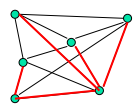
算法第三次测验

Chapter 10 Greedy Algorithms on Graphs

Minimum Spanning Trees最小生成树

问题描述：

给出城市的集合，如何在城市之间修路，要求路的总长最短，并且所有的城市之间可达。



抽象模型：

已知：无向连通图a connected undirected graph G

每条边都有权值weight（例如：长度）

目标：获得能够连接所有节点的边集合，并且有最小的权值

最小生成树定义：

无向图的最小生成树是能保持图连通的边子集。最小生成树一定是棵树（没有环circle）——原因是如果有环的话，我们可以取走一条边，剩下的节点还是连通

Kruskal’s algorithm

这个算法是用来求最小生成树，并且是基于贪心算法。

算法基本步骤：

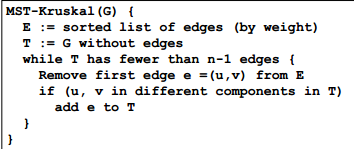
1. 新建图G，G中拥有原图中相同的节点，但没有边

2. 将原图中所有的边按权值从小到大排序

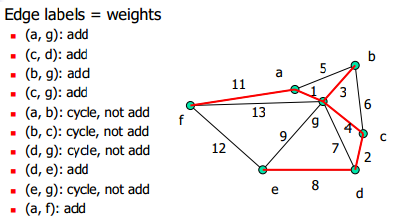
3. 从权值最小的边开始，如果这条边连接的两个节点于图G中不在同一个连通分量中，则添加这条边到图G中

4. 重复3，直至图G中所有的节点都在同一个连通分量中

基于以上步骤，我们用E来存储所有排序之后的边，T是节点的集合，每次选择当前最小权值的边，如果此边的两个端点在T之中不在一个连通分量之中，就将这条边添加到T之中。我们可知，有n个点的集合，最少要n-1条边才能使得图连通起来，所以如果T的节点之间少于n-1条边，就继续寻找新边。



样例：

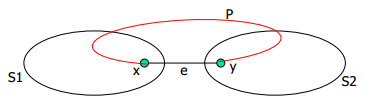


证明Kruskal’s算法的正确性：

我们假设有条边在真正的最小生成树MST T\*之中，但是不在Kruskal’s算法生成的 之中。边e把T\*分成两部分，S1和S2。

因为e不在之中，用其他边连接x,y。因此在这个图之中，至少有一个边e’连接S1，S2。e’一定比e有更小的权值，因为每次选择最小权值的边。

因此我们可以把e从T\*之中删除，并且把e’添加到T\*之中获得一个权重更小的树。这与T\*的定有是矛盾的。



Kruskal’s Algorithm:运行时间

排序的时间复杂度：O(mlogm)

M条边，n个点

因为每次查找边的长度，都要先找到端点之后才能获得，所以m=O(n^2)

因此排序的时间复杂度是O(m log n)

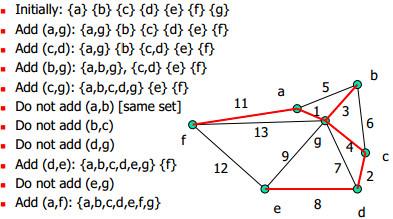
在循环之中的时间复杂度

验证两点结点是否连通：O(m)

连接两个节点，当新边增加的时候：O(n)

Kruskal’s using Disjoint Sets

使用互斥的集合作为图的存储方法



数据结构：

互斥集合的数目

每个集合包含的从同一个ground set得出的元素

并且高效地支持以下操作：

Find(u):输入一个元素u，返回包含这个元素的集合名字

Union(u,v): 将u,v两个集合合并

Using Union-find in MST Algorithms

Find() 和Union()操作支持MST算法：

·每个节点最初都是单个的结合

·Edge joining vertices-> merge into same component->merge the disjoint sets边连接节点，伪代码中的add e to T->合并到同一个连通分量->合并互斥的集合

·验证两个顶点u,v是否属于同一个连通分量：Find(u)=Find(v)

·合并顶点u,v，先找到它们所在的集合，然后合并它们：Union(Find(u),Find(v))

我们需要数据结构支持union和find操作：

1. 数组Array

A[i]=是元素i的集合名



1,2,7在集合1里面，3，4在集合3里面……

Find(i)：返回A[i]，O(1)

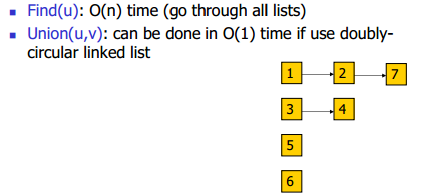
Union(i,j):找到i和j的集合名字，然后将所这两个集合名字的元素都改成相同的名字,O(n)



