目 录

1 插入排序

1.1 算法思路

为 A[n-1] 找到一个合适的位置，然后把它插入到那里。从右到左扫描这个有序的子数组，直到遇到第一个小于等于 A[n-1]，然后就把 A[n-1] 插在该元素的后面。

1.2 算法实现代码

```
1  void insertion_sort(int *A, int len)
2  {
3      int i,j;
4      int temp;
5      for(i = 1; i < len; ++i)
6      {
7          temp = A[i];
8          for(j = i - 1; j > 0; --j)
9          {
10             if(A[j] <= temp)
11                 break;
12             A[j+1] = A[j];
13         }
14         A[j+1] = temp;
15     }
16 }
```

1.3 算法分析

- 最坏情况下，输入是一个严格递减的数组，此时键值比较次数为

$$C_{worst}(n) = \frac{(n-1)n}{2} \in \Theta(n^2) \tag{1}$$

- 最坏情况下，插入排序和选择排序的键值比较次数是完全一致的。
- 最优情况下，输入是一个升序排列的数组。对于有序的数组，键值比较次数为

$$C_{best}(n) = n - 1 \in \Theta(n) \tag{2}$$

- 该算法的平均键值比较次数是最差情况的一半，键值比较次数为

$$C_{avg}(n) \approx \frac{n^2}{4} \in \Theta(n^2) \tag{3}$$

2 深度优先查找

2.1 算法思路

从任意顶点开始访问图，然后将该顶点标记为已访问。在每次迭代的时候，该算法紧接着处理与当前顶点邻接的未访问顶点。这个过程一直持续，直到遇到一个无法通向未访问邻接顶点的顶点。此时，该算法沿着来路后退一条边，并试着从那里访问未访问的顶点。当后退到起始顶点的时候，如果起始顶点所邻接的所有顶点都被访问过了，那就从未访问过的顶点开始，重复上述过程。

2.2 算法实现代码

```
typedef struct Vertex Vertex;
struct Vertex
{
    int order;
    int visited;
    int parent;
    int count;
    Vertex** edges;
};
void dfs(Vertex *v, int *count)
{
    Vertex *edge;
    ++(*count);
    v->visited = 1;
    visited->count = *count;
    for(edge = v->edges; edge != NULL; ++edge)
    {
        if(edge->visited == 0)
        {
            edge->parent = v->order;
            dfs(edge, *count);
        }
    }
}
void dfs_search(Vertex *V, int num)
{
    int i;
    int count = 0;
    for(i = 0; i < num; ++i)
    {
        if(v[i].visited == 0)
        {
            (v+i)->parent = -1;
            dfs(v+i, &count);
        }
    }
}
```

2.3 算法分析

- 对于邻接矩阵表示法，该算法的时间复杂度为 $\Theta(|V|^2)$。对于邻接链表表示法，该遍历的时间复杂度为 $\Theta(|V|+|E|)$。
- 深度优先搜索可以检查图的连通性。当算法停止后，检查一下是否所有的顶点都被访问过了。如果都被访问过了，那么这个图是连通的，否则它是不连通的。
- 深度优先搜素可以检查图的无环性。利用 DFS 森林形式的表示法，如果 DFS 森林不包含回边，那么这个图就是无环的。

3 广度优先查找

3.1 算法分析

首先访问所有和初始顶点邻接的顶点，然后是离它两条边的所有未访问顶点。以此类推，直到所有与初始化顶点同在一个连通分量中的顶点都访问过了为止。如果仍然存在未被访问的顶点，该算法必须从该顶点，重复上述过程。

3.2 算法实现代码

```
typedef struct Vertex Vertex;
struct Vertex
{
    int visited;
    int count;
    Vertex** edges;
    int parent;
    int order;
};

void bfs(Vertex* v, int* count, int num)
{
    ++(*count);
    v->count = *count;
    v->visited = 1;
    int i;
    int front = 0;
    int rear = -1;
    Vertex* temp;
    Vertex* adjacent;
    Vertex** queue = (Vertex**)malloc(num * sizeof(Vertex*));
    queue[++rear] = v;
    while(front <= rear)
    {
        temp = queue[front++];
        for(i = 0; temp->edges[i] != NULL; ++i)
```

```
27              {
28                  adjacent = temp->edges[i];
29                  if(adjacent->visited == 0)
30                  {
31                      ++(*count);
32                      adjacent->visited = 1;
33                      adjacent->parent = temp->order;
34                      adjacent->count = *count;
35                      queue[++rear] = adjacent;
36                  }
37              }
38          }
39          free(queue);
40      }
41
42      void bfs_search(Vertex* v, int num)
43      {
44          int count = 0;
45          int i;
46          for(i = 0; i < num; ++i)
47          {
48              if(visited[i].visited == 0)
49              {
50                  v->parent = -1;
51                  bfs(v, &count, num);
52              }
53          }
54      }
```

3.3 算法分析

- 对于邻接矩阵表示法，广度优先搜索的时间复杂度属于 $\Theta(|V|^2)$。而对于邻接链表表示法，它属于 $\Theta(|V|+|E|)$。
- BFS 森林有两种不同类型的边：树向边和交叉边。树向边是指向原先未访问顶点的边，交叉边是指向那些已访问顶点的边。
- 类似于 DFS 的做法，可以用 BFS 来检查图的连通性和无环性。
- BFS 可以用来求两个给定顶点间边数量最少的路径。

4 拓扑排序

5 生成排列

6 生成子集

7 假币问题

8 俄式乘法

9 约瑟夫斯问题

10 计算中值和选择问题

11 插值查找

12 二叉查找树的查找和插入