

**实例1.1 最大子列和问题 (20 分)**

**#include**<stdio.h>

**int** **main**(){

**int** n,i,thissum,maxsum;

thissum**=**maxsum**=**0;

scanf("%d",**&**n);

**int** list[n];

**for**(i**=**0;i**<**n;i**++**)scanf("%d",**&**list[i]);

**for**(i**=**0;i**<**n;i**++**){

thissum**+=**list[i];

**if**(thissum**<**0)thissum**=**0;

**else** **if**(thissum**>**maxsum)maxsum**=**thissum;

}

printf("%d",maxsum);

**return** 0;

}

**习题2.1 简单计算器 (20 分)**

**#include**<stdio.h>

**int** **main**(){

**int** sum**=**0,a;

**char** b;

scanf("%d",**&**sum);

scanf("%c",**&**b);

**while**(b**!=**'='){

scanf("%d",**&**a);

**switch**(b){

**case** '+'**:**sum**+=**a;**break**;

**case** '-'**:**sum**-=**a;**break**;

**case** '\*'**:**sum**\*=**a;**break**;

**case** '/'**:**

**if**(a**!=**0)sum**/=**a;

**else**{

printf("ERROR");

**return** 0;

}

**break**;

**default:**printf("ERROR");**return** 0;

}

scanf("%c",**&**b);

}

printf("%d",sum);

**return** 0;

}

**习题2.2 数组循环左移 (20 分)**

**#include**<stdio.h>

**int** **main**(){

**int** m,n,temp,i,j;

scanf("%d %d",**&**n,**&**m);

**int** a[n];

m**=**m**%**n;

**for**(i**=**0;i**<**n;i**++**)scanf("%d",**&**a[i]);

**for**(i**=**0;i**<**m;i**++**){

temp**=**a[0];

**for**(j**=**0;j**<**n;j**++**)a[j]**=**a[j**+**1];

a[n**-**1]**=**temp;

}

printf("%d",a[0]);

**for**(i**=**1;i**<**n;i**++**)printf(" %d",a[i]);

**return** 0;

}

**习题2.3 数列求和-加强版 (20 分)**

**#include**<stdio.h>

**int** **main**(){

**int** A,N,i,carry**=**0;*//carry表示进位的数*

scanf("%d %d",**&**A,**&**N);

**if**(N**==**0)printf("0");

**else**{

**int** a[N];

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

a[i]**=**(A**\***(N**-**i)**+**carry)**%**10;

carry**=**(A**\***(N**-**i)**+**carry)**/**10;

}

**if**(carry)printf("%d",carry);

**for**(i**=**N**-**1;i**>=**0;i**--**)printf("%d",a[i]);

}

**return** 0;

}

**习题2.8 输出全排列**

**#include** <stdio.h>

**int** visited[10]**=**{0};

**int** list[10];

**void** **dfs**(**int** n,**int** m){

**int** i;

**if**(m**==**n**+**1){

**for**(**int** i**=**1;i**<=**n;i**++**)

printf("%d",list[i]);

printf("\n");

}

**else**{

**for**(i**=**1;i**<=**n;i**++**){

**if**(**!**visited[i]){

list[m]**=**i;

visited[i]**=**1;

dfs(n,m**+**1);

visited[i]**=**0;

}

}

}

}

**int** **main**(){

**int** n;

scanf("%d", **&**n);

dfs(n,1);

**return** 0;

}

**习题3.4 最长连续递增子序列**

**#include**<stdio.h>

**int** **main**(){

**int** n,i,j**=**0,num**=**1,max**=**1;

scanf("%d",**&**n);

**int** a[n];

**for**(i**=**0;i**<**n;i**++**)scanf("%d",**&**a[i]);

**if**(n**==**1)printf("%d",a[0]);

**else**{

**for**(i**=**0;i**<**n**-**1;i**++**){

**if**(a[i]**<**a[i**+**1]){

num**+=**1;

**if**(max**<**num){

max**=**num;

j**=**i**+**1;

}

}**else** num**=**1;

}

**for**(i**=**max;i**>**0;i**--**){

printf("%d",a[j**-**i**+**1]);

**if**(i**>**1)printf(" ");

}

}

**return** 0;

}

**习题3.6 一元多项式的乘法与加法运算**

**#include**<stdio.h>

**#include**<stdlib.h>

**typedef** **struct** PolyNode**\*** Polynomial;

**struct** PolyNode {

**int** Coef;

**int** Expon;

Polynomial Next;

};

**typedef** Polynomial PtrToPolyNode;

Polynomial **Read**();

**void** **Attach**(**int** coef, **int** expon, PtrToPolyNode**\*** Rear);

Polynomial **Mult**(Polynomial P1, Polynomial P2);

Polynomial **Add**(Polynomial P1, Polynomial P2);

**void** **PrintReslut**(Polynomial P);

**int** **main**(){

Polynomial P1, P2, PM, PA;

P1 **=** Read();

P2 **=** Read();

PM **=** Mult(P1, P2);

PA **=** Add(P1, P2);

PrintReslut(PM);

PrintReslut(PA);

**return** 0;

}

Polynomial **Read**(){

Polynomial P **=** (Polynomial)malloc(**sizeof**(**struct** PolyNode));

P**->**Next **=** NULL;

Polynomial Rear **=** P, t;

**int** n, coef, expon;

scanf("%d", **&**n);

**while** (n**--**){

scanf("%d %d", **&**coef, **&**expon);

Attach(coef, expon, **&**Rear);

}

t **=** P;

P **=** P**->**Next;

free(t);

**return** P;

}

**void** **Attach**(**int** coef, **int** expon, PtrToPolyNode**\*** Rear){

Polynomial P **=** (Polynomial)malloc(**sizeof**(**struct** PolyNode));

P**->**Coef **=** coef;

P**->**Expon **=** expon;

P**->**Next **=** NULL;

(**\***Rear)**->**Next **=** P;

**\***Rear **=** P;

}

Polynomial **Mult**(Polynomial P1, Polynomial P2){

**if** (**!**P1 **||** **!**P2) **return** NULL;

Polynomial P **=** (Polynomial)malloc(**sizeof**(**struct** PolyNode));

P**->**Next **=** NULL;

Polynomial t1 **=** P1, t2 **=** P2, Rear **=** P, t;

**int** coef, expon;

**while** (t2){

Attach(t1**->**Coef **\*** t2**->**Coef, t1**->**Expon **+** t2**->**Expon, **&**Rear);

t2 **=** t2**->**Next;

}

t1 **=** t1**->**Next;

**while** (t1){

t2 **=** P2;

Rear **=** P;

**while** (t2){

coef **=** t1**->**Coef **\*** t2**->**Coef;

expon **=** t1**->**Expon **+** t2**->**Expon;

**while** (Rear**->**Next **&&** Rear**->**Next**->**Expon **>** expon)Rear **=** Rear**->**Next;

**if** (Rear**->**Next **&&** Rear**->**Next**->**Expon **==** expon){

Rear**->**Next**->**Coef **+=** coef;

**if** (Rear**->**Next**->**Coef **==** 0){

t **=** Rear**->**Next;

Rear**->**Next **=** t**->**Next;

free(t);

}

}**else**{

t **=** (Polynomial)malloc(**sizeof**(**struct** PolyNode));

t**->**Coef **=** coef;

t**->**Expon **=** expon;

t**->**Next **=** Rear**->**Next;

Rear**->**Next **=** t;

}

t2 **=** t2**->**Next;

}

t1 **=** t1**->**Next;

}

t **=** P;

P **=** P**->**Next;

free(t);

**return** P;

}

Polynomial **Add**(Polynomial P1, Polynomial P2){

Polynomial P **=** (Polynomial)malloc(**sizeof**(**struct** PolyNode));

P**->**Next **=** NULL;

Polynomial Rear **=** P, t;

**while** (P1 **&&** P2){

**if** (P1**->**Expon **>** P2**->**Expon){

Attach(P1**->**Coef, P1**->**Expon, **&**Rear);

P1 **=** P1**->**Next;

}**else** **if** (P1**->**Expon **<** P2**->**Expon){

Attach(P2**->**Coef, P2**->**Expon, **&**Rear);

P2 **=** P2**->**Next;

}**else**{

**if**(P1**->**Coef**+**P2**->**Coef)

Attach(P1**->**Coef**+**P2**->**Coef, P2**->**Expon, **&**Rear);

P1 **=** P1**->**Next;

P2 **=** P2**->**Next;

