习题参考答案

习题

11.1

算法运行时间为$O(nW)$。这似乎是在多项式时间内解决了 0/1 背包问题，但是该算法并非严格意义上的多项式时间算法。它的运行时间依赖背包载重量的大小。一个多项式时间算法应该仅仅依赖于物品的数目，而不是他们本身重量或价值的大小。因此，DPKnapsack(I,W) 算法是一个伪多项式时间算法。

11.2

最优化问题 LongPathLenght 可以在多项式时间内解决，显然可以在多项式时间内判定 LongestPath。反过来，如果 LongestPath 能够在多项式时间内判定，假设答案是“是”则我们可以逐步减少 k 的值，然后再调用判定算法。否则，我们可以增加 k 的值，然后再调用判定算法，由于$0 < k < |V|$，调用判定算法的次数是有限次，因此，可以在多项式时间里解决最优化问题 LongPathLenght。

11. 3

注意这里 TSP 是判定问题，可以类似书上图着色问题及上题类似求解，略。

11.4

简单，略。

11.5

考虑如下算法：

```
RunSlow(n)
1    s←a
2    for i←1 to n do
3         s←Concatenate(s, s)
```

调用O(n)个子程序，每个花线性时间。第一次调用Concatenate(s, s)，连接两个a，花时间O(2n)，第2次调用连接大小各为2的字符串，花时间$O(2^2n)$，总共n次调用，所用的时间为

$O(2n)+ O(2^2n)+\cdots+ O(2^nn)= \sum_{k=1}^{n} 2^k n$，时间复杂度显然不是多项式时间复杂度。如果常数次调用，则仍然是多项式时间。

11. 6

只要找到该问题的一个多项式时间验证算法即可。主要理解同构的概念。

设$G_1 = (V_1, E_1)$，$G_2 = (V_2, E_2)$ 表示两个图，若存在双射函数 $f: V_1 \to V_2$，使得$(u,v) \in E_1$ 当且仅当$(f(u), f(v)) \in E_2$，则两图同构。

设输入为：$G_1 = (V_1, E_1)$ 和$G_2 = (V_2, E_2)$，证书为函数 f，只需验证是否有$(u,v) \in E_1$ 当且仅当$(f(u), f(v)) \in E_2$。如果图用邻接矩阵存储，可在$O(|V|^2)$ 时间内验证。我们也可以构造一个非确定性算法：

GraphIsOmorphism(G1, G2)

1 **if** G1 does not have the same number of vertices as G2 **then return** false.
2 nondeterministically guess a permutation (bijection) f of m vertices.
3 **for** each pair of vertices (u, v) in G1 **do**
4 verify that (u, v) is an edge in G1 if and only if (f(u), f(v)) is an edge of G2
5 **if** all edges agree, **then return** true
6 **else return** false

11. 7

如果HamCycle∈ P，首先注意到对回路中的每个顶点精确地有两条边与之相连。可以如下找汉密尔顿回路:选择一个顶点$v \in V$，令E_v表示所有与v相连的边的集合。找到一对边$e_1, e_2 \in E_v$使得$G' = (V, E - E_v) \cup \{e_1, e_2\}$含有汉密尔顿回路，这个能够在多项式时间里通过尝试所有可能的一对边来实现。对图G'，类似上述过程求解。最终在多项式时间里得到一个图$H = (V, C)$，其中C为所求的一个汉密尔顿回路。

11.8

要证明 HamPath 为 NP 问题，只要找到一个多项式时间验证算法，具体如下：
输入为<G, u, v>,证书为一个顶点序列 v1,v2,…….vn,
1 检查是否有 v1=u,vn=v;
2 检查该序列是否包含了图中所有的顶点；
3 检查序列中任意两个相邻点在图中是否存在边；
上述三步显然可以在多项式时间内完成，得证。

11.9

由于停机问题是不可判定的，因此它不是 NP 问题，按照定义，显然它也不是一个 NPC 问题。对于 NPC 问题 SAT，我们可以构造一个多项式时间转换算法：任给一个输入命题公式，该算法对该公式枚举其变元的所有赋值，如果存在赋值使其为真，则停机，否则进入无限循环。这样，判断公式是否可满足便转化为判断以公式为输入的算法是否停机。因此，NPC 问题 SAT 可以多项式时间约简到停机问题，所以，停机问题是难问题。

