

[All Contests](#) > [OOAIA-2023-Lab-3](#) > [Graph representations](#)

Graph representations

🔒 locked

Problem

Submissions

Leaderboard

Discussions

Submitted 4 months ago • Score: 100.00

Status: **Accepted**

✓	Test Case #0	✓	Test Case #1	✓	Test Case #2
✓	Test Case #3	✓	Test Case #4	✓	Test Case #5
✓	Test Case #6	✓	Test Case #7	✓	Test Case #8
✓	Test Case #9	✓	Test Case #10	✓	Test Case #11
✓	Test Case #12	✓	Test Case #13	✓	Test Case #14

Submitted Code

Language: C++20

[🔗 Open in editor](#)

```
1 #include <iostream>
2 #include <stack>
3 using namespace std;
4 class Vertex
5 {
6     private:
7         int n;
8         Vertex* next;
9         Vertex* pp;
10        int ID;
11        int OD;
12    public:
13        Vertex(int nv)
14        {
15            n = nv;
16            ID = 0;
17            OD = 0;
18            next = NULL;
19            pp = this;
20        }
21        Vertex()
22        {
23        }
24        Vertex & operator = (const Vertex & rhs)
25        {
26            n = rhs.n;
27            ID = rhs.ID;
28            OD = rhs.OD;
29            next = rhs.next;
```

```
31         pp = rhs.pp;
32         return *this;
33     }
34     //ID = 0;
35     //OD = 0;
36
37     int val()
38     {
39         return n;
40     }
41     void modp(Vertex* t)
42     {
43         pp = t;
44     }
45     Vertex* nextp()
46     {
47         return next;
48     }
49     void nextm(Vertex* ll)
50     {
51         next = ll;
52     }
53     Vertex* modq()
54     {
55         return pp;
56     }
57     int IDM()
58     {
59         return ID;
60     }
61     void INID()
62     {
63         ID++;
64     }
65     void INOD()
66     {
67         OD++;
68     }
69     int ODM()
70     {
71         return OD;
72     }
73 };
74 class Edge
75 {
76     private:
77         Vertex *n; Vertex* m;
78     public:
79         Edge(Vertex nv, Vertex ne)
80         {
81             m = new Vertex;
82             n = new Vertex;
83             *m = ne; *n=nv;
84         }
85         Edge() {
86             //m=0; n=0;
87         }
88         int st()
89         {
90             return n->val();
91         }
92         int ed()
93         {
94             return m->val();
95         }
96         Vertex stv()
```

```
97     {
98         return (*n);
99     }
100     Vertex* pedv()
101     {
102         return m;
103     }
104     Vertex edv()
105     {
106         return (*m);
107     }
108 };
109 class Graph
110 {
111     protected:
112         int numVertices;
113         int numEdges;
114     public:
115         /* operator GraphUsingMatrix(int nv, int ne) ()
116         {
117             GraphUsingMatrix G = GraphUsingMatrix(nv, ne);
118             return &G;
119         }
120         operator GraphUsingList(int nv, int ne) ()
121         {
122             GraphUsingList G = GraphUsingList(nv, ne);
123             return &G;
124         }*/
125         Graph(int nv, int ne) : numVertices(nv), numEdges(ne) {}
126         Graph() : numVertices(0), numEdges(0){};
127         virtual void addEdge(Edge e)=0;
128         virtual int outDegree(Vertex v) = 0;
129         virtual int inDegree(Vertex v)=0;
130         virtual bool reach(Vertex v1, Vertex v2)=0;
131         virtual bool detectCycle()=0;
132
133         /* Define virtual functions here*/
134 };
135
136
137
138 class GraphUsingMatrix:public Graph
139 {
140     private:
141
142         bool **arr;
143         int *brr;
144         int *crr;
145         int v;
146         int e;
147     public:
148         GraphUsingMatrix(int nv, int ev)
149         {
150             v = nv;
151             e = ev;
152             arr = new bool*[nv];
153             for (int ii=0; ii<nv; ii++)
154             {
155                 arr[ii] = new bool[nv];
156             }
157             for (int ii=0; ii<nv; ii++)
158             {
159                 for (int jj=0; jj<nv; jj++)
160                 {
161                     arr[ii][jj] = false;
162                 }
163             }
164         }
165 }
```

```
163     }
164     brr = new int[nv];
165     for (int ii=0; ii<nv; ii++)
166     {
167         brr[ii] = 0;
168     }
169     crr = new int[nv];
170     for (int ii=0; ii<nv; ii++)
171     {
172         crr[ii] = 0;
173     }
174 }
175
176 void addEdge(Edge e)
177 {
178     arr[e.st()][e.ed()] = true;
179     crr[e.st()] ++;
180     brr[e.ed()] ++;
181 }
182 int inDegree(Vertex v)
183 {
184     return brr[v.val()];
185 }
186 int outDegree(Vertex v)
187 {
188     return crr[v.val()];
189 }
190 void reach1(Vertex v1, Vertex v2, bool ar[], int* k)
191 {
192     if ((*k) == 1)
193     {
194         return;
195     }
196     ar[v1.val()] = true;
