1. 链式存储的一元多项式乘法运算的算法

```
typedef struct Node* Ptr;
struct Node{
    int coef;
    int expon;
    Ptr next;
};
typedef Ptr List;

List Init(int c,int e){//O(1)
    List L;
    L=(List)malloc(sizeof(struct Node));
    L->expon=e;
    L->coef=c;
    L->next=NULL;
    return L;
}

int size(List L){//O(n)
    List t1=L;
    int res=0;
    while(t1){
        t1=t1->next;
        res++;
    }
    return res;
}
List Mul(List L1,List L2){//O(m*n)
    int len1= size(L1),len2= size(L2);
    List res= Init(0,0),t1=L1;
    while(t1){
        List t2=L2;
        while(t2){
            Add(res,t1->coef*t2->coef,t1->expon+t2->expon);
            t2=t2->next;
        }
        t1=t1->next;
    }
    Sort(res);
    return res;
}
```

2.符号排序

```
typedef int Pos;
typedef struct Node* Ptr;
struct Node{
    char *v;
    Pos top;
    int size;
};
typedef Ptr Stack;
Stack Init(int size){
    Stack res=(Stack)malloc(sizeof(struct Node));
    res->v=(char*) malloc(size*sizeof(char));
    res->top=-1;
    res->size=size;
    return res;
```

```
}
bool pair(char a,char b){
    switch (a) {
        case '(':return b==')';
        case '[':return b==']';
        case '{':return b=='}';
        case '<':return b=='>';
    }
}
void push(Stack s,char v){
    s->v[++(s->top)]=v;
}
void pop(Stack s){
    if(s->top<0)return;
    s->top--;
}
bool check(char x[]){
    Stack s= Init(9999);
    int i=0;
    while(x[i+1]!='\0'){
            if (x[i] == '(' || x[i] == '[' || x[i] == '{' || (x[i] == '/' && x[i + 1] == '*'))
                push(s, x[i] == '/' ? '<' : x[i]);
            if (x[i] == ')' || x[i] == ']' || x[i] == '}' || (x[i] == '*' && x[i + 1] == '/')){
                if(pair(s->v[s->top],x[i]=='*'?'>':x[i])){
                    pop(s);
                    if(x[i]=='*') i++;
                }else{
                    return false;
                }
            }
        i++;
    }
    if (x[i] == '(' || x[i] == '[' || x[i] == '{' )return false;
    if ((x[i] == ')' || x[i] == ']' || x[i] == '}') && s->top==0) {
        return pair(s->v[s->top],x[i]);
    }
    return s->top==-1;
}
```

中缀表达式转换为后缀表达式

```
typedef int Position;
typedef double ElementType;
typedef struct SNode* PtrToSNode;
struct SNode{
    ElementType *Data;
    Position Top;
    int MaxSize;
};
typedef PtrToSNode Stack;
Stack CreateStack(int MaxSize){
    Stack S=(Stack)malloc(sizeof(struct SNode));
    S->Data=(ElementType*)malloc(MaxSize*sizeof(ElementType));
    S->Top=-1;
    S->MaxSize=MaxSize;
    return S;
}
ElementType Top(Stack s){
    return s->Data[s->Top];
}
bool IsFull(Stack S){
```

```
    return (S->Top==S->MaxSize-1);
}
bool Push(Stack S,ElementType X){
    if(IsFull(S)){
        printf("Stack Full\n");
        return false;
    }else{
        S->Data[++(S->Top)]=X;
        return true;
    }
}
bool IsEmpty(Stack S){
    return(S->Top==-1);
}
ElementType Pop(Stack S){
    if(IsEmpty(S)){
        printf("Stack Empty\n");
        return -1;
    }else{
        return(S->Data[(S->Top)--]);
    }
}
typedef enum{num,opr,end} Type;

Type GetOp(char* Expr, int* start, char* str){
    int i=0;
    while((str[0]=Expr[(*start)++])==' ');

    while(str[i]!=' '&&str[i]!='\0')
        str[++i]=Expr[(*start)++];
    if(str[i]=='\0')
