Complexity Classes

Definition 1. Let's define the following:

1. Any problem that can be solved by using a polynomial time algorithm (i.e., time=$O(n^k)$, for some $k \in \mathbb{N}$) is called tractable. Otherwise, we call it intractable.

2. A decision problem is a problem whose answer is a yes or no.

3. An optimization problem is a problem whose solution is a minimization or maximization of certain quality.

Examples

Uniqueness Problem: Are all elements in S distinct? (D)

Are there two elements that are equal? (D)

Element Count: Find the element with the highest frequency. (O)

Coloring Problem: Is G k-colorable? (D)

Find the min k such that G is k-colorable. (O)

Clique Problem: Does G contain k-clique (complete subgraph of size k)? (D)

Find the maximum k such that G contains k-clique (the chromatic number of G, $\chi(G)$). (O)

Remark. If we can find an algorithm A that solves the decision-version then we can solve the optimization-version by doing a binary search with algorithm A.

Definition 2. A deterministic algorithm is an algorithm whose steps (and hence its output) are completely determined once its input is fixed.

1 Complexity Classes

1.1 The Class P

This class consists of all problems that can be solved by a poly-time deterministic algorithm such as sorting, searching, majority element, shortest paths, and 2-colorability. Notice that the 2-colorability is equivalent to checking if G is bipartite or if it contains no cycles of odd lengths.

Theorem 1. *The class P is closed under complement, i.e., $A \in P \iff A^c \in P$.*

1.2 The Class NP

This class consists of all problems that can be solved nondeterministically by using a poly-time deterministic algorithm. This is done via two phases:

Guessing Phase: There is a procedure for guessing possible solutions in polynomial time.

Verifying Phase: There is poly-time deterministic algorithm to verify whether the guess solution is correct or not.

Examples

3-Colorability problem: Is G 3-colorable?

Hamiltonian problem: Does the graph G contain a cycle that visit every vertex exactly once?

Traveling Salesman: Can you visit certain cities such that the total distance is k?

Vertex Cover: Find the largest subset $C \subset V$ of vertices such that for every edge (x, y) in the graph G, the subset C contains at least one of the vertices x and y. That is, the edge is either totally in C or it is incident to C.

Independent Set: Find the largest subset of non-adjacent vertices. Note that if C is cover then C^c is independent.

3-SAT: Given a boolean formula f in conjunctive normal form CNF where each clause consists of three literals, for example

$$f = (x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_3 \vee \bar{x_1} \vee \bar{x_2}) \wedge (x_2 \vee x_2 \vee \bar{x_1}) \, .$$

Is f satisfiable? Can you find values for the variables that make f true.

And many others...

Theorem 2.

$$P \subseteq NP$$

The question that stays unsolved up to the time of writing is whether $P = NP$? Nobody believe that the answer of this question is yes!

1.3 The Class NP-Complete

This class consists of all problems Π that satisfy the following:

1. $\Pi \in NP$
2. Π is NP-hard, that is, if $\widetilde{\Pi} \in NP$, then $\widetilde{\Pi} \propto_{\text{poly}} \Pi$, that is, $\widetilde{\Pi}$ can be reduced to Π in polynomial time. This means that there is a poly-time algorithm that transfer any instance of $\widetilde{\Pi}$ to an instance of Π.

NP-complete problems are the hardest problems in NP class and if one can solve any NP-complete problem then he also can solve all of the problems in NP class.

2 Dealing with NP-Completeness

There are many techniques to deal with NP-complete problems. True that we can't solve them in polynomial time, but we can solve them. One can do that by using the following design techniques:

1. Backtracking
2. Branch and bound
3. Approximation
4. Randomization

Backtracking and branch and bound are searching-based techniques.

3 Backtracking

One can use backtracking to solve the n-queens problem or the 3-colorability problem which we are going to study here.

3.1 3-Colorability

Given an undirected graph $G = (V, E)$, find a **legal coloring** such that no adjacent vertices have the same color. For simplicity, say the colors are numbered 1, 2, and 3. If $|V| = n$ then any coloring could be written as $C_1, C_2, \ldots, C_n$ where $C_i \in \{1, 2, 3\}$ is the color of the vertex i. Clearly, there exists 3^n possible colorings represented by a ternary tree called the search tree. Each path from the root to a leaf represents a coloring assignment. The question is how do we find the a legal coloring. We use backtracking as follows.

Algorithm 3-ColorRec

<u>input:</u> graph $G = (V, E)$

<u>output:</u> legal coloring $C[1..n]$

for $k \leftarrow 1$ to n **do**
    $C[k] \leftarrow 0$
end for
$flag \leftarrow false$
graphColor(1)
if $flag$ **then** output C
else output ”no solution”

Algorithm graphColor(k)

for $color \leftarrow 1$ to 3 **do**
    $C[k] \leftarrow color$
    if C is legal coloring **then**
        $flag \leftarrow true$; exit
    else if C is partial coloring **then**
        graphColor$(k + 1)$
    end if
end for

Here is also the iterative version of this algorithm.

Algorithm 3-ColorIter

input: graph $G = (V, E)$

output: legal coloring $C[1..n]$

```
for k ← 1 to n do
    C[k] ← 0
end for
flag ← false; k ← 1
while k ≥ 1 do
    while C[k] ≤ 2 do
        C[k] ← C[k] + 1
        if C is legal coloring then
            flag ← true
            exit (from the two while loops)
        else if C is partial coloring then
            k ← k + 1        % advance
        end if
    end while
    C[k] ← 0
    k ← k − 1        % backtrack
end while
if flag then output C
else output "no solution"
```

The worst-case running time of this backtracking algorithm is clearly $O(n3^n)$. Notice that at each step we need to check the legality of the coloring by checking the correct coloring of each vertex in $O(n)$ time.

Example

Let us use the backtracking algorithm to find a legal 3-coloring for the following graph (if such exists). We draw the search tree followed by the algorithm and we assume that the three colors are named R, G and B for red, green and blue.