Problem 1

(a) $f = \Omega(g)$

(b) $f = \mathrm{O}(g)$

(c) $f = \Theta(g)$

Problem 2

Suppose $G = (V, E)$

1. Suppose we have G is connected and G does not contain a cycle.

   We can prove by induction that a connected graph G with n nodes has at least $n - 1$ edges.

   When $n = 1$, it is true with no edges.

   When $n > 1$, when we add a new node to the previous set, we need to add an edge to connect the new node to previous nodes in the set. So the number of the edges increase by 1. So, the number of edges m of G have the constraint $m \geq n - 1$.

   Suppose we have a connected gragh G' has $n - 1$ edges. And we need to add at least 1 edges to form the graph G. For $\forall\ u, v \in V$, there exist a path $v - w$, if we add an edge $e = (u, v)$. e and path $v - w$ will form a circle, which is contradictory with G does not contain a cycle.

   So, G can only be G' with $n - 1$ edges.

2. Suppose we have G is connected and G has $n - 1$ edges.

   We can prove G does not contain a cycle by induction.

   When $n = 1$, a graph with a single node does not contain a cycle.

   When $n > 1$, the new node v we add to the previous set must connected to one of the nodes before. Since v is not connected to the previous set of nodes before we add that edge, any path from v to other nodes must through that edge. v cannot form any cycle with previous nodes because there is not any other edges back to v.

   Thus, G does not contain a cycle.

3. Suppose G does contain a cycle and G has $n - 1$ edges.

   We can prove G is connected by induction.

   When $n = 1$, a graph with single node is connected.

   When we add a new node to the previous set of nodes. Suppose we can add new two edges to the graph connected the new vertex x and u, v in the previous set of nodes perspectively. Since the previous graph is connected, there exist a path $u - v$ in the previous graph. But the new two edges $(x, u), (x, v)$ and path $u - v$ form a cycle which is contradictory to G does not contain a cycle.

   So we can only add a single edge every time we add a new node. The new edge connects the new node x to one of the previous node v. Since the previous graph is connected, there exist a path between v and any other nodes. x can access any other nodes by through the edge (x, v) and the path from v to that node. Thus, x can access any other nodes in the new graph.

   Thus, G is connected.

Problem 3

Algorithm: We use $length(u, v)$ to represent the length of path $u - v$

$Find\_Diameter(T)$:
$BFS\ from\ random\ v \in V,\ dis_i \leftarrow length(i, v)$
$p \leftarrow the\ point\ v\ with\ MAX(dis_v)$
$BFS\ from\ p,\ dis_i \leftarrow length(i, p)$
$return\ MAX_{v \in V} dis_v$

To prove the correctness of this algorithm, we need to prove that p we got at the first BFS is located at one end of the diameter of the tree. Suppose the diameter of the tree is path $s - t$. Suppose that p is neither s nor t. And the point we start our first BFS is v. We have the following situations:

1. If v is on the path $s - t$. Suppose p need access t through v (Otherwise, we can swap s and t.)

   $$\begin{aligned} & length(v, s) < length(v, p) \\ \Rightarrow\ & length(v, s) + length(v, t) < length(v, p) + length(v, t) \\ \Rightarrow\ & length(s, t) < length(p, t) \end{aligned}$$

   $length(s, t) < length(p, t)$ is contradictory to our assumption that $s - t$ is the diameter.

2. If v is not on the path $s - t$. And path $v - p$ and path $s - t$ have no common nodes.

   1. Suppose v need to access to s or t via node x on the path $s - t$. And do not need to through the nodes on path $v - p$.

      $$\begin{aligned} & length(v, s) < length(v, p) \\ \Rightarrow\ & length(v, s) + length(x, t) - length(x, v) < length(v, p) + length(x, t) + length(x, v) \\ \Rightarrow\ & length(s, t) < length(p, t) \end{aligned}$$

      Which is contradictory to the assumption.

