**实验三 蛮力法（枚举）**

1. **实验目的**
2. 掌握蛮力算法的基本思想。
3. 熟练掌握蛮力算法求解问题的过程。
4. **实验要求**
5. 能运用蛮力法解决实际问题，加深对蛮力算法的理解及应用；
6. 理解蛮力算法的适用范围，体会蛮力算法的优缺点。
7. 得到正确的程序运行结果
8. **主要仪器及耗材**

计算机及相关软件

1. **实验内容**

1. 设计一个蛮力算法,对于给定的x0,计算下面多项式的值:

P(x)=anxn+an-1xn-1+…+a1x+a0

**问题：**

1. 这个算法最坏情况下的效率类型是什么？

**O(n\*n),分析在下面。**

1. 如果你设计的算法属于Θ(n2),请你为该算法设计一个线性的算法。

**见函数fx2.**

1. 对于该问题来说,能不能设计一个比线性效率还要好的算法呢?

**不能，线性效率已经最好了。**

**因为计算任意一个多项式在任意点x的值，都必须处理它的n+1个系数。**

1. 编程实现你设计的算法。

**/\***

**P(x)=a(n)\*pow(x,n)+a(n-1)\*pow(x,n-1)+...+a(1)\*pow(x,1)+a(0)**

**\*/**

**#include<iostream>**

**#include<cstdio>**

**using namespace std;**

**int a[2000];**

**//O(n\*n) 按顺序直接算**

**int fx1(int n,int x) {**

**int power,p=0;**

**for(int i=n; i>=0; i--) {**

**power=1;**

**for(int j=1; j<=i; j++) {**

**power=power\*x;**

**}**

**p=p+a[i]\*power;**

**}**

**return p;**

**}**

**//O(n) 从后往前按X的次方算**

**int fx2(int n,int x) {**

**int power=1,p=a[0];**

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

**power=power\*x; //x的n次幂**

**p=p+a[i]\*power;**

**}**

**return p;**

**}**

**int main() {**

**int n=3;**

**//printf("输入n的值和系数：");**

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

**//for(int i=0;i<=n;i++)**

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

**a[3]=3,a[2]=2,a[1]=1,a[0]=1;**

**printf("输入x的值：");**

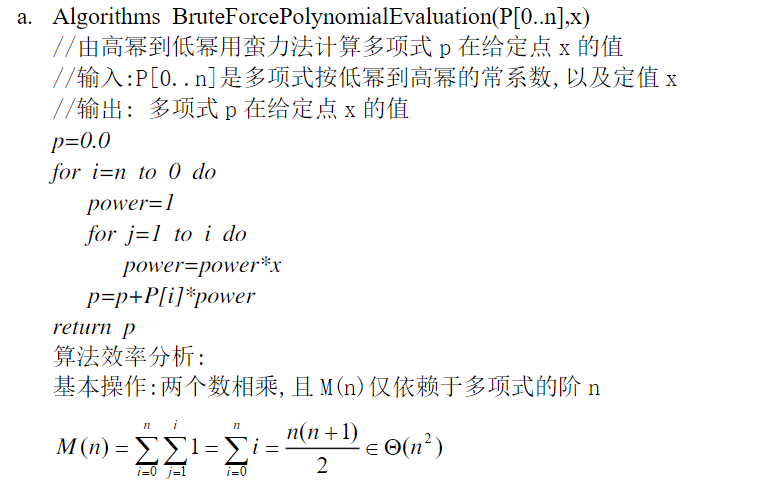
**int x;**

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

**printf("%d\n",fx2(n,x));**

**return 0;**

**}**



2. 应用选择排序对序列E,X,A,M,P,L,E按照字母顺序排序。算法描述如下：

算法 SelectionSort(A[0..n-1])

//该算法用选择排序对给定的数组排序

//输入：一个可排序数组A[0..n-1]

//输出：升序排序的数组A[0..n-1]

for i←0 to n-2 do

min ←i

for j←i+1 to n-1 do

if A[j]< A[min]

min ←j

swap A[i] and A[min]

**问题：**

1. 选择排序稳定吗？

**不稳定！**

②. 如果用链表实现以上的排序，能不能获得和数组版相同的Θ(n2)的效率？

**链表交换，可以的！**

1. 编程证明你的结论。

**/\***

**应用选择排序对序列E,X,A,M,P,L,E按照字母顺序排序。**

**选择排序不稳定，存在交换操作**

**\*/**

**#include<iostream>**

**#include<cstdio>**

**using namespace std;**

**char A[]= {'E','X','A','M','P','L','E'};**

**int n=7;**

**void SelectionSort() {**

**int minn;**

**for(int i=0; i<=n-2; i++) {**

**minn=i;**

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

**if(A[j]<A[minn])**

**minn=j;**

**}**

**swap(A[i],A[minn]);**

**}**

**}**

**int main() {**

**SelectionSort();**

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

**printf("%c ",A[i]);**

**return 0;**

**}**

3. 应用冒泡排序对序列E,X,A,M,P,L,E按照字母顺序排序。算法描述如下：

算法 BubbleSort(A[0..n-1])