197     if (arr[v1.val()][v2.val()]==true)
198     {
199         *k = 1;
200         return;
201     }
202     else
203     {
204         for (int ii=0; ii<v&&(*k)==0; ii++)
205         {
206             if (arr[v1.val()][ii]==true)
207             {
208                 if (ar[ii]==false)
209                 {
210                     Vertex a(ii);
211                     reach1(a, v2, ar, k);
212                 }
213             }
214         }
215     }
216     return;
217 }
218 bool reach(Vertex v1, Vertex v2)
219 {
220     if (v1.val()==v2.val())
221     {
222         return true;
223     }
224     bool bb[v] = {false};
225     int k = 0;
226     reach1(v1, v2, bb, &k);
227     if (k==1){return true;}
228     else {return false;}
```

```
229     }
230     bool detectCycle()
231     {
232         stack<int> st;
233         stack<int> lt;
234         bool hl[v] = {false};
235         bool hh[v] = {false};
236         int h;
237         int help[v] = {0};
238         bool his;
239         for (int ii=0; ii<v; ii++)
240         {
241             if (hh[ii]==false)
242             {
243                 st.push(ii);
244                 help[ii]=1;
245                 while(!st.empty())
246                 {
247                     h = st.top();
248                     help[h]=1;
249                     lt.push(h);
250                     hh[h] = true;
251                     hl[h] = true;
252                     //st.pop();
253                     his = true;
254                     for (int ii=0; ii<v; ii++)
255                     {
256                         if (arr[h][ii]==true)
257                         {
258                             if (help[ii]==1)
259                             {
260                                 return true;
261                             }
262                             else if (hl[ii]==true)
263                             {
264                                 continue;
265                             }
266                             his=false;
267                             st.push(ii);
268                         }
269                     }
270                 }
271                 if (his==true)
272                 {
273                     help[st.top()]=2;
274                     st.pop();
275                 }
276             }
277             while(!lt.empty())
278             {
279                 hl[lt.top()]=false;
280                 lt.pop();
281             }
282         }
283     }
284     return false;
285 }
286
287 };
288 class GraphUsingList:public Graph
289 {
290     private:
291
292     Vertex* arr;
293     int v;
294     int e;
```

```

295     public:
296     GraphUsingList(int nv, int ev)
297     {
298         v = nv;
299         e = ev;
300         arr = new Vertex[nv];
301         for (int ii=0; ii<nv; ii++)
302         {
303             //Vertex l(ii);
304             arr[ii] = Vertex(ii);
305             arr[ii].nextm(NULL);
306             arr[ii].modp(arr + ii);
307         }
308     }
309     int Ver()
310     {
311         return v;
312     }
313     void addEdge(Edge e)
314     {
315         Vertex* lop = e.pedv();
316         //cout<<(arr[3].modq())->val()<<endl;
317         (*(arr[e.st()].modq())).nextm(lop);
318         //cout<<arr[e.st()].val()<<" "<<(arr[e.st()].nextp())->val()<<" "<<(arr[e.st()].modq())-
>val()<<endl;
319         (arr[e.st()]).modp(lop);
320         //cout<<arr[e.st()].val()<<" "<<(arr[e.st()].nextp())->val()<<" "<<(arr[e.st()].modq())-
>val()<<endl;
321         Vertex* kop = NULL;
322         (e.edv()).nextm(kop);
323         arr[e.ed()].INID();
324         arr[e.st()].INOD();
325         //cout<<arr[e.st()].val()<<" "<<(arr[e.st()].nextp())->val()<<" "<<(arr[e.st()].modq())-
>val()<<endl;
326         //cout<<(arr[3].modq())->val()<<endl;
327         //cout<<"help"<<endl;
328     }
329     int inDegree(Vertex(v))
330     {
331         return arr[v.val()].IDM();
332     }
333     int outDegree(Vertex(v))
334     {
335         return arr[v.val()].ODM();
336     }
337     void reach1(Vertex v1, Vertex v2, bool ar[], int* k)
338     {
339         if ((*k)==1)
340         {
341             return;
342         }
343         Vertex* p = &(arr[v1.val()]); //cout<<(*p).val();
344         p = p->nextp(); //cout<<(*p).val();
345         ar[v1.val()] = true;
346         int ii=0;
347         while (ii<arr[v1.val()].ODM()&&(*k)==0)
348         {
349             if ((*p).val()==v2.val())
350             {
351                 (*k)=1; //cout<<"a"<<v2.val()<<(*p).val();
352                 return;
353             }
354             else if (ar[(*p).val()]==true)
355             {
356                 p = p->nextp(); //cout<<"b";

