第9次作业

## 第6章习题

6.20设计算法以求解从vi到vj之间的最短路径。（每条边的长度为1）

6.21设计算法以求解距离v0最远的一个顶点。

6.23分别用prim算法和Kruskal算法求解下图的最小生成树，标注出中间求解过程的各状态。

图6-2 题6.23图

**6**

**1**

**2**

**7**

**3**

**4**

**5**

19

6

17

9

20

17

19

169

15

20

24

13

6.25在实现Kruskal算法时，如何判断某边和已选边是否构成回路？

6.26对下列AOV网，完成如下操作：

（1）按拓扑排序方法进行拓扑排序，写出中间各步的入度数组和栈的状态值，并写出拓扑序列。

（2）写出左图所示AOV网的所有的拓扑序列。

图6-3 题6.26图

**2**

**1**

**3**

**4**

**6**

**5**

**2**

**3**

**1**

**6**

**4**

**5**

**7**

(a)

(b)

**8**

**9**

**10**

6.28对下面的图，求出从顶点1到其余各顶点的最短路径。

图6-4 题6.28图

**3**

**2**

**1**

**6**

**5**

**7**

**4**

25

10

20

7

15

6

15

6

10

20

4

5

**8**

图6-27 例6-6的邻接表图

info ptr

4

1

2

3

4

5

6

7

8

**∧**

3

2

**∧**

5

4

3

**∧**

7

**∧**

4

**∧**

8

**∧**

5

5

**∧**

8

4

**∧**

6

6.20

#include <iostream>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAXN = 1005;

const int INF = 1e9;

int n; // 图的大小

int m[MAXN][MAXN]; // 邻接矩阵

bool vis[MAXN]; // 记录已经访问过的节点

int dist[MAXN]; // 记录最短距离

// Dijkstra算法求解最短路径

void dijkstra(int s, int t) {

fill(dist, dist + n, INF); // 将距离初始化为无穷大

fill(vis, vis + n, false); // 将访问标记初始化为false

dist[s] = 0; // 起点到起点的距离为0

for (int i = 0; i < n - 1; i++) {

int k = -1; // 选出未访问过的距离最短的节点

for (int j = 0; j < n; j++) {

if (!vis[j] && (k == -1 || dist[j] < dist[k]))

k = j;

}

vis[k] = true; // 标记访问过

for (int j = 0; j < n; j++) {

if (!vis[j] && m[k][j] && dist[k] + 1 < dist[j]) // 有边连接且通过k到j的距离更短

dist[j] = dist[k] + 1; // 更新距离

}

}

// 输出结果（这里只输出到t的最短距离，如果需要输出路径，则可以利用一个记录最短距离来源的数组来实现）

cout << "最短路径长度为：" << dist[t] << endl;

}

int main() {

// 读取数据

cout << "请输入图的大小和边的数量：" << endl;

int e;

cin >> n >> e;

memset(m, 0, sizeof(m)); // 初始化邻接矩阵，0表示这两个顶点之间没有边

cout << "请输入边的起点和终点：" << endl;

for (int i = 0; i < e; i++) {

int u, v;

cin >> u >> v;

m[u][v] = 1; // 有边连接，则将对应的邻接矩阵元素设为1

m[v][u] = 1; // 无向图需要在两个顶点之间都设置

}

// 求解最短路径

cout << "请输入起点和终点：" << endl;

int s, t;

cin >> s >> t;

dijkstra(s, t);

return 0;

}

6.21

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

const int N = 100;

vector<int> g[N]; // 邻接表表示

queue<int> q; // 存放BFS队列

int depth[N]; // 节点的深度

bool vis[N]; // 节点的访问状态

int bfs(int start) {

q.push(start);

vis[start] = true;

depth[start] = 0;

while (!q.empty()) {

int cur = q.front();

q.pop();

for (auto i : g[cur]) { // 遍历cur的所有邻居节点

if (!vis[i]) {

q.push(i);

vis[i] = true;

depth[i] = depth[cur] + 1;

}

}

}

int max\_depth = 0, pos;

for (int i = 0; i < N; i++) { // 找到最大深度的节点

if (depth[i] > max\_depth) {

max\_depth = depth[i];

pos = i;

}

}

return pos;

}

int main() {

int n, m, start;

cin >> n >> m >> start;

for (int i = 0; i < m; i++) {

int u, v;

cin >> u >> v;

g[u].push\_back(v);

