数据结构作业(第十一次)

PB20111686 黄瑞轩

约定

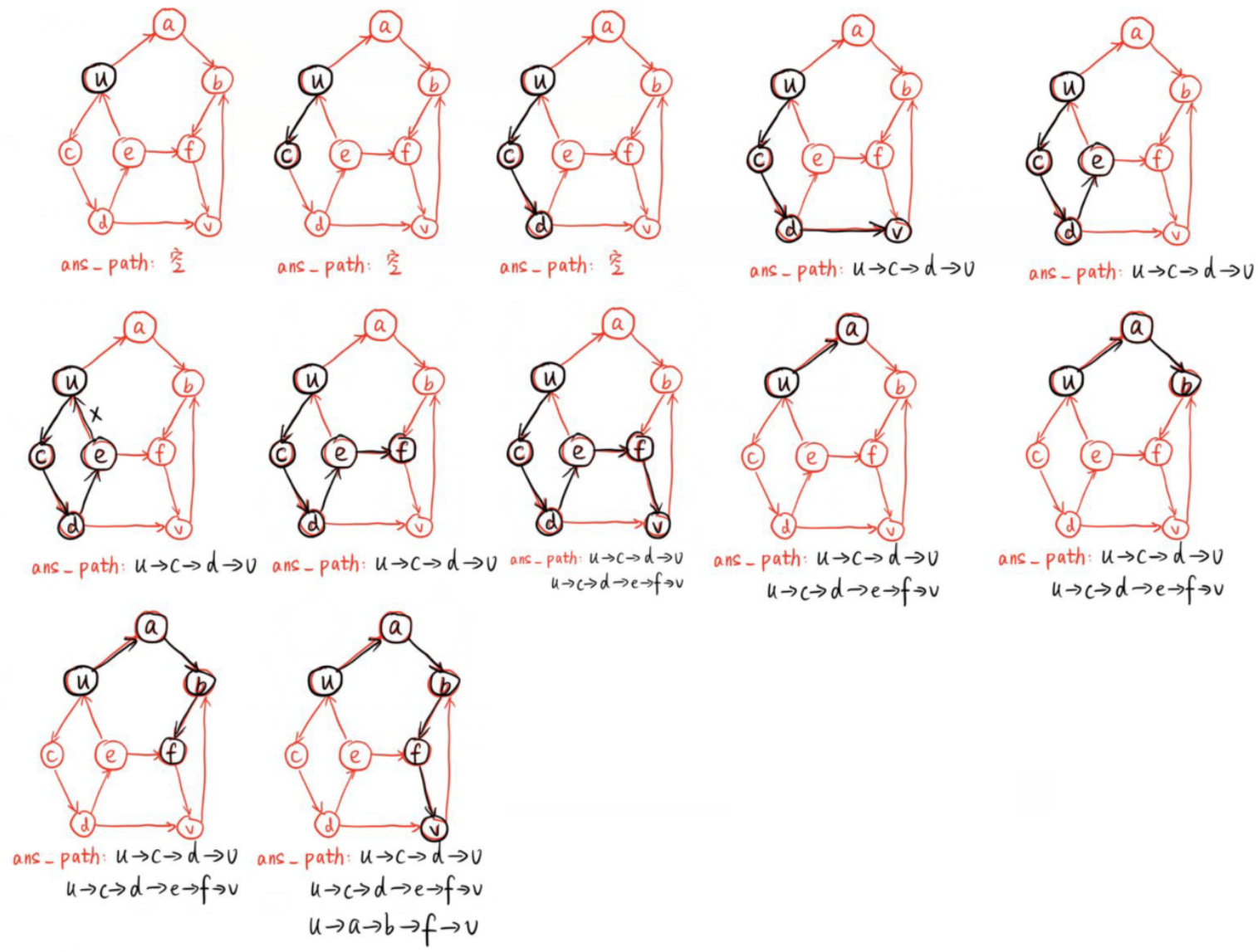
本作业中图全部采用邻接表形式存储，其定义如下：

```
#define MAX_VERTEX_NUM 20
typedef int InfoType;
typedef int VertexType;
typedef struct ArcNode {
    int adjVex;               //边指向的顶点序号
    struct ArcNode* nextArc;  //下一条边
    InfoType* info;           //边上信息
} ArcNode;
typedef struct VNode {
    VertexType data;          //顶点信息
    ArcNode* firstArc;        //第一条边指针
} VNode, AdjList[MAX_VERTEX_NUM];
typedef struct {
    AdjList vertices;
    int vexNum, arcNum;       //当前顶点数和弧数
    int kind;                 //图的种类标志
} ALGraph;
```

