数据结构及其算法习题课

Qirun Zeng、Weixin Chen

2024 年 6 月 7 日

目录

习题 2.6

试写一个算法，对带头结点的单链表实现就地逆置。

习题 2.6

试写一个算法，对带头结点的单链表实现就地逆置。

分析

- 从头结点的下一个结点开始，采用头插法，逐个将后继结点插入到头结点之后

习题 2.6

试写一个算法，对带头结点的单链表实现就地逆置。

分析

- 从头结点的下一个结点开始，采用头插法，逐个将后继结点插入到头结点之后
- 逆置完成后，头结点的后继结点即为原链表的尾结点

习题 2.6

试写一个算法，对带头结点的单链表实现就地逆置。

Algorithm 1: 就地逆置

```
1 Function Reverse(L):
2     p ← L.next;
3     L.next ← NULL;
4     while p ≠ NULL do
5         q ← p.next;
6         p.next ← L.next;
7         L.next ← p;
8         p ← q;
9     end
```

目录

编写算法，将二叉树所有左右子树互换

Algorithm 2: Mirror

```
1 Function Mirror(T):
2     if T = NULL then
3         return NULL;
4     end
5     T.left, T.right ← Mirror(T.right), Mirror(T.left);
6     return T;
```

先序遍历和中序遍历序列构造二叉树

Algorithm 3: PreInCreate

```
1  Function PreInCreate(pre, in, n):
2      if n = 0 then
3          return NULL;
4      end
5      T ← new Node;
6      T.data ← pre[0];
7      k ← 0;
8      while in[k] ≠ pre[0] do
9          k ← k + 1;
10     end
11     T.left ← PreInCreate(pre + 1, in, k);
12     T.right ← PreInCreate(pre + k + 1, in + k + 1, n − k − 1);
13     return T;
```

中序遍历和后序遍历序列构造二叉树

Algorithm 4: InPostCreate

```
1  Function InPostCreate(in, post, n):
2      if n = 0 then
3          return NULL;
4      end
5      T ← new Node;
6      T.data ← post[n − 1];
7      k ← 0;
8      while in[k] ≠ post[n − 1] do
9          k ← k + 1;
10     end
11     T.left ← InPostCreate(in, post, k);
12     T.right ← InPostCreate(in + k + 1, post + k, n − k − 1);
13     return T;
```

判断一棵树是不是完全二叉树

Algorithm 5: IsComplete

```
1  Function IsComplete(T):
2      Q.push(T), flag ← false;
3      while Q ≠ empty do
4          p ← Q.pop();
5          if p = NULL then
6              flag ← true;
7          end
8          else
9              if flag then
10                 return false;
11             end
12             Q.push(p.left), Q.push(p.right);
13         end
14     end
15     return true;
```

习题 6.4

一棵深度为 H 的满二叉树具有如下性质：第 H 层上所有结点都是叶子结点，其余各层上每个结点都有 k 棵非空子树。如果从 1 开始按自上而下、自左向右的次序对全部结点编号，问:

1. 各层的结点数目是多少?
2. 编号为 i 的结点的父结点 (若存在) 的编号是多少?
3. 编号为 i 的结点的第 j 个孩子 (若存在) 的编号是多少?
4. 编号为 i 的结点有右兄弟的条件是什么? 其右兄弟的编号是多少?

answer

1. 第 i 层有 k^{i-1} 个结点
2. i 的父结点编号为 $\left\lfloor \frac{i-2}{k} \right\rfloor + 1$
3. $i \times k + j - k + 1$
4. 需要满足：

$$i \neq \left(\left\lfloor \frac{i-2}{k} \right\rfloor + 1 \right) \cdot k + k - k + 1 \tag{1}$$

$$i - \left\lfloor \frac{i-2}{k} \right\rfloor \cdot k \neq k + 1 \tag{2}$$

$$2 + (i-2) \mod k \neq k + 1 \tag{3}$$

$$i \mod k \neq 1 \tag{4}$$

目录

习题 9.7

判断一棵树是不是二叉排序树

Algorithm 6: 判断二叉排序树

```
1 Function IsBST(T, min, max):
2     if T = NULL then
3         return true;
4     end
5     if T.data ≤ min or T.data ≥ max then
6         return false;
7     end
8     return IsBST(T.left, min, T.data) and IsBST(T.right, T.data,
        max);
```

目录

习题 10.3

判别以下序列是否为堆（小顶堆或大顶堆），若不是，则把它调整为堆

1. (96,86,48,73,35,39,42,57,66,21);
2. (12,70,33,65,24,56,48,92,86,33);

遇到堆的题目，建议先转化为完全二叉树，判断是否满足堆的性质

堆的插入

把堆想象成一个线性表，直接在线性表的末尾插入新元素，然后调整堆的性质。一个结点的父亲的下标为 $i/2$，左儿子的下标为 $2i$，右儿子的下标为 $2i+1$。

