BCS009 数据结构与算法 作业 6 石雨宸 D23090120503

1. 广义表(a,(b,c,d),e)的表尾是＿((b,c,d),e)＿＿。
2. 设二维数组 A[1...m,1...n]按行存储在一维数组 B 中，则二维数组元素 A[i,j]在一维数组 B 中的下标为（ A ）。

A．n\*(i-1)+j B． n\*(i-1)+j-1 C．i\*(j-1) D．j\*m+i-1

1. 设有一个二维数组A[M][N]按行优先顺序存储，假设A[0][0] 存放位置在644(10), A[2][2] 存放位置在676(10),每个元素占一个空间， 问A[3][3] (10)存放在什么位置？（脚注(10) 表示用10进制表示）。

为了确定A[3][3]的存储位置，我们需要首先计算A[2][2]相对于A[0][0]的偏移量，然后使用这个偏移量来推断A[3][3]的位置，由于数组是按行优先顺序存储的，从A[0][0]到A[2][2]，我们需要跨越2行，每行有N个元素。因此，总的偏移量是： 2\* N +2

A[2][2]的位置 = A[0][0]的位置+（2N+2）

676 = 644 + 2N+2

N = 15

A[0][0]到A[3][3]的偏移量为3\*N + 3

A[3][3]的位置 = 644 + （3\*15+3）= 692

1. 设计一个用于存储双层集合的存储结构，所谓双层集合是指这样的集合，其中每个元素又是一个集合（称为集合元素），该集合元素由普通的整数元素构成。

例如，S={{1，3}，{1，7，8}，{5，6}}。

使用vector嵌套

#include <iostream>

#include <vector>

using namespace std;

int main()

{

*// 定义双层集合*

    vector<vector<int>> S = {{1, 3}, {1, 7, 8}, {5, 6}};

*// 遍历双层集合*

    for (const auto &subset : S)

    {

        for (int elem : subset)

        {

            cout << elem << " ";

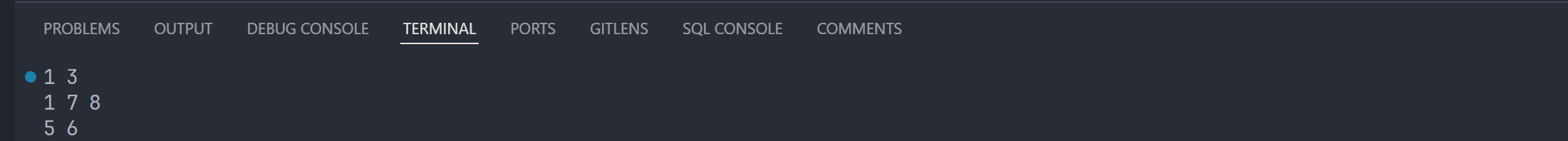
        }

        cout << endl;

    }

    return 0;

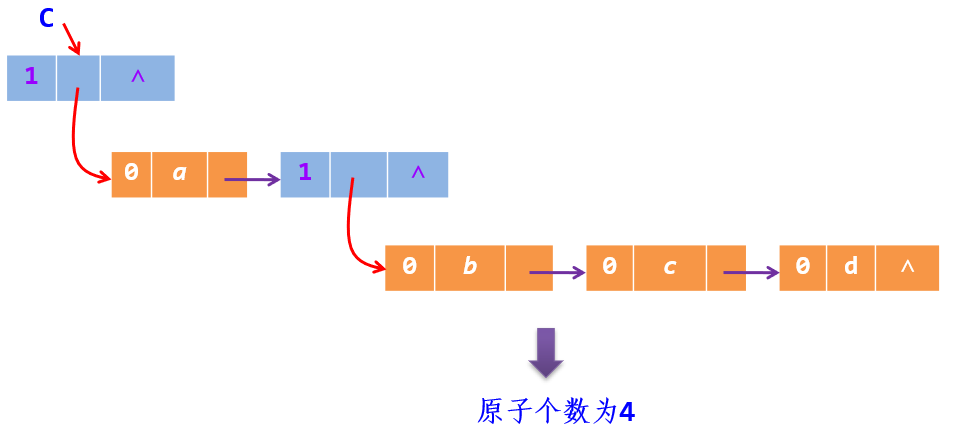
}



1. 一个稀疏矩阵采用压缩后，和直接采用二维数组存储相比会失去（ B ）特性。

A.顺序存储 B.随机存取

C.输入输出 D.以上都不对

6. 对于采用链式存储结构的广义表g，设计一个算法求原子个数。

设计一个递归函数来遍历广义表，如果当前节点是原子节点，则原子个数加1；如果当前节点是表节点，则递归遍历其子表，并累加子表中的原子个数

#include <stdio.h>

#include <stdlib.h>

typedef struct GLNode

{

    int tag; *// 0表示原子，1表示子表*

    union

    {

        char atom;

        struct

        {

            struct GLNode \*hp, \*tp; *// 子表结点的指针域，hp指向表头，tp指向表尾*

        } ptr;

    } data;

} \*Glist;

*// 创建一个原子节点*

Glist createAtom(char *atom*)

{

    Glist node = (Glist)malloc(sizeof(struct GLNode));

    node->tag = 0;

    node->data.atom = atom;

    return node;

}

*// 创建一个子表节点*

Glist createSublist(Glist *hp*, Glist *tp*)

{

    Glist node = (Glist)malloc(sizeof(struct GLNode));

    node->tag = 1;

    node->data.ptr.hp = hp;

    node->data.ptr.tp = tp;

    return node;

}

*// 递归计算广义表中的原子个数*

int countAtoms(Glist *glist*)

{

    if (glist == NULL)

    {

        return 0;

    }

    if (glist->tag == 0)

    {

        return 1; *// 原子节点*

    }

    else

    {

*// 子表节点，递归计算表头和表尾的原子个数*

        return countAtoms(glist->data.ptr.hp) + countAtoms(glist->data.ptr.tp);

    }

}

int main()

{

*// 创建广义表*

    Glist a = createAtom('a');

    Glist b = createAtom('b');

    Glist c = createAtom('c');

    Glist d = createAtom('d');

    Glist sublist = createSublist(b, c);

    Glist glist = createSublist(a, sublist);

*// 计算原子个数*

    int atomCount = countAtoms(glist);

    printf("Atom Number: %d\n", atomCount);

*// 释放内存*

    free(a);

    free(b);

    free(c);

    free(d);

    free(sublist);

    free(glist);

    return 0;

}

