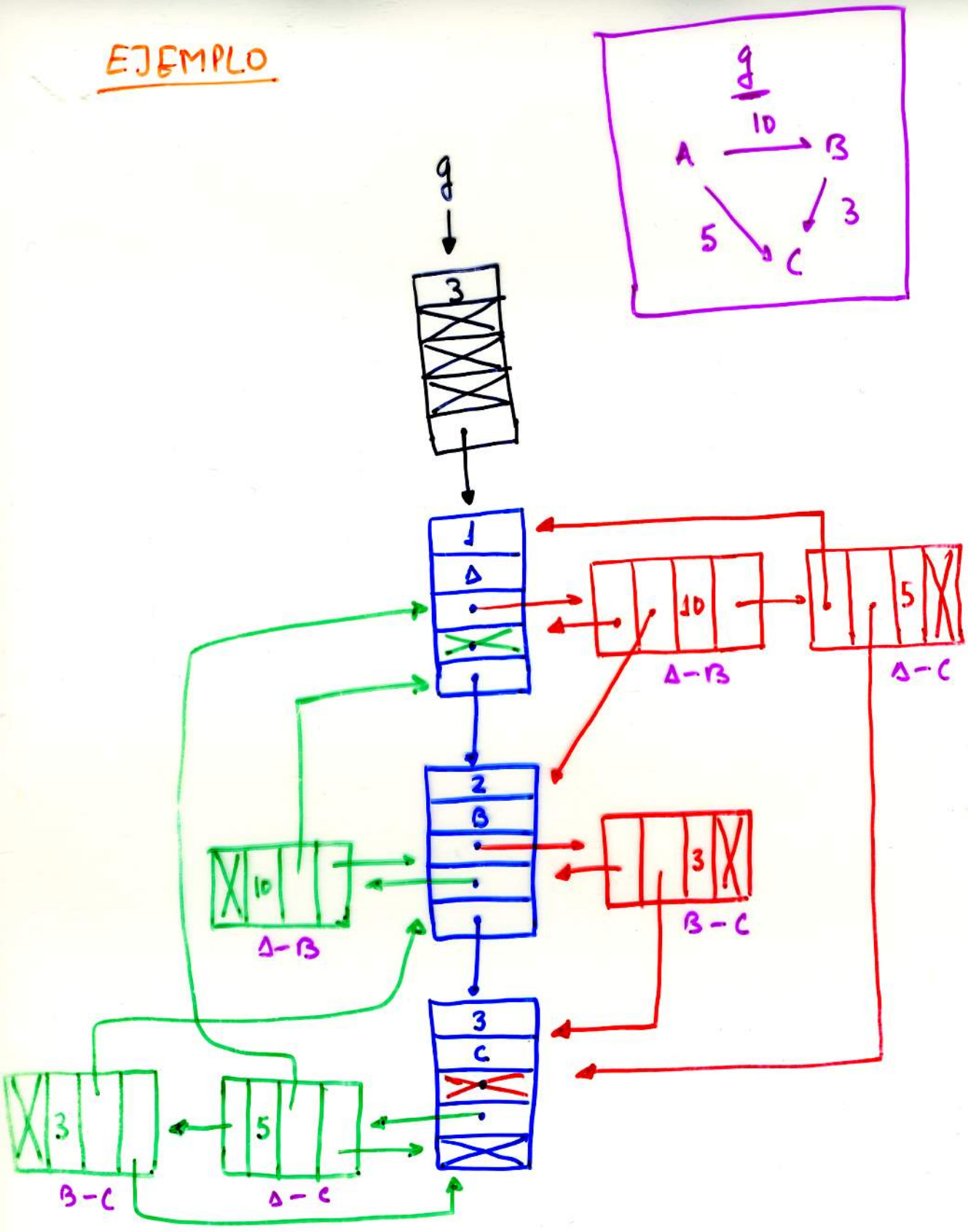
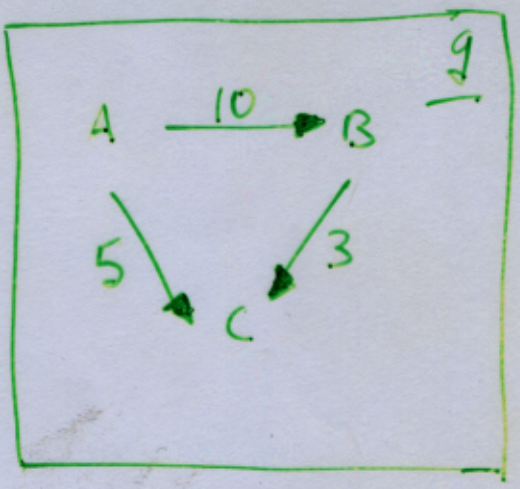
EJEMPLO




g
A
10
B
5
3
C

g
3
1
A
10
5
A-B
A-C
2
B
3
B-C
3
C

```
template<class Tn, class Ta>
  class arco {

      private:
        nodo<Tn,Ta> * origen;
        nodo<Tn,Ta> * destino;
        Ta valor;

      public:
        arco() { };
        arco(const arco<Tn,Ta> & a)
            { origen = a.origen;
              destino=a.destino;
              valor = a.valor;
            };
        arw(nodo<Tn,Ta> * o, nodo<Tn,Ta> * d, Ta v)
            { origen=o; destino=d; valor=v;
            };
        Ta etqArco() { return valor;};
        nodo<Tn,Ta> * nodoDestino()
                {return destino;
                };
        nodo<Tn,Ta> * nodoOrigen()
                {return origen;
                };
  };
```

```
template < class Tn, class Ta >
  class nodo {
      private:
        int ind; /* codigo interno de numeracion */
        Tn etiq;
        list < arco <Tn, Ta > > ady;
        list < arco <Tn, Ta > > inc;
      public:
        nodo() {};
        nodo (Tn et, int i)
          { etiq = et;
            ind = i;
          };
        nodo (const nodo <Tn, Ta> & n)
          { ind = n.ind;
            etiq = n.etiq;
            ady = n.ady;
            inc = n.inc;
          };
        Tn etiqueta()
          { return etiq;
          };
        int indice()
          { return ind;
          };
```

```
void nuevoAdy (nodo<Tn,Ta> * destino, Ta valor)
    {
    arco<Tn,Ta> a (this, destino, valor);
    ady.push_back(a);
    };
void nuevoInc (nodo<Tn,Ta> * origen, Ta valor)
    {
    arco<Tn,Ta> a (origen, this, valor);
    inc.push_back(a);
    };
typedef list<arco<Tn,Ta>>::iterator ady_iterator;
typedef list<arco<Tn,Ta>>::iterator inc_iterator;
ady_iterator abegin()
        { return ady.begin();
        };
ady_iterator aend()
        { return ady.end();
        };
inc_iterator ibegin() { return inc.begin();};
inc_iterator iend() { return inc.end();};
ady_iterator borrarAdy (ady_iterator i)
        { return ady.erase(i);
        };
inc_iterator borrarInc (inc_iterator i)