        (*start)--;
    str[i]='\0';
    if(i==0){return end;}
    else if(isdigit(str[0])||isdigit(str[1]))
        return num;
    else
        return opr;
}
int precede(char op){
    int x;
    switch(op){
        case '*': x=2; break;
        case '/': x=2; break;
        case '+': x=1; break;
        case '-': x=1; break;
        default : x=0;
    }
    return x;
}

char *Reserve(char *e) {
    Stack s1 = CreateStack(100);
    char *c;
    c = (char *) malloc(sizeof(char) * 50);
    int i = 0, j = 0;
    char ch;
    Push(s1, '@');
    ch = e[i++];
    while (ch != 0) {
        if (ch == '(') {
            Push(s1, ch);
            ch = e[i++];
        } else if (ch == ')') {
```

```
            while (Top(s1) != '(') {
                c[j++] = Pop(s1);
                c[j++]=' ';
            }
            ch = Pop(s1);
            ch = e[i++];
        } else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            char w;
            w = Top(s1);
            while (precede(w) >= precede(ch)) {
                c[j++] = Pop(s1);
                c[j++]=' ';
                w = Top(s1);
            }
            Push(s1, ch);
            ch = e[i++];
        } else {
            c[j++]=ch;
            ch=e[i++];
            c[j++]=' ';
        }
    }
    ch = Pop(s1);
    while (ch != '@') {
        c[j++] = ch;
        c[j++] = ' ';
        ch = Pop(s1);
    }
    c[j++]='\0';
    return c;
}
ElementType PostfixExp(char* Expr){
    Stack S;
    Type T;
    ElementType Op1,Op2;
    char str[MAXOP];
    int start = 0;
    S= CreateStack(MAXOP);
    Op1=Op2=0;
    while( (T=GetOp(Expr,&start,str))!=end){
        if(T==num)
            Push(S, atof(str));
        else{
            if(!IsEmpty(S)) Op2= Pop(S);
            else Op2=INFINITY;
            if(!IsEmpty(S)) Op1= Pop(S);
            else Op2=INFINITY;
            switch (str[0]) {
                case '+':Push(S,Op1+Op2);break;
                case '*':Push(S,Op1*Op2);break;
                case '-':Push(S,Op1-Op2);break;
                case'/':if(Op2!=0.0)Push(S,Op1/Op2);else {printf("Wrong!\n");Op2=INFINITY;}break;
                default:printf("Wrong Opeator\n");Op2=INFINITY;break;
            }
            if(Op2 >= INFINITY)break;
        }
    }
    if(Op2<INFINITY)
        if(!IsEmpty(S))
            Op2= Pop(S);
        else Op2=INFINITY;
        free(S);
        return Op2;
```

}4.双端队列
```
#include <stdio.h>
#include <stdbool.h>
#include <malloc.h>
typedef int ElementType;
typedef int Pos;
typedef struct Node* Ptr;
struct Node{
	ElementType *v;
	Pos front,rear;
	int size;
};
typedef Ptr Dqueue;

Dqueue Init(int size){
	Dqueue res=(Dqueue)malloc(sizeof(struct Node));
	res->v=(ElementType*) malloc(size*sizeof(ElementType));
	res->front=res->rear=0;
	res->size=size;
	return res;
}
bool IsFull(Dqueue Q){
	return((Q->rear+1)%Q->size==Q->front);
}
bool IsEmpty(Dqueue Q){
	return(Q->front==Q->rear);
}
bool Inject(Dqueue Q,ElementType X){
	if(IsFull(Q)){
		printf(("Dqueue is full\n"));
		return false;
	}
	else{
		Q->rear=(Q->rear+1)%Q->size;
		Q->v[Q->rear]=X;
		return true;
	}
}

ElementType Eject(Dqueue Q){
	if(IsEmpty(Q)){
		printf("Dqueue is empty\n");
		return -1;
	}else{
		ElementType t=Q->v[Q->rear];