//该算法用冒泡排序对给定的数组排序

//输入：一个可排序数组A[0..n-1]

//输出：升序排序的数组A[0..n-1]

for i←0 to n-2 do

for j←0 to n-2-i do

if A[j+1]< A[j]

swap A[j] and A[j+1]

**问题：**

1. 冒泡排序稳定吗？

**稳定**

1. 请证明,如果对列表比较一遍之后没有交换元素的位置,那么这个表已经排好序了,算法可以停止了。

**如果某一趟排序没有发生位置的变化，说明当前的序列已经是A0<A1<A2<...<An 即已经是完全有序的情况了，那么便没有必要再进行比较。**

③. 结合所做的改进,为冒泡排序写一段伪代码。

**/\* 直接写代码吧....**

**应用冒泡排序对序列E,X,A,M,P,L,E按照字母顺序排序**

**\*/**

**#include<iostream>**

**#include<cstdio>**

**using namespace std;**

**char A[]= {'E','X','A','M','P','L','E'};**

**int n=7;**

**void BubbleSort() {**

**int flag=0;**

**for(int i=0; i<=n-2; i++) {**

**for(int j=0; j<=n-2-i; j++) {**

**if(A[j+1]<A[j]) {**

**swap(A[j],A[j+1]);**

**flag=1;**

**}**

**}**

**if(!flag)**

**return;**

**}**

**}**

**int main() {**

**BubbleSort();**

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

**printf("%c ",A[i]);**

**return 0;**

**}**

1. 请证明改进的算法最差效率也是平方级的。

**最差情况还是一样的，相当于没有哪次排完后已经有序，没有提前结束排序。**

4. 以下是蛮力字符串匹配算法：

算法 BruteForceStringMatch(T[0..n-1],P[0..m-1])

//该算法实现了蛮力字符串匹配

//输入：一个n个字符的数组T[0..n-1],代表一段文本，一个m个字符的数组P[0..m-1],代表一个模式

//输出：如果查找成功，返回文本的第一个匹配子串中第一个字符的位置，否则返回-1.