原先5是属于集合5的，2属于集合1，节点2和5需要合并，就是合并集合1,5，我们随意选择一个相同的名字——1，并把原先集合1,5的所有元素都置为1

1. 链表

用链表的形式把所有的对象都放在一个集合（用相同的名字）



双向循环链表doubly circular linked list

1. 数组和链表Array-and-Linked-List

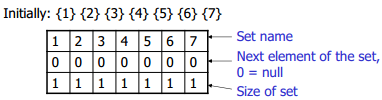
对于每个元素，需要知道：

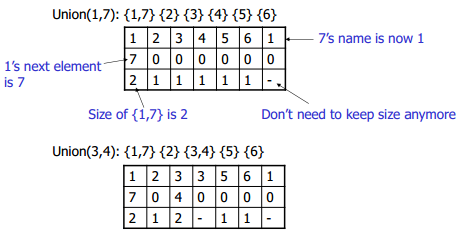
集合名称

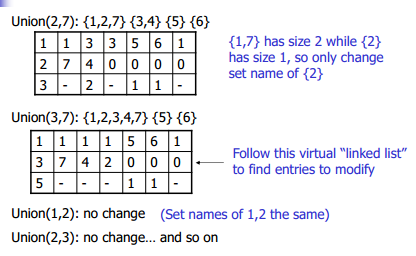
元素中指向的下一个元素

集合的大小

Size of set is kept because we use the weighted-union heuristic: always modify the smaller set







如何输出链表：如果我们要输出第一个集合{1,2,3,4,7}，先搜索A[1].next，得知下一个元素是3。然后搜索A[3].next，得知是7。一直搜索直到A[i].next=0,表明是链表结尾

Kruskal’s时间复杂度

Find(u)是O(1)

Union(u,v)最差情况下是O(n)

Union()分析：

每次union调用的时候，只有最小的集合发生变动

变动之后，变动集合的大小会翻倍

因此每次调用至少改动logn次，总的时间复杂度是Union=O(nlogn)

总的运行时间

O(mlogn)用来排序

O(m)用来FindO(nlogn)用来Union

所以总共O(mlogn)

有没有另外的贪心算法：

思路——选取一个点，每次选择离这个节点最近的节点

缺点——只能生成一棵MST，不能生成森林（当图是由几个连通子图构成的时候）

Prim’s Algorithm

算法思想

At every step, find minimum-weight outgoing edge

Enlarge component

输入：一个加权连通图，其中顶点集合为V，距离数组D；

初始化：S = {s}，其中s为集合V中的任一节点（起始点），D记录S集合到非S集合点的距离；

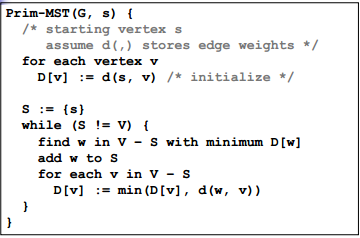
重复下列操作，直到S = V：

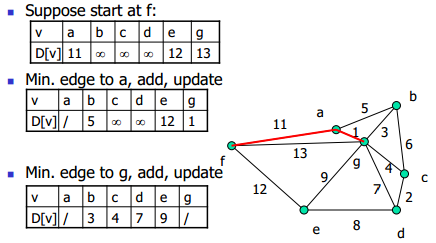
在数组D中选取权值最小的边D[w]，其中w为集合V-S中的节点（如果存在有多条满足前述条件即具有相同权值的边，则可任意选取其中之一）；

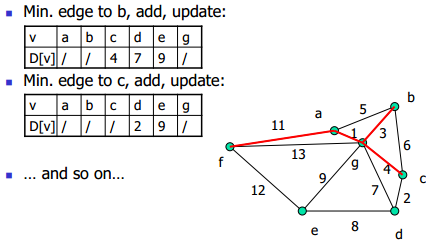
将w加入集合S中，查看因为w的加入导致S中起点s到非S集合中点的距离变化值，并修改



伪代码







Prim’s时间复杂度

寻找最小边Find minimum：O(n)

改变权值:O(m),只要考虑不在S之中的边

如果使用数组array的话，

O(n)查找最小值，O(1)改变数值

总共O(m+n^2)=O(n^2)

使用一个堆heap（使用堆的数据结构，此数据结构也是由数组实现）的话



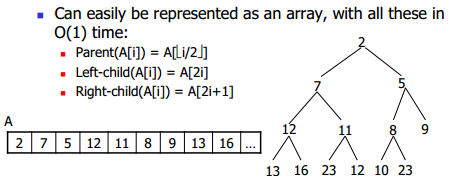
Heap

将元素存储为完全二叉树

从左到右排列节点，do not start a new level unless it is full

平衡二叉树Balanced (height = O(log n))

