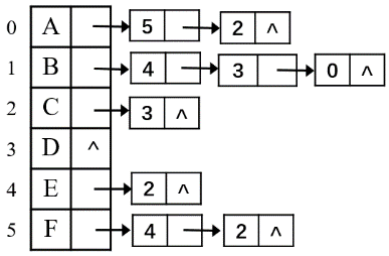
哈尔滨工业大学(深圳)2023 年春《数据结构》
第三次作业 图结构

学号	220110515	姓名	金正达	成绩	

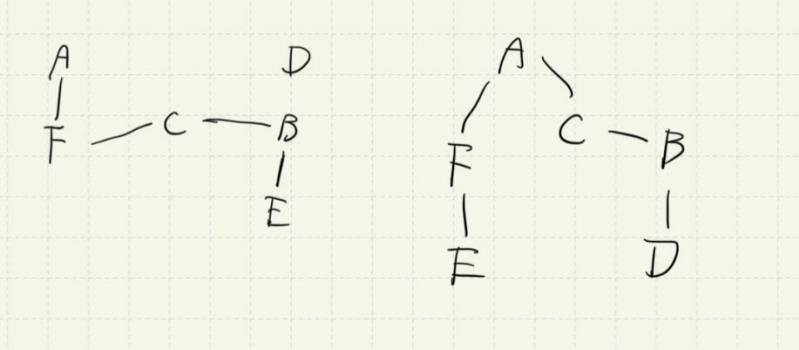
1、简答题



1-1 已知图的邻接表如图所示，给出以顶点 A 为起点的一次深度优先（先深，DFS）和广度优先（先广，BFS）的搜索序列,并给出相应的生成树/森林。

答：DFS: AFCBED

BFS: ACFBED



1-2 对一个图进行遍历可以得到不同的遍历序列，那么导致得到的遍历序列不唯一的因素有哪些?

答：

1. 节点的选择顺序，例如，对于深度优先搜索算法（DFS），在选择下一个节点时，可以按照不同的规则选择，如选择最左侧的节点或最右侧的节点。
2. 节点的访问时机，例如，广度优先搜索算法（BFS）中，可以选择在将节点加入队列时访问节点，也可以选择在将节点出队列时访问节点。
3. 图的结构，例如，对于有向图中存在环路的情况，遍历算法可能进入环路并在环路内部反复遍历，导致多个不同的遍历序列。
4. 起始节点的选择。

