

7.9

- 1->5->2->3->6->4
- 1->5->2->6->3->4
- 1->5->6->2->3->4
- 5->1->2->3->6->4
- 5->1->2->6->3->4
- 5->1->6->2->3->4
- 5->6->1->2->3->4
- 其中，用7.5.1节算法求得为5->6->1->2->3->4

7.10

关键路径为a->G->H->K->J->E->w

	a	A	B	C	D	E	F	G	H	I	J	K	w
e	0	1	6	17	3	34	4	3	13	1	31	22	44
l	0	20	24	26	19	34	8	3	13	7	31	22	44

	<a, A>	<a, B>	<a, D>	<a, F>	<a, G>	<a, I>	<A, C>	<B, C>
e	0	0	0	0	0	0	1	6
l	19	18	16	4	0	6	20	24
	<D, C>	<D, E>	<D, J>	<F, E>	<F, H>	<G, w>	<G, H>	<I, H>
e	3	3	3	4	4	3	3	1
l	19	26	25	23	8	23	3	7
	<C, E>	<H, C>	<H, J>	<H, K>	<K, J>	<J, E>	<J, w>	<E, w>
e	17	13	13	13	22	31	31	34
l	26	22	27	13	22	31	32	34

7.11

	a	b	c	d	e	f	g	S
0	0	15	2	12	INF	INF	INF	{a}
1	0	15	2	12	10	6	INF	{a, c}
2	0	15	2	11	10	6	16	{a, c, f}
3	0	15	2	11	10	6	16	{a, c, e, f}
4	0	15	2	11	10	6	14	{a, c, e, f, d}
5	0	15	2	11	10	6	14	{a, c, e, f, d, g}
6	0	15	2	11	10	6	14	{a, c, e, f, d, g, b}

7.14

```

1  struct Arc {
2      Arc* nxt;
3      int node;
4  }
5  struct Node{
6      int val;
7      Arc* nxt;
8  }
9  unordered_map<int, Node> G;
10 void addEdge(int begin, int end) {
11     Arc *newArc = new Arc();
12     newArc->nxt = G[begin].nxt;
13     G[begin].nxt = newArc;
14     newArc->node = end;
15 }
16 void Init() {
17     int n, m;
18     cin >> n >> m;
19     for (int i = 0; i < n; i++) {
20         int val;
21         cin >> val;
22         G[val].val = val;
23         G[val].nxt = nullptr;
24     }
25     for (int i = 0; i < m; i++) {
26         int begin, end;
27         cin >> begin >> end;
28         addEdge(begin, end);
29     }
30 }

```

7.15

```
1 void InsertVex(G, v) {
2     G.vexMap[v] = true;
3 }
4 void DeleteVex(G, v) {
5     G.vexMap.erase(v);
6 }
7 void InsertArc(G, v, w) {
8     G.arcMap[v][w] = true;
9 }
10 void DeleteArc(G, v, w) {
11     G.arcMap[v][w] = false;
12 }
```

7.19

```
1 struct Node{
2     int val;
3     Arc *nxt;
4 }
5 struct Arc{
6     int lNode, rNode;
7     Arc *lnxt, *rnxt;
8 }
9 void addEdge(int begin, int end) {
10     Arc *newArc = new Arc();
11     newArc->lNode = begin;
12     newArc->rNode = end;
13     newArc->lnxt = G[begin].nxt;
14     G[begin].nxt = newArc;
15     newArc->rnxt = G[end].nxt;
16     G[end].nxt = newArc;
17 }
18 vector<Node> G;
19 void Init() {
20     int n, m;
21     cin >> n >> m;
22     for (int i = 0; i < n; i++) {
23         int val;
24         cin >> val;
25         G.push_back({val, nullptr});
26     }
27     for (int i = 0; i < m; i++) {
28         int begin, end;
29         cin >> begin >> end;
30         addEdge(begin, end);
31     }
32 }
```

7.22

```
1  vector<vector<int>> G;
2  vector<bool> vis;
3  bool dfs(int cur, int tar) {
4      if (cur == tar) return true;
5      vis[cur] = true;
6      bool ans = false;
7      for (auto &x : G[cur])
8          if (!vis[x])
9              ans |= dfs(x, tar);
10     return ans;
11 }
12 bool solve() {
13     int n, m;
14     cin >> n >> m;
15     G.resize(n);
16     vis.resize(n);
17     for (int i = 0; i < m; i++) {
18         int x, y;
19         cin >> x >> y;
20         G[x].push_back(y);
21     }
22     int vi, vj;
23     cin >> vi >> vj;
24     return dfs(vi, vj);
25 }
```

7.23

```
1  vector<vector<int>> G;
2  vector<bool> vis;
3  bool bfs(int begin, int tar) {
4      queue<int> my_queue;
5      my_queue.push(begin);
6      vis[begin] = true;
7      while (!my_queue.empty()) {
8          int now = my_queue.front();
9          my_queue.pop();
10         for (auto &x : G[now])
11             if (!vis[x]) {
12                 my_queue.push(x);
13                 vis[x] = true;
14                 if (x == tar) return true;
15             }
16     }
17     return false;
18 }
19 bool solve() {
20     int n, m;
21     cin >> n >> m;
22     G.resize(n);
23     vis.resize(n);
24     for (int i = 0; i < m; i++) {
25         int x, y;
26         cin >> x >> y;
```

```

27     G[x],push_back(y);
28 }
29 int vi, vj;
30 cin >> vi >> vj;
31 return bfs(vi, vj);
32 }

```

7.24

```

1  vector<vector<int>>> G;
2  vector<bool> vis;
3  void solve() {
4      int n, m;
5      cin >> n >> m;
6      G.resize(n);
7      vis.resize(n);
8      for (int i = 0; i < m; i++) {
9          int x, y;
10         cin >> x >> y;
11         G[x],push_back(y);
12     }
13     int vi, vj;
14     cin >> vi >> vj;
15     stack<int> my_stack;
16     my_stack.push(1);
17     while (!my_stack.empty()){
18         int cur = my_stack.top();
19         my_stack.pop();
20         vis[cur] = true;
21         for (auto &x : G[cur])
22             if (!vis[cur]) my_stack.push(x);
23     }
24 }

```

7.34

```

1  void solve() {
2      int tot = 0;
3      queue<int> my_queue;
4      for (int i = 0; i < n; i++)
5          if (!in[i]) my_queue.push(i);
6      while (!my_queue.empty()) {
7          int cur = my_queue.front();
8          my_queue.pop();
9          idx[cur] = tot++;
10         for (auto &x : G[cur]) {
11             in[x]--;
12             if (!in[x]) my_queue.push(x);
13         }
14     }
15 }

```

7.35

```
1  int tot = 0;
2  vector<bool> vis;
3  void dfs(int cur) {
4      tot++;
5      vis[cur] = true;
6      for (auto &x : G[cur])
7          if (!vis[x]) dfs(x);
8  }
9  int solve() {
10     int id = -1;
11     for (int i = 0; i < n; i++)
12         if (!in[i]) {
13             if (id == -1) id = i;
14             else return -1    // -1 for no
15         }
16     vis.resize(n);
17     dfs(id);
18     return tot == n ? id : -1;
19 }
```

7.38

```
1  void dfs(int cur) {
2      if (G[cur].size()) {
3          dfs(G[cur][0]);
4          dfs(G[cur][1]);
5      }
6      print(data[cur]);
7  }
8  void solve() {
9      for (int i = 0; i < n; i++)
10         if (!in[i]) dfs(i)
11 }
```