}

}

**while** (P1){

Attach(P1**->**Coef, P1**->**Expon, **&**Rear);

P1 **=** P1**->**Next;

}

**while** (P2) {

Attach(P2**->**Coef, P2**->**Expon, **&**Rear);

P2 **=** P2**->**Next;

}

t **=** P;

P **=** P**->**Next;

free(t);

**return** P;

}

**void** **PrintReslut**(Polynomial P){

**if** (P **==** NULL) printf("0 0\n");

**else**{

printf("%d %d", P**->**Coef, P**->**Expon);

P **=** P**->**Next;

**while** (P){

printf(" %d %d", P**->**Coef, P**->**Expon);

P **=** P**->**Next;

}

printf("\n");

}

}

**习题3.8 符号配对**

**#include** <stdio.h>

**#define MAXN 100**

**int** S[MAXN], top, newline;

**void** **push**( **int** e );

**void** **pop**();

**int** **token**( **char** c );**/**

**int** check( **int** t );

**int** **main**() {

**char** c;

**int** t, out;

newline **=** 1;

top **=** **-**1;

out **=** 0;

**while** (1) {

scanf("%c", **&**c);

t **=** token(c);

**switch** (t) {

**case** 100**:**

out **=** 1;

**break**;

**case** 0**:**

newline **=** 1;

**break**;

**default:** {

newline **=** 0;

**if** (t **>** 10)

out **=** check(t);

**break**;

}

}

**if** (out)

**break**;

}

**if** (out **==** 1 **&&** top **==** **-**1)

printf("YES\n");

**else** {

printf("NO\n");

**if** (out **>** 0) {

**switch** (S[top]) {

**case** 1**:**

printf("/\*");

**break**;

**case** 2**:**

printf("{");

**break**;

**case** 3**:**

printf("[");

**break**;

**case** 4**:**

printf("(");

**break**;

**default:**

**break**;

}

printf("-?\n");

} **else** {

printf("?-");

**if** (t **==** 11)

printf("\*/\n");

**else**

printf("%c\n", c);

}

}

**return** 0;

}

**void** **push**( **int** e ) {

S[**++**top] **=** e;

}

**void** **pop**() {

top**--**;

}

**int** **token**( **char** c ) {

**int** t;

**if** (c **==** '.' **&&** newline) {

scanf("%c", **&**c);

t **=** token(c);

**if** (**!**t)

**return** 100;

**else**

**return** t;

}

**switch** (c) {

**case** '\n'**:**

**return** 0;

**case** '{'**:**

push(2);

**return** 2;

**case** '['**:**

push(3);

**return** 3;

**case** '('**:**

push(4);

**return** 4;

**case** '/'**:**

scanf("%c", **&**c);

**if** (c **==** '\*') {

push(1);

**return** 1;

} **else**

**return** token(c);

**case** '}'**:**

**return** 12;

**case** ']'**:**

**return** 13;

**case** ')'**:**

**return** 14;

**case** '\*'**:**

scanf("%c", **&**c);

**if** (c **==** '/') {

**return** 11;

} **else**

**return** token(c);

**default:**

**return** **-**1;

}

}

**int** **check**( **int** t ) {

**if** (top **==** **-**1)

**return** **-**1;

**if** (S[top] **!=** (t **-** 10))

**return** 1;

**else** {

pop();

**return** 0;

}

}

**习题3.9 堆栈操作合法性**

**#include** <stdio.h>

**#define MAXS 101**

**#define MAXN 50**

**int** S[MAXN], top, M;

**int** **IsEmpty**(){

**return** (top**==-**1)**?** 1**:**0;

}

**int** **IsFull**(){

**return** (top**==**M**-**1)**?** 1**:**0;

}

**int** **push**(){

**if** (IsFull()) **return** 0;

**else** { S[**++**top] **=** 1; **return** 1; }

}

**int** **pop**(){

**if** (IsEmpty()) **return** 0;

**else** { top**--**; **return** 1; }

}

**int** **main**(){

**int** N, i, j;

**char** str[MAXS];

scanf("%d %d\n", **&**N, **&**M);

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

scanf("%s", str);

j **=** 0; top **=** **-**1;

**while** (str[j]**!=**'\0') {

**if** ((str[j]**==**'S') **&&** (**!**push())) **break**;

**if** ((str[j]**==**'X') **&&** (**!**pop())) **break**;

j**++**;

}

**if** ((str[j]**==**'\0') **&&** IsEmpty()) printf("YES\n");

**else** printf("NO\n");

}

**return** 0;

}

**习题3.10 汉诺塔的非递归实现**

**#include**<stdio.h>

**#define MaxSize 100**

**typedef** **struct** {

**int** N;

**char** A;

**char** B;

**char** C;

} ElementType;

ElementType ERROR;

**typedef** **struct** {

ElementType Data[MaxSize];

**int** Top;

} Stack;

**void** **Push**( Stack **\***PtrS, ElementType item ){

**if** ( PtrS**->**Top **==** MaxSize**-**1 ) {

printf("堆栈满");

**return**;

}

**else** {

PtrS**->**Data[**++**(PtrS**->**Top)] **=** item;

**return**;

}

}

ElementType **Pop**( Stack **\***PtrS ){

**if** ( PtrS**->**Top **==** **-**1 ) {

printf("堆栈空");

**return** ERROR;

}

**else** {

PtrS**->**Top **--**;

**return** ( PtrS**->**Data[PtrS**->**Top**+**1] );

}

}

**void** **Hanoi**( **int** n ) {

ElementType P, toPush;

Stack S;

P.N **=** n; P.A**=**'a'; P.B**=**'b'; P.C**=**'c';

S.Top**=** **-**1;

Push(**&**S, P);

**while** ( S.Top **!=** **-**1 ) {

P **=** Pop(**&**S);

**if** ( P.N **==** 1)

printf("%c -> %c\n", P.A, P.C);

**else** {

toPush.N **=** P.N **-** 1; toPush.A **=** P.B; toPush.B **=** P.A; toPush.C **=** P.C;

Push( **&**S, toPush );

toPush.N **=** 1; toPush.A **=** P.A; toPush.B **=** P.B; toPush.C **=** P.C;

Push( **&**S, toPush );

toPush.N **=** P.N **-** 1; toPush.A **=** P.A; toPush.B **=** P.C; toPush.C **=** P.B;

Push( **&**S, toPush );

}

}

}

**int** **main**(){

**int** n;

ERROR.N **=** **-**1;

scanf("%d", **&**n);

Hanoi(n);

**return** 0;

}

**习题3.11 表达式转换**

**#include** <stdio.h>

**#include** <string.h>

**#define MAXL 20**

**char** S[MAXL];

**int** top **=** **-**1;

**typedef** **enum** {lparen, rparen, plus, minus, times, divide, eos, operand}

Precedence;

Precedence **P** ( **char** op ){

**switch** ( op ) {

**case** '+'**:** **return** plus;

**case** '-'**:** **return** minus;

**case** '\*'**:** **return** times;

**case** '/'**:** **return** divide;

**case** '\0'**:** **return** eos;

**default:** **return** operand;

}

}

**void** **ToPostfix**( **char** **\***expr ){

**int** i, j, L;

**char** pexpr[2**\***MAXL**+**2];

i **=** j **=** 0; L **=** strlen(expr);

**while** ( i**<**L ) {

**while** (isdigit(expr[i]) **||** (expr[i]**==**'.'))

pexpr[j**++**] **=** expr[i**++**];

**if** (j **&&** isdigit(pexpr[j**-**1]))

pexpr[j**++**] **=** ' ';

**if** (i**==**L) **break**;

**switch**(expr[i]) {

**case** '('**:** S[**++**top] **=** '('; **break**;

**case** ')'**:**

**while** (S[top]**!=**'(') {

pexpr[j**++**] **=** S[top**--**];

pexpr[j**++**] **=** ' ';

**if** (top **==** **-**1) **return**;

}

top**--**;

**break**;

**default:**

**if** ( **!**i **||** (**!**isdigit(expr[i**-**1]) **&&** (expr[i**-**1]**!=**')')) ) {

**if** (expr[i]**==**'+') **break**;

**else** **if** (expr[i]**==**'-') { pexpr[j**++**] **=** expr[i]; **break**; }

}

**while** (top **>-**1 **&&** S[top]**!=**'(' **&&** P(expr[i]) **<=** P(S[top])) {

pexpr[j**++**] **=** S[top**--**];

pexpr[j**++**] **=** ' ';

