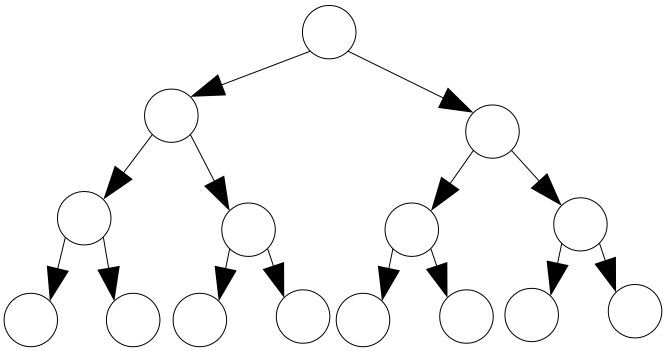
Kahendpuu on **täielik** *(ingl perfect / complete)*, kui tema kõigil tasemetel on maksimaalne võimalik arv sõlmi ja kõik lehed paiknevad samal tasemel (vt Joonis 2). Võib defineerida ka nii: kahendpuu on **täielik**, kui kõik tema lehed paiknevad ühel tasemel ja kõigil ülejäänud tippudel on kaks last. Peaagu täielikus kahendpuus võivad lehed puududa vaid viimasel tasemel nõ paremalt poolt.

Täielikus kahendpuus on võimalik tippude arvu järgi leida puu kõrgust ja veel mõningaid näitajaid.

Puu kõrgus: kui puus on `M` tippu, siis puu kõrgus on $\log_2(M+1)$.

Tippude arv ja lehtede arv: puus kõrgusega `h` on $2^h - 1$ tippu ja 2^{h-1} lehte

Võimalik on välja arvutada puu **tippude arv mingil tasemel**: tasemel `N` on 2^N tippu.



Joonis 2: Täielik kahendpuu

Puude kasutamine

Puukujulisi struktuure elust. Sugupuude ehitamisel saab joonistada puud kahes suunas: eellaste puud ja järglaste puud. Esimene neist (nn kõukude tabel) on kahendpuu, teine mitte. Tõsi küll – sugupuus võivad tekkida lõikuvad alampuud, kui on toimunud näiteks sugulusabielud ja seetõttu peaksime rääkima hoopis graafist. Aga visuaalselt esitatakse seda siiski puuna. Spordis kasutatakse olümpiasüsteemis võistluste tabelit (kaotaja langeb välja – seda tabelit hakatakse erandina lehtedest ehitama). Raamatuid (ennekõike teadustekste) jagatakse alapeatükkidesse, mille loogikas võime ka näha puu kuju. Kindlasti leiame veel näiteid, kus täiesti tavapärane info on puukujuline.

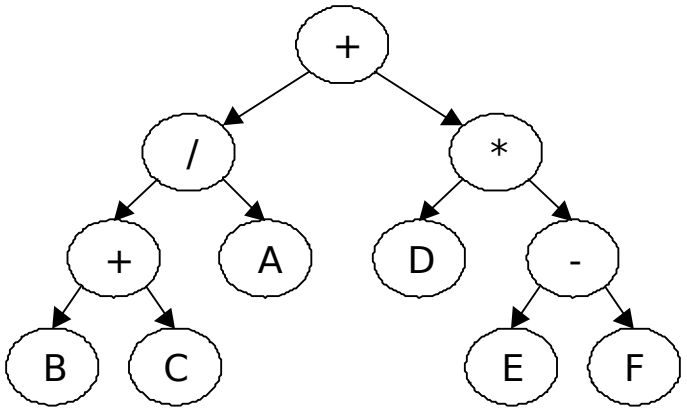
Puid kasutatakse arvuti mälus andmestruktuurina: on otsustamispuud, süntaksipuud, koodipuud jne Lisaks räägitakse veel kataloogipuust, kuhu arvuti kõvakettale pandud materjali organiseerida saab.

Puude abil saab esitada ka sellist infot, mis esmapilgul kohe puuna ei paista.

