**Лекция № 9.** Графтағы алгоритмдер.

**МАҚСАТЫ:** Графтағы алгоритмдер, графтарды талдау, графта бойымен және енімен іздеу, жұптастыру, бояу графтағы тиімді алгоритмдер, Дейкстра, Прим, Краскал алгоритмдері, «алгоритм» ұғымының математикалық моделдерінің эквиваленттілігі, алгоритмдер теориясының тезистері ұғымдарымен таныстыру.

**СҰРАҚТАР**:

1. Графтағы алгоритмдер,
2. Графтарды талдау, графта бойымен және енімен іздеу,
3. Жұптастыру, бояу графтағы тиімді алгоритмдер,
4. Дейкстра,Прим, Краскал алгоритмдері,
5. «Алгоритм» ұғымының математикалық моделдерінің эквиваленттілігі,
6. Алгоритмдер теориясының тезистері.

**НЕГІЗГІ ҰҒЫМДАР:** Графтағы алгоритмдер. Графтарды талдау. Графта бойымен және енімен іздеу. Жұптастыру. Бояу Графтағы тиімді алгоритмдер. Дейкстра,Прим, Краскал алгоритмдері. «Алгоритм» ұғымының математикалық моделдерінің эквиваленттілігі, алгоритмдер теориясының тезистері.

Алгоритмді жобалағанда және дербес жағдайда деректердің құрылымында ағаш кең қолданыста болады. Оқырманды әдебиет жағынан графтың қағидасына көнілін бұрсақ, түйіншек, қабырға, парақ, жұрағат, ұл, сол жұрағат, оң жұрағат, баба, әке, түп, тармақ және басқа да ұғымдарды пайдаланамыз.

**Графта қысқаша жолдарды іздестіру**

*Кіретін деректер:*

- Граф G өлше- қабырғалармен (Салмақ жайлы айтқанда қабырңанын ұзындығы деп түсінуге болады, егер геометриялық граф жайлы айтса немесе басқа да қабырғаның санды мінездемелерінде де солай түсінуге болады)

- s бастаманың шыңы (қалған барлық шыңға дейінгі қашықтықты есептейтін шың)

- dist алабы[1.. n], (dist[i] –s шыңынан i шыңына дейінгі қысқа қашықтық).

- up алабы[1..n], (up[i] –s-тен i-ге дейінгі қысқа жолдың соңынының алдындағы шың).

**Дейкстраның алгоритмі**

Дейкстраның алгоритмі графқа арнайы қарсылы емес салмақты шыңдары бар есептерді дұрыс есептейді. Егер графта қарсы салмағы бар қабырға болса, бірақ қарсы суммарлы салмақпен цикл болмаған жағдайда, есеп шығу үшін Форд немесе Баллманның алгоритмін пайдалансақ болады. 1. up[1.n] алабын нөлдермен толтыру.

2.Әрбір i шыңына dist[i] кілтінің орнынаdist[i] –максималды мүмкін санды (ол графтағы қысқа жолдардың көбісінің ұзындығынан үлкен болуы қажет; есептеу барысында бұл сан азаяды және соңында s шыңынан i шыңына дейінгі қысқа жолдағы ұзындықпен алмасады).

3. Графтың шыңдарынан басымды кезек ұйымдастыру, кілттің орнына *dist* [*i*], *i*= 1, 2, …, *n*. аумағын алу.

4. s шыңының кілтін 0-ге ауыстыру.

5. Әлі де кезек бос емес жағдайында6 – 7 операцияларын орындау.

6. Басым кезектен *r*0 элементін таңдап алу(өшіре);

7.Әрбір r0 шектес r ш 8,9 операциялапын орындау.

8. *delta*= *dist*[*r*] – (*dist*[*r*0] + *L*(*r*0, *r*)) аумағын есептеу;

9. Егер *delta* > 0, онда *r элементінің кілті dist* [*r*] *delta аумағына ке міту және up* [*r*] ескірген аумағын *r*0 ауыстыру;

**Ортақ мәліметтер.** Ізденістің ағаштары сөздіктерді берілгеннің абстракты үлгісі ретінде қарауға арналған. Тал-шыбықтары үшін сөздіктің тамашасының сияқты деректердің абстракт үлгісінің арнаулы.Басымды кезектер сияқты, олар да көпті ұсынады, бірақ басқа операциялы жиынмен осылай, яғни

• ***Search*** - берілген кілтпен элементті іздестіру,

• ***Minimum*** –минималды кілтпен элементті іздестіру,

• ***Maximum*** –максималды кілтпен элементті іздестіру,

• ***Predecessor*** –іздестіру алдыңдағы кілтпен элементті іздестіру,

• ***Successor***–келесі кілтпен элементті іздестіру,

• ***Insert*** –өзінің кілт арқылы элементті енгіз,

• ***Delete*** –көрсетілген элементті өшіру.

Ізденістің ағашы сөздік ретінде де, басымды кезек ретінде де пайдалануы мүмкін.

Негізгі операций орындалуының уақыты ағаштың биіктігіне пропорционал. Егер әрбір жұпты ағаштын ішкі түйіншегінің екі жұрағаты болады, сонда оның ұзындығы және негізгі операциялардыә орындалу уақыты түйінше санының логарифміне пропорционал. Керісінше, егер ағаш н түйіншектен құралған сызықты шынжырды көрсетсе, бұл уақыт до Θ(*n*) дейін көбейеді. Байкаусыз ізденіс жұпты ағашы бұл өзі Ο (lоg *n*) екені белгілі, сондықтан бұл жағдайда Θ(lоg *n*) негізгіі операцияларды есептеу уақыты болады.

Әрине,пайда болатын iздестiру екiлiк ағашының тәжiрибелерiнде кездейсоқтан алыс бола алады. Алайда, бiз ағаштарды теңдеуiш бойымен арнайы өлшемдер қабылданып n түйiндермен ағаштарын биiктiк болғанын кепiлдiк бере аламыз 0(log n ). Тәсiлдемелердiң бiрлерн мысалды төменде қарап шығамыз. (қызыл-қара ағаштар - АВЛ, дербес жағдайда ағаштар).Сонымен бiрге мәлiметтер үшiн әсiресе ыңғайлы кез келген (дискте) рұқсаты бар екiншi жад сақталған Б-ағаштар қарастырылады. Бұл туралы:

Iздестiрудің екiлiк ағашының ұсынысы. Iздестiру екiлiк ағашы әрбiр түйiнiне өлшенген элемент сәйкестiкке орнатылған түбiрлiк екiлiк ағашы деп аталады. Әрбiр түйiн x үшiн келесi шартты орындайды.