Algorithm 7: Insert

```
1 Function Insert(H, x):
2     H.data[H.size] ← x;
3     i ← H.size;
4     while i > 1 and x < H.data[i/2] do
5         H.data[i] ← H.data[i/2];
6         i ← i/2;
7     end
8     H.data[i] ← x;
9     H.size ← H.size + 1;
```

堆的删除

删除堆顶元素，把堆的最后一个元素放到堆顶，自上而下调整

Algorithm 8: Delete

```
1  Function Delete(H):
2      x ← H.data[1], y ← H.data[H.size − 1], H.size ← H.size − 1, i ← 1;
3      while 2i < H.size do
4          j ← 2i;
5          if j < H.size and H.data[j] > H.data[j + 1] then
6              j ← j + 1;
7          end
8          if y ≤ H.data[j] then
9              break;
10         end
11         H.data[i] ← H.data[j], i ← j;
12     end
13     H.data[i] ← y;
14     return x;
```

习题 10.5

Algorithm 9: 试以单链表为存储结构，实现简单选择排序算法

```
1  Function SelectSort(L):
2      p ← L.next;
3      while p ≠ NULL do
4          q ← p.next;
5          min ← p;
6          while q ≠ NULL do
7              if q.data < min.data then
8                  min ← q;
9              end
10             q ← q.next;
11         end
12         swap(p.data, min.data);
13         p ← p.next;
14     end
```

目录

约瑟夫环

- **思路：**只需要一个循环链表，每次删除第 m 个结点，直到链表为空

约瑟夫环

- **思路：**只需要一个循环链表，每次删除第 m 个结点，直到链表为空
- **实现：**略

约瑟夫环

- **思路：**只需要一个循环链表，每次删除第 m 个结点，直到链表为空
- **实现：**略
- **优化：**考虑到长度只有 n，而 m 可能很大，可以通过取模运算优化

约瑟夫环

- **思路**：只需要一个循环链表，每次删除第 m 个结点，直到链表为空
- **实现**：略
- **优化**：考虑到长度只有 n，而 m 可能很大，可以通过取模运算优化
- **bug**：取模运算的结果可能为 0，需要特殊处理

约瑟夫环

- **思路：**只需要一个循环链表，每次删除第 m 个结点，直到链表为空
- **实现：**略
- **优化：**考虑到长度只有 n，而 m 可能很大，可以通过取模运算优化
- **bug：**取模运算的结果可能为 0，需要特殊处理
- **实现：** $m = (m-1)\%(n--)+1;$

目录

串的模式匹配算法

Algorithm 10: Next

```
1  Function Next(s, t):
2      next[0] ← −1;
3      j ← 0, k ← −1;
4      while j < t.length do
5          if k = −1 or t[j] = t[k] then
6              j ← j + 1, k ← k + 1;
7              next[j] ← k;
8          end
9          else
10             k ← next[k];
11         end
12     end
13     return next;
```

目录

最小生成树问题

- Kruskal 算法
- Prim 算法

Kruskal 算法

Algorithm 11: Kruskal

```
1  Function Kruskal(G):
2      E ← sort(G.edges);
3      T ← NULL;
4      for i ← 1 to G.vexnum do
5          parent[i] ← i;
6      end
7      for i ← 1 to E.length do
8          if Find(E[i].u) ≠ Find(E[i].v) then
9              T ← T ∪ {E[i]};
10             Union(E[i].u, E[i].v);
11         end
12     end
13     return T;
```

目录

基本概念

- 逻辑结构相关：

基本概念

- 逻辑结构相关：
  - 顶点，边，度，路径，环，有向/无向图，图/网，生成树，搜索树

基本概念

- 逻辑结构相关：
  - 顶点，边，度，路径，环，有向/无向图，图/网，生成树，搜索树
  - 连通/强连通，拓扑排序，关键路径，最短路径，最小生成树

基本概念

- 逻辑结构相关：
  - 顶点，边，度，路径，环，有向/无向图，图/网，生成树，搜索树
  - 连通/强连通，拓扑排序，关键路径，最短路径，最小生成树
- 存储结构相关：

基本概念

- 逻辑结构相关：
  - 顶点，边，度，路径，环，有向/无向图，图/网，生成树，搜索树
  - 连通/强连通，拓扑排序，关键路径，最短路径，最小生成树
- 存储结构相关：
  - 十字链表，邻接多重表，边表：给一个图，能写出各种存储结构

基本概念

- 逻辑结构相关：
  - 顶点，边，度，路径，环，有向/无向图，图/网，生成树，搜索树
  - 连通/强连通，拓扑排序，关键路径，最短路径，最小生成树
- 存储结构相关：
  - 十字链表，邻接多重表，边表：给一个图，能写出各种存储结构
  - 顶点数组 + 邻接表/邻接矩阵；在不同的场景下使用合适的存储结构，实现对问题的建模