Mõned näited puudest:

- Arvutiteaduses on näiteks aritmeetikaavaldised esitatavad kahendpuuna.
- Kahendpuusse saab paigutada sõnad tähestiku järjekorras ja siis nende hulgast kiiresti otsida.
- Puustruktuur sobib juhul, kui on oluline informatsiooni hierarhia (näiteks sugupuud või siis hoopis mõne organisatsiooni ülesehitus). Ei pruugi olla kahendpuu.
- n-järku puud võib aga kasutada ka hoopis selliselt, et igas sõlmes ei ole täielik info mõne objekti kohta vaid hoopis üks osa (näiteks täht). Terve sõna (väärtus) saadakse aga kokku puud juurest kuni leheni läbides, näiteks kontrollimaks sõnade leidumist sõnastikus. Kui leheni jõudes sellist vastet ei leita, siis järelikult sellist sõna pole. Sellist puud nimetatakse inglise keeles *trie*, sõnast *retrie.*
- Kahendpuusse saab paigutada morsetähestiku kodeeringu tänu sellele, et kood koosneb kahest erinevast sümbolist. See puu on *trie* näide.
- Näitena tehtud nn loomapuu on aga tüüpilise otsustuspuu näiteks. Igal järgmisel sammul tehakse otsus mingi variandi kasuks (millega mitmed võimalikud vastused välistatakse) ja lõpuks jõutakse vastuseni.
- Male või mõne muu mängu käikude analüüsimiseks peaks ka puu sobima (tipp on antud käik ja tema järglased kõik võimalikud järgmised käigud). Sellits analüüsipuud saavad ülesehitada malet mängivad programmid.

Joonisel (vt Joonis 3), on näha kahendpuu, mille abil kujutatakse aritmeetikaavaldist `(b+c)/a+d(e-f)`.



Joonis 3: Kahendpuu, mis kujutab aritmeetikaavaldist

Operatsioonid

Edaspidi peetakse puu all silmas kahendpuud, kui ei ole väidetud midagi muud.

Peamised tegevused, mida puuga ette võetakse, on

- uue sõlme lisamine vastavalt mingitele reeglitele (enamasti leheks, kuid võimalik ka kuhugi vahepealsele tasemele)
- sõlme kustutamine
- kogu puu läbimine info kättesaamiseks (alustades juurest)
- tüüpiliselt tähendab puust info otsimine puu sõlmede läbimist alustades juurest ja liikudes ühte teed pidi mingi leheni, info võidakse kätte saada ka enne leheni jõudmist.

Puu sõlmedest informatsiooni lugemiseks tuleb need sõlmed kuidagi **läbida** *(ingl traverse).* Erinevalt pinust ja järjekorrast ei tähenda info puust lugemine ühtlasi vastava sõlme kustutamist. Kuna puu on oma olemuselt rekursiivne struktuur, siis on seda ka mitmed algoritmid, mida puule rakendada saab (st rekursiivselt on need algoritmid kõige lihtsamalt ja kaunimalt kirja pandavad, kuid saab ka ilma hakkama). Tegevus rakendatakse rekursiivselt igale alampuule. Puu läbimisel käiakse igas sõlmes täpselt üks kord ja saadakse sõlmedes olevatest väärtustest mingi lineaarne järgnevus või töödeldakse sõlmi vastavas järjekorras. Läbimise käigus puust midagi ei eemaldata.

Kogu puu läbimise kolm klassikalist järgnevust on:

1. **Lõppjärjekord** *(Postorder e. Endorder)*
   1. Läbi vasak alampuu.
   2. Läbi parem alampuu.
   3. Väljasta (töötle) juur (tegevust korratakse iga alampuu jaoks).

   Sellise tegevuse tulemusena saab pildil 1 oleva puu väljundiks `b d e c a`: ja pildil 2 oleva puu väljundiks: `B C + A / D E F - * +`. Vanema ja tema kahe järglase kohta kehtib seega järgmine läbimise järjestus: vasak järglane, parem järglane, vanem.

2. **Eesjärjekord** *(Preorder)*
   1. Väljasta (töötle) juur.
   2. Läbi vasak alampuu.
   3. Läbi parem alampuu (igal alampuul on oma juur, mida väljastada).

   Puu pildil 1 annab sel juhul tulemuseks `a b c d e` ja puu pildil 2 vastavalt `+ / + B C A * D - E F`. Vanema ja tema kahe järglase kohta kehtib seega järgmine läbimise järjestus: vanem, vasak järglane, parem järglane.

