



Деревья

Добрый доктор Айболит, он под деревом сидит
К.Чуковский. Айболит

- Рекурсивная структура данных: вершина + ссылки на поддеревья
- Терминология: корень, предок, потомок (отец, сын)
- Рекурсивный обход дерева: функция для вершины вызывает себя для потомков
- Жадный алгоритм (ветвление) – функция для вершины выбирает единственного потомка (цикл, линейная рекурсия)





Представление деревьев

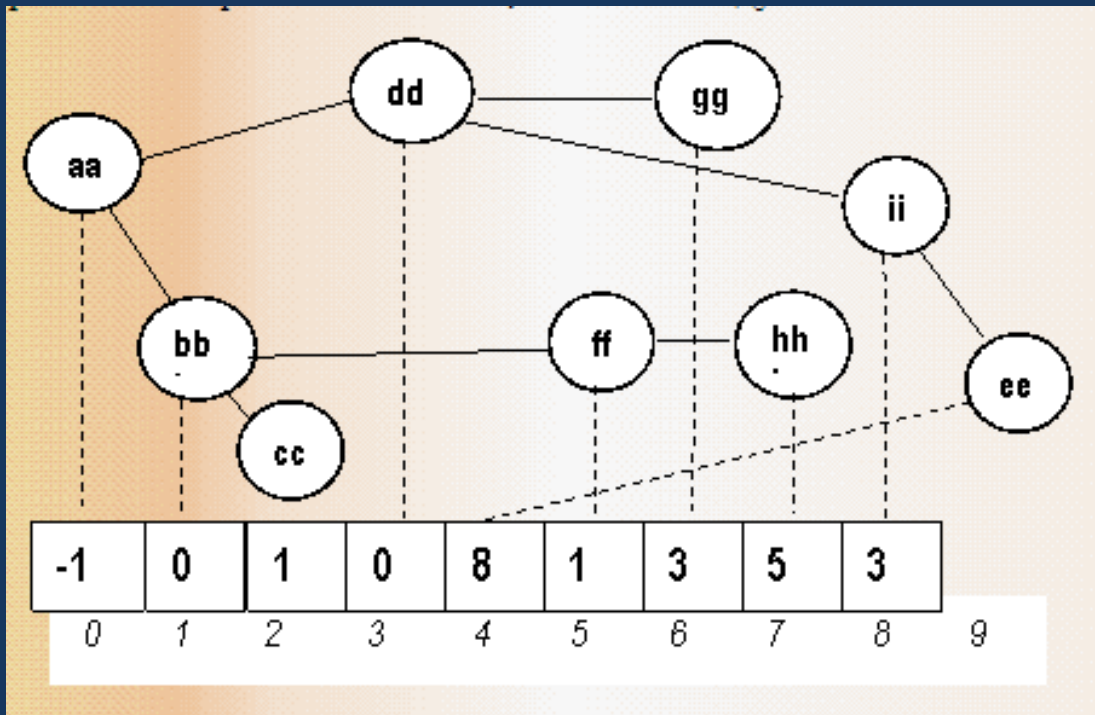
Дерево как логическая СД, формы представления – физическая СД

- если используется рекурсивный или циклический алгоритм, начинающий работать с корневой вершины дерева, то необходимы только прямые ссылки от предка к потомкам;
- если алгоритм предполагает навигацию по дереву во всех направлениях, как вверх, так и вниз по дереву (например, в древовидной системе каталогов), то предполагается наличие как прямых, так и обратных ссылок от потомков к предкам (в системе каталогов – ссылка на родительский каталог)
- возможны алгоритмы, которые работают с деревом, начиная с терминальных вершин. Тогда кроме ссылок от потомков к предкам необходима еще структура данных, объединяющая терминальные вершины (например, массив указателей).



Представление деревьев

Дерево в массиве с индексами предков (представление в реляционной БД)



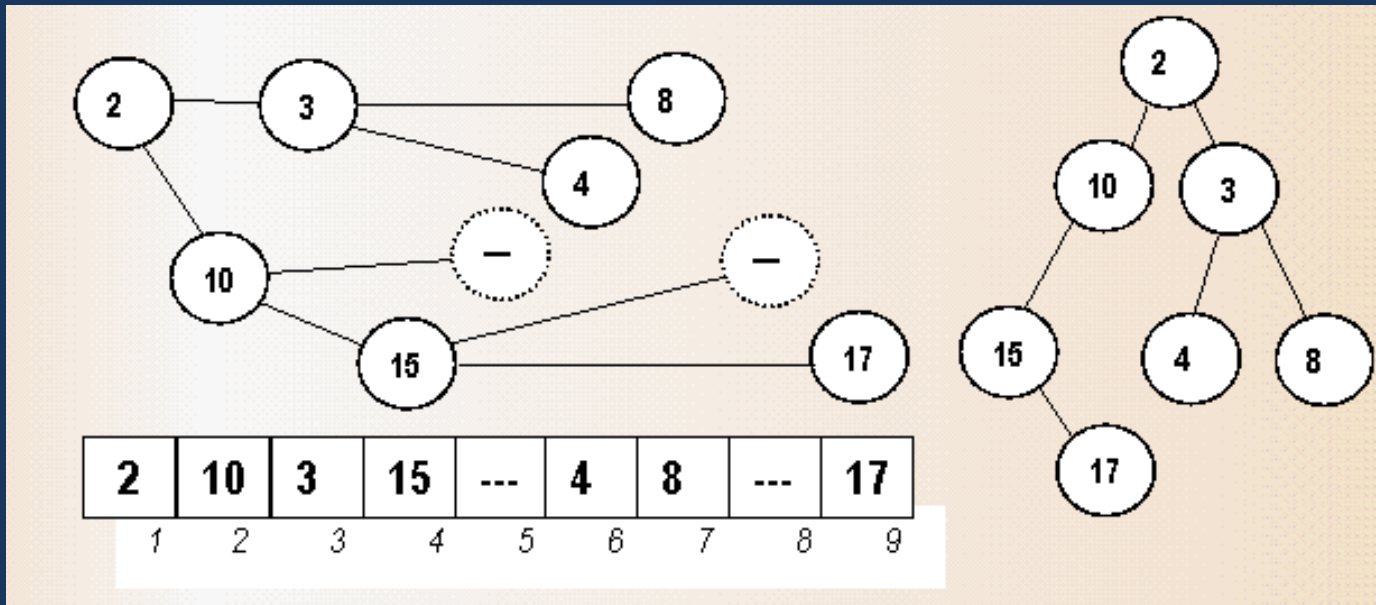
```
struct mtree{  
    char *s;  
    int parent;  
};  
mtree A1[]={  
    {"aa",-1},  
    {"bb",0},  
    {"cc",1},  
    {"dd",0},  
    {"ee",8},  
    {"ff",1},  
    {"gg",3},  
    {"hh",5},  
    {"ii",3},  
    {"jj",7},  
    {"kk",7}};
```

```
void scan_m(mtree A[], int n, int k,int level){ // k - индекс текущей вершины  
    printf("l=%d node=%d s=%s\n",level,k,A[k].s);  
    for (int i=0;i<n;i++)  
        if (A[i].parent==k) scan_m(A,n,i,level+1);  
}
```



Представление деревьев

Пирамида: Дерево в массиве с вычислением адресов потомков
Потомки для $n \rightarrow 2n$ и $2n+1$, корень $n=1$



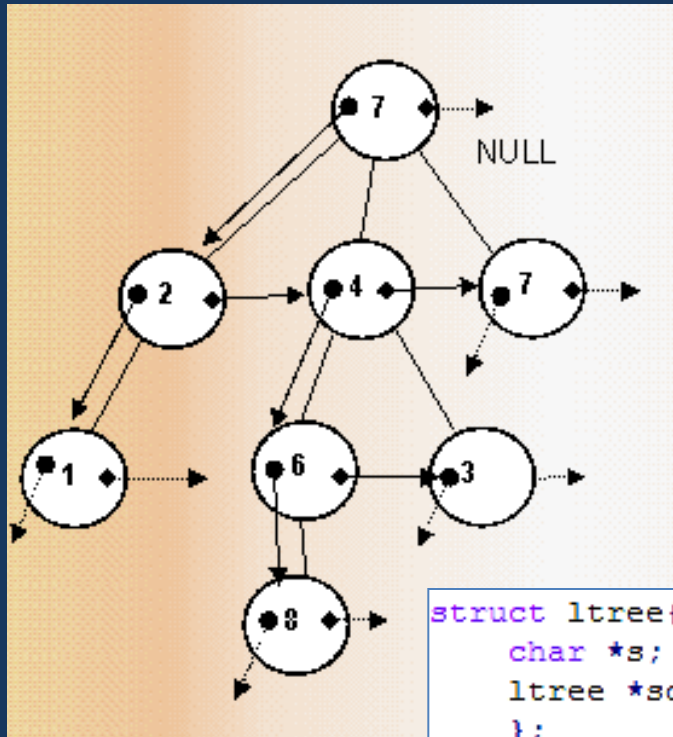
```
void scan_2(int A[], int n, int k, int level){    // k - индекс текущей вершины
    if (k >= n) return;
    if (A[k] == -1) return;
    printf("l=%d node=%d val=%d\n", level, k, A[k]);
    scan_2(A, n, 2*k, level+1);
    scan_2(A, n, 2*k+1, level+1);
}

//          0 1 2 3 4 5 6 7 8 9 10
int A2[]={-1,1,2,3,4,5,6,7,8,9,10};
```



Представление деревьев

Ветвящийся список: «старший сын» и «следующий брат»



```
void scan_l(ltree *p, int level){  
    if (p==NULL) return;  
    printf("l=%d val=%s\n",level,p->s);  
    for (ltree *q=p->son;q!=NULL;q=q->bro)  
        scan_l(q,level+1);  
}
```

```
struct ltree{  
    char *s;  
    ltree *son,*bro;    // Указатели на старшего сына  
                        // и младшего брата  
};  
  
ltree A={"aa",NULL,NULL}, // Последняя в списке  
      B={"bb",NULL,&A},  
      C={"cc",NULL,&B},    // Список потомков - концевых вершин A,B,C  
      D={"dd",NULL,NULL},  
      E={"ee",&C,NULL},  
      F={"ff",&D,&E},    // Список потомков G - вершин F,E  
      G={"gg",&F,NULL},  
      *ph = &G;
```



Представление деревьев

Массив ветвей (индекс предка-индекс потомка)

