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

=>：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。

<=：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 <=> v至少有2个子节点x和y。

=>：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。

<=：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

  =>：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。

  <=：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$i \to j$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$\overline{x}$，则有路径$x \to v_1 \to v_2 \to ... \to \overline{x}$和$\overline{x} \to u_1 \to u_2 \to ... \to x$
- 若x取TRUE，则v_1要取TRUE，v_2也要取TRUE……$\overline{x}$也要取TRUE，矛盾。
- 若x取FALSE，则$\overline{x}$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$\overline{x}$的拓扑序：
    - $x < \overline{x}$，即可能$x \to \overline{x}$，x取FALSE。
    - $x = \overline{x}$，无解。
    - $x > \overline{x}$，即可能$\overline{x} \to x$，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。

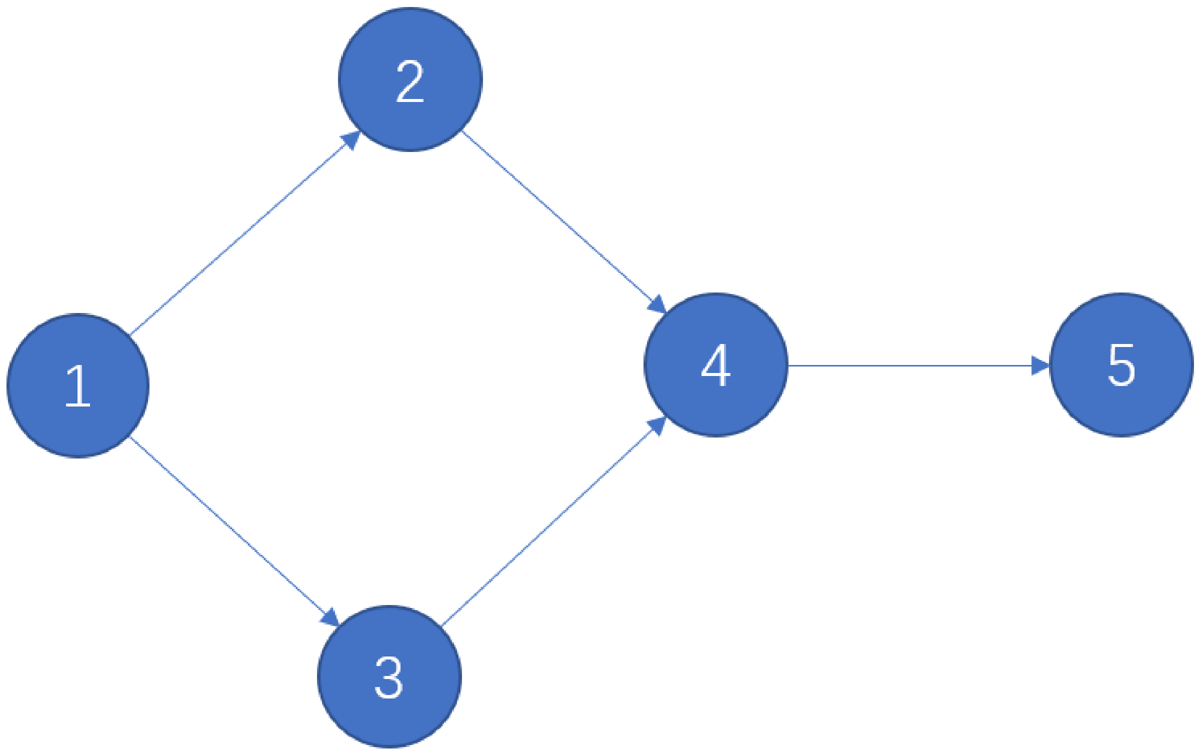
因为每个变量的真假都有确定的拓扑序，所以一定能按照这个规则确定真假值，并且没有冲突。

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的为访问过的邻居，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，需要额外判断是否是有共同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-DEGGREE-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：$O(n^2 logn)$
  - 堆实现优先队列-稀疏图：$O(nlogn)$
- Kruskal
  - 稠密图：$O(n^2 logn)$
  - 稀疏图：$O(nlogn)$
- 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重。
  - 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