X түбiрi бар сол ағаш астындағы барлық х түйiндерiнiң салмағынан аз немесе тең, онының оң жақ ағаш астындасының түйiндерiнiң салмағы артық немесе түйiн x салмағына тең.

*Келесi түрдiң түйiндерiмен мұндай ағаш өкiлдiк етедi*

*Node* = (*element*, *key*, *left*, *right*, *parent*)

T ағашқа рұқсат сiлтеме root көмегiмен жүзеге асырылады.

Екiлiк iздестiру ағашымен операциясы. Көрсетемiз, екiлiк iздестiру ағаштары операция Search, Minimum, Maximum, Successor және Predecessor уақытқа орындауға мүмкiндiк береді (h ), ағаштың биiктiгi h.

1. (Search ) iздестiру. Iздестiрудiң процедурасы k және x iздестiрудi iстелетiн ағаш астынданың түбiрiне көрсеткiш кiре берiске iзделiп отырған кiлт алады. Ол (егер болса) k немесе (егер мұндай шың жоқ жағдайда) nil кiлтпен шыңға көрсеткiштi қайтарады.

**Procedure** Search (x, k);

**begin**

**if**(x = nil) or (k = key [x]) **then** Return x;

**if**(k < key [x]) **then** Return Search (left[x], k) **else**

Return Search (right[x], k)

**end**.

Бiз iздестiрудiң процессiнде түбiрден қозғаламыз, кілтті k кiлтпен салыстыра, x ағымдағы шың сақталған қозғаймыз. Егер ол тең болса, iздестiру аяқталады. Егер k < key[x], онда iздестiру x сол ағаш астындада созылады. Егер k > key[x], онда iздестiру оң ағаш астында созылады. Iздестiру жолының ұзындығы ағаштың биiктiгiн асып түспейдi, сондықтан iздестiрудiң уақыты (h - ағаштың биiктiгi) (h ) O бар.

Процедура iздестiру итератив нобайы

**Procedure** IterativeSearch (x, k);

**begin**

**While** (x ≠ nil) and (k ≠ key [x]) **do**

**If** k < key [x] **then** x:= left[x] **else** x:= right[x];

Return (x)

**end**.

Минимум және максимум. Iздестiру ағашында ең төменгi кiлтпен элемент табуға болады, көрсеткіш left түбiрден өте шыға nil-ге жеткенімізге дейін табуға болады. (x ) Minimum(x) процедура түбiрi бар ағаш астынданың табылған элементiне көрсеткiштi қайтарады.

**Procedure** Minimum(x);

**begin While** left [x] ≠ nil **do** x:= left[x]; Return (x) **end**.

Алгоритм Maximum симметричен:

**Procedure** Maximum(x);

**begin While** (right [x] ≠ nil) **do** x:= right[x]; Return (x) **end**.

Екі алгоритм O(h) уақытты (ағаш бойымен тек қана төмен қозғалғандықтан) ағаштың биiктiгi талап етедi.

Келесi және алдыңғы элементтер. Егер х кейбір ағаштың көрсеткіші болса, сол Successor(x) рәсімі сілтегішті түйіншекке мен кейінгі x элемент немесе nil, егер көрсетілген элемент соңғы ағашқа қайтарады

**Procedure** Successor**(**x**);**

**begin**

**If** (right[x] ≠ nil) **then** Return Minimum (right[x]);

y:= p[x];

**while**(y ≠ nil) and (x=right [y]) **do** {x:= y; y:= parent[y]};

Return y

**end**.

Келтiрiлген процедура екi жағдайды бөлек қарастырылады. Егер шың x оң ағаш астындағысы бос болмаса, онда келесi x элементке - бұл ағаш астындада ең төменгi элемент және ([x ] right) Minimum тең бол. Болғанша, егер шың x оң ағаш астындасы бос болса, онда x жүр өз ата-анасын сол ұл болатын шыңды үстiне таппаймыз. Бұл (егер ол барып тұр) ата-ана және iзделiп отырған элемент болады. Процедура Successor ағашындағы жұмысын уақыты биiктiгінде(h ), өйткенi бiз тек - үстiне, немесе тек қана төмен қарай қозғаламыз. Predecessor процедура симметриялық.

Элементтiң қосымшасы.Процедура Insert (T, z) ағаш T қолайлы орынына элемент берiлгенге толықсытады.

Келтір- рәсім оқшау екі уақиғаны қарайды. x шыңының оң поддерево жым-жылас, сол кейінгі x элементті - ең төмен элемент осы поддереве және Minimum(right[x]) тең. x шыңының оң поддерево жым-жылас, сол вверх от x барамыз, әлі де шыңды таппаймыз, өзінің родителя сол ұлымен болып табыламын. Осы родитель(ол болса) және искомым элементпен болады. Successor рәсімінің жұмысының уақыты биіктіктің h ағашында болады Ο (h), себебі біз либо ғана вверх, либо ғана төмен жүреміз. Predecessor рәсімі симметрична.

Элементтің қосымшасы. Insert(T, z) рәсімі тапсырынды элементті T ағашының лайық жеріне үстейді.

**Procedure** Insert(T, z);

**begin**

y := nil; x := root;

**while**(x ≠ nil) **do** {y := x; **if** key[z] < key[x] **then** x := left[x] **else**

x := right[x]};

p [z] := y;

**if**y = nil **then** root := z **else if** key[z] < key[y] **then** left[y] := z **else**

right [y]:= z

**end**.

қосымша уақытты Ο (h) биіктіктің h ағашы үшін сұрайды.

**Элементті өшіру.** Өшіру рәсімінің өлшемі өшірілетін биіктіктегі z көрсеткіші болып табылады.Өшірудің үш жолы бар.Егер z-тің баласы жоқ болса,онда ата-анасының сәйкес полясына nil-ді қоя салу жеткілікті.Егер z-та бір бала болса, ата-анасымен бірге қалдырып, z-ті алып тастауға болады.Ал егер де баласы екеу болса, z-тің артындағы өз баласы жоқ келесі *y* элементін тауып аламыз.Енді кілт пен қосымша деректерді z биіктігінен *y* биіктігінекөшіруге болады, ал *y* биіктігін жоғарыда көрсетілген әдіспен өшіруге болады.

