NP-Completeness

Ekkapot Charoenwanit

SSE, TGGS

ekkapot.c@tggs.kmutnb.ac.th

Polynomial Time $\mathcal{O}(n^k)$ is Good.

- Most (if not all) problems we have studied so far can be solved efficiently
  - Searching
  - Sorting
  - Single-Source Shortest Path
  - All-Pair Shortest Path
  - Fractional Knapsack
  - etc.
- All of these problems can be solved in polynomial time wrt. the input length.
- Polynomial Time $\implies$ Easy (or Tractable)

Exponential Time $\Omega(c^n)$ is Bad.

The Sirindhorn International TGGS Thai-German Graduate School of Engineering

- Some problems are hard(er) to solve.
  - Tower of Hanoi
  - N-Chess
- Lowers bound on their running time have been shown to be exponential wrt. the input length.
- Exponential Time $\Longrightarrow$ Hard (or Intractable)

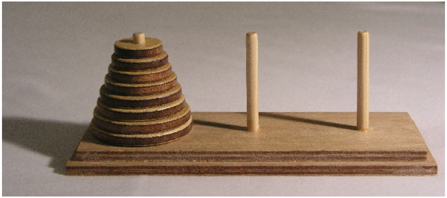


Figure: Tower of Hanoi[1]

[1] Image Courtesy of Wikipedia

Don't Worry about the Extremes !!!

What if k is large for $\mathcal{O}(n^k)$?

What about a problem with a polynomial runtime $\mathcal{O}(n^{1000000})$?

- Is it still considered efficient?

Don't be concerned too much with such extremes:

- Such extreme cases are extremely rare (if ever existing) in practice unless deliberately designed to be slow.
- Many problems could be solved with less efficient polynomial algorithms when they were first discovered than the currently best ones.

Bridging the Algorithmic Gap

Before merge sort was invented,

- we knew only algorithms like selection sort, insertion sort, all of which can sort in $\mathcal{O}(n^2)$ time

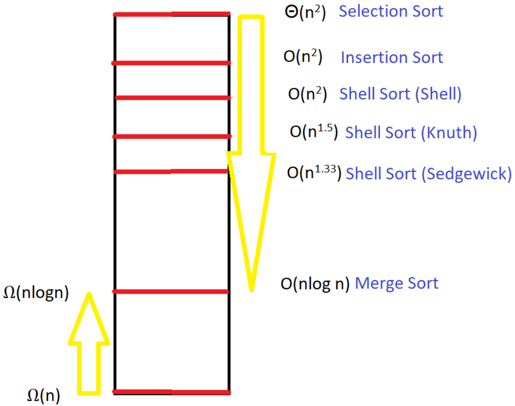
Later, it was proven that

- any comparison-based sorting algorithm needs at least $\Omega(n \log n)$ time.
- people then started to search for a sorting algorithm that needs at most $\mathcal{O}(n \log n)$.

When merge sort was discovered, which needs $\mathcal{O}(n \log n)$,

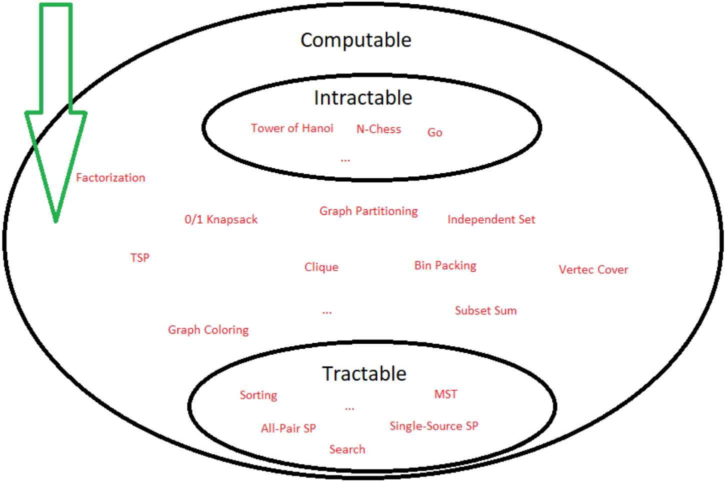
- the algorithmic gap for the sorting problem was closed
- we are certain that we cannot find any better algorithm than $\mathcal{O}(n \log n)$ because of the lower bound $\Omega(n \log n)$

Bridging the Algorithmic Gap



Classes of Problems

What class(es) are these problems in?