   2. Suppose v need to access to s or t via node x on the path $s - t$. And need to through the node x' on path $v - p$.

      $$\begin{aligned} & length(v, s) < length(v, p) \\ \Rightarrow\ & length(x', s) < length(x', p) \\ \Rightarrow\ & length(x', s) - length(x, x') < length(x', p) + length(x', x) \\ \Rightarrow\ & length(x, s) < length(x, p) \\ \Rightarrow\ & length(x, s) + length(x, t) < length(x, p) + length(x, t) \\ \Rightarrow\ & length(s, t) < length(p, t) \end{aligned}$$

      Which is contradictory to the assumption.

3. If v is not on the path $s - t$. And path $v - p$ and path $s - t$ have common nodes.

   Suppose v need to access s through the first node x on path $s - t$.

   $$\begin{aligned} & length(v, s) < length(v, p) \\ \Rightarrow\ & length(v, s) - length(x, v) + length(x, t) < length(v, p) - length(x, v) + length(x, t) \\ \Rightarrow\ & length(s, t) < length(p, t) \end{aligned}$$

   Which is contradictory to the assumption.

We can draw the conclusion that p is either s or t in this situation. So, the algorithm can produce the correct answer.

Problem 4

We use dis_i to represent the current shortest path from s to i.

```
Current Point is: s
    dis[a] <- 1
    dis[c] <- 2
    dis[e] <- 2
Current Point is: a
    dis[b] <- 2
Current Point is: b
    dis[v] <- 5
Current Point is: c
    dis[d] <- 3
    dis[f] <- 3
Current Point is: e
Current Point is: d
Current Point is: f
    dis[g] <- 9
Current Point is: v
The final result is dis[v] = 5
```

Problem 5

$dis[x]$ represent the mininum length from v to x, $count[x]$ represent the number of shortest path from x. $N_{(x)}$ represent the set of vertexes adjacent to x. $length(u, v)$ represent the length of the edge (u, v)

$$
\begin{aligned}
&FindNumberofShortestPath(G(V,E),v,w):\\
&\quad S\leftarrow\{v\}, dis[v]\leftarrow 0\\
&\quad for\ integer\ i\leftarrow 1\ to\ |V|-1\\
&\qquad let\ v\leftarrow x,\ x\in V\ \wedge\ x\notin S, dis[x]=MIN_{i\in V\wedge i\notin S}\ dis[i]\\
&\qquad add\ v\ to\ S\\
&\qquad for\ node\ u\in N_{(v)}\\
&\qquad if\ dis[u]=dis[v]+length(v,u)\ then\\
&\qquad\quad count[u]\leftarrow count[u]+count[v]\\
&\qquad else\ if\ dis[u]>dis[v]+length(v,u)\ then\\
&\qquad\quad dis[u]\leftarrow dis[v]+length(u,v), count[u]\leftarrow count[v]\\
&\quad return\ count[w]
\end{aligned}
$$

Note that the algorithm is adapted from Dijkstra's algorithm. Thus the running time of the algorithm is $O(n^2)$

Problem 6

- Algorithm:
  - Sort all the items by the cost-effectiveness($\frac{v_i}{w_i}$) with decreasing order.
  - Try to take all the items with larger cost-effectiveness. Until W is not enough to take the whole next item.
  - Take fraction of the next item until reach the weight bound of W.
- Prove:
  - Suppose the final weight of the item we choose is w_i' respective.

- The final total value V we get can be represent by $V = \sum w_i' \times \frac{v_i}{w_i}$
- Suppose we take out part of w_i' we chose and swap some item j we did not choose. Suppose this part is $p, p \leq w_i'$
- The new total value $V' = V - p \times \frac{v_i}{w_i} + p \times \frac{v_j}{w_j} = V - p \times (\frac{v_i}{w_i} - \frac{v_j}{w_j})$
- Because we sorted the original items by $\frac{v_i}{w_i}$. $\frac{v_i}{w_i} - \frac{v_j}{w_j} \geq 0$.
- Thus $V' \leq V$. So, we can prove that our V is maximal.