**Қызыл-қара ағаштар**

*h* биіктіктегі екі жақтыіздестіру ағашының негізгі операциясын O (h) іс-әрекеттері арқылы жүзеге асыруға болады.Егер ағаштардың биіктігі кішкене болса, нәтижелі болады. Операция нәтижесін арттыру үшін ағаштың биіктігі Ο (log *n*) өлшемінде керек болса,

ағаштарды салудың тиімді әдістері қолданылады. Бұндай әдістер ағаштарды тепе-теңдікте ұстау деп аталады. Тепе-теңдік сапасының әр түрлі өлшемдері қолданылады.

Ο (log *n*) өлшемдегі биіктікке қол жету үшін кепіл баға беретін іздеу ағаштарының теңестірілген бір түрі қызыл-қара ағаштар деп аталады.

Осындай **теңестіру** кезінде жиі кездесетіні АВЛ- теңестіру, онда әр бөліктегі сол жақ ағаш түбінің биіктігі оң жағынан бірлікке ғана ерекшеленеді.

**Қызыл-қара ағаштар-** іздеудің кеңейтілген екі жақты ағашы,биіктігі қара және қызыл болып бөлінеді:

Әр бөлігі не қара , не қызыл.

Әр парағы (nil-бөлігі)-қара.

Егер бөлігі қызыл болса, онда екі баласы қара.

1. Бас-басы түйіншек либо қызыл, либо қара.

2. Бас-басы парақ(nil- түйіншек) - қара.

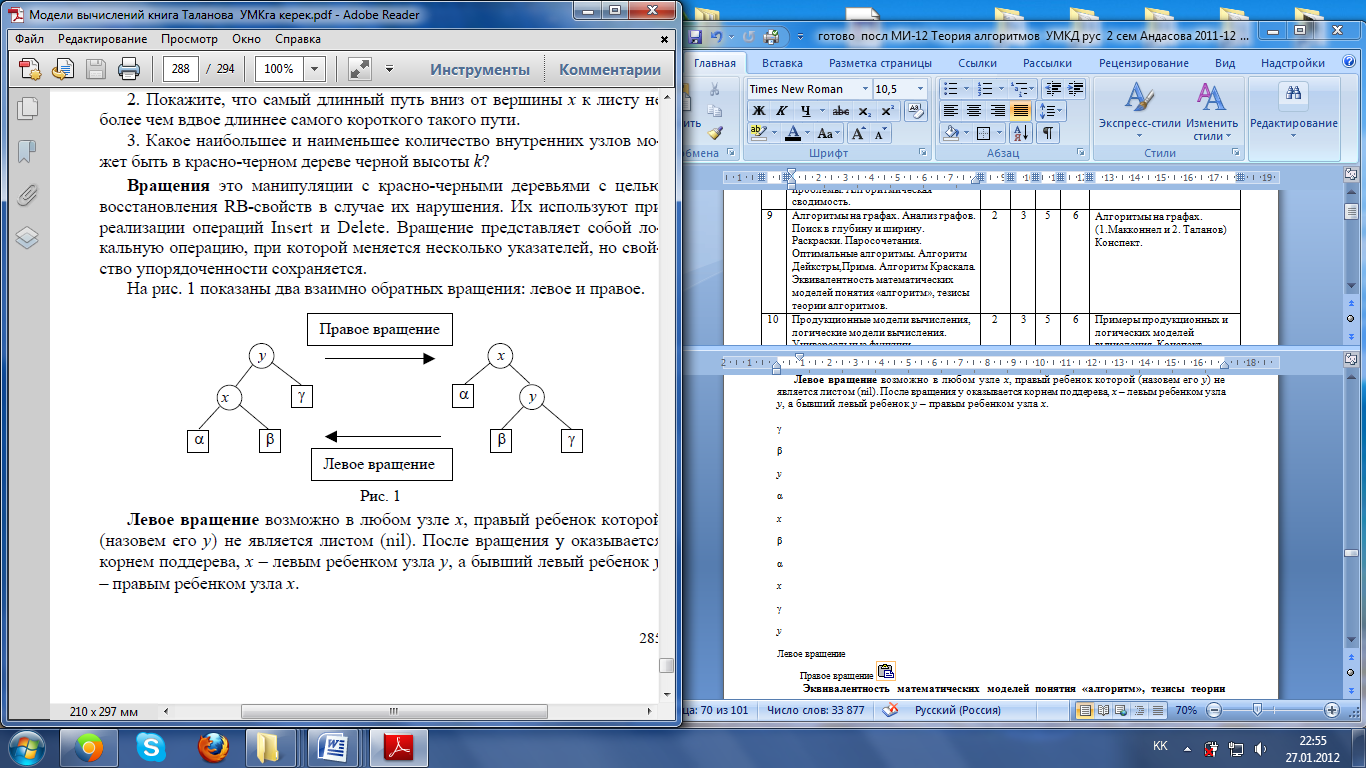
3. Түйіншек қызыл, сол оба оның баласының қара.

4. Төмен түптен жапырақтарға, баратын барлық жолдар қара түйіншектің бірдей санын асырайды.

Түбірінен жапырақтарына дейінгі жолдар бірыңғай мөлшерде қара бөліктерді құрайды. 1-4 дейінгі қасиеттер RB-қасиеттері деп аталады. Қызыл-қара ағашының бөліктерін былай белгілейміз:

*Node* = (*color*, *key*, *left*, *right*, *parent*).

**Айналу** – бұл RB-қасиеттері бүлінген жағдайда қызыл-қара ағаштарды қалпына келтіру үшін жасалатын әрекеттер.Оларды Insert және Delete операцияларын жүзеге асыруда орындаймыз.Айналу кезінде бірнеше көрсеткіш ауысуы мүмкін, бірақ тәртібі сақталатын жүйелі операция болып табылады.

1-суретте біржақты екі айналу түрі көрсетілген:сол жаққа және оң жаққа

Жайылып өсетін ағаштар үшін бір операцияға есептегенде есептік құны *O* (log *n*)-ді құрайды.