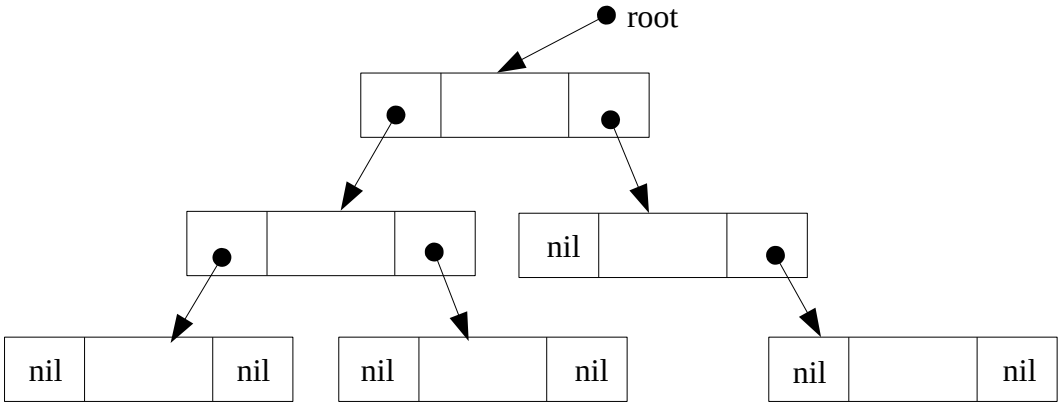
3. **Keskjärjekord** *(Inorder)*
   1. Läbi vasak alampuu.
   2. Väljasta (töötle) juur.
   3. Läbi parem alampuu (loomulikult on igal alampuul jälle oma juur, mida väljastada).

Puu pildil 1 annab tulemuseks: `b a d c e` ning puu pildil 2 annab vastuseks: `B + C / A + D * E - F`. Vanema ja tema kahe järglase kohta kehtib seega järgmine läbimise järjestus: vasak järglane, vanem, parem järglane.

NB! Kui puud läbitakse, siis kokkuleppeliselt minnakse enne vasakusse alampuusse ja seejärel paremasse alampuusse, ehkki saaks teha ka vastupidi.

Kahendpuu realisatsioon (keel C)

Puu on kujutatav sarnaselt juba tuttavate andmestruktuuridega nii staatiliselt massiivina kui ka dünaamiliselt. **Dünaamiline** realisatsioon on eelistatud, sest see ei nõua esialgset suurt mälu eraldamist ja on ka loomulikum. Puu iga sõlm sisaldab lisaks infole kahte viida- ehk aadressitüüpi välja: `LLINK` ja `RLINK`. Puuga on seotud viit puu juurele `root` ehk puu juure aadress (vt Joonis 4). Kui puu on tühi, on `root == NULL`. Vastasel juhul on `root` väärtuseks puu juure aadress. Kui mingi sõlme üks alampuudest on tühi, kirjutatakse vastavatesse viidaväljadesse tühja viida tähis `NULL`



Joonis 4: Kahendpuu realisatsioon (nil - tühi viit ehk C-keelne NULL)

C keeles saab puu sõlme jaoks kirjeldada järgmise struktuuri:

```
struct node {
       int key;
       struct node *llink, *rlink;
            }
struct node *root;
```

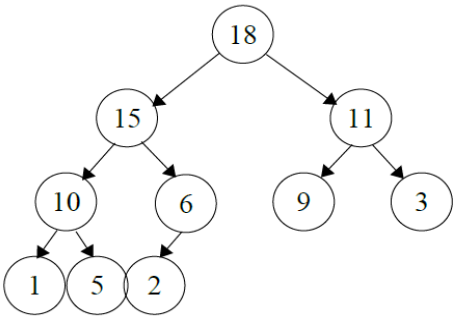
Lahtiseletatuna: puu sõlmes on kaks aadressi- ehk viidavälja (vasakule alampuule `llink` ja paremale alampuule `rlink`) ja täisarvuline võtmeväli `(key)`. Muutuja deklaratsioon kirjeldab puu juure, kui viida sellist tüüpi sõlmele `(root)`. Puu juure aadress peab olema puuga töötades alati meeles, muidu kaob võimalus kogu puud veel mälust üles leida.

Kahendpuu staatiline realisatsioon tehakse massiivi kasutades. Sõlmede omavahelised seosed määratakse indeksite kaudu. Selleks on mitu võimalust. On võimalik hoida massiivi elemendis lisaks infole ka kummagi järglase asukoha indeksit (sarnaselt dünaamilisele aadressile). Teine võimalus on info paigutada massiivi nii, et laste ja vanemate indeksite vahel kehtivad teatud reeglid ning neid indekseid on võimalik väljaarvutada.

Puu sõlmede info paigutatakse massiivi nii, et juur on esimene, talle järgnevad juure lapsed jne. Indeksi `i` kohal oleva elemendi lähimate naabrite indeksid saab leida järgmiselt (eeldusel, et puu juure indeks on 1):

- elemendi `i` vasak laps paikneb indeksil `2*i`
- elemendi `i` parem laps paikneb indeksil `2*i + 1`
- elemendi `i` vanem paikneb indeksil `i / 2` (NB! Täisarvuline jagamine!)