1-3 已知有 6 个顶点（顶点编号为 0 ~ 5）的有向带权图 *G*，其邻接矩阵 *A* 为上三角矩阵，按行为主序（行优先）保存在如下的一维数组中。

4	6	∞	∞	∞	5	∞	∞	∞	4	3	∞	∞	3	3

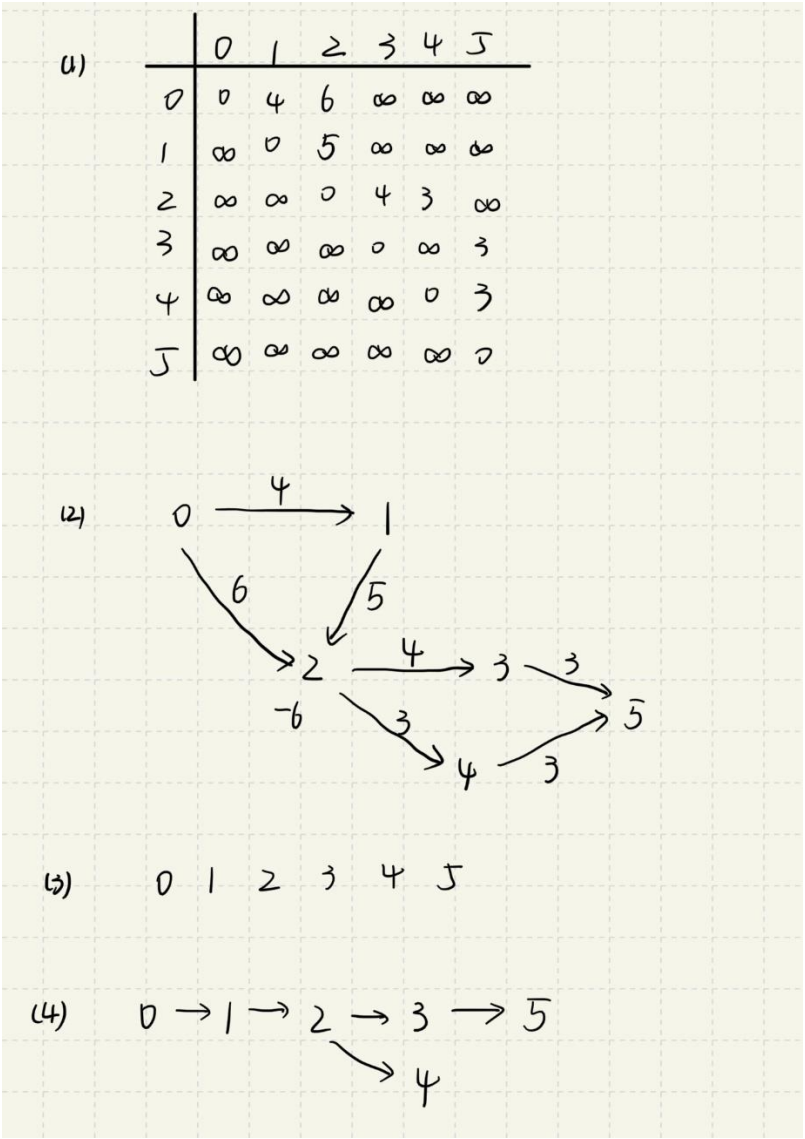
（1）写出图 *G* 的邻接矩阵 *A*；

（2）画出有向带权图 *G*；

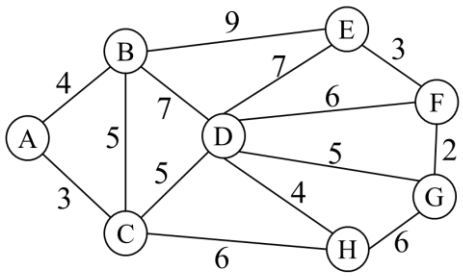
（3）给出一个拓扑序列；

（4）求图 *G* 的关键路径，并计算该关键路径的长度。

答:



1-4 无向网如下图所示，回答下列为题：



(1)按照 Prim 算法求其最小生成树，填表完成求解过程；

Prim 算法求解过程(1) ∑w=26

No.	U	V-U	本轮需考察的边及其权值	最近邻/A
1	{A}	{BCDEFGH}	AC:3,AB:4	C
2	{AC}	{BDEFGH}	AB:4,CB:5,CD:5,CH:6	B

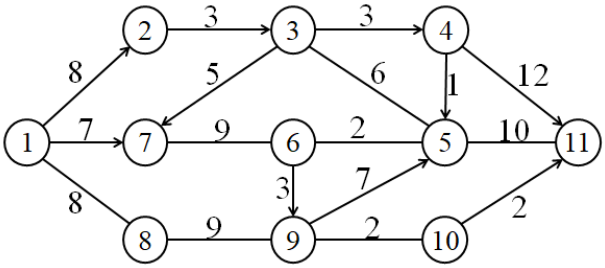
3	{ACB}	{DEFGH}	CD:5,CH:6,BD:7,BE:9	D
4	{ACBD}	{EFGH}	CH:6,BE:9,DE:7,DF:6,DG:5,DH:4	H
5	{ACBDH}	{EFG}	BE:9,DE:7,DF:6,DG:5,HG:6	G
6	{ACBDHG}	{EF}	BE:9,DE:7,DF:6,GF:2	F
7	{ACBDHGF}	{E}	BE:9,DE:7,FE:3	E
8	{ACBDHGFE}	{}	N/A	N/A