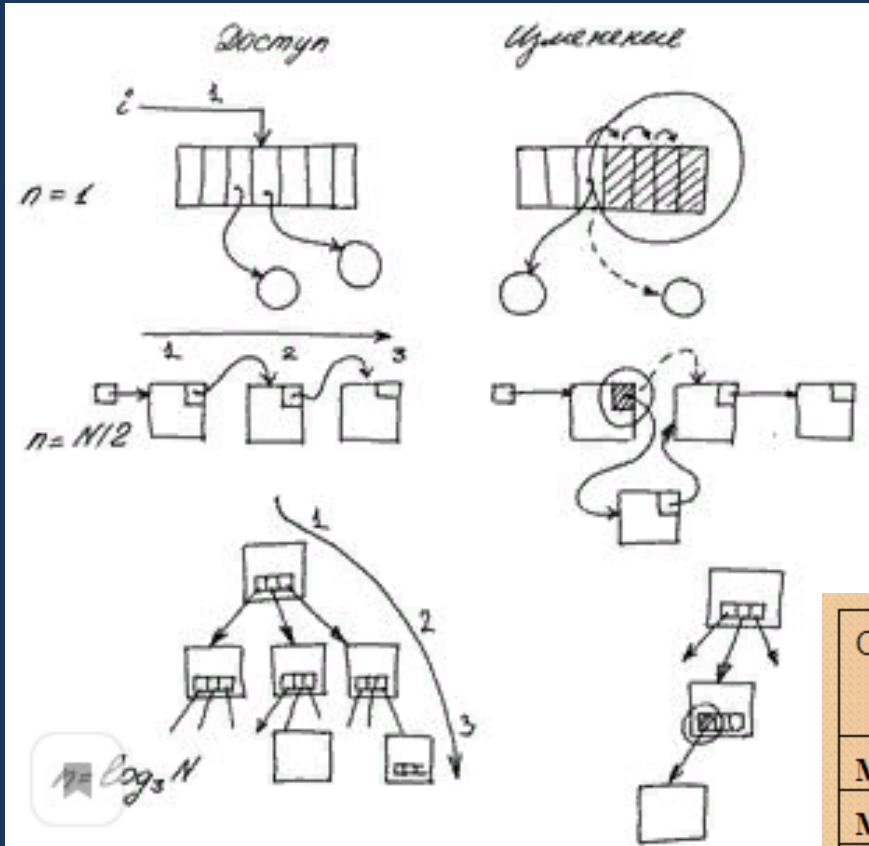
```
// Представление дерева в виде таблицы ветвей
void scan_3(int A[][2],int k,int level){    // k - индекс текущей вершины
    printf("l=%d node=%d\n",level,k);
    for (int i=0;A[i][0]!=-1;i++){
        if (A[i][0]==k) scan_3(A,A[i][1],level+1);
    }
}
int A3[][2]={0,1},{0,2},{1,3},{1,4},{2,5},{1,6},{3,7},{3,8},{4,9},{5,10},{6,11},{7,12},{8,13},{9,14},{10,15},{11,16},{12,17},{13,18},{14,19},{15,20},{16,21},{17,22},{18,23},{19,24},{20,25},{21,26},{22,27},{23,28},{24,29},{25,30},{26,31},{27,32},{28,33},{29,34},{30,35},{31,36},{32,37},{33,38},{34,39},{35,40},{36,41},{37,42},{38,43},{39,44},{40,45},{41,46},{42,47},{43,48},{44,49},{45,50},{46,51},{47,52},{48,53},{49,54},{50,55},{51,56},{52,57},{53,58},{54,59},{55,60},{56,61},{57,62},{58,63},{59,64},{60,65},{61,66},{62,67},{63,68},{64,69},{65,70},{66,71},{67,72},{68,73},{69,74},{70,75},{71,76},{72,77},{73,78},{74,79},{75,80},{76,81},{77,82},{78,83},{79,84},{80,85},{81,86},{82,87},{83,88},{84,89},{85,90},{86,91},{87,92},{88,93},{89,94},{90,95},{91,96},{92,97},{93,98},{94,99},{95,100},{96,101},{97,102},{98,103},{99,104},{100,105},{101,106},{102,107},{103,108},{104,109},{105,110},{106,111},{107,112},{108,113},{109,114},{110,115},{111,116},{112,117},{113,118},{114,119},{115,120},{116,121},{117,122},{118,123},{119,124},{120,125},{121,126},{122,127},{123,128},{124,129},{125,130},{126,131},{127,132},{128,133},{129,134},{130,135},{131,136},{132,137},{133,138},{134,139},{135,140},{136,141},{137,142},{138,143},{139,144},{140,145},{141,146},{142,147},{143,148},{144,149},{145,150},{146,151},{147,152},{148,153},{149,154},{150,155},{151,156},{152,157},{153,158},{154,159},{155,160},{156,161},{157,162},{158,163},{159,164},{160,165},{161,166},{162,167},{163,168},{164,169},{165,170},{166,171},{167,172},{168,173},{169,174},{170,175},{171,176},{172,177},{173,178},{174,179},{175,180},{176,181},{177,182},{178,183},{179,184},{180,185},{181,186},{182,187},{183,188},{184,189},{185,190},{186,191},{187,192},{188,193},{189,194},{190,195},{191,196},{192,197},{193,198},{194,199},{195,200},{196,201},{197,202},{198,203},{199,204},{200,205},{201,206},{202,207},{203,208},{204,209},{205,210},{206,211},{207,212},{208,213},{209,214},{210,215},{211,216},{212,217},{213,218},{214,219},{215,220},{216,221},{217,222},{218,223},{219,224},{220,225},{221,226},{222,227},{223,228},{224,229},{225,230},{226,231},{227,232},{228,233},{229,234},{230,235},{231,236},{232,237},{233,238},{234,239},{235,240},{236,241},{237,242},{238,243},{239,244},{240,245},{241,246},{242,247},{243,248},{244,249},{245,250},{246,251},{247,252},{248,253},{249,254},{250,255},{251,256},{252,257},{253,258},{254,259},{255,260},{256,261},{257,262},{258,263},{259,264},{260,265},{261,266},{262,267},{263,268},{264,269},{265,270},{266,271},{267,272},{268,273},{269,274},{270,275},{271,276},{272,277},{273,278},{274,279},{275,280},{276,281},{277,282},{278,283},{279,284},{280,285},{281,286},{282,287},{283,288},{284,289},{285,290},{286,291},{287,292},{288,293},{289,294},{290,295},{291,296},{292,297},{293,298},{294,299},{295,300},{296,301},{297,302},{298,303},{299,304},{300,305},{301,306},{302,307},{303,308},{304,309},{305,310},{306,311},{307,312},{308,313},{309,314},{310,315},{311,316},{312,317},{313,318},{314,319},{315,320},{316,321},{317,322},{318,323},{319,324},{320,325},{321,326},{322,327},{323,328},{324,329},{325,330},{326,331},{327,332},{328,333},{329,334},{330,335},{331,336},{332,337},{333,338},{334,339},{335,340},{336,341},{337,342},{338,343},{339,344},{340,345},{341,346},{342,347},{343,348},{344,349},{345,350},{346,351},{347,352},{348,353},{349,354},{350,355},{351,356},{352,357},{353,358},{354,359},{355,360},{356,361},{357,362},{358,363},{359,364},{360,365},{361,366},{362,367},{363,368},{364,369},{365,370},{366,371},{367,372},{368,373},{369,374},{370,375},{371,376},{372,377},{373,378},{374,379},{375,380},{376,381},{377,382},{378,383},{379,384},{380,385},{381,386},{382,387},{383,388},{384,389},{385,390},{386,391},{387,392},{388,393},{389,394},{390,395},{391,396},{392,397},{393,398},{394,399},{395,400},{396,401},{397,402},{398,403},{399,404},{400,405},{401,406},{402,407},{403,408},{404,409},{405,410},{406,411},{407,412},{408,413},{409,414},{410,415},{411,416},{412,417},{413,418},{414,419},{415,420},{416,421},{417,422},{418,423},{419,424},{420,425},{421,426},{422,427},{423,428},{424,429},{425,430},{426,431},{427,432},{428,433},{429,434},{430,435},{431,436},{432,437},{433,438},{434,439},{435,440},{436,441},{437,442},{438,443},{439,444},{440,445},{441,446},{442,447},{443,448},{444,449},{445,450},{446,451},{447,452},{448,453},{449,454},{450,455},{451,456},{452,457},{453,458},{454,459},{455,460},{456,461},{457,462},{458,463},{459,464},{460,465},{461,466},{462,467},{463,468},{464,469},{465,470},{466,471},{467,472},{468,473},{469,474},{470,475},{471,476},{472,477},{473,478},{474,479},{475,480},{476,481},{477,482},{478,483},{479,484},{480,485},{481,486},{482,487},{483,488},{484,489},{485,490},{486,491},{487,492},{488,493},{489,494},{490,495},{491,496},{492,497},{493,498},{494,499},{495,500},{496,501},{497,502},{498,503},{499,504},{500,505},{501,506},{502,507},{503,508},{504,509},{505,510},{506,511},{507,512},{508,513},{509,514},{510,515},{511,516},{512,517},{513,518},{514,519},{515,520},{516,521},{517,522},{518,523},{519,524},{520,525},{521,526},{522,527},{523,528},{524,529},{525,530},{526,531},{527,532},{528,533},{529,534},{530,535},{531,536},{532,537},{533,538},{534,539},{535,540},{536,541},{537,542},{538,543},{539,544},{540,545},{541,546},{542,547},{543,548},{544,549},{545,550},{546,551},{547,552},{548,553},{549,554},{550,555},{551,556},{552,557},{553,558},{554,559},{555,560},{556,561},{557,562},{558,563},{559,564},{560,565},{561,566},{562,567},{563,568},{564,569},{565,570},{566,571},{567,572},{568,573},{569,574},{570,575},{571,576},{572,577},{573,578},{574,579},{575,580},{576,581},{577,582},{578,583},{579,584},{580,585},{581,586},{582,587},{583,588},{584,589},{585,590},{586,591},{587,592},{588,593},{589,594},{590,595},{591,596},{592,597},{593,598},{594,599},{595,600},{596,601},{597,602},{598,603},{599,604},{600,605},{601,606},{602,607},{603,608},{604,609},{605,610},{606,611},{607,612},{608,613},{609,614},{610,615},{611,616},{612,617},{613,618},{614,619},{615,620},{616,621},{617,622},{618,623},{619,624},{620,625},{621,626},{622,627},{623,628},{624,629},{625,630},{626,631},{627,632},{628,633},{629,634},{630,635},{631,636},{632,637},{633,638},{634,639},{635,640},{636,641},{637,642},{638,643},{639,644},{640,645},{641,646},{642,647},{643,648},{644,649},{645,650},{646,651},{647,652},{648,653},{649,654},{650,655},{651,656},{652,657},{653,658},{654,659},{655,660},{656,661},{657,662},{658,663},{659,664},{660,665},{661,666},{662,667},{663,668},{664,669},{665,670},{666,671},{667,672},{668,673},{669,674},{670,675},{671,676},{672,677},{673,678},{674,679},{675,680},{676,681},{677,682},{678,683},{679,684},{680,685},{681,686},{682,687},{683,688},{684,689},{685,690},{686,691},{687,692},{688,693},{689,694},{690,695},{691,696},{692,697},{693,698},{694,699},{695,700},{696,701},{697,702},{698,703},{699,704},{700,705},{701,706},{702,707},{703,708},{704,709},{705,710},{706,711},{707,712},{708,713},{709,714},{710,715},{711,716},{712,717},{713,718},{714,719},{715,720},{716,721},{717,722},{718,723},{719,724},{720,725},{721,726},{722,727},{723,728},{724,729},{725,730},{726,731},{727,732},{728,733},{729,734},{730,735},{731,736},{732,737},{733,738},{734,739},{735,740},{736,741},{737,742},{738,743},{739,744},{740,745},{741,746},{742,747},{743,748},{744,749},{745,750},{746,751},{747,752},{748,753},{749,754},{750,755},{751,756},{752,757},{753,