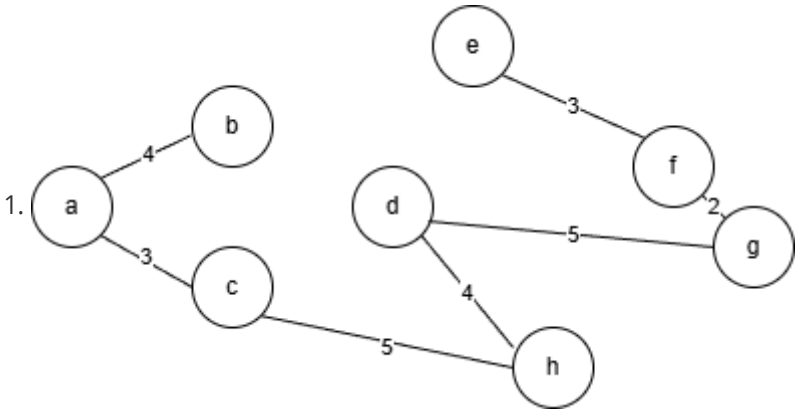
HW6

一. 选择题

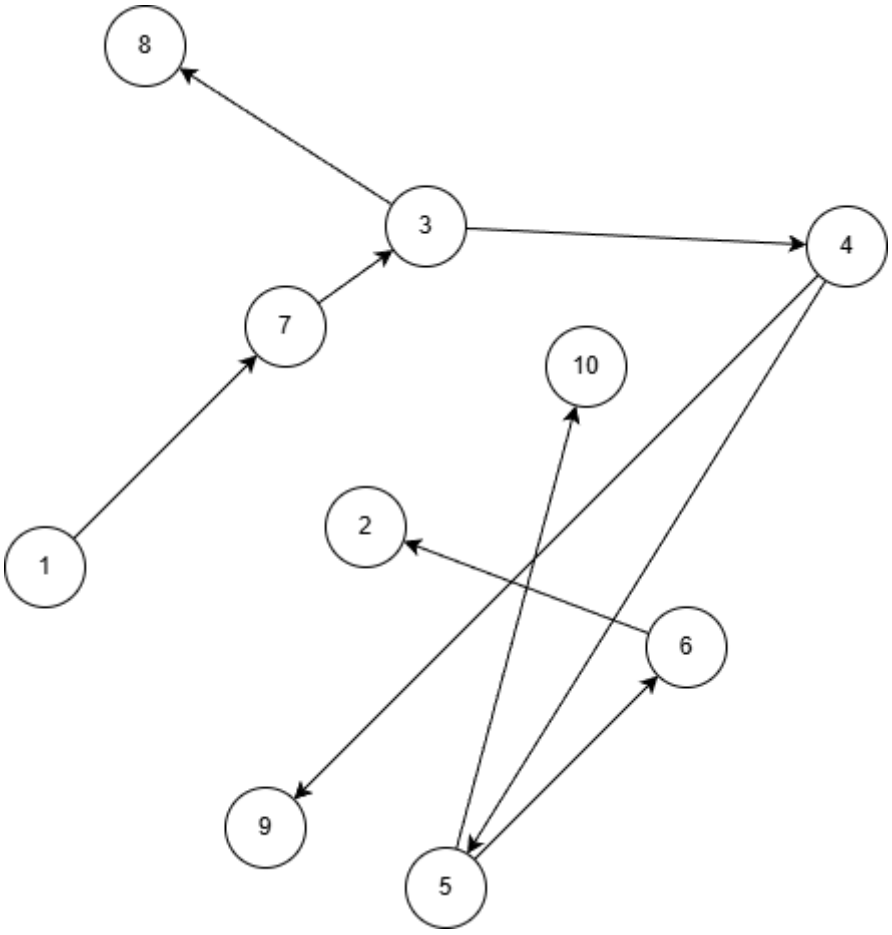
1. D，D;

