Data structure

Graph

Kim JaeHwan

Chapter 14. Graph Representations

Terminology

Graph $G = (V, E)$
V : a finite set of vertices/nodes
n : $|V|$, the number of vertices.
E : a finite set of edges/arcs (v, w) where $v, w \in V$
e : $|E|$, the number of edges.

Adjacent.

Type of graphs

- Complete
- Sparse < Dense
- Directed, Undirected
- weighted
- labeled

subgraph, induced subgraph.
Induced subgraph includes all the edges that have both endpoints in the inducing set V_s, whereas an ordinary subgarph may miss some.

Path, Cycle, Self-loop, Acylic, Connected graph, In a directed graph, strongly connected, weakly connected. connected components, SCC.
Tree, Spanning tree, minimum-cost spanning tree.

Graph representations : Adjacency matrix, Adjacency list(linked / array)

Chapter 15. Graph Traversals

Visiting the vertices of a graph in some specific order, DFS, BFS. There are some troublesome cases : If the graph is not connected, unreachable to all vertices. If the graph is contains cycles, infinite loop.Solution is maintainint a mark for each vertex.

DFS

DFS : Depth First Search, Going as deep as possible on each child before going to the next sibling.
Recursive implementation :

```
DFS (v) {
    mark[v] := visited;
    for(each unvisited vertex w adjacent from v)
        DFS(w);
}
```

Time complexity of DFS is $O(n + e)$ for adjacency list, $O(n^2)$ for adjacency matrix.
Applications of DFS:

1. Finding path between two vertices.
2. Identify connected components. (*)

Topological Sort (DFS-based, Stack-based):

```
TOPSORT(v) {
    mark[v] := visited;
    for (each unvisited vertex adjacent from v)
        TOPSORT(w);
    print(v)
}
}
```

I can't understand this code;;
Queue-based topological sort is possible, check the indegree of each vertex.

BFS

BFS : Breadth First Search, Going as broad as possible on each depth before going to the next depth
Queue-based implementation :

```
Visit/Mark a start vertex & put into the queue;
Repeat until the queue is empty {
    Remove a vertex from the queue;
    Visit/Mark its unvisited adjacent vertices;
    Put the newly visited vertices into the queue
}
```

Same time complexity as DFS, Same properties with respect to graph connectivity, connected components, spanning tree, path finding. There are problems for which BFS is more suitable than DFS and vice versa

Chapter 16. Shortest Path

Weighted digraph, (nonnegative cost $C[i,j]$)
Path langth / cost
Shortest path problem(SPP) : finding a path btw. two vertices such that the path length is minimized.

The type of shortest path problems :

- Single-pair SPP : Single source, single destination
  Greedy algorithm : making the locally optimal choice at each stage with the hope of finding a global optimum.
  Possible algorithm : DFS, BFS...
- Single-source SPP : Single source, all destinations
  Dijkstra's algorithm:
  variables : $D[i]$: distance from source to vertex i.

  ! Write a pseudocode with your own words. !

  Why Dijkstra's algorithm works?

  Complexity

  Further Questions : cyclic? negative?
  Could we do better for acyclic graph?

- Single-destination SPP : (not covered)
- All-pairs SPP : All sources, all destinations Floyd's algorithm : DP solution.
  $A_k[i,j] := min\{A_{k-1}[i,j], A_{k-1}[i,k] + A_{k-1}[k,j]\}$

  ! Write a pseudocode with your own words. !

  Complexity

Other shortest path problems :

- Bellman-Ford algorithm :
  Solves single-source SPP with negative edge weights.
  Time complexity : $O(n \cdot e)$, but slower than Dijkstra's algorithm.
- Johnson's algorithm :
  Solves all-pairs SPP with negative edge weights but no negative cycles.
  Time complexity : $O(n \cdot e + n^2 \cdot log(n))$, may be faster than Floyd's on sparse graph.
- A* algorithm :
  Solve single-pair SPP, using heuristics to speed up the search.

Note here with your hands.

Chapter 17. Minimum Spanning Tree

Spanning tree T of G is a tree that includes all the vertices of the original graph G. Minimum-cost spanning tree (MST) is a spanning tree whose tree cost is minimum. MST can be used in internet infrastructure, road networks, electronic circuit, water pipes, etc.
Possible Greedy Strategies: Prim's algorithm (a.k.a Prim-Jarnik algorithm), Kruskal's algorithm, Sollin's algorithm.

- Prims's algorithm :
  Start with a 1-vertex tree and grow it into an n-vertex tree by repeatedly adding a cheapest edge and vertex.
  Idea
  Algorithm
  Implement(code)
  Why it works?
  Time complexity

- Kruskal's algorithm :
  Start with an n-vertex forest, Consider edges in order of increasing edge cost, select edge if it does not form a cycle together with the edges already selected. (using union-find)
  Idea
  Union find and disjoint sets data type
  dynamic connectivity
  naive linking
  link by size
  path compression
  Complexity

- Sollin's algorithm :
  Start with an n-vertex forest, each components selects a least-cost edge to connect to another component. Eliminate duplicated selections and possible cycles, Repeat until only 1 component remains.
  Idea
  Algorithm
  Complexity

Euclidean MST
Scientific application :

Uniqueness of MST
Prim vs. Kruskal

Note here with your hands.