Graph Algorithm

- Graph, a pervasive data structure in computer science. Hundreds of interesting computational problems defined in terms of graph.
- A graph $G = (V, E)$, V set of vertices $\{v_1, v_2, \ldots, v_n\}$, E set of edges $\{(u, v) : u, v \in V\}$.

A graph $G = (V, E)$

V a set of vertices, $|V|$: number of vertices.
E a set of edges, $|E|$: number of edges.
In the book, it might uses V to represent $|V|$ and E to represent $|E|$.
Time complexity is defined in terms of the two variables V and E.
Vertex set of a graph G: $V[G]$, edge set $E[G]$.

Representation of Graphs

- Adjacency list,
  - provide a compact way to represent sparse graph ($|E|$ is much less than $|V|^2$)
  - Adjacency matrix, $G = (V, E)$ consists of an array Adj of $|V|$ lists.
  - For each $u \in V$, $Adj[u]$ contains pointers to all the vertices v, s.t., $(u, v) \in E$.
  - $Adj[u]$ consists of all the vertices adjacent to u in G.

Adjacency Matrix

- $G = (V, E)$, vertices are numbered $1, 2, \ldots, |V|$.
- A $|V| \times |V|$ matrix $A = (a_{i,j})$ s.t.,

$$a_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- A graph is directed, edges are arcs.
- Weighted, edges has an associated weight,
- weight function : $w : E \to R$,
- $w(u, v)$: weight of edge $(u, v) \in E$.

Minimum Spanning Tree

- Design of electronic circuit, to connect n pings, try to use the least amount of wire, there are $n-1$ wires.
- Model the problem as a weighted graph $G=(V,E)$, weights are distance between pings.
- Find a spanning tree T that total weight $w(T)=\sum_{(u,v)\in T} w(u,v)$ is the least.
- spanning tree, the tree span the graph,
- the least cost, minimum-spanning-tree problem.

Growing a minimum spanning tree

- Input: a connected, undirected graph $G = (V, E)$,
- with weight function $w : E \rightarrow R$.
- wish to find the minimum spanning tree of G.

Greedy Strategy, A "Generic Strategy"

- Grwoing a minimum spanning tree one at a time.
- maintain the loop invariant, "prior to each iteration, A is a subset of some minimum spanning tree".
- At each step, determine an edge (u, v) that can be added to A without violating the invariant ($A \cup \{(u, v)\}$ is also a subset of the minimum spanning tree).
- (u, v) a safe edge of A.
- Keep on inserting safe edges until MST is formed.
- Tricky part is to find a safe edge.

Some definitions

- A **cut** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V.
- An edge $(u, v) \in E$ **crosses** the cut $(S, V - S)$ if one of its endpoints is in S and and the other is in $V - S$.
- A cut **respects** the set A of edges if no edges in A crosses the cut.
- An edge is a **light edge** crossing a cut if its weight is the minimum of any edge crossing the cut.

Theorem: Let $G = (V, E)$ be a connected undirected graph with a real-value weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing $(S, V - S)$. The edge $(u.v)$ is safe for A.

Proof Let T be an minimum spanning tree that includes A and assume that T does not contain the light edge (u, v).
We shall construct another minimum spanning tree T' that includes $A \cup \{(u, v)\}$.
Since T does not include (u, v), inserting (u, v) forms a cycle with the edges on the path p from u to v in T.
u and v are on opposite sides of the cut $(S, V - S)$, there must be an edge on the path crosses the cut $(S, V - S)$. Let (x, y) be the edge.
Note that both (x, y) and (u, v) are edge crossing and (u, v) is the light edge.

Removing (x, y) breaks the tree T; adding (u, v) reconnects a tree T'. We have $T' = T - \{(x, y)\} \cup \{(u, v)\}$.

$$\begin{aligned} w(T') &= w(T) - w(x, y) + w(u, v) \\ &\leq w(T), \quad \text{since } (u, v) \text{ is light.} \end{aligned}$$

But T is minimum spanning tree, so $w(T) \leq w(T')$; thus T' must be a minimum spanning tree.
It remain to show that (u, v) is a safe edge for A. We have $A \subseteq T'$, since $A \subseteq T$ and $(x, y) \notin A$; $A \cup \{(u, v)\} \subseteq T'$. And, since T' is the minimum spanning tree, (u, v) is safe for A.

