问题求解（四）习题

助教：宋锦华

第一周作业

- **Exercise 2.3.1.7. Design a representation of graphs by words over $\Sigma_{\mathbf{bool}}$.**

Let $a_i \in \Sigma_{bool}^n, i = 1, \ldots, n$, then a graph of n vertices can be represented as $a_1 a_2 \ldots a_n$. Or code {0, 1, #} with words of fixed length.

- **Exercise 2.3.1.8. Design a representation of weighted graphs, where weights are some positive integers, using the alphabet $\{\mathbf{0}, \mathbf{1}, \#\}$.**

The set of integers is $U = \{0,1\}^+$ (the length is not fixed!). If $M_G = [a_{ij}]_{i,j=1,\ldots,n}$ is an adjacency matrix of a graph G of n vertices, a representation of weighted graphs is

$$a_{11}\#a_{12}\#\ldots\#a_{1n}\#\#a_{21}\#a_{22}\#\ldots\#a_{2n}\#\#\ldots\#\#a_{n1}\#a_{n2}\#\ldots\#a_{nn},$$

where $a_{ij} \in U$.

- **Exercise 2.3.3.8. Describe a polynomial-time verifier for**
- **(i) HC,**
- **(ii) VC, and**
- **(iii) CLIQUE.**

(i) A: Input: $(x, c) \in \{0,1,\#\}^* \times \Sigma_{bool}^*$
(1) x is a representation of a graph and we represent the set of vertices as $\{x_1, x_2, \ldots, x_n\}$. c codes a sequence of vertices: $I = \{x_{i1}, x_{i2}, \ldots, x_{in}, x_{i1}\}$.
(2) If each pair of adjacent vertices have an edge, then A accepts (x, c) otherwise A rejects.

(ii) A: Input: $(x, c) \in \{0,1,\#\}^{+} \times \Sigma_{bool}^{*}$

(1) $x = u\#w \in \{0,1,\#\}^{+}$ where $u \in \{0,1\}^{+}$ and w represents a graph with n vertices. Let $k = Number(u)$. $c \in \Sigma_{bool}^{n}$ codes k vertices using k symbols of 1.

(2) If each edge of the graph is incident to at least one vertex in c, then A accepts (x, c), otherwise A rejects.

(iii) A: Input: $(x, c) \in \{0,1,\#\}^{+} \times \Sigma_{bool}^{*}$

(1) $x = u\#w \in \{0,1,\#\}^{+}$ where $u \in \{0,1\}^{*}$ and w represents a graph. Let $k = Number(u)$. c codes k vertices.

(2) If each pair of the k vertices has an edge, then A accepts (x, c), otherwise A rejects.

第二周作业

- ***34.1-2*** **Give a formal definition for the problem of finding the longest simple cycle in an undirected graph. Give a related decision problem. Give the language corresponding to the decision problem.**

（1）Input: an undirected graph G

Output: the longest simple cycle

（2）Input: an undirected graph G and an integer k,

Output: “yes” if exists a simple cycle of length at least k in G; “no” otherwise

（3）Longest-simple-cycle= $\{< G, u, v, k >$:

$G = (V, E)$ is an undirected graph, $u, v \in V$,

$k \geq 0$, is an integer, and there exists a path

from u to v in G consisting of at least k edges$\}$.

- ***34.1-3*** **Give a formal encoding of directed graphs as binary strings using an adjacency matrix representation. Do the same using an adjacency-list representation. Argue that the two representations are polynomially related.**

（1） Adjacency matrix representation:

Coding: a_{ij} = no edge $\Rightarrow 00, i \rightarrow j \Rightarrow 01$ and $\# \Rightarrow 11$

$G = [a_{ij}]_{i,j=1,..,n}: a_{11}a_{12} \dots a_{1n}\#a_{21}a_{22} \dots a_{2n}\# \dots \#a_{n1}a_{n2} \dots a_{nn}$

（2） Adjacency-list representation:

Coding: $b_{ij} \in \{0,1\}^m$ represents a vertex adjacent to vertex i and $c_i \in \{0,1\}^k$ is an integer representing the number of adjacent vertices

$G = c_1 b_{11} \dots b_{c_1 1}\#\dots\#c_n b_{n1} \dots b_{c_n 1}$

（3） Provide 2 polynomial-time transform functions (not prove the space complexity)

- ***34.1-5*** **Show that if an algorithm makes at most a constant number of calls to polynomial-time subroutines and performs an additional amount of work that also takes polynomial time, then it runs in polynomial time. Also show that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.**

(1) Induction. $n = k + 1,\ O(n^k) + O(dn^l) = O(n^k)$.

(2) A counter-example: Consider an algorithm that calls $O(n)$ subroutines each taking linear time. The first call can produce $O(n)$ output which can be concatenated to the original input and used as input to the next giving it time $O(2n)$. The total time used is then $\sum_{k=1}^{n} 2^k n$.

- ***34.2-3*** **Show that if HAM-CYCLE∈P, then the problem of listing the vertices of a hamiltonian cycle, in order, is polynomial-time solvable.**

Solution 1: Pick an edge e and test whether $G' = (V, (E - e))$ still contains a hamilton cycle. Recursively apply the procedure until no edge can be deleted. This can be done in polynomial time by trying all possible edges.
Solution 2: Pick a node $v \in V$ and let E_v be the edges incident to v. Compute a pair $e_1, e_2 \in E_v$ such that $G' = (V, (E - E_v) \cup (e_1, e_2))$ contains a hamilton cycle. This can be done in polynomial time by trying all possible pairs. Recursively apply the procedure on another node w for the graph G'.

A constant number of calls to polynomial time HAM-CYCLE

- ***34.2-4*** **Prove that the class NP of languages is closed under union, intersection, concatenation, and Kleene star. Discuss the closure of NP under complement.**

Let $L_1, L_2 \in NP$, then there exist two-input polynomial-time algorithms A_1, A_2 that can verify language L_1, L_2 and the certificates satisfy $|y_1| = O(|x|^{c_1})$,$|y_2| = O(|x|^{c_2})$

(1) Union: $A(x, y) = A_1(x, y) \vee A_2(x, y)$

(2) Intersection: $A(x, y) = A1(x, y_1) \wedge A2(x, y_2)$ where $y = y_1 y_2$

(3) Concatenation: $A(x, y) = A_1(x_1, y_1) \wedge A2(x_2, y_2)$ where $x = x_1 x_2$ and $y = y_1 y_2$

(4) Kleene star: $L_0 = \emptyset, L_1 = L, L_i = L_{i-2} L_{i-1}, L^* = \bigcup_n L_i$. Use induction.

(5) Discuss.

- ***34.2-6*** **A *hamiltonian path* in a graph is a simple path that visits every vertex exactly once. Show that the language HAM-PATH** $= \{< G, u, v >$**: there is a Hamiltonian path from** u **to** v **in graph** $G\}$ **belongs to NP.**

Verifier. A sequence of vertices: $\{x_1, x_2, \ldots, x_n\}$.

- ***34.2-11*** **Let G be a connected, undirected graph with at least 3 vertices, and let G^3 be the graph obtained by connecting all pairs of vertices that are connected by a path in G of length at most 3. Prove that G^3 is hamiltonian. (*Hint:* Construct a spanning tree for G, and use an inductive argument.)**

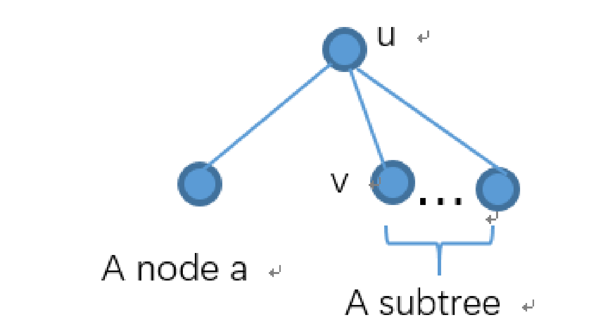
Stronger proposal: G^3 has a Hamiltonian path from $\boldsymbol{u}$ to $\boldsymbol{v}$ where $\boldsymbol{u}$ is a root node of any SPAN-TREE(G) and $d(u,v) = 1$ in G. This stands for every spanning tree.