Parent ≤ both children (for min-heaps; max-heap is opposite)当用数组存储二叉树的时候



堆（小根堆）的操作

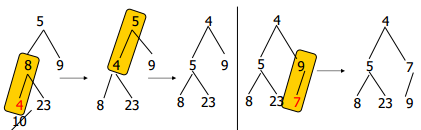
小根堆就是父节点的值一定小于孩子节点

修改节点

修改节点，并且将此节点的值和父节点的值进行比较，如果小的话进行交换，直到比父节点值大或者交换到根节点

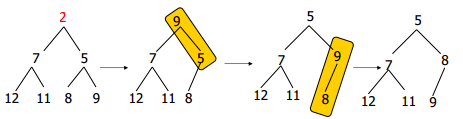
插入节点

在树的最后一个叶子节点边上插入新的节点，与父节点进行大小比较，如果值小的话交换，直到比父节点值大或者交换到根节点



删除最小值

最小值一定是根节点，每次删除根节点。然后根节点的元素替换成最后一个元素，再进行值的大小调整。



使用小根堆的方法能够减少时间复杂度。每次取走根节点，然后树做以上调整。就把寻找最小值的时间复杂度从O(n)减少到O(logn)。因为最多调整树高的操作数，而树高为logn。有兴趣的可以去看看堆排序。

The Shortest Path Problem最短路径问题

问题描述：

输入：一张有向图G，边权值（edge weight）的集合，起点s

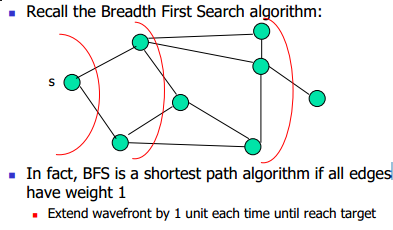
输出：找到s到其他所有点的最短路径

最差的思路就是使用穷举法，找出所有的路径，然后再选择最短的。

最常见的错误思路：每一步选择最短的边，使用贪心的思想。

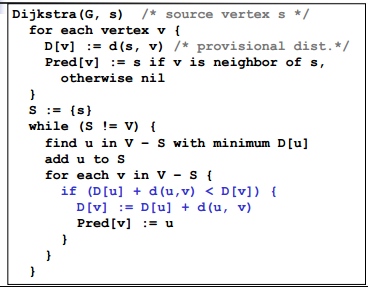
相似算法：BFS

BFS回顾



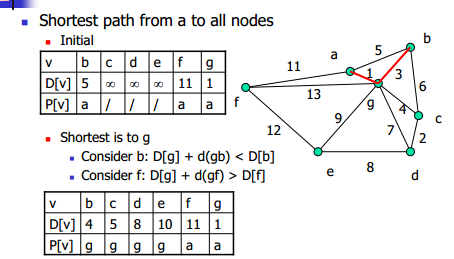
BFS可以求得边权值是1的时候的最短路径。我们在BFS的思想上，扩展了此算法，形成了一个新算法Dijkstra’s algorithm（这个算法非常重要，但是有最短路径的问题，基本上绕不开这个算法。国内学习到这块的时候，和bellman-ford一起学，两者在求最短路径输入上的区别就是Dijkstra要求权值非负）

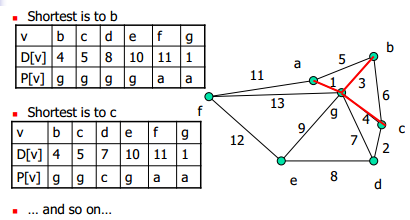
伪代码：

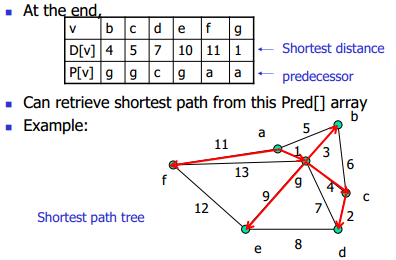


代码解释：

D这个数组存储了起点s到各个点的距离（只能经过S集合里面的点，经过S集合之外点的路径不算）。Pre这个数组存储了最短路径终点之前的那个节点。一开始初始集合只有起点。然后每一步随意选择一个不在S之中的节点，放到S之中，并且修改起点s到最新加入的节点路径长度。







注意：这个代码和Prim’s Algorithm还是很像的，千万不要搞混。Prim’s Algorithm是求最小生成树的，基于无向图，D数组存储的是初始集合S到V-S集合距离集合。而Dijkstra是求最短路径，D数组存储的是起点s到其他所有节点的最短路径长度（只经过S集合中的节点）。但是两者都在添加新的节点进初始集合S的时候，有个修改D数组的操作。

Dijkstra时间复杂度

