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

```
#include <iostream>
#include <stack>
using namespace std;
class Vertex
{
    private:
      int n;
      Vertex* next;
      Vertex* pp;
      int ID;
      int OD;
    public:
      Vertex(int nv)
      {
          n = nv;
          ID = 0;
          OD = 0;
          next = NULL;
          pp = this;
      }
    Vertex()
    {

    }
      Vertex & operator = (const Vertex & rhs)
      {
          n = rhs.n;
          ID = rhs.ID;
          OD = rhs.OD;
          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()
```

```
      {
        return (*n);
      }
      Vertex* pedv()
      {
          return m;
      }
      Vertex edv()
      {
        return (*m);
      }
};
class Graph
{
  protected:
    int numVertices;
    int numEdges;

  public:
           /* operator GraphUsingMatrix(int nv, int ne) ()
        {
           GraphUsingMatrix G = GraphUsingMatrix(nv, ne);
           return &G;
        }
            operator GraphUsingList(int nv, int ne) ()
        {
           GraphUsingList G = GraphUsingList(nv, ne);
           return &G;
        }*/
    Graph(int nv, int ne) : numVertices(nv), numEdges(ne) {}
    Graph() : numVertices(0), numEdges(0){};
    virtual void addEdge(Edge e)=0;
    virtual int outDegree(Vertex v) = 0;
    virtual int inDegree(Vertex v)=0;
    virtual bool reach(Vertex v1, Vertex v2)=0;
    virtual bool detectCycle()=0;

  /* Define virtual functions here*/
};


class GraphUsingMatrix:public Graph
{
    private:

        bool **arr;
        int *brr;
        int *crr;
        int v;
        int e;
    public:
        GraphUsingMatrix(int nv, int ev)
        {
           v = nv;
           e = ev;
           arr = new bool*[nv];
           for (int ii=0; ii<nv; ii++)
           {
            arr[ii] = new bool[nv];
           }
           for (int ii=0; ii<nv; ii++)
           {
              for (int jj=0; jj<nv; jj++)
              {
                 arr[ii][jj] = false;
              }
```

```
            }
            brr = new int[nv];
            for (int ii=0; ii<nv; ii++)
            {
               brr[ii] = 0;
            }
            crr = new int[nv];
            for (int ii=0; ii<nv; ii++)
            {
               crr[ii] = 0;
            }
        }

        void addEdge(Edge e)
        {
           arr[e.st()][e.ed()] = true;
           crr[e.st()] ++;
           brr[e.ed()] ++;
        }
        int inDegree(Vertex(v))
        {
            return brr[v.val()];
        }
        int outDegree(Vertex(v))
        {
            return crr[v.val()];
        }
        void reach1(Vertex v1, Vertex v2, bool ar[], int* k)
        {
          if ((*k) == 1)
          {
            return;
          }
            ar[v1.val()] = true;
            if (arr[v1.val()][v2.val()]==true)
            {
               *k = 1;
               return;
            }
            else
            {
               for (int ii=0; ii<v&&(*k)==0; ii++)
               {
                  if (arr[v1.val()][ii]==true)
                  {
                    if (ar[ii]==false)
                    {
                     Vertex a(ii);
                     reach1(a, v2, ar, k);
                    }
                  }
               }
            }
            return;
        }
        bool reach(Vertex v1, Vertex v2)
        {
              if (v1.val()==v2.val())
              {
              return true;
              }
              bool bb[v] = {false};
              int k = 0;
              reach1(v1, v2, bb, &k);
              if (k==1){return true;}
              else {return false;}
```

```
        }
        bool detectCycle()
        {
          stack<int> st;
          stack<int> lt;
          bool hl[v] = {false};
          bool hh[v] = {false};
          int h;
          int help[v] = {0};
          bool his;
          for (int ii=0; ii<v; ii++)
          {
            if (hh[ii]==false)
            {
            st.push(ii);
            help[ii]=1;
          while(!st.empty())
          {
             h = st.top();
             help[h]=1;
             lt.push(h);
             hh[h] = true;
             hl[h] = true;
             //st.pop();
             his = true;
             for (int ii=0; ii<v; ii++)
             {
              if (arr[h][ii]==true)
              {
                if (help[ii]==1)
                {
                  return true;

                }
                else if (hl[ii]==true)
                {
                  continue;
                }
                his=false;
                st.push(ii);
              }
             }
             if (his==true)
             {
              help[st.top()]=2;
              st.pop();
             }
          }
          while(!lt.empty())
          {
                hl[lt.top()]=false;
                lt.pop();
                }
            }
          }
          return false;
        }

};
class GraphUsingList:public Graph
{
    private:

         Vertex* arr;
         int v;
         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
350                if ((*p).val()==v2.val())
351                {
352                   (*k)=1;//cout<<"a"<<v2.val()<<(*p).val();
353                   return;
354                }
355               else if (ar[(*p).val()]==true)
356                {
357                 p = p->nextp();//cout<<"b";
```

```
                ii++;
               }
               else{
                   reach1(arr[(*p).val()], v2, ar, k);//cout<<"c";
                   p = p->nextp();
                   ii++;
               }
           }
           return;
       }
       bool reach(Vertex v1, Vertex v2)
       {
           int h=0;
           bool dr[v] = {false};
           reach1(v1, v2, dr, &h);
             if (h == 1)
             {
                 return true;
             }
             else{
               return false;
             }
       }
       bool detectCycle()
       {
         stack<int> st;
         stack<int> lt;
         bool hl[v] = {false};
         bool hh[v] = {false};
         int h;
         Vertex* hv;
         int help[v] = {0};
         bool his;
         for (int ii=0; ii<v; ii++)
         {
           if (hh[ii]==false)
           {
         st.push(ii);
         help[ii]=1;
         while(!st.empty())
         {
             h = st.top();
             help[h] = 1;
             lt.push(h);
             hh[h] = true;
             hl[h] = true;
             his = true;
             //st.pop();
               hv = arr[h].nextp();
               while(hv!=NULL)
               {

                 if (help[(*hv).val()]==1)
                 {
                   return true;
                 }
                 if (hl[(*hv).val()]==true)
                 {
                   hv = hv->nextp();
                   continue;
                 }
                 his = false;
                 st.push((*hv).val());
                 hv = hv->nextp();
               }
               if (his == true)
```

```
                {
                  help[st.top()]=2;
                  st.pop();
                }

          }
          while(!lt.empty())
          {
                hl[lt.top()]=false;
                lt.pop();
                }
            }
          }
          return false;
        }


};
/*Define the derived classes here*/


/* DO NOT CHANGE THE CODE BELOW */
int main()
{
  int N;
  cin >> N;
  Graph * g;
  int command;
  const int SPARSITYRATIO = 5;
  for (int i = 0; i < N; i++)
  {
    cin >> command;
    switch (command)
    {
      case 1: /* initialize number of vertices and edges */
      {
        int nv,ne;
        cin >> nv >> ne;
        if (ne/nv > SPARSITYRATIO)
           g = new GraphUsingMatrix(nv,ne);
        else
           g = new GraphUsingList(nv,ne);
        break;
      }

      case 2: /* Add edge */
      {
       int v,w;
        cin >> v >> w;
        g->addEdge(Edge(Vertex(v), Vertex(w)));
        break;
      }

      case 3: /* Reachability query */
      {
        int v,w;
        cin >> v >> w;
        cout << g->reach(Vertex(v), Vertex(w)) << endl;
        break;
      }

      case 4: /* Detect Cycle */
      {
        cout << g->detectCycle() << endl;
        break;
      }
```

```
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 }
```