14-15 дәріс. ГРАФТАРҒА КІРІСПЕ.

ГРАФТАРДЫҢ ІРГЕЛІ АЛГОРИТМІ, ЕҢ ҚЫСҚА ЖОЛДЫ ЕСЕПТЕУ

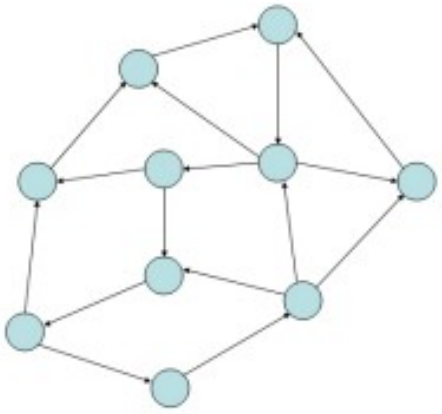
Графтарға кіріспе. Графтарды нақты өмірде қолданудың көптеген мысалдары бар. Осындай мсалдардың бірі – интернет. Интернеттегі желілердің өзара байланысы ***граф*** деп аталады. Компьютерлер графтардың түйіндері болып келеді, бірақ кез келген компьютер өзге машинаға тізбектей жалғанбағандықтан, машиналар арасындағы байланыс осы графтардың бір бөлігі болып табылады.

Тағы да бір мысал келтірер болсақ, теміржол станцияларының арасындағы хабарламалар желісінің трафигі немесе ұшақтардың қалалар арасында ұшуы. Мысалдағы түйіндер – теміржол станциялары және қалаларға байланыстар – рельстер және сапарлар.

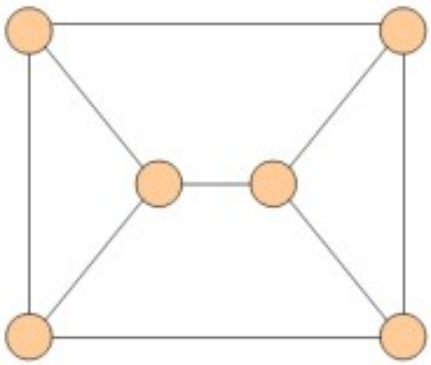
1-сурет

Анықталған графтар

Бағытталған граф V төбе немесе түйіндер жиынынан және E қабырғалардан немесе доғалардан тұрады. Әрбір қабырғаның бастаушы және сынақ түйіні бар, сондықтан қабырға бойынша бастаушы түйіннен сыни түйінге дейін жүруге болады.



2-сурет



3-сурет

Графтардың ерекшелігі, оларды көрнекі түрде бейнелеуге болады және графтар есептің маңызды қасиеттерін сипаттауда тиімді болып келеді.

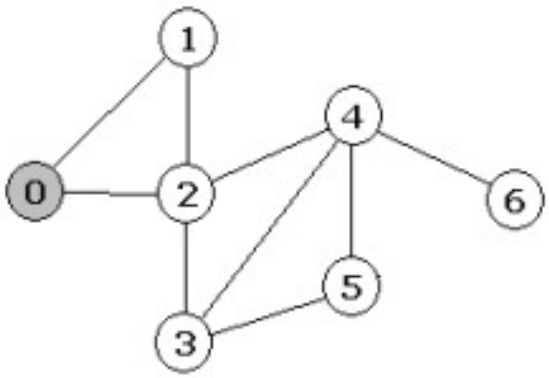
s және t екі түйінді қарастырайық, s-тен t-ға баратын жолдың бар-жоғын анықтаймыз. Бұл мәселені ***жұптың бар болу мәселесі*** деп атайды. Мұнымен графтың кез келген екі төбесі арасында жолдың бар-жоғын анықтайтын барлық жұптың бар болу мәселесі тығыз байланысты болып келеді. Компьютерлік желіде бұл бір компьютерден екінші компьютерге хабарлама тарату мүмкіндігін білдіреді.

Практикада көп жағдайда кез келген қабырға әртүрлі бағыттарда жылжуы мүмкін болатын бағытталмаған графтармен жұмыс жасауға тура келеді. Бағытталмаған шектерді бағытталған қабырғалар жұбы ретінде қарастыруға болады, сондықтан біз бағытталған қабырғаларды терең қарастырамыз.

