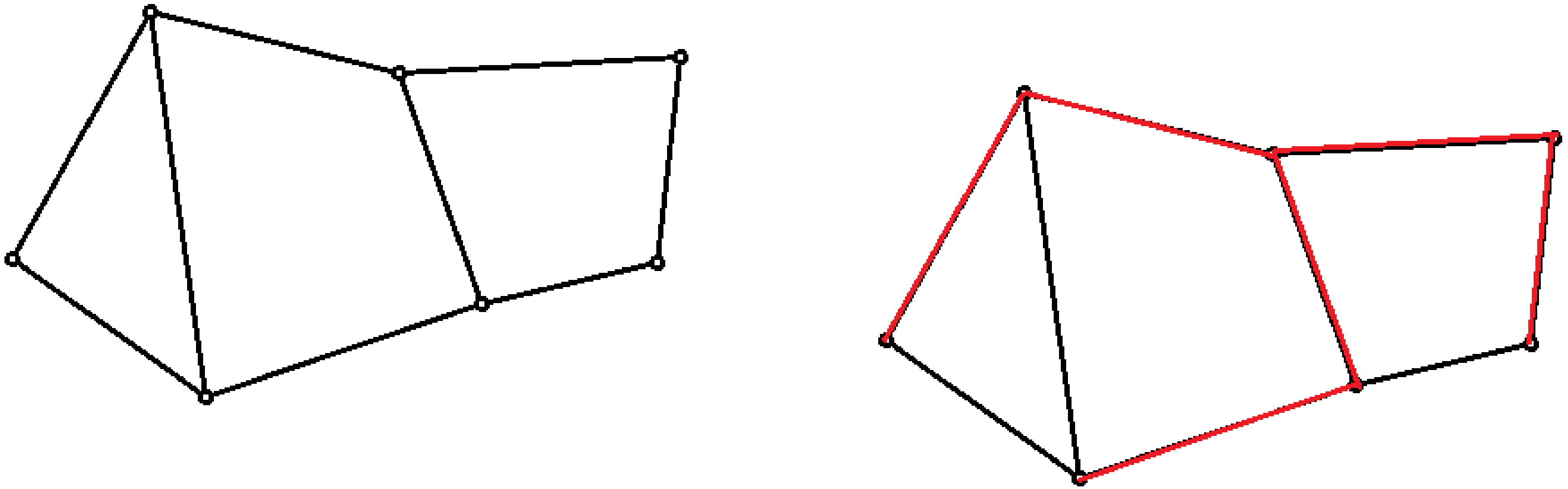
Графы 5

Остовное дерево

- *Остовным деревом* (остовом) связного графа G называется его подграф, содержащий все вершины графа G и являющийся деревом.



Построение остовного дерева связного графа

- Пусть дан связный граф G. Остовное дерево D будем строить по шагам, точнее, будем строить некоторый подграф D графа G, а затем покажем, что он является остовным деревом.
- Шаг 1. Выбираем произвольную вершину и помещаем ее в подграф D.
- Шаг 2. Если D содержит все вершины графа, то завершает работу, в противном случае, переходим к шагу 3.
- Шаг 3. Находим вершину v, которая еще не попала в подграф D и является смежной некоторой вершине w, находящейся в подграфе D. Поскольку граф G является связным, то такая вершина найдется. Помещаем вершину v и ребро (v,w) в подграф D. Переходим к шагу 2.

Обоснование алгоритма построения остовного дерева связного графа

- Поскольку алгоритм завершает работу только тогда, когда все вершины графа G окажутся в подграфе D, то D является остовом графа G.
- На первом шаге в подграф D помещена 1 вершина, на шаге 3 в подграф D помещаются 1 вершина и 1 ребро, причем вершина и ребро выбираются так, чтобы не нарушить связность подграфа D.
- Таким образом, мы построим связный граф, у которого ребер на единицу меньше, чем вершин, т.е. дерево. Значит, подграф D является остовным деревом графа G.