758},{754,759},{755,760},{756,761},{757,762},{758,763},{759,764},{760,765},{761,766},{762,767},{763,768},{764,769},{765,770},{766,771},{767,772},{768,773},{769,774},{770,775},{771,776},{772,777},{773,778},{774,779},{775,780},{776,781},{777,782},{778,783},{779,784},{780,785},{781,786},{782,787},{783,788},{784,789},{785,790},{786,791},{787,792},{788,793},{789,794},{790,795},{791,796},{792,797},{793,798},{794,799},{795,800},{796,801},{797,802},{798,803},{799,804},{800,805},{801,806},{802,807},{803,808},{804,809},{805,810},{806,811},{807,812},{808,813},{809,814},{810,815},{811,816},{812,817},{813,818},{814,819},{815,820},{816,821},{817,822},{818,823},{819,824},{820,825},{821,826},{822,827},{823,828},{824,829},{825,830},{826,831},{827,832},{828,833},{829,834},{830,835},{831,836},{832,837},{833,838},{834,839},{835,840},{836,841},{837,842},{838,843},{839,844},{840,845},{841,846},{842,847},{843,848},{844,849},{845,850},{846,851},{847,852},{848,853},{849,854},{850,855},{851,856},{852,857},{853,858},{854,859},{855,860},{856,861},{857,862},{858,863},{859,864},{860,865},{861,866},{862,867},{863,868},{864,869},{865,870},{866,871},{867,872},{868,873},{869,874},{870,875},{871,876},{872,877},{873,878},{874,879},{875,880},{876,881},{877,882},{878,883},{879,884},{880,885},{881,886},{882,887},{883,888},{884,889},{885,890},{886,891},{887,892},{888,893},{889,894},{890,895},{891,896},{892,897},{893,898},{894,899},{895,900},{896,901},{897,902},{898,903},{899,904},{900,905},{901,906},{902,907},{903,908},{904,909},{905,910},{906,911},{907,912},{908,913},{909,914},{910,915},{911,916},{912,917},{913,918},{914,919},{915,920},{916,921},{917,922},{918,923},{919,924},{920,925},{921,926},{922,927},{923,928},{924,929},{925,930},{926,931},{927,932},{928,933},{929,934},{930,935},{931,936},{932,937},{933,938},{934,939},{935,940},{936,941},{937,942},{938,943},{939,944},{940,945},{941,946},{942,947},{943,948},{944,949},{945,950},{946,951},{947,952},{948,953},{949,954},{950,955},{951,956},{952,957},{953,958},{954,959},{955,960},{956,961},{957,962},{958,963},{959,964},{960,965},{961,966},{962,967},{963,968},{964,969},{965,970},{966,971},{967,972},{968,973},{969,974},{970,975},{971,976},{972,977},{973,978},{974,979},{975,980},{976,981},{977,982},{978,983},{979,984},{980,985},{981,986},{982,987},{983,988},{984,989},{985,990},{986,991},{987,992},{988,993},{989,994},{990,995},{991,996},{992,997},{993,998},{994,999},{995,1000},{996,1001},{997,1002},{998,1003},{999,1004},{1000,1005},{1001,1006},{1002,1007},{1003,1008},{1004,1009},{1005,1010},{1006,1011},{1007,1012},{1008,1013},{1009,1014},{1010,1015},{1011,1016},{1012,1017},{1013,1018},{1014,1019},{1015,1020},{1016,1021},{1017,1022},{1018,1023},{1019,1024},{1020,1025},{1021,1026},{1022,1027},{1023,1028},{1024,1029},{1025,1030},{1026,1031},{1027,1032},{1028,1033},{1029,1034},{1030,1035},{1031,1036},{1032,1037},{1033,1038},{1034,1039},{1035,1040},{1036,1041},{1037,1042},{1038,1043},{1039,1044},{1040,1045},{1041,1046},{1042,1047},{1043,1048},{1044,1049},{1045,1050},{1046,1051},{1047,1052},{1048,1053},{1049,1054},{1050,1055},{1051,1056},{1052,1057},{1053,1058},{1054,1059},{1055,1060},{1056,1061},{1057,1062},{1058,1063},{1059,1064},{1060,1065},{1061,1066},{1062,1067},{1063,1068},{1064,1069},{1065,1070},{1066,1071},{1067,1072},{1068,1073},{1069,1074},{1070,1075},{1071,1076},{1072,1077},{1073,1078},{1074,1079},{1075,1080},{1076,1081},{1077,1082},{1078,1083},{1079,1084},{1080,1085},{1081,1086},{1082,1087},{1083,1088},{1084,1089},{1085,1090},{1086,1091},{1087,1092},{1088,1093},{1089,1094},{1090,1095},{1091,1096},{1092,1097},{1093,1098},{1094,1099},{1095,1100},{1096,1101},{1097,1102},{1098,1103},{1099,1104},{1100,1105},{1101,1106},{1102,1107},{1103,1108},{1104,1109},{1105,1110},{1106,1111},{1107,1112},{1108,1113},{1109,1114},{1110,1115},{1111,1116},{1112,1117},{1113,1118},{1114,1119},{1115,1120},{1116,1121},{1117,1122},{1118,1123},{1119,1124},{1120,1125},{1121,1126},{1122,1127},{1123,1128},{1124,1129},{1125,1130},{1126,1131},{1127,1132},{1128,1133},{1129,1134},{1130,1135},{1131,1136},{1132,1137},{1133,1138},{1134,1139},{1135,1140},{1136,1141},{1137,1142},{1138,1143},{1139,1144},{1140,1145},{1141,1146},{1142,1147},{1143,1148},{1144,1149},{1145,1150},{1146,1151},{1147,1152},{1148,1153},{1149,1154},{1150,1155},{1151,1156},{1152,1157},{1153,1158},{1154,1159},{1155,1160},{1156,1161},{1157,1162},{1158,1163},{1159,1164},{1160,1165},{1161,1166},{1162,1167},{1163,1168},{1164,1169},{1165,1170},{1166,1171},{1167,1172},{1168,1173},{1169,1174},{1170,1175},{1171,1176},{1172,1177},{1173,1178},{1174,1179},{1175,1180},{1176,1181},{1177,1182},{1178,1183},{1179,1184},{1180,1185},{1181,1186},{1182,1187},{1183,1188},{1184,1189},{1185,1190},{1186,1191},{1187,1192},{1188,1193},{1189,1194},{1190,1195},{1191,1196},{1192,1197},{1193,1198},{1194,1199},{1195,1200},{1196,1201},{1197,1202},{1198,1203},{1199,1204},{1200,1205},{1201,1206},{1202,1207},{1203,1208},{1204,1209},{1205,1210},{1206,1211},{1207,1212},{1208,1213},{1209,1214},{1210,1215},{1211,1216},{1212,1217},{1213,1218},{1214,1219},{1215,1220},{1216,1221},{1217,1222},{1218,1223},{1219,1224},{1220,1225},{1221,1226},{1222,1227},{1223,1228},{1224,1229},{1225,1230},{1226,1231},{1227,1232},{1228,1233},{1229,1234},{1230,1235},{1231,1236},{1232,1237},{1233,1238},{1234,1239},{1235,1240},{1236,1241},{1237,1242},{1238,1243},{1239,1244},{1240,1245},{1241,1246},{1242,1247},{1243,1248},{1244,1249},{1245,1250},{1246,1251},{1247,1252},{1248,1253},{1249,1254},{1250,1255},{1251,1256},{1252,1257},{1253,1258},{1254,1259},{1255,1260},{1256,1261},{1257,1262},{1258,1263},{1259,1264},{1260,1265},{1261,1266},{1262,1267},{1263,1268},{1264,1269},{1265,1270},{1266,1271},{1267,1272},{1268,1273},{1269,1274},{1270,1275},{1271,1276},{1272,1277},{1273,1278},{1274,1279},{1275,1280},{1276,1281},{1277,1282},{1278,1283},{1279,1284},{1280,1285},{1281,1286},{1282,1287},{1283,1288},{1284,1289},{1285,1290},{1286,1291},{1287,1292},{1288,1293},{1289,1294},{1290,1295},{1291,1296},{1292,1297},{1293,1298},{1294,1299},{1295,1300},{1296,1301},{1297,1302},{1298,1303},{1299,1304},{1300,1305},{1301,1306},{1302,1307},{1303,1308},{1304,1309},{1305,1310},{1306,1311},{1307,1312},{1308,1313},{1309,1314},{1310,1315},{1311,1316},{1312,1317},{1313,1318},{1314,1319},{1315,1320},{1316,1321},{1317,1322},{1318,1323},{1319,1324},{1320,1325},{1321,1326},{1322,1327},{1323,1328},{1324,1329},{1325,1330},{1326,1331},{1327,1332},{1328,1333},{1329,1334},{1330,1335},{1331,1336},{1332,1337},{1333,1338},{1334,1339},{1335,1340},{1336,1341},{1337,1342},{1338,1343},{1339,1344},{1340,1345},{1341,1346},{1342,1347},{1343,1348},{1344,1349},{1345,1350},{1346,1351},{1347,1352},{1348,1353},{1349,1354},{1350,1355},{1351,1356},{1352,1357},{1353,1358},{1354,1359},{1355,1360},{1356,1361},{1357,1362},{1358,1363},{1359,1364},{1360,1365},{1361,1366},{1362,1367},{1363,1368},{1364,1369},{1365,1370},{1366,1371},{1367,1372},{1368,1373},{1369,1374},{1370,1375},{1371,1376},{1372,1377},{1373,1378},{1374,1379},{1375,1380},{1376,1381},{1377,1382},{1378,1383},{1379,1384},{1380,1385},{1381,1386},{1382,1387},{1383,1388},{1384,1389},{1385,1390},{1386,1391},{1387,1392},{1388,1393},{1389,1394},{1390,1395},{1391,1396},{1392,1397},{1393,1398},{1394,1399},{1395,1400},{1396,1401},{1397,1402},{1398,1403},{1399,1404},{1400,1405},{1401,1406},{1402,1407},{1403,1408},{1404,1409},{1405,1410},{1406,1411},{1407,1412},{1408,1413},{1409,1414},{1410,1415},{1411,1416},{1412,1417},{1413,1418},{1414,1419},{1415,1420},{1416,1421},{1417,1422},{1418,1423},{1419,1424},{1420,1425},{1421,1426},{1422,1427},{1423,1428},{1424,1429},{1425,1430},{1426,1431},{1427,1432},{1428,1433},{1429,1434},{1430,1435},{1431,1
```