		Q->rear==0?Q->rear=Q->size-1:Q->rear--;
		return t;
	}
}

bool Push(Dqueue Q,ElementType X){
	if(IsFull(Q)){
		printf("Dqueue is full\n");
		return false;
	}else{
		Q->v[Q->front]=X;
		Q->front==0?Q->front=Q->size-1:Q->front--;
		return true;
	}
}
ElementType Pop(Dqueue Q){
	if(IsEmpty(Q)){
		printf("Dqueue is empty\n");
		return -1;
```

```
    }else{
        Q->front=(Q->front+1)%Q->size;
        return Q->v[Q->front];
    }
}


int main() {
    printf("DQueue test:\n1-Push 2-Inject 3-Pop 4-Eject -1Exit\n");
    Dqueue Q= Init(200);
    int i,k;
    scanf("%d",&i);
    while(i!=-1){
        switch (i) {
            case 1:
                scanf("%d",&k);
                Push(Q,k);
                break;
            case 2:
                scanf("%d",&k);
                Inject(Q,k);
                break;
            case 3:
                printf("value= %d \n",Pop(Q));
                break;
            case 4:
                printf("value= %d \n",Eject(Q));
                break;
        }
        scanf("%d",&i);
    }
    return 0;
}
```

1.后序遍历和中序遍历结果：

```
#include <stdio.h>
#include <malloc.h>
#include <stdbool.h>
typedef int ElementType;
typedef struct TNode* Position;
typedef Position BinTree;
struct TNode{
    ElementType Data;
    BinTree Left;
    BinTree Right;
};
BinTree CreatTree(ElementType *in,ElementType *post,int N){
    BinTree T;
    int p;
    if(!N)return NULL;
    T=(BinTree) malloc(sizeof(struct TNode));
    T->Data=post[N-1];
    T->Left=T->Right=NULL;
    for(p=0;p<N;p++)
        if(in[p]==post[N-1])break;
    T->Left= CreatTree(in,post,p);
    T->Right= CreatTree(in+p+1,post+p,N-p-1);
    return T;
}
bool Is_Search(BinTree tree){
    bool flag=true;
```

```
    if(tree->Left && tree->Left->Data>tree->Data) flag=false;
    if(tree->Right && tree->Right->Data<tree->Data) flag=false;
    if(tree->Left &&flag) flag=Is_Search(tree->Left);
    if(tree->Right &&flag) flag=Is_Search(tree->Right);
    return flag;
}
void printPreorder(BinTree tree){
    if(!tree)return;
    printf("%d ",tree->Data);
    printPreorder(tree->Left);
    printPreorder(tree->Right);
}
int main() {
    int n,i,t;
    int in[1005],post[1005];
    BinTree tree;
    printf("Begin test:type sum of number\n");
    scanf("%d",&n);
    for(i=0;i<n;i++){
        scanf("%d",&t);
        in[i]=t;
    }
    for(i=0;i<n;i++) {
        scanf("%d", &t);
        post[i] = t;
    }
    tree=CreatTree(in,post,n);
    printPreorder(tree);
    if(Is_Search(tree))
        printf("\nIs BST\n");
    else
        printf("\nNot BST\n");
    return 0;
}
```

计算二叉树最大的宽度

```
#include <stdio.h>
#include <malloc.h>
#include <stdbool.h>

typedef int ElementType;
typedef struct TNode* TreePos;
typedef TreePos BinTree;
struct TNode{
    ElementType Data;
    BinTree Left;
    BinTree Right;
};

typedef BinTree ElementType2;
typedef int Position;
typedef struct QNode* PtrToQNode;
struct QNode{
    ElementType2* Data;
    Position Front,Rear;
    int MaxSize;
};
typedef PtrToQNode Queue;

Queue CreateQueue(int MaxSize){
    Queue Q=(Queue)malloc(sizeof(struct QNode));
    Q->Data=(ElementType2*) malloc(MaxSize*sizeof(ElementType2));
    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,ElementType2 X){