**if** (top **==** **-**1) **break**;

}

S[**++**top] **=** expr[i];

**break**;

}

i**++**;

}

**while** (top**>-**1) {

pexpr[j**++**] **=** S[top**--**];

pexpr[j**++**] **=** ' ';

}

pexpr[j**-**1] **=** '\0';

printf("%s\n", pexpr);

}

**int** **main**(){

**char** s[MAXL**+**1];

**int** i **=** 0;

scanf("%s", s);

ToPostfix(s);

**return** 0;

}

**练习4.1 根据后序和中序遍历输出先序遍历**

**#include** <stdio.h>

**#include** <stdlib.h>

**#define MAXN 30**

**typedef** **struct** TreeNode **\***Tree;

**struct** TreeNode {

**int** Element;

Tree Left;

Tree Right;

};

Tree **BuildTree**( **int** inorder[], **int** postorder[], **int** N ){

Tree T;

**int** p;

**if** (**!**N) **return** NULL;

T **=** (Tree)malloc(**sizeof**(**struct** TreeNode));

T**->**Element **=** postorder[N**-**1];

T**->**Left **=** T**->**Right **=** NULL;

**for** (p**=**0; p**<**N; p**++**)

**if** (inorder[p]**==**postorder[N**-**1]) **break**;

T**->**Left **=** BuildTree( inorder, postorder, p );

T**->**Right **=** BuildTree( inorder**+**p**+**1, postorder**+**p, N**-**p**-**1 );

**return** T;

}

**void** **Preorder\_output**( Tree T ){

**if** (**!**T) **return**;

printf(" %d", T**->**Element);

Preorder\_output(T**->**Left);

Preorder\_output(T**->**Right);

}

**int** **main**(){

Tree T;

**int** inorder[MAXN], postorder[MAXN], N, i;

scanf("%d", **&**N);

**for** (i**=**0; i**<**N; i**++**) scanf("%d", **&**postorder[i]);

**for** (i**=**0; i**<**N; i**++**) scanf("%d", **&**inorder[i]);

T **=** BuildTree(inorder, postorder, N);

printf("Preorder:");

Preorder\_output(T); printf("\n");

**return** 0;

}

**练习4.2 平衡二叉树的根**

**#include** <stdio.h>

**#include** <stdlib.h>

**typedef** **struct** Node **\***Tree;

**struct** Node {

**int** key, h;

Tree left, right;

};

**int** **maxh**( **int** h1, **int** h2 ){

**return** h1**<**h2 **?** h2:h1;

}

**int** **Height**( Tree T ){

**if** (T) **return** T**->**h;

**else** **return** **-**1;

}

Tree **LL**( Tree K2 ){

Tree K1;

K1 **=** K2**->**left;

K2**->**left **=** K1**->**right;

K1**->**right **=** K2;

K2**->**h **=** maxh(Height(K2**->**left), Height(K2**->**right)) **+** 1;

K1**->**h **=** maxh(Height(K1**->**left), K2**->**h) **+** 1;

**return** K1;

}

Tree **RR**( Tree K2 ){

Tree K1;

K1 **=** K2**->**right;

K2**->**right **=** K1**->**left;

K1**->**left **=** K2;

K2**->**h **=** maxh(Height(K2**->**left), Height(K2**->**right)) **+** 1;

K1**->**h **=** maxh(Height(K1**->**right), K2**->**h) **+** 1;

**return** K1;

}

Tree **LR**( Tree K3 ){

Tree K1, K2;

K1 **=** K3**->**left;

K2 **=** K1**->**right;

K1**->**right **=** K2**->**left;

K3**->**left **=** K2**->**right;

K2**->**left **=** K1;

K2**->**right **=** K3;

K1**->**h **=** maxh(Height(K1**->**left), Height(K1**->**right)) **+** 1;

K3**->**h **=** maxh(Height(K3**->**left), Height(K3**->**right)) **+** 1;

K2**->**h **=** maxh(K1**->**h, K3**->**h) **+** 1;

**return** K2;

}

Tree **RL**( Tree K3 ){

Tree K1, K2;

K1 **=** K3**->**right;

K2 **=** K1**->**left;

K1**->**left **=** K2**->**right;

K3**->**right **=** K2**->**left;

K2**->**right **=** K1;

K2**->**left **=** K3;

K1**->**h **=** maxh(Height(K1**->**left), Height(K1**->**right)) **+** 1;

K3**->**h **=** maxh(Height(K3**->**left), Height(K3**->**right)) **+** 1;

K2**->**h **=** maxh(K1**->**h, K3**->**h) **+** 1;

**return** K2;

}

Tree **Insert**( **int** K, Tree T ){

**if** (**!**T) {

T **=** malloc(**sizeof**(**struct** Node));

T**->**key **=** K; T**->**h **=** 0;

T**->**left **=** T**->**right **=** NULL;

}

**else** {

**if** (K **<** T**->**key) {

T**->**left **=** Insert(K, T**->**left);

**if** (Height(T**->**left) **-** Height(T**->**right) **>** 1)

**if** (K **<** T**->**left**->**key) T **=** LL(T);

**else** T **=** LR(T);

}

**else** {

T**->**right **=** Insert(K, T**->**right);

**if** (Height(T**->**right) **-** Height(T**->**left) **>** 1)

**if** (K **>** T**->**right**->**key) T **=** RR(T);

**else** T **=** RL(T);

}

T**->**h **=** maxh(Height(T**->**left), Height(T**->**right)) **+** 1;

}

**return** T;

}

**int** **main**(){

**int** N, K, i;

Tree T **=** NULL;

scanf("%d", **&**N);

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

scanf("%d", **&**K);

T **=** Insert(K, T);

}

printf("%d\n", T**->**key);

**return** 0;

}

**练习4.3 堆中的路径**

**#include** <stdio.h>

**#define MAXN 1001**

**#define MINH -10001**

**int** H[MAXN], size;

**void** **Create** (){

size **=** 0;

H[0] **=** MINH;

}

**void** **Insert** ( **int** X ){

**int** i;

**for** (i**=++**size; H[i**/**2] **>** X; i**/=**2)

H[i] **=** H[i**/**2];

H[i] **=** X;

}

**int** **main**(){

**int** n, m, x, i, j;

scanf("%d %d", **&**n, **&**m);

Create();

**for** (i**=**0; i**<**n; i**++**) {

scanf("%d", **&**x);

Insert(x);

}

**for** (i**=**0; i**<**m; i**++**) {

scanf("%d", **&**j);

printf("%d", H[j]);

**while** (j**>**1) {

j **/=** 2;

printf(" %d", H[j]);

}

printf("\n");

}

**return** 0;

}

**习题4.5 顺序存储的二叉树的最近的公共祖先问题**

**#include** <stdio.h>

**#define MAXN 1001**

**int** **NCA**( **int** T[], **int** p1, **int** p2){

**int** p;

**while** (p1 **!=** p2) {

**if** (p1 **>** p2) { p**=**p1; p1**=**p2; p2**=**p; }

**while** (p2**>**p1) p2**/=**2;

}

**return** p1;

}

**int** **main**(){

**int** n, i, T[MAXN];

**int** p1, p2, p;

T[0] **=** 0;

scanf("%d", **&**n);

**for** (i**=**1; i**<=**n; i**++**) scanf("%d", **&**T[i]);

scanf("%d %d", **&**p1, **&**p2);

**if** (**!**T[p1]) printf("ERROR: T[%d] is NULL\n", p1);

**else** **if** (**!**T[p2]) printf("ERROR: T[%d] is NULL\n", p2);

**else** {

p **=** NCA(T, p1, p2);

printf("%d %d\n", p, T[p]);

}

**return** 0;

}

**习题5.13 词频统计**

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**typedef** **enum** {false, true} **bool**;

**#define MAXTable 50653**

**#define KEYLENGTH 15**

**typedef** **char** ElementType[KEYLENGTH**+**1];

**typedef** **int** Index;

**typedef** **struct** LNode **\***PtrToLNode;

**struct** LNode {

ElementType Data;

**int** Count;

PtrToLNode Next;

};

**typedef** PtrToLNode Position;

**typedef** PtrToLNode List;

**typedef** **struct** TblNode **\***HashTable;

**struct** TblNode {

**int** TableSize;

List Heads;

};

HashTable **CreateTable**(){

HashTable H;

**int** i;

H **=** (HashTable)malloc(**sizeof**(**struct** TblNode));

