**Лекция №9** Алгоритмы на графах.

**ЦЕЛЬ**: Ознакомить с понятиями: Алгоритмы на графах. Анализ графов. Поиск в глубину и ширину. Раскраски. Паросочетания. Оптимальные алгоритмы. Алгоритм Дейкстры,Прима. Алгоритм Краскала.

**ВОПРОСЫ**:

1. Алгоритмы на графах. Анализ графов.
2. Поиск в глубину и ширину. Раскраски. Паросочетания.
3. Оптимальные алгоритмы. Алгоритм Дейкстры, Прима. Алгоритм Краскала.

**ОСНОВНЫЕ ПОНЯТИЯ:** Алгоритмы на графах. Анализ графов. Поиск в глубину и ширину. Раскраски. Паросочетания. Оптимальные алгоритмы. Алгоритм Дейкстры,Прима. Алгоритм Краскала.

Деревья находят широкое применение при проектировании алгоритмов и, в частности, структур данных. Отсылая читателя к литературе по теории графов, будем пользоваться такими понятиями как узел, ребро, лист, потомок, сын, левый потомок правый потомок, предок, отец, корень, ветвь и другими. Регулярным деревом назовем дерево, в котором фиксировано максимально возможное (как правило, небольшое) число потомков для каждого из его узлов. Регулярность дерева позволяет фиксировать число полей, достаточное для представления любого узла. Для регулярных деревьев более экономным по памяти может оказаться представление дерева с помощью массива. Как недостаток такого представления графа можно отметить неудобство при динамической модификации графа, например, добавление к графу ребра может потребовать большого количества пересылок в массиве inf.

**Нахождения кратчайших путей в графе .**

*Входные данные:*

• Граф *G* с взвешенными ребрами (под весами можно понимать дли-ны ребер, если речь идет о геометрическом графе, или любые дру-гие числовые характеристики ребер).

• Стартовая вершина *s* (вершина от которой вычисляются расстоя-ния до всех остальных вершин).

*Выходные данные:*

• Массив *dist*[1.. *n*], (*dist*[*i*] – кратчайшее расстояние от вершины s до вершины *i*).

• Массив *up*[1..*n*], (*up*[*i*] – предпоследняя вершина в кратчайшем пу-ти из *s* в *i*).

***Замечание.*** Приводимый ниже алгоритм Дейкстры корректно решает задачу для графов с неотрицательными весами вершин. Если же в графе есть ребра с отрицательными весами, но нет циклов с отрицательным суммарным весом, то для решения задачи можно использовать алгоритм Форда, Беллмана.

**Алгоритм Дейкстры***. см литературу.*

**Общие сведения. Деревья поиска** предназначены для представления словарей как абстрактного типа данных. Также как и приоритетные очереди они представляют взвешенные множества, но с другим набором операций, а именно • ***Search*** – поиск элемента с заданным ключом, • ***Minimum*** – поиск элемента с минимальным ключом, • ***Maximum*** – поиск элемента с максимальным ключом, • ***Predecessor*** – поиск элемента с предыдущим ключом, • ***Successor*** – поиск элемента со следующим ключом, • ***Insert*** – вставка элемента со своим ключом, • ***Delete*** – удаление указанного элемента. Дерево поиска может быть использовано и как словарь, и как приоритетная очередь. Время выполнения основных операций пропорционально высоте дерева.

**Представление двоичных деревьев поиска.** Двоичным деревом поиска называется корневое двоичное дерево, каждому узлу которого по-ставлен в соответствие взвешенный элемент. При этом для каждого узла *x* выполняется следующее условие. *(подробное описание и применение см. литературу)*

**Операции с двоичным поисковым деревом.** Покажем, что двоичные поисковые деревья позволяют выполнять операции *Search*, *Minimum*, *Maximum*, *Successor* и Predecessor за время Ο (*h*) , где *h* – высота дерева. *(подробное описание и применение см. литературу)*

**Красно-черные деревья.** Мы видели, что основные операции с двоичным поисковым деревом высоты *h* могут быть выполнены за Ο (*h*) действий. Деревья эффективны, если их высота мала, но если не принимать специальные меры при выполнении операций, малая высота не гарантируется, и в этом случае дере-вья не более эффективны, чем списки. Для повышения эффективности операций, используют различные приемы перестройки деревьев, так чтобы высота дерева была величиной Ο (log *n*). Такие приемы называются балансировкой деревьев. При этом используются разные критерии качества балансировки. Одним из видов сбалансированных деревьев поиска являются так называемые красно-черные деревья, для которых предусмотрены операции балансировки, га-рантирующие оценку высоты величиной Ο (log *n*). Частным случаем такой балансировки является АВЛ-балансировка. *(подробное описание и применение см. литературу)*

**Красно-черное дерево** – это расширенное двоичное дерево поиска, вершины которого разделены на красные (red) и черные (black) так, что 1. Каждый узел либо красный, либо черный. 2. Каждый лист (nil-узел) – черный. 3. Если узел красный, то оба его ребенка черные. 4. Все пути, идущие вниз от корня к листьям, содержат одинаковое количество черных узлов. Свойства 1–4 называют RB-свойствами. Узлы красно-черного дерева будем представлять записями вида *Node* = (*color*, *key*, *left*, *right*, *parent*).