基本概念

- 逻辑结构相关：
  - 顶点，边，度，路径，环，有向/无向图，图/网，生成树，搜索树
  - 连通/强连通，拓扑排序，关键路径，最短路径，最小生成树
- 存储结构相关：
  - 十字链表，邻接多重表，边表：给一个图，能写出各种存储结构
  - 顶点数组 + 邻接表/邻接矩阵；在不同的场景下使用合适的存储结构，实现对问题的建模
  - 顶点数组 + 逆邻接表

基本操作

- 遍历：深度优先搜索，广度优先搜索

基本操作

- 遍历：深度优先搜索，广度优先搜索
- 最短路径：Dijkstra 算法，Floyd 算法

基本操作

- 遍历：深度优先搜索，广度优先搜索
- 最短路径：Dijkstra 算法，Floyd 算法
- 最小生成树：Prim 算法，Kruskal 算法

基本操作

- 遍历：深度优先搜索，广度优先搜索
- 最短路径：Dijkstra 算法，Floyd 算法
- 最小生成树：Prim 算法，Kruskal 算法
- 拓扑排序：AOV 网，AOE 网

图

基本操作

- 遍历：深度优先搜索，广度优先搜索
- 最短路径：Dijkstra 算法，Floyd 算法
- 最小生成树：Prim 算法，Kruskal 算法
- 拓扑排序：AOV 网，AOE 网
- 关键路径：AOE 网

7.9

假设有向图以邻接表作为存储结构。试基于图的深度优先搜索策略写一算法，判断有向图中是否存在由顶点 V_i 至顶点 $V_j(i \neq j)$ 的路径。

假设有向图以邻接表作为存储结构。试基于图的广度优先搜索策略写一算法，判断有向图中是否存在由顶点 V_i 至顶点 $V_j(i \neq j)$ 的路径。

目录

查找表

基本概念

- 逻辑结构相关：

查找表

基本概念

- 逻辑结构相关：
  - 静态、动态查找表

查找表

基本概念

- 逻辑结构相关：
  - 静态、动态查找表
  - 索引，关键字

查找表

基本概念

- 逻辑结构相关：
  - 静态、动态查找表
  - 索引，关键字
  - 平均查找长度

查找表

基本概念

- 逻辑结构相关：
  - 静态、动态查找表
  - 索引，关键字
  - 平均查找长度
- 存储结构相关：

查找表

基本概念

- 逻辑结构相关：
  - 静态、动态查找表
  - 索引，关键字
  - 平均查找长度
- 存储结构相关：
  - 顺序表，有序表

查找表

基本概念

- 逻辑结构相关：
  - 静态、动态查找表
  - 索引，关键字
  - 平均查找长度
- 存储结构相关：
  - 顺序表，有序表
  - 二叉排序树，平衡二叉排序树，平衡因子

查找表

基本概念

- 逻辑结构相关：
  - 静态、动态查找表
  - 索引，关键字
  - 平均查找长度
- 存储结构相关：
  - 顺序表，有序表
  - 二叉排序树，平衡二叉排序树，平衡因子
  - 哈希查找表

查找表

基本操作

- 折半/二分查找

查找表

基本操作

- 折半/二分查找
- 分块查找

查找表

基本操作

- 折半/二分查找
- 分块查找
- 二叉排序树的构造、插入、删除和查找

查找表

基本操作

- 折半/二分查找
- 分块查找
- 二叉排序树的构造、插入、删除和查找
- 平衡二叉排序树的构造，插入和删除

查找表

基本操作

- 折半/二分查找
- 分块查找
- 二叉排序树的构造、插入、删除和查找
- 平衡二叉排序树的构造，插入和删除
- 哈希表的构造、插入和查找

查找表

基本操作

- 折半/二分查找
- 分块查找
- 二叉排序树的构造、插入、删除和查找
- 平衡二叉排序树的构造，插入和删除
- 哈希表的构造、插入和查找
- 哈希表的冲突处理

9.5

已知关键字序列 $\{10, 25, 33, 19, 06, 49, 37, 76, 60\}$, 哈希地址空间为 $0 \sim 10$, 哈希函数次 $H(Key) = Key \mod 11$, 求:

1. 用开放定址线性探测法处理冲突，构造哈希表 $HT1$，分别计算在等概率情况下 $HT1$ 查找成功和查找失败的 ASL
2. 用开放定址二次探测法处理冲突，构造 $HT2$ 查找成功的 ASL
3. 用拉链法解决冲突，构造哈希表 $HT3$ 的 ASL

排序算法复杂度及稳定性

排序方法	最好	平均	最坏	空间复杂度
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
希尔排序	$O(n\log n)$	$O(n(\log n)^2)$	$O(n(\log n)^2)$	$O(1)$
直接选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
堆排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
快速排序	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$
归并排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
基数排序	$O(d(n+r))$	$O(d(n+r))$	$O(d(n+r))$	$O(n+r)$