H**->**TableSize **=** MAXTable;

H**->**Heads **=** (List)malloc(H**->**TableSize**\*sizeof**(**struct** LNode));

**for**( i**=**0; i**<**H**->**TableSize; i**++** ) {

H**->**Heads[i].Data[0] **=** '\0';

H**->**Heads[i].Count **=** 0;

H**->**Heads[i].Next **=** NULL;

}

**return** H;

}

**int** **V** ( **char** c ){

**if** (c**>=**'0' **&&** c**<=**'9') **return** (c**-**'0');

**if** (c**==**'\_') **return** 36;

**return** (c**-**'a'**+**10);

}

**int** **Hash**( ElementType Key ){

**int** i, h **=** 0;

**for** (i**=**0; i**<**3; i**++**) {

**if** (Key[i]**==**'\0') {

h**\*=**37;

**if** (i**<**2) h**\*=**37;

**break**;

}

**else** h **=** h**\***37 **+** V(Key[i]);

}

**return** h;

}

**bool** **IsWordChar**( **char** c ){

**if** ( (c**>=**'a'**&&** c**<=**'z') **||** (c**>=**'A'**&&** c**<=**'Z') **||** (c**>=**'0'**&&** c**<=**'9')**||** (c **==** '\_') )

**return** true;

**else**

**return** false;

}

**#define MAXWORDLEN 80**

**int** **GetAWord**( ElementType word ){

**char** tempword[MAXWORDLEN**+**1], c;

**int** len **=** 0;

scanf("%c", **&**c);

**while**( c**!=**'#' ){

**if**( IsWordChar(c) ) {

**if** (c**>=**'A' **&&** c**<=**'Z') c **=** c**-**'A'**+**'a';

tempword[len**++**] **=** c;

}

scanf("%c", **&**c);

**if**( len **&&** **!**IsWordChar(c) )

**break**;

}

**if** (c**==**'#') **return** **-**1;

tempword[len] **=** '\0';

**if**( len**>**KEYLENGTH ){

tempword[KEYLENGTH] **=** '\0';

len **=** KEYLENGTH;

}

strcpy(word, tempword);

**return** len;

}

**void** **Show**( HashTable H, **double** percent ){ **int** diffwordcount **=** 0;

**int** maxf **=** 0;

Position L;

**int** i, j, k, lowerbound, count **=** 0;

**for**( i**=**0; i**<**H**->**TableSize; i**++** ) {

diffwordcount **+=** H**->**Heads[i].Count;

L **=** H**->**Heads[i].Next;

**while**( L ){

**if**( maxf**<**L**->**Count ) maxf **=** L**->**Count;

L **=** L**->**Next;

}

}

printf("%d\n", diffwordcount);

lowerbound **=** (**int**)((**double**)diffwordcount **\*** percent);

count **=** 0;

**for**( j**=**maxf; j**>=**1; j**--** ) {

**for**( k**=**0; k**<**H**->**TableSize; k**++** ) {

L **=** H**->**Heads[k].Next;

**while**( L ){

**if**( j**==**L**->**Count ) {

printf("%d:%s\n", L**->**Count, L**->**Data);

count**++**;

}

**if** (count **==** lowerbound) **return**;

L **=** L**->**Next;

}

}

}

}

Position **Find**( HashTable H, ElementType Key ){

Position P;

Index Pos;

Pos **=** Hash( Key );

P **=** H**->**Heads[Pos].Next;

**while**( P **&&** strcmp(P**->**Data, Key) )

P **=** P**->**Next;

**return** P;

}

**void** **InsertAndCount**( HashTable H, ElementType Key ){

Position P, NewCell;

Index Pos;

P **=** Find( H, Key );

**if** ( **!**P ) {

NewCell **=** (Position)malloc(**sizeof**(**struct** LNode));

strcpy(NewCell**->**Data, Key);

NewCell**->**Count **=** 1;

Pos **=** Hash( Key );

H**->**Heads[Pos].Count**++**;

P **=** H**->**Heads**+**Pos;

**while** (P**->**Next **&&** strcmp(P**->**Next**->**Data, Key)**<**0) P **=** P**->**Next;

NewCell**->**Next **=** P**->**Next;

P**->**Next **=** NewCell;

}

**else** {

P**->**Count**++**;

}

}

**void** **DestroyTable**( HashTable H ){

**int** i;

Position P, Tmp;

**for**( i**=**0; i**<**H**->**TableSize; i**++** ) {

P **=** H**->**Heads[i].Next;

**while**( P ) {

Tmp **=** P**->**Next;

free( P );

P **=** Tmp;

}

}

free( H**->**Heads );

free( H );

}

**int** **main**(){

HashTable H;

ElementType word;

**int** length;

H **=** CreateTable();

**while**(1){

length **=** GetAWord( word );

**if**( length**>**0 ){

InsertAndCount( H, word );

}

**else** **break**;

}

Show( H, 10.0**/**100 );

DestroyTable( H );

**return** 0;

}

**实例6.1 六度空间**

**#include**<stdio.h>

**#include**<stdlib.h>

**#define MaxVertexNum 1000**

**#define ERROR -1**

**typedef** **enum** {false, true} **bool**;

**typedef** **int** Vertex;

**typedef** Vertex ElementType;

**typedef** **int** Position;

**struct** QNode {

ElementType **\***Data;

Position Front, Rear;

**int** MaxSize;

};

**typedef** **struct** QNode **\***Queue;

Queue **CreateQueue**( **int** MaxSize ){

Queue Q **=** (Queue)malloc(**sizeof**(**struct** QNode));

Q**->**Data **=** (ElementType **\***)malloc(MaxSize **\*** **sizeof**(ElementType));

Q**->**Front **=** Q**->**Rear **=** **-**1;

Q**->**MaxSize **=** MaxSize;

**return** Q;

}

**bool** **IsFull**( Queue Q ){

**return** ((Q**->**Rear**+**1)**%**Q**->**MaxSize **==** Q**->**Front);

}

**bool** **AddQ**( Queue Q, ElementType X ){

**if** ( IsFull(Q) ) {

printf("队列满");

**return** false;

}

**else** {

Q**->**Rear **=** (Q**->**Rear**+**1)**%**Q**->**MaxSize;

Q**->**Data[Q**->**Rear] **=** X;

**return** true;

}

}

**bool** **IsEmpty**( Queue Q ){

**return** (Q**->**Front **==** Q**->**Rear);

}

ElementType **DeleteQ**( Queue Q ){

**if** ( IsEmpty(Q) ) {

printf("队列空");

**return** ERROR;

}

**else** {

Q**->**Front **=**(Q**->**Front**+**1)**%**Q**->**MaxSize;

**return** Q**->**Data[Q**->**Front];

}

}

**typedef** **struct** ENode **\***PtrToENode;

**struct** ENode{

Vertex V1, V2;

};

**typedef** PtrToENode Edge;

**typedef** **struct** AdjVNode **\***PtrToAdjVNode;

**struct** AdjVNode{

Vertex AdjV;

PtrToAdjVNode Next;

};

**typedef** **struct** Vnode{

PtrToAdjVNode FirstEdge;

} AdjList[MaxVertexNum];

**typedef** **struct** GNode **\***PtrToGNode;

**struct** GNode{

**int** Nv;

**int** Ne;

AdjList G;

};

**typedef** PtrToGNode LGraph;

**#define SIX 6**

**int** Visited[MaxVertexNum];

LGraph **CreateGraph**( **int** VertexNum ){

Vertex V;

LGraph Graph;

Graph **=** (LGraph)malloc( **sizeof**(**struct** GNode) );

Graph**->**Nv **=** VertexNum;

Graph**->**Ne **=** 0;

**for** (V**=**0; V**<**Graph**->**Nv; V**++**)

Graph**->**G[V].FirstEdge **=** NULL;

**return** Graph;

}

**void** **InsertEdge**( LGraph Graph, Edge E ){

PtrToAdjVNode NewNode;

NewNode **=** (PtrToAdjVNode)malloc(**sizeof**(**struct** AdjVNode));

NewNode**->**AdjV **=** E**->**V2;

NewNode**->**Next **=** Graph**->**G[E**->**V1].FirstEdge;

Graph**->**G[E**->**V1].FirstEdge **=** NewNode;

NewNode **=** (PtrToAdjVNode)malloc(**sizeof**(**struct** AdjVNode));

NewNode**->**AdjV **=** E**->**V1;

