数据结构与算法第四次作业

陈梓乐 19336035
部分代码可在 gitee.com/Czile/homework 中找到

1. 用行优先的储存顺序用一维数组储存上三角矩阵，求 `a_ij` 在数组中的位置

首先，求出 `a_ij` 是在第 `i` 行的第几列：

```
col of a_ij = k = j - i;
```

再者，利用求和的知识，给出 `a_ik` 的一个表达，并将 `k` 的表达式代入：

```
B(0, j) = 0
B(i, j) = B(i - 1, m) + k
        = n + (n - 1) + (n - 2) + ··· + (n - i + 2) + k
        = (2n - i + 2)(i - 1) / 2 + j - i   (i <= j)
B(i, j) = n * (n + 1) / 2                   (i >  j)
```

2. 求三对角矩阵下标到地址的映射

用i, j表示k = B(i, j)

```
B(i, j) = i - 1             (i == j)
        = n + i - 1         (i <  j)
        = 2n + j - 2        (i >  j)
```

用k表示i, j

```
i, j = k + 1 , k + 1            (k < n)
     = k - n + 1, k - n + 2     (n <= k < 2n - 1)
     = k - 2n + 3, k - 2n + 2   (2n - 1 <= k < 3n - 2)
```

3. 写出稀疏矩阵对应的顺序表与十字链表的储存表示

顺序表

```
                          (4, 5, 15)  //(rows, cols, number of zero)
0 0 2 0 8                 (1, 3, 2)
3 0 0 0 0       ==>       (1, 5, 8)
0 0 1 5 0                 (2, 1, 2)
0 0 0 0 0                 (3, 3, 1)
                          (3, 4, 5)
```

十字链表

```
first -> (1, 3, 2) -> (1, 5, 8)
             |
             V
         (3, 3, 1) -> (3, 4, 5)
first -> (2, 1, 2)
```

4. 非连续子串匹配

```cpp
// Return true if t matches s in the weak situation: Not continious match.
bool NCmatch(string &s, string &t) {
    auto it = s.begin();
    for (auto &c: t) {
        it = find(s.begin(), s.end(), c);
        if (it == s.end())
            return false;
    }
    return true;
}

// Return true if t matches s1 and s2.
bool isCommonString(string &s1, string &s2, string &t) {
    return NCmatch(s1, t) && NCmatch(s2, t);
}
```

5. 寻找矩阵中的鞍点

```cpp
vector<pair<int, int>> findSpecificPoints(double **p, int n, int m) {
    vector<pair<int, int>> ans;
    for (auto i = 0; i < n; ++i){        // Find the minimum of the row
        auto _min = 0, min = 0;
        for (auto j = 1; j < m; ++j)
            _min = (p[i, _min] < p[i, j]) ? _min : j;
        for (auto j = 1; j < m; ++j)
            min = (p[_min, min] < p[_min, j]) ? j : min;
        if (min == i)
            ans.push_back(make_pair(i, _min));
    }
    return ans;
}
```

时间复杂度为O(nm)

6. 实现KMP算法

```cpp
unsigned isSubstr(const string &s, const string &t) {
    // 计算next数组
    int * next = new int [t.length()];
    next[0] = -1;
    int k = -1, j = 0, i = 0;
    while (j < t.length() - 1)
        k = (k == -1 || t[j] == t[k]) ? next[++j] = ++k : next[k];

    // 开始匹配
    i = j = 0;
    while (i < s.length() && j < int(t.length()))
        // 若 j 匹配成功、或者第一个就匹配失败了，就移动i，否则回溯匹配
        j = (j == -1 || s[i] == t[j]) ? (++i, ++j) : next[j];
    delete [] next;
    //若 j 匹配成功则返回位置，否则返回匹配失败的代码：s的长度
    return (j == t.length()) ? i - j : s.length();
}
```