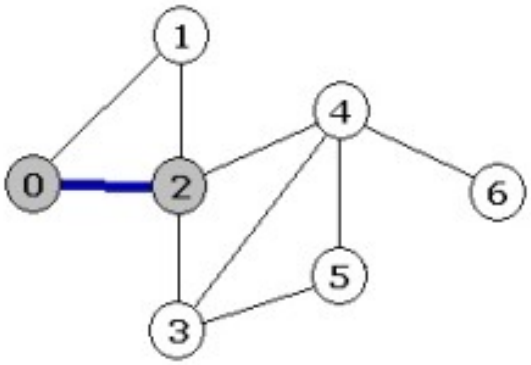
Ені бойынша іздеу алгоритмі

Ені бойынша іздеу алгоритмі граф түйіндерін бастапқы түйіннен қашықтығының өсу ретімен іздейді. Ені бойынша іздеу алгоритмі түйінге қолжетімділік сұрақтарына жауап ретінде қолданылады. Ені бойынша іздеу алгоритмін бастапқы түйіндерді анықтау үшін қолданамыз. Графтар өзге түйіндермен біріктірілмейтін түйіндерден тұруы мүмкін болғандықтан, бұл өте маңызды.

4-тен 8-ге дейінгі суреттер тізбегі ені бойынша іздеу алгоритмін қарапайым графтарда көрсеткен.



4-сурет. Ені бойынша іздеу алгоритмінің басталуы

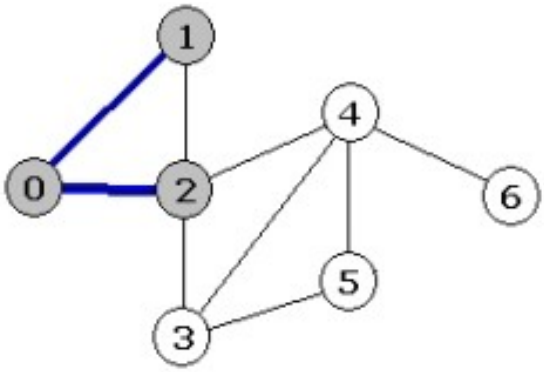


5-сурет. 2-түйін ашық

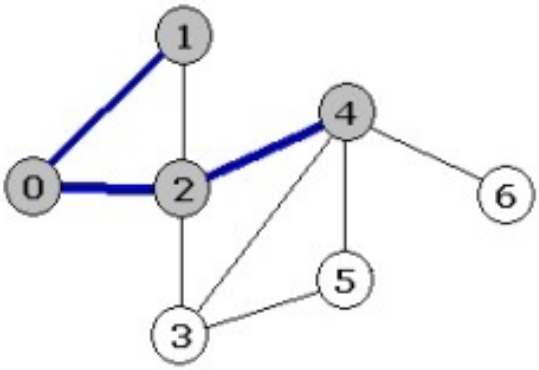
Ені бойынша іздеу алгоритмінің бастапқы түйіні 4-суретте боялған. Алгоритм осы түйіннен тараған әрбір қабырғаны қарастырудан басталады. Алгоритм осы қабырғаларды қарап болған соң ғана жаңа түйін табылады. Бұл «табылған» түйіндер кезекте тұрады. Ені бойынша іздеу алгоритмі өз кезегінде жаңа түйінді анықтау үшін әрбір табылған жаңа түйінді тексереді.

5-суреттен көріп тұрғанымыздай, алгоритм 0-түйінмен байланысқан қабырғаларды зерттеу арқылы 2-түйінді анықтады. Түйін анықталғаннан кейін ол табылған, бірақ әлі зерттелмеген түйін ретінде кезекке қойылады. Содан соң қалған қабырғаларды зерттеу арқылы алгоритм аяқталады. Бұл шеттегі 1-түйінге алып келеді.