Problem 7

Code:

```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int,int> PII;
const int N=15;
int c[N],pre[N];
int n,tot;
bool vis[N];
vector<PII> e[N],ans;
inline char Change(int x)
{
    if(x==1) return 's';
    if(x==n) return 'v';
    else return x-2+'a';
}
inline void Prim()
{
    c[1]=0;
    for(int i=1;i<=n;i++)
    {
        int v=0;
        for(int j=1;j<=n;j++)
        {
            if(vis[j]) continue;
            if(!v||c[v]>c[j]) v=j;
        }
        if(v!=1) ans.push_back({pre[v],v}),tot+=c[v];
        vis[v]=true;
        cout<<"Current Point is: "<<v<<endl;
        for(auto u : e[v])
            if(c[u.first]>u.second)
            {
                c[u.first]=u.second,pre[u.first]=v;
                cout<<"\tc["<<u.first<<"] <- "<<u.second<<endl;
            }
    }
}
int main()
{
    memset(c,0x3f,sizeof(c));
    n=12;
    e[1].push_back({2,5}),e[1].push_back({5,5});
```

```
e[2].push_back({1,5}),e[2].push_back({3,2}),e[2].push_back({5,4}),e[2].push_back({7,7});
    e[3].push_back({2,2}),e[3].push_back({4,4}),e[3].push_back({7,1});
    e[4].push_back({3,4}),e[4].push_back({7,3}),e[4].push_back({8,6});
e[5].push_back({1,5}),e[5].push_back({2,4}),e[5].push_back({6,3}),e[5].push_back({9,1});
    e[6].push_back({5,3}),e[6].push_back({7,2}),e[6].push_back({10,1});
e[7].push_back({2,7}),e[7].push_back({3,1}),e[7].push_back({4,3}),e[7].push_back({6,2}),e[7].push_back({8,8}),e[7].push_back({10,9}),e[7].push_back({11,1});
e[8].push_back({4,6}),e[8].push_back({7,8}),e[8].push_back({11,1}),e[8].push_back({12,4});
    e[9].push_back({5,1}),e[9].push_back({10,5});
e[10].push_back({6,1}),e[10].push_back({7,9}),e[10].push_back({9,5}),e[10].push_back({11,2});
e[11].push_back({7,1}),e[11].push_back({8,1}),e[11].push_back({10,2}),e[11].push_back({12,8});
    e[12].push_back({8,4}),e[12].push_back({11,8});
    Prim();
    cout<<"The final spanning tree is: ";
    for(auto x : ans) cout<<"("<<x.first<<' '<<x.second<<"), ";
    cout<<endl;
    cout<<"The minimum cost is: "<<tot<<endl;
    return 0;
}
```

Result:

```
Current Point is: 1
    c[2] <- 5
    c[5] <- 5
Current Point is: 2
    c[3] <- 2
    c[5] <- 4
    c[7] <- 7
Current Point is: 3
    c[2] <- 2
    c[4] <- 4
    c[7] <- 1
Current Point is: 7
    c[3] <- 1
    c[4] <- 3
    c[6] <- 2
    c[8] <- 8
    c[10] <- 9
    c[11] <- 1
Current Point is: 11
    c[8] <- 1
    c[10] <- 2
    c[12] <- 8
Current Point is: 8
    c[12] <- 4
```

```
Current Point is: 6
    c[5] <- 3
    c[10] <- 1
Current Point is: 10
    c[6] <- 1
    c[9] <- 5
Current Point is: 4
Current Point is: 5
    c[9] <- 1
Current Point is: 9
    c[5] <- 1
Current Point is: 12
The final spanning tree is: (1 2), (2 3), (3 7), (7 11), (11 8), (7 6), (6 10), (7 4), (6 5), (5 9), (8 12),
The minimum cost is: 24
```