

P4

8.2

```
DFS(w)
stack stk
stk.push(w)
while !stk.empty() do
    u = stk.top()
    if u.color == WHITE then
        u.color = GRAY
        < Preorder processing of node u >
        if u has prev neighbor then
            v = u.prevNeighbor
            u.prevNeighbor = null
            < Backtrack processing of edge uv >
        if u has not next neighbor then
            < Postorder processing of node u >
            u.color = BLACK
            stk.pop()
        else
            v = u.nextNeighbor()
            if v.color == WHITE then
                < Exploraty processing of edge uv >
                stk.push(v)
                u.prevNeighbor = v
            else
                < Checking edge uv >
```

8.5

\Rightarrow : v 是割点, $G' = G/\{v\}$, G' 不连通, 则 G 中存在两点 w 和 x , 在 G 中连通, 在 G' 中不连通。假设有一条 w 到 v 的路径, 使得其不经过 v , 则在 G' 这条路径仍然存在, w 和 x 仍然连通, 矛盾。

\Leftarrow : 若 w 到 x 所有路径上都经过了点 v , 则删去点 v 后 w 和 x 不连通, 即 G 不连通, 所以 v 是割点。

8.7

记有向图 G 的收缩图为 G' 。

假设 G' 有环, 则存在两个不同点 x 和 y , x 和 y 相互到达。

即在 G 中, 存在 $u \in x$ 和 $v \in y$, u 和 v 相互到达, 则 u 和 v 必属于一个强连通片, 矛盾。

8.8

- 第一次DFS不能换BFS。DFS是在顶点所在子树全部遍历完成后再进栈, 首节点的活动区间包含同一个强连通片所有其他节点的活动区间。而BFS是在顶点的孩子进队列后就进栈, 此时该顶点的孩子还没有进栈, 不满足推论8.1。
- 第二次DFS可以换BFS。只要保证能遍历到顶点就行。

8.9

无向连通图的深度优先遍历树的根节点 v 是割点 $\Leftrightarrow v$ 至少有2个子节点 x 和 y 。

\Rightarrow : v 是割点, 如果没有子节点, 则 v 不是割点, 如果只有1个子节点, 那么删去 v 图还是连通的。

\Leftarrow : 若删去 v 后, x 和 y 不连通 (若仍连通, 则 x 和 y 应该在同一子树中), v 是割点。

8.10

仍然正确。

证明方法仍然是定理8.5

8.11

- 引理8.9

\Rightarrow : 假设有BE指向 v 的祖先, 则删去 uv 后图仍连通, uv 不是桥, 矛盾。

\Leftarrow : 删去 uv 后, v 为根的子树无法到达 v 的祖先, 图不连通, 所以 uv 是桥。

- 定理8.6

证明方法和定理8.5类似。

8.14

- 找环, 就是找BE, 找不到则无解。
- 找到BE后, 以BE任意端点作为顶点进行DFS, 遍历边时进行定向, 方向与遍历方向相同。
- 复杂度 $O(n + m)$

注意是无向图, 找BE要看是不是当前边的反向边!

8.15

- 若 G 中有桥 uv , 则记删去 uv 后形成的两个连通分量 x 和 y , 无论 uv 如何定向, 也无法保证 x 和 y 中的点能相互到达。
- 从任意节点开始DFS, 边的方向与边的方向相同。

8.19

```
TOPO()
queue q;
for each node u do
    if u.indeg == 0 then
        q.push(u)
while !q.empty() do
    u = q.front()
    result.push(u)
    q.pop()
    for each neighbor v of u do
        v.indeg--;
        if v.indeg == 0:
            q.push(v)
return result
```

如果有回路, 则回路上的点不在 $result$ 中。

8.20

- 直接DFS，若有没有搜到的点，则不能到达图中其他所有节点。
- 缩点，找到入度为0的点，从该点开始DFS，若有没有搜到的点，则不能到达图中其他所有节点。

8.22

- 缩点
- 出度为0的强连通片之间比较强连通片点的个数，最小的即是影响力最小的点。
- 入度为0的强连通片进行DFS，统计每个强连通片能搜到的顶点个数，找到最大的即是影响力最大的点。

不能进行DAG DP/记忆化搜索！

8.24

关键路径/拓扑/DAG DP

$$f[u] = \max(f[v]) + 1, (u, v) \in E$$

8.26

1)

- 建图，有 n 个顶点，每个顶点对应一个小孩，如果 i 恨 j ，则有 $i \rightarrow j$ 的有向边。（要写出建图过程）
- 求拓扑序，如果有环则不存在满足要求的排法。

2)

- 同（1）一样建图。
- 同8.24。若有环则不存在满足要求的排法。

8.28

$$(1) x_1 = TRUE, x_2 = TRUE, x_3 = FALSE, x_4 = TRUE$$

(2) 答案不唯一。

(3) 按要求建图即可。

(4)

- 如果有同时包含 x 和 \bar{x} ，则有路径 $x \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow \bar{x}$ 和 $\bar{x} \rightarrow u_1 \rightarrow u_2 \rightarrow \dots \rightarrow x$
- 若 x 取TRUE，则 v_1 要取TRUE， v_2 也要取TRUE..... \bar{x} 也要取TRUE，矛盾。
- 若 x 取FALSE，则 \bar{x} 为TRUE， u_1 要取TRUE， u_2 也要取TRUE..... x 也要去TRUE，矛盾。