Corollary Let $G = (V, E)$ be a connected weighted undirected graph. Let A be a subset of E that is included in some minimum spanning tree of G, and let C be a connected component (tree) in the forest $G_A = (V, A)$. If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is safe for A.
proof The cut $(C, V - C)$ respect A, and (u, v) is a light edge for the cut.

Kruskal's Algorithm

- Given a weighted undirected graph $G = (V, E)$, preprocess the edges
- Maintain a set A that is a forest.
-
  - sort the edges according their weights.
  - put edges in a priority queue
- Iteratively do the following,
  - Take out the least weight edge, check if it is safe (form cycle?).
  - Include the edge if the edge is a safe
  - until a single tree is formed.

Prim's Algorithm

- Maintain a set A that is a single tree.
- Start with a tree having a single node s,
- choose the least-weight edge connecting the tree to a vertex not in the tree.
- until the a tree connecting all vertices.

Time Complexity

- Kruskal's Algorithm,
  - Presort or heap operation $O(E \log E)$,
  - For each edge, check if it is safe, $\alpha(V)$ (two FINDs). There are at most E edges.
  - Insert it into the tree and make two trees in the forest become one, $O(1)$, (a UNION, $n - 1 times$).
  - Total cost $O(E \log E) + O(E\alpha(V)$
- Prim's Algorithm
  - V elements stored in the Fibonacci Heap.
  - EXTRACT-MIN can be done in $O(\log n)$ amortized time.
  - DECREASE-KEY can be done in $O(1)$ amortized time. There are at most E times DECREASE-KEY.
  - Total cost is $O(E + V \lg V)$.

Breadth First Search

- One of the simplest algorithm.
- Dijkstra's single source shortest path algorithm and Prim's Minimum Spanning tree algorithm used similar ideas.
- Given $G = (V, E)$ and a distinguished source vertex s, bfs systematically explores the edges of G to discover every vertex reachable from s.

- Assume the graph is stored in an adjacency list $Adj[\ ]$.
- Each vertex has a color, WHITE, GRAY, BLACK. All vertex starts with WHITE. Once discovered, change to non-white. Need to distingush non-white to ensure the search is in a breadth first manner. Color of u stored in $Color[u]$
- BFS construct a BFT (breadth first tree). Initial contains a root s. Whenever a white vertex v is discovered while scanning the neighborhood of u, edge (u, v) is added to the tree. We say u the predecessor of v. Predecessor of u stored in $\pi[u]$.

- in BFT, the distance between u to the source s is stored in $d[u]$.
- BFS needs a first-in, first-out queue Q.

run through an example

```
BFS(G, s)
for each vertex u ∈ V[G] − {s}
   do color[u] ← WHITE
      d[u] ← ∞
      π[u] ← NIL
color[s] ← GRAY
d[s] ← 0
π[s] ← NIL
Q ← ∅
EnQUEUE(Q, s)
while Q ≠ ∅
   do u ← DEQUEUE(Q)
      for each v ∈ Adj[u]
         do if color[v] = WHITE
               then color[v] ← GRAY
                    d[v] ← d[u] + 1
                    π[v] ← u
                    ENQUEUE(Q, v)
      color[u] ← BLACK
```

run through an example.

Run Time

- Each vertex enqueued and dequeued at most once, the queue operations take $O(V)$ time.
- List in $Adj[u]$ is scaned when u is colored black. The length of the list is $O(E)$
- total time is $O(V + E)$

Shortest Path

- Define the shortest-path distance $\delta(s, v)$ from s to v the minimum number of edges in any path from s to v.
- A path of length $\delta(s, v)$ from s to v is said to be a shortest path from s to v.

