Fundamental Algorithm Techniques
Problem Sets #7 & #8

Zhumakhanbet Akbota

1 Problem Set #7: Graph Play & Bron-Kerbosch

1.1 Graph Transpose

For directed graph G, transpose $\text{rev}(G)$ reverses all edges: $(u, v) \to (v, u)$. Algorithm: Iterate through all edges, add reverse. Time: $O(V + E)$.

1.2 Graph Inverse

For undirected graph G, inverse $\bar{G}$ has edges that don't exist in G. If G is dense, $\bar{G}$ is sparse, and vice versa. Relationship: $|E(G)| + |E(\bar{G})| = \frac{|V|(|V|-1)}{2}$.

1.3 Dual Graphs

Dual graph only defined for planar graphs. Requires planar embedding to define faces. Non-planar graphs (e.g., K_5) have no well-defined faces, hence no dual.

1.4 Bron-Kerbosch Algorithm

Finds all maximal cliques using backtracking:

- R: current clique
- P: candidate vertices
- X: excluded vertices

When $P = \emptyset$ and $X = \emptyset$, R is maximal.

2 Problem Set #8: SCC, Euler Tour, Topological Sort

2.1 Problem 1: Strongly Connected Components

1. Graph Reversal Algorithm:

1. Initialize empty adjacency list for each vertex
2. For each edge (u, v) in G, add (v, u) to $\text{rev}(G)$
3. Time: $O(V + E)$ - visit each vertex and edge once

2. SCC Graph is Acyclic: Proof by contradiction: If $\text{scc}(G)$ had a cycle, vertices in different SCCs would be mutually reachable, contradicting maximality of SCCs. Therefore $\text{scc}(G)$ is acyclic.

3. $\textbf{scc}(\textbf{rev}(G)) = \textbf{rev}(\textbf{scc}(G))$: Reversing edges preserves reachability relationships. If u can reach v in G, then v can reach u in $\text{rev}(G)$. Strong components remain the same, only their connections reverse.

4. Reachability in G vs $\textbf{scc}(G)$: u can reach v in G if and only if $S(u)$ can reach $S(v)$ in $\text{scc}(G)$. Proof: If u can reach v in G, path goes through SCCs, so $S(u)$ can reach $S(v)$. Conversely, if $S(u)$ can reach $S(v)$, there's a path in G from u to v.

2.2 Problem 2: Euler Tour

1. Necessary and Sufficient Condition: G has Euler tour if and only if $\deg_{\text{in}}(v) = \deg_{\text{out}}(v)$ for all $v \in V$.

Proof: ($\Rightarrow$) In Euler tour, each edge enters and leaves a vertex once. ($\Leftarrow$) Start from any vertex, follow edges. Condition ensures we can always continue and return to start, forming a cycle. Merge cycles to get Euler tour.

2. Algorithm (O(E) time):

1. Check condition: in-degree(v) = out-degree(v) for all v
2. Start from any vertex, find cycle using unused edges
3. If cycle found, merge with main tour
4. Repeat until all edges used

This is Hierholzer's algorithm.

2.3 Problem 3: Topological Sort

Given course dependencies: $\{A \to B, A \to C, B \to C, B \to D, C \to E, D \to E, D \to F, G \to F, G \to E\}$

Starting from A:

1. A has no prerequisites: add A
2. B, C depend on A: add B or C (e.g., B)
3. C, D available: add C or D (e.g., C)
4. D, E available: add D or E (e.g., D)
5. E, F available: add E or F (e.g., E)
6. F, G available: add F or G (e.g., F)
7. Add G

One possible order: $A \to B \to C \to D \to E \to F \to G$
Starting from G: $G \to F \to E$ (then $A \to B \to C \to D$): $G \to F \to E \to A \to B \to C \to D$
Multiple valid orderings exist due to independent courses.