(2)按 Kruskal 算法求其最小生成树，按顺序给出边。

1	2	3	4	5	6	7
FG:2	**FE:3**	**AC:3**	**DH:4**	**AB:4**	**DG:5**	**CD:5**

注：2 和 3，4 和 5，6 和 7 位置均可互换。

1-5 求混合（有向和无向混合）图单源最短路径,填写表格完成 Dijkstra 算法各步骤。设源点为顶点①。



Dijkstra 算法求单源最短路径

No	S	w	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]	D[8]	D[9]	D[10]	D[11]
0	**{1}**	**-**	**8**	∞	∞	∞	∞	**7**	**8**	∞	∞	∞
1	**{1,7}**	**7**	**8**	∞	∞	∞	**16**	**7**	**8**	∞	∞	∞
2	**{1,7,6}**	**6**	**8**	**11**	∞	∞	**16**	**7**	**8**	**17**	∞	∞
3	**{1,7,6,2,8}**	**2,8**	**8**	**11**	∞	∞	**16**	**7**	**8**	**17**	∞	∞
4	**{1,7,6,2,8,3,9}**	**3,9**	**8**	**11**	**14**	**17**	**16**	**7**	**8**	**17**	∞	∞
5	**{1,7,6,2,8,3,9,4,5}**	**4,5**	**8**	**11**	**14**	**15**	**16**	**7**	**8**	**17**	∞	**26**
6	**{1,7,6,2,8,3,9,4,5,11}**	**11**	**8**	**11**	**14**	**15**	**16**	**7**	**8**	**17**	∞	**25**
7	**{1,7,6,2,8,3,9,4,5,11}**	**-**	**8**	**11**	**14**	**15**	**16**	**7**	**8**	**17**	∞	**25**
8	**{1,7,6,2,8,3,9,4,5,11,10}**	**10**	**8**	**11**	**14**	**15**	**16**	**7**	**8**	**17**	**19**	**25**
9	**{1,7,6,2,8,3,9,4,5,11,10}**	**-**	**8**	**11**	**14**	**15**	**16**	**7**	**8**	**17**	**19**	**21**
10												

2、算法设计

（1） 采用 C 或 C++语言设计数据结构；

（2） 给出算法的基本设计思想；

（3） 根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释；

（4） 说明你所设计算法的时间复杂度和空间复杂度。

2-1 图的存储结构实践。

自定义图的邻接矩阵和邻接表两种存储结构。以下两项任选其一：

(1) 创建图的邻接矩阵，设计算法自动生成邻接表，或：

(2) 创建图的邻接表，设计算法自动生成邻接矩阵。

要求能够打印图的邻接矩阵和邻接表，进行验证。

答：（1）：

```
const int N = 20;
const int MAX = 100000;

typedef struct node
{
    int vex;
    int data;
    struct node* next;
}Node;

typedef struct
{
    Node* vexList[N];
    int n;
    int e;
}AdjGraph;

void AddToLast(Node* node, Node* sentinal) {
    Node* p;
    for (p = sentinal;  p->next != NULL; p = p->next);
    p->next = node;
}
void CreatAdjGraph(int** matrix, AdjGraph* adjGraph) {
    for (int i = 0; i < adjGraph->n; i++) {
        for (int j = 0; j < adjGraph->n; j++) {
            if (matrix[i][j] != MAX) {
                Node* node = (Node*)malloc(sizeof(Node));
                node->data = matrix[i][j];
                node->vex = j;
                node->next = NULL;
                AddToLast(node, adjGraph->vexList[i]);
            }
        }
    }
}
```

```
}

void PrintList(Node* node, int vex) {
    for (Node* p = node->next; p != NULL; p = p->next) {
        printf("%d->%d cost %d\n", vex, p->vex, p->data);
    }
}

void PrintAdjGraph(AdjGraph* adjGraph) {
    for (int i = 0; i < adjGraph->n; i++) {
        PrintList(adjGraph->vexList[i], i);
    }
}

void PrintMatrix(int** matrix, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (matrix[i][j] != MAX) {
                printf("%d -> %d cost %d\n", i, j, matrix[i][j]);
            }
        }
    }
}
```