Б-ағаштары дегеніміз- магниттік дискіде немесе тікелей жеткізілетін құралдарда ақпаратты сақтауды қамтамасыз ететін теңестірілген ағаштардың бір түрі. Б-ағаштары қызыл–қара ағаштарға ұқсайды.Айырмашылығы мынада: қолданылатын дисктің мінездемесіне қарай Б-ағаштарының бөліктерінде көп бала, іс жүзінде мыңға дейін болуы мүмкін.Осыған байланысты ағаштың биіктігі *O* (log *n*)-ді бағалау нәтижесі бойынша қызыл–қара ағаштарға қарағанда айтарлықтай аз мөлшерде. Қызыл–қара ағаштар сияқты Б-ағаштары да *O* (log *n*) мерзімінде көптеген n өлшемдегі әр қилы операцияларды жүзеге асыра алады.

x түйіншегі, сақта- n[x] кілттердің, n имеет[x] + 1 бала-шағалардың. x кілттерге деген сақтал- қызмет ет- барлық оның бұтақтарын бас n бөлетін шекаралармен[x] + 1 топтардың; үшін бас-басы топты бір x бала-шағаларынан жауап береді. При ізденісте Б-дереве біз искомый кілтті мен n салыстырамыз[x] ара x сақталатын кілттермен, қарамастан және дейін ре-зультатам салыстыр- сайлаймыз

**Дискте берілген деректер құрылымдағы жұмыстардың ерекшеліктері.**

Б-ағаштарымен қызмет атқаратын алгаритмдердің оперативті есте сақтау қабілеті барлық ақпараттың аз мөлшерін ғана (бөліктердің белгілі санын) қамтиды.

Диск естің үлкен аумағы болып табылады, келесі х нысанымен жұмыс істеу үшін біз *Disk-Read*(*x*) арнайы(дискке жазу)операциясын орындауымыз керек.

Бағдарламаның жұмыс уақыты негізінен осы операциялардың санымен анықталады, сондықтан да Б-ағашының бөлігі дисктің бір бөлшегін түгелдей толтыру үшін бар ақпаратты бірден оқып, жазып отыру керек.Осылайша, тармақтану дәрежесі(бөліктегі бала саны) өлшемімен анықталады.

Б-ағашының типтік тармақтану дәрежесі элементтің өлшеміне байланысты 50 мен 2000 арасында болады. Іздеу барысында тармақтану дәрежесінің артуы ағаш биіктігі мен дискке оралу санын тез арада қысқартады. Мысалы,биіктігі 2 және 1001 дәрежедегі Б- ағашы миллиардтан астам кілтті сақтауы мүмкін. Тамырды үнемі оперативті есте сақтауға болатынын ескерсек, керекті кілтті іздегенде, дискке екі рет оралу жеткілікті.

**Ескерту.** Б-ағаштарының басқа жиі пайдаланылады ақпарат көп орыны бар жапырақтарда орналасады, себебі кілтті сақтамауға да болады, ал ішкі бөліктерде тек қана кілттер мен балаларға көрсеткіштер сақталады.

**Б-ағаштарымен жүргізілетін негізгі операциялар.** Б-ағашының тамыры әрқашанда оперативті есте болады деп санауға болады,өйткені дисктен оқу операциясы тамыр үшін қажет етілмейді; дегенмен де біз ылғи да тамырды өзгерткенде, дискте сақтауымыз керек.

**Б-ағашында іздеу.**Б-ағашында іздеу қос ағашта іздеуге ұқсас. Айырмашылығы мынада: біз х-тің әр бөлігінде (*n*[*x*] + 1)-тің екеуін емес, бір вариантын таңдаймыз.Іздеу барысында ағаштың тамырдан жапырақтарына дейінгі бөліктері қаралады.Сондықтан да дискке оралу көрсеткіші θ(*h*) = θ(log*t n*) тең, мұндағы *h* – ағаштың биіктігі, ал *n* – кілттер саны. *n*[*x*] ≤ 2*t* болғанда, онда while циклы *O*(*t*) рет, және де санау уақыты *O*(*th*) = *O*(*t*⋅log*t n*)-ға тең.

және оны орналастыратын іс-шара арқылы жасалады.Оны *O*(1) уақыт ішінде дисктен оқу операциясын қолданбай-ақ жүзеге асыруға болады.

Y бөлігінде қалыптасқанша 2*t* бала болды ; қалыптасқаннан кейін t ең кішісі қалды, ал қалғандары *t жаңа z бөлігінің балалары болады*, олар өз кезегінде х бөлігінің баласы болады. *y* бөлігінің медиана-кілті *x* бөлігіне қосылады да*y* бөлігі мен оның артындағы *z* бөлігін айырып тұрады.

Элементке қосылған Б-ағашы. Бұл операция қолданудан алдын процедурада қолданылады.Барлығының сынуынан бөліп екі бөлікке бөлінеді.Атқарушы t-1 элементі әрқайсысында болады.

Алдын бұл кілт-медианасы key [y] жіберіледі. Ол х бөлігі у қойылады. Бұл мән орындалған жағдайда х толық болмайды. Алайда у-түбір процедурасы анологиялық жүмыс атқарады. Бұл жағдайда ағаш ұлғайады.

Insert процедура Б-ағашқа к элемент қосады. Түбірден жапыраққа дейін бір рет өтеді. *O*(*th*) = *O*(*t*·log*t n*) уақыт керек және *O*(*h*) дискіге қарасты. Ағаштың биіктігі һ болса жұмыстың барысы бойынша ұсақтау процедурасы бойынша кездесетін толық бөлінеді. Толық бетпен ауыстыратын, яғни толық емес бетке жақын болады.Ал оны бөлуге болады. Ол оған жақын қосымшалар. Кілті сол үшін жоғары көтеріледі және оны толық емес элементтерге қосамыз.

Б-ағашын өшірілуі. Б-ағашының элементтен өшірілуі ол аналитикалыққа кіреді,қиын болмаса да. Оқырмандарға көрсетілетін процедураларда мүмкін өшіріледі. Бөлікте құралған О һ жүктеледі. Б-ағаштан ұзын һ, сосын оған барлық процедуралар жүктеледі *O*(*t*·*h*) = *O*(*t*·log*t n*).

Қорытындыда байқайтынымыз аға балансталған және Б-ағашы кітаптан қарастыру болады. Кнута Ахо, Хопкрофта және Ульмана және Седжвика. Тура баламадан Б-ағашын кітаптан Кормена және басқалар Тибас және Сиджвик қарастырады және олардың әр түрлі болатынын айтты. Ағаштың құрамында қосылатын қызыл-қара және 2-3-4 ағаштар.

1970 жылы Хопкрофт бұл тұжырымдамаларды 2-3 ағаштарды және алдында айтылған Б-ағашын және 2-3-4 ағаштарды қарастырды. Бұл ағаштардың әрқайсысы бұтақтарды немесе үш бұтақтан бола алады. Б-ағашын арнайы Байером және Мак Крейтом 1972 жылы қарастырды.

