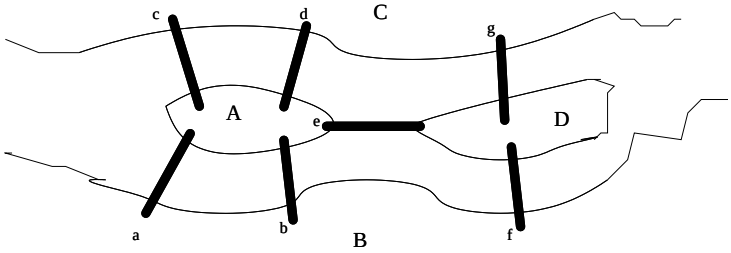
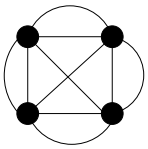
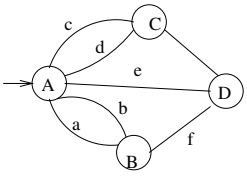
GRAPHS

- Introduced by Euler 1736, Koeingsberg bridge problem (EAST PRUSSIA)

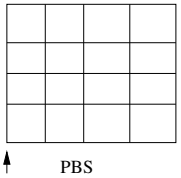


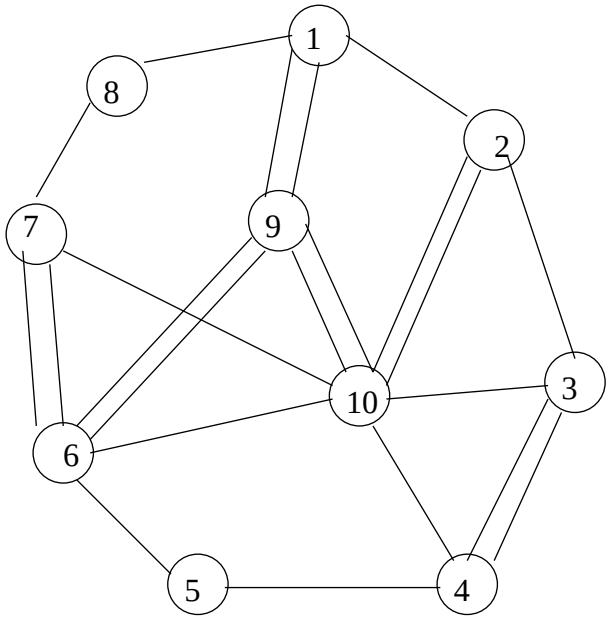
- Problem: Starting at some land area, is it possible to walk across all the bridges exactly once returning to the starting land area?

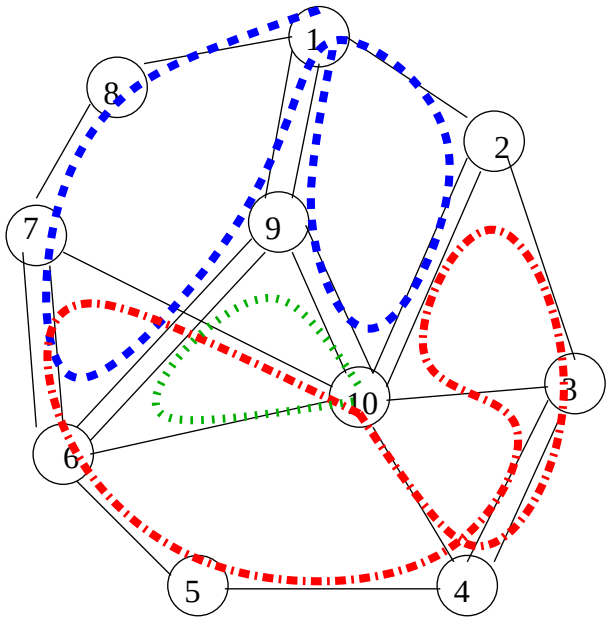




EULER CIRCUIT
MULTI–GRAPH




1
8
2
7
9
10
3
6
5
4


1
8
2
7
9
10
3
6
5
4

Theorem

Multi-Graph G has an euler circuit if and only if G is connected and every node is of even degree.

Algorithm by Stephen Barnard. (Discuss Quickly)

```
function EULER(v:vertex) Returns Path
  { path:=NULL;
    for all vertices w adjacent to v
        and edge(v,w) not yet used do
      { mark (v,w) used;
        path:={(v,w)} || EULER(w) || path;
        // concatenate represented by ||
      }
    return path;
   }
```

C++ code appears elsewhere.