Lemma Let $G = (V, E)$ be a directed or undirected graph, and $s \in V$ be an arbitrary vertex. Then, for any edge $(u, v) \in E$,

$$\delta(s, v) \leq \delta(s, u) + 1.$$

Proof If u is reachable from s, then so is v. The shortest path from s to v cannot be longer than the shortest path from s to v followed by the edge (u, v), i.e., $\delta(s, v) \leq \delta(s, u) + 1$.
If u is not reachable from s, then $\delta(s, u) = \infty$, and the inequlity holds.

To show that BFS properly computes $d[v] = \delta(s, v)$ for each vertex $v \in V$, we first show that $d[v]$ bounds $\delta(s, v)$.

Lemma Let $G = (V, E)$ be a directed or undirected graph, and suppose that bfs run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, the value $d[v]$ computed by BFS satisfies $d[v] \geq \delta(s, v)$.

Proof Induction on the number of ENQUEUE operations.
Inductive hypothesis is $d[v] \geq \delta(s, v)$ for all $v \in V$.
Basis, immediately after s is enqueued. Inductive hypothesis is true since $d[s] = 0 = \delta(s, s)$ and $d[v] = \infty \geq \delta(s, v)$ for all $v \in V - \{s\}$.

For inductive step, consider a white vertex v is discovered during the search from a vertex u. The inductive hypothesis implies that $d[u] \geq \delta(s, u)$. From the assignment performed by line 15 and from previous lemma, we have

$$\begin{aligned} d[v] &= d[u] + 1 \\ &\geq \delta(s, u) + 1 \\ &\geq \delta(s, v) \end{aligned}$$

Vertex v is then enqueued, and it is never enqueued again because it is GRAY. $d[v]$ never changes, inductive hypothesis is maintained.

To show $d[v] = \delta(s, v)$, we first show that at all times, there are at most two distinct d values in the queue.

Lemma During the execution of BFS on a graph $G = (V, E)$, the queue Q contains the vertices $< v_1, v_2, \dots, v_r >$, where v_1 is the head of Q and v_r is the tail. The $d[v_r] \leq d[v_1] + 1$ and $d[v_i] \leq d[v_{i+1}]$ for $i = 1, 2, \dots, r - 1$.

Proof Induction in the number of queue operations. Initially, when the queue contains only s, the lemma holds.

For the inductive step, we must prove that the lemma holds after both dequeuing and enqueuing a vertex.

Dequeue v_1 is dequeue and v_2 becomes the head. By inductive hypothesis, $d[v_1] \leq d[v_2]$ and $d[v_r] \leq d[v_1] + 1$, thus $d[v_r] \leq d[v_2] + 1$.

Enqueue v is enequeued in line 17, it becomes v_{r+1}. At this moment, the vertex u has been removed from the queue and we are scanning the adjacency list of u. By inductive hypothesis, the new head v_1 has $d[v_1] \geq d[u]$.
Thus $d[v_{r+1}] = d[v] = d[u] + 1 \leq d[v_1] + 1$.
We also have $d[v_r] \leq d[u] + 1$ and so
$d[v_r] \leq d[u] + 1 = d[v] = d[v_{r+1}]$.

Coroloary Suppose that vertices v_i and v_j are enqueued during the execution of BFS, and that v_i is enqueued before v_j. Then $d[v_i] \leq d[v_j]$ at the time that v_j is enqueued.

Theorem: Correctness of breadth-first search
Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$.
The BFS discovers every vertex $v \in V$ that is reachable from the source s, and upon termination, $d[v] = \delta(s, v)$ for all $v \in V$.
Moreover, for any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is a shortest from s to $\pi[v]$ followed by the edge $(\pi[v], v)$.

Proof Suppose that the theorem is not true. Some vertex receives a d value $\neq$ the shortest path distance.
Let v be the vertex with minimum $\delta(s, v)$ that receives such incorrect d value.
1. It is obvious $s \neq v$. 2. By previous lemma, $d[v] \geq \delta(s, v)$, we must have $d[v] > \delta(s, v)$. v must be reachable from s (otherwise $\delta(s, v) = \infty \geq d[v]$).
Let u be the vertex immediately preceding v on the shortest path from s to v. Then we have

$$\delta(s, v) = \delta(s, u) + 1$$

Now we have $\delta(s, u) < \delta(s, v)$; because how we choose v, we have $d[u] = \delta(s, u)$. Putting all these properties together, we have

$$d[v] > \delta(s, v) = \delta(s, u) + 1 = d[u] + 1.$$

Now look at the pseudo code. At the time BFS chooses to dequeue vertex u from Q in line 11. At this time, vertex v is either white, gray, or black. We show that in each of the cases, we can derive contradiction.

