

- 教材讨论
 - JH第3章第2节第1、2小节

问题1： 伪多项式时间算法

- U和Value(h)-U之间有什么联系和区别？
A是它们中哪个问题的算法？ 复杂度是多少？

Definition 3.2.1.1. *Let U be an integer-valued problem, and let A be an algorithm that solves U . We say that A is a **pseudo-polynomial-time algorithm for U** if there exists a polynomial p of two variables such that*

$$\text{Time}_A(x) = O(p(|x|, \text{Max-Int}(x)))$$

for every instance x of U .

Definition 3.2.1.2. *Let U be an integer-valued problem, and let h be a non-decreasing function from \mathbb{N} to \mathbb{N} . The **h -value-bounded subproblem of U** , **Value(h)- U** , is the problem obtained from U by restricting the set of all input instances of U to the set of input instances x with $\text{Max-Int}(x) \leq h(|x|)$.*

- 你能给出素数判定问题的一个伪多项式时间算法吗？

问题1: 伪多项式时间算法 (续)

- 什么是KP?

在KP的DP算法中, 以下概念各是什么含义?

- $I = (w_1, \dots, w_n, c_1, \dots, c_n, b)$

- $I_i = (w_1, w_2, \dots, w_i, c_1, c_2, \dots, c_i, b)$

- triple $(k, W_{i,k}, T_{i,k}) \in \left\{ 0, 1, 2, \dots, \sum_{j=1}^i c_j \right\} \times \{0, 1, 2, \dots, b\} \times Pot(\{1, \dots, i\})$

- TRIPLE_i $TRIPLE_i$

问题1：伪多项式时间算法 (续)

- 你能基于上述概念解释这个算法的基本过程吗？
- 你能证明这个算法的正确性吗？
你会计算它的时间复杂度吗？（如何体现伪多项式？）

Algorithm 3.2.2.2 ((DPKP)).

Input: $I = (w_1, w_2, \dots, w_n, c_1, c_2, \dots, c_n, b) \in (\mathbb{N} - \{0\})^{2n+1}$, n a positive integer.

Step 1: $TRIPLE(1) := \{(0, 0, \emptyset)\} \cup \{(c_1, w_1, \{1\}) \mid \text{if } w_1 \leq b\}$.

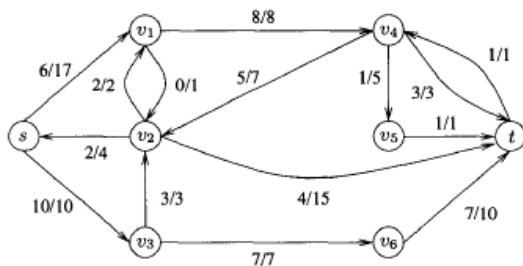
Step 2: **for** $i = 1$ **to** $n - 1$ **do**
 begin $SET(i + 1) := TRIPLE(i)$;
 for every $(k, w, T) \in TRIPLE(i)$ **do**
 if $w + w_{i+1} \leq b$ **then**
 $SET(i + 1) := SET(i + 1) \cup \{(k + c_{i+1}, w + w_{i+1}, T \cup \{i + 1\})\}$;
 Set $TRIPLE(i + 1)$ as a subset of $SET(i + 1)$ containing exactly one triple (m, w', T') for every achievable profit m in $SET(i + 1)$ by choosing a triple with the minimal weight for the given m
 end
 end

Step 3: Compute $c := \max\{k \in \{1, \dots, \sum_{i=1}^n c_i\} \mid (k, w, T) \in TRIPLE(n)$ for some w and $T\}$.

Output: The index set T such that $(c, w, T) \in TRIPLE(n)$.

问题1: 伪多项式时间算法 (续)

- 什么是最大流问题?
- 你能解释Ford-Fulkerson算法的基本过程吗?
- 你会计算它的时间复杂度吗? (如何体现伪多项式?)
- 你能举一个糟糕的例子吗?



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: $S := \{s\}; \bar{S} := V - S$;

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

begin find an edge $e = (u, v) \in E(S, \bar{S}) \cup E(\bar{S}, S)$ such that $res(e) > 0$

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

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

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

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

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

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

问题2: strongly NP-hard

- 一个strongly NP-hard问题可能存在伪多项式时间算法吗？为什么？

Definition 3.2.4.1. *An integer-valued problem U is called **strongly NP-hard** if there exists a polynomial p such that the problem $\text{Value}(p)\text{-}U$ is NP-hard.*

- 如何证明一个问题是strongly NP-hard？你能以TSP为例来说明吗？
- 为什么这句话成立？
 - Every weighted version of an optimization graph problem (e.g., WEIGHT-VCP) is strongly NP-hard if the original “unweighted” version (e.g., MIN-VCP) is NP-hard.

问题3：参数化

- 从不同的角度，谈谈你对参数化的理解、参数化的意义以及与伪多项式算法的关系

Definition 3.3.1.1. Let U be a computing problem, and let L be the language of all instances of U . A **parameterization** of U is any function $Par: L \rightarrow \mathbb{N}$ such that

- (i) Par is polynomial-time computable, and
- (ii) for infinitely many $k \in \mathbb{N}$, the **k -fixed-parameter set**

$$Set_U(\mathbf{k}) = \{x \in L \mid Par(x) = k\}$$

is an infinite set.

We say that A is a **Par -parameterized polynomial-time algorithm** for U if

- (i) A solves U , and
- (ii) there exists a polynomial p and a function $f: \mathbb{N} \rightarrow \mathbb{N}$ such that, for every $x \in L$,

$$Time_A(x) \leq f(Par(x)) \cdot p(|x|).$$

问题3：参数化 (续)

- 对于一个问题，可能有多种参数化方法，如何评价其好坏？
 - Capture the inherent difficulty of particular input instances.
 - One can design a practical parameterized polynomial-time algorithm.
 - Most of the problem instances occurring in the considered application have this parameter reasonably small.

Definition 3.3.1.1. Let U be a computing problem, and let L be the language of all instances of U . A **parameterization of U** is any function $Par: L \rightarrow \mathbb{N}$ such that

- (i) Par is polynomial-time computable, and
- (ii) for infinitely many $k \in \mathbb{N}$, the **k -fixed-parameter set**

$$Set_U(k) = \{x \in L \mid Par(x) = k\}$$

is an infinite set.

We say that A is a **Par -parameterized polynomial-time algorithm for U** if

- (i) A solves U , and
- (ii) there exists a polynomial p and a function $f: \mathbb{N} \rightarrow \mathbb{N}$ such that, for every $x \in L$,

$$Time_A(x) \leq f(Par(x)) \cdot p(|x|).$$

问题3：参数化 (续)

- 什么是VC问题？
- 你理解它的两个参数化算法了吗？
为什么说第二种更好？

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").

We consider the following divide-and-conquer strategy. Let (G, k) be an input instance of the vertex cover problem. Take an arbitrary edge $\{v_1, v_2\}$ of G . Let G_i be the subgraph of G obtained by removing v_i with all incident edges from G for $i = 1, 2$. Observe that

$$(G, k) \in \text{VC} \text{ iff } [(G_1, k-1) \in \text{VC} \text{ or } (G_2, k-1) \in \text{VC}].$$

Obviously, $(G_i, k-1)$ can be constructed from G in time $O(|V|)$. Since, for every graph H , $(H, 1)$ is a trivial problem that can be decided in $O(|V|)$ time and the recursive reduction of (G, k) to subinstances of (G, k) can result in solving at most 2^k subinstances of (G, k) , the complexity of this divide-and-conquer algorithm is in $O(2^k \cdot n)$.