NewNode**->**Next **=** Graph**->**G[E**->**V2].FirstEdge;

Graph**->**G[E**->**V2].FirstEdge **=** NewNode;

}

LGraph **BuildGraph**(){

LGraph Graph;

Edge E;

**int** Nv, i;

scanf("%d", **&**Nv);

Graph **=** CreateGraph(Nv);

scanf("%d", **&**(Graph**->**Ne));

**if** ( Graph**->**Ne **!=** 0 ) {

E **=** (Edge)malloc( **sizeof**(**struct** ENode) );

**for** (i**=**0; i**<**Graph**->**Ne; i**++**) {

scanf("%d %d", **&**E**->**V1, **&**E**->**V2);

E**->**V1**--**; E**->**V2**--**;

InsertEdge( Graph, E );

}

}

**return** Graph;

}

**void** **InitializeVisited**( **int** Nv ){

Vertex V;

**for** ( V**=**0; V**<**Nv; V**++** )

Visited[V] **=** false;

}

**int** **SDS\_BFS**( LGraph Graph, Vertex S ){

Queue Q;

Vertex V, Last, Tail;

PtrToAdjVNode W;

**int** Count, Level;

Q **=** CreateQueue( MaxVertexNum );

Visited[S] **=** true;

Count **=** 1;

Level **=** 0;

Last **=** S;

AddQ (Q, S);

**while** ( **!**IsEmpty(Q) ) {

V **=** DeleteQ(Q);

**for**( W**=**Graph**->**G[V].FirstEdge; W; W**=**W**->**Next ) {

**if** ( **!**Visited[W**->**AdjV] ) {

Visited[W**->**AdjV] **=** true;

Count**++**;

Tail **=** W**->**AdjV;

AddQ (Q, W**->**AdjV);

}

}

**if** ( V**==**Last ) {

Level**++**;

Last **=** Tail;

}

**if** ( Level**==**SIX ) **break**;

}

**return** Count;

}

**void** **Six\_Degrees\_of\_Separation**( LGraph Graph ) {

Vertex V;

**int** count;

**for**( V**=**0; V**<**Graph**->**Nv; V**++** ) {

InitializeVisited( Graph**->**Nv );

count **=** SDS\_BFS( Graph, V );

printf("%d: %.2f%%\n", V**+**1, 100.0**\***(**double**)count**/**(**double**)Graph**->**Nv);

}

}

**int** **main**(){

LGraph G **=** BuildGraph();

Six\_Degrees\_of\_Separation(G);

**return** 0;

}

**练习7.1 排序**

**#include** <stdio.h>

**#define MAXN 100000**

**int** A[MAXN];

**void** **Insertion\_Sort**( **int** A[], **int** N ) {

**int** P, i, Tmp;

**for** ( P**=**1; P**<**N; P**++** ) {

Tmp **=** A[P];

**for** ( i**=**P; i**>**0 **&&** A[i**-**1]**>**Tmp; i**--** )

A[i] **=** A[i**-**1];

A[i] **=** Tmp;

}

}

**int** **main**(){

**int** N, i;

scanf("%d", **&**N);

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

scanf("%d", **&**A[i]);

Insertion\_Sort(A, N);

printf("%d", A[0]);

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

printf(" %d", A[i]);

printf("\n");

**return** 0;

}

**习题8.1 银行排队问题之单队列多窗口服务**

**#include** <stdio.h>

**#include** <stdlib.h>

**#define MAXN 1000**

**#define MaxWindow 10**

**#define MaxProc 60**

**typedef** **enum**{false, true} **bool**;

**typedef** **struct** People ElementType;

**struct** People {

**int** T;

**int** P;

};

**typedef** **int** Position;

**struct** QNode {

ElementType **\***Data;

Position Front, Rear;

**int** MaxSize;

};

**typedef** **struct** QNode **\***Queue;

Queue **CreateQueue**( **int** MaxSize );

**bool** **IsFull**( Queue Q );

**bool** **AddQ**( Queue Q, ElementType X );

**bool** **IsEmpty**( Queue Q );

ElementType **DeleteQ**( Queue Q );

**int** **FindNextWindow**( **int** W[], **int** K, **int** **\***WaitTime ){

**int** WinAvail;

**int** MinW **=** MaxProc**+**1;

**int** i;

**for** ( i**=**0; i**<**K; i**++** )

**if** ( W[i] **<** MinW ) {

MinW **=** W[i]; WinAvail **=** i;

}

**\***WaitTime **=** MinW;

**for** ( i**=**0; i**<**K; i**++** )

W[i] **-=** MinW;

**return** WinAvail;

}

**void** **QueueingAtBank**( Queue Q, **int** N ){

**struct** People Next;

**int** K;

**int** TotalTime;

**int** CurrentTime;

**int** Window[MaxWindow];

**int** WaitTime;

**int** WinAvail;

**int** i, j;

**int** MaxWait;

**int** cnt[MaxWindow];

scanf("%d", **&**K);

**for** ( i**=**0; i**<**K; i**++** ) {

Window[i] **=** 0;

cnt[i] **=** 0;

}

TotalTime **=** CurrentTime **=** 0;

MaxWait **=** 0;

**while** ( **!**IsEmpty(Q) ) {

WinAvail **=** FindNextWindow( Window, K, **&**WaitTime );

CurrentTime **+=** WaitTime;

Next **=** DeleteQ(Q);

**if** ( CurrentTime **>=** Next.T ) {

TotalTime **+=** (CurrentTime **-** Next.T);

**if** ((CurrentTime **-** Next.T)**>**MaxWait)

MaxWait **=** (CurrentTime **-** Next.T);

}

**else** {

WaitTime **=** Next.T **-** CurrentTime;

**for** ( j**=**0; j**<**K; j**++** ) {

Window[j] **-=** WaitTime;

**if** ( Window[j] **<=** 0 ) {

Window[j] **=** 0;

**if** (j **<** WinAvail) WinAvail **=** j;

}

}

CurrentTime **=** Next.T;

}

Window[WinAvail] **=** Next.P;

cnt[WinAvail]**++**;

}

WaitTime **=** 0;

**for** (i**=**0; i**<**K; i**++**)

**if** (Window[i]**>**WaitTime) WaitTime **=** Window[i];

CurrentTime **+=** WaitTime;

printf("%.1f %d %d\n", ((**double**)TotalTime**/**(**double**)N), MaxWait, CurrentTime);

printf("%d", cnt[0]);

**for**( i**=**1; i**<**K; i**++**) printf(" %d", cnt[i]);

printf("\n");

}

**int** **main**(){

**int** N;

Queue Q;

**int** i;

ElementType X;

scanf("%d", **&**N);

Q **=** CreateQueue(N**+**1);

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

scanf("%d %d", **&**X.T, **&**X.P);

**if** (X.P **>** MaxProc) X.P **=** MaxProc;

AddQ(Q, X);

}

QueueingAtBank(Q, N);

**return** 0;

}

Queue **CreateQueue**( **int** MaxSize ){

Queue Q **=** (Queue)malloc(**sizeof**(**struct** QNode));

Q**->**Data **=** (ElementType **\***)malloc(MaxSize **\*** **sizeof**(ElementType));

Q**->**Front **=** Q**->**Rear **=** 0;

Q**->**MaxSize **=** MaxSize;

**return** Q;

}

**bool** **IsFull**( Queue Q ){

**return** ((Q**->**Rear**+**1)**%**Q**->**MaxSize **==** Q**->**Front);

}

**bool** **AddQ**( Queue Q, ElementType X ){

**if** ( IsFull(Q) ) {

printf("队列满");

**return** false;

}

**else** {

Q**->**Rear **=** (Q**->**Rear**+**1)**%**Q**->**MaxSize;

Q**->**Data[Q**->**Rear] **=** X;

**return** true;

}

}

**bool** **IsEmpty**( Queue Q ){

**return** (Q**->**Front **==** Q**->**Rear);

}

ElementType **DeleteQ**( Queue Q ){

Q**->**Front **=**(Q**->**Front**+**1)**%**Q**->**MaxSize;

**return** Q**->**Data[Q**->**Front];

}

**习题8.2 银行排队问题之单队列多窗口加VIP服务**

**#include** <stdio.h>

**#include** <stdlib.h>

**#define MAXN 1000**

**#define MaxWindow 10**

**#define MaxProc 60**

**typedef** **enum** {false, true} **bool**;

**typedef** **struct** People ElementType;

**struct** People {

**int** T;

**int** P;

**int** VIP;

};