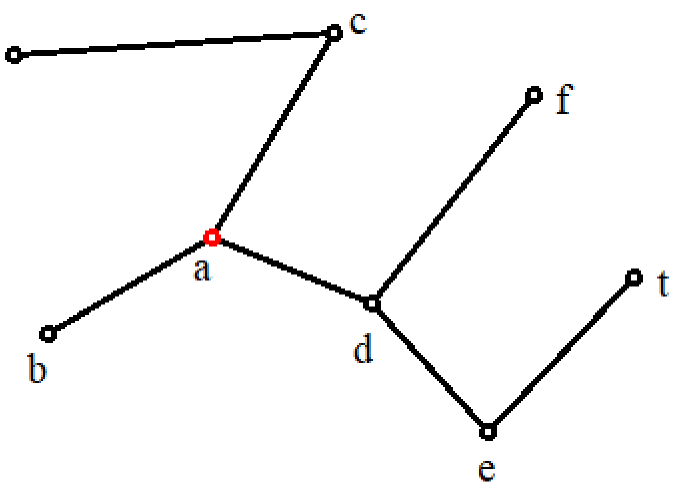
Остовной лес (остов) не связного графа G

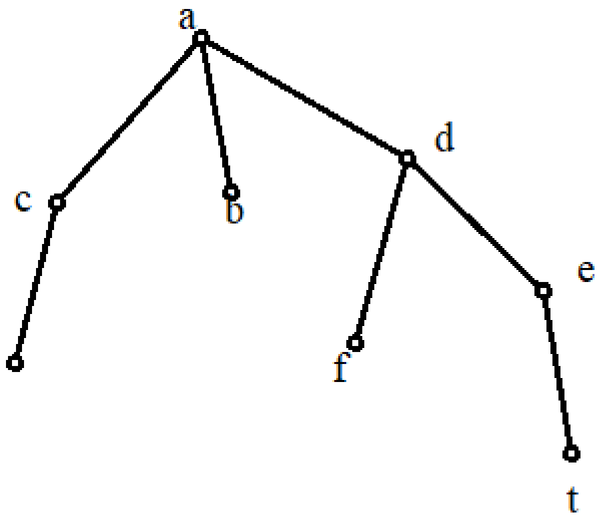
- *Остовным лесом* (остовом) не связного графа G называется его подграф, содержащий все вершины графа G, являющийся лесом и имеющим тоже число компонент связности, что и граф G.

Дерево, как модель иерархических отношений 1

- Мы определили, дерево, как связный граф без циклов. Все вершины дерева равноправны. Но для моделирования иерархических отношений, которые часто встречаются как в практической жизни, так и в математике, и в программировании, используется дополненное определение дерева.
- В этом случае вершины называют, как правило, узлами, а ребра задают отношения, которые называют родительскими. Один узел выделен как корень.
- Если дерево представить веревочной моделью, в которой вершины узелки, а ребра нитки, то в этом случае, взяв за узелок, названный корнем и встряхнув за нее все дерево, мы получим дерево в новом определении.

Деревья





Дерево, как модель иерархических отношений 2

- Корень может иметь сыновей.
- Каждый из его сыновей так же может иметь своих сыновей, т.е. быть родителем для этих узлов и т.д. Каждый узел, кроме корня, имеет одного родителя. Узлы, не имеющие сыновей, называются листьями.
- Итак, новое определение дерева отличается от старого, тем, что узлы упорядочиваются.
- Какое определение дерева используется в конкретный момент времени зависит от контекста.

Представление дерева в памяти ЭВМ 1

- Для работы с деревьями в библиотеке языка C# введен класс *TreeNode*.
- Объекты, которого имеют двойственную природу, их можно рассматривать или как поддерево или как узел, который совпадает с корнем этого поддерева.
- Свойство *Text* определяет имя корня.
- Свойство *Nodes* – является списком всех сыновей корня этого поддерева.
- Каждый сын – это в свою очередь объект типа *TreeNode*
- Удаление узла приводит к удалению всех его потомков, т.е. всего поддерева, корнем которого является удаляемый узел.
- Реализация такой структуры возможна, поскольку в списке *Nodes* хранятся ссылки на узлы являющиеся сыновьями.