**Кеңістікте іздеу.**

Жоғары алгоритмнің жұмысы, жоғары нүктелері мен қабырғаларын қарастырады. Қандайда бір жағдайда орындалатын нүктелерді алдыннан қарастырады. Қарастырылмаған шыңда жана деп алайық. Соған қатысты қарастырылған шың ашық болып қала береді.Сосын ғана жабық болады.

Қабырғаны кеңдігіне қарай қарау идеясының мәні алдын ала тандалған немесе берілген басты а шыңының кезекпен өшірілгендерді қарастыра алуында. Басқаша айтқанда, ең алдымен а шыңның өзі қарастырылады, сосын барлыщ шыңдар, а-мен қосылған,яғни 1 қашықтығының арасында орналасқан,ал содан кейін 2 қашықтығының арасында орналасқандар және т.б.

а бастамалық шыңға іздеу алгоритмын қарастырайық. Ең бірінші а шыңы қарастырылады, ол жалғыз ашық шыңға айналады. Бұл шың активті болады. Әрі қарай әрбір қадам х ашық шыңынан басталады. Ары қарай активті шыңға қатысты қабырғалар зерттеледі. Егер бұндай қабырға жаңа х шыңы у-ке қосылса, онда у шыңы қарастырылады да, ашық шыңға айналады. Активті шыңға қатысты зертелгеннен кейін, ол активті болмай, жабыққа айналады. Содан кейін жаңа активті шың тандалады, және жұмыс барысы қайталанады.Көптеген ашық шыңдар бос болғанда, процессс аяқталады.

Кеңістікте іздеудің басты ерекшелігі, ол графқа көшу амалы басқашалығында, Алдындағы ашық шыңдардың бірінен тандалынады. Стартқа шың жақын болса, ол тез қарастырылады.

(BFS –ағылшын тіліндегі алгоритмнің мағынасы – Breadth First Search) біт старттық а шыңынан кеңістіктегі іздеу шыңындағы процедурасын қарастырайық. Көптеген барлық шың қарастырылады. Х шыңына жақын, *Q-ашық шыңның кезегі.*

**Procedure** BFS(*a*)

1 *a шыңын қарастыру*

2 *a* ⇒*Q*

3 **while** *Q* ≠ ∅**do**

4 *x* ⇐*Q*

5 **for** *y* ∈*V*(*x*) **do**

**6 исследовать ребро (x, y)**

**7 if вершина y новая**

8 **then***y шыңын қарастыру*

9 *y* ⇐*Q*

**Тереңдігіне қарай іздеу әдісі.**

Тереңдігіне іздеу-графты көшіру стратегиясындағы көптік орнатылғандардан ең маңыздысы юолып табылады . Бұл қадам жаңа ойларды орындай алатындай және де жанындағылар зерттелгенде, бір қадам артқа тастайды. Тереңдәкке іздеу методы кхптеген аттармен анықталған, мысалы «бэктрекинг», «қайтаруымен іздеу».

Жаңа ұғымдар, ашық және белсенді шыңдардың ізденісі үшін осындай тереңждік мағынасына ие. Және де сол сияқты ізденісте енеді. Бұл бір белсенді шыңының ылғи астамын ар екенін белгілейміз.

Айналып кетуінің орындалуы бастапқы шыңының қатысуымен дегеннен басталады, белсенді қайсыбір және бірден-бір ашық шыңмен болады. Кейін қақтығыс шыңға А қабырға және у шыңы қатысуымен таңдалады. Ол ашық және белсенді болады. Ізденісте енді А шыңы белсенді қалғаны осы кездерден бйқаймыз. Оған барлық қақтығыстың қабырғаларын әлі де зерттеуге болмайды. Ендігі осы сияқты және алдыңғы ізденісте ірбңр кезекті адым ашық шыңның көпшілігінен таңдау бір ғана емес зерттеу қабырғалардан, (х,у) әйтпесе сол қабырға зерттеледі. У жаңа шыңы, ол ашыққа дегенінің қатысуымен айгалады.( уа.

Басты өзгелік ендігі ізденістен, алдыңғы ізденіске тереңдігінің аралық сапасы белсенді, сол ашық шыңдардан ғана құрылады. Нешінші рет шыңның қатысуы соңғы болады. Ол үшін талғамның мынадай ережесінің жүзеге асуы ашық шыңының көпшілігіне сақтап, құрылымымен жіңішесі болфп табылады. Аш-шыңдар ғана осы тәртіпте жіңішке болып бүктеледі, олар арадағы қандай да бір ашылыстың, ал аралық сапа белсенді соңғы шың шық. Сол жүйілік 2.2 суретте көрсетілген.

Жіңішке үшін ашық шыңдарды Sарқылы белгілейміз, қалған белгілер баяғы мағынаны сақтайды, және де алғашқы тараудағы top(s) арқылы мағынасы аз ғана болады. Соңғы қосылған элемент сыртының элементі. Байлныстың бір компоненті айналып кетуінің рәсімі ізденістер әдісінің тереңдікке А шыңымен бағаланады. Мүмкін сонда келесі бейнемен жазуға болады.

**Procedure** DFS(*a*)

1 А шыңына тоқталу

2 *a*⇒*S*

3 **while** *S* ≠ ∅**do**

4 *x* = top(*S*)

5 **if** емес зерттеу қабырғасы (*x*, *y*) бар

6 **then** зерттеу қабырғасы (*x*, *y*)