What class(es) do those problems in the middle belong to?

- They are problems whose polynomial-time algorithms have not been yet discovered.
- However, we still cannot prove that there exist lower bounds that are exponential.
- That is, their algorithmic gaps are still large.

So are they considered tractable or intractable?

- No one knows the answer yet.
- But, most theoretical computer scientists believe that they are intractable and cannot be solved in polynomial time.
- These problems are the topic of our discussion today.

Decision Problems

Decision problem

- X is a set of strings
- Instance: string s
- Algorithm A solves X: $A(s) = yes \iff s \in X$

Every optimization problem can be turned into a corresponding decision problem.

- Shortest-Path-Opt: Find a shortest path between u and v in G.
- Shortest-Path-Dec: Is there a shortest path between u and v of length at most k in G?

Optimization Problems $\Longrightarrow$ Decision Problems

By introducing an integer k, an optimization problem can be cast into a corresponding decision problem.

- Minimization Problem $\Longrightarrow$ at most k
- Maximization Problem $\Longrightarrow$ at least k

Optimization Problems $\Longrightarrow$ Decision Problems : Minimization Problem

Travelling Salesman (Minimization Problem):

- TSP-Opt: Given a directed graph $G = (V, E)$, find a shortest tour that visits each vertex in V exactly once except for the first one.
- TSP-Dec: Given a directed graph $G = (V, E)$, is there a tour with length at most k that visits each vertex in V exactly once except for the first vertex?

Optimization Problems $\Longrightarrow$ Decision Problems : Maximization Problem

Independent Set (Maximization Problem):

- IS-Opt: Given an undirected graph $G = (V, E)$, find a largest possible subset $W \subseteq V$ s.t. no pair of vertices in W is connected by an edge.
- IS-Dec:Given an undirected graph $G = (V, E)$, is there a subset $W \subseteq V$ of size at least k s.t. no pair of vertices in W is connected by an edge?

Hardness : Decision Problems VS. Optimization Counterparts

Thai-German Graduate School of Engineering

Let Q_{DEC} be the decision problem of an optimization problem Q_{OPT}.

It is obvious that

- Q_{DEC} is hard $\implies$ Q_{OPT} is also hard.

By the contrapositive,

- Q_{OPT} is easy $\implies$ Q_{DEC} is also easy.

This result suggests that

- we can efficiently solve Q_{DEC} by
  1. efficiently solving Q_{OPT}
  2. comparing the output of Q_{OPT} and the value k of Q_{DEC}.

But But But, some Q_{DEC} can be used to solve their corresponding Q_{OPT}.

- Caveat: not always the case for any pair of (Q_{DEC}, Q_{OPT}).

Graph Colouring:

- GC-OPT:Given an undirected graph $G = (V, E)$, find the minimum number of colours that can be assigned to the vertices s.t. no two adjacent vertices are of the same colour.
- GC-DEC:Given an undirected graph $G = (V, E)$, is there a colour assignment to the vertices using at most k colours s.t. no two adjacent vertices are of the same colour?

Graph Colouring: Using GC_{OPT} to solve GC_{DEC}

Use GC_{OPT} to solve GC_{DEC}:

$GC_{DEC}(G, k)\{$

  $c = GC_{OPT}(G)$

  return $c <= k$

$\}$

Use GC_{DEC} to solve GC_{OPT}:

```
GC_OPT(G = (V, E)){
    for (k = 1 : k <= |V| : k + +) do
      if (GC_DEC(G, k)) then
            return k
      end if
    end for
}
```

GC_{DEC} can solve GC_{OPT} by repeatedly asking $GC_{DEC}(G, k)$ outputs a yes for different values of k.