g[v].push\_back(u); // 无向图

}

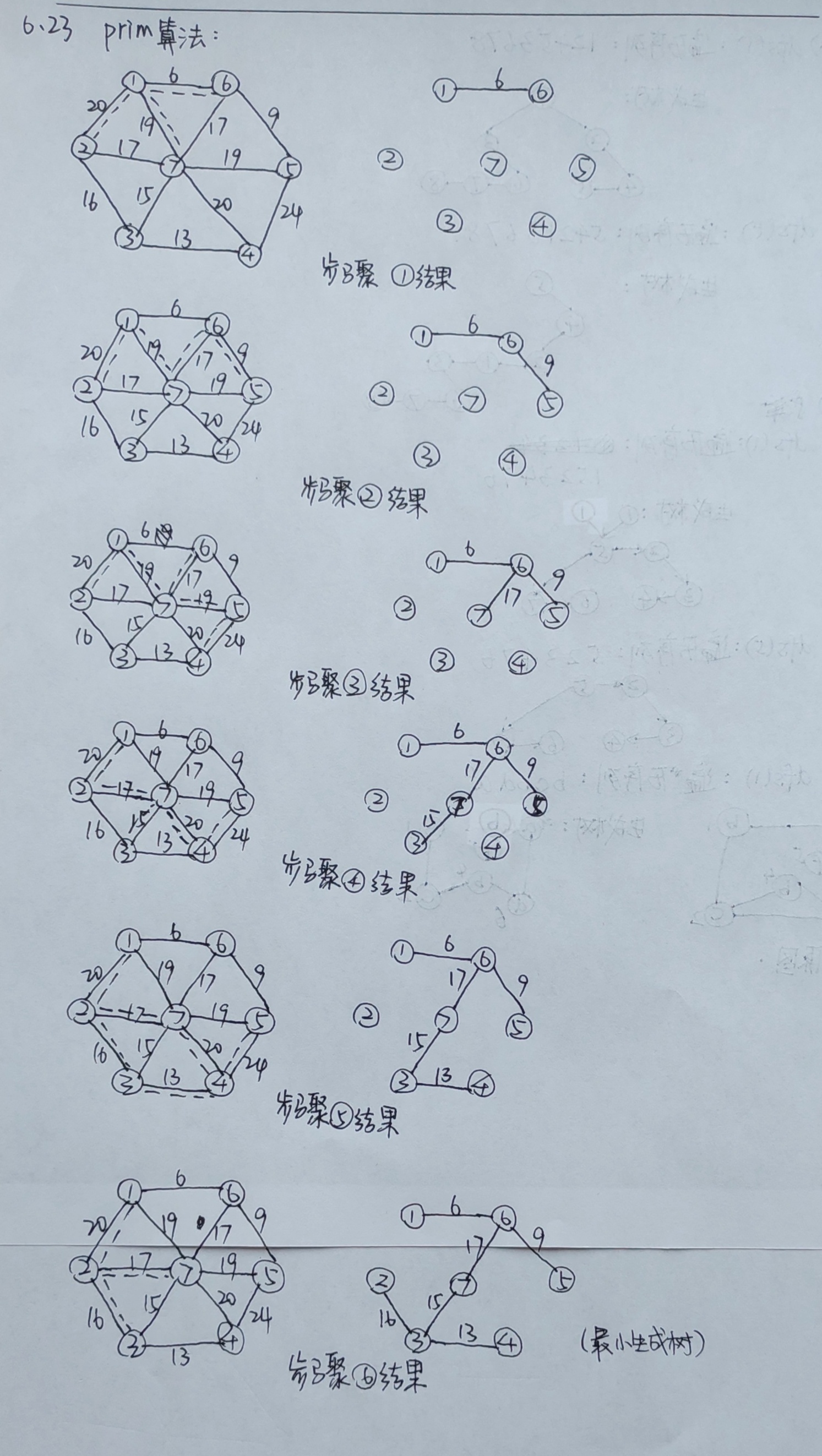
int end = bfs(start);