**typedef** **int** Position;

**struct** QNode {

ElementType **\***Data;

Position Front, Rear;

**int** MaxSize;

Position VIPFront, VIPRear;

**int** **\***VIPCustomer;

**int** VIPSize;

};

**typedef** **struct** QNode **\***Queue;

**bool** **VIPIsFull**(Queue Q);

**bool** **AddVIP**(Queue Q, Position P);

**bool** **VIPIsEmpty**(Queue Q);

ElementType **DeleteVIP**(Queue Q);

Queue **CreateQueue**(**int** MaxSize);

**bool** **IsFull**(Queue Q);

**bool** **AddQ**(Queue Q, ElementType X);

**bool** **IsEmpty**(Queue Q);

ElementType **DeleteQ**(Queue Q);

Queue **CreateQueue**(**int** MaxSize)

{

Queue Q **=** (Queue)malloc(**sizeof**(**struct** QNode));

Q**->**Data **=** (ElementType **\***)malloc(MaxSize **\*** **sizeof**(ElementType));

Q**->**Front **=** Q**->**Rear **=** 0;

Q**->**MaxSize **=** MaxSize;

Q**->**VIPCustomer **=** (**int** **\***)malloc(MaxSize **\*** **sizeof**(**int**));

Q**->**VIPFront **=** Q**->**VIPRear **=** 0;

Q**->**VIPSize **=** 0;

**return** Q;

}

**bool** **VIPIsFull**(Queue Q)

{

**return** (Q**->**VIPSize **==** (Q**->**MaxSize **-** 1));

}

**bool** **AddVIP**(Queue Q, Position P)

{

**if** (VIPIsFull(Q)) {

printf("VIP队列满");

**return** false;

} **else** {

Q**->**VIPRear **=** (Q**->**VIPRear **+** 1) **%** Q**->**MaxSize;

Q**->**VIPCustomer[Q**->**VIPRear] **=** P;

Q**->**VIPSize**++**;

**return** true;

}

}

**bool** **IsFull**(Queue Q)

{

**return** ((Q**->**Rear **+** 1) **%** Q**->**MaxSize **==** Q**->**Front);

}

**bool** **AddQ**(Queue Q, ElementType X)

{

**if** (IsFull(Q)) {

printf("队列满");

**return** false;

} **else** {

Q**->**Rear **=** (Q**->**Rear **+** 1) **%** Q**->**MaxSize;

Q**->**Data[Q**->**Rear] **=** X;

**if** (X.VIP **==** 1) **return** AddVIP(Q, Q**->**Rear);

**else** **return** true;

}

}

**bool** **IsEmpty**(Queue Q)

{

**return** (Q**->**Front **==** Q**->**Rear);

}

ElementType **DeleteQ**(Queue Q)

{

ElementType X;

**while** (**!**IsEmpty(Q) **&&** (Q**->**Data[(Q**->**Front **+** 1) **%** Q**->**MaxSize].VIP **==** **-**1))

Q**->**Front **=** (Q**->**Front **+** 1) **%** Q**->**MaxSize;

**if** (IsEmpty(Q)) {

X.T **=** **-**1;

**return** X;

}

Q**->**Front **=** (Q**->**Front **+** 1) **%** Q**->**MaxSize;

**if** (Q**->**Data[Q**->**Front].VIP **==** 1)

X **=** DeleteVIP(Q);

**else**

X **=** Q**->**Data[Q**->**Front];

**return** X;

}

**bool** **VIPIsEmpty**(Queue Q)

{

**return** (Q**->**VIPSize **==** 0);

}

ElementType **DeleteVIP**(Queue Q)

{

ElementType X;

Position P;

**if** (**!**VIPIsEmpty(Q)) {

Q**->**VIPFront **=** (Q**->**VIPFront **+** 1) **%** Q**->**MaxSize;

P **=** Q**->**VIPCustomer[Q**->**VIPFront];

Q**->**VIPSize**--**;

Q**->**Data[P].VIP **=** **-**1;

X **=** Q**->**Data[P];

} **else**

X **=** DeleteQ(Q);

**return** X;

}

**bool** **IsVipHere**(Queue Q, **int** CurrentTime)

{

Position P;

**if** (VIPIsEmpty(Q)) **return** false;

P **=** Q**->**VIPCustomer[(Q**->**VIPFront **+** 1) **%** Q**->**MaxSize];

**if** (Q**->**Data[P].T **>** CurrentTime) **return** false;

**else** **return** true;

}

**int** **FindNextWindow**(**int** W[], **int** K, **int** **\***WaitTime)

{

**int** WinAvail;

**int** MinW **=** MaxProc **+** 1;

**int** i;

**for** (i **=** 0; i **<** K; i**++**)

**if** (W[i] **<** MinW) {

MinW **=** W[i];

WinAvail **=** i;

}

**\***WaitTime **=** MinW;

**for** (i **=** 0; i **<** K; i**++**)

W[i] **-=** MinW;

**return** WinAvail;

}

**void** **QueueingAtBank**(Queue Q, **int** N)

{

**struct** People Next;

**int** K;

**int** TotalTime;

**int** CurrentTime;

**int** Window[MaxWindow];

**int** WaitTime;

**int** WinAvail;

**int** i, j;

**int** MaxWait;

**int** cnt[MaxWindow];

**int** VIPWindow;

scanf("%d %d", **&**K, **&**VIPWindow);

**for** (i **=** 0; i **<** K; i**++**) {

Window[i] **=** 0;

cnt[i] **=** 0;

}

TotalTime **=** CurrentTime **=** 0;

MaxWait **=** 0;

**while** (**!**IsEmpty(Q)) {

WinAvail **=** FindNextWindow(Window, K, **&**WaitTime);

CurrentTime **+=** WaitTime;

**if** ((WinAvail **==** VIPWindow) **&&** (IsVipHere(Q, CurrentTime)))

Next **=** DeleteVIP(Q);

**else**

Next **=** DeleteQ(Q);

**if** (Next.VIP **&&** (Window[VIPWindow] **==** 0)) WinAvail **=** VIPWindow;

**if** (Next.T **==** **-**1) **break**;

**if** (CurrentTime **>=** Next.T) {

TotalTime **+=** (CurrentTime **-** Next.T);

**if** ((CurrentTime **-** Next.T) **>** MaxWait)

MaxWait **=** (CurrentTime **-** Next.T);

} **else** {

WaitTime **=** Next.T **-** CurrentTime;

**for** (j **=** 0; j **<** K; j**++**) {

Window[j] **-=** WaitTime;

**if** (Window[j] **<** 0) {

Window[j] **=** 0;

**if** (j **<** WinAvail) WinAvail **=** j;

}

}

CurrentTime **=** Next.T;

}

**if** (Next.VIP **&&** (Window[VIPWindow] **==** 0)) WinAvail **=** VIPWindow;

Window[WinAvail] **=** Next.P;

cnt[WinAvail]**++**;

}

WaitTime **=** 0;

**for** (i **=** 0; i **<** K; i**++**)

**if** (Window[i] **>** WaitTime) WaitTime **=** Window[i];

CurrentTime **+=** WaitTime;

printf("%.1f %d %d\n", ((**double**)TotalTime **/** (**double**)N), MaxWait, CurrentTime);

printf("%d", cnt[0]);

**for** (i **=** 1; i **<** K; i**++**) printf(" %d", cnt[i]);

printf("\n");

}

**int** **main**()

{

**int** N;

Queue Q;

**int** i;

ElementType X;

scanf("%d", **&**N);

Q **=** CreateQueue(N **+** 1);

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

scanf("%d %d %d", **&**X.T, **&**X.P, **&**X.VIP);

**if** (X.P **>** MaxProc) X.P **=** MaxProc;

AddQ(Q, X);

}

QueueingAtBank(Q, N);

**return** 0;

}

**习题8.3 银行排队问题之单窗口“夹塞” 版**

**#include** <stdio.h>

**#include** <malloc.h>

**#include** <string.h>

**#define MaxProc 60**

**#define MAXTEAM 100**

**#define MAXCHAR 3**

**#define MAXNAME 26426**

**struct** People {

**char** Name[MAXCHAR**+**1];

**int** T;

**int** P;

};

**struct** TeamQueueRecord {

**int** Tfront;

**int** Trear;

**int** Tsize;

**struct** People **\***Customer;

};