- Q1: Is there a better algorithm Q_{OPT} than this one?
- Hint: Divide and Conquer

Problem Classification

The P Class

The problems in the P Class are decision problems that can be solved (decided) in polynomial time.

- In other words, given a problem instance, they output yes or no in polynomial time.

Examples of the problems in P includes:

- Minimum-Spanning-Tree: Given an undirected graph G, is there a spanning tree having a total length of at most k?
- Longest-Common-Subsequent: Given two strings X and Y, is there a common substring of both X and Y having a length of at least k?
- Majority: Given an array A of length N, if there is a majority s.t. the majority appears in A at least $\frac{N}{2} + 1$ times?
- Primality-Test***: Given an integer N, determine if N is a prime number.

The NP Class

The problems in the NP Class are decision problems for which any yes can be verified in polynomial time.

- In other words, given a yes instance together with a certificate, they can verify it in polynomial time.
- The encoding length of a certificate must be at most polynomial in length wrt. the encoding length of the given instance.

Majority $\in$ NP?

We know that Majority $\in$ P

- there exists a polynomial-time algorithm that can decide in $\mathcal{O}(N^2)$ time.

```
MAJ_DEC(A[1...N]){
    for (i = 1 : i <= N : i++) do
        c = count(A, A[i])
        if (c > N/2) then
                return true
        end if
    end for
    return false
}
```

How can we show Majority $\in$ NP?

MAJ^{YES}_{CER} : Majority Yes-Certifier

- Instance: $A[1...N]$
- Yes-Certificate: maj
- Runtime: $\Theta(N)$

$MAJ^{YES}_{CER}(A[1...N], maj)\{$
    $c = 0$
    for $(i = 1 : i <= N : i++)$ **do**
      if $(A[i] == maj)$ **then**
          $c = c + 1$
      end if
    end for
    return $c > (N/2)$
$\}$

There exists a polynomial-time algorithm that can verify in $\Theta(N)$ time that the array A of length N has a majority by giving it the majority maj as a certificate.

$MAJ_{DEC} \in NP$

As we have found MAJ_{CER}^{YES}, which is a polynomial-time certifier, we can now conclude that

- $MAJ_{DEC} \in NP$

Actually, since we know $MAJ_{DEC} \in P$, we need not have looked for such a polynomial-time certifier MAJ_{DEC} to prove that it is in NP.

- Q2: Why?
- Q3: Show that $P \subseteq NP$.

GC^{YES}_{CER} : Graph-Coloring Yes-Certifier

- Instance : $G[1...N]G[1...N]$ and k
- Yes-Certificate : $color[1...N]$
- Runtime : $\mathcal{O}(N^2)$

$GC^{YES}_{CER}(G[1...N][1...N], k, color[1...N])\{$
    $nColors = max(color)$
    if $(nColors > k)$ **then**
        return *false*
    end if
    for $(i = 1 : i <= N : i++)$ **do**
        for $(j = i + 1 : j <= N : j++)$ **do**
            if $(G[i][j] == true$ **and** $color[i] == color[j])$ **then**
                return *false*
            end if
        end for
    end for
    return *true*
$\}$

$GC_{DEC} \in NP$

Since there exists a polynomial-time algorithm GC^{YES}_{CER} that can verify a yes instance,

- $GC_{DEC} \in NP$

Excercise:

- *Q4*: Prove the Subset-Sum Problem $\in$ NP.

The co-NP Class

The problems in the co-NP Class are decision problems for which any no can be verified in polynomial time.

- In other words, given a no instance together with a certificate, they can verify it in polynomial time.
- The encoding length of a certificate must be at most polynomial in length wrt. the encoding length of the given instance.

Note that the co- prefix stands for complementary in the sense that it is the complementary problem class to the NP class.