由图的邻接矩阵生成邻接表的算法时间复杂度为O(V*V),V 为图的顶点个数，空间复杂度为 O(1)。

2-2 设具有 n 个顶点的有向图用邻接表存储（不存在逆邻接表）。试写出计算所有顶点入度的算法，将每个顶点的入度值分别存入一维数组 int Indegree[n]中。

答：

```
const int N = 20;
const int MAX = 100000;

typedef struct node
{
    int vex;
    int data;
    struct node* next;
}Node;

typedef struct
{
    Node* vexList[N];
    int n;
    int e;
}AdjGraph;

int SearchVex(Node* node, int vex) {
    for (Node* p = node->next; p != NULL; p = p->next) {
        if (p->vex == vex) {
            return 1;
        }
    }
    return 0;
}

void CaculateDegrees(AdjGraph* adjGraph, int* Indegree) {
    for (int i = 0; i < adjGraph->n; i++) {
        for (int j = 0; j < adjGraph->n; j++) {
            Indegree[i] += SearchVex(adjGraph->vexList[j], i);
        }
    }
}
```

时间复杂度为 O(V*V)，V 为图的顶点个数，空间复杂度为 O(1)。

2-3 一个连通图采用邻接表作为存储结构，设计一个算法，实现从顶点 v 出发的深度优先遍历的非递归过程。

答：

```
51  void DFS(AdjGraph* adjGraph, int v) {
52      Stack* stack = (Stack*)malloc(sizeof(Stack));
53      InitialStack(stack);
54      int visited[adjGraph->n];
55      for (int i = 0; i < adjGraph->n; i++) {
56          visited[i] = 0;
57      }
58      Push(stack, v);
59      visited[v] = 1;
60      while(!isEmpty(stack)) {
61          int currVex = Pop(stack);
62          printf("%d,", currVex);
63
64          Node* p = adjGraph->vexList[currVex];
65          while(p) {
66              int adjVex = p->vex;
67              if (visited[adjVex] == 0) {
68                  Push(stack, adjVex);
69                  visited[adjVex] = 1;
70              }
71              p = p->next;
72          }
73      }
74  }
75
```

时间复杂度 O(V+E)，空间复杂度为 O(V)，其中 V 为图的顶点个数，E 为图的边数。

2-4 采用链接表存储结构，编写一个判别无向图中任意给定的两个顶点(u，v)之间是否存在一条长度为 k 的简单路径

答：

```
76  int isReachable(AdjGraph* adjGraph, int u, int v, int k, int* visited){
77      if (k == 0 && u == v) {
78          return 1;
79      }
80      visited[u] = 1;
81      Node* p = adjGraph->vexList[u];
82      while (p) {
83          if (!visited[p->vex] && isReachable(adjGraph, p->vex, v, k - 1, visited)) {
84              return 1;
85          }
86          p = p->next;
87      }
88      visited[u] = 0;
89      return 0;
90  }
91
92  int HasPath(AdjGraph* adjGraph, int u, int v, int k) {
93      if (k == 0 && u == k) {
94          return 1;
95      }
96      int visited[N];
97      for (int i = 0; i < N; i++) {
98          visited[i] = 0;
99      }
100     visited[u] = 1;
101     Node* p = adjGraph->vexList[u];
102     while(p) {
103         if (isReachable(adjGraph, p->vex, v, k-1, visited)) {
104             return 1;
105         }
106         p = p->next;
107     }
108     return 0;
109 }
110
```

时间复杂度 O(V+E)，空间复杂度为 O(V)，其中 V 为图的顶点个数，E 为图的边数。