        { return inc.erase(i);};
};
```

```
template <class Tn, class Ta>
class grafo {
private:
  list <nodo <Tn, Ta> > *g;
  int comp;
public:
typedef list <nodo <Tn, Ta> >:: iterator nodeiterator;
typedef profundidadIterator <Tn, Ta> profIterator;
grafo ( );
grafo ( const grafo <Tn, Ta> & gr )
      { ncomp = gr.ncomp;
        g = gr.g;
      };
grafo <Tn, Ta> copiar Grafo( );
void insertarNodo (Tn etq);
void insertar Arco ( nodo <Tn, Ta> * origen,
                     nodo <Tn, Ta> * destino, Ta valor);
arco <Tn, Ta> * buscar Arco ( nodo <Tn, Ta> * o,
                              nodo <Tn, Ta> * d);
nodo <Tn, Ta> * buscarnodo ( Tn x);
nodo <Tn, Ta> * buscarnodo (int i);
```

```
nodeIterator nbegin();
nodeIterator nend();
profIterator pbegin();
profIterator pend();
int numeroNodos()
        {return ncomp;}
        };
bool vacio()
        {return ncomp == 0;
        };
};
```

```
template <class Tn, class Ta>
    grafo<Tn, Ta>::grafo()
    {
      g = new list<nodo<Tn, Ta>>();
      ncomp = 0;
    }
template <class Tn, class Ta>
  void grafo<Tn, Ta>::insertarNodo(Tn etq)
    {
      nodo<Tn, Ta> aux(etq, ncomp);
      ncomp++;
      g->push_back(aux);
    }
template <class Tn, class Ta>
 void grafo<Tn, Ta>::insertarArco(nodo<Tn, Ta> *origen,
                  nodo<Tn, Ta> *destino, Ta valor)
    {
      origen->nuevoAdy(destino, valor);
      destino->nuevoInc(origen, valor);
    }
```

```
template <class Tn, class Ta>
 grafo <Tn, Ta>::nodeIterator grafo <Tn, Ta>::nbegin()
    {
     return g->begin();
    }
template <class Tn, class Ta>
 grafo <Tn, Ta>:: nodeIterator grafo <Tn, Ta>::nend()
    {
     return g->end();
    }
template <class Tn, class Ta>
 nodo <Tn, Ta> * grafo <Tn, Ta>::buscarNodo(Tn x)
    {
      nodeIterator itr;
      for (itr = nbegin(); itr != nend(); )
         if (itr -> etiqueta() == x)
             return &(* itr);
         else itr++;
      return NULL;
    }
```

```
template <class Tn, class Ta>
nodo<Tn,Ta> * grafo<Tn,Ta>::buscarNodo(int i)
{
  int ic;
  nodeIterator itr;
  for(ic=0, itr=nbegin(); ic<i; ic++, itr++);