(5) 要证明实例 I 是可满足的，则需要找到一种对每个变量真假状态的设置方案，见（6）

(6)

算法：

- 图 G 缩点得到图 G'
- 求图 G' 的拓扑序。
- 图 G 中每个顶点 x 的序号即为在 G' 中强连通片的拓扑序。
- 对每个变量 x 和其反值 \bar{x} 的拓扑序：
 - $x < \bar{x}$ ，即可能 $x \rightarrow \bar{x}$ ， x 取FALSE。
 - $x = \bar{x}$ ，无解。
 - $x > \bar{x}$ ，即可能 $\bar{x} \rightarrow x$ ， x 取TRUE。
- 复杂度 $O(n + m)$

证明：

要证明算法正确性，则需要证明算法结束后，每个变量都已经确定了取值。

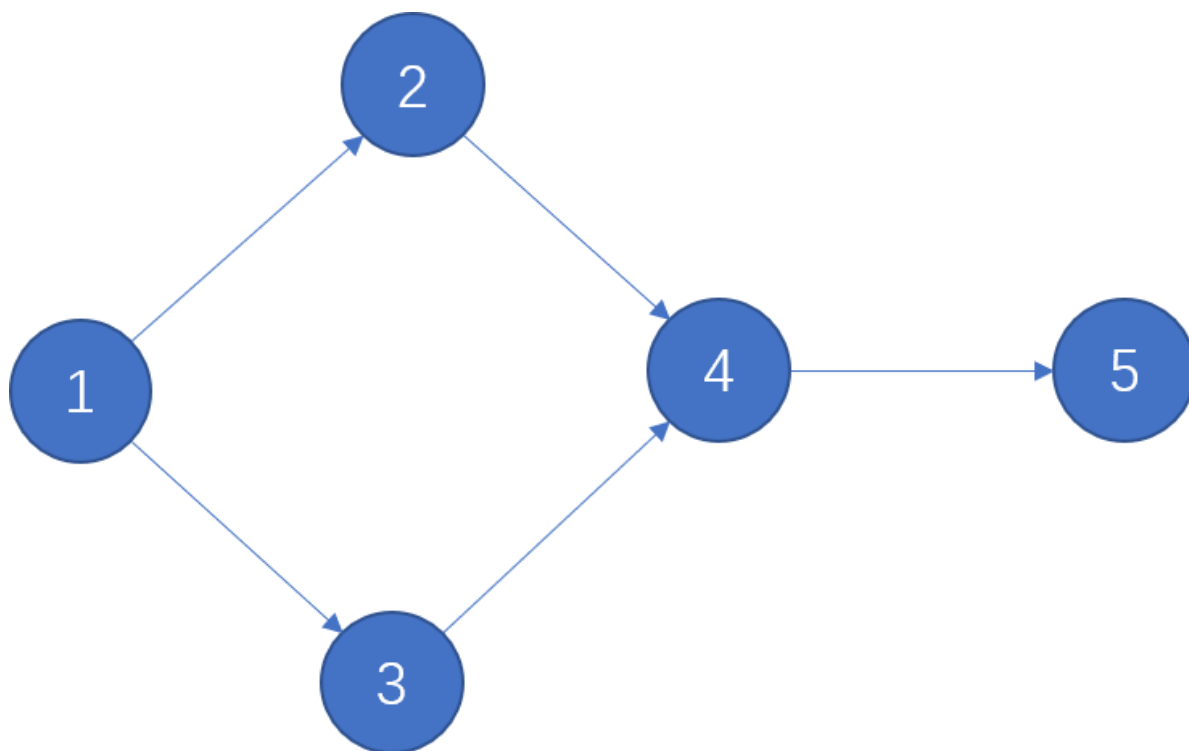
因为每个变量的真假都有确定的拓扑序，所以一定能按照这个规则确定真假值，并且没有冲突。

9.1

- 假设在遍历 v 时发现了BE，指向了 w ，则在之前BFS中，队列弹出 w ，会发现 v 为 w 的访问过的邻居，则BE其实是TE，矛盾。
- 假设在遍历 v 时发现了DE，指向了 w ，实际上还是TE，矛盾。

9.2

不成立。



刚发现3时，有3->4->5白色路径，但是5的祖先可能是2。

9.3

DFS可以判断是否为二分图。

- DFS不需要队列的空间开销，更适合深度大、节点度数小的图。
- BFS不需要进行递归，更适合深度小，节点度数大的图。

9.4

- DFS
 - 有向图和无向图都是找BE（灰色节点）
- BFS
 - 有向图，找BE（黑色祖先），注意黑色节点不一定是BE，也可能是CE，需要额外判断是否有共同parent。
 - 无向图，遇到灰色节点（CE）。

9.5

本质还是找环，和9.4一样。

如何控制到 $O(n)$? 遍历边的次数其实还是 $O(n)$ 而不是 $O(m)$ 。

9.8

(1) 深度优先遍历树或者宽度优先遍历树就是最小生成树。

(2)

