Introduction To Algorithm

Third Edition

Answer

Xia Ding

December 19, 2015

22.1

22.1–1

Both of them take $O(|V|)$ time.

22.1–2

22.1–3

Algorithm 1 and 2. Transpose of matrix takes time $O(V^2)$. Transpose of list takes time $O(V + E)$.

Algorithm 1 TRANSPOSE-ADJ-MATRIX(G)

```
1: G^T = G
2: for i = 1 to |V| do
3:     for j = 1 to i do
4:         exchange G^T.A[i, j] with G^T.A[j, i]
5:     end for
6: end for
7: return G^T
```

Algorithm 2 TRANSPOSE-ADJ-LIST(G)

```
1: for i = 1 to |V| do
2:     for j in G.Adj[i] do
3:         INSERT(G^T.Adj[j], i)
4:     end for
5: end for
6: return G^T
```

22.1–4

Algorithm 3.
We use an auxiliary matrix A size of $|V| \times |V|$ to identify whether there is a edge between v, w.

Algorithm 3 ELIMINATE-MULT(G)

```
1: A = CREATE-MATRIX(|V|, |V|) {create a |V| × |V| empty matrix}
2: for i = 1 to |V| do
3:    for jinG.Adj[i] do
4:       if A[i, j] == 0 and A[j, i] == 0 then
5:          Insert(G'.Adj[i], j)
6:          Insert(G'.Adj[j], i)
7:       end if
8:       A[i, j] = 1
9:       A[j, i] = 1
10:   end for
11: end for
12: return G'
```

22.1–5

Algorithms 4 and 5.

1. For adjacency-matrix, square of a graph is square of it's adjacency-matrix. Use Strassen's algorithm to multiple matrix, running time is $O(|V|^{2.8})$. Then scan the result to set the non-zero number to 1. Total time is $O(|V|^{2.8})$.

2. For adjacency-list. Use a auxiliary $|V| \times |V|$ matrix, running time is $O(V + E^2)$

Algorithm 4 SQUARE-GRAPH-MATRIX(G)

```
1: G^2.A = MATRIX-MULTIPLE(G.A, G.A) {Strassen's algorithm}
2: for i = 1 to |V| do
3:    for j = 1 to |V| do
4:       if G^2.A[i, j] ≠ 0 then
5:          G^2.A[i, j] = 1
6:       end if
7:    end for
8: end for
```

Algorithm 5 SQUARE-GRAPH-LIST(G)

```
1: A = CREATE-MATRIX(|V|, |V|)
2: for i = 1 to |V| do
3:    for all j in G.Adj[i] do
4:       for all k in G.Adj[j] do
5:          if A[i, j] == 0 then
6:             INSERT(G^2.Adj[i], j)
7:             A[i, j] = 1
8:          end if
9:          if A[i, k] == 0 then
10:            INSERT(G^2.Adj[i], k)
11:            A[i, k] = 1
12:         end if
13:      end for
14:   end for
15: end for
```

22.1–6

Algorithm 6.

Algorithm 6 UNIVERSAL-SINK(G)

```
1: TODO
```

22.1–7

$$BB^T(i,j) = \sum_{e \in E} b_{ie} b_{ej}^T = \sum_{e \in E} b_{ie} b_{ej}$$

- If $i = j$, then $b_{ie}b_{je} = 1$(it's $1 \cdot 1$ or $(-1) \cdot (-1)$) whenever e enters or leaves vertex i, and 0 otherwise.
- If $i \neq j$, then $b_{ie}b_{je} = -1$ when $e = (i, j)$ or $e = (j, i)$, and 0 otherwise.

Thus,

$$BB^T = \begin{cases} \text{degree of} i = \text{in-degree} + \text{out-degree} & \text{if } i = j \\ -(\text{number of edges connecting}) i \text{ and } j & \text{if } i \neq j \end{cases}$$

22.1–8

1. It takes $O(1)$ time.
2. Disadvantage is wasting space or doesn't save much time when searching a node's out-vertices.

3. Using binary search tree. Can save time when determining whether an edge is in the graph, and doesn't waste space, compared to linked list representation, but use more time compared to hash table.

22.2

22.2–1