#include<stdio.h>

#define MaxSize 100005

int main()

{

int Data[MaxSize];

int Next[MaxSize];

int List[MaxSize];

int firstadd,N,K;

scanf("%d %d %d",&firstadd,&N,&K);

for(int i=0;i<N;i++)

{

int tmpadd,tmpdata,tmpnext;

scanf("%d %d %d",&tmpadd,&tmpdata,&tmpnext);

Data[tmpadd] = tmpdata;

Next[tmpadd] = tmpnext;}

int sum = 0;

while(firstadd != -1)

{

List[sum++] = firstadd;

firstadd = Next[firstadd];

}

for(int i = 0;i<sum - sum%K;i += K) //反转链表

{

for(int j= 0;j<K/2;j++)

{

int t=List[i+j];

List[i+j] = List[i+K-j-1];

List[i+K-j-1] = t;

}

}

for(int i=0;i<sum-1;i++)

printf("%05d %d %05d\n",List[i],Data[List[i]],List[i+1]);

printf("%05d %d -1\n",List[sum-1],Data[List[sum-1]]);

return 0;

}

别人的优秀代码

#include<stdio.h> //c++

#include<iostream>

#define MaxSize 100005

using namespace std;

int main(){

int Data[MaxSize];

int Next[MaxSize];

int list[MaxSize];

int FirstAdd,N,K;

scanf("%d %d %d",&FirstAdd,&N,&K);

for(int i=0;i<N;i++){

int tmpAdd,tmpData,tmpNext;

scanf("%d %d %d",&tmpAdd,&tmpData,&tmpNext);

Data[tmpAdd] = tmpData;

Next[tmpAdd] = tmpNext;

}

int sum=0; // 累计有效结点数

while(FirstAdd!=-1){ // 当尾结点为 -1 时结束

list[sum++] = FirstAdd; // 记录所有Address

FirstAdd = Next[FirstAdd]; // 找下一个结点

}

for(int i=0;i<sum-sum%K;i+=K){ // 每 K 个结点一个区间

for(int j=0;j<K/2;j++){ // 反转链表

int t = list[i+j];

list[i+j] = list[i+K-j-1];

list[i+K-j-1] = t;

}

}

for(int i=0;i<sum-1;i++)

printf("%05d %d %05d\n",list[i],Data[list[i]],list[i+1]);

printf("%05d %d -1\n",list[sum-1],Data[list[sum-1]]);

return 0;

}

#include <stdio.h> //c

#include <stdlib.h>

#define MAXSIZE 100005

typedef int ElementType;

typedef struct \_node {

ElementType data;

ElementType next;

} Node;

//反转链表函数，每rev\_num 个节点，反转一下

int reverList(Node list[], int rev\_num,int head,int N)

{

int i;

int flag=1;

int last\_head; //用于记录上一次反转后，子链表的最后节点地址

//待与下一次反转的子链表的头部相连

int rev\_head = head;//下一次待反转子链表的头节点地址

int mid\_head = head;//反转过程中的中间节点地址

int mid\_next; //反转过程中的中间节点地址

while( N-rev\_num >= 0)//当剩余节点不够rev\_num个时，则不再反转

{

N -= rev\_num;

//反转子链表

for( i=0; i<rev\_num-1; i++) {

mid\_next = list[rev\_head].next;

list[rev\_head].next = list[mid\_next].next;

list[mid\_next].next = mid\_head;

mid\_head = mid\_next;

}//反转子链表

//从第二次开始，需要把这次反转后的子链表的头节点连接到上一段的尾部

if( flag == 0 ){

list[last\_head].next = mid\_head;

}

//记录第一次反转后的头结点，作为最终List的头节点并返回

if(flag) {

head = mid\_head;

flag = 0;

}

last\_head = rev\_head;

rev\_head = list[rev\_head].next;//下次开始的头结点地址

mid\_head =rev\_head;

}

return head;

}

printList(Node list[], int head)

{

int next = head;

while( list[next].next != -1){

printf("%05d %d %05d\n",next,list[next].data,list[next].next);

next = list[next].next;

}

printf("%05d %d %d\n",next,list[next].data,list[next].next);

}

int main()

{

int Head, N, rev\_num;

scanf("%d %d %d",&Head, &N, &rev\_num);

Node list[MAXSIZE];

int i,add\_front,add\_next,number;

for( i=1; i<=N; i++) {

scanf("%d %d %d",&add\_front,&number,&add\_next);

list[add\_front].data = number;

list[add\_front].next = add\_next;

}

int next = Head;

int num = 1;//所给数据节点并不都是链表中的节点，用于计数真正的节点个数

while( list[next].next != -1){

num++;

next = list[next].next;

}

if(rev\_num > 1) //如果是1，则不需要反转，原序输出

Head = reverList(list, rev\_num, Head,num);

printList(list, Head);

return 0;

}