# ПРОЕКТИРОВАНИЕ ШАБЛОНА С++ МОДЕЛИРОВАНИЯ ПОИСКОВЫХ ЗАДАЧ В ГРАФАХ.

**Русакова Зинаида Николаевна**
*с.н.с., к.т.н., доцент*
*МГТУ им. Н.Э. Баумана*
*Г. Москва*

В работе рассматривается проектирование обобщенного компонента класса *С++,* осуществляющего построение деревьев решения в поисковых алгоритмах в графах, позволяющего создавать шаблонные коды. В языке С++ обобщенное программирование поддерживается с помощью шаблонов. В шаблонах используются параметризованные типы как параметры, которые передаются в качестве аргументов. Определение шаблонного класса предваряется кодом template < class T >, который сообщает, что определяется шаблон, < class T > - спецификатор типа для переменных, получающих тип как значение. Спецификаторы типов заменяются реальными типами при реализации класса.

На основе обобщенных классов С++ разработан параметризованный класс Poisk_graf, предназначенный для решения задач обработки графов. В методах класса осуществляется решение поисковых задач и формирование деревьев решения графах. В разработанном шаблоне класса Poisk_graf используются три спецификатора типа: template < class T , class Tr , class Tv > class Poisk_graf .

Параметры шаблона передаются как параметры класса и заменяются реальными типами, которые определяются используемыми алгоритмами решения и математическими моделями.

Проектирование шаблона выполняется следующей постановке задачи. Рассматриваются взвешенные конечные графом без циклов и петель, ориентированные в одном направлении. В качестве формальной модели описания графа используется представление графа списком ребер.

Для описания ребер графа используются следующие основные параметры:

номера узлов, инцидентных ребру; метка ребра, соответствующая включению ребра в дерево решения, вес ребра - величина, определяющая его различные физические параметры: расстояние, время, стоимости, штрафы, которая должна быть минимизирована.

Ребро графа описывается классом, полями которого объявляются начальная и конечные вершины - int firstNum, lastNum, вес ребра, метка ребра, которая изначально равна 0, при включении ребра в дерево решения метка устанавливается в 1. В конструктор класса передаются параметры для инициализации.

```
class Rebro {
public:
   int firstNum, lastNum, ves, metka;
```

```
    Rebro(){}
    Rebro(int fn, int ln, int vs):firstNum(fn), lastNum(ln), ves(vs), metka(0){}
    void print(){ cout << "nodes " << firstNum << " " << lastNum ;        }
};
```

Классические методы, лежащие в основе многочисленных алгоритмов обработки графов, методы поиска в ширину и в глубину. Эти методы описывают семейство алгоритмов поиска, которые обеспечивают посещение всех вершин и ребер связного графа. На базе этих методов создаются их модификации, такие как обобщенные алгоритмы и “жадные” алгоритмы.

При решении задач поиска в графах вводится понятие открытых и закрытых вершин или белые и черные вершины. При открытии вершины она становится серой. Вершина – открытая, пока не порождены ее потомки. Эти вершины составляют фронт вершин, являющихся потомками вершины раскрытия. Вершина –закрытая, если в процессе поиска порождены все ее потомки.

Вершины графа изначально раскрашиваются одинаково - все вершины белые. Если в процессе просмотра списка графа для определения потомков вершины открывается белая вершина, она становится серой. Вершина становится черной, когда порождены все потомки

Эти вершины хранятся и обрабатываются в двух списках поиска: список открытых вершин и список закрытых вершин. Вершины графа в процессе поиска из списка открытых или белых вершин переписываются в список закрытых или черных. Этот список закрытых вершин содержит в себе дерево решения задачи, из которого определяются оптимальные пути и деревья покрытия.

Методы поиска в ширину и в глубину отличаются стратегией формирования списков закрытых и открытых вершин. Стратегия метода поиска в ширину: вершина для раскрытия выбирается из головы списка открытых вершин, а потомки записываются в хвост списка открытых. В этом случае моделируется очередь. Стратегия метода поиска в глубину: вершина для раскрытия выбирается из головы списка открытых вершин, но и потомки записываются в голову (механизм стека).

