Last time: dynamic programming
Today: Elementary graph algorithm (textbook Chapter 22)

We express a graph as $G = (V, E)$ that means a set of vertices and edges

There are always two standard ways to represent a graph:

1. an adjacency matrix

\[
\begin{array}{ccc}
1 & 0 & 0 & 1 \\
2 & 0 & 0 & 1 \\
3 & 1 & 1 & 0 \\
\end{array}
\]

2. a collection of adjacency lists

They can store either directed or undirected graphs.

Next we will talk about two types of graph searching algorithms:

1. Breath-first algorithms
   - Simple strategy
   - Computes shortest paths from the start note
• Expands “frontier”, all nodes at distance k are expanded before any node at distance k+1 is expanded

Take the word puzzle example mentioned before: TRAIN → CRANE

We finish all the possibilities of this layer before go to next one

Example:

Steps of searching algorithm:

We need a queue (Q), 2 arrays (color, parent) to help not go back to node which has discovered to record predecessor of each node (at most one)

source = s

0) Place a in Q, a is discovered  Q:a
1) Take top node (here is a) in Q, go through its adjacent list  Q:a b c
   • Put in Q if doesn’t discovered
   • Color these as ‘d’
   Color top node as ‘c’ and remove top node from Q  Q:a b c
2) b is top node, remove  Q:b c
3) c is top node → find d → dequeued
4) Q:e (d is expanded)
5) Q:-(e is expanded)

Illustration:
color[a]= d

Q: a
Q: b c
Q: c
Q: d
Q: e
Search tree:

if node b go first:

if node c go first:

Complexity of the algorithm:
- enqueue/dequeue’s constant time
- every node expanded once
- \( O(|V|) \) to enqueue/dequeue and color expanded node
- \( O(|E|) \) to handle all neighbors during all expansions

\( \Rightarrow \) Total running time of BFS is \( O(|V|+|E|) \)

Example of applying BFS: Beam search(M-algorithm)

It goes in BFS fashion, only keeps the first best \( n^{th} \) results, it is much faster than dynamic programming.

2. Depth-first search

- gives priority to longer nodes
- only when all of v’s descendents are explored, do we explore some “shorter” node u
- we will have two kinds of nodes discovered (the nodes in search)
Pseudo code (contains two functions: MAIN and DFS_VISIT):

**DFS_VISIT(u)**
- color[u] ← discovered
- time ← time + 1
- dt[u] ← time
- for ∀v in u’s adjacent list
  - if color[v] == undiscovered
    - parent[v] = u
    - DFS_VISIT(v)
  - end
- end
- color[u] ← finished
- time ← time + 1
- dt[u] ← time

**MAIN**
- for ∀u ∈ V
  - color[u] ← undiscovered
  - parent[u] ← NULL
  - time ← 0
- for ∀u ∈ V
  - if color[u] == undiscovered
    - DFS_VISIT(u)

Illustration:
- we assume white color of notes as undiscovered; gray as discovered but not finished; black as finished

1/
```
  a ---- b
  |     |
  c ---- d ---- e
```

2/
```
  a      2/
  |      |
  c ---- d ---- e
```

1/
The search tree is:

If we get more complexity graph, add a node ‘f’, the procedure of the algorithm should be:
The complexity of DFS is obviously $O(|V|+|E|)$

There are several properties of DFS algorithm:

1. Since discovery and finishing time of each nodes have parenthesis structure, we can express search procedure as:
   
   \[
   [ \text{a} [ \text{b} [ \text{c} [ \text{d} [ \text{e} ] ] ] ] [ \text{f} ] ]
   \]
   
   it will help us to store the search result in tree

2. DFS could help us find loops in graph

   When the search goes to the 7th step, we will find b was unfinished but has discovered, then we know that there is a loop in this graph.

3. DFS could help us find forward edges

   When we goes to 8th step, we find a finished node c, so we can tell that there is a forward edge in this graph (a $\rightarrow$ c).

   On other priority we search c first, we also can come out that there exists a forward edge in this graph.
4, DFS can be used to perform Topological Sort:

We can first use Depth-first algorithm to search for the whole graph and come out one possible solution:

watch, shirt, tie, socks, undershorts, pants, shoes, belt, jacket