- 先随便遍历一次生成一个遍历树。
- 对剩下的11条边的每条边 uv ，从树上找 u 到 v 的路径，找到路径上最大边并与 uv 比较，如果 uv 更小则替换掉。

(3)

- 先只考虑权重为1的边，DFS或者BFS遍历生成一个森林。
- 对森林的每棵树看作一个点，只考虑权重为2的边，遍历生成一个遍历树，即为所求。

9.11

- 根据相识关系构造无向图，顶点表示候选的被邀请人，边表示两者相识。
- 修改K-DEGREE-SUBGRAPH算法，现在删除的方式是该点在原图的度小于5或在补图的度小于5都需要删去。
- 第7-10行处理完邻点后还需要对非邻点进行处理。
- 复杂度 $O(n^2)$

不能分两步求，对补图删点后，原图可能不一定满足五度子图的条件！

P5

10.3

1) e_1 一定在。

- Kruskal: 权值最小的边第一个被选中。
- MCE: 任意切中 e_1 都是最小的边。

2) e_2 是 e_1 重边就不在, 不是就在。

- Kruskal: e_2 的两点 u, v , 如果 u, v 有一个仍未在 MST 中则会将 e_2 加入 MST。
- MCE: 记 e_2 中与 e_1 不同的一点为 u , $V_1 = \{u\}$, $V_2 = V - V_1$, e_2 是 V_1 和 V_2 的 MCE。

3) 不一定在, 可能产生环。

- Kruskal: e_3 的两点 u, v , 可能已经都在 MST 中, 则不会将 e_3 加入 MST。
- MCE: 无论如何划分切, 都无法保证 e_3 一定是 MCE。

10.6

- Prim
 - 数组实现优先队列: $O(n^2)$
 - 堆实现优先队列-稠密图: $O(n^2 \log n)$
 - 堆实现优先队列-稀疏图: $O(n \log n)$
- Kruskal
 - 稠密图: $O(n^2 \log n)$
 - 稀疏图: $O(n \log n)$
- Prim 适合稠密图, Kruskal 适合稀疏图。

10.10

1) 无需更新。

2) 在原 MST 上添加 e 形成一个环, 删去环上最大边。

3) 无需更新。

4) 在原 MST 上删去 e 形成两个连通分支, 添加连接两个连通分支的权值最小的边。

10.13

- 对 Kruskal 算法进行改进
- 初始化先将 S 中的所有边加入 MST, 并对 S 中关联的点加入到并查集中。
- 之后再从 $E - S$ 中依次挑选权重最小且不会产生环的边加入 MST。

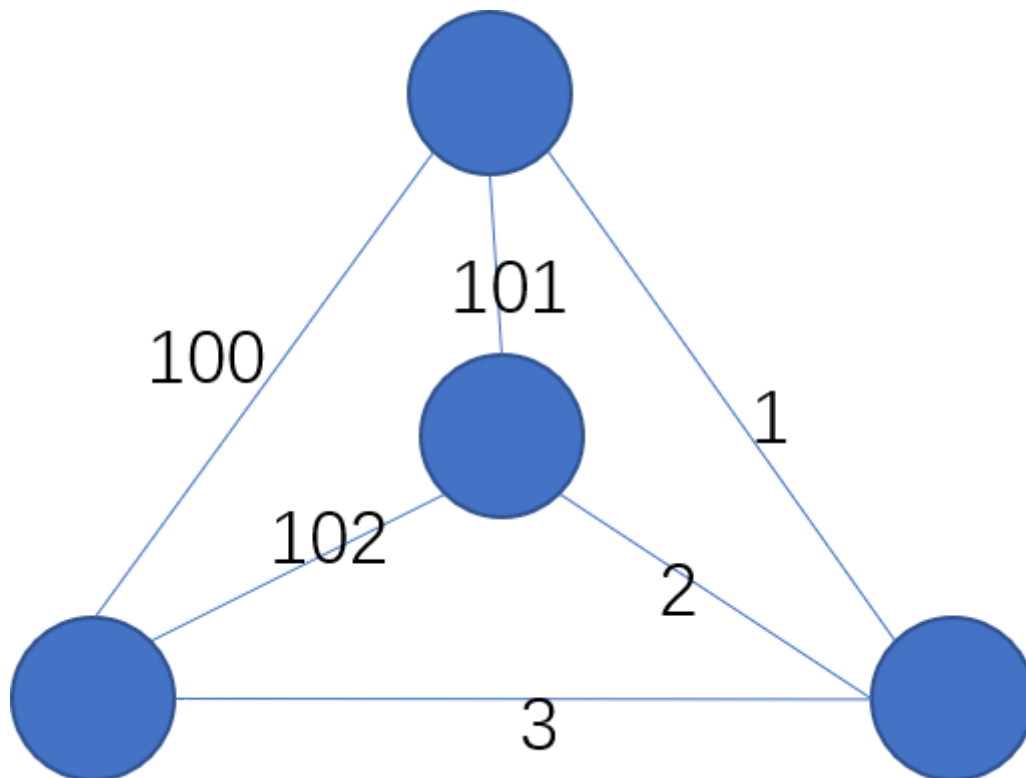
10.14

- 记 $e = \{u, v\}$, 删去权值大于等于 e 的边。
- 从 u 出发搜索 v 。
 - 如果能搜到, 说明 e 是某环上唯一的最重边, 不可能在 MST 中。
 - 如果搜不到, 假设 e 不在任意 MST 上, 则任意 MST 加上 e 都会形成圈, 且 e 是圈上唯一最重边, 因此 u 能搜到 v , 矛盾, 所以 e 在某个最小生成树上。