Трудоёмкость алгоритмов на деревьях

Связь между «глубиной» дерева и количеством вершин:

$$N = 1 + m + m^2 + m^3 + \dots + m^L < m^{L+1}, \quad L < \log_m N$$



Оценка T	Извлечение по ЛН	Вставка, удаление	Поиск по значению (упоряд.)
Массив	1	N	log(N)
Массив указателей	1(2)	N	log(N)
Список	N	2(3)	N
Дерево сбаланс.	log(N)	1...log(N)	log(N)
Дерево несбаланс.	log(N)...N	1...log(N)	log(N)...N



Рекурсивный обход дерева

Линейный поиск в дереве, параметры дерева в целом:

- Индуктивный подход: текущая вершина возвращает ???, которое формируется от полученного от потомков ??? по и известному алгоритму (сумма ЧЕГО, максимум ЧЕГО)

```
// Алгоритмы, основанные на полном обходе дерева
struct tree1{
    int val;
    int n;
    tree1 *ch[10];
}
//----- Количество вершин в дереве
int F1(tree1 *p){
    int s=1;
    for (int i=0; i < p->n; i++) s+=F1(p->ch[i]);
    return s;
}
//----- Сумма значений в вершине дерева
int F2(tree1 *p){
    int s=p->val;
    for (int i=0; i < p->n; i++) s+=F2(p->ch[i]);
    return s;
}
```

```
//----- Максимальное значение в вершине дерева
int F3(tree1 *p){
    int s=p->val;
    for (int i=0; i < p->n; i++)
        { int vv=F3(p->ch[i]); if (vv > s) s=vv; }
    return s;
}
//----- Максимальная длина ветви дерева
int F4(tree1 *p){
    int s=0;
    for (int i=0; i < p->n; i++)
        { int vv=F4(p->ch[i]); if (vv > s) s=vv; }
    return s+1;
}
```

```
//----- Суммарное расстояние до корня - степень сбалансированности
int F6(tree1 *p, int L){
    int s=L;
    for (int i=0; i < p->n; i++)
        s+=F6(p->ch[i], L+1);
    return s;
}
```

```
//----- Поиск первого значения, удовлетворяющего условию
tree1 *F7(tree1 *p, int vv){
    if (p->val == vv) return p;
    for (int i=0; i < p->n; i++){
        tree1 *q=F7(p->ch[i]);
        if (q!=NULL) return q;
    }
    return NULL;
}
// Действие, выполняемое потомком
// Найдено в текущей вершине - вернуть
// Найдено у потомка – прекратить обход
// Обработка результата предком
```



Структуры данных на деревьях

Логический (порядковый номер) – зависит от порядка обхода (рекурсивного):

- «эгоистический» - предок – потомки
- «альтруистический» - потомки – предок
- «слева направо» - двоичное дерево, левое поддереву – предок – правое поддереву

Размещение данных:

- Линейная неупорядоченная структура данных на основе нумерации вершин
- Упорядоченные данные - двоичное дерево (слева направо)
- Вертикально упорядоченное дерево – минимум в вершине дерева

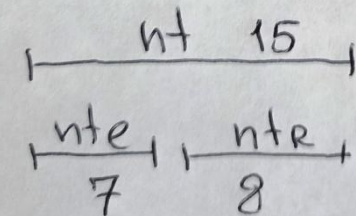
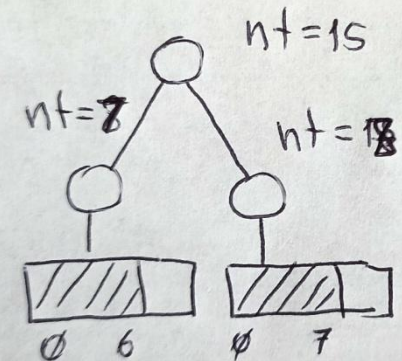
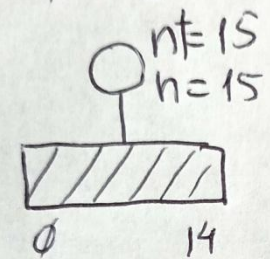
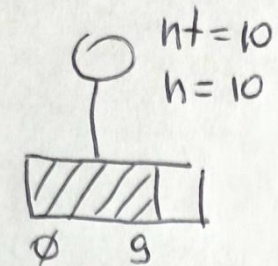


Дерево с данными в конечных вершинах

```
struct node{
    int size;           // Размерность ДМУ
    int n;              // Количество значений в ДМУ
    int nt;             // Общее количество в поддереве
    node *left,*right;
    int *data;          // ДМ данных
};

node *create(int sz0){ // Конечная вершина без данных
    node *pp = new node();
    pp->left=pp->right=NULL;
    pp->size=sz0;
    pp->data = new int[sz0];
    pp->n = pp->nt = 0;
}
```

```
void extend(node *pp){ // Проверить конечную и разделить на 2
    if (pp->n!=pp->size)
        return;
    pp->left = create(pp->size);
    pp->right = create(pp->size);
    int dd = pp->size/2;
    for(int i=0;i<dd;i++) // Половина в левое поддерево
        pp->left->data[i]=pp->data[i];
    pp->left->n = pp->left->nt = dd;
    for(int i=dd;i<pp->size;i++) // Половина в правое поддерево
        pp->right->data[i-dd]=pp->data[i];
    pp->right->n = pp->right->nt = pp->size - dd;
}
```





Дерево с данными в конечных вершинах

```
void scan(node *pp, int level, int idx){
    if (pp==NULL) return;
    if (pp->left==NULL){                // Конечная вершина
        printf("\nlevel=%d A[%d]=", level, idx);
        for(int i=0; i<pp->n; i++)
            printf("%d ", pp->data[i]);
        printf("\n");
    }
    else{
        scan(pp->left, level+1, idx);
        scan(pp->right, level+1, idx+pp->left->n);
    }
}

void add(node *pp, int v){
    if (pp->left==NULL){
        pp->data[pp->n++] = v;
        pp->nt++;
        extend(pp);
    }
    else{
        add(pp->right, v);
    }
}

int main(){
    node *root = create(10);
    for(int i=0; i<100; i++)
        add(root, i);
    scan(root, 1, 0);
}
```

Нужно в обеих ветках

```
D:\Temp\gugu\bin\Debug\gugu.exe

level=2 A[0]=0 1 2 3 4
level=3 A[5]=5 6 7 8 9
level=4 A[10]=10 11 12 13 14
level=5 A[15]=15 16 17 18 19
level=6 A[20]=20 21 22 23 24
level=7 A[25]=25 26 27 28 29
level=8 A[30]=30 31 32 33 34
level=9 A[35]=35 36 37 38 39
level=10 A[40]=40 41 42 43 44
level=11 A[45]=45 46 47 48 49
level=12 A[50]=50 51 52 53 54
level=13 A[55]=55 56 57 58 59
level=14 A[60]=60 61 62 63 64
level=15 A[65]=65 66 67 68 69
level=16 A[70]=70 71 72 73 74
level=17 A[75]=75 76 77 78 79
level=18 A[80]=80 81 82 83 84
level=19 A[85]=85 86 87 88 89
level=20 A[90]=90 91 92 93 94
level=20 A[95]=95 96 97 98 99
Process returned 0 (0x0)   executi
Press any key to continue.
```



Дерево с данными в конечных вершинах

Далее – ООП нотация:

- Функции – методы класса
- Рекурсия – вызов метода через указатель на потомка
- Текущая вершина – текущий объект (контекст класса)

```
struct node{
    int size;           // Размерность ДМУ
    int n;              // Количество значений в ДМУ
    int nt;             // Общее количество в поддереве
    node *left,*right;
    int *data;          // ДМ данных
    node(int sz0){      // Конечная вершина без данных
        left=right=NULL;
        size=sz0;
        data = new int[sz0];
        n = nt = 0;
    }
};
```

```
void scan(int level, int idx){
    if (left==NULL){    // Конечная вершина
        printf("\nlevel=%d A[%d]=" ,level,idx);
        for(int i=0;i<n;i++)
            printf("%d ",data[i]);
    }
    else{
        left->scan(level+1,idx);
        right->scan(level+1,idx+left->n);
    }
}
```

```
void add(int v){
    nt++;
    if (left==NULL){
        data[n++] = v;
        extend();
    }
    else{
        right->add(v);
    }
};

int main(){
    node *root = new node(10);
    for(int i=0;i<100;i++)
        root->add(i);
    root->scan(1,0);
}
```



Дерево с данными в конечных вершинах

Балансировка — выравнивание ветвей дерева

```
// Обход с линейным сливанием данных в массив
void scanForBalance(int v[],int &idx){
    if (left==NULL){
        for(int i=0;i<n;i++){
            v[idx++] = data[i];
        }
    }
    else{
        left->scanForBalance(v,idx);
        right->scanForBalance(v,idx);
    }
}

// Заполнение вершин делением интервала пополам
void loadFrom(int v[],int a, int b, int sz){
    nt = b-a+1;
    if (nt>=sz){
        int m = (a+b+1)/2;
        left = new node(sz);
        left->loadFrom(v,a,m-1,sz);
        right = new node(sz);
        right->loadFrom(v,m,b,sz);
    }
    else{
        n = nt;
        for(int i=0;i<nt;i++){
            data[i]=v[a+i];
        }
    }
}

node *balance(){
    int *vv = new int[nt];
    int idx=0;
    scanForBalance(vv,idx);
    node *out = new node(size);
    out->loadFrom(vv,0,nt-1,size);
    return out;
}
```

```
level=1 total=100 A[0]=
level=2 total=50 A[0]=
level=3 total=25 A[0]=
level=4 total=12 A[0]=
level=5 total=6 A[0]=0 1 2 3 4 5
level=5 total=6 A[6]=6 7 8 9 10 11
level=4 total=13 A[12]=
level=5 total=6 A[12]=12 13 14 15 16 17
level=5 total=7 A[18]=18 19 20 21 22 23 24
level=3 total=25 A[25]=
level=4 total=12 A[25]=
level=5 total=6 A[25]=25 26 27 28 29 30
level=5 total=6 A[31]=31 32 33 34 35 36
level=4 total=13 A[37]=
level=5 total=6 A[37]=37 38 39 40 41 42
level=5 total=7 A[43]=43 44 45 46 47 48 49
level=2 total=50 A[50]=
level=3 total=25 A[50]=
level=4 total=12 A[50]=
level=5 total=6 A[50]=50 51 52 53 54 55
level=5 total=6 A[56]=56 57 58 59 60 61
level=4 total=13 A[62]=
level=5 total=6 A[62]=62 63 64 65 66 67
level=5 total=7 A[68]=68 69 70 71 72 73 74
level=3 total=25 A[75]=
level=4 total=12 A[75]=
level=5 total=6 A[75]=75 76 77 78 79 80
level=5 total=6 A[81]=81 82 83 84 85 86
level=4 total=13 A[87]=
level=5 total=6 A[87]=87 88 89 90 91 92
level=5 total=7 A[93]=93 94 95 96 97 98 99
```



Дерево с данными в конечных вершинах

```
void insert(int idx,int v){
    if (idx>=nt){
        add(v);
        return;
    }
    if (left==NULL){
        for(int k=n-1;k>=idx;k--){
            data[k+1]=data[k];
        }
        data[idx]=v;
        n++;
        nt++;
        extend();
        return;
    }
    nt++;
    if (idx<left->nt)
        left->insert(idx,v);
    else
        right->insert(idx-left->nt,v);
};
```

```
int main(){
    node *root = new node(10);
    for(int i=0;i<100;i++){
        root->add(i);
    }
    root->scan(1,0);
    node *two = root->balance();
    for(int k=0;k<20;k++){
        two->insert(27,k+200);
    }
    two->scan(1,0);
}
```

```
level=1 total=120 A[0]=
level=2 total=70 A[0]=
level=3 total=25 A[0]=
level=4 total=12 A[0]=
level=5 total=6 A[0]=0 1 2 3 4 5
level=5 total=6 A[6]=6 7 8 9 10 11
level=4 total=13 A[12]=
level=5 total=6 A[12]=12 13 14 15 16 17
level=5 total=7 A[18]=18 19 20 21 22 23 24
level=3 total=45 A[25]=
level=4 total=32 A[25]=
level=5 total=26 A[25]=
level=6 total=21 A[25]=
level=7 total=16 A[25]=
level=8 total=11 A[25]=
level=9 total=6 A[25]=25 26 219 218 217 216
level=9 total=5 A[31]=215 214 213 212 211
level=8 total=5 A[36]=210 209 208 207 206
level=7 total=5 A[41]=205 204 203 202 201
level=6 total=5 A[46]=200 27 28 29 30
level=5 total=6 A[51]=31 32 33 34 35 36
level=4 total=13 A[57]=
level=5 total=6 A[57]=37 38 39 40 41 42
level=5 total=7 A[63]=43 44 45 46 47 48 49
level=2 total=50 A[70]=
level=3 total=25 A[70]=
level=4 total=12 A[70]=
level=5 total=6 A[70]=50 51 52 53 54 55
level=5 total=6 A[76]=56 57 58 59 60 61
level=4 total=13 A[82]=
level=5 total=6 A[82]=62 63 64 65 66 67
level=5 total=7 A[88]=68 69 70 71 72 73 74
level=3 total=25 A[95]=
level=4 total=12 A[95]=
level=5 total=6 A[95]=75 76 77 78 79 80
level=5 total=6 A[101]=81 82 83 84 85 86
level=4 total=13 A[107]=
level=5 total=6 A[107]=87 88 89 90 91 92
level=5 total=7 A[113]=93 94 95 96 97 98 99
```