If v is white: Line 15 set $d[v] = d[u] + 1$, contradiction to the inequality.

If v is black, v was already removed from the queue, according to the corollary, $d[v] < d[u]$, contradiction to the inequality.

If v is gray, it was graied when w was dequeue. w was removed from Q earlier than u and $d[v] = d[w] + 1$. From the corollary, $d[w] < d[u]$, so we have $d[v] \leq d[u] + 1$, condicting the equation. Thus we conclude $d[v] = \delta(s, v)$ for all $v \in V$.
To conclude the proof, observe that $\pi[v] = u$, then $d[v] = d[u] + 1$. Thus we obtain a shortest path from s to v by taking the shortest path from s to $\pi[v]$ than follow the edge $(\pi[v], v)$ to v.

Breadth First Tree

For a graph $G = (V, E)$ with source s, we define the *predecesor subgraph* of G as $G_\pi = (V_\pi, E_\pi)$, where

$$V_\pi = \{v \in V : \pi[v] \neq \mathrm{NIL}\} \cup \{s\}$$

and

$$E_\pi = \{(\pi[v], v) : v \in V_\pi - \{s\}.$$

Predecessor subgraph is a breadth-first tree. The path from s to v is unique, and it is the shortest path. Edges in E_π are called the tree edges.

Depth-first search

- Strategy: to search "deeper" in the graph whenever possible.
- Edges are explored out of the most recently discovered vertex v that still has unexplored edges leaving it.
- When there is no way out from v, the search "backtrack" to the vertex from which v was discovered.
- Process continues until we have discovered all the vertices reachable from source.
- If any undiscovered vertex remain, then one of them is sellected as a new source.

- Vertex v is discovered while scanning the adjacency list of a discovered vertex u, v's predecessor field $\pi[v] = u$.
- DFS produces a predecessor subgraph of G, it is a forest.

$G_\pi = (V, E_\pi)$, where
$E_\pi = \{(\pi[v], v)\} : v \in V \text{ and } \pi[v] \neq \text{NIL}\}$.
Edges in E_π are called *tree edge*.

Vertices have color to indicate their states.

- Initially WHITE,
- Become GRAY when it is discovered,
- Balckened when it is finished, i.e., adjancency list has been examined completely.

DFS *timestamps* each vertices, each vertex has two timestamps.

- $d[v]$: the first timestamp, records when v is first discovered.
- $f[v]$: records when the search finishes examining v's adj. list.

Timestamps are ranged integers ranged from 1 to $2|V|$.
For every v, $d[v] < f[v]$.

DFS(G)

for each vertex $u \in V[G]$

  do $color[u] \leftarrow$ WHITE

    $\pi[u] \leftarrow$ NIL

$time \leftarrow 0$

for each vertex $u \in V[G]$

  do if $color[u] =$ WHITE

    then DFS-Visit(u)

DFS-Visit(u)

```
color[u] ← GRAY
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]
    do if color[v] = WHITE
        then π[v] ← u
            DFS-Visit(v)
color ← BLACK
f[u] ← time ← time + 1
```

run through an example

- Results depends on the order of vertices examined
- depends on what stored in the data structure (the *Adj* list)
- run time is $\Theta(V + E)$.

Properties of the DFS

- Predecessor subgraph G_π forms a forest.
- v is a decendant of u in the DFS forest iff v is discovered during the time in which u is gray.
- discovery and finishing time have parenthesis structure

Theorem: Parenthesis theorem
In any DFS of a (direct or undirected) graph $G = (V, E)$, for any two vertices u and v, exactly one of the following 3 conditions hold:

- the intervals $[d[u], f[u]]$ and $[d[v], f[v]]$ are entirely disjoint, and neither u nor v are decendant of the other in DFS tree.
- the intervals $[d[u], f[u]]$ is contained entirely within interval $[d[v], f[v]]$, and u is a decendant of v in DFS tree,
- the intervals $[d[v], f[v]]$ is contained entirely within interval $[d[u], f[u]]$, and v is a decendant of u in DFS tree.