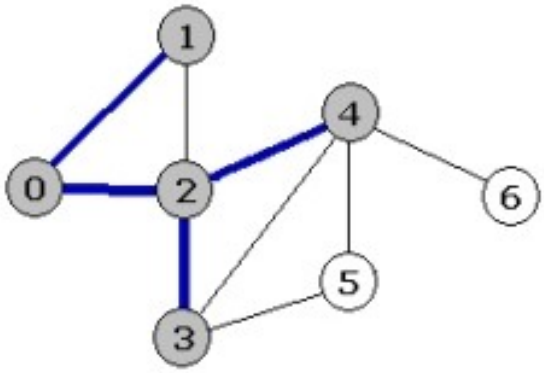
Егер графтағы қабырғалардың барлығы қарастырылса, онда түйін толығымен зерттелген болып саналады. Алгоритм зерттеу үшін басқа түйінді таңдайды. Бұл таңдау кездейсоқ ретпен орындалмайды. Таңдау кезектегі бірінші табылған бірақ әлі зерттелмеген түйін ретінде анықталған. Дәл қазір бұл 2-түйін болып табылады. 7 және 8-суреттерде алгоритм осы түйінді зерттеп жатқанын көреміз



6-сурет. 1-түйін ашық



7-сурет. 4-түйінді анықтау

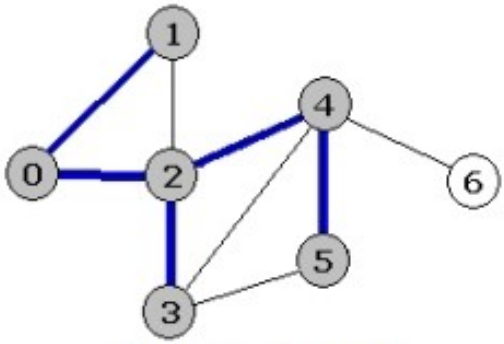


8-сурет. 3-түйінді анықтау

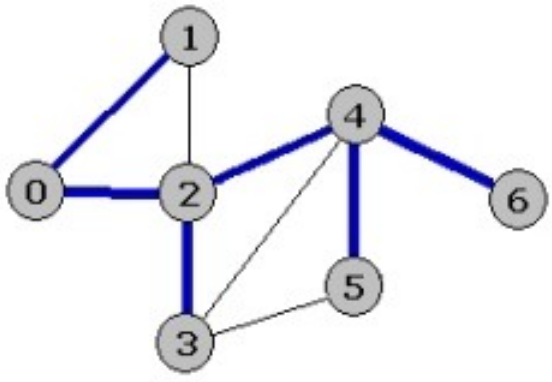
2-түйінмен байланысатын барлық қабырғаларды зерттеп болғаннан соң 1-түйінмен байланысатын қабырғаларды қарастырамыз. Бірақ бұл бұрын зерртелген түйіндерге алып келгендіктен алгоритм қабырғаны 4-түйінге қосылған деп есептеп басқа түйінге ауысады.

4-түйінге 4 қабырға қосылған. Олардың ішінде екі қабырға бұрын зерттелген түйіндерге алып келеді. Алгоритм 5 және 6-түйіндерге алып келетін қалған екі қабырғаны қарастырады.

Бұл екі түйін «ашық» болып белгіленіп, кезекке қойылады. Содан соң алгоритм 3-түйінге қосылған қабырғаларды зерттеуге кіріседі.



9-сурет. 5-түйін табылды



10-сурет. 6-түйін табылды

Сонымен, алгоритм графтың барлық түйіндерін тапты. Біз мұны көре аламыз, себебі 10-суретте барлық түйінде штрихтелген. Бірақ алгоритм мұны білмейді. Ол әлі күнге дейін 3, 5-түйіндермен байланысқан қабырғаларды қарастырып жүр. Егер бұлар бастапқы қарастырылған түйінге келсе, алгоритм бітті деп есептеледі.