10.15

- 1) 错。反例：桥
- 2) 对。假设 e 属于某个MST，则可以把环上那个不在MST的边和 e 替换，权值变更小，矛盾。
- 3) 对。用MCE框架证明， $e = \{u, v\}$, $V_1 = \{u\}$, $V_2 = V - V_1$, e 是MCE。
- 4) 对。分是不是桥两种情况证明。不是桥用反证法。

5) 错。



- 6) 错。三角形，边权2、2、3。
- 7) 对。没有边权值限制。

10.16

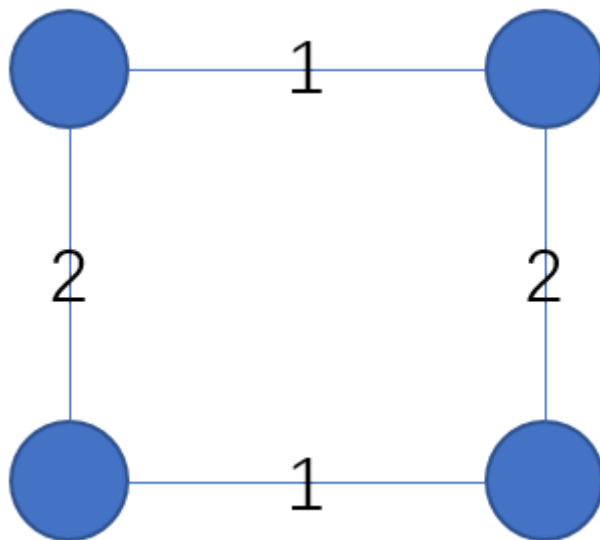
两个图有相同的边权大小序，且由于边权各不相同，MST唯一。

10.17

- 假设 $\exists e = \{u, v\} \in T \cap H$ ，且 e 不是 H 的任意一个MST上。
- $T' = T - \{e\}$ 后， u 和 v 在 T' 的两个不相交点集 V_1, V_2 上。
- $T' \cap H$ 应该也是两个不相交点集 V_1, V_2 。
- 又因为 H 的MST上没有 e ，但是有 u, v ，则必然有条边 e' ，连接着 V_1, V_2 。
- 因为 e' 是 H 的MST，所以 $e'.weight \leq e.weight$ 。
- 但是 e 也是 G 的MST，所以 $e.weight \leq e'.weight$ 。
- 所以 $e.weight = e'.weight$ 。
- 那么就可以把 e' 替换成 e ，还是 H 的MST，矛盾。

10.21

不正确。反例：



10.23

- 1) MST铺设管道，在挖井代价最小的房子挖井。
- 2) 增加一个超级源点，连向所有房子，边权为房子的挖井代价。在新图上求MST。

10.25

```

+---B---+
2|       |-2
A-----C
  3

```

10.27

- 设负权边为 $e = (\{u, v\}, w)$ 。
- $G' = G - \{e\}$ ，以 s 、 u 、 v 为源点在 G' 上运行三次Dijkstra算法，得到 $dis_s[t]$, $dis_u[t]$, $dis_v[t]$ 。
- $dis[t] = \min\{dis_s[t], dis_s[u] + w + dis_v[t], dis_s[v] + w + dis_u[t]\}$

10.31

- 最小生成树不变，因为偏序关系不变。
- 最短路径可能发生变化。例如四条边权重为1、1、1、4，增加1后变为2、2、2、5，明显改变了最短路径。

10.33

对Dijkstra增加点权操作。

- $Cost[s] = c_s$
- $Cost[v] = \min\{Cost[v], Cost[u] + l_e + c_v\}, e = (u, v)$

10.34

仍然正确。

Fringe以外的边仍然是正的，不影响证明。

10.36

注意 $l_e > 0$, 所以可以边Dijkstra边求 $best[u]$ 。

- $best[s] = 0$ 。
- 若 $dis[v] > dis[u] + l_e, e = (u, v)$, 更新 $best[v] = best[u] + 1$ 。
- 若 $dis[v] = dis[u] + l_e, e = (u, v)$, 更新 $best[v] = \min\{best[v], best[u] + 1\}$ 。

也可以在最短路径生成图上跑一遍BFS。

10.38

1) 把权值大于 L 的边删去, 从 s 出发DFS搜 t 即可。

2) 更改Dijkstra的三角不等式: $cap[v] = \min\{cap[v], \max\{cap[u], l_e\}\}, e = (u, v)$

11.1

- 将 S 写成 c 进制的形式。
- 记该算法得出的是 $(a_{n-1}, \dots, a_1, a_0)$, 假设有更优的做法, 得出的是 $(a'_{n-1}, \dots, a'_1, a'_0)$ 。