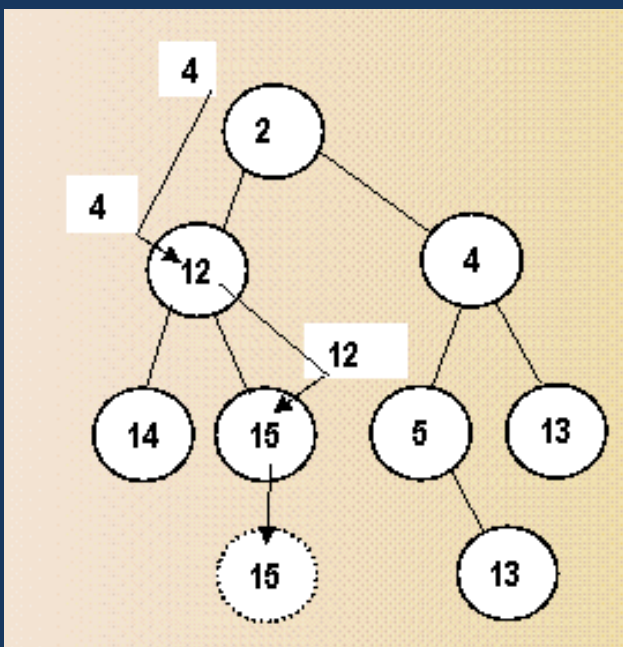
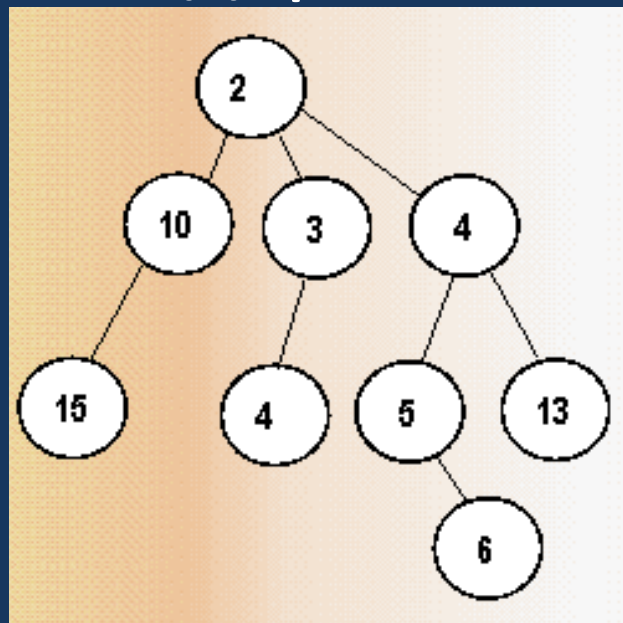


Вертикально упорядоченное дерево

Основная идея: в вершине дерева всегда минимальное значение

Построение дерева:

- алгоритм основывается на ветвлении: выполнение его для корневой вершины сопровождается выбором потомка, для которого алгоритм повторяется линейно - рекурсивно или циклически
- «замещение»: если новое значение меньше, чем значение в текущей вершине, то оно вытесняет последнее, занимая его место. Вытесненное значение участвует в последующем погружении вместо нового. Такой процесс не нарушает свойства упорядоченности дерева в глубину
- для продолжения алгоритма может быть выбран любой потомок. Если выбирать потомка с минимальным количеством вершин в поддереве, то в процессе его построения будет поддерживаться его сбалансированность





Вертикально упорядоченное дерево

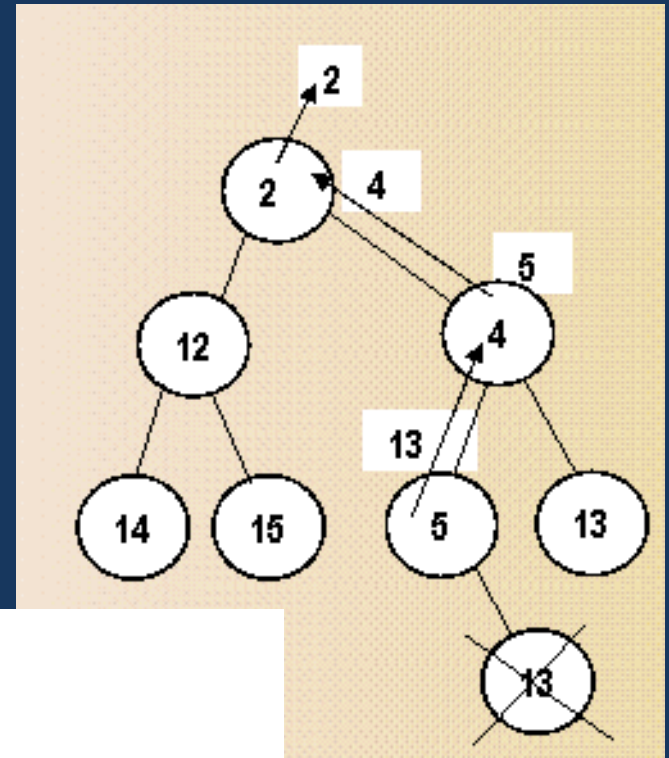
Используется ссылка на указатель: ссылка может быть NULL, тогда на нее подвешивается новая вершина

```
// Построение дерева - вытеснение в в поддерево
// с минимальным количеством вершин
void insert(vtree *&p, int v){
    if (p==NULL){                                // Найдено свободное место
        p=new vtree;                             // Создать новую вершину
        p->l = p->r = NULL;
        p->cnt = 1;
        p->val = v;
        return;
    }
    p->cnt++;                                     // Увеличить счетчик вершин
    if (v < p->val){                               // Вытеснение меньшим большего
        int c=p->val; p->val=v; v=c;              // из текущей вершины
    }
    if (p->l == NULL || p->r !=NULL && p->l->cnt < p->r->cnt)
        insert(p->l,v);                          // Выбор свободной ветви или
    else                                          // поддерева с минимумом вершин
        insert(p->r,v);
}
```



Вертикально упорядоченное дерево

Получение данных: «снятие» данных, с замещением минимального значения от прямых потомков

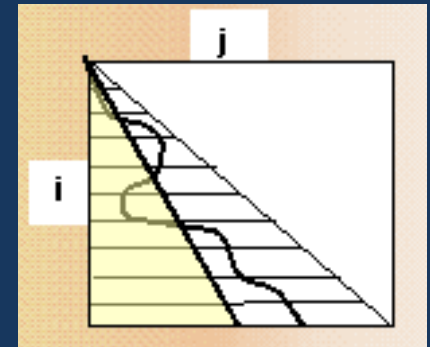
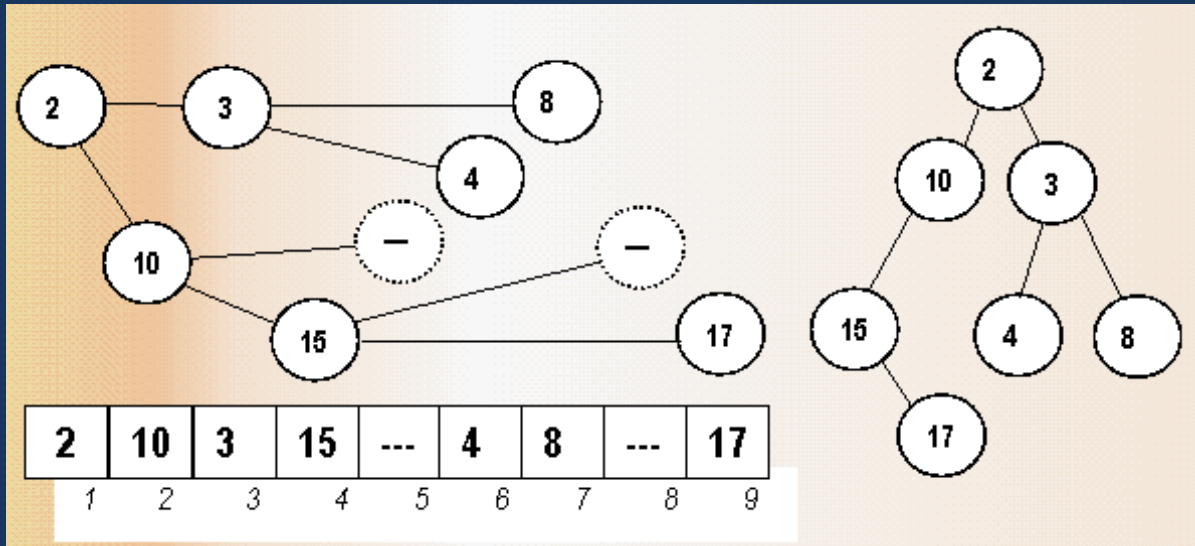


```
// "Сливание" дерева - замещение значения в текущей вершине
// значением минимального потомка
int shift(vtree *&p){
    if (p==NULL) return 0;
    int vv=p->val;
    if (p->l==NULL && p->r==NULL){
        delete p;
        p = NULL;
        return vv;
    }
    if (p->l==NULL || p->r!=NULL && p->r->val < p->l->val)
        p->val = shift(p->r);
    else
        p->val = shift(p->l);
    return vv;
}
```



Пирамидальная сортировка

Вертикально упорядоченное дерево с 2 потомками в массиве с вычисляемыми индексами потомков (пирамида)