7 **if** у жаңа шың

8 **then** у шыңына тоқталу

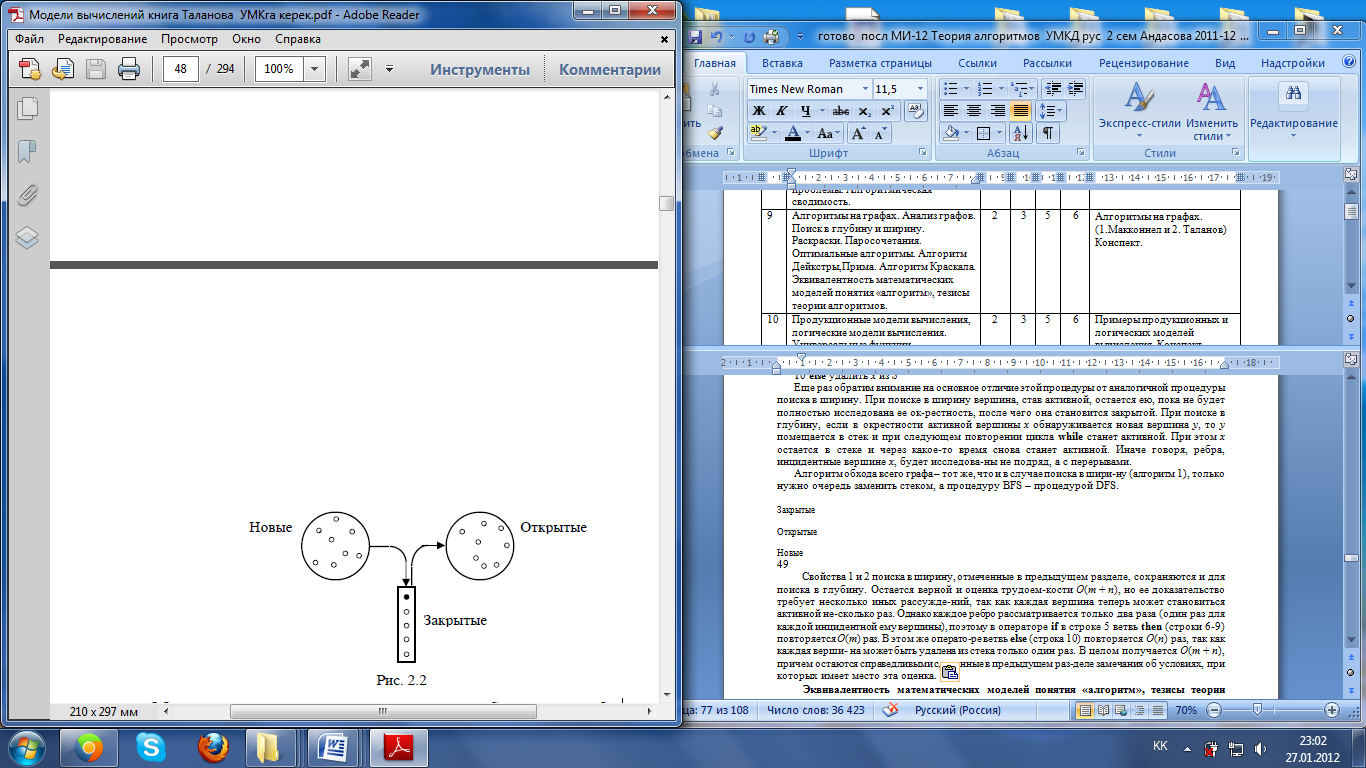
9 *Sy*⇒

10 **else**х-ті s –тен алыстату

Тағы бір рет көңіл аударсақ осы рәсімінің негізгі өзгелігіне ізденістер тәрізді рәсімін енді көзге іле аламыз. Алдыңғы ізденісте шың белсенді қойылып, сонымен қалады, әлі-де толық зерттелмегендіктен оның маңайы болмайды., одан кейін ол берік болады.

Алдыңғы ізденісте тереңдігінің, х белсенді шыңының маңайында у жаңа шыңы кездессе, сол стекке салады. Және алдыңғы while топтамасының келесі қайталаушылығында белсенді болады. Бұл ретте х стекте қалады және әлдеқайдағы уақыт аралығында тағы белсенді бола түседі. Әтпесе айтарлық, өақтығыстар х шыңының қабырғалары үздік-үздік зерттеуі қатар болады.

Алгоритмнің барлық бағандарының айналып кетуі, сол жағдайдағы кеңейуінің арасындағы ізденіс. (1-алгоритм). Тек қана стекті ауыстыруы керек. Ал рәсімі BFS-рәсімі DFS.



1-ші және 2-ші сипаттау ізденісі алғашқы тарауды белгіленуі сақталдаы. Және де ізденісі үшін тереңдетіледі. O(m+n) дұрыс арапшылығы қалады, бірақ онығң айғағы бірнеше өзге пайымдарын сұрайды . себебі, енді сол сияқты бірнеше шың реттің белсенді болуы мүмкін. Алайда бас-басы қабырғаға ғана екі рет қарастырылады, сол себептен IF операторы 5 ші жолда then (6-9 жолдар) реттің тармағы . O(m) қайталанылады. Осы оператордың else (10 жол) реттің тармағы O(n) қайталанылады, сол сияқты әрбір бастапқы шыңы стектен бір рет алыстатылуы мүмкін. Арадағы бүтіндікте O(m+n) алынады. Алғашқы тарауда ескертпелер туралы сол жерді сарапшалаған әділдік жасағандар қалады, алдындағы жер нешінші рет сол бағаны қамтиды.