则有 $i_0 > i_1 > \dots > i_t, a'_{i_0} < a_{i_0}$, 且

$$a_{i_0} c^{i_0} + \sum_{j=1}^t a_{i_j} c^{i_j} = a'_{i_0} c^{i_0} + \sum_{j=1}^t a'_{i_j} c^{i_j}$$

$$(a_{i_0} - a'_{i_0}) c^{i_0} = \sum_{j=1}^t (a'_{i_j} - a_{i_j}) c^{i_j} < c^{i_0} \sum_{j=1}^t (a'_{i_j} - a_{i_j})$$

$$a_{i_0} + \sum_{j=1}^t a_{i_j} < a'_{i_0} + \sum_{j=1}^t a'_{i_j}$$

不是更优解, 矛盾。

12.1

- 初始化: $GO[i][j] = j$
更新时加入: $GO[i][j] = GO[i][k]$
- 初始化: $FROM[i][j] = i$
更新时加入: $FROM[i][j] = FROM[k][j]$

12.2

1) 类似10.38, 更改三角不等式: $cap(s, v) = \max\{cap(s, v), \min\{cap(s, u), c(u, v)\}\}$, 注意每次取的是最大的 $cap(s, u)$ 。

2) 同样更改三角不等式: $D[i][j] = \max\{D[i][j], \min\{D[i][k], D[k][j]\}\}$ 。

12.4

- 添加点 s , 连向 S 的所有点, 权值为0。
- 添加点 t , 连向 T 的所有点, 权值为0。
- 从 s 使用Dijkstra算法, 答案为 $dis[t]$ 。

12.7

- 以 v_0 为源点使用Dijkstra算法。
- 对转置图以 v_0 为源点使用Dijkstra算法。
- $dis(i, j) = dis(i, v_0) + dis(v_0, j)$

P6

13.1

- $S_{ij} = \emptyset$ 时, $c[i, j] = 0$
- $S_{ij} \neq \emptyset$ 时, $c[i, j] = \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\}$

初始化 ($O(n^3)$) :

```
for i := 1 to n do
  for j := 1 to n do
    for k := 1 to n do
      if f[i] <= s[k] and f[k] <= s[j] then
        s[i, j].add(k)
```

状态转移:

```
for i := n downto 1 do
  for j := 1 to n do
    if s[i, j].empty() then
      c[i, j] = 0
    else
      for k in s[i, j] do
        c[i][j] = max(c[i][j], c[i][k]+c[k][j]+1)
```

13.2

$f[i, j]$ 表示A的前*i*个数是否存在元素和为*j*。

- 初始化:
 - $f[i, 0] = True$
 - $f[i, j] = False, i = 0 \ \&\& \ j \neq 0$
- 状态转移:
 - $f[i, j] = f[i - 1, j] \text{ if } s_i > j$
 - $f[i, j] = f[i - 1, j] || f[i - 1, j - s_i], \text{ if } s_i \leq j$

时间复杂度 $O(nS)$

13.4

$f[i]$ 表示以第*i*个数结尾的最长非减序列的长度

- 初始化: $f[1] = 1$
- 状态转移: $f[i] = \max\{1, f[j] + 1\}, j < i, A[j] \leq A[i]$

时间复杂度 $O(n^2)$

13.5

$f[i, j]$ 表示前*i*个数的*j*-划分最低代价。

先求出X的前缀和 $S[1..n]$

- 初始化:
 - $f[i, 1] = S[i]$
 - $f[1, j] = S[1]$
- 状态转移: $f[i, j] = \min_{k=1}^{i-1} \{\max\{f[k, j-1], S[i] - S[k]\}\}$

时间复杂度 $O(kn^2)$, 空间复杂度 $O(kn)$

改进:

- $f[k, j-1]$ 是关于 k 的增函数, 而 $S[i] - S[k]$ 是关于 k 的减函数, 可以转化成找两函数交点的问题, 可以用二分法解决, 时间复杂度为 $O(kn \log n)$
- $f[*, j]$ 只与 $f[*, j-1]$ 有关, 空间复杂度可以降为 $O(n)$

13.8

1) $f[i, j]$ 表示 X 的前 i 个字符组成的串和 Y 的前 j 个字符组成的串的最长公共子序列。

- 初始化: $f[i, j] = 0, i = 0 || j = 0$
- 状态转移: $f[i, j] = \max\{f[i, j-1], f[i-1, j], f[i-1, j-1] + I\{X[i] = Y[j]\}\}$

2) $f[i, j]$ 表示 X 的前 i 个字符组成的串和 Y 的前 j 个字符组成的串的最长公共子序列。

- 初始化: $f[i, j] = 0, i = 0 || j = 0$
- 状态转移:

$$f[i, j] = \max\{f[i, j-1], f[i-1, j], f[i-1, j-1] + I\{X[i] = Y[j]\}, f[i, j-1] + I\{X[i] = Y[j]\}\}$$

