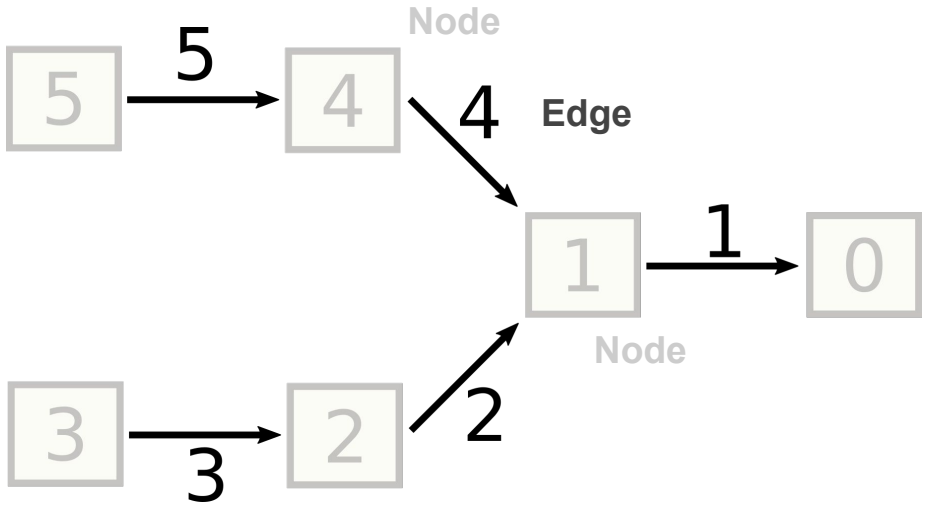
Graph concept

- Nodes can be:
  - Atoms, residues, indexes, strings
- Edges can be:
  - Bonds, whatever



Motivation to create a Generic Graph Class

- In GMML we have graphs and graph algorithms in many places
  - AtomNode, ResidueNode are linked lists; no edges.
  - CondensedSequence uses its own graphs.
  - Glyfinder uses the current generic “GraphDS”, but writes its own functions.
  - ResidueLinkage should be an edge.
  - Etc.
  - We have **independant, type-specific graphs and functions.**

Motivation to create a Generic Graph Class

- GraphDS class is meant to be generic, but is effectively a struct
  - Just holds the data, doesn’t have any functions e.g. cycle detection, subgraph matching.
  - Uses void pointers and records type, user must keep track of this.
  - So far only glyfinder is using it.

Key concepts for any class design

- Only the class should manipulate its own data.
  - Don’t give up control: difficult to change later.
  - Most of gmml is functional programming, rather than object oriented.

Examples of bad ideas:

- Subgraph matching in Grafting program in Glylib:
  - int is_A_in_B(int *pn0match, int A_ln, char A_table[][40][8], int An0lines, int B_ln, char B_table[][40][8], int Bn0lines, int *pB_neighbr, int fTtoLG[][40], int nbc);
- E.g. A_table: where declared? where instantiated? where used?
- Functional programing leads to 400 line for loops. You think through a problem and then code that.
- Complicated problems require that huge amounts of “state” are tracked.
- Works great for small problems!

Why I hate CondensedSequenceSpace:

You need to be a genius to understand it.