```

```
358         ii++;
359     }
360     else{
361         reach1(arr[(*p).val()], v2, ar, k); //cout<<"c";
362         p = p->nextp();
363         ii++;
364     }
365 }
366 return;
367 }
368 bool reach(Vertex v1, Vertex v2)
369 {
370     int h=0;
371     bool dr[v] = {false};
372     reach1(v1, v2, dr, &h);
373     if (h == 1)
374     {
375         return true;
376     }
377     else{
378         return false;
379     }
380 }
381 bool detectCycle()
382 {
383     stack<int> st;
384     stack<int> lt;
385     bool hl[v] = {false};
386     bool hh[v] = {false};
387     int h;
388     Vertex* hv;
389     int help[v] = {0};
390     bool his;
391     for (int ii=0; ii<v; ii++)
392     {
393         if (hh[ii]==false)
394         {
395             st.push(ii);
396             help[ii]=1;
397             while(!st.empty())
398             {
399                 h = st.top();
400                 help[h] = 1;
401                 lt.push(h);
402                 hh[h] = true;
403                 hl[h] = true;
404                 his = true;
405                 //st.pop();
406                 hv = arr[h].nextp();
407                 while(hv!=NULL)
408                 {
409                     if (help[(*hv).val()]==1)
410                     {
411                         return true;
412                     }
413                     if (hl[(*hv).val()]==true)
414                     {
415                         hv = hv->nextp();
416                         continue;
417                     }
418                     his = false;
419                     st.push((*hv).val());
420                     hv = hv->nextp();
421                 }
422                 if (his == true)
423                     return true;
424             }
425         }
426     }
```

```
424         {
425             help[st.top()]=2;
426             st.pop();
427         }
428
429     }
430     while(!lt.empty())
431     {
432         hl[lt.top()]=false;
433         lt.pop();
434     }
435 }
436 }
437 return false;
438 }
439
440
441 };
442 /*Define the derived classes here*/
443
444
445 /* DO NOT CHANGE THE CODE BELOW */
446 int main()
447 {
448     int N;
449     cin >> N;
450     Graph * g;
451     int command;
452     const int SPARSITYRATIO = 5;
453     for (int i = 0; i < N; i++)
454     {
455         cin >> command;
456         switch (command)
457         {
458             case 1: /* initialize number of vertices and edges */
459             {
460                 int nv,ne;
461                 cin >> nv >> ne;
462                 if (ne/nv > SPARSITYRATIO)
463                     g = new GraphUsingMatrix(nv,ne);
464                 else
465                     g = new GraphUsingList(nv,ne);
466                 break;
467             }
468
469             case 2: /* Add edge */
470             {
471                 int v,w;
472                 cin >> v >> w;
473                 g->addEdge(Edge(Vertex(v), Vertex(w)));
474                 break;
475             }
476
477             case 3: /* Reachability query */
478             {
479                 int v,w;
480                 cin >> v >> w;
481                 cout << g->reach(Vertex(v), Vertex(w)) << endl;
482                 break;
483             }
484
485             case 4: /* Detect Cycle */
486             {
487                 cout << g->detectCycle() << endl;
488                 break;
489             }

```



```
490
491     case 5: /* In-degree */
492     {
493         int v;
494         cin >> v;
495         cout << g->inDegree(Vertex(v)) << endl;
496         break;
497     }
498
499     case 6: /* Out-degree */
500     {
501         int v;
502         cin >> v;
503         cout << g->outDegree(Vertex(v)) << endl;
504         break;
505     }
506
507     default:
508         break;
509 }
510 }
511 }
```