**Комбинаторные свойства красно-черных деревьев.** *подробное описание и применение см. литературу)*

**Б-деревья** это один из видов сбалансированных деревьев, при котором обеспечивается эффективное хранение информации на магнитных дисках и других устройствах с прямым доступом. Б-деревья похожи на красно-черные деревья. Разница в том, что в Б-дереве узел может иметь много детей, на практике до тысячи, в зависимости от характеристик ис-пользуемого диска. Благодаря этому константа в оценке *O* (log *n*) для высоты дерева существенно *меньше*, чем для черно-красных деревьев. Как и черно-красные деревья, Б-деревья позволяют реализовать многие операции с множествами размера n за время *O* (log *n*). **Основные операции с Б-деревьями.** *(подробное описание и применение см. литературу)*

**Особенности работы со структурами, данных размещаемых на диске.** Алгоритмы, работающие с Б-деревьями, хранят в оперативной памяти лишь небольшую часть всей информации (фиксированное число секторов). Диск рассматривается как большой участок памяти, работа с которым происходит следующим образом: перед тем как работать с объектом *x*, мы должны выполнить специальную операцию *Disk-Read*(*x*) (чтение с диска). После внесения изменений в наш объект *x*, мы выполняем операцию *Disk-Write* (*x*) (запись на диск). *(подробное описание и применение см. литературу)*

**Метод поиска в ширину.** *(подробное описание и применение см. литературу)*

Идея поиска в ширину состоит в том, чтобы посещать вершины в порядке их удаленности от некоторой заранее выбранной или указанной стартовой вершины *a*. Иначе говоря, сначала посещается сама вершина *a*, затем все вершины, смежные с *a*, то есть находящиеся от нее на расстоя-нии 1, затем вершины, находящиеся от *a* на расстоянии 2, и т.д.

**Метод поиска в глубину** *(подробное описание и применение см. литературу)*

Поиск в глубину – вероятно, наиболее важная ввиду многочисленности приложений стратегия обхода графа. Идея этого метода – идти вперед в неисследованную область, пока это возможно, если же вокруг все исследовано, отступить на шаг назад и искать новые возможности для продвижения вперед. Метод поиска в глубину известен под разными назва-ниями, например, «бэктрекинг», «поиск с возвращением».

**Раскраска вершин** *(подробное описание и применение см. литературу)*

*Раскраской* вершин графа называется назначение цветов его вершинам. Обычно цвета – это числа 1, 2, ..., *k*. Задача о раскраске состоит в нахождении правильной раскраски данного графа *G* в наименьшее число цветов. Это число называется *хроматическим числом* графа и обозначается χ(*G*).

**Раскраска ребер.** Наряду с задачей о раскраске вершин имеется задача о раскраске ребер графа, когда цвета назначаются ребрам. *(подробное описание и применение см. литературу)*

**Паросочетания и реберные покрытия.**Паросочетанием в графе называется множество ребер, попарно не имеющих общих вершин. Задача о паросочетании состоит в том, чтобы в данном графе найти паросочетание с наибольшим числом ребер. *(подробное описание и применение см. литературу)*

**Задача об оптимальном каркасе и алгоритм Прима .** В *алгоритме Прима* на каждом шаге рассматривается частичное решение задачи, представляющее собой дерево. *(подробное описание и применение см. литературу)*

**Алгоритм Краскала.** Другой жадный алгоритм для задачи об оптимальном каркасе известен как *алгоритм Краскала*. В нем тоже на каждом шаге рассматривается частичное решение. Отличие от алгоритма Прима состоит в том, что в алгоритме Краскала частичное решение всегда представляет собой остовный лес *F* графа *G*, то есть лес, состоящий из всех вершин графа *G* и некоторых его ребер. *(подробное описание и применение см. литературу)*

**ВЫВОДЫ:**

Алгоритм Дейкстры корректно решает задачу для графов с неотрицательными весами вершин. Деревья поиска предназначены для представления словарей как абстрактного типа данных. Деревья эффективны, если их высота мала, но если не принимать специальные меры при вы-полнении операций, малая высота не гарантируется, и в этом случае дере-вья не более эффективны, чем списки. Для повышения эффективности операций, используют балансировку деревьев.