  return &(*itr);
}
template <class Tn, class Ta>
arco<Tn,Ta> * grafo<Tn,Ta>::buscarArco(
            nodo<Tn,Ta> * o, nodo<Tn,Ta> *d)
{
  nodo<Tn,Ta>::ady_iterator itr;
  bool enc=false;
  for(itr=o->abegin(); itr!=o->aend()
        && !enc;)
    if(itr->nodoDestino()==d)
      enc=true;
    else itr++;
  if(enc) return &(*itr);
  else return NULL
}
```

```
template <class Tn, class Ta>
grafo<Tn,Ta>::nodeIterator grafo<Tn,Ta>::
  borrarNodo (grafo<Tn,Ta>::nodeIterator n)
{
  nodo<Tn,Ta>::adyIterator a, ad;
  nodo<Tn,Ta>::iniIterator i, id;
  nodo<Tn,Ta> * nady;
  nodo<Tn,Ta> * nini, * nact;
  grafo<Tn,Ta>::nodeIterator s, old;

 for (a=n->abegin(); a!=n->aend(); )
    {
     nact = a->nodoOrigen();
     nady = a->nodoDestino();
     for (id=nady->ibegin(); id->nodoOrigen()!=
                    nact; ++id);
     id=id->nodoDestino()->borrarIn(*id);
     a = a->nodoOrigen()->borrarAdy(a);
    }
for (i=n->ibegin(); i!=n->iend(); )
    {
    nact = i->nodoDestino();
    nini = i->nodoOrigen();
    for (ad=nini->abegin(); ad->nodoDestino()!=
                    nact; ++ad);
    ad = ad->nodoOrigen()->borrarAdy(ad);
    i = i->nodoDestino()->borrarIn(*i);
    }
```

```
    ncomp--;
    old = s = g->erase(n);
    for (; s!=nend(); ++s)
        s->ind--;
    return old;
}

template <class Tn, class Ta>
void grafo<Tn,Ta>::borrarNodo(nodo<Tn,Ta> *n)
{
    nodeIterator itr;
    for (itr=nbegin(); *itr!=n; ++itr);
    if (itr!=nend())
        borrarNodo(itr);
}
```

```
template < class Tn, class Ta >
  grafo <Tn, Ta> grafo <Tn, Ta>:: copiarGrafo()
  {
    grafo <Tn, Ta> aux;
    grafo <Tn, Ta>:: nodeIterator n;
    nodo <Tn, Ta>:: ady_iterator a;
   for (n = nbegin(); n!=nend(); ++n)
      aux.insertarNodo (n->etiqueta());
   for (n = nbegin(); n!=nend(); ++n)
      for (a = n->abegin(); a!= n->aend(); ++a)
        aux.insertarArco (aux.buscarNodo (
                          a->nodoOrigen()->etiqueta()),
                           aux.buscarNodo (
                          a->nodoDestino()->etiqueta() ),
                           a->etqArco() );
   return aux;
  }
```