Primality Testing (Prime)

Provided a certificate c, $Prime^{NO}_{CER}$ verifies a given no instance N in polynomial time.

- Instance: N
- No-Certificate: c
- Runtime: $\Theta((\log n)^2)$

$Prime^{NO}_{CER}(N, c)\{$
    return $(N \mod c) == 0$
$\}$

Therefore, Prime $\in$ co-NP.

Actually,

- Prime is also in NP.
- But, a yes-certificate (called Pratt certificate) is quite tricky to find.

NP-Complete

Reduction

That Q_i is polynomially reducible to Q_j is denoted by

- $Q_i \leq_p Q_j$.
- all instances of Q_i are transformed to corresponding instances of Q_j.

$Q_i \leq_p Q_j$ is equivalent to saying:

- Q_i is no harder than Q_j.
- Q_j is at least as hard as Q_i.
- If Q_j is efficiently solvable, so is Q_i.

Reduction : SQR $\leq_p$ MULT

- Instances of SQR: x
- Instances of MULT: a , b
- Instance Transformation: $a = x, b = x$

$SQR(x)\{$
    $a = x, b = x$
    return $MULT(a, b)$
$\}$

We have just shown that SQR $\leq_p$ MULT.

- SQR is no harder than MULT.

Reduction : MULT $\leq_p$ SQR

- Instances of MULT: a, b
- Instances of SQR: x , y
- Instance Transformation: $x = a + b, y = a - b$

$MULT(a, b)\{$
    $x = a + b, y = a - b$
    return $(SQR(x) - SQR(y))/4$
$\}$

We have just shown that MULT $\leq_p$ SQR.

- MULT is no harder than SQR.

MULT $\leq_p$ SQR and SQR $\leq_p$ MULT

Since MULT $\leq_p$ SQR and SQR $\leq_p$ MULT,

- MULT and SQR are equally hard (also equally easy).
- Their hardness levels are the same.

What reduction can tell?

Suppose $A \leq_p B$.

- If B has a polynomial time algorithm, so does A.
- If B is easy, so is A.
- If A is hard, so is B.
- A is no harder than B.

If we can polynomially reduce any pair of problems A and B to each other,

- A and B are equally hard.

What reduction can tell?

Suppose $A \leq_p B$.

- If B has a polynomial time algorithm, so does A.
- If B is easy, so is A.
- If A is hard, so is B.
- A is no harder than B.

If we can polynomially reduce any pair of problems A and B to each other,

- A and B are equally hard.

the NPC Class

Cook-Levin Theorem shows that *SAT* is *NP-Complete*:

That is, all problems in *NP* can be reduced to *SAT*.

- $\forall q \in NP : q \leq_p SAT$

What does it mean for a problem *A* to be *NP-Complete*?

- *A* is among the hardest problems in *NP*.
- If we know how to solve *A* in polynomial time, we also know how to solve all problems in NP in polynomial time.
- That is, we would be able to solve hard problems such as Vertex Cover, Independent Set etc in polynomial time as well.
- One immediate result is that $P = NP$.

But, until now, we have not discovered any polynomial-time algorithm for any problems in *NP*.

So it still remains a mystery whether $P = NP$.

Independent Set (IS) $\in NPC$: $SAT \leq_p IS$

Vertex Cover (VC) $\in NPC : IS \leq_p VC$

Clique $\in$ *NPC* : $IS \leq_p$ *Clique*

Graph Colouring: Solution to Q1

Use GC_{DEC} to solve GC_{OPT}:

$GC_{OPT}(G = (V, E))\{$
  $low = 1, high = |V|, k = high$
  while $low <= high$ **do**
    $mid = (low + high)/2$
    if $GC_{DEC}(G, mid)$ **then**
      if $mid < k$ **then**
        $k = mid$
      end if
      $high = mid - 1$
    else
      $low = mid + 1$
    end if
  end while
  return k
$\}$