Элементы списков открытых и закрытых вершин определяются классом, где поле целого типа num соответствует номеру выбранной в процессе поиска вершины.

```
class Tnum {
public:
    int num;
    void print(){ cout <<" num "<< this->num <<endl; }
 Tnum(){}      Tnum(int x){num=x; } };
```

Для отображения дерева решения используется класс, в котором определяются поля для записи информации об обратном ребре, содержащем поля целого типа, определяющих вершины предка и потомка: predok, potomok:

```
class Obr_Rebro {
```

```
public:
   int predok, potomok;
   Obr_Rebro(){}
   Rebro(int fn, int ln,):firstNum(fn), lastNum(ln) {}
   void print(){ cout << "nodes " << predok << " " << potomok <<endl;}
};
```

Списки открытых и закрытых вершин и список, описывающий граф сети, создаются на основе методов разработанного шаблонного класса List, моделирующего двунаправленный списки объектов.

В классе List определяются адресные поля: указатели на первое и последнее звено и реализованы основные методы обработки списка: добавить в голову, добавить в хвост, взять из головы, взять из хвоста, просмотр списка.

```
template < class T >
class List {
public:
   Elem <T> * first,* last,* cur;
   List<T>()          {first=0; last=0; cur=0; }
// прототипы методов
   void add (T temp ); //добавить в хвост
   void add_head ( T temp); //добавить в голову
 void del_xwost() ; //удалить из хвоста
   void del_head ( ); // удалить из головы
   void print_Lst()
   { cur=first; while(cur!=0){ cur->print();      cur=cur->next;}}
};
```

Звено списка описывается шаблонным классом с информационным полем, тип которого передается как параметр и двумя указателя на следующий и предыдущий элемент.

```
template < class T >
class Elem {
 public:
          T data;
          Elem * next, * prev;
          Elem( T d ){ data= d; n=0;       next=0; prev=0;}
           void print(){        data.print();          }
    };
```

*Описание шаблона класса моделирования обхода графов.*

Полями параметризованного класса моделирования обхода графов объявляются списки открытых и закрытых вершин для реализации базовых методов поиска в ширину и глубину и список ребер, описывающий граф.

```
   List <Tr> listOpenNodes;
   List <Tr> listCloseNodes;
```

```
List <T> sreb;
```

Списки открытых и закрытых вершин и список, описывающий граф сети, создаются на основе методов класса шаблонного класса List. Типы звеньев передаются как параметр. Поля целого типа – вершины источника и цели incep, cel, которые используются при проверке связности графа и определения путей. Поля flagys, flagnot - целые переменные, определяющие завершение поиска.

В конструктор класса с параметрами передаются аргументы: список вершин графа, вершина источника и целевая вершина. В конструкторе класса вызываются конструкторы создания списков открытых и закрытых вершин и выполняется их инициализация.

В качестве параметров шаблона используются классы, описывающие звенья списков, представляющие ребра графа, вершины, обратные ребра. Параметры шаблона передаются как параметры класса и при реализации соответствуют вышеописанным классам Rebro, Tnum, Obr_Rebro.

```
template < class T , class Tr, class Tv >
class Poisk_graf {
public:
int flagys, flagnot, incep, cel ;
   List <Tr> listOpenNodes; //список открытых вершин
   List <Tr> listCloseNodes; // список закрытых вершин
List <T> sreb; список ребер
Poisk_graf <T, Tr > (){}
Poisk_graf <T, Tr> ( List <T> p, int inw, int celw){//конструктор
   sreb = p;    incep=inw; cel= celw;
   flagys=1, flagnot=1;
   listOpenNodes=List <Tr>();
listOpenNodes.add(incep);
listCloseNodes=List <Tr>(); }
// прототипы базовых методов класса
 int potomki_sh (); //поиск по образцу - очередь
 int potomki_gl (); //поиск по образцу - стек
 int poisk1_gl() ; // построение дерева решения поиск в глубину
 int poisk1_sh_() ; // построение дерева решения поиск в ширину
};
```

В методах класса реализующих поиск поиска в ширину и в глубину формируются списки открытых и закрытых вершин. В основе стратегии всех методов поиска лежит поиск по образцу, целью которого является порождение всех потомков вершины, выбранной для раскрытия. Эти вершины потомки определяют ребра, по которым можно перейти из вершины раскрытия (предка) в вершину потомка. В алгоритме поиска по образцу из головы списка открытых вершин выбирается вершина раскрытия. Для поиска ребер инцидентных этой вершине в цикле просматривается список ребер графа. Цель просмотра

сформировать список всех потомков выбранной вершины, составляющих фронт серых вершин. Эти вершины составляют очередь и записываются в хвост списка открытых вершин.