**typedef** **struct** TeamQueueRecord **\***TeamQueue;

**struct** QueueRecord {

**int** front;

**int** rear;

**int** size;

TeamQueue TeamQ;

};

**typedef** **struct** QueueRecord **\***Queue;

**int** Team[MAXNAME];

**struct** TeamNode {

**int** Position;

**int** Size;

} TeamInfo[MAXTEAM];

**int** **NameHash**( **char** name[] )

{

**int** i, j;

i **=** name[0] **-** 'A';

j **=** 1;

**while** (name[j]**!=**'\0')

i **=** (i**<<**5) **+** name[j**++**] **-** 'A';

**return** i;

}

**int** **Read\_and\_Set\_Teams**( **int** M )

{

**int** L, MaxL;

**int** i, j;

**char** name[MAXCHAR**+**1];

**for** (i**=**0; i**<**MAXNAME; i**++**)

Team[i] **=** **-**1;

MaxL **=** 0;

**for** (i**=**0; i**<**M; i**++**) {

scanf("%d ", **&**L);

**if** (L **>** MaxL) MaxL **=** L;

**for** (j**=**0; j**<**L; j**++**) {

scanf("%s", name);

Team[NameHash(name)] **=** i;

}

}

**for** (i**=**0; i**<**M; i**++**)

TeamInfo[i].Size **=** 0;

**return** MaxL;

}

Queue **CreateQueue**( **int** MaxQSize, **int** MaxTSize )

{

Queue Q;

**int** i;

Q **=** malloc( **sizeof**( **struct** QueueRecord ) );

Q**->**TeamQ **=** malloc( **sizeof**( **struct** TeamQueueRecord ) **\*** MaxTSize );

Q**->**size **=** 0;

Q**->**front **=** 0;

Q**->**rear **=** **-**1;

**for** ( i**=**0; i**<**MaxTSize; i**++** ) {

Q**->**TeamQ[i].Customer **=**

malloc( **sizeof**( **struct** People ) **\*** MaxQSize );

Q**->**TeamQ[i].Tsize **=** 0;

Q**->**TeamQ[i].Tfront **=** 0;

Q**->**TeamQ[i].Trear **=** **-**1;

}

**return** Q;

}

**void** **AddQ**( Queue Q, **struct** People X )

{

**int** i, pos, r;

**if** ( X.P **>** MaxProc ) X.P **=** MaxProc;

i **=** Team[NameHash(X.Name)];

**if** ( (i **==** **-**1) **||** (**!**TeamInfo[i].Size) ) {

Q**->**rear**++**;

r **=** **++**Q**->**TeamQ[Q**->**rear].Trear;

Q**->**TeamQ[Q**->**rear].Customer[r].T **=** X.T;

Q**->**TeamQ[Q**->**rear].Customer[r].P **=** X.P;

strcpy(Q**->**TeamQ[Q**->**rear].Customer[r].Name, X.Name);

Q**->**TeamQ[Q**->**rear].Tsize**++**;

Q**->**size**++**;

**if** ( i **!=** **-**1 ) {

TeamInfo[i].Position **=** Q**->**rear;

TeamInfo[i].Size**++**;

}

}

**else** {

pos **=** TeamInfo[i].Position;

r **=** **++**Q**->**TeamQ[pos].Trear;

Q**->**TeamQ[pos].Customer[r].T **=** X.T;

Q**->**TeamQ[pos].Customer[r].P **=** X.P;

strcpy(Q**->**TeamQ[pos].Customer[r].Name, X.Name);

Q**->**TeamQ[pos].Tsize**++**;

TeamInfo[i].Size**++**;

}

}

**struct** People **FrontQ**( Queue Q )

{

**struct** People X;

**int** f;

f **=** Q**->**TeamQ[Q**->**front].Tfront;

X.T **=** Q**->**TeamQ[Q**->**front].Customer[f].T;

X.P **=** Q**->**TeamQ[Q**->**front].Customer[f].P;

strcpy(X.Name, Q**->**TeamQ[Q**->**front].Customer[f].Name);

**return** X;

}

**void** **DeleteQ**( Queue Q )

{

**int** i, f;

f **=** Q**->**TeamQ[Q**->**front].Tfront;

Q**->**TeamQ[Q**->**front].Tfront**++**;

Q**->**TeamQ[Q**->**front].Tsize**--**;

i **=** Team[NameHash(Q**->**TeamQ[Q**->**front].Customer[f].Name)];

**if** ( i **!=** **-**1 ) TeamInfo[i].Size**--**;

**if** ( **!**Q**->**TeamQ[Q**->**front].Tsize ) {

Q**->**front**++**;

Q**->**size**--**;

}

}

**int** **IsEmpty**( Queue Q )

{

**return** ( Q**->**size **==** 0 );

}

**struct** People **Enter**( **int** **\***i )

{

**struct** People X;

**char** c;

**if** ((**\***i)) {

scanf("%s %d %d%c", X.Name, **&**X.T, **&**X.P, **&**c);

(**\***i)**--**;

}

**else** X.T **=** **-**1;

**return** X;

}

**double** **QueueingAtBank**( Queue Q, **int** N )

{

**struct** People Next, Wait;

**int** TotalTime, CurrentTime;

**int** i **=** N;

TotalTime **=** CurrentTime **=** 0;

Wait **=** Enter( **&**i );

AddQ( Q, Wait );

**if** (**!**i) {

Next **=** FrontQ(Q);

printf("%s\n", Next.Name);

**return** 0.0;

}

**else** Wait **=** Enter( **&**i );

**while** ( **!**IsEmpty(Q) **||** (Wait.T **>=** 0) ) {

**if** ( **!**IsEmpty(Q) ) {

Next **=** FrontQ(Q);

printf("%s\n", Next.Name);

**if** ( CurrentTime **>=** Next.T )

TotalTime **+=** (CurrentTime **-** Next.T);

**else**

CurrentTime **=** Next.T;

CurrentTime **+=** Next.P;

**while** ( (Wait.T **>=** 0) **&&** (Wait.T **<=** CurrentTime) ) {

AddQ( Q, Wait );

Wait **=** Enter( **&**i );

}

DeleteQ(Q);

}

**else** {

AddQ( Q, Wait );

Wait **=** Enter( **&**i );

}

}

**return** ((**double**)TotalTime**/**(**double**)N);

}

**int** **main**()

{

**int** N, M, MaxQSize;

Queue Q;

scanf("%d %d\n", **&**N, **&**M);

MaxQSize **=** Read\_and\_Set\_Teams(M);

Q **=** CreateQueue( MaxQSize, N );

printf("%.1lf\n", QueueingAtBank( Q, N ));

**return** 0;

}

**习题8.4 畅通工程之最低成本建设问题**

**#include** <stdio.h>

**#include** <stdlib.h>

**typedef** **enum** {false, true} **bool**;

**#define MaxVertexNum 1000**

**typedef** **int** Vertex;

**typedef** **int** WeightType;

**typedef** **struct** ENode **\***PtrToENode;

**struct** ENode {

Vertex V1, V2;

WeightType Weight;

};

**typedef** PtrToENode Edge;

**typedef** **struct** AdjVNode **\***PtrToAdjVNode;

**struct** AdjVNode {

Vertex AdjV;

WeightType Weight;

PtrToAdjVNode Next;

};

**typedef** **struct** Vnode {

PtrToAdjVNode FirstEdge;

} AdjList[MaxVertexNum];

**typedef** **struct** GNode **\***PtrToGNode;

**struct** GNode {

**int** Nv;

**int** Ne;

AdjList G;

};

**typedef** PtrToGNode LGraph;

LGraph **CreateGraph**(**int** VertexNum) {

Vertex V;

LGraph Graph;

Graph **=** (LGraph)malloc(**sizeof**(**struct** GNode));

Graph**->**Nv **=** VertexNum;

Graph**->**Ne **=** 0;

**for** (V **=** 0; V **<** Graph**->**Nv; V**++**)

Graph**->**G[V].FirstEdge **=** NULL;

**return** Graph;

}

**void** **InsertEdge**(LGraph Graph, Edge E) {

PtrToAdjVNode NewNode;

NewNode **=** (PtrToAdjVNode)malloc(**sizeof**(**struct** AdjVNode));

NewNode**->**AdjV **=** E**->**V2;

NewNode**->**Weight **=** E**->**Weight;

NewNode**->**Next **=** Graph**->**G[E**->**V1].FirstEdge;

Graph**->**G[E**->**V1].FirstEdge **=** NewNode;