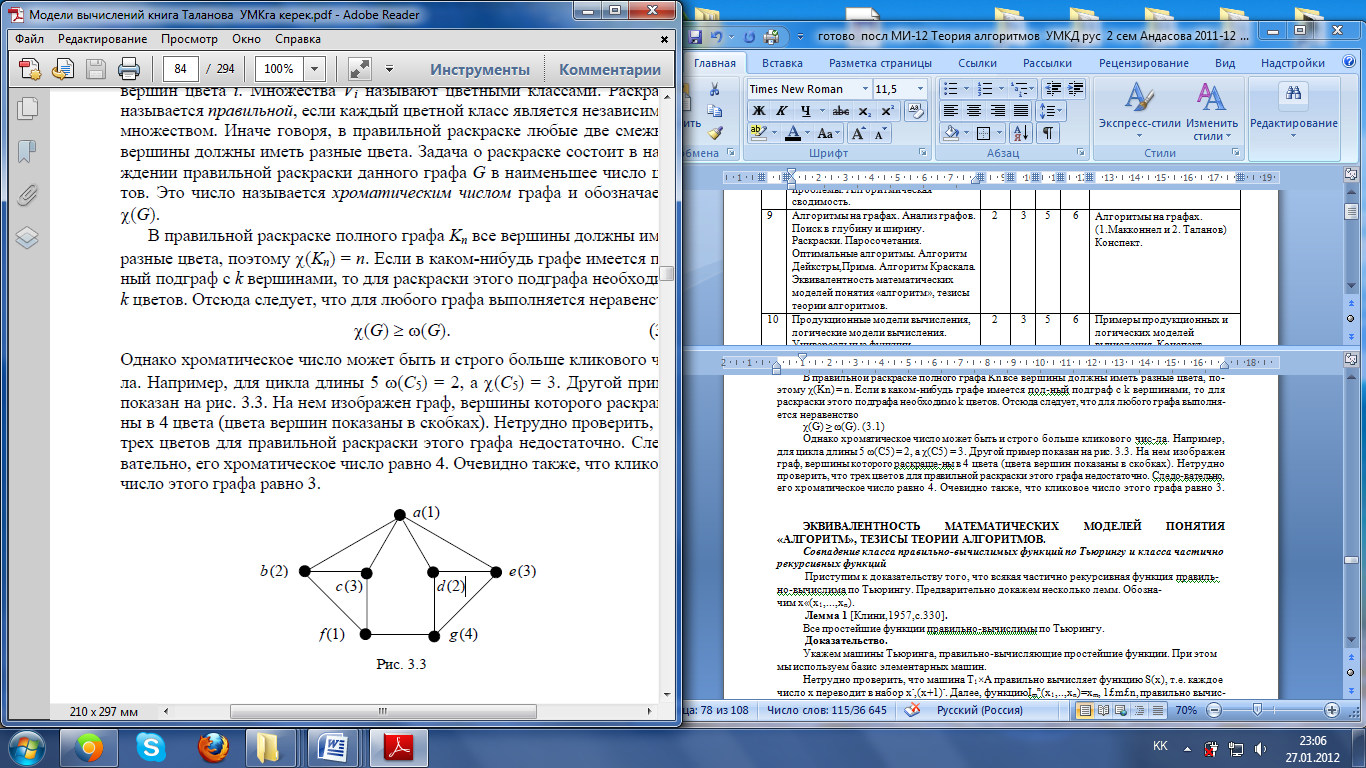
Шыңның өрнегі

Шыңның өрнегінің бағаны деп-түстің мақсатындағы оның шыңдарын айтады. Кәдімгі түстер –ол 1,2...,к сандары. Онда өрнек функция болып табылады, белгілі бір көпшілікте шыңның ббағаны және қабылдаушы тағайынды көпшілікті шың сияқты{1,2,...,3}. Шыңды тағы бөлу көпшілік шыңы ретінде қарастыруға болды. *V* = *V*1 ∪*V*2 ∪ … ∪*Vk*, *Vi* – і түсінің көпшілігі үшін. *Vi* –көпшілкгі түсті сыныптармен атайды. Өрнек дұрыс деп аталады-егерде әрбір түс көпшілікті сыныптармен тәуелсіз болса, айтпесе дұрыс шыңның әрбір екі шектес шыңы әртүрлі түстерді қажет етеді. Тапсырыс туралы өрнекте түстің аздаған санына айталмыш бағаны G дұрыс өрнегін табу құрылады. Сол сан хроматикалық сан бағаны деп аталады және көрсетіледі x(G)

Дұрыс шыңның толық бағаны knбарлық өрнегі бөлек –бөлек түстерді керек етеді. Сол себептен χ(Kn) = n. Егерде қайсы-бірі бағанда толық подграф шыңдарымен, сол осы шыңның мына подграфы к түстерін қажет етеді. Осыған тиіс кез келген бағаны үшін емесабатшылық орындалады. χ(G) ≥ ω(G). (3.1)

Алайда хроматиклық сан үлкен қатал сан болуы мүмкін. Айталық: ұзындықтың топтамасы үшін 5 ω(C5) = 2, ал χ(C5) = 3. Басқа мысалдар 3.3 сіретте көрсетілген.

Онда бағаны нешінші болған шыңдары 4-түсте (шыңның түстері жақшада көрсетілген) бейнеленген. Оны тексеру қиын емес, үш түс осы дұрыс боялған оның бағаны жеткіліксіз. 4-мысалда осы бағанның емесабытшылығы орындалады.



**Қабырғаның өрнегі**

Түзумен берілген тапсырма туралы шыңның өрнегінің тапсырмасы туралы бағананың қабырғасының өрнегінде бар түстер қабырғаларына қашан тағайындалады. Қабырғаның өрнегі –көрінген ортақ шың, берілген екі қабырғада аламаштаса дұрыс болады. Боялған әртүрлі түстермен. Түстің бағаны G қабырғасының дұрыс өрнегі үшін қжетті, ең төмен саны бағанның хроматикалық әріпсанымен аталады және

X’(G) арқылы көрсетіледі. Δ(G) арқылы шыңның χ′(*G*). Ең көп дәрежесін белгілейміз.

Сәйкестік және қабырғаның жабындылары. Үмітсіз ортақ шыңдар сәйкестік болғанда қабырғаның көпшілігі қос-қостан аталады. Сәйкестік бағандар көпшілікті өрнегі деп-қабырғаның ортақ шыңы болмау н айтамыз. Тапсырма сәйкестік жайлы ол берілген бағанда сәйкестік үлкен санның өрнегімен құрылады. Ол сан баған үшін G деп алып оны π(G). айтамыз. Өрнекті жабық бағанмен мынадай көпшілікте аталады, егерде оның әрбір өрнегінің ұштарының бағасы қақтығыс болуы бірде бір өрнекпен болуы мүмкін. Қабырғасының аздаған саны қабырғада жабық бағанды G арқылы ρ(G) белгілейміз. Қабырғаның жабындысының шыңдармен оңаша өмір сүруі ғана қалғанын байқаймыз.

Сәйкестік ұйғарымы сияқты шыңның тәуелсіз көпшілігінің ұйғарымын, сәйкестік анда санда қабырғанңы тәуелсіз көпшілігін айтамыз. Сол аналогия тағы тығыз байланыстың өрнегінің жабындыларының арасында қатайады. Сол сияқты шыңның өздерінің арасындағы жабынды тәуелсіз және тоқулы көпшіліктер келісімі. Тіпті бірдей сан білдірсе осы байланысты, ашып-жарып мынадай көріністі ескертеміз, G тәуелсіздігінің а (сандары).

**В-ағаштар**

Сыртқы жадыдағы деректерді іздеу үшін негізгі «ағаш тәріздес» аппараты В-ағаштар болады. Осы механизмдер негізінде келесі ойлар бар. Біріншіден, жалпы қолжетімдік уақыты дәйекті түрде орналасқан мәліметтер көлемімен емес, магниттік шаласының келу уақытымен анықталатын сыртқы жадыдағы деректер құрылымы жайлы сөз қозғағаннан кейін, бір рет сыртқы жадыға сұрау салған соң бірнеше мәлімет алған тиімді, сонымен қатар бұл жерде негізгі жадының үнемді қолданылуын ескеру қажет. Негізгі жадыны ұйымдастыруды бірдей көлемді парақтар жинағы түрінде қарастыратын болсақ, онда осы парақты сыртқы жадымен мәлімет алмасу бірлігі деп қарастырған жөн. Екіншіден, сыртқы жадыдағы іздеу құрылымын құрастырудың тиімді түрін қолану барысында кез келген кілт бойынша ақпаратты іздеу саны алдын ала мәлім сыртқы жадымен ақпарат алмасуды қажет етуі тиіс.