(1) $n \leq 4$: trivial

(2) If $n = k$, hypothesis does hold.

(3) When $n = k + 1$, for each spanning tree, we can adjust it to this tree:

If delete a, the graph which has k vertices has a Hamiltonian cycle $ua_1a_2 \dots a_kv$. Then the Hamiltonian cycle of G^3 is $uava_ka_{k-1} \dots a_1u$.



- ***34.3-2*** **Show that the $\leq_P$ relation is a transitive relation on languages. That is, show that if $L1 \leq_P L2$ and $L2 \leq_P L3$, then $L1 \leq_P L3$.**

There exists polynomial time computable reduction functions f_1, f_2:
$x \in L_1 \Leftrightarrow f_1(x) \in L_2$ and $x \in L_2 \Leftrightarrow f_2(x) \in L_3$
Function $g(x) = f_2(f_1(x))$ is polynomial time computable and $x \in L_1 \Leftrightarrow g(x) \in L_3$

- **34.4-3 Professor Jagger proposes to show that SAT $\leq_p$ 3-CNF-SAT by using only the truth-table technique in the proof of Theorem 34.10, and not the other steps….. Show that this strategy does not yield a polynomial-time reduction.**

To form a truth table, there needs 2^n rows, where n is the number of variables.

- ***34.4-5*** **Show that the problem of determining the satisfiability of Boolean formulas in disjunctive normal form is polynomial-time solvable.**

  (1) If a formula is given in disjunctive normal form, we can simply check if any of the AND clauses can be satisfied to determine if the entire formula can be satisfied.
  (2) A conjunctive clause is satisfiable iff there does not exist both a variable and its negation.
  (3) polynomial time

- ***34.4-7*** **Let 2-CNF-SAT be the set of satisfiable boolean formulas in CNF with exactly 2 literals per clause. Show that 2-CNF-SAT∈P. Make your algorithm as efficient as possible. (*Hint:* Observe that $x \vee y$ is equivalent to $\neg x \rightarrow y$. Reduce 2-CNF-SAT to an efficiently solvable problem on a directed graph.)**

(1) A directed graph $G = (V, E)$ is defined as:

$V = \{v_1, v_2, \dots, v_n, \overline{v_1}, \overline{v_2}, \dots, \overline{v_n}\}$ where v_i and $\overline{v_i}$ corresponds to variable x_i and $\neg x_i$.

E: as $x \vee y \Leftrightarrow \neg x \rightarrow y \Leftrightarrow \neg y \rightarrow x$, construct two edges for each clause

Since CNF, SAT $\Leftrightarrow$ every edge should be satisfiable.

(2) This formula is satisfied if and only if no pair of complimentary literals are in the same strongly connected component of G. (only a path is wrong, see I)

I. If there are paths from v to u and vice versa, then in any truth assignment the corresponding literals must have the same value since a path is a chain of implications. Thus, a pair of complimentary literals is not satisfied.

II. Conversely, if no pair of complementary literals are in the same strongly connected component. (prove satisfiability)

- Consider the dag obtained by contracting each strongly connected component to a single vertex. This dag induces a total order using topological sort.
- For each x_i, if the component of v_i precedes the component of $\overline{v}_i$, set $x_i = 0$ else set $x_i = 1$.
- This is a valid truth assignment, i.e., that (i) all literals in the same component are assigned the same values and (ii) if a component B is reachable from A, then A, B cannot be assigned 1, 0. (prove $1 \rightarrow 0$ dose not exist)

(3) polynomial time

- ***34.5-6* Show that the hamiltonian-path problem is NP-complete.**

4 steps!