    if(IsFull(Q)){
        printf("error queue is full\n");
        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);
}

int Q_Length(Queue q){
    return(q->Rear - q->Front + q->MaxSize) % q->MaxSize;
}
ElementType2 DeleteQ(Queue Q){
    if(IsEmpty(Q)){
        printf("queue is empty\n");
        return -1;
    }else{
        Q->Front=(Q->Front+1)%Q->MaxSize;
        return Q->Data[Q->Front];
    }
}

int Max_Lenth(BinTree T){
    Queue Q;
    int res=1;
    if(!T)return 0;
    Q= CreateQueue(10005);
    AddQ(Q,T);
    while(!IsEmpty(Q)){
        int c= Q_Length(Q);
        //printf("c=%d\n",c);
        res=c>res?c:res;
        while(c--){
            BinTree BT= DeleteQ(Q);
            if(BT->Left) AddQ(Q,BT->Left);
            if(BT->Right) AddQ(Q,BT->Right);
        }
    }
    return res;
}

BinTree CreatTree(ElementType *in,ElementType *post,int N){
    BinTree T;
    int p;
    if(!N)return NULL;
    T=(BinTree) malloc(sizeof(struct TNode));
    T->Data=post[N-1];
    T->Left=T->Right=NULL;
    for(p=0;p<N;p++)
        if(in[p]==post[N-1])break;
    T->Left= CreatTree(in,post,p);
```

```
        T->Right= CreatTree(in+p+1,post+p,N-p-1);
        return T;
    }

    int main() {
        int n,i,t;
        int in[10005],post[10005];
        BinTree tree;
        printf("Create BinTree:type sum of number\n");
        scanf("%d",&n);
        for(i=0;i<n;i++){
            scanf("%d",&t);
            in[i]=t;
        }
        for(i=0;i<n;i++) {
            scanf("%d", &t);
            post[i] = t;
        }
        tree=CreatTree(in,post,n);
        printf("\nmax_lenth is %d\n", Max_Lenth(tree));
        return 0;
}
```

求最近的公共祖先结点的编号

```
#include <stdio.h>
#include <malloc.h>
#include <stdbool.h>


#define Swap(a,b) a^=b,b^=a,a^=b;
typedef int ElementType;
typedef struct LNode *PtrToLNode;
struct LNode{
    ElementType Data[10005];
    int Last;
};
typedef PtrToLNode List;
typedef List Tree;

int NCA(int p1,int p2){
    while(p1!=p2){
        if(p1>p2){Swap(p1,p2)}
        while(p2>p1)p2/=2;
    }
    return p1;
}

int main() {
    int n,i,p1,p2,p;
    Tree T;
    T=(Tree) malloc(sizeof(struct LNode));
    T->Data[0]=0;
    T->Last=0;
    scanf("%d",&n);
    for(T->Last=1;T->Last<=n;T->Last++)
        scanf("%d",&T->Data[T->Last]);
    T->Last--;
    scanf("%d %d",&p1,&p2);
    if(!T->Data[p1])printf("NULL\n");
    else if(!T->Data[p2])printf("NULL\n");
    else{
        p=NCA(p1,p2);
        printf("%d %d\n",p,T->Data[p]);
    }
    return 0;
```

}
邻接表-邻接矩阵
```
#include <stdio.h>
int G[999][999];
//void printG(int **G,int n)
void printG(int G[999][999], int n) {
    int i, k;
    for (i = 0; i < n; i++) {
        for (k = 0; k < n; k++) {
            printf("%d ", G[i][k]);
        }
        printf("\n");
    }
}
int main() {
    printf("Input:\n");
    int n, m;
    scanf_s("%d", &n); scanf_s("%d", &m);
    while (m--) {
        int t1, t2, t3;
        scanf_s("%d", &t1); scanf_s("%d", &t2); scanf_s("%d", &t3);
        G[t1][t2] = t3;
        G[t2][t1] = t3;
    }
    printG(&G, n);
    return 0;
}
```
迪杰斯特拉
```
#include <stdio.h>
#include<stdbool.h>
#include<stdlib.h>
typedef int Vertex;
typedef struct GNode *PtrToGNode;
#define INFINITY INT_MAX