Классикалық B-ағаштар

Классикалық B-ағаштар механизмі 1970 жылы Бэйер және Маккрейтпен ұсынылған. N ретінің B-ағашы сыртқы жадының сатылы байланысқан беттері (ағаштың әрбір басы – бет) түрінде бейнеленіледі, олар келесі қасиеттерге ие:

Әрбір бет 2\*n элементтерінен көп емес (кілтпен бірге жазбалар).

Әрбір бет түбірліктен өзге, n-нен кем емес элементтен құралады.

Егер В-ағаштың ішкі (беттік емес) басы m кілттерінен құралса, онда m+1 бет-тұқымдары бар.

Барлық парақтық беттер бір деңгейде орналасқан.

B+-ағаштар

Классикалық В-ағаштар ұйымдастырылу сызбасы қарапайым, дегенмен практикалық қолданыс үшін тиімді емес. Оның ең басты себебі практикалық қолданыста көбіне сыртқы жадыда тек қана кілттерді емес, сонымен қатар жазбаларды сақтау қажет. В-ағашында элементтер ішкі де, сыртқы да парақтық беттерінде орналасады, ал жазба көлемі үлкен болуы мүмкін. Ішкі беттері көп элементтерден құрала алмайды, оның себебі ағаш терең болуы мүмкіндігінде. Осыған орай, ағаштың төменгі деңгейлерінде кілттер мен жазбаларға қолжету үшін сыртқы жадымен көп мәлімет алмасу қажет.

Деректер қорындағы индекстерді ұйымдастыру үшін B+-ағаштар түрлері

B+-ағаштар деректер қорларындағы индекстерді ұйымдастыру үшін қарқынды түрде қолданылады. Ең алдымен, аталмыш ағаштар екі қасиеттерімен айқындалады: кез келген кілт пен тақырыпты іздеу үшін сыртқы жадымен мәлімет алмасу санының алдын ала болжанылуы мен ағаш бұтақтарының көп болуына байланысты осы мәлімет алмасу санының ең үлкен кестелерді индекстеу кезінде де көп болмауы.

R-ағаштар және олардың кеңістік деректер қорындағы индекстерді ұйымдастыру үшін қолдану

Кеңістік деректер қоры индексін ұйымдастыру үшін қолданылатын В-ағаштары механизмінің тағы бір ұлғайған түрі – R-ағаштары. В-ағаштары сияқты олар бұтақ тәріздес болып келеді, алайда R-ағаштарындағы мәлімет В-ағаштарындағы мәліметтерден айырмашылығы бар. Парақтық беттерде орналасқан кеңістік объектілердің идентификаторларынан өзге R-ағаштарында индекстелетін объектінің шекаралары туралы ақпарат сақталады.

**Сыртқы жадыдағы іздеу үшін хештеу әдістері .**Хэштеу негізіндегі (қажет мәліметтерді бір рет сұраудан соң алу мүмкіндігі) сыртқы жадыдағы мәліметтерге қол жеткізу идеясы тиімділігі сонша, одан бас тарту қиын. Сыртқы жадыдағы мәліметтерді хеширлеудің авторы Витольд Литвин болып табылады. Бастапқы идея айқын көрінеді: егер хэштеу негізінде мәліметтерді басқару кезінде негізгі жадыда хэш-функция қажетті элементтің адресін өндірсе, онда сыртқы жадыға сұрау салғанда дисктік кеңістіктің блогының нөмірін генерациялау қажет. Басты кедергілер коллизияларға қатысты.

**Кеңейетін хэштеу .**Кеңейетін хэштеу (Extendible Hashing) негізінде негізгі жадыдағы сандық іздеу ағаштарын қолдануда жатыр. Негізгі жадыда сандық іздеудің бинарлық ағаштары негізінде жасалған анықтама бар, олардың кілттері хэш-функциялар мәні. Осы жағдайда, сандық іздеу ағашындағы іздеу «сәтті» жүргізіледі, басқаша айтқанда сыртқы жадының кейбір блогына алып келеді.

**Сызықтық хэштеу.** Сызықтық хэштеу идеясы (Витольд Литвин) негізгі жадыда анықтаманы орналастырмауда болып отыр. Әдістің бастысы сыртқы жады блогына жіберу үшін әрқашан хэш-функцияның кіші мәні биттерін қолдану. Егер бөлшектеу қажеттілігі туындаса, онда жіберу дұрыс қалатындай етіліп, блок бойынша қайта үлестіріледі.

Деректер қорындағы индекстерді ұйымдастыру үшін хэштеуді қолдану

Деректер қорында хэштеу әдісі бүгінгі күні сирек қолданылады. Хэштеу әдісін бастапқы кезден қолданып келе жатқан жүйелердің ерекше түрі ретінде Ingres-ті айта аламыз. Оның себебі анық. Хэштеу пайда болған күннен ерекше кілт бойынша іздеуге бағдарланған еді. Ең кең таралған әдістер толыққанды қамтамасыз ете алмайды. Дегенмен, бірте-бірте хэштеу технологиясы және В-ағаштар технологиясымен бірігіп, әлемдегі ең негізгі деректер қорына айналуы мүмкін.

Деректер қорындағы іздеудің жанама әдістері:Байланыс индекстері және биттік шкаланы қолдану негізіндегі индекстер.

**ҚОРЫТЫНДЫ:** Алгоритмді жобалағанда және дербес жағдайда деректердің құрылымында ағаш кең қолданыста болады. Дейкстраның алгоритмі графқа арнайы қарсылы емес салмақты шыңдары бар есептерді дұрыс есептейді. Iздестiру ағашында ең төменгi кiлтпен элемент табуға болады.Қызыл-қара ағаштар- іздеудің кеңейтілген екі жақты ағашы,биіктігі қара және қызыл болып бөлінеді.Б-ағаштарымен қызмет атқаратын алгаритмдердің оперативті есте сақтау қабілеті барлық ақпараттың аз мөлшерін ғана (бөліктердің белгілі санын) қамтиды. Сыртқы жадыдағы деректерді іздеу үшін негізгі «ағаш тәріздес» аппараты В-ағаштар болады.B+-ағаштар деректер қорларындағы индекстерді ұйымдастыру үшін қарқынды түрде қолданылады.R-ағаштар және олардың кеңістік деректер қорындағы индекстерді ұйымдастыру үшін қолданады.