(1) Given an undirected graph $G(V, E)$, output the cardinality of an independent set $I \subseteq V$ of maximum cardinality.

- Observation: If I is a maximum indepdent set and a vertex v is not in I, then at least one of the neighbors of v is in I.

  In other words, for every vertex v and any maximum independent set I, either v or at least one the neighbors of v is guaranteed to belong to I.

- $IndSetCard(G)$:

  1. if $|V| = 0$ return 0
  2. choose a vertex v in G
  3. return $1 + \max\{IndSetCard(G \backslash N[y] : y \in N[v])\}$ //$N[v]$ is the set comprising v and all the vertices adjacent to v

- The correctness is due to above observation. Since both including and excluding every vertex in $N[v]$ is considered, for every maximum independent set I that has a subset V' of vertices in $N[v]$, algorithm considers including V' into I. Hence, this algorithm indeed does an exhaustive search, but without signficantly blowing up the exponent in time as we prove below.

- Analysis: let $T(n)$ be the number of invocations of IndSetCard when there are n nodes in G

  then, $T(n) \leq 1 + \sum_{v_i \in N[v]} T(n - d(v_i) - 1)$,

  instead of choosing an arbitrary vertex v, choosing a vertex v of minimum degree in step2 of $IndSetCard$, leads to $T(n) \leq 1 + \sum_{i=1}^{d(v)+1} T(n - d(v) - 1)$

  unraveling the recurrence yields, $T(n) = O^*((d(v) + 1)^{n/(d(v)+1)})$, which is $O^*(3^{n/3})$. [1] [2]

  Though this is an exponential time algorithm, this algorithm's time complexity is improved from $O^*(2^n)$ time of a naive exponential-time algorithm for this problem.

(2) Given a graph $G(V, E)$ and an integer k, find a vertex cover of size at most k, if it exists. Otherwise, output that G does not have a vertex cover of size k.

- $ApplyTwoRules(G, k)$:

[1] $O^*()$ hides polynomial factors in $O()$

[2] for $s = (d(v) + 1)$, $t = s^{n/s}$ reaches maximum when $s = 3$: $\ln t = (n/s) \ln s$, leading to $\frac{1}{t}\frac{dt}{ds} = \frac{n}{s^2}(-\ln s + 1)$; $\frac{dt}{ds} = 0 \Rightarrow \ln s = 1 \Rightarrow s = e$; and, s must be an integer

Repeatedly apply these two rules one after the other, terminate when neither reduces the number of vertices in graph

(a) If the current graph G' contains an isolated vertex v, then delete v from G'.

(b) If there is a vertex v of degree $\geq k+1$ in the current graph G', then delete v from G', and decrement the current value of k by 1. (That is, include v in VC.)

- Let (G'', k'') be the instance left upon termination of $ApplyTwoRules$. Then, G'' has a vertex cover of size at most k'' iff G has a vertex cover of size at most k.
- If G'' has a vertex cover of size at most k'' then $|V(G'')| \leq k^2 + k$ and $|E(G'')| \leq k^2$.
  - due to following reasons: (i) VC of G'' has size at most k'', (ii) $k'' \leq k$, (iii) each node in that VC has degree at most k, and (iv) there are no isolated vertices in G''
- Corollary: If $|V(G'')| > k^2 + k$ or $|E(G'')| > k^2$ then the input graph G has no vertex cover of size k.
- Algorithm:
  1. $G'' \leftarrow ApplyTwoRules(G, k)$
  2. if $|V(G'')| > k^2 + k$ or $|E(G'')| > k^2$ then
     i. output "G does not have a VC of size k" and return
  3. if G'' has a vertex cover VC'' of size k'' then //compute naively
     i. return VC'' together with all the vertices chosen in rule (b) of $ApplyTwoRules$ as the vertex cover of G
  4. else
     i. output "G does not have a VC of size k"
- Analysis: takes $O^*(2^{k^2+k})$ time; as compared to $O^*(n^k)$-time naive algorithm, exponential dependance on k has been moved out of the exponent on n and into a separate function

  For an input of size n, a parameter k, and a constant c independent of n and k, an algorithm with running time $f(k) \cdot n^c$ is called a *parameterized* (a.k.a., *fixed-parameter tractable (FPT)*) algorithm.

(3) Yet another algorithm for cardinality vertex cover that uses *bounded search trees*:

- If the maximum degree of G is d and there is a vertex cover of size at most k, then G has at most kd edges.

  A consequence: Since $d \leq n - 1$, if G has a vertex cover of size at most k, the G has at most $k(n-1)$ edges.

  If G contains more than $k(n-1)$ edges, then G has no vertex cover of size at most k.

- For any edge $e(u, v) \in G$, graph G has a vertex cover S of size at most k iff at least one of $G - \{u\}$ and $G - \{v\}$ has a vertex cover of size at most $k - 1$.
  - for a vertex cover S, S contains at least one of u or v; w.l.o.g., it contains u, the set $S - \{u\}$ must cover all edges in $G - u$; therefore $S - \{u\}$ is a vertex cover of size at most $k - 1$ for $G - u$
  - conversely, if one of $G - u$ (or $G - v$) has a vertex cover T of size at most $k - 1$, then $T \cup \{u\}$ covers all edges in G
- Algorithm:
  1. If G contains no edges, then the empty set is a vertex cover
  2. If G contains $> k(n - 1)$ edges, then it has no k-node vertex cover
  3. Else for any edge $e(u, v) \in G$
     (i) recursively check if either of $G - \{u\}$ or $G - \{v\}$ has a vertex cover T of size $k - 1$
     (ii) if neither does, G has no k-node vertex cover
     (iii) else output $T \cup \{u\}$ (resp. $T \cup \{v\}$) if $G - \{u\}$ (resp. $G - \{v\}$) has a vertex cover of size $k - 1$
- Analysis: Let $T(n, k)$ denote the running time of this algorithm. Then,
  - $T(n, k) \leq 2T(n - 1, k - 1) + O(k(m + n))$
    $T(n, 1) \leq O(n)$
  - Solving this recurrence, yields $T(n, k)$ is $O(2^k k(m + n))$.

  note the improvement in $f(k)$ from $O^*(2^{k^2+k})$ time in the previous algorithm to $O(2^k k(m + n))$