3) $f[i, j, k]$ 表示 X 的前 i 个字符组成的串和 Y 的前 j 个字符组成的串的最长公共子序列 (X 的字符重复出现次数不超过 k 次)。

- 初始化: $f[i, j, k] = 0$
- 状态转移:
 - 若 $X[i] = Y[j]$:
 - $f[i, j, 1] = \max\{f[i-1, j-1, k] + 1\}$
 - $f[i, j, k] = \max\{f[i, j-1, k-1] + 1\}$
 - 若 $X[i] \neq Y[j]$:
 - $f[i, j, 0] = \max\{f[i, j-1, k], f[i-1, j, k]\}$

13.9

$f[i, j]$ 表示以 $T[i]$ 开头和以 $T[j]$ 结尾的两个相同连续子串的长度。

- 初始化: $f[i, j] = 0$
- 状态转移: $f[i, j] = f[i+1, j-1] + 1, T[i] = T[j]$
- $ans = \max\{f[i, j]\}$

13.10

$f[i, j]$ 表示 $A[1..i]$ 和 $B[1..j]$ 的最短公共超序列。

$$f[i, j] = \begin{cases} B[1..j], & i = 0 \\ A[1..i], & j = 0 \\ S[i-1, j-1].add(A[i]), & i, j > 0 \& A[i] = B[j] \\ S[i, j-1].add(B[j]), & i, j > 0 \& A[i] \neq B[j] \& ||S[i, j-1]|| < ||S[i-1, j]|| \\ S[i-1, j].add(A[i]), & i, j > 0 \& A[i] \neq B[j] \& ||S[i, j-1]|| \geq ||S[i-1, j]|| \end{cases}$$

13.11

1) 不正确。 $X = \langle ABC \rangle, Y = \langle BACA \rangle, Z = \langle A_X B_Y A_Y C_Y B_X A_Y C_X \rangle, Z' = \langle ABAC \rangle$

2) $f[i, j]$ 为 $X[1..i]$ 和 $Y[1..j]$ 能否合成为 $Z[1..i+j]$ 。

- $f[i, j] = (f[i-1, j] \& \& Z[i+j] = X[i]) \vee (f[i, j-1] \& \& Z[i+j] = Y[j])$

3) $f[i, j, k]$ 为 $X[1..i]$ 和 $Y[1..j]$ 合成为 $Z[1..k]$ 需要删除的元素集合。

$$f[i, j, k] = \min \begin{cases} f[i-1, j, k-1], & Z[k] = X[i] \\ f[i, j-1, k-1], & Z[k] = Y[j] \\ f[i-1, j, k].add(X[i]) \\ f[i, j-1, k].add(Y[j]) \\ f[i, j, k-1].add(Z[k]) \end{cases}$$

13.12

1) $f[i]$ 表示 $s[1..i]$ 是否合法。

- $f[i] = \text{True} \text{ iff } \exists j < i, f[j] \& \& \text{dict}(s[j+1..i])$

2) $f[i]$ 表示 $s[1..i]$ 的最后一个合法单词的起始位置-1。

- $f[i] = j \text{ iff } f[j] > 0 \& \& \text{dict}[j+1..i]$
- 递归求解合法单词序列: $\text{getSeq}(n) = \text{getSeq}(f[n]) + S[f[n] + 1..n]$

13.13

1) $f[i, j]$ 表示 $S[i..j]$ 的最长回文子序列长度。

- 初始化: $f[i, j] = 0$
- 状态转移: $f[i, j] = \max\{f[i+1, j], f[i, j-1], f[i+1, j-1] + 2I\{S[i] = S[j]\}\}$

2)

- 先根据1) 求出 $f[i, j]$ 。
- $s[i]$ 表示 $S[1..i]$ 可以拆分的最少回文数量。
 - 初始化: $s[0] = 0, s[i] = \infty (i > 0)$
 - 状态转移: $s[i] = \min_{j=0}^{i-1} \{s[j] + 1\}, f[j+1][i] = i - j$

时间复杂度和空间复杂度都为 $O(n^2)$

13.15

1) $f[i, j]$ 表示 $x[1..i]$ 能否兑换金额 j 。

- $f[i, j] = f[i-1, j] \vee f[i, j-x_i]$

2) $f[i, j]$ 表示 $x[1..i]$ 能否兑换金额 j 。

- $f[i, j] = f[i-1, j] \vee f[i-1, j-x_i]$

3) $f[i, j, k]$ 表示 $x[1..i]$ 能否兑换金额 j , 且使用不超过 k 枚硬币。

- $f[i, j, k] = f[i-1, j, k] \vee f[i, j-x_i, k-1]$
- 时间复杂度为 $O(nvk)$

13.16

$f[i]$ 表示顶点 i 所在子树的最小顶点覆盖的大小, 且顶点 i 在该最小顶点覆盖中。

$g[i]$ 表示顶点 i 所在子树的最小顶点覆盖的大小, 且顶点 i 不在该最小顶点覆盖中。

- 状态转移：
 - $f[i] = 1 + \sum_{parent[j]=i} \min\{f[j], g[j]\}$
 - $g[i] = \sum_{parent[j]=i} f[j]$