Пример

- *TreeNode t=new TreeNode("Корень");*
- Данная команда создает узел – объект типа *TreeNode* , который будет корнем нашего дерева.
- *TreeNode t1=new TreeNode("C1");*
- *t.Nodes.Add(t1);*
- Данная команда добавляет узел *t1* к дереву *t*. Теперь к узлу *t1* можно обращаться и по ссылке *t1* и по номеру *0* в списке *t.Nodes*, кроме того, к этому узлу можно обратиться через свойство *t.FirstNode* поскольку данный узел стоит в списке первым.
- Свойство *t.LastNode* возвращает ссылку на последний элемент в списке *t.Nodes*. Свойство *t.Parent* возвращает ссылку на родительский узел, если *t* корень, она будет равна *null*.
- Свойство *t1.Index* возвращает номер узла *t1*, в списке *Nodes*.

Просмотр дерева

- Для просмотра дерева и работы с ним создан компонент *TreeView*, который является контейнером для объектов *TreeNode* Котпонент *TreeView* имеет свойство *Nodes*, которое является списком объектов *TreeNode*. Котпонент *TreeView* визуальный - это прямоугольник, в котором нарисовано дерево, похожее на дерево папок и файлов.

- Каждый узел может быть свернутым и развернутым, может иметь имя и значок. Свойства для чтения *IsExpanded, IsVisible; IsSelected;* имеют булев тип и сообщают, является ли узел развернутым, видимым и выделенным, соответственно.

- Чаще всего работа происходит с выделенным узлом, поэтому класс *TreeView* имеет свойство *SelectedNode*, которое указывает на выделенный узел, это свойство предназначено и для чтения и для записи.

- Команда *treeView1.SelectedNode.Remove();* удалит выделенный узел, а *treeView1.SelectedNode.Text="op";* переименует выделенный узел.

Работа с деревом

- У класса *TreeView* есть события , связанные с действиями над деревьями :
- *AfterSelect*, *BeforeSelect*, *AfterCollapse BeforeCollapse*, *AfterExpand* BeforeExpand, Первые два события происходят при выделении узла.
- Событие *BeforeSelect* до выделения, *AfterSelect* - после выделения.
- Следующие два события связаны со сворачивание узла, а последние два с разворачиванием узла.
- Все обработчики этих событий имеют формальный параметр e, который, через свойство *Node* передает ссылку на узел, с которым происходит соответствующее событие.

Пример

- *private void treeView1_BeforeCollapse(object sender, System.Windows.Forms.TreeViewCancelEventArgs e)*
- *{*
- *label2.Text=e.Node.Text;*
- *}*
-
- Данный фрагмент программы в момент сворачивания узла будет выводить его имя на экран.

Обходы дерева

Обход дерева – это список, в который заносятся узлы по мере их прохождения.

Наиболее распространены три следующих обхода :

1) прямой;
2) обратный;
3) симметричный.

Определение прямого обхода дерева

- Если дерево T есть нулевое дерево, то в список обхода заносится пустая запись.
- Если дерево состоит из одного узла, то в список записывается этот узел.
- Рассмотрим более сложное дерево T,
  - n-корень дерева T,
  - $T_1, \ldots, T_к$, - поддеревья дерева T, корнями которых, являются сыновья узла n.
- При прохождении в прямом порядке узлов дерева T сначала посещается корень n, затем узлы поддерева $T_1, \ldots, T_k$, так же в прямом порядке.