Sorteerimise materjalis on massiivis paikneva puustruktuuri kohta järgmine näide (vt Joonis 5)



Joonis 1 Kahendkuhi puuna

1	2	3	4	5	6	7	8	9	10
18	15	11	10	6	9	3	1	5	2

Joonis 5: Kahendpuu ja vastav massiiv

Puu läbimine

Järgnev rekursiivne funktsioon läbib puu **inorder**-järjestuses ning trükib välja läbitud sõlmes oleva võtme väärtuse.

```
/* Funktsioon läbib puu inorder-järjekorras
   Funktsiooni sisendiks on sõlme aadress, esimesel väljakutsel juure,
   järgmistel väljakutsetel järgmiste sõlmede oma. */
void Inorder(struct node *current) {
   if (current->llink != NULL) {
       Inorder(current->llink);
   }
   printf("%d", current->key);
   if (current->rlink != NULL) {
       Inorder(current->rlink);
   }
}
```

Postorder ja **preorder** läbimine kulgevad sama skeemi järgi. Tuleb vaid väljatrüki asukohta muuta: preorder järjekorra puhul esimese `if-i` ette ja postorder järjekorra puhul teise `if-i` taha.

Puu loomine

Sõltuvalt puu kasutamise eesmärgist ja selles sisalduvast infost saab puud mitmel erineval viisil konstrueerida, st millisel viisil andmed puu sõlmedesse paigutatakse. Ühe ja väga olulise meetodiga puutume kokku kahendotsingupuu juures. Siin aga olgu toodud variant, kus massiiviga etteantud sõlmed paigutatakse puusse nii, et puu kõrgus oleks minimaalne (st tühje alampuid saab olla vaid viimasel tasemel) ja puu on ideaalses tasakaalus. Puud ehitades võetakse väärtused järjest massiivist ning esimene väärtus satub puu juureks, järgmine juurele vasakuks alampuuks, ülejärgmine sümbol eelmisele tipule vasakuks alampuuks jne.

Puu kõrguse määramiseks peab aga sel juhul olema eelnevalt teada lisatavate sõlmede arv `n`.

Rekursiivne algoritm on järgmine:

1. Võta üks sõlm juureks
2. Ehita rekursiivselt vasak alampuu `nl` elemendist, kus `nl = n / 2`.
3. Ehita rekursiivselt parem alampuu `nr` elemendist, kus `nr = n - nl - 1`.

Funktsioon C-keeles on allpool. Ta kasutab eelnevalt kirjeldatud sõlme kuju ja väljade nimesid. Tuleb aru saada, et funktsioon ei ole üksinda kasutatav, vaid ta tuleb välja kutsuda, andes ette puusse pandavate elementide arvu, samuti peab olema olemas massiiv `andmed`, kus võetakse puusse lisatavad väärtused. Sama funktsiooni nägime morsepuu ülesehitamisel.

```
struct node *EhitaPuu(int n) // n on puu sõlmede arv
    struct node *uus;
    int nl, nr;
    if (n==0) return NULL;
    else {
          nl = n / 2;
          nr = n - nl - 1;
          uus = malloc(sizeof *uus);
// "andmed" imiteerib massiivi, kust sõlmedesse väärtused võetakse
          uus->key = andmed[i];
          i++;
          uus->llink = NULL;
          uus->rlink = NULL;
//Järgnevad 2 rekursiivset sama funktsiooni väljakutset
          uus->llink = EhitaPuu(nl); //nl on tippude arv vasakus alampuus
          uus->rlink = EhitaPuu(nr); //nr on tippude arv paremas alampuus
          return uus;
     }
}
```

Kustutamine

Üks võimalus on puu tervikuna kustutada

Muutuvate andmete puhul võib olla vajadus puud nii kasvatada kui ka kahandada. Sõlme kustutamine juhul, kui tegemist on puu lehega, on lihtne, ülejäänud sõlmede puhul muutub algoritm keerulisemaks ja sõltub sellest, millistel põhimõtetel puu on ehitatud (kas peab säiluma tippude järjestus nagu kahendotsingupuus vms). Kahendotsingupuust tuleb lähemalt juttu otsimisele pühendatud materjalis ja loengus.

Üldise puu kujutamine kahendpuuna

Alati ei sobi andmete hoidmiseks kasutada kahendpuud või teist järku puud. Samal ajal on arvutis mugavam iga puud kujutada 2. järku puuna (kui mitte kahendpuuna). Vastasel juhul on võimatu määrata, mitu viidavälja igasse sõlme teha tuleks. Kahendpuuna kujutamise idee on järgmine: Kaotatakse nii palju vertikaalseid linke kui võimalik. Vanemaga jäetakse vertikaalsuunas seotuks vaid 1. laps, ülejäänud lapsed seotakse üksteise külge paremat viita pidi ehk vennad on järjest üksteisele justkui parematekks alampuudeks.