二. 应用题

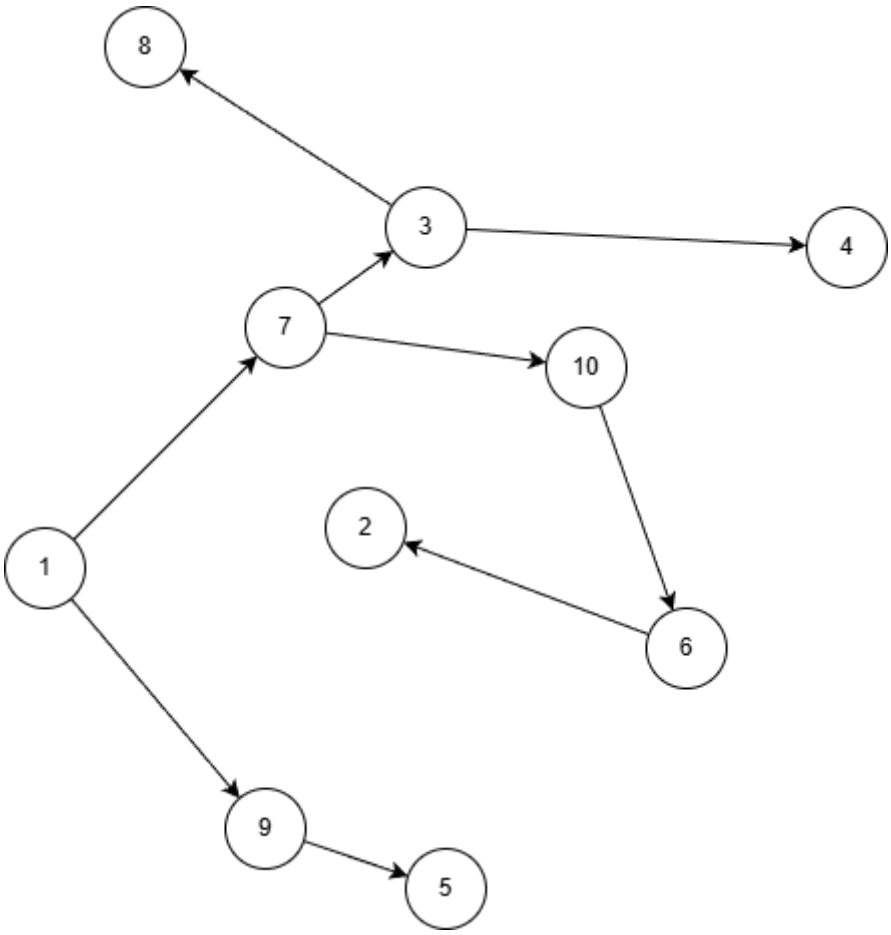
1.



2.
   - 深度优先生成树：



   - 广度优先生成树：



3.

终点	i = 1	i = 2	i = 3	i = 4	i = 5	i = 6
b	15 (a, b)	15 (a, b)	15 (a, b)	15 (a, b)	15 (a, b)	**15** **(a, b)**
c	**2** **(a, c)**					
d	12 (a, d)	12 (a, d)	11 (a, c, f, d)	**11** **(a, c, f, d)**		
e	∞	10 (a, c, e)	**10** **(a, c, e)**			
f	∞	**6** **(a, c, f)**				
g	∞	∞	16 (a, c, f, g)	16 (a, c, f, g)	**14** **(a, c, f, d, g)**	
S **终点集**	{a, c}	{a, c, f}	{a, c, f, e}	{a, c, f, e, d}	{a, c, f, e, d, g}	{a, c, f, e, d, g, b}

三. 算法设计题

1.

```
void DFS2(AGraph* G, int v)
{
    // 1. 判断是否输入合法
    if (G == NULL) return;
    if (v < 0 || v >= G->vexnum) return;

    // 2. 初始化
```

```
    bool* visited = (bool*)calloc(G->vexnum, sizeof(bool));
    if (!visited)
    {
        printf("CreateError!\n");
        return;
    }

    ArcNode** curArc = (ArcNode**)malloc(G->vexnum *
sizeof(ArcNode*));
    if (!curArc)
    {
        printf("CreateError!\n");
        free(visited);
        return;
    }
    for (int i = 0; i < G->vexnum; i++)
        curArc[i] = G->vertics[i].firstarc;

    struct Stack
    {
        int adjvex[MAX_VERTEX_NUM];
        int top;
    } stack;

    stack.top = -1;

    // 3. 核心逻辑
    printf("使用非递归方式进行 DFS 中...\n");

    printf("%d ", G->vertics[v].data);
    visited[v] = true;
    stack.adjvex[++stack.top] = v; // 栈的 push 操作

    while (stack.top != -1) // 栈不为空
    {
        int current = stack.adjvex[stack.top]; // 访问栈顶元素
        ArcNode* p = curArc[current];

        while (p != NULL && visited[p->adjvex])
            p = p->nextarc;

        curArc[current] = p;

        if (p != NULL)
        {
            int next = p->adjvex;
            printf("-> %d ", G->vertics[next].data);
            visited[next] = true;
            stack.adjvex[++stack.top] = next;
            curArc[current] = p->nextarc;
        }
        else stack.top--; // 出栈操作
    }

    free(visited);
    free(curArc);
    printf("\n");
}
```

2.

```
static bool DFS_PathLenK(AGraph* G, int u, int v, InfoType k, bool visited[])
{
    if (k < 0) return false;
    if (u == v) return k == 0;

    visited[u] = true;

    ArcNode* p = G->vertics[u].firstarc;
    while(p != NULL)
    {
        int next = p->adjvex;
        InfoType weight = p->info;

        if (!visited[next])
        {
            if (DFS_PathLenK(G, next, v, k-weight, visited))
            {
                visited[u] = false;
                return true;
            }
        }
        p = p->nextarc;
    }

    // 回溯
    visited[u] = false;
    return false;
}

bool HasPathLenK(AGraph* G, int u, int v, InfoType k)
{
    // 参数检验
    if (G == NULL || u < 0 || u >= G->vexnum || v < 0 || v >= G->vexnum)
    {
        printf("InputError!\n");
        return false;
    }

    bool* visited = (bool*)calloc(G->vexnum, sizeof(bool));
    if (!visited)
    {
        printf("CreateError!\n");
        return false;
    }

    bool result = DFS_PathLenK(G, u, v, k, visited);

    free(visited);
    return result;
}
```