Preface

The goal of this book is to serve as a preparation guide for coding competitions. I should mention that this is a *personalized* reference book for myself and my team. As a consequence, more attention is given over to writing for Java and C++.

Some competition environments allow texts with a certain criterea. ICPC World Finals rules that a team may bring a **Team Reference Document**, a document containing up to 25 pages of reference materials, single-sided, letter or A4 size. This book does not follow the requisite, but we advise you to look at our appendix section.

We have included an two appendixes. Appendix 1 is devoted to common data structures we encountered using Java and C++. Appendix 2 includes common mathematical formulas for reference.

Contents

Chapter 1

Graphs

Graphs are a fundamental data structure in the world of programming. Formally a **graph** G consists of a finite nonempty set V of objects called **vertices** and a set E called **edges**. G is an ordered pair of sets V and E.

$$G = (V, E)$$

Graphs are often used to represent physical entities (a network of roads, relationship between people). Common graph problems include shortest paths, number of minimum cuts, strongly connected components ....

1.1 Breadth-First Search

"Cautionary search"

1.2 Depth-First Search

If BFS is the cautious and tentative exploration strategy, then **Depth-First Search** (DFS) is its more aggressive cousin. DFS explores aggressively, delving deeper into the graph and backtracks only when necessary.

The implementation of DFS is similar to BFS, but instead of using a queue, we use a stack. Another method is to use recursion.

Recursive Version:

```
void search(Node root){
  if(root == null) return;
  visit(root);
  root.visited = true;
  for each (Node n in root.adjacent) {
    if (n.visited == false) {
      search(n);
    }
  }
}
```

Running time: $O(V + E)$

Why use DFS over BFS? DFS has its own catalog of applications that can't be replicated using BFS.

- Computing a topological ordering of directed acyclic graphs[1]
- Computing strongly connected components of directed graphs

Topological Sort

Khoa is really interested in artificial intelligence, so much so that before he graduates, he wants to take the course offered by the computer science department, CS 4242 (A). However, before he can enroll in CS 4242, he must have completed its prerequisites. Help him construct an ordering which he can follow to take CS 4242.

The prerequisites for CS 4242 are CS 3304 Data Structures (D) and MATH 3332 Probability (P).
The prerequisite for CS 3304 is MATH 1190 Calculus (C).
The prerequisite for MATH 3332 is MATH 1190.

A **topological ordering** of a directed graph G is a labeling f of G's nodes such that:

1. the $f(v)$'s are the set $1, 2, \ldots, n$
2. $(u, v) \in G \Rightarrow f(u) < f(v)$

[1] Note: G has a directed cycle $\Longrightarrow$ No topological ordering

As long as our structure is a directed acyclic graph (DAG), we can compute a topological ordering. Every DAG has a sink vertex. Remove the sink vertex, and the remaining graph is a DAG. (Unless $|v| < 1$) The pseudocode for a topological sort is as follows:

```
DFS(graph G, start vertex s) {
  visit(s);
  for each edge (s,v) {
    if(v.visited == false)
      DFS(G, v);
  }
  set f(s) = current_label
  current_label--;
}

topologic_sort(graph G) {
  mark all nodes unexplored;
  current_label = n;
  for each vertex $v \in G$ {
    if(v.visited == false)
      DFS(G, v);
  }
}
```

Running time: $O(V + E)$

Proof. Take any edge, (u, v). We will show that $f(u) < f(v)$.

Case 1: u is visited by DFS before v
u will recursively call v, which will continue DFS. Eventually v will exhaust its adjacent neighbors and become a sink vertex before u. $f(u) < f(v)$

Case 2: v is visited by DFS before u
v makes recursive DFS calls. u will never be visited during these recursively calls, otherwise the graph would contain a directed cycle. As a result, v will become a sink vertex before u is visited. $f(u) < f(v)$ □

Strongly Connected Components

The **strongly connected components** of a directed graph G are the equivalence classes of the relation $u \sim v \Leftrightarrow \exists$ path(u, v) and

path(v, u)
Why DFS?
Kosaraju's Two Pass Algorithm

Theorem 1.2.1. *Can compute SCC in* $O(m + n)$

Index