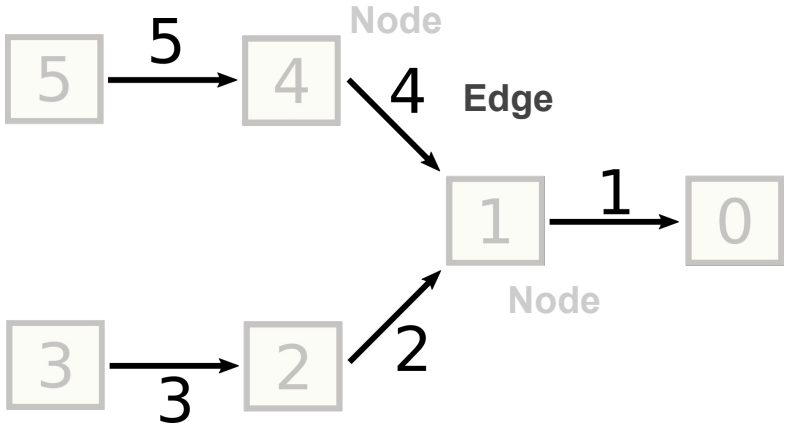
```
class CondensedSequence
{
    public:
        ////////////////////////////////////////////////////////////
        //                      TYPE DEFINITION                   //
        ////////////////////////////////////////////////////////////
        typedef std::vector<CondensedSequenceResidue*> CondensedSequenceResidueVector;
        typedef std::vector<gmml::CondensedSequenceTokenType> CondensedSequenceTokenTypeVector;
        typedef std::vector<CondensedSequenceResidue*> CondensedSequenceResidueTree;
        typedef std::vector<CondensedSequenceGlycam06Residue*> CondensedSequenceGlycam06ResidueTree;
        typedef std::pair<std::string, RotamersAndGlycosidicAnglesInfo*> RotamerNameInfoPair;
        typedef std::vector<RotamerNameInfoPair> CondensedSequenceRotamersAndGlycosidicAnglesInfo;
        typedef std::map<int, std::vector<std::vector<double> > > IndexLinkageConfigurationMap;
        typedef std::map<int, std::vector<std::vector<std::string> > > IndexConfigurationNameMap;
        typedef std::map<int, std::string> IndexNameMap;
        typedef std::map<int, std::string> DerivativeMap;
        enum class Reordering_Approach {PRESERVE_USER_INPUT, LOWEST_INDEX, LONGEST_CHAIN};
        ////////////////////////////////////////////////////////////
```

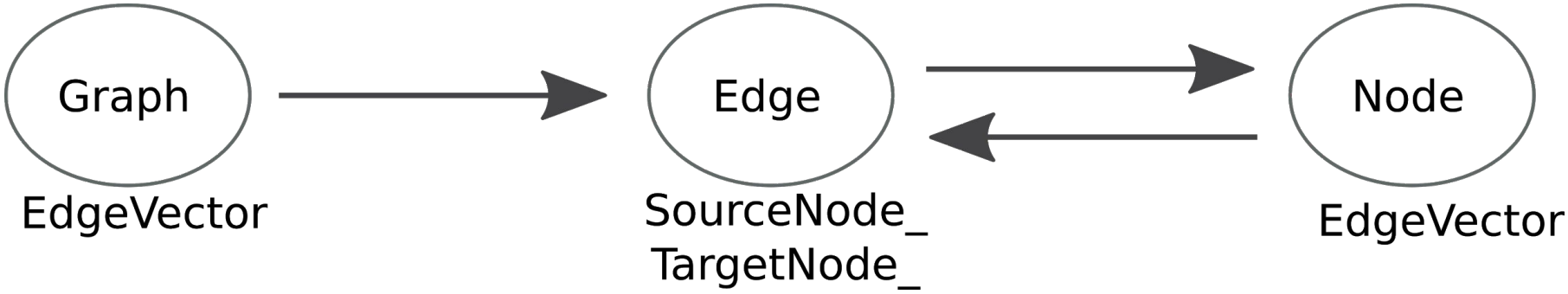
Key concepts for any class design

- Only the class should manipulate its own data.
  - Don’t give up control of data: makes it difficult to change.
  - Most of gmml is functional programming, rather than object oriented.
- Resource allocation is instantiation.
  - The constructor should be designed to allocate everything.
  - The destructor is in charge of releasing everything.

Graph Class Design

- Nodes can be:
  - Atoms, residues, indexes, strings





Template classes in C++

- Not classes, but a template that the **compiler uses** to build classes

```
template <class T>
class Node
{
private:
    T* objectPtr_;
    std::vector<Edge<T>*> edges_;
};
```

What the compiler generates:

```
Class Node
{
private:
    int* objectPtr_;
    std::vector<Edge<int>*> edges_;
};
```

```
//////////Main/////////////
Graph<int> intGraph(vectorOfEdges);
```

Template classes in C++ CAVEATS!

- The compiler needs to see function **definitions** as it creates the class.
  - For normal classes just the declaration is fine.
  - Cleanest solution I found was to include the definitions in the header.
  - Inline for one liners.

```
std::vector<Node<T>*> GetNeighbors(); // Can't
std::vector<T*> GetNodesNeighborsObjects(); //
template <typename T>
    std::vector<Node<T>*> Node<T>::GetNeighbors()
    {
        std::vector<Node<T>*> neighbors;
        for(auto &edge : this->GetEdges())
        {   // Incoming Edges should be ignored
            if (edge->GetTarget() != this)
            {
                neighbors.push_back(edge->GetTarget());
            }
        }
        return neighbors;
    }
```

Template classes in C++ CAVEATS!