时间复杂度为 $O(n)$

13.18

$f[i]$ 表示到第 i 个旅店的最小惩罚。

- 状态转移： $f[i] = \min_{j=1}^{i-1} \{f[j] + (200 - (a_i - a_j))^2, a_i - a_j < 200\}$

更新 $f[i]$ 时记录每个旅店最小惩罚对应的前一个旅店位置 $last[i]$

13.23

$f[i]$ 表示顶点 i 所在子树的友好度评分总和最大值，且顶点 i 在名单中。

$g[i]$ 表示顶点 i 所在子树的友好度评分总和最大值，且顶点 i 不在名单中。

- 状态转移：
 - $f[i] = \sum_{parent[j]=i} g[j]$
 - $g[i] = \sum_{parent[j]=i} \max\{f[j], g[j]\}$

13.24

换个角度思考，想象是两个骑手。

$f[i, j]$ 表示对前 i 个店划分后的最短路程， $i > j$ ，且第一个骑手以 i 店为结尾，第二个骑手以 j 店为结尾。

- 初始化： $f[1, 1] = 0, f[2, 1] = dist(1, 2)$
- 状态转移：
 - $f[i + 1, j] = f[i, j] + dist(i, i + 1), j = 1, 2, \dots, i - 1$
 - $f[i + 1, i] = \min_{j=1}^{i-1} \{f[i, j] + dist(j, i + 1)\}$

本质上是讨论 p_{i+1} 分配给哪个骑手。

时间复杂度为 $O(n^2)$

P7

20.1

1)

- CLIQUE
 - 优化问题：输入无向图，输出最大团大小。
 - 判定问题：输入无向图和 k ，判断图中是否有大小为 k 的团。
- KNAPSACK
 - 优化问题：输入 n 个物品、每个物品大小和价值、背包大小，输出背包能装物品的最大价值。
 - 判定问题：输入 n 个物品、每个物品大小和价值、背包大小和价值 k ，输出背包能否装价值不小于 k 的物品。
- INDEPENDENT-SET
 - 优化问题：输入无向图，输出最大独立集。
 - 判定问题：输入无向图和 k ，输出图中是否存在大小为 k 的独立集。
- VERTEX-COVER
 - 优化问题：输入无向图，输出最小点覆盖。
 - 判定问题，输入无向图和 k ，输出图中是否存在大小为 k 的点覆盖。

2)

- CLIQUE
 - 优化 \Rightarrow 判定：多项式内解出优化问题，再把优化结果和 k 比较即可。
 - 优化 \Leftarrow 判定：对 k 的所有取值进行判定，即可解决优化问题，因为 k 不超过 $|V|$ ，故还是多项式时间。
- KNAPSACK
 - 优化 \Rightarrow 判定：多项式内解出优化问题，再把优化结果和 k 比较即可。
 - 优化 \Leftarrow 判定：对 k 的所有取值进行判定，即可解决优化问题，因为 k 不超过价值总和，故还是多项式时间。
- INDEPENDENT-SET
 - 优化 \Rightarrow 判定：多项式内解出优化问题，再把优化结果和 k 比较即可。
 - 优化 \Leftarrow 判定：对 k 的所有取值进行判定，即可解决优化问题，因为 k 不超过 $|V|$ ，故还是多项式时间。
- VERTEX-COVER
 - 优化 \Rightarrow 判定：多项式内解出优化问题，再把优化结果和 k 比较即可。
 - 优化 \Leftarrow 判定：对 k 的所有取值进行判定，即可解决优化问题，因为 k 不超过 $|V|$ ，故还是多项式时间。

3)

- CLIQUE
 - 给定无向图和 k ，验证一个点集其是不是大小为 k 的团。
 - 验证完全图 $O(V^2)$ ，验证点数为 k ， $O(V)$ 。
 - 所以是NP问题。
- KNAPSACK
 - n 个物品、每个物品大小和价值、背包大小、价值 k ，验证一个物品集合是不是能放入背包且价值不小于 k 。

- 验证大小之和不大于背包容量 $O(n)$, 验证价值之和不小于 k 也是 $O(n)$
- 所以是NP问题。
- INDEPENDENT-SET
 - 给定无向图和 k , 验证一个点集其是不是大小为 k 的点独立集。
 - 验证点和点之间是否相邻 $O(V^2)$, 验证点数为 k , $O(V)$ 。
 - 所以是NP问题。
- VERTEX-COVER
 - 给定无向图和 k , 验证一个点集其是不是大小为 k 的点覆盖。
 - 验证每个边的两个端点是不是至少有一个在点覆盖中 $O(EV)$, 验证点数为 k , $O(V)$ 。
 - 所以是NP问题。

20.2

假设 $P \leq_P Q$ 且 $Q \leq_P R$ 。

- $P \leq_P Q$: 存在一个转换函数 $T_1(x)$, 且 $T_1(x)$ 为多项式时间, 使得对于 P 的合法输入 x , 转换为 $T_1(x)$ 是 Q 的合法输入, 且两者输出相同。
- $Q \leq_P R$: 存在一个转换函数 $T_2(x)$, 且 $T_2(x)$ 为多项式时间, 使得对于 Q 的合法输入 x , 转换为 $T_2(x)$ 是 R 的合法输入, 且两者输出相同。