Определение обратного обхода дерева

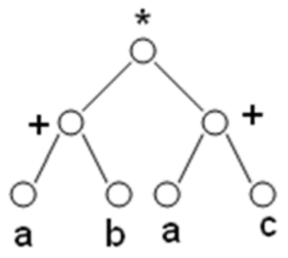
- Если дерево T есть нулевое дерево, то в список обхода заносится пустая запись.
- Если дерево состоит из одного узла, то в список записывается этот узел.
- Рассмотрим более сложное дерево T,
  - n-корень дерева T,
  - $T_1, \ldots, T_к$, - поддеревья дерева T, корнями которых, являются сыновья узла n.
- При прохождении в обратном порядке узлов дерева T, сначала посещается в обратном порядке все узлы поддерева T_1, затем T_2, затем T_3, …, $T_к$, так же в обратном порядке и затем корень n.

Определение симметричного обхода дерева

- Если дерево T есть нулевое дерево, то в список обхода заносится пустая запись.
- Если дерево состоит из одного узла, то в список записывается этот узел.
- Рассмотрим более сложное дерево T,
  - n-корень дерева T;
  - $T_1, \ldots, T_к$, - поддеревья дерева T, корнями которых, являются сыновья узла n.
- При прохождении в симметричном порядке узлов дерева T, сначала посещаются в симметричном порядке все узлы поддерева T_1, далее корень n, затем последовательно в симметричном порядке все узлы поддеревьев $T_2, \ldots T_к$.

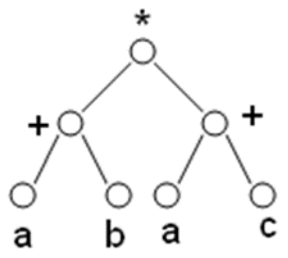
Помеченные деревья

- Часто каждому узлу дерева ставят в соответствие метку-значение. Дерево, у которого узлами в соответствие поставлены метки называются помеченными деревом.
- Метка узла – это не имя узла, а значение, которое храниться в узле. Рассмотрим дерево синтаксического разбора арифметического выражения: (a+b)*(a+c). Оно представлено на рисунке



Обходы помеченных деревьев

Часто при обходе дерева составляется список не имён узлов, а их меток.
Прямой обход : *,+,a,b,+,a,c
Обратный обход : a,b,+,a,c,+,*
Симметричный обход : a,+,b,*,a,+,c



В случае прямого обхода дерева мы получаем префиксную форму выражения, где оператор предшествует левому и правому операндам. Такая форма выражения называется прямой польской записью.

Обратное упорядочивание меток даёт постфиксное представление выражения, в этом представлении оператор следует сразу за операндами. Такая форма выражения называется обратной польской записью.

Отметим, что в префиксных и постфиксных формах нет необходимости отделять или выделять отдельные выражения скобками.

Обходы графа

- ***Обход*** графа – это некоторое систематическое перечисление его вершин или ребер.
- Среди обходов наибольшую известность получили «Поиск в глубину» и «Поиск в ширину».
- Эти алгоритмы лежат в основе многих конкретных алгоритмов на графах.

Поиск в глубину

При реализации этого алгоритма используется вспомогательный объект – стек. Кроме того, попавшие в стек вершины метятся.

- 1 шаг. В стек помещаем произвольную вершину и метим ее.
- 2 шаг. Из стека извлекаем вершину и помещаем в список обхода. Обозначим, извлеченную из стека вершину буквой х. В стек помещаем все непомеченные смежные вершине х вершины, попавшие в стек вершины метим.
- Шаг 3. Проверяем, есть ли в стеке вершины, если есть, то переходим к шагу 2, в противном случае завершаем работу.

Очевидно, что если граф связный, то мы перечислим все его вершины, если не связный, то только те вершины, которые находятся той же компоненте связности, что и выбранная вершина.

Поиск в ширину

При реализации этого алгоритма используется вспомогательный объект – очередь. Кроме того, попавшие в очередь вершины метятся.

- 1 шаг. В очередь помещаем произвольную вершину и метим ее.
- 2 шаг. Из очереди извлекаем вершину и помещаем в список обхода. Обозначим, извлеченную из очереди вершину буквой x. В очередь помещаем все непомеченные смежные вершине x вершины. Попавшие в очередь вершины метим.
- Шаг 3. Проверяем, есть ли в очереди вершины, если есть, то переходим к шагу 2, в противном случае завершаем работу.