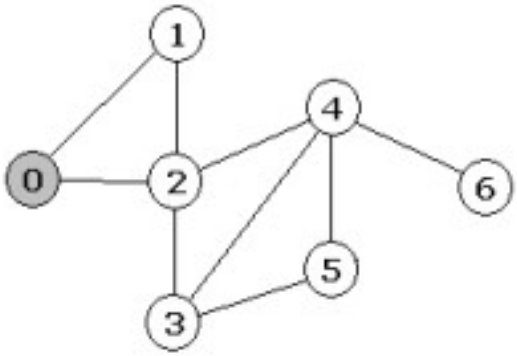
```
1:  void bfs(vector< list<int> >& adj_lists, int start_node) {
2:
3:    queue<int> not_yet_explored;
4:     set<int> discovered;
5:
6:
7:    // Mark the start node as being discovered,
8:    // and place it in the queue of nodes to explore
9:    not_yet_explored.push(start_node);
10:   discovered.insert(start_node);
11:
12:   while (! not_yet_explored.empty()) {
13:
14:       // Get a node to explore.
15:       int node_to_explore = not_yet_explored.front();
16:       not_yet_explored.pop();
17:
18:      // Examine all the edges of the node
19:      list<int>::iterator edges = adj_lists[node_to_explore].
    begin();
20:      for  (  ;  edges  !=  adj_lists[node_to_explore].end();
    edges++) {
21:
22:          // See if the edge leads to a node that we
23:          // have not yet discovered
24:          if (discovered.count(*edges) == 0) {
25:
26:              // We have discovered a new node!
27:              // Add this node to the queue of nodes
28:              // to explore.
29:              discovered.insert(*edges);
30:              not_yet_explored.push(*edges);
31:
32:              cout << "Found " << *edges <<
33:              " from " << node_to_explore << endl;
34:
35:            }
36:        }
37:    }
38:
    }
```

11-листинг. Ені бойынша іздеу алгоритмінің орындалуы

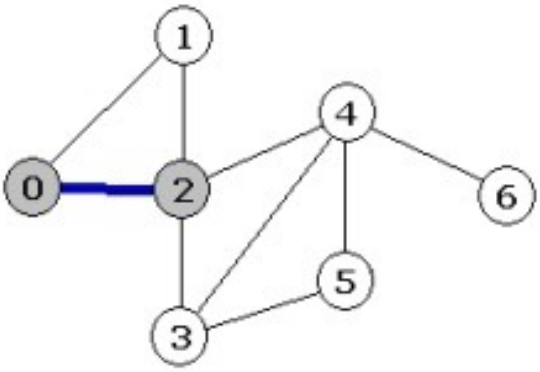
Ені бойынша іздеу алгоритмі 11-листингте келтірілген. Бұл программа 1-7 суреттеріндегі графиктерді талдайды. Бұл графтағы түйіндер сан түріндегі сілтеме болғандықтан, біз ***int*** типті вектор тізімін қолдана аламыз.

Тереңдігі бойынша іздеу алгоритмі

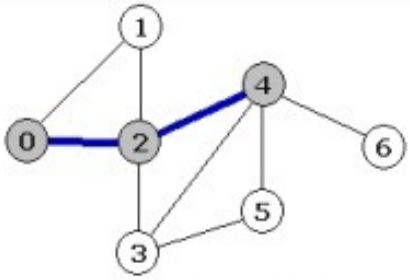
Тереңдігі бойынша іздеу алгоритмі граф түйіндерін түйін басынан бастап қашықтықтың өсуінің кері ретімен талдайды. Тереңдігі бойынша іздеу алгоритмі граф тереңдігінен түйіндерді зерттейді, содан соң ағымдағы түйінге жақын тұрған түйіннен бастап қайта зерттей бастайды. 12-ден 18-ге дейінгі суреттер алдыңғы мысалда келтірілген графты тереңінен іздеу алгоритмі бойынша зерттеуді бейнелейді.



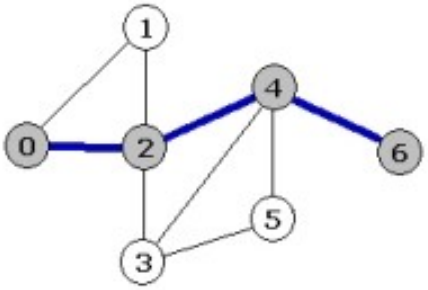
12-сурет. Тереңдігі бойынша іздеу алгоритмнің бірінші қадамы



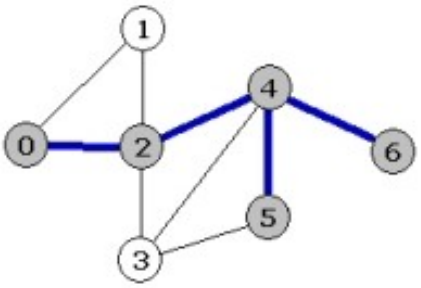
13-сурет. Тереңдігі бойынша іздеу алгоритмнің екінші қадамы