对于 P 的合法输入 x , $T_1(x)$ 为 Q 的合法输入, $T_2(T_1(x))$ 是 R 的合法输入, 且 $T_2(T_1(x))$ 为多项式时间, 三者输出相同。

则 $T_2(T_1(x))$ 为 $P \leq_P R$ 的转换函数。

所以 \leq_P 是一个传递关系。

20.3

$$O(n^2) + O(n^4) = O(n^4)$$

20.4

- 如果P问题能在多项式时间内归约到Q问题, 说明P问题可以通过多项式时间的计算以及多项式次黑盒的调用Q问题的算法来解决。
- 排序问题多项式归约到选择问题, 排序算法可以是 n 次调用选择算法。
- 选择问题多项式归约到选择问题, 选择算法可以是调用1次排序算法然后遍历一遍完成选择。

20.5

更一般的, 我们把问题求解阶为 p 的数归约到求解阶为 q 的数。即求解阶为 q 的算法 (简称 Q 算法) 解决求解阶为 p 的算法 (简称 P 算法) 。

- 第一步, 调用一次 Q 算法, 找到第 q 小的数 t 。
- 第二步, 比较 p 和 q 的关系, 再遍历一遍数组并和 t 进行比较, 删除不可能是阶为 p 的数。当删到只剩一个数时算法停止, 输出该数。否则继续第三步。
- 第三步, 更新 p 的值, 返回第一步。

第二步至少删一个数, 所以调用 $O(n)$ 次算法即可实现 p 算法。

p 是一个变动的值, q 是一个固定值, 因为 Q 算法是已知算法。

如果还剩余 k 个元素 ($k < q$), 此时 q 算法无法调用, 但可用 $O(q^2)$ 的代价额外处理即可。

21.1

- 1) 如果任意一个NP完全问题可以在多项式时间内解决, 则所有NP问题都可以在多项式时间内归约到NP完全问题, 从而多项式时间内解决。
- 2) 证逆否命题: 如果存在一个NP完全问题可能在多项式时间内解决, 则所有NP完全问题都存在多项式时间的解。(因为NPC问题也是NP问题, 根据1) 的结论可以直接得出)

21.3

- 1) 从 n 个点中选 k 个点, 验证 k 个点是否是团。 $O(\binom{n}{k}) = O(\frac{n!}{k!(n-k)!}) = O(n^k)$
- 2) 伪最大团问题和最大团问题的区别在于 k 的性质:
 - 伪最大团问题 k 是一个常数, 不依赖于 n ;
 - 最大团问题 k 是一个参数, 依赖于 n 。

所以该多项式算法不能证明 $P=NP$ 。

21.4

- 1) 对于一个析取范式 $C_1 \vee C_2 \vee \dots \vee C_n$ 只需要一项 C_i 为真即可。
 - 当 C_i 为真时当且仅当 $\forall l \in C_i, \neg l \notin C_i$ 。
 - 多项式时间内可以检查一个项是否为真, 即平方级别代价遍历即可。
- 2) CNF-SAT的输入转化为DNF-SAT的输入这一步暂未发现多项式级别算法, 转换函数 $T(x)$ 不是多项式时间的。

21.5

- 1) 子集和问题: 给定自然数 S 和 $A = \{s_1, s_2, \dots, s_n\}$ 。
 - 令背包大小为 S , 且 $k = S$, 物品大小和价值都是 $A = \{s_1, s_2, \dots, s_n\}$, 使用背包问题的算法, 输出是否存在大小和不超过 S , 且价值和不低于 $k = S$ 的放法。
 - 输出结果就是子集和问题的结果。
- 2) $f[i, j]$ 表示前 i 个物品装进大小为 j 背包的最大价值。
 - 状态转移: $f[i, j] = \max\{f[i-1, j], f[i-1, j - \text{weight}[i]] + \text{value}[i]\}$
- 3) 时间复杂度和空间复杂度都是 $O(nW)$ 。
 - 时间复杂度讲的是算法相对于输入规模的代价。
 - 输入规模的代价指的是输入的二进制编码的长度。
 - 输入规模, 其实是 $\log W + \sum_{i=1}^n \log \text{weight}[i] + \sum_{i=1}^n \log \text{value}[i] = O(n \log W)$ 。
 - $O(nW)$ 相对于 $O(n \log W)$ 是指数级别的, 因此不代表找到了一个多项式时间的算法。

21.6

- 先证明SET-COVER是NP问题:
 - 对于一个给定解, 判定其大小是否为 k , 再检查全集中每个元素是否都在这个解中, 多项式时间代价。
- 再证明DOMINATION-SET可以多项式归约到SET-COVER:
 - 对于一个DOMINATION-SET, 给定无向图 G , 令全集 U 为顶点集合 V , 对于每个顶点 v_1, v_2, \dots, v_n , 将其和其所有邻居为子集, 也就是 S_1, S_2, \dots, S_n , 调用集合覆盖算法, 判定是否存在大小为 k 的集合覆盖。
 - 集合覆盖算法的判定结果就是支配集问题的判定结果, 即是否存在大小为 k 的支配集。
- 所以SET-COVER是NPC问题。