(1) By exercise 34:2 - 6 the hamiltonian-path problem is in NP.

(2) To show that the problem is NP-hard construct a reduction from the hamilton-cycle problem. $HC \leq_{\boldsymbol{p}} HP$

- Given a graph G pick any vertex v and make a copy of v, u that is connected to the same vertices as v. this graph has a hamiltonian path from v to u.
- others

(3) $x \in HC \Leftrightarrow f(x) \in HP.$

(4) polynomial time

第三周作业

- **Exercise 3.3.2.7. Combine Algorithm 3.3.2.4 and the above divide-and-conquer algorithm to design a faster algorithm for VC than the presented ones.**

Algorithm 3.3.2.4. Input: (G, k), where $G = (V, E)$ is a graph and k is a positive integer.

Step 1: Let H contain all vertices of G with degree greater than k.
if $|H| > k$, **then output**("reject") {Observation 3.3.2.2};
if $|H| \leq k$, **then** $m := k - |H|$ and G' is the subgraph of G obtained
by removing all vertices of H with their incident edges.

Step 2: **if** G' has more than $m(k+1)$ vertices $[|V - H| > m(k+1)]$ **then output**("reject") {Observation 3.3.2.3}.

Step 3: Apply an exhaustive search (by backtracking) for a vertex cover of size at most m in G'.
if G' has a vertex cover of size at most m, **then output**("accept"), **else output**("reject").

Which step can be accelerated?

Step3. Exhaustion method $\rightarrow$ divide-and-conquer strategy.

$(G, m) \in VC \; iff \; [(G_1, m-1) \in VC \; or \; (G_2, m-1) \in VC]$.

Complexity:

$O(n) + O(1) + O\left(2^m m(k+1)\right) < O(2^k n)$.

- **Exercise 3.3.2.8. Let, for every Boolean function $\boldsymbol{\Phi}$ in CNF, $\boldsymbol{Var(\Phi)}$ be the number of variables occurring in $\boldsymbol{\Phi}$. Prove that MAX-SAT is fixed-parameter-tractable according to *Var*.**

Design a *Var*-parameterized polynomial-time algorithm

$$Var(\Phi) = k$$

Exhaustive method: $O(2^k n)$

- **Exercise 3.3.2.9. Let $((X, F), k), F \subseteq Pot(X)$, be an instance of the decision problem $Lang_{sc}$. Let, for every $x \in X$, $num_F(x)$ be the number of sets in F that contain x. Define**

$$Pat\big((X, F), k\big) = \max\big\{k, \max\{num_F(x) | x \in X\}\big\}$$

- **that is a parameterization of $Lang_{sc}$. Find a *Pat*-parameterized polynomial time algorithm for $Lang_{sc}$.**

Set cover problem. *Pot(X)* is the set of all subsets of the set X.
Similar to VCP, use divide-and-conquer.
Divide: $((X_i, F_i), k-1)$: Select any $S_i \in F$. Let $X_i = X \backslash S_i, F_i = f(F, i)$, function f delete S_i and delete all elements of S_i in $S_1, \dots S_{i-1}, S_{i+1}, \dots, S_l$
$((X_T, F_T), 0)$ is trivial. Complexity is $O(Pat^{Pat}|X|)$.

- 思考题：说明可以用确定的多项式算法解网络最大流问题，不必限制输入

Change step 3 and 4 of Algorithm 3.2.3.10

(1) Edmonds–Karp algorithm: Find a shortest path P by BFS

(2) Dinic's blocking flow algorithm: Build a layered graph with BFS on the residual graph

Algorithm 3.2.3.10 (The Ford-Fulkerson Algorithm).

Input: $(V, E), c, s, t$ of a network $H = ((V, E), c, \mathbb{Q}^+, s, t)$.

