CSP 复赛复习 - 数据结构

By 奇思妙学

1.1 链表

基本概念

- 单链表：每个节点包含数据和指向下一个节点的指针
- 双向链表：每个节点包含指向前驱和后继的指针
- 循环链表：尾节点指向头节点形成环

时间复杂度

- 访问：$O(n)$
- 插入/删除：$O(1)$（已知位置时）

```cpp
// 单链表节点定义
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

// 双向链表节点定义
struct DoublyListNode {
    int val;
    DoublyListNode* prev;
    DoublyListNode* next;
    DoublyListNode(int x) : val(x), prev(nullptr), next(nullptr) {}
};
```

1.2 栈

后进先出 (LIFO) 结构

基本操作

- `push(x)`：入栈，$O(1)$
- `pop()`：出栈，$O(1)$
- `top()`：获取栈顶，$O(1)$
- `empty()`：判空，$O(1)$

```
int stk[N]; // 栈数组
int top = -1; // 栈顶指针

void push(int x) {
    stk[++top] = x;
}

void pop() {
    if (top >= 0) top--;
}

int top() {
    return stk[top];
}

bool empty() {
    return top == -1;
}
```

1.3 队列

先进先出 (FIFO) 结构

基本操作

- `push(x)`：入队，$O(1)$
- `pop()`：出队，$O(1)$
- `front()`：获取队首，$O(1)$
- `empty()`：判空，$O(1)$

```
int que[N]; // 队列数组
int front = 0, rear = -1; // 队首和队尾指针

void push(int x) {
    que[++rear] = x;
}

void pop() {
    if (front <= rear) front++;
}

int front() {
    return que[front];
}

bool empty() {
    return front > rear;
}
```

2. 简单树

2.1 树的定义与基本概念

相关术语

- 节点：树的基本单位
- 根节点：没有父节点的节点
- 子节点、父节点、兄弟节点
- 叶子节点：没有子节点的节点
- 深度：从根到该节点的路径长度
- 高度：从该节点到最深叶子的路径长度

2.2 二叉树

定义：每个节点最多有 2 个子节点

特殊性质

- 第 i 层最多有 2^{i-1} 个节点
- 深度为 k 的树最多有 $2^k - 1$ 个节点

```cpp
// 二叉树节点定义
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

2.3 二叉树遍历

前序遍历：根 → 左 → 右

```
void preorder(TreeNode* root) {
    if (root == nullptr) return;
    cout << root->val << " ";
    preorder(root->left);
    preorder(root->right);
}
```

中序遍历：左 → 根 → 右

```
void inorder(TreeNode* root) {
    if (root == nullptr) return;
    inorder(root->left);
    cout << root->val << " ";
    inorder(root->right);
}
```

后序遍历：左 → 右 → 根

```
void postorder(TreeNode* root) {
    if (root == nullptr) return;
    postorder(root->left);
    postorder(root->right);
    cout << root->val << " ";
}
```

3. 特殊树

3.1 完全二叉树

定义：除最后一层外，其他层节点都满，且最后一层节点靠左排列

数组表示法

- 节点 i 的左子节点：$2i+1$
- 节点 i 的右子节点：$2i+2$
- 节点 i 的父节点：$\left\lfloor \frac{i-1}{2} \right\rfloor$

```
int tree[N]; // 完全二叉树数组
int size; // 当前节点数

int getLeftChild(int index) {
    return 2 * index + 1;
}

int getRightChild(int index) {
    return 2 * index + 2;
}

int getParent(int index) {
    return (index - 1) / 2;
}
```

3.2 哈夫曼树

定义：带权路径长度最短的二叉树

构造方法

1. 将所有权值作为单独的树
2. 每次选择权值最小的两棵树合并
3. 重复直到只剩一棵树

哈夫曼编码：左路径为 0，右路径为 1

3.3 二叉搜索树

性质

- 左子树所有节点值 < 根节点值
- 右子树所有节点值 > 根节点值
- 左右子树也都是二叉搜索树

操作复杂度

- 搜索：平均 $O(\log n)$，最坏 $O(n)$
- 插入：平均 $O(\log n)$，最坏 $O(n)$
- 删除：平均 $O(\log n)$，最坏 $O(n)$

```
TreeNode* searchBST(TreeNode* root, int val) {
    if (root == nullptr || root->val == val) return root;
    if (val < root->val) return searchBST(root->left, val);
    return searchBST(root->right, val);
}
```

4. 简单图

4.1 图的基本概念

相关术语

- 顶点 (Vertex)、边 (Edge)
- 有向图、无向图
- 权重、度（入度、出度）
- 路径、环、连通性

4.2 邻接矩阵

适用场景：稠密图，顶点数 $n \leq 1000$，**空间复杂度**：$O(n^2)$

存储方式：二维数组 `g[i][j]` 表示边 (i, j) 的权重

```
int g[N][N]; // N 为矩阵大小

void addEdge(int u, int v, int w) {
    g[u][v] = w;
    // 无向图需要 g[v][u] = w;
}

bool hasEdge(int u, int v) {
    return g[u][v] != 0;
}
```

4.3 邻接表

适用场景：稀疏图，节省空间

存储方式：每个顶点维护一个链表，存储其邻接顶点，**空间复杂度**：$O(n+m)$，其中 n 为顶点数，m 为边数

邻接表（推荐）

```
vecotr<int> e[N];
// 加边
void add(int u,int v){
    e[u].push_back(v);
}
// 遍历 u 的所有边，终点为 v
for(auto v: e[u]){
    // ...
}
```

链式邻接表

```
// 边结构
struct Edge {
    int to, next, weight;
} edges[M]; // M 为边数

int head[N]; // 每个顶点的第一条边
int edgeCount; // 边计数器

void init() {
    edgeCount = 0;
    for (int i = 0; i < N; i++) head[i] = -1;
}

void addEdge(int u, int v, int w) {
    edges[edgeCount] = {v, head[u], w};
    head[u] = edgeCount++;
}
```

祝大家 CSP 复赛取得好成绩！