- The compiler needs to see function **definitions** as it creates the class.
  - For normal classes just the declaration is fine.
  - Cleanest solution I found was to include the definitions in the header.
  - Inline for one liners.

```
std::vector<Node<T>*> GetNeighbors(); // Can't
std::vector<T*> GetNodesNeighborsObjects(); //
```

```
template <typename T>
    std::vector<T*> Node<T>::GetNodesNeighborsObjects()
    {
        std::vector<T*> neighborObjects;
        for(auto &edge : this->GetEdges())
        {
            neighborObjects.push_back(edge->GetTarget()->GetObjectPtr());
        }
        return neighborObjects;
    }
```

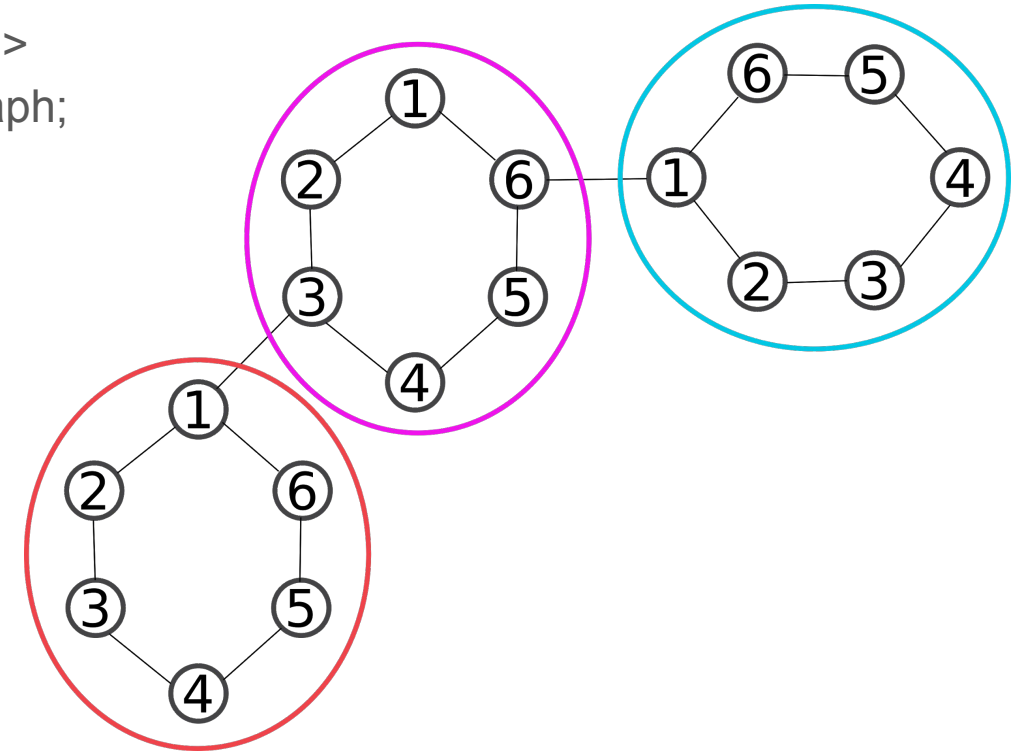
Template classes in C++ CAVEATS!

- The compiler needs to see function **definitions** as it creates the class.
  - For normal classes just the declaration is fine.
  - Cleanest solution I found was to include the definitions in the header.
  - Inline for one liners.

```
inline void SetEdges(std::vector<Edge<T>*> edges) {edges_ = edges;}
inline void AddEdge(Edge<T>* edge) {edges_.push_back(edge);}
```

Layered graphs

- Residue Graph
  - Node could be a Graph<Atom>
- Graph<Graph<Atom>> residueGraph;



Current limitations

- Leaks memory through use of **new:** replace with shared_ptr once design is stable.
- I haven’t figured out Typedefs

Functionality Goals

- Cycle detection is semi-functional.
  - Can say if node is in a cycle
  - Can’t say which cycle
- Subgraph matching is next.
- Pruning (e.g. of hydrogens).
- Graph constructor and graph construction details need hammered out.