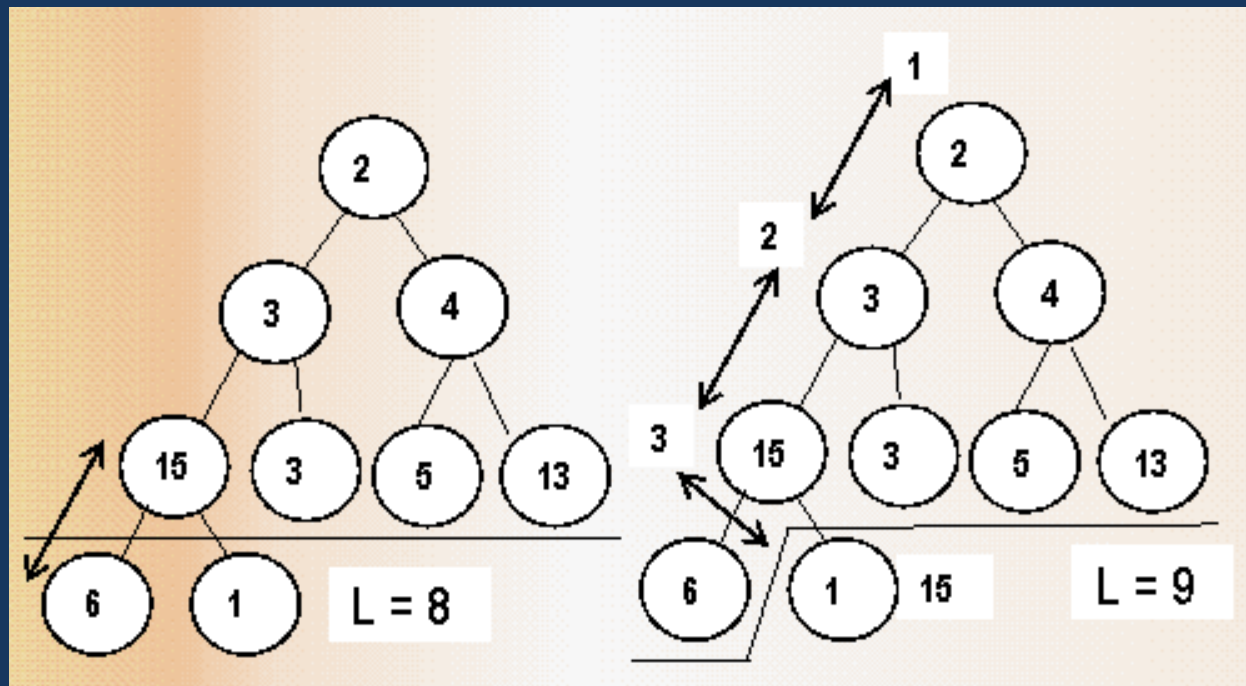
Сравнение с «погружением» - шаг = 1, трудоемкость $N^2/4$ $T=O(N^2)$
Пирамида = переменный шаг погружения 1, 2, 4, 8... = $\log_2 N$ $T=O(N \log_2 N)$

```
//----- Вставка погружением, подозрительно похожая на обмен
void sort(int in[],int n) {
  for ( int i=1; i<n; i++)    // Пока не достигли " дна" или меньшего себя
    for ( int k=i; k !=0 && in[k] < in[k-1]; k--){
      int c=in[k]; in[k]=in[k-1]; in[k-1]=c;
    }
}
```



Пирамидальная сортировка

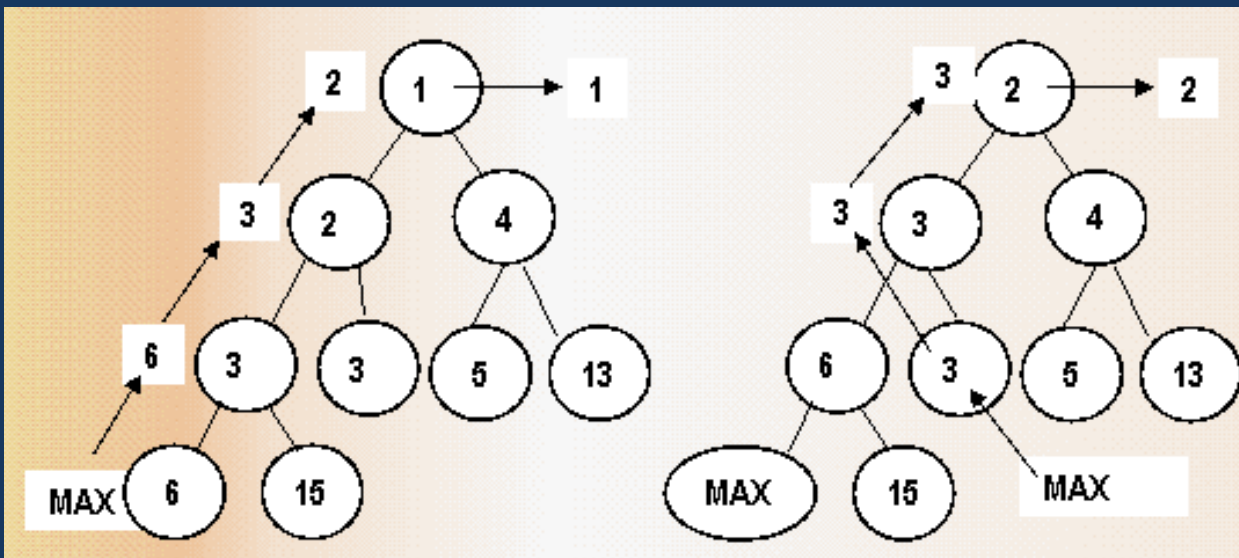
```
// Погружение в дерево путем обмена с предком
void set_tree(int A[],int n){
int l,i;
for (l=2;l<=n;l++){           // l - индекс погружаемой вершины
    for (i=l; i!=1;i=i/2){     // индекс предка = i/2
        if (A[i]>A[i/2]) break; // предок меньше потомка - выйти
        int c=A[i];           // поменяться с предком
        A[i]=A[i/2];
        A[i/2]=c;
    }}
}
```





Пирамидальная сортировка

```
// -ортировка - n раз выбрать из дерева с замещением на минимального потомка
// »звлекаемые элементы в конце массива, не имеющие потомков,
// заменяются на MAXINT. «значение MAXINT обозначает отсутствие вершины
void sort(int A[],int B[], int n){
for (int i=1; i<=n; i++){          // повторить n раз
    int j,k;
    B[i]=A[1];
    for (j=1;j<=n;j=k){           // j=k - переход к замещающему потомку
        k=2*j;                    // k - левый потомок
        if (k>n) { A[j]=MAXINT; break; } // нет потомков - MAXINT (затычка)
        if (k+1>n) { A[j]=A[k]; A[k]=MAXINT; break; } // нет второго потомка - взять первого
        if (A[k+1]<A[k]) k++;       // иначе - выбрать минимального из них
        A[j]=A[k];                // скопировать значение потомка в предка
    }
}
```





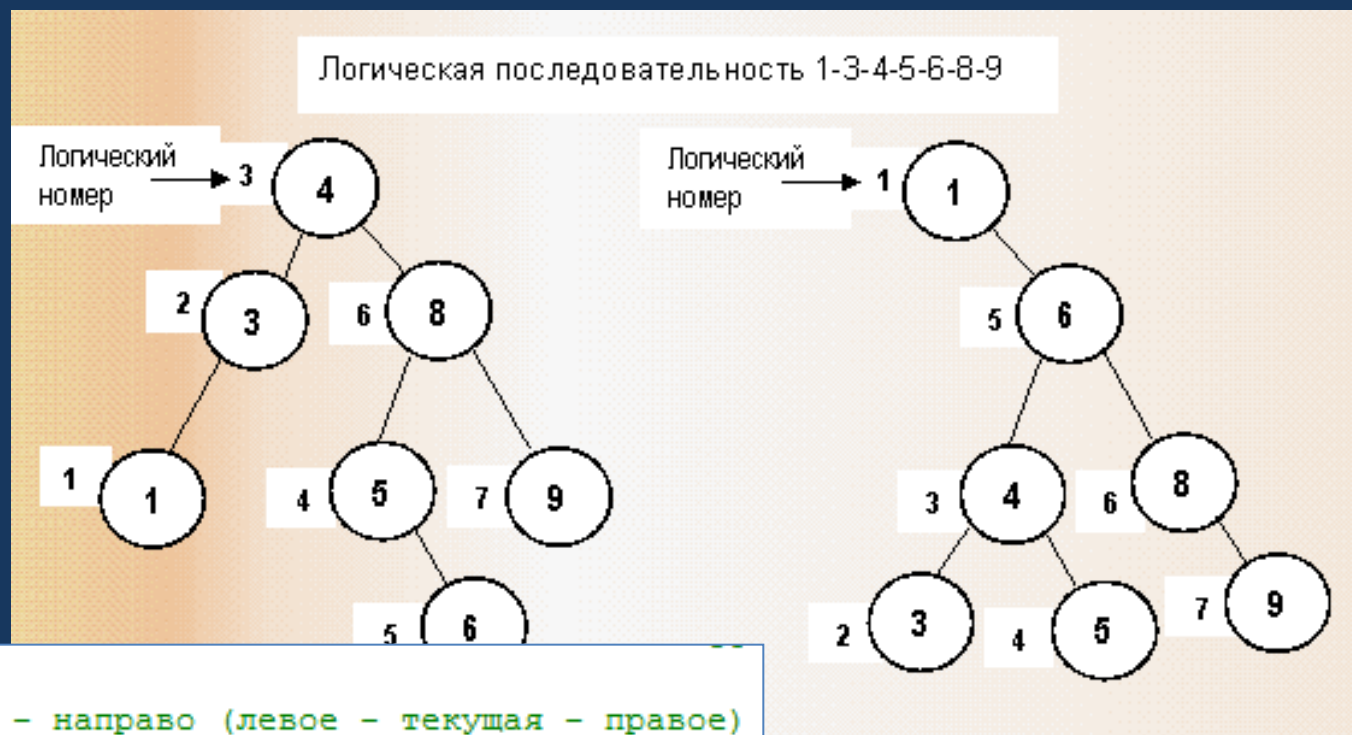
Двоичное дерево

BST – Binary Search Tree

- Упорядочено «слева направо»: в левом – меньшие корневой вершины, в правом – большие
- Обход «левое-текущее-правое» - упорядочение по возрастанию
- Поиск, вставка – линейно-рекурсивные алгоритмы (цикл)
- Трудоемкость поиска/включения $\log_2 N$ (сбалансированное), N - вырожденное в список
- Топология дерева зависит от порядка следования значений при включении
- Включение упорядоченной последовательности – вырождение в список
- Балансировка (выравнивание ветвей)
 - Топологические изменения при вставке (Адельсон-Вельский)
 - «Сливание» данных в упорядоченную структуру и построение нового дерева путем деления интервала на 2 равные части с корневой вершиной посередине (аналог «сбора мусора»)



Двоичное дерево



```
// Двоичное дерево
// Обход дерева слева - направо (левое - текущая - правое)
struct btree{
    int cnt;           // Количество вершин в дереве
    char *s;
    btree *l,*r;       // Левый и правый потомки
};
```

```
// Обход дерева
void scan(btree *p, int level, int &ln){
    if (p==NULL) return;
    scan(p->l, level+1,ln);
    printf("l=%d n=%d cnt=%d :%s\n", level, ln, p->cnt, p->s);
    ln++;
    scan(p->r, level+1,ln);
}
```



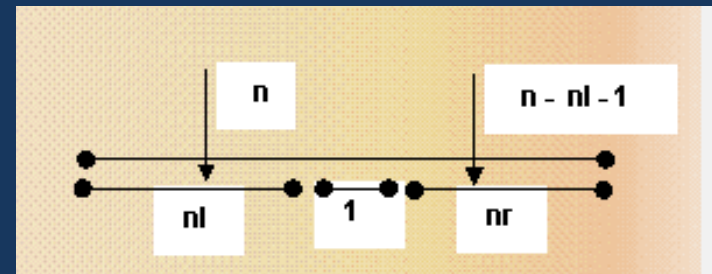
Двоичное дерево

// Поиск вершины с заданным значением - рекурсия

```
btree *search(btree *p, char *key){
    if (p==NULL) return NULL;
    int k=strcmp(key,p->s);
    if (k==0) return p;
    if (k<0) return search(p->l,key);
    return search(p->r,key);
}
```