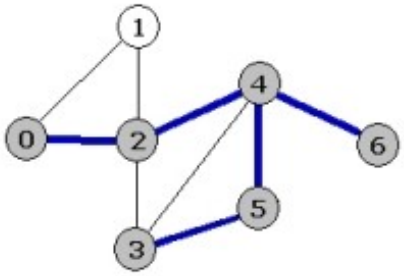
14-сурет. Тереңдігі бойынша іздеу алгоритмнің үшінші қадамы



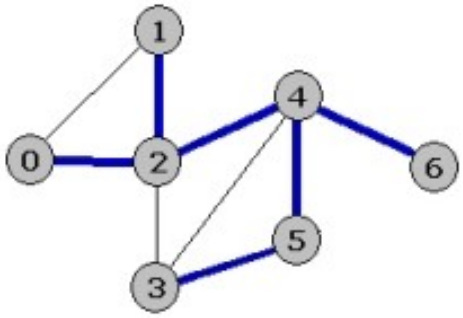
15-сурет. Тереңдігі бойынша іздеу алгоритмнің төртінші кезеңі



16-сурет. Тереңдігі бойынша іздеу алгоритмнің бесінші кезеңі



17-сурет. Тереңдігі бойынша іздеу алгоритмнің алтыншы кезеңі



15-сурет. Тереңдігі бойынша іздеу алгоритмнің қорытынды кезеңі

Жоғарыда айтып кеткеніміздей, тереңдігі бойынша іздеу алгоритмінде түйіндер табылғанына қарай зерттеледі. Тереңдігі бойынша іздеу алгоритмі жаңадан табылған түйіндерді зерттейді. Қолданылмаған түйіндерді сақтау үшін стек қолданылады. Тереңдігі бойынша іздеу алгоритміне түйіндерімен қайта оралу соңғы төбеден басталады. Мұндай стратегияны «соңғы енгізілген бірінші зерттеледі» деп атайды. Ол ***dfs*** алгоритмінің итеративті нұсқасында қолданылатын айқын стекке және рекурсивті нұсқалары арқылы генерацияланатын төбелердің айқындалмаған стегіне сәйкес келеді. Бұл зерттелмеген түйіндерді сақтау үшін контейнерді қолдану қажеттілігін жояды.

```
1:  void  dfs_helper(vector<  list<int>  >&  adj_lists,  set<int>&
2:  discovered, int node) {
3:
4:      // Examine all the edges of the node
5:      list<int>::iterator edges = adj_lists[node].begin();
6:      for ( ; edges != adj_lists[node].end(); edges++) {
7:
8:          // See if the edge leads to a node that we
9:          // have not yet discovered
10:         if (discovered.count(*edges) == 0) {
11:
12:             // We have discovered a new node!
13:             // Add this node to the queue of nodes
14:             // to explore.
15:             discovered.insert(*edges);
16:             cout << "Found " << *edges <<
17:             " from " << node << endl;
18:             dfs_helper(adj_lists, discovered, *edges);
19:         }
20:     }
21: }
22:
23: void dfs(vector< list<int> >& adj_lists, int start_node) {
24:
25:     // Mark the start node as being discovered
26:     set<int> discovered;
27:     discovered.insert(start_node);
28:     dfs_helper(adj_lists, discovered, start_node);
29: }
```

2-листинг. Тереңдігі бойынша іздеу алгоритмінің орындалуы

Ең қысқа жолды есептеу

Ең қысқа жолды есептеу графтағы түйіндер арасындағы ең қысқа жолды анықтайтын алгоритмдер отбасымен жабдықтайды. Графтардың түрлеріне сәйкес әртүрлі тәсілдер талап етілгендіктен ең қысқа жолды табудың бірнеше алгоритмдері бар.

Өлшеусіз графтарда түйіндер арасындағы ең қысқа жолды табуды есептеу үшін ені бойынша іздеу алгоритмі қолданылуы мүмкін.

Өлшемді графтар үшін әртүрлі және күрделі алгоритмдер қолданылады. Егер өлшемді графтың салмағы оң болса, онда біз ***Дейкстра алгоритмін*** қолдана аламыз. Теріс салмақты қабырғалары бар графтардың қысқа жолды анықтау мәселесін ***БеллманФорд алгоритмін*** қолданып шығарамыз.