11.10

如果有$L_1 \leq_P L_2$，且$L_2 \leq_P L_3$，则存在多项式可计算得函数f_1和f_2，使得对任意的$x \in \{0, 1\}^*$，$x \in L_1$当且仅当$f_1(x) \in L_2$，$x \in L_2$当且仅当$f_2(x) \in L_3$。函数$f_2(f_1(x))$是多项式可计算的且满足$x \in L_1$当且仅当$f_2(f_1(x)) \in L_3$，故所证成立。

11.11

两个差不多，这个更直接。

11.12

因为对于具有 m 个命题变元的公式ϕ，构造真值表需要2^m行，显然是指数级的，因此

不能导致多项式约简。

11.13

因为我们只需要检查任一个子句是否可满足，从而可决定整个子句是否可满足。

11.14

令这个判定算法为 A，其时间复杂度为 $O(n^k)$。任给一个命题公式 ϕ，我们调用 A(ϕ)，如果返回假，则停止，否则说明 ϕ 为真，我们可继续如下：令 $x_1=1$，我们得到一个命题公式 ϕ_1，如果 A(ϕ_1)返回假，则我们只需要令 $x_1=0$ 即可，否则 $x_1=1$。重复上面过程。如果有 *n* 个变元，上述过程只要重复 *n* 次，就可以找到可满足赋值。这相当于调用 *n* 次 A(ϕ)，故仍然是多项式时间复杂度 $O(nn^k)=O(n^{k+1})$。

11.15

算法思想：先根据给定的 2-CNF 构造有向图 G(V,E)，对每一个正文字及相应的负文字各设置一个顶点。假设公式中有 n 个独立的正文字，则|V|=2n。对于每个子句执行如下操作，假设子句为(x∨y)，则添加一条从¬x 到 y 的边及一条从¬y 到 x 的边。我们可得如下结论：

2-CNF 是不可满足的，当且仅当存在一个文字 x，在图 G 中存在从 x 到¬x 及从¬x 到 x 的路径。

证明： 设 ρ 是给定的 2-CNF 的一个真值指派。

一般地，对于某个正文字 x，设 ρ(x)=T ，若要使给定的 2-CNF 为真，则 x 指向的变元也必须为 T，依此类推。而 ¬x=F，指向 ¬x 的变元也必须为 F，依此类推。

所以若给定的 2-CNF 可满足，必不存在从 x 到 ¬x 的路径，同理，不存在从 ¬x 到 x 的路径。

实现：通过深度优先搜索求图 G 的强连通分支，然后在每个强连通分支里判断是否同时存在一个正文字及相应的负文字。

分析：

构造图：O(|*V*|+|*E*|)

求强连通分支：O(|*V*|+|*E*|)

判断：O(|*V*|)

11.16

If we did not require the vector x to have integer values, then this is the linear programming problem and is solvable in polynomial time. This one is more difficult. As usual it is easy to show that 0-1 INT is in NP. Just guess the values in x and multiply it out.A reduction from 3-SAT finishes the proof. In order to develop the mapping from clauses to a matrix we must change a problem in logic into an exercise in arithmetic. Examine the following chart. It is just a spreadsheet with values for the variables x_1, x_2, and x_3 and values for some expressions formed from them.

Expressions	Values							
X_1	0	0	0	0	1	1	1	1
X_2	0	0	1	1	0	0	1	1
X_3	0	1	0	1	0	1	0	1
$+ X_1 + X_2 + X_3$	0	1	1	2	1	2	2	3
$+ X_1 + X_2 - X_3$	0	-1	1	0	1	0	2	1
$+ X_1 - X_2 - X_3$	0	-1	-1	-2	1	0	0	-1
$- X_1 - X_2 - X_3$	0	-1	-1	-2	-1	-2	-2	-3

Above is a table of values for arithmetic expressions. Now we shall interpret the expressions in a logical framework. Let the plus signs mean *true* and the minus signs mean *false*. Place *or*'s between the variables. So, $+x_1 + x_2 - x_3$ now means that

x_1 is *true*, *or* x_2 is *true*, *or* x_3 is *false*.

If 1 denotes *true* and 0 means *false*, then we could read the expression as $x_1=1$ *or* $x_2=1$ *or* $x_3=0$.

