Algorihtms - 5 Graph

Graph Representation

G = (V, E)
V is a set of nodes/vertices, E are edges

Adjacency list:

> space: $O(V)$
> time list adj. to u $\theta(degree(u))$
> time to determine if $(u, v) \in E : \theta(degree(u))$

Adjacency Matrix representation:

> space: $\theta(V^2)$
> time to list adj. to u $\theta(V)$
> time determine if $(u, v) \in E : \theta(1)$

trade of between space and time

Graph Search Problem

BFS - Breadth First Search

```
BFS(V, E, s)
    for each u in V - {s}
        u.d = inf
    s.d = 0
    ENQUEUE(Q, s)
    while Q != Empty
        u = DEQUEUE(Q)
        for each v in G.Adj[u]
            if v.d = inf
                v.d = u.d + 1
                ENQUEUE(Q, v)
```

O(V+E)

DFS - Depth First Search

```
DFS(G)
    for each u in G.v
        u.color = WHITE
    time = 0
    for each u in G.V
        if u.color = WHITE
            DFS-VISIT()
DFS-VISIT(G, u)
    time = time + 1
    u.d = time
    u.color = GRAY
    for each v in G.Adj[u]
        if v.color = WHITE
            v.pi = u // predecessor subgraph
            DFS_VISIT(G, v)
    u.color = BLACK
    time = time + 1
    u.f = time
```

Uses

> determine connected components
> find cycles in directed/undirected graph
> subroutine in topological sort of a directed graph
> find strongly connected components of a directed graph

Labeling the edges of a directed graph

> edge (u, v)
> Tree edges u.d < v.d < v.f < u.f
> back edge v.d < u.d < u.f < v.f
> forward edges u.d < v.d < v.f < u.f
> cross edges v.d < v.f < u.d < u.f

Theorem (white-path theorem)

> in the DFS forest, v is a descendant of u if and only if at time u is discovered there is a path, from u to v consisting of only white vertices(except u which is colored grey)

> Theorem:
> DFS on an undirected connected graph produces an ordering so that every edges is a tree edge or a back edge

Topological Sorting

DAG - directed acyclic graph

Lemma: a directed graph G is acyclic if and only if a DFS of G has no back edges
Given a DAG, we can create a total order of the vertices -> topological sort

Topological Sort(G)

> ```
> TOPOLOGICAL-SORT(G)
>     call DFS(G)
>     as each vertex is finished, insert into front of lined list
>     return the linked list
> ```
>
> prove:
>
> > It suffices to prove that if u-> w then w.f < u.f and thus u is placed before w in the topological sorting algorithm
> >
> > To prove this, it suffices to prove that for any edge (u, v) then v.f < u.f and u is placed before v in the topological sorting algorithm.
> >
> > for any edge (u,v) explored by DFS:
> >
> > > v is grey -> impossible exists v.d but not v.f (thus (u,v) is a back -edge result in a circle)
> > > v is white -> v is a descendant of u : v.f < u.f
> > > v is black -> if v has finishing time v.f must < u.f
>
> applications:
>
> > inheritance for c++ classes or java interfaces
> > prerequisites scheduling

Connectedness

> **undirected graph**
>
> > graph G is connected if every u, v in C then there is a path from u to v
> > **connected component:**
> >
> > > maximal set of vertices such that for all u, v in C, u->v(so indicated v->u)
> >
> > The undirected graph is connected if there is only one tree in the DFS forest
>
> **directed graph (digraph)**
>
> > graph G is connected if every u, v in C then there is a path from u to v, and a path from v to u
> > **strongly connected component:(SCC)**
> >
> > > maximal set of vertices such that for all u, v in C, u->v, v-> u
>
> > Lemma:
> >
> > > let C, C' be distinct strongly connected components. Let u,v in C and u',v' in V', and suppose u->u' then it is not possibile that v'->v
> >
> > Lemma:
> >
> > > Let C and C' be distinct strongly connected components let u in C and v' in C', (u, v') in E then f(C) > f(C') (f(C) means max{v.f for v in C})
> > >
> > > proof:
> > > let x be the first discovered vertex in C or C'
> > > if x in C
> > >
> > > > at tiem x.d, all the vertexx in C ????
> > >
> > > corollary:

> > How can we find the SCC of G:
> >
> > > observations 1:
> > >
> > > > when DFS-VISIT(G,u) finishes all nodes reachable from u has been visited.
> > > > It will get stuck in the component. source, sink
> > >
> > > observations 2: G^T and G have the same SCC
> > >
> > > ```
> > > SCC(G)
> > >     call DFS(G) to compute finishing times u.f for all u
> > >     compute GT
> > >     call DFS(GT) but in the main loop consider vertices in order decreasing u.f
> > >     output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC
> > > ```

Spanning Trees

> weight of graph
>
> Greedy algorithm - find minimum spanning tree
>
> **Kruskal**
>
> > find smallest edge
>
> > ```
> > KRUSKAL(G,w)
> >     A = empty
> >     for each vertex v in G.V
> >         MAKE-SET(v)
> >     sort the edges of G.E into nondecreasing order by weigh w // O(ElogE)
> >     for each  (u, v) taken from the sorted list # O(E+V)
> >         if FIND-SET(u) != FIND-SET(v)
> >             A = A U{(u, v)}
> >             UNION(u, v)
> >     return A
> > ```
>
> ...TBC