NewNode **=** (PtrToAdjVNode)malloc(**sizeof**(**struct** AdjVNode));

NewNode**->**AdjV **=** E**->**V1;

NewNode**->**Weight **=** E**->**Weight;

NewNode**->**Next **=** Graph**->**G[E**->**V2].FirstEdge;

Graph**->**G[E**->**V2].FirstEdge **=** NewNode;

}

LGraph **BuildGraph**() {

LGraph Graph;

Edge E;

**int** Nv, i;

scanf("%d", **&**Nv);

Graph **=** CreateGraph(Nv);

scanf("%d", **&**(Graph**->**Ne));

**if** (Graph**->**Ne **!=** 0) {

E **=** (Edge)malloc(**sizeof**(**struct** ENode));

**for** (i **=** 0; i **<** Graph**->**Ne; i**++**) {

scanf("%d %d %d", **&**E**->**V1, **&**E**->**V2, **&**E**->**Weight);

E**->**V1**--**;

E**->**V2**--**;

InsertEdge(Graph, E);

}

}

**return** Graph;

}

**void** **PercDown**(Edge ESet, **int** p, **int** N) {

**int** Parent, Child;

**struct** ENode X;

X **=** ESet[p];

**for** (Parent **=** p; (Parent **\*** 2 **+** 1) **<** N; Parent **=** Child) {

Child **=** Parent **\*** 2 **+** 1;

**if** ((Child **!=** N **-** 1) **&&** (ESet[Child].Weight **>** ESet[Child **+** 1].Weight))

Child**++**;

**if** (X.Weight **<=** ESet[Child].Weight)

**break**;

**else**

ESet[Parent] **=** ESet[Child];

}

ESet[Parent] **=** X;

}

**void** **InitializeESet**(LGraph Graph, Edge ESet) {

Vertex V;

PtrToAdjVNode W;

**int** ECount;

ECount **=** 0;

**for** (V **=** 0; V **<** Graph**->**Nv; V**++**) {

**for** (W **=** Graph**->**G[V].FirstEdge; W; W **=** W**->**Next) {

**if** (V **<** W**->**AdjV) {

ESet[ECount].V1 **=** V;

ESet[ECount].V2 **=** W**->**AdjV;

ESet[ECount**++**].Weight **=** W**->**Weight;

}

}

}

**for** (ECount **=** Graph**->**Ne **/** 2; ECount **>=** 0; ECount**--**)

PercDown(ESet, ECount, Graph**->**Ne);

}

**void** **Swap**(Edge a, Edge b) {

**struct** ENode t;

t **=** **\***a;

**\***a **=** **\***b;

**\***b **=** t;

}

**int** **GetEdge**(Edge ESet, **int** CurrentSize) {

Swap(**&**ESet[0], **&**ESet[CurrentSize **-** 1]);

PercDown(ESet, 0, CurrentSize **-** 1);

**return** CurrentSize **-** 1;

}

**typedef** Vertex ElementType;

**typedef** **int** SetName;

**typedef** ElementType SetType[MaxVertexNum];

**void** **InitializeVSet**(SetType S, **int** N) {

**int** i;

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

S[i] **=** **-**1;

}

SetName **Find**(SetType S, ElementType X) {

**if** (S[X] **<** 0)

**return** X;

**else**

**return** S[X] **=** Find(S, S[X]);

}

**void** **Union**(SetType S, SetName Root1, SetName Root2) {

**if** (S[Root2] **<** S[Root1]) {

S[Root2] **+=** S[Root1];

S[Root1] **=** Root2;

} **else** {

S[Root1] **+=** S[Root2];

S[Root2] **=** Root1;

}

}

**bool** **CheckCycle**(SetType VSet, Vertex V1, Vertex V2) {

Vertex Root1, Root2;

Root1 **=** Find(VSet, V1);

Root2 **=** Find(VSet, V2);

**if** (Root1 **==** Root2)

**return** false;

**else** {

Union(VSet, Root1, Root2);

**return** true;

}

}

**int** **Kruskal**(LGraph Graph) {

WeightType TotalWeight;

**int** ECount, NextEdge;

SetType VSet;

Edge ESet;

InitializeVSet(VSet, Graph**->**Nv);

ESet **=** (Edge)malloc(**sizeof**(**struct** ENode) **\*** Graph**->**Ne);

InitializeESet(Graph, ESet);

TotalWeight **=** 0;

ECount **=** 0;

NextEdge **=** Graph**->**Ne;

**while** (ECount **<** Graph**->**Nv **-** 1) {

NextEdge **=** GetEdge(ESet, NextEdge);

**if** (NextEdge **<** 0)

**break**;

**if** (CheckCycle(VSet, ESet[NextEdge].V1, ESet[NextEdge].V2) **==** true) {

TotalWeight **+=** ESet[NextEdge].Weight;

ECount**++**;

}

}

**if** (ECount **<** Graph**->**Nv **-** 1)

TotalWeight **=** **-**1;

**return** TotalWeight;

}

**int** **main**() {

**int** w;

LGraph G **=** BuildGraph();

w **=** Kruskal(G);

**if** (w **==** **-**1)

printf("Impossible\n");

**else**

printf("%d\n", w);

**return** 0;

}

**习题8.5 畅通工程之局部最小花费问题**

**#include** <stdio.h>

**#define MAXN 101**

**#define INF 1000000000**

**int** belongto[MAXN], newcost[MAXN][MAXN], cost[MAXN][MAXN];

**int** data[MAXN][MAXN];

**int** n, newn;

**int** **min**(**int** a, **int** b) {

**return** (a **<** b) **?** a : b;

}

**long** **prim**(**int** n, **int** mat[][MAXN]) {

**int** min[MAXN];

**long** ret **=** 0;

**int** v[MAXN], i, j, k;

**for** (i **=** 0; i **<** n; i**++**) {

min[i] **=** INF;

v[i] **=** 0;

}

**for** (min[j **=** 0] **=** 0; j **<** n; j**++**) {

**for** (k **=** **-**1, i **=** 0; i **<** n; i**++**) {

**if** (**!**v[i] **&&** (k **==** **-**1 **||** min[i] **<** min[k])) {

k **=** i;

}

}

v[k] **=** 1;

ret **+=** min[k];

**for** (i **=** 0; i **<** n; i**++**) {

**if** (**!**v[i] **&&** mat[k][i] **<** min[i]) {

min[i] **=** mat[k][i];

}

}

}

**return** ret;

}

**void** **dummy**() {

**int** i, j, temp;

**for** (i **=** 0; i **!=** newn; **++**i) {

**for** (j **=** 0; j **!=** newn; **++**j) newcost[i][j] **=** INF;

}

**for** (i **=** 0; i **!=** n; **++**i) **--**belongto[i];

**for** (i **=** 0; i **!=** n; **++**i) {

**for** (j **=** 0; j **!=** n; **++**j) {

temp **=** min(cost[i][j], newcost[belongto[i]][belongto[j]]);

newcost[belongto[i]][belongto[j]] **=** newcost[belongto[j]][belongto[i]] **=** temp;

}

}

}

**int** **findComponents**(**int** n, **int** mat[][MAXN], **int** **\***id) {

**int** ret, k, i, j, m;

**for** (k **=** 0; k **<** n; id[k**++**] **=** 0);

**for** (ret **=** k **=** 0; k **<** n; k**++**) {

**if** (**!**id[k]) {

**for** (id[k] **=** **-**1, ret**++**, m **=** 1; m;) {

**for** (m **=** i **=** 0; i **<** n; i**++**) {

**if** (id[i] **==** **-**1) {

**for** (m**++**, id[i] **=** ret, j **=** 0; j **<** n; j**++**) {

**if** (**!**id[j] **&&** mat[i][j]) {

id[j] **=** **-**1;

}

}

}

}

}

}

}

**return** ret;

}

**int** **main**() {

**int** i;

scanf("%d", **&**n);

**for** (i **=** 0; i **!=** n **\*** (n **-** 1) **/** 2; **++**i) {

**int** x, y, t, link;

scanf("%d %d %d %d", **&**x, **&**y, **&**t, **&**link);

**--**x;

**--**y;

cost[x][y] **=** cost[y][x] **=** t;

**if** (link) data[x][y] **=** data[y][x] **=** 1;

**else** data[x][y] **=** data[y][x] **=** 0;

}

newn **=** findComponents(n, data, belongto);

dummy();

printf("%ld\n", prim(newn, newcost) **%** INF);

**return** 0;

}