Step 1: Determine an initial flow function f of H (for instance, $f(e) = 0$ for all $e \in E$); $HALT := 0$

Step 2: $S := \{s\}$; $\overline{S} := V - S$;

Step 3: **while** $t \notin S$ and $HALT$=0 **do**

begin find an edge $e = (u, v) \in E(S, \overline{S}) \cup E(\overline{S}, S)$ such that

$res(e) > 0$

–$c(e) - f(e) > 0$ if $e \in E(S, \overline{S})$ and $f(e) > 0$ if $e \in E(\overline{S}, S)$";

if such an edge does not exist **then** $HALT := 1$

else if $e \in E(S, \overline{S})$ **then** $S := S \cup \{v\}$

else $S := S \cup \{u\}$;

$\overline{S} := V - S$

end

Step 4: **if** $HALT$= 1 **then return** (f, S)

else begin find an augmenting path P from s to t, which consists of vertices of S only; –this is possible because both s and t are in S";

compute $res(P)$;

determine f' from f as described in Lemma 3.2.3.9

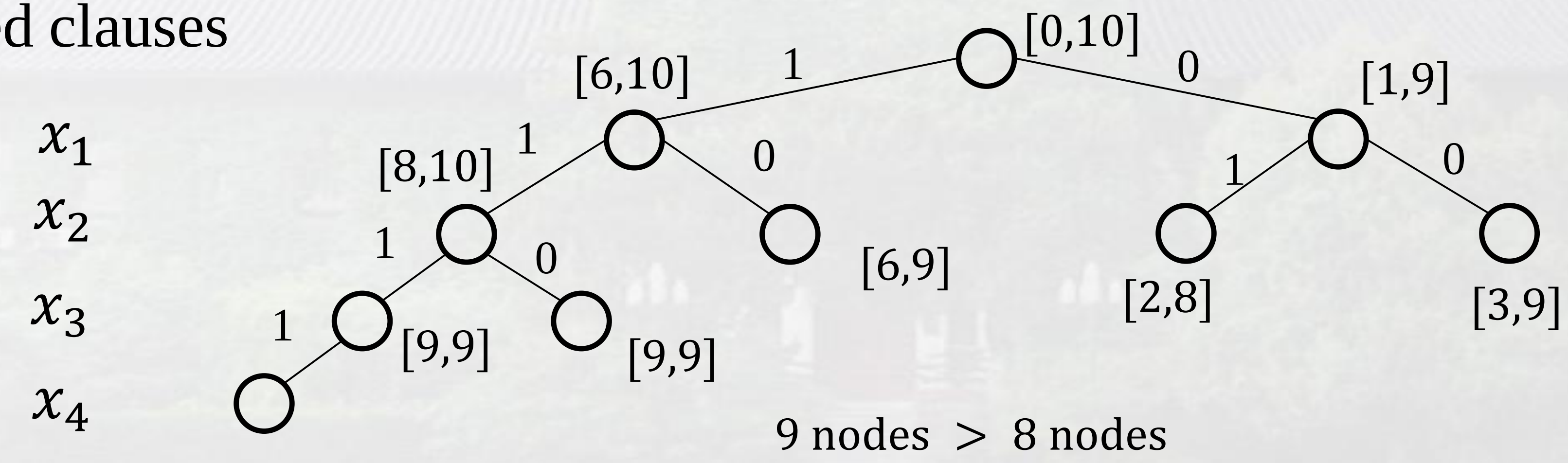
end;