#define MaxVertexNum 999
struct GNode {
    int Nv;
    int G[999][999];
    Vertex S, D;
};
typedef PtrToGNode MGraph;

Vertex FindMinDist(MGraph Graph, int dist[], int collected[])
{
    Vertex MinV, V;
    int MinDist = INFINITY;

    for (V = 0; V < Graph->Nv; V++) {
        if (collected[V] == false && dist[V] < MinDist) {
            MinDist = dist[V];
            MinV = V;
        }
    }
    if (MinDist < INFINITY)
        return MinV;
    else return -1;
}
void DFS(int path[],int dist[],int n,int *sum) {
    if (n == 0) { printf("0->"); return; }
    *sum += dist[n];
    DFS(path,dist, path[n],&sum);
    printf("%d ->", n);
}
bool Dijkstra(MGraph Graph, int dist[], int path[], Vertex S)
```

```
{
    int collected[MaxVertexNum];
    Vertex V, W;
    for (V = 0; V < Graph->Nv; V++) {
        dist[V] = Graph->G[S][V];
        if (dist[V] < INFINITY)
            path[V] = S;
        else
            path[V] = -1;
        collected[V] = false;
    }
    dist[S] = 0;
    collected[S] = true;
    while (1) {
        V = FindMinDist(Graph, dist, collected);
        if (V == -1)
            break;
        collected[V] = true;
        for (W = 0; W < Graph->Nv; W++)
            if (collected[W] == false && Graph->G[V][W] < INFINITY) {
                if (Graph->G[V][W] < 0)
                    return false;
                if (dist[V] + Graph->G[V][W] < dist[W]) {
                    dist[W] = dist[V] + Graph->G[V][W];
                    path[W] = V;
                }
            }
    }
    return true;
}
int main(){
    int i,k,n,sum=0;
    FILE* fp;
    fp = fopen("D:\\123.txt", "r");
    int dist[99], path[99];
    if (fp == NULL)
    {
        printf("fail to open! \n");
        return -1;
    }
    fscanf(fp, "%d", &n);
    MGraph G;
    G = (MGraph)malloc(sizeof(struct GNode));
    G->Nv = n;
    for (i = 0; i < n; i++)
        for (k = 0; k < n; k++) {
            fscanf(fp, "%d", &G->G[i][k]);
            G->G[i][k] = G->G[i][k] == 0 ? G->G[i][k] = INFINITY : G->G[i][k];
        }
    Dijkstra(G, dist,path, 0);
    for (i = 0; i < n; i++) {
        DFS(path,dist, i,&sum);
        printf("    distance=%d\n", sum);
        sum = 0;
    }
    fclose(fp);
    return 0;
```

六度空间

```
#include <stdio.h>
#include<queue>
#include<stdlib.h>
using namespace std;

int G[9999][9999];
```

```
bool vis[9999];
int dis[9999];
int n;
int BFS(int index) {
    queue<int> q;
    int i,sum=0,last=index,res=1,ll;
    q.push(index);
    vis[index] = true;
    while (!q.empty()) {
        if (sum == 6)break;
        int t = q.front();
        for (i = 0; i < n; i++) {
            if (G[t][i] != 0 && !vis[i]) {
                res++;
                q.push(i);
                vis[i] = true;
                ll = i;
            }
        }
        if (t == last) { sum++; last = ll; }
        q.pop();
    }
    return res;
}
int main() {
    int i, k, m, sum = 0;
    scanf_s("%d", &n); scanf_s("%d", &m);
    while (m--) {
        int t1, t2;
        scanf_s("%d", &t1); scanf_s("%d", &t2);
        G[--t1][--t2] = 1; G[t2][t1] = 1;
    }
    for (i = 0;i < n; i++) {
        memset(vis, false, sizeof(bool) * 999);
        printf("%d: %.2f%%\n",i+1,(float)BFS(i)*100/(float)n);
    }
    return 0;
}
```