// Поиск вершины с заданным значением - цикл

```
btree *search1(btree *p, char *key){
    while(p!=NULL){
        int k=strcmp(key,p->s);
        if (k==0) return p;
        if (k<0) p=p->l;
        else p=p->r;
    }
    return NULL; }
}
```



// Создание терминальной вершины

```
btree *create(char *ss){
    btree *q=new btree;
    q->l = q->r = NULL;
    q->cnt = 1;
    q->s = ss;
    return q; }
```

// Поиск вершины по логическому номеру

```
btree *get_n(btree *p, int n){
    if (p==NULL) return NULL;
    if (n >= p->cnt) return NULL;
    if (p->l!=NULL){
        int ll=p->l->cnt;
        if (n<ll) return get_n(p->l,n);
        n-=ll;
    }
    if (n-- ==0) return p;
    return get_n(p->r,n);
}
```

// Включение с сохранением порядка

```
void insert(btree *&p, char *ss){
    if (p == NULL) { p=create(ss); return; }
    p->cnt++;
    if (strcmp(ss,p->s)<0)
        insert(p->l,ss);
    else
        insert(p->r,ss);
}
```



Двоичное дерево. Балансировка

```
// Обход с сохранением вершин в порядке возрастания
```

```
void scan(btrees *qq[], btrees *p, int &ln){
```

```
    if (p==NULL) return;
```

```
    scan(qq, p->l, ln);
```

```
    qq[ln++]=p;
```

```
    scan(qq, p->r, ln);
```

```
}
```

```
void balance(btrees *&ph){
```

```
    int sz=ph->cnt;
```

```
    btrees **pp=new btrees*[sz];
```

```
    int idx=0;
```

```
    scan(pp, ph, idx);
```

```
    ph=balance(pp, 0, sz-1);
```

```
}
```

```
// Обход дерева
```

```
void scan(btrees *p, int level, int &ln){
```

```
    if (p==NULL) return;
```

```
    scan(p->l, level+1, ln);
```

```
    printf("l=%d n=%d cnt=%d :%s\n", level, ln, p->cnt, p->s);
```

```
    ln++;
```

```
    scan(p->r, level+1, ln);
```

```
}
```

```
double sum (btrees *p, int level){
```

```
    if (p==NULL) return 0;
```

```
    return level+sum(p->l, level+1)+sum(p->r, level+1);
```

```
}
```

```
btrees *balance(btrees *pp[], int a, int b){
    if (a>b) return NULL;
    btrees *q; // Интервал =1 - конечная вершина
    if (a==b) { q=pp[a]; q->l=q->r=NULL; q->cnt=1; return q; }
    int m=(a+b)/2;
    q=pp[m]; // Вершина в середине интервала
    q->l=balance(pp, a, m-1); // Поддерево на левом интервале
    q->r=balance(pp, m+1, b); // Поддерево на правом интервале
    q->cnt=b-a+1;
    return q;
}
```



Двоичное дерево. Удаление вершины

```
// Удаление по логическому номеру
char *remove(btrees *p, int n){ // Возвращает удаляемую строку
    if (p == NULL) return NULL;
    if (n >= p->cnt) return NULL;
    p->cnt--;
    if (p->l!=NULL){
        int ll=p->l->cnt;
        if (n<ll) // В интервале левого - рекурсия
            return remove(p->l,n);
        n-=ll;
    }
    if (n-- ==0) { // Удалить корневую
        char *ss = p->s;
        btrees *q; // p - ссылка на указатель
                   // (адрес указателя на текущую вершину)
        if (p->l==NULL && p->r==NULL) // Нет потомков
            { delete p; p=NULL; return ss;}
        if (p->l==NULL) // Только правый -
            { q=p->r; delete p; p=q; // указатель на него
              return ss; } // на место указателя на текущий
        if (p->r==NULL) // Только левый -
            { q=p->l; delete p; p=q; return ss; }
        p->s = remove(p->r,0); // на место текущей строки
        return ss; // [0] из правого поддерева
    }
    else // В интервале правого - рекурсия
        return remove(p->r,n);
    }
}
```



Двоичное дерево

```
int main() {
    btree *ph=NULL;
    int k=0;
    char c[80];
    c[1]=0;
    for(int i=0;i<=26;i++){
        c[0]='A'+i;
        insert(ph, strdup(c));
    }
    puts("-----");
    k=0; scan(ph, 1, k);
    printf("mid=%lf\n", sum(ph, 1)/ph->cnt);
    getchar();
    balance(ph);
    puts("-----");
    k=0; scan(ph, 1, k);
    printf("mid=%lf\n", sum(ph, 1)/ph->cnt);
    getchar();
    FILE *fd=fopen("revolution.txt", "r");
    //-----
```

D:\Temp\fufu\bin\Debug\fufu

```
l=5 n=4 cnt=23 :E
l=6 n=5 cnt=22 :F
l=7 n=6 cnt=21 :G
l=8 n=7 cnt=20 :H
l=9 n=8 cnt=19 :I
l=10 n=9 cnt=18 :J
l=11 n=10 cnt=17 :K
l=12 n=11 cnt=16 :L
l=13 n=12 cnt=15 :M
l=14 n=13 cnt=14 :N
l=15 n=14 cnt=13 :O
l=16 n=15 cnt=12 :P
l=17 n=16 cnt=11 :Q
l=18 n=17 cnt=10 :R
l=19 n=18 cnt=9 :S
l=20 n=19 cnt=8 :T
l=21 n=20 cnt=7 :U
l=22 n=21 cnt=6 :V
l=23 n=22 cnt=5 :W
l=24 n=23 cnt=4 :X
l=25 n=24 cnt=3 :Y
l=26 n=25 cnt=2 :Z
l=27 n=26 cnt=1 :[
mid=14.000000
```

$$L = N/2$$

$$L = \log_2(N)-1$$

```
-----
l=4 n=0 cnt=2 :A
l=5 n=1 cnt=1 :B
l=3 n=2 cnt=6 :C
l=5 n=3 cnt=1 :D
l=4 n=4 cnt=3 :E
l=5 n=5 cnt=1 :F
l=2 n=6 cnt=13 :G
l=4 n=7 cnt=2 :H
l=5 n=8 cnt=1 :I
l=3 n=9 cnt=6 :J
l=5 n=10 cnt=1 :K
l=4 n=11 cnt=3 :L
l=5 n=12 cnt=1 :M
l=1 n=13 cnt=27 :N
l=4 n=14 cnt=2 :O
l=5 n=15 cnt=1 :P
l=3 n=16 cnt=6 :Q
l=5 n=17 cnt=1 :R
l=4 n=18 cnt=3 :S
l=5 n=19 cnt=1 :T
l=2 n=20 cnt=13 :U
l=4 n=21 cnt=2 :V
l=5 n=22 cnt=1 :W
l=3 n=23 cnt=6 :X
l=5 n=24 cnt=1 :Y
l=4 n=25 cnt=3 :Z
l=5 n=26 cnt=1 :[
mid=4.037037
```



Двоичное дерево

```
//-----  
FILE *fd=fopen("revolution.txt","r");  
ph=NULL;  
while(fgets(c,80,fd)!=NULL) {  
    c[strlen(c)-1]=0; // Затереть \n в конце "aaaaa\n"  
    insert(ph,strdup(c));  
}  
fclose(fd);  
puts("-----");  
k=0;scan(ph,1,k);  
printf("mid=%lf\n",sum(ph,1)/ph->cnt);  
getchar();  
balance(ph);  
puts("-----");  
k=0;scan(ph,1,k);  
printf("mid=%lf\n",sum(ph,1)/ph->cnt);  
getchar();  
insert(ph,"Aaaaa");  
insert(ph,"Bbbbbb");  
insert(ph,"Ccccc");  
insert(ph,"Ddddd");  
insert(ph,"Eeeee");  
insert(ph,"Fffff");  
puts(remove(ph,14));  
puts(remove(ph,14));  
puts(remove(ph,10));  
puts(remove(ph,10));  
puts("-----"); k=0;scan(ph,1,k);  
getchar();  
}
```

```
-----  
l=2 n=0 cnt=11 :  
l=3 n=1 cnt=10 :  
l=4 n=2 cnt=9 :  
l=6 n=3 cnt=5 :  
l=8 n=4 cnt=3 :  
l=9 n=5 cnt=2 :  
l=10 n=6 cnt=1 :  
l=7 n=7 cnt=4 :All i c  
l=5 n=8 cnt=8 :Alright  
l=6 n=9 cnt=2 :Alright  
l=7 n=10 cnt=1 :Alrigh  
l=1 n=11 cnt=38 :Beatl  
l=6 n=12 cnt=1 :But if  
l=5 n=13 cnt=7 :But wh  
l=8 n=14 cnt=1 :But wh  
l=7 n=15 cnt=4 :Don't  
l=8 n=16 cnt=2 :Don't  
l=9 n=17 cnt=1 :Don't  
l=6 n=18 cnt=5 :Don't  
l=4 n=19 cnt=12 :We al  
l=5 n=20 cnt=4 :We all  
l=7 n=21 cnt=1 :We all  
l=6 n=22 cnt=3 :We'd a  
l=7 n=23 cnt=1 :We're  
l=3 n=24 cnt=22 :Well  
l=4 n=25 cnt=9 :Well y  
l=6 n=26 cnt=7 :Well y  
l=8 n=27 cnt=4 :Well y  
l=9 n=28 cnt=3 :Well y  
l=10 n=29 cnt=2 :Well  
l=11 n=30 cnt=1 :You a  
l=7 n=31 cnt=6 :You as  
l=8 n=32 cnt=1 :You be  
l=5 n=33 cnt=8 :You sa  
l=2 n=34 cnt=26 :You s  
l=4 n=35 cnt=2 :You say you'll change the const  
l=5 n=36 cnt=1 :You tell me it's the institution  
l=3 n=37 cnt=3 :You tell me that it's evolution  
mid=6.078947  
  
l=5 n=0 cnt=1 :  
l=4 n=1 cnt=3 :  
l=5 n=2 cnt=1 :  
l=3 n=3 cnt=8 :  
l=5 n=4 cnt=1 :  
l=4 n=5 cnt=4 :  
l=5 n=6 cnt=2 :  
l=6 n=7 cnt=1 :All i can tell you  
l=2 n=8 cnt=18 :Alright alright  
l=5 n=9 cnt=1 :Alright alright  
l=4 n=10 cnt=4 :Alright alright  
l=5 n=11 cnt=2 :Beatles - Revoluti  
l=6 n=12 cnt=1 :But if you go carr  
l=3 n=13 cnt=9 :But when you talk  
l=5 n=14 cnt=1 :But when you want  
l=4 n=15 cnt=4 :Don't you know it'  
l=5 n=16 cnt=2 :Don't you know it'  
l=6 n=17 cnt=1 :Don't you know kno  
l=1 n=18 cnt=38 :Don't you know yo  
l=5 n=19 cnt=1 :We all want to cha  
l=4 n=20 cnt=4 :We all want to cha  
l=5 n=21 cnt=2 :We all want to cha  
l=6 n=22 cnt=1 :We'd all love to s  
l=3 n=23 cnt=9 :We're doing what w  
l=5 n=24 cnt=1 :Well you know  
l=4 n=25 cnt=4 :Well you know  
l=5 n=26 cnt=2 :Well you know  
l=6 n=27 cnt=1 :Well you know  
l=2 n=28 cnt=19 :Well you know  
l=5 n=29 cnt=1 :Well you know  
l=4 n=30 cnt=4 :You ain't going to  
l=5 n=31 cnt=2 :You ask me for a c  
l=6 n=32 cnt=1 :You better free yo  
l=3 n=33 cnt=9 :You say you got a  
l=5 n=34 cnt=1 :You say you want a  
l=4 n=35 cnt=4 :You say you'll cha  
l=5 n=36 cnt=2 :You tell me it's t  
l=6 n=37 cnt=1 :You tell me that i  
mid=4.500000
```