goto Step 2

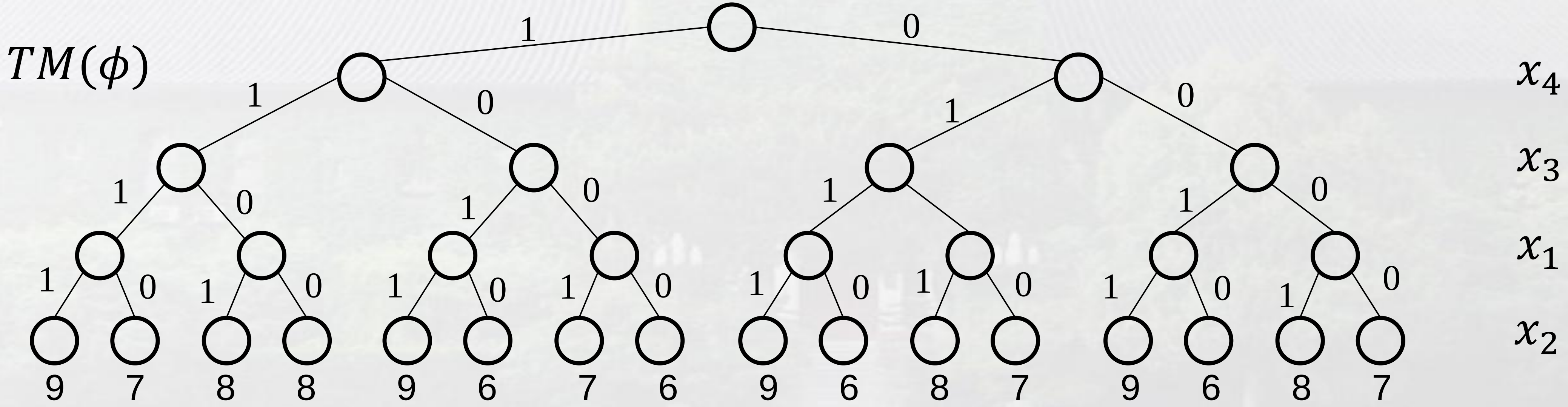
第四周作业

- **Exercise 3.4.2.1. Perform branch-and-bound of $TM(x)$ in Figure 3.7 by breadth-first-search and compare its time complexity (the number of generated vertices) with the depth-first-search strategies depicted in Figures 3.8 and 3.9.**

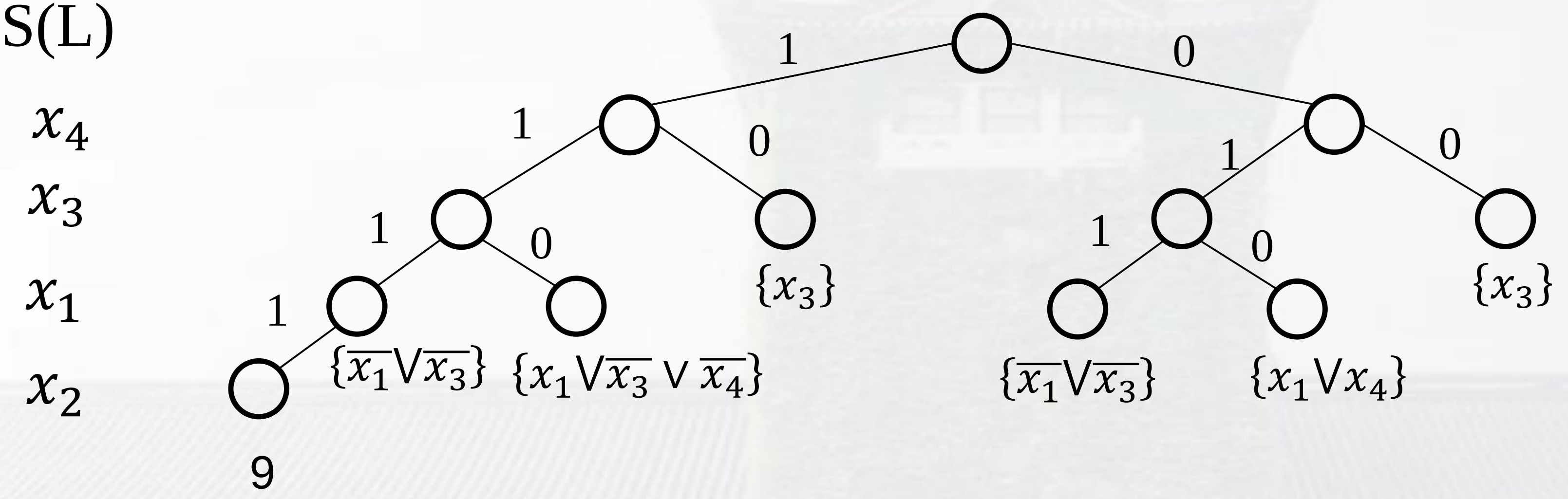
How to prune? Define $[a, b]$: the minimum and maximum possible number of satisfied clauses