Graph Represention

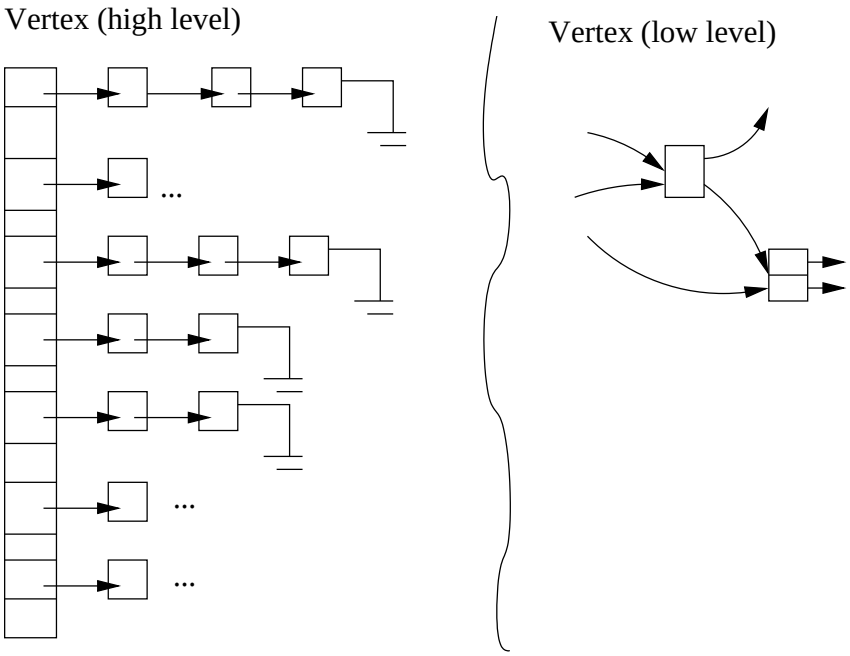
This implementation takes linear time with respect to the number of nodes and edges in the graph.

EDGE

V[0]	V[0]
V[1]	next[0]
next[0]	V[1]
next[0]	next[1]
Mark	Mark

Mgraph

n	n
e	e
Vertex[]	Vertex[]



```
class Mgraph;
class Edge;

class Edge{
friend class Mgraph;
public:
        Edge()  {mark=0;};
private:
        int v[2];
        Edge *next[2];
        int mark;
};
```

```
class Mgraph{
public:
        Mgraph();
        void input();
        void eulercycle();
        void eulercycle(int currentv);
        Edge *nextedge(int cv);
private:
        int n;
        int e;
        Edge *vertex[MAXVNUM];
};

Mgraph::Mgraph(){

  int i;

  for(i=0;i<MAXVNUM;i++)
    vertex[i]=0;
}
```

```
void Mgraph::input(){

  int i;
  Edge *te;

  cout<<"Please enter the number of vertices and
  cin>>n>>e;
  cout<<"Please enter the edges"<<endl;

  for(i=0;i<e;i++){
    te=new Edge();
    cin>>te->v[0]>>te->v[1];
    cout<<te->v[0]<< " " <<te->v[1]<<endl;
    te->next[0]=vertex[te->v[0]];
    vertex[te->v[0]]=te;
    te->next[1]=vertex[te->v[1]];
    vertex[te->v[1]]=te;
  }
}
```

```
Edge *Mgraph::nextedge(int cv){
  Edge *te,*tmp;
  te=vertex[cv];
  while(te && te->mark){
    tmp=te;
    if(cv==te->v[0])
      te=te->next[0];
    else
      te=te->next[1];
    delete tmp;
  }
  if(te){
    te->mark=1;
    if(cv==te->v[0])
      vertex[cv]=te->next[0];
    else
      vertex[cv]=te->next[1];
    return te;
  }
  vertex[cv]=0;
  return 0; }
```

```
void Mgraph::eulercycle(int currentv){

  Edge *ce;

  while(ce=nextedge(currentv)){
    if(currentv==ce->v[0])
      eulercycle(ce->v[1]);
    else
      eulercycle(ce->v[0]);
  }
  cout<<currentv<<"  ";
  return;
}

void Mgraph::eulercycle(){

  cout<<"The Euler cycle of this graph is"<<endl;
  eulercycle(1);
  cout<<endl;
}
```

```
main()
{
  Mgraph mg;

  mg.input();
  mg.eulercycle();
}
```

Sample Input

```
4 9
1 3
1 2
2 4
2 3
2 3
2 4
2 3
3 4
3 4
```

```
Sample Input
10 22
1 2
1 3
1 5
1 6
2 3
3 5
3 10
4 5
4 5
4 10
4 10
5 6
5 8
5 9
5 9
6 7
6 8
7 8
7 8
7 8
```

```
8 9
9 10
```