Now note that in each row headed by an arithmetic expression there is a minimum value and it occurs exactly once. Find exactly which column contains this minimum value. The first expression row has a zero in the column where each x_i is also zero. Look at the expression. Recall that $+x_1 + x_2 + x_3$ means that at least one of the x_i should have the value 1. So, the minimum value occurs when the expression is *not satisfied*.

Look at the row headed by $+x_1 - x_2 - x_3$. This expression means that x_1 should be a 1 or one of the others should be 0. In the column containing the minimum value this is again not the case.

The points to remember now for each expression row are:

a) Each has *exactly* one column of minimum value.
b) This column corresponds to a nonsatisfying truth assignment.
c) Every other column satisfies the expression.
d) All other columnms have higher values.

Here is how we build a matrix from a set of clauses. First let the columns of the matrix correspond to the variables from the clauses. The rows of the matrix represent the clauses - one row for each one. For each clause, put a 1 under each variable which is not complemented and a -1 under those that are. Fill in the rest of the row with zeros. Or we could say:

$$a_{i,j} = \begin{cases} 1 \text{ if } v_j \in \text{ clause i} \\ -1 \text{ if } \overline{v_j} \in \text{ clause i} \\ 0 \text{ otherwise} \end{cases}$$

The vector b is merely made up of the appropriate minimum values plus one from the above chart. In other words:

b_i = 1 - (the number of complemented variables in clause i).

The above chart provides the needed ammunition for the proof that our construction is correct. The proper vector x is merely the truth assignment to the variables which satisfies all of the clauses. If there is such a truth assignment then each value in the vector Ax will indeed be greater than the minimum value in the appropriate chart column.

If a 0-1 valued vector x does exist such that $Ax \geq b$, then it from the chart we can easily see that it is a truth assignment for the variables which satisfies each and every clause. If not, then one of the values of the Ax vector will always be less than the corresponding value in b. This means that the that at least one clause is not satisfied for any truth assignment.

Here is a quick example. If we have the three clauses:

$$(x_1, \overline{x_3}, x_4), (\overline{x_2}, \overline{x_3}, x_4), (x_1, x_2, x_3)$$

then according to the above algorithm we build A and b as follows.

$$\begin{bmatrix} 1 & 0 & -1 & 1 \\ 0 & -1 & -1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \geq \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$$

Note that everything comes out fine if the proper values for the x_i are put in place. If x_3 is 0 then the first entry of Ax cannot come out less than 0 nor can the second ever be below -1. And if either x_2 or x_1 is 1 then the third entry will be at least 1.

11.17

将子集合问题约简到 Partition，即 SubsetSum$\leq_P$ Partition：

任给子集合问题的一个实例 $S=\{x_1,x_2,\cdots,x_n\}$ 以及一个正整数 t，我们可以构造一个划分实例：令 $s=\sum_{i=1}^{n}x_i$，令 $x_{n+1}=3s-t$，$x_{n+2}=3s-(s-t)=2s+t$。便可以得到划分实例：$S=\{x_1,x_2,\cdots,x_n,x_{n+1},x_{n+2}\}$。容易验证如果子集合的实例有一个"yes"解当且仅当划分实例有一个"yes"解。

11.18

将 Partition 问题约简到 BinPacking，即 Partition$\leq_P$ BinPacking：

任意给定 Partition 问题的一个实例，即一个物品的集合 $S=\{s_1,\cdots,s_i,\cdots,s_n\}$，其中 s_i 为正整数，我们可以如下构造一个 BinPacking 实例，物品集合一样，箱子的重量 $W=\frac{1}{2}\sum_{i=1}^{n}s_i$，箱子的个数 k=2。

容易验证划分的实例有一个“yes”答案当且仅当 BinPacking 的实例有一个“yes”答案。

11.19

有两种证明方法：一种是将 Partition$\leq_P$ ParallelScheduling problem with two machine，然后再约简到 ParallelScheduling。另一种办法是直接将 Partition$\leq_P$ ParallelScheduling。

11.20

用动态规划算法求解，其时间复杂度为 $O(nt)$，当 t 表示成一元形式，是一个常数，因此整个算法时间复杂度是多项式。

11.21

HamCycle$\leq_P$ 最长简单回路问题，事实上，前者是后者的一个特例。