- **Exercise 3.4.2.2. Take the ordering $x4, x3, x1, x2$ of the input variables of the formula $\phi(x4, x3, x1, x2)$ and build the backtrack tree $TM(\phi)$ according to this variable ordering. Use this $TM(\phi)$ as the base for the branch-and-bound method. Considering different search strategies, compare the number of visited vertices with the branch-and-bound implementations presented in Figures 3.8 and 3.9.**

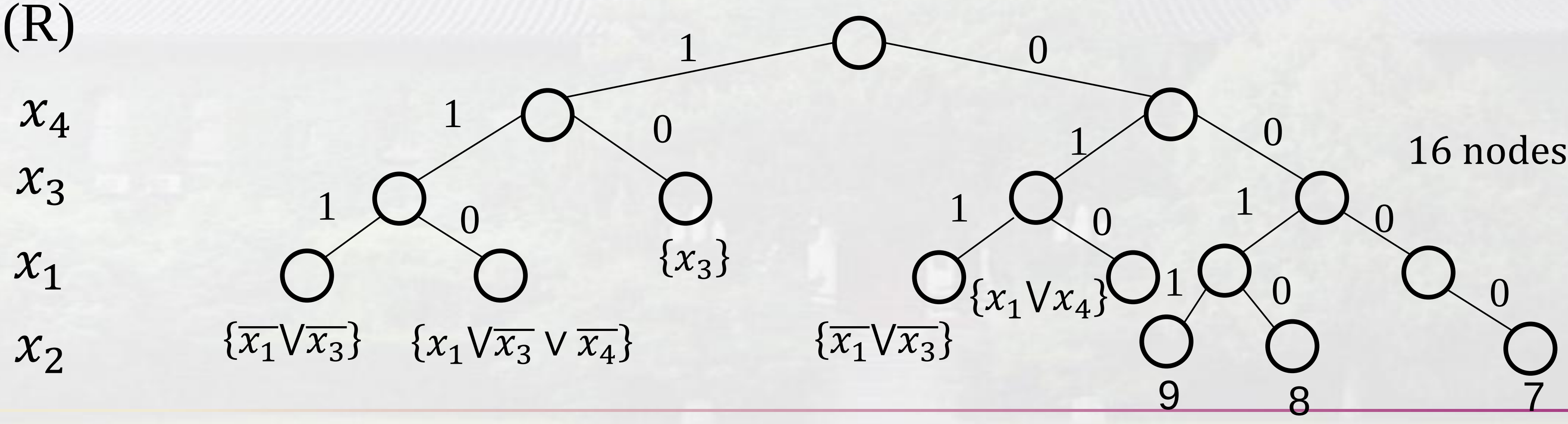


DFS(L)



12 nodes > 8 nodes

DFS(R)



16 nodes < 18 nodes

- 思考题：采用**branch-and-bound**方法解背包问题，并分别给出对你的方法有利与不利的输入

Knapsack Problem: KP $(b, \{w_1, w_2, \dots w_n\}, \{c_1, c_2, \dots, c_n\})$

(1) Build a backtracking tree. In every inner vertex of the search tree one branches according to two possibilities: whether an item is in the knapsack.

(2) Pruning and BFS/DFS. The current feasible solution $cost = c$.

I. $\sum w_{k_i} > b$

II. Current cost $= a$, current weight $= d$, the best $\frac{c_i}{w_i}$ $(w_i < b - d)$ of the rest items is q. If $a + (b - d) * q < c$, cut. (Find the largest rest cost?).

(3) Good input: the optimal solution are found first and the pruning strategy is used many times.

Bad input: it is hard to prune the tree

Q&A