В задаче поиска по образцу в цикле по списку ребер графа проверяется условие, что вершина раскрытия listOpenNodes.first->data.num совпадает с начальной вершиной текущего ребра по адресу sreb.cur : sreb.cur->data.firstNum и метка ребра равна 0, т.е. ребро не пройдено и оно может быть включено в дерево решения:

listOpenNodes.first->data.num==sreb.cur->data.firstNum

Если условия выполняются, то конечная вершина записывается в хвост списка открытых вершин в методе поиска в ширину и записывается в голову в методе поиска в глубину. Эти вершины составляют фронт серых вершин, инцидентные им ребра включаются в дерево решения, при этом метка ребра устанавливается в 1: listOpenNodes.add(sreb.cur->data.lastNum),

где sreb.cur->data.lastNum потомок вершины раскрытия.

Поиск по образцу реализуется двумя различными методами класса. Один метод формирует очередь в случае поиска в ширину, другой формирует стек при поиске в глубину. Методы возвращают число потомков. Описание метода поиска по образцу для случая формирования очереди: потомки записываются в хвост списка открытых вершин listOpenNodes:

```
template < class T , class Tr>
int Poisk_graf <T,Tr>:: potomki_sh (){
 int i,j;  j=0; //число вершин
 sreb.cur=sreb.first; //начальное звено списка ребер
 while ( sreb.cur )     { //цикл по ребрам графа
          if( listOpenNodes.first->data.num ==sreb.cur->data.firstNum
           && sreb.cur->data.lastNum ==cel) //проверка цели
 { flagys =0; j++; break;}
    else
 if(listOpenNodes.first->data.num==sreb.cur->data.firstNum &&
 sreb.cur->data.metka==0 )         {
           listOpenNodes.add(sreb.cur->data.lastNum);
          j++;    }
 sreb.cur=sreb.cur->next; // переход к следующему ребру графа
    }      return j;
}
```

Построение дерева решения по методу поиска в ширину и поиск всех достижимых вершин и всех возможных путей выполняется в методах класса по следующему алгоритму.

Пока список открытых вершин не пуст:

вычисляется число потомков и формируется фронт серых вершин, записанных в хвост списка j=potomki_sh ();

вершина из головы списка открытых вершин удаляется и записывается в голову списка закрытых вершин - черных вершин, у которых порождены все потомки, инцидентная ветвь составляет ребро дерева решения:

```
listCloseNodes.add_head(listOpenNodes.first->data.num);
listOpenNodes.del_head();
```

Описание метода класса формирования дерева поиска в ширину :

```
template < class T , class Tr>
 int Poisk_graf <T, Tr>::int poisk1_sh_wse_puti() {
          int i,j,k, n;
          listCloseNodes.first=0;
           while ( listOpenNodes.first!=0 && flagys && flagnot){
    j=potomki_sh ();
    listCloseNodes.add_head(listOpenNodes.first->data.num);
           if (!flagys )
           { listCloseNodes.add_head( cel );
                        listCloseNodes.print_List();
                        flagys=1;     return 1;                    }
           else
           if ( j==0 && listOpenNodes.first==0 )
                  { flagnot=0; return 0;}
           }
   }
```

В этом методе определяются все возможные пути из начальной вершины в целевую. Оптимальный путь в смысле минимизации какого - либо критерия от веса ребер определяется в этом случае и для отрицательных значений весов. Для определения кратчайшего пути по количеству вершин в поиске по образцу проверяется вхождение потомка в списке открытых вершин: если вершина есть в списке, повторно она не включается.

Формирование деревьев поиска в глубину осуществляется другим методом, в котором используется механизм стека для формирования и обработки серых вершин: пока не обнаружены все вершины, достижимые из исходной идти “вглубь”, пока есть не пройденные исходящие ребра, и возвращаться и искать другой путь, когда таких ребер нет.

Разработанный шаблон класса является базой для решения многочисленных задач обработки графов. В его методах выполняется формирование деревьев поиска, позволяющих определить оптимальные пути в смысле минимизации цены поиска при возможных отрицательных значениях весов ребер путем выбора из всех возможных путей. В частном случае определяется оптимальный путь в смысле числа вершин.

## Список литературы

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест. Алгоримы. Построение и Анализ., МЦНМО, Москва, 2000, 960 с.

2. Страуструп Б. Язык программирования С++. СПб.: Бином, 1999. с. 990 .

3. Русакова З.Н. Динамические структуры данных и вычислительные алгоритмы Visual C++. Санкт-Петербургбург.2014, 272 с.

4. Русакова З.Н. Система моделирования и интеллектуализации задач принятия решений. Инженерный журнал: Наука и инновации #2.14/2013, с.9.