Proof if $d[u] < d[v]$, there are two subcases depending on $d[v] < f[u]$ or not.
if $d[v] < f[u]$, v is discovered while u was gray. Thus v is a decendant of u. Furthermore, after all the outgoing edges of v are explored, the search return to u, $f[v] < d[v]$. We conclude $[d[v], f[v]]$ is entirely in the interval of $[d[u], f[u]]$.
The other case $f[u] < d[v]$. By the inequality $d[u] < f[u]$, we have the two intervals are disjoint. Thus neither vertices was discovered while the other was gray, and so neither vertex is a decendant of the other.
The other case that $d[v] < d[u]$ is similar.

Corollary: (Nesting of decendant's intervals)
v is proper decendant of u in DFS forest for a directed of undirected graph G iff
$d[u] < d[v] < f[v] < f[u]$.

Theorem: (White-path theorem)
In a DFS forest of $G = (V, E)$ v is a decendant of u iff at the time $d[u]$ that the search discovers u, vertex v can be reached from u along a path consisting entirely of white vertices.
Proof

Proof :
$\Rightarrow$ Assume v is a decendant of u. Let w be any vertex on the path between u and v. w is a decendant of u. By the corollary, $d[u] < d[w]$ so w is white at time $d[u]$.
$\Rightarrow$ Suppose that vertex v is reachable from u along a path of white vertex at time $d[u]$, but v does not become a decendant of u in DFT.
Without loss of generality, assume that every vertices along the path become a decendant of u, (otherwise, we can let v be the closest vertex to u along the path that does not become a decendant of u). Let w be the predecessor of v in the path, so that w is a decendant of u.
By Corollary, $f[w] \leq f[u]$.
Note that v must be discovered after u is discovered, but before w is finished. Therefore $d[u] < d[v] < f[w] \leq f[u]$. By previous theorem, $[d[v], f[v]]$ is conbtained entirely within the interval $[d[u], f[u]]$. By corollary, v myst be decendant of u.

We can define four edge types in terms of the depth-first forest G_π producedby a DFS on G

- *Tree edges*: Edges in the DF forest G_π.
- *Back edges*: Edge (u, v) connecting u to an ancestor v in DF forest. Self-loop, which may occur in directed graphs, are considered to be back edges.
- *Forward edges*: (u, v) are nontree edges connecting a vertex u to a decendant v in DF tree.
- *Cross edges*: All other edges.

- DFS can be modified to classify edges as it encounters them.
- Key idea: edge (u, v) can be classified by the color of the vertex v that is reached when the edge is first explored.
  - WHITE indicates tree edges
  - GRAY indicates a back edge
  - BLACK indicate foreard or cross edge

Theorem In a DFS of an undirected graph G, every edge of G is either a tree edge or a back edge.

Proof Let (u, v)be an arbotrary edge of G, and suppose without loss of generallity that $d[u] < d[v]$. Then v must be discovered and finished before we finieh u, since v is in u's adjancency list. If (u, v) is explored first in the direction from u to v, the v is discovered until that time, otherwise, we could have explored this edge already in the direction from v to u. Thus (u, v) become a tree edge. If (u, v) is explored first in th direction from v to u, then (u, v) is a back edge, since u is till gray at the time the edge is first explored.

Topological Sort

- Apply DFS to perform a topological sort of a *directed acylic graph*, (acyclic: no cycle) or *dag*.
- A topological sort of a dag $G = (V, E)$, a linear order of all vertices s.t. if G contains an edge (u, v), then u appears before v in the ordering.
- If the graph is not acyclis, no linear order is possible.

TOPOLOGICAL-SORT(G)

1. call DFS(G) to compute finishing time $f[v]$ for rach vertex v.
2. as each vertex is finished, insert it onto the front of the linked list.
3. return the linked list of vertices.

run through an example in pp 550