Двоичное дерево

```
-----
l=5 n=0 cnt=1 :
l=4 n=1 cnt=3 :
l=5 n=2 cnt=1 :
l=3 n=3 cnt=8 :
l=5 n=4 cnt=1 :
l=4 n=5 cnt=4 :
l=5 n=6 cnt=2 :
l=6 n=7 cnt=1 :All i can tell you
l=2 n=8 cnt=18 :Alright alright
l=5 n=9 cnt=1 :Alright alright
l=4 n=10 cnt=4 :Alright alright
l=5 n=11 cnt=2 :Beatles - Revoluti
l=6 n=12 cnt=1 :But if you go carr
l=3 n=13 cnt=9 :But when you talk
l=5 n=14 cnt=1 :But when you want
l=4 n=15 cnt=4 :Don't you know it'
l=5 n=16 cnt=2 :Don't you know it'
l=6 n=17 cnt=1 :Don't you know kno
l=1 n=18 cnt=38 :Don't you know yo
l=5 n=19 cnt=1 :We all want to cha
l=4 n=20 cnt=4 :We all want to cha
l=5 n=21 cnt=2 :We all want to cha
l=6 n=22 cnt=1 :We'd all love to s
l=3 n=23 cnt=9 :We're doing what w
l=5 n=24 cnt=1 :Well you know
l=4 n=25 cnt=4 :Well you know
l=5 n=26 cnt=2 :Well you know
l=6 n=27 cnt=1 :Well you know
l=2 n=28 cnt=19 :Well you know
l=5 n=29 cnt=1 :Well you know
l=4 n=30 cnt=4 :You ain't going to
l=5 n=31 cnt=2 :You ask me for a c
l=6 n=32 cnt=1 :You better free yo
l=3 n=33 cnt=9 :You say you got a
l=5 n=34 cnt=1 :You say you want a
l=4 n=35 cnt=4 :You say you'll cha
l=5 n=36 cnt=2 :You tell me it's t
l=6 n=37 cnt=1 :You tell me that i
mid=4.500000
```

```
-----
l=5 n=0 cnt=1 :
l=4 n=1 cnt=3 :
l=5 n=2 cnt=1 :
l=3 n=3 cnt=9 :
l=5 n=4 cnt=1 :
l=4 n=5 cnt=5 :
l=5 n=6 cnt=3 :
l=7 n=7 cnt=1 :Aaaaaa
l=6 n=8 cnt=2 :All i can tell you is brother you ha
l=2 n=9 cnt=18 :Alright alright
l=5 n=10 cnt=1 :Bbbbbb
l=4 n=11 cnt=2 :Beatles - Revolution (lennon/mccart
l=3 n=12 cnt=8 :But when you want money for people
l=5 n=13 cnt=2 :Ccccc
l=6 n=14 cnt=1 :Ddddd
l=4 n=15 cnt=5 :Don't you know it's gonna be alrigh
l=5 n=16 cnt=2 :Don't you know it's gonna be alrigh
l=6 n=17 cnt=1 :Don't you know know it's gonna be a
l=1 n=18 cnt=40 :Don't you know you can count me ou
l=6 n=19 cnt=2 :Eeeee
l=7 n=20 cnt=1 :Fffff
l=5 n=21 cnt=3 :We all want to change the world
l=4 n=22 cnt=6 :We all want to change the world
l=5 n=23 cnt=2 :We all want to change your head
l=6 n=24 cnt=1 :We'd all love to see the plan
l=3 n=25 cnt=11 :We're doing what we can
l=5 n=26 cnt=1 :Well you know
l=4 n=27 cnt=4 :Well you know
l=5 n=28 cnt=2 :Well you know
l=6 n=29 cnt=1 :Well you know
l=2 n=30 cnt=21 :Well you know
l=5 n=31 cnt=1 :Well you know
l=4 n=32 cnt=4 :You ain't going to make it with any
l=5 n=33 cnt=2 :You ask me for a contribution
l=6 n=34 cnt=1 :You better free your mind instead
l=3 n=35 cnt=9 :You say you got a real solution
l=5 n=36 cnt=1 :You say you want a revolution
l=4 n=37 cnt=4 :You say you'll change the constitut
l=5 n=38 cnt=2 :You tell me it's the institution
l=6 n=39 cnt=1 :You tell me that it's evolution
```



Двоичное дерево в массиве

ООП-нотация

```
// Двоичное дерево в массиве
struct btree{
    char **p;
    int sz;
}

//-----
void init(){ sz=10; p=new char*[sz];
    for (int i=0;i<sz;i++) p[i]=NULL;
}

//-- число вершин в поддереве
int size(int n){
    if (n>=sz || p[n]==NULL) return 0;
    return 1+size(2*n)+size(2*n+1); }

//-- Обход дерева
void scan(int n, int level, int &ln){
    if (n>=sz || p[n]==NULL) return;
    scan(2*n,level+1,ln);
    printf("l=%d n=%d :%s\n", level, ln, p[n]);
    ln++;
    scan(2*n+1,level+1,ln);}

//-- Поиск вершины по логическому номеру
char *get_n(int m, int n){
    if (m>=sz || m>=size(n)) return NULL;
    int ll=size(2*n);
    if (m<ll) return get_n(m,2*n);
    m-=ll;
    if (m-- ==0) return p[n];
    return get_n(m,2*n+1);
}
```

```
// Включение с сохранением порядка
void insert(int n, char *ss){
    if (n>=sz){
        sz*=2;
        p=(char**)realloc(p,sz*sizeof(char*));
        for (int i=sz/2;i<sz;i++) p[i]=NULL;
    }
    if (p[n] == NULL) { p[n]=ss; return; }
    if (strcmp(ss,p[n])<0)
        insert(2*n, ss);
    else
        insert(2*n+1, ss);}
```



Двоичное дерево в массиве

ООП-нотация

```
//--- Балансировка дерева
void balance(char *pp[], int a, int b){
    if (a>b) return;
    int m=(a+b)/2;
    insert(1,pp[m]);
    balance(pp,a,m-1);
    balance(pp,m+1,b);
}

//-- Обход дерева с сохранением в линейном массиве ук
void set(char *pp[], int n, int &ln){
    if (n>=sz || p[n]==NULL) return;
    set(pp,2*n,ln);
    pp[ln++]=p[n];
    set(pp,2*n+1,ln);}

void balance(){
    int sz1=size(1),ln=0;
    char **pp=new char*[sz1];
    set(pp,1,ln);
    delete p;
    init();
    balance(pp,0,sz1-1);
}
```

```
double sum(int n, int level){
    if (n>=sz || p[n]==NULL) return 0;
    return sum(2*n,level+1)+level+sum(2*n+1,level+1);
}

int cnt(int n){
    if (n>=sz || p[n]==NULL) return 0;
    return cnt(2*n)+1+cnt(2*n+1);
}
```



Двоичное дерево в массиве

ООП-нотация

```
int main() {
    btree X;
    X.init();
    int k=0;
    FILE *fd=fopen("revolution.txt","r");
    char c[80];
    while(fgets(c,80,fd)!=NULL) { c[strlen(c)-
    fclose(fd);
    puts("-----");
    k=0;X.scan(1,1,k);
    printf("mid=%lf\n",X.sum(1,1)/X.cnt(1));
    X.balance();
    puts("-----");
    k=0;X.scan(1,1,k);
    printf("mid=%lf\n",X.sum(1,1)/X.cnt(1));
    X.insert(1,"aaaaa");
    X.insert(1,"bbbbbb");
    X.insert(1,"ccccc");
    X.insert(1,"ddddd");
    X.insert(1,"eeeeee");
    X.insert(1,"ffffff");
    puts("-----");
    k=0;X.scan(1,0,k);
    for (k=X.size(1)-1;k>=0;k--) puts(X.get_n(
    getchar();
}
```

```
-----
l=2 n=0 :
l=3 n=1 :
l=4 n=2 :
l=6 n=3 :
l=8 n=4 :
l=9 n=5 :
l=10 n=6 :
l=7 n=7 :All i can tell
l=5 n=8 :Alright alright
l=6 n=9 :Alright alright
l=7 n=10 :Alright alright
l=1 n=11 :Beatles - Revo
l=6 n=12 :But if you go
l=5 n=13 :But when you t
l=8 n=14 :But when you w
l=7 n=15 :Don't you know
l=8 n=16 :Don't you know
l=9 n=17 :Don't you know
l=6 n=18 :Don't you know
l=4 n=19 :We all want to
l=5 n=20 :We all want to
l=7 n=21 :We all want to
l=6 n=22 :We'd all love
l=7 n=23 :We're doing wh
l=3 n=24 :Well you know
l=4 n=25 :Well you know
l=6 n=26 :Well you know
l=8 n=27 :Well you know
l=9 n=28 :Well you know
l=10 n=29 :Well you know
l=11 n=30 :You ain't goin
l=7 n=31 :You ask me for
l=8 n=32 :You better fre
l=5 n=33 :You say you go
l=2 n=34 :You say you wa
l=4 n=35 :You say you'll
l=5 n=36 :You tell me it's the ins
l=3 n=37 :You tell me that it's evolution
mid=6.078947
-----
l=3 n=0 :
l=4 n=1 :
l=5 n=2 :
l=6 n=3 :
l=7 n=4 :
l=8 n=5 :
l=9 n=6 :
l=10 n=7 :All i can tell you is br
l=2 n=8 :Alright alright
l=4 n=9 :Alright alright
l=5 n=10 :Alright alright
l=6 n=11 :Beatles - Revolution <le
l=7 n=12 :But if you go carrying p
l=3 n=13 :But when you talk about
l=5 n=14 :But when you want money
l=4 n=15 :Don't you know it's gonn
l=5 n=16 :Don't you know it's gonn
l=6 n=17 :Don't you know know it's
l=1 n=18 :Don't you know you can c
l=4 n=19 :We all want to change th
l=5 n=20 :We all want to change th
l=6 n=21 :We all want to change yo
l=7 n=22 :We'd all love to see the
l=3 n=23 :We're doing what we can
l=2 n=24 :Well you know
l=3 n=25 :Well you know
l=4 n=26 :Well you know
l=5 n=27 :Well you know
l=6 n=28 :Well you know
l=9 n=29 :Well you know
l=8 n=30 :You ain't going to make
l=7 n=31 :You ask me for a contrib
l=10 n=32 :You better free your mi
l=7 n=33 :You say you got a real s
l=9 n=34 :You say you want a revol
l=8 n=35 :You say you'll change th
l=9 n=36 :You tell me it's the ins
l=10 n=37 :You tell me that it's e
mid=5.894737
-----
mid=5.894737
-----
mid=6.078947
-----
```