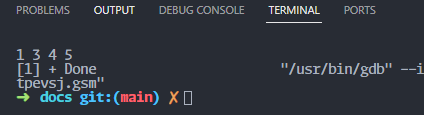
【编写完整程序，实现如下算法。将源码、测试用例以及运行结果截屏，整合在word文件中提交】

1. 已知A、B和C为三个元素值递增有序的线性表，现要求对表A作如下运算： 删去那些既在表B中出现又在表C中出现的元素。试分别以两种存储结构（一种顺序结构，一种链式的）编写实现上述运算的算法。

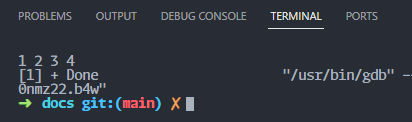
|  |
| --- |
| 1. */\*\** 2. *\* @project        : docs* 3. *\* @file           : oj1\_1.cpp* 4. *\* @dir            : ~/projects/docs/Homework/Algorithms/oj* 5. *\* @date           : 2024/03/10* 6. *\* @author         : Kaikai* 7. *\* @brief          : 顺序结构* 8. *\* \*/* 9. #include <vector> 10. #include <algorithm> 11. #include <iostream> 12. std::vector<int> removeCommonElements(std::vector<int> &*A*, std::vector<int> &*B*, std::vector<int> &*C*) 13. { 14. std::vector<int> result; 15. for (int i = 0; i < *A*.size(); i++) 16. { 17. if (!std::binary\_search(*B*.begin(), *B*.end(), *A*[i]) || !std::binary\_search(*C*.begin(), *C*.end(), *A*[i])) 18. { 19. result.push\_back(*A*[i]); 20. } 21. } 22. return result; 23. } 24. int main() 25. { 26. *// 顺序结构测试* 27. std::vector<int> A = {1, 2, 3, 4, 5}; 28. std::vector<int> B = {2, 4}; 29. std::vector<int> C = {2, 5}; 30. std::vector<int> result = removeCommonElements(A, B, C); 31. for (int i : result) 32. { 33. std::cout << i << " "; 34. } 35. std::cout << std::endl; 36. return 0; 37. } |

程序运行结果：



|  |
| --- |
| */\*\**  *\* @project        : docs*  *\* @file           : oj1\_2.cpp*  *\* @dir            : ~/projects/docs/Homework/Algorithms/oj*  *\* @date           : 2024/03/10*  *\* @author         : Kaikai*  *\* @brief          : 链式结构*  *\* \*/*  #include <iostream>  struct Node  {      int data;      Node \*next;  };  bool find(Node \**head*, int *val*)  {      Node \*current = *head*;      while (current)      {          if (current->data == *val*)          {              return true;          }          current = current->next;      }      return false;  }  Node \*removeCommonElements(Node \**A*, Node \**B*, Node \**C*)  {      Node dummy;      Node \*tail = &dummy;      while (*A*)      {          if (!find(*B*, *A*->data) || !find(*C*, *A*->data))          {              tail->next = new Node{*A*->data, nullptr};              tail = tail->next;          }  *A* = *A*->next;      }      return dummy.next;  }  int main()  {  *// 链式结构测试*      Node \*A = new Node{1, new Node{2, new Node{3, new Node{4, new Node{5, nullptr}}}}};      Node \*B = new Node{2, new Node{4, new Node{5, nullptr}}};      Node \*C = new Node{3, new Node{5, nullptr}};      Node \*result = removeCommonElements(A, B, C);      while (result)      {          std::cout << result->data << " ";          result = result->next;      }      std::cout << std::endl;      return 0;  } |

程序运行结果：



2，假设有两个按元素值递增有序排列的线性表A和B，均以单链表作存储结构，试编写算法将A表和B表归并成一个按元素值递减有序（即非递增有序，允许值相同）排列的线性表C，并要求利用原表（即A表和B表）的结点空间存放表C。

链式结构：

void ReverseListWhile(Node \**head*)

{

    Node \*prior = *head*;

    Node \*curr = *head*->next;

    Node \*next = *head*->next->next;

    while (next != nullptr)

    {

        curr->next = prior;

        prior = curr;

        curr = next;

        next = curr->next;

    }

    curr->next = prior;

*head*->next->next = NULL;

*head*->next = curr;

}

Node \*joinLinks(Node \**A*, Node \**B*)

{

    Node \*result = new Node{-1, nullptr};

    Node \*tail = result;

    Node \*head\_A = new Node{-1, *A*};

    Node \*head\_B = new Node{-1, *B*};

    ReverseListWhile(head\_A);

    ReverseListWhile(head\_B);

    while (head\_A->next || head\_B->next)

    {

        if (!head\_B->next || (head\_A->next && head\_A->next->data > head\_B->next->data))

        {

            tail->next = new Node{head\_A->next->data, nullptr};

            head\_A = head\_A->next;

        }

        else

        {

            tail->next = new Node{head\_B->next->data, nullptr};

            head\_B = head\_B->next;

        }

        tail = tail->next;

    }

    return result->next;

}

int main()

{

*// 链式结构测试*

    Node \*A = new Node{1, new Node{2, new Node{3, new Node{4, new Node{5, nullptr}}}}};

    Node \*B = new Node{2, new Node{4, new Node{5, nullptr}}};

    Node \*C = new Node{3, new Node{5, nullptr}};

    Node \*result = removeCommonElements(A, B, C);

    result = joinLinks(A, B);

    while (result)

    {

        std::cout << result->data << " ";

        result = result->next;

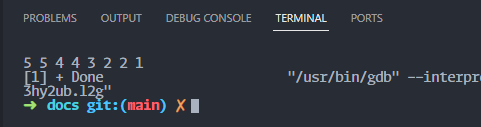
    }

    std::cout << std::endl;

    return 0;

}

程序运行结果：



顺序结构：

|  |
| --- |
| vector<int> joinVectors(vector<int> &*A*, vector<int> &*B*)  {      vector<int> result;      int length = *A*.size() + *B*.size();      int ptr\_A = *A*.size() - 1;      int ptr\_B = *B*.size() - 1;      for (int i = 0; i < length; i++)      {          if (ptr\_B < 0 || (ptr\_A >= 0 && *A*[ptr\_A] > *B*[ptr\_B]))          {              result.push\_back(*A*[ptr\_A]);              ptr\_A--;          }          else          {              result.push\_back(*B*[ptr\_B]);              ptr\_B--;          }      }      return result;  }  int main()  {  *// 顺序结构测试*      std::vector<int> A = {1, 2, 3, 4, 5};      std::vector<int> B = {2, 4};      std::vector<int> C = {2, 5};      std::vector<int> result = removeCommonElements(A, B, C);      std::cout << std::endl;      result = joinVectors(A, B);      for (int i : result)      {          std::cout << i << " ";      }      return 0;  } |

运行结果：