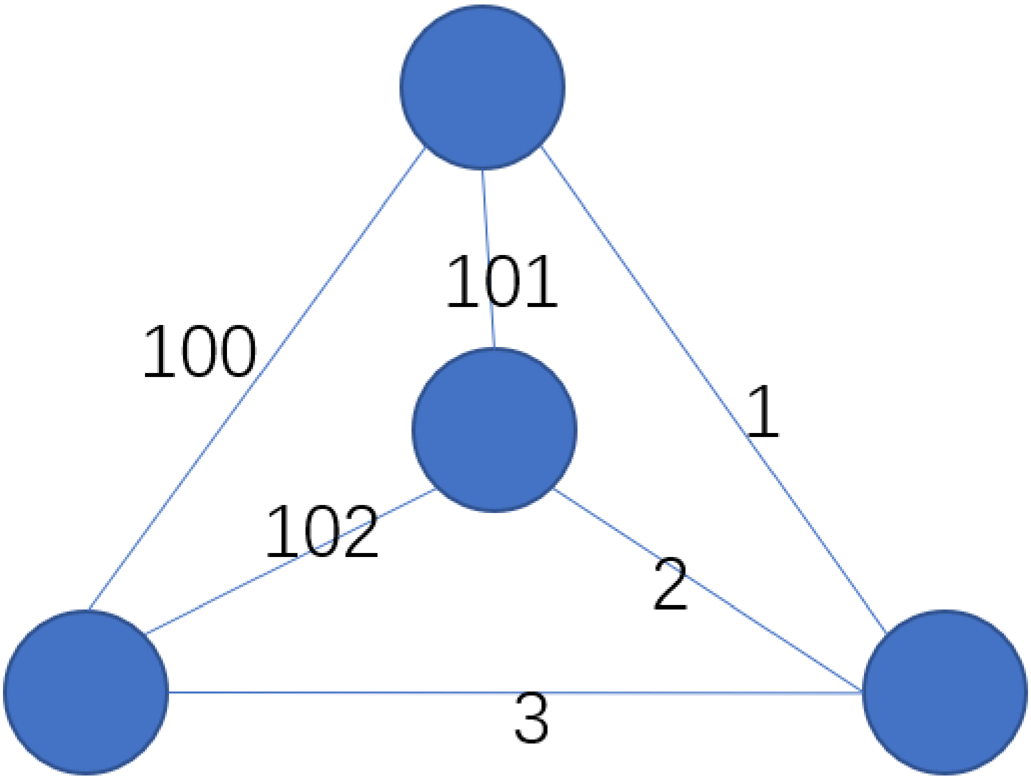
1）错。反例：桥

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

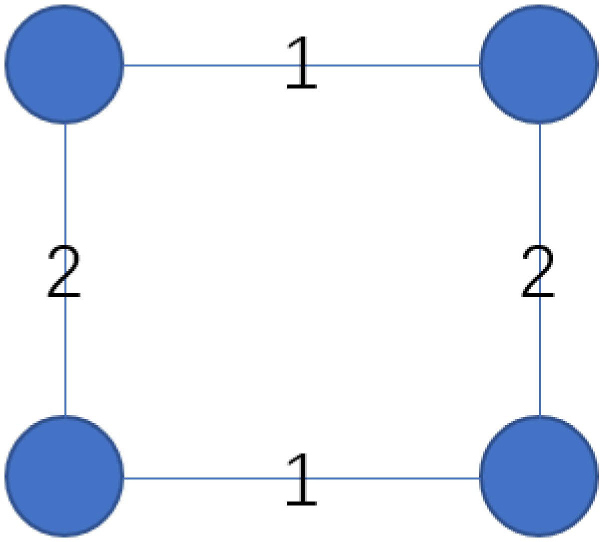
两个图有相同的边权大小序，且由于边权各不相同，MST唯一。

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$e'.weight \le e.weight$。
- 但是e也是G的MST，所以$e.weight \le e'.weight$。
- 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$(a_{n-1}, \ldots, a_1, a_0)$，假设有更优的做法，得出的是$(a'_{n-1}, \ldots, a'_1, a'_0)$。

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

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

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

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

  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：
  - $f[i,j] = f[i-1,j]\ if\ s_i > j$
  - $f[i,j] = f[i-1,j] || f[i-1,j-s_i],\ if\ s_i \leq j$

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$O(knlogn)$
- $f[*,j]$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字串的长度。

- 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。

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

13.11

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

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

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

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。

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

2）$f[i]$表示$s[1..i]$的最后一个合法单词的起始位置-1。

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

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$S[1..i]$可以拆分的最少会问数量。
  - 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。

- $f[i,j] = f[i-1,j] || f[i, j-x_i]$

2）$f[i,j]$表示$x[1..i]$能否兑换金额j。

- $f[i,j] = f[i-1,j] || f[i-1, j-x_i]$

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。

- $f[i,j,k] = f[i-1,j,k] || f[i, j-x_i, k-1]$
- 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：
  - $f[i] = 1 + \Sigma_{parent[j]=i} \min\{f[j], g[j]\}$
  - $g[i] = \Sigma_{parent[j]=i} f[j]$

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：$f[i] = \min\limits_{j=1}^{i-1}\{f[j] + (200 - (a_i - a_j))^2, a_i - a_j < 200\}$

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：
  - $f[i] = \Sigma_{parent[j]=i} g[j]$
  - $g[i] = \Sigma_{parent[j]=i} \max\{f[j], g[j]\}$

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：
  - $f[i+1, j] = f[i, j] + dist(i, i+1), j = 1, 2, \ldots, i-1$
  - $f[i+1, i] = \min_{j=1}^{i-1}\{f[i, j] + dist(j, i+1)\}$

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=>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<=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=>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<=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=>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<=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=>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<=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$P \le_P Q$且$Q \le_P R$。

- $P \le_P Q$：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。
- $Q \le_P R$：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$P \le_P R$的转换函数。

所以$\le_P$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，验证者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$C_1 \vee C_2 \vee \cdots \vee C_n$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$A = \{s_1, s_2, \cdots, s_n\}$。

- 令背包大小为S，且$k = S$，物品大小和价值都是$A = \{s_1, s_2, \cdots, s_n\}$，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：$f[i, j] = \max\{f[i-1, j], f[i-1, j - weight[i]] + value[i]\}$

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$logW + \Sigma_{i=1}^{n} log\, weight[i] + \Sigma_{i=1}^{n} log\, value[i] = O(nlogW)$。
- $O(nW)$相对于$O(nlogW)$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$v_1, v_2, \cdots, v_n$，将其和其所有邻居为子集，也就是$S_1, S_2, \cdots, S_n$，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。