7.28

```
vector<vector<int>> ans_path;
bool Find(vector<int> v, int t) {
    int s = v.size();
    for (int i = 0; i < s; i++) {
        if (v[i] == t)return true;
    }
    return false;
}
void FindAllPath(ALGraph* G, int u, int v, vector<int> track) {
    if (u == v) {
        ans_path.push_back(track);
        return;
    }
    for (auto i = G->vectices[u]->firstArc; i; i = t->nextArc) {
        if (Find(track, i->adjVex))return;
        track.push_back(i->adjVex);
        FindAllPath(G, i->adjVex, v, track);
    }
    return;
}
```

手工模拟如下：



7.31

```
//下面这个算法判断u是否可达v
vector<vector<int>> ans_path;
void FindAllPath(ALGraph* G, int u, int v, vector<int> track);
//...省略算法7.28
vector<vector<int>> ans_strongly_connected_component;
string isVisited;
string null_string;
void Init(int s, string* str) {
    str->clear();
    for (int j = 0; j < s; j++)str->append("0");
}
void FindSCCByVex(ALGraph* G, int v) {
    null_string[v] = '1';
    isVisited[v] = '1';
    for (auto i = G->vectices[v]->firstArc; i; i = t->nextArc) {
        if (isVisited[i->adjVex] == '1')return;
        vector<int> par;
        par.push_back(i->adjVex);
        FindAllPath(G, i->adjVex, v, par);
        if (ans_path.size()) {
            //i->adjVex也可达v
            ans_path.clear();
            null_string[i->adjVex] = '1';
            isVisited[i->adjVex] = '1';
            FindSCCByVex(G, i->adjVex);
        }
    }
    return;
}
void FindSCC(ALGraph* G) {
    while (isVisited.find('0') != isVisited.npos) {
        Init(G->vexNum, &null_string);
        FindSCCByVex(G, isVisited.find('0'));
        ans_strongly_connected_component.push_back(null_string);
        //强连通分量以01串的形式存在答案容器中
    }
    return;
}
```

时间复杂度：算法7.28的时间复杂度是$\mathcal{O}(\nu+\varepsilon)$，最坏情况下，DFS要遍历整个图才能给出两个顶点之间是否可达，要遍历整个图才能给出一个强连通分量，最坏情况下的时间复杂度是$\mathcal{O}((\nu+\varepsilon)^2)$。

7.37

```
#define MAX MAX_INT_INST //一个合适的最大值
void LongestPath(ALGraph* G, int s)
{
    stack<int> S;
    int vn = G->vexNum;
    int dist[vn];

    bool* visited = new bool[vn];
    for (int i = 0; i < V; i++) visited[i] = false;
    // 拓扑排序，已在之前作业实现，省略
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            topologicalSort(i, visited, S);
    // 初始化路径长度
    for (int i = 0; i < V; i++) dist[i] = MAX;
    dist[s] = 0;
    while (!S.isEmpty()) {
        // 获得拓扑排序中下一个顶点
        int u = S.top();
        S.pop();
        // 更新其所有相邻顶点的最短路径
        if (dist[u] != MAX) {
            for (auto i = G->vertices[u]->firstArc; i; i=i->nextArc){
                if (dist[i->adjVex] < dist[u] + i->data)
                    dist[i->adjVex] = dist[u] + i->data;
            }
        }
    }
    // 打印最长路径
    for (int i = 0; i < V; i++)
        (dist[i] == MAX) ? cout << "NO" : cout << dist[i] << " ";
    delete [] visited;
}
```

时间复杂度：拓扑排序的时间复杂度是$\mathcal{O}(\nu+\varepsilon)$，输出关键路径的最坏时间复杂度是$\mathcal{O}((\nu+\varepsilon)^2)$。

9.26

```
int binary_search(int arr[], int start, int end, int khey) {
    if (start > end)
        return -1;
    int mid = start + ((end - start) >> 1);
    if (arr[mid] > khey)
        return binary_search(arr, start, mid - 1, khey);
    else if (arr[mid] < khey)
        return binary_search(arr, mid + 1, end, khey);
    else
        return mid;
}
```

9.28

设分区跨度为k，算法如下。

```
int block_search(int arr[], int start, int end, int khey, int k) {
    // 先找到要找的数据在哪一块
    int s = start;
    int e = end + k;
    while (e <= end) {
        if (khey >= arr[s] && khey < arr[e])break;
        s += k;
        e += k;
    }
    // 再对找到的块进行二分查找
    return e > end ? binary_search(arr, s, end, khey) : binary_search(arr, s, e, khey);
}
```

优点：节省每次查找时的判断边界的时间；

缺点：增大了空间使用；

技巧：当每一块(k)较小时再使用监视哨。