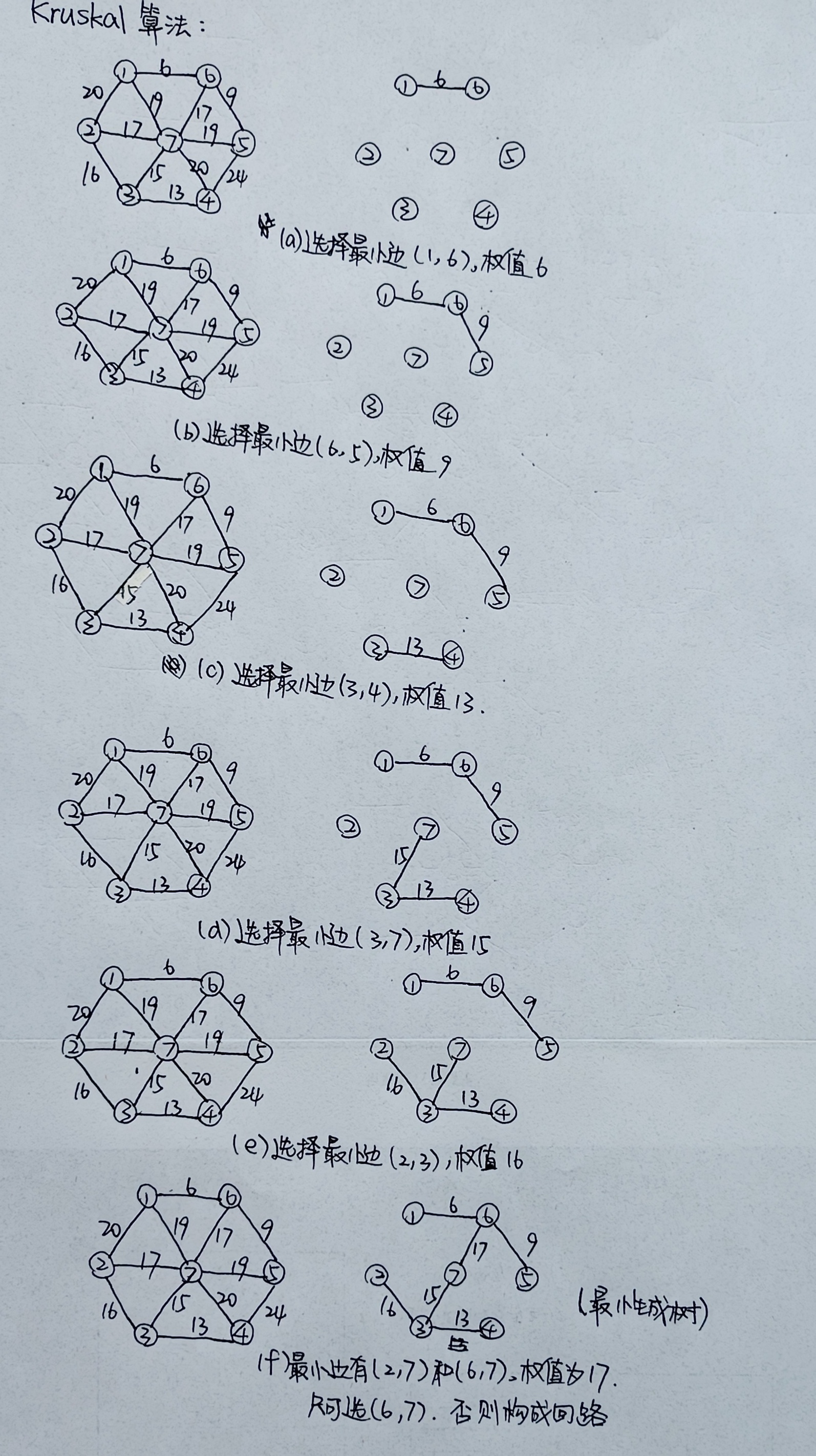
cout << "距离v0最远的顶点是：" << end << endl;

return 0;

}

6.23



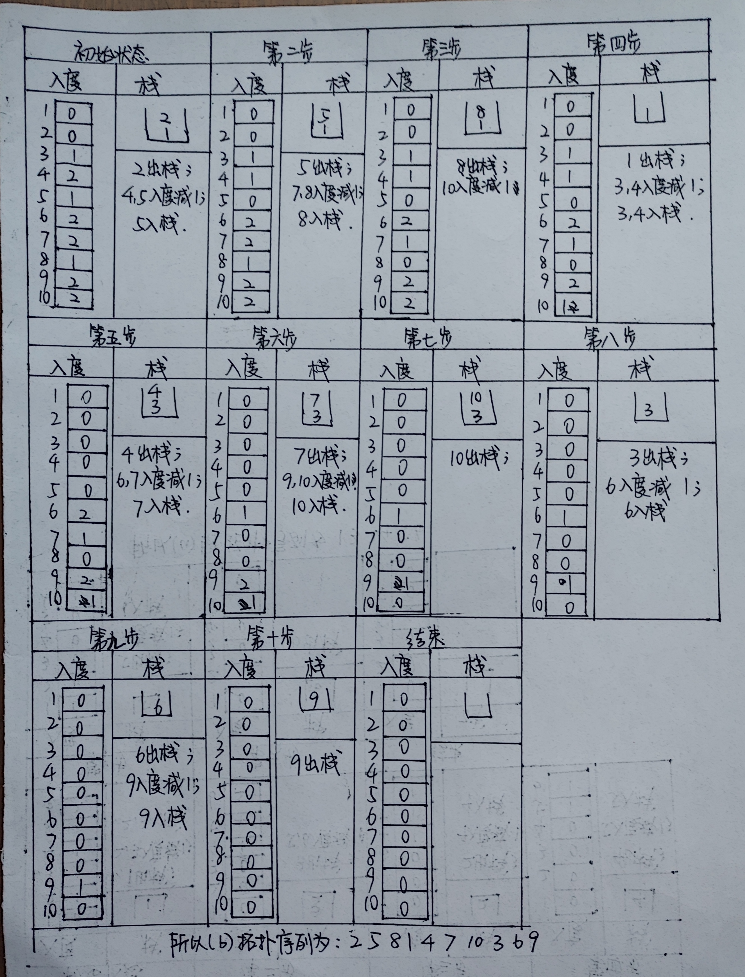


6.25 通过判断边所连接的两个节点是否处于同一连通分量，即可判断选择的边和已选边是否构成回路。

6.26

(1)





1. 1，2，4，3，5，6或1，3，2，4，5，6.

6.28