for i←0 to n-m do

j←0

while j<m and P[j]=T[i+j] do

j←j+1

if j=m return i

return -1

**问题：**

1. 实现蛮力字符串匹配算法，输入实例：T=”abcaacbacacdfgcds”,P=”acb”,输出4。

**#include<iostream>**

**#include<cstdio>**

**#include<string.h>**

**using namespace std;**

**char T[]="abcaacbacacdfgcds",P[]="acb";**

**int BruteForceStringMatch(int n,int m){**

**int j=0;**

**for(int i=0;i<=n-m;i++)**

**{**

**j=0;**

**while(j<m&&P[j]==T[i+j]){**

**j=j+1;**

**}**

**if(j==m)**

**return i;**

**}**

**return -1;**

**}**

**int main()**

**{**

**int n=strlen(T);**

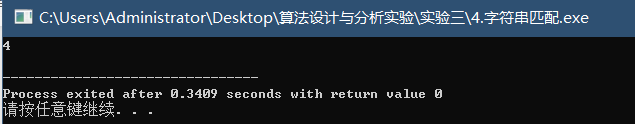
**int m=strlen(P);**

**//printf("%d %d",n,m);**

**printf("%d\n",BruteForceStringMatch(n,m));**

**return 0;**

**}**



②. 在算法中加入一个计数器，如果要在下面的文本中查找模式“GANDHI“。则蛮力算法将要执行的字符的比较次数是多少？

THERE IS MORE TO LIFE THEN INCREASEING ITS SPEED

假设查找前已知文本的长度，在这里它是47个字符。

**在目标串中并不能找到模式串，比较次数为43次。**

1. 修改算法，求出给定文本中匹配的子串的个数。例如：输入实例：T=”acdaacdacacdfgcds”,P=”acb”,输出3。

**这里题目应该写错了...P应该=“acd”。**

**#include<iostream>**

**#include<cstdio>**

**#include<string.h>**

**using namespace std;**

**int cnt=0;**

**char T[]="acdaacdacacdfgcds";**

**char P[]="acd";**

**int BruteForceStringMatch(int n,int m){**

**int j=0;**

**for(int i=0;i<=n-m;i++)**

**{**

**// cnt++;**

**j=0;**

**while(j<m&&P[j]==T[i+j]){**

**j=j+1;**

**}**

**if(j==m)**

**cnt++;**

**}**

**}**

**int main()**

**{**

**int n=strlen(T);**

**int m=strlen(P);**

**//printf("%d %d",n,m);**

**printf("%d\n",BruteForceStringMatch(n,m));**

**printf("%d\n",cnt);**

**return 0;**

**}**

1. 修改算法，使算法能够实现求在一段给定的文本中查找以A开始、以B结束的子串的数量(例如，在CABAAXBYA中，有4个这样的子串)。

**有4个这样的子串？？？ 不就两个嘛....QAQ**

**#include<iostream>**

**#include<cstdio>**

**#include<string.h>**

**using namespace std;**

**int cnt=0;**

**char T[]="CABAAXBYA";**

**int BruteForceStringMatch(int n){**

**bool flag=false;**

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

**if(T[i]=='A'&&!flag){**

**flag=true; //开始了**

**}**

**if(T[i]=='B'&&flag){**

**flag=false; //结束了**

**cnt++;**

**}**

**}**

**}**

**int main()**

**{**

**int n=strlen(T);**

**//printf("%d %d",n,m);**

**//printf("%d\n",BruteForceStringMatch(n));**

**BruteForceStringMatch(n);**

**printf("%d\n",cnt);**

**return 0;**

**}**

思考：在第③中，你能实现把子串”acb”替换成“xyz“吗？**不包含“xyz”**

5. 请对凸包问题进行建模，并写一个程序，实现凸包问题的蛮力算法。

**static void CalculateConvexHull(struct Point \*pArray,int length)**

**{**

**for(int x=0;x<length;x++) {**

**if(mark[x])**

**continue;**

**for(int y=0;y<length;y++) {**

**if(x==y || mark[y])**

**continue;**

**for(int z=0;z<length;z++) {**

**if(x==z || y==z || mark[z])**

**continue;**

**for(int i=0;i<length;i++) {**

**if(x==i || y==i || z==i || mark[i])**

**continue;**

**//在三角形内**

**if(PointIsInner(pArray[x],pArray[y],pArray[z],pArray[i]))**

**mark[i]=true;**

**}**

**}**

**}**

**}**

**}**

**/\* 打印凸包点 \*/**

**void PrintConvexHull(struct Point \*pArray,int &length)**

**{**

**using std::cout;**

**using std::endl;**

**CalculateConvexHull(pArray,length);**

**//凸包点缓冲区**

**Point \*tmp=new Point[length];**

**int j=0;**

**for(int i=0;i<length;i++) {**

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

**cout<<pArray[i].x<<","<<pArray[i].y<<endl;**

**tmp[j++]=pArray[i];**

**}**

**}**

**//更新**

**memcpy(pArray,tmp,sizeof(struct Point)\*j);**

**length=j;**

**}**

1. 应用穷举查找，把书中做了一个开头的分配问题的实例补充完整，并写出一个程序，实现分配问题的蛮力算法。请查阅资料了解匈牙利方法解决任务分配问题。

**既然是用蛮力算法，那么直接进行全排列即可，用C++的STL库即可轻松完成。**

**#include <cstdio>**

**#include <algorithm>**

**using namespace std;**

**#define n 4**

**#define max(a, b) (a) > (b) ? (a) : (b)**

**struct it {**

**int w, v;**

**};**

**it goods[n + 1];**

**int main() {**

**//input**

**int c[n][n] = { //代价矩阵, c[0][2]表示第一个人完成第三个任务所花费的代价**

**{9, 2, 7, 8},**

**{6, 4, 3, 7},**

**{5, 8, 1, 8},**

**{7, 6, 9, 4}**

**};**

**int a[n] = {0, 1, 2, 3};**

**int minv = 1 << 29;**

**do {**

**int people = 0;**

**int value = 0;**

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

**// printf("%d ", a[i]);**

**value += c[people++][a[i]];**

**}**

**// printf("%d\n", value);**

**minv = min(minv, value);**

**} while(next\_permutation(a, a + n)); //排列组合**

**//output**

**printf("%d", minv);**

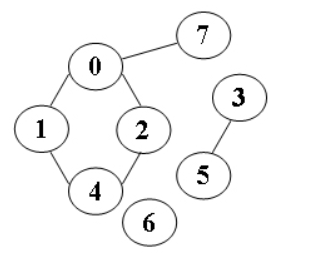
**return 0;**

**}**

7. 设计一个程序，对于给定的图，它能够输出：

a. 图的顶点，按照被DFS遍历第一次访问到的先后次序；

**直接在PTA题目图的基础上进行实现。**



**#include <stdio.h>**

**#define MVNum 100**

**typedef struct {**

**char vexs[MVNum]; //顶点阵**

**int arcs[MVNum][MVNum]; //边阵**

**int vexnum,arcnum; //顶点数 边数**

**} AMGraph;**

**int visited[MVNum];**

**void CreateUDN(AMGraph &G);//实现细节隐藏**

**void DFS(AMGraph G, int v) {**

**printf("%d ",G.vexs[v]);**

**visited[v] =1;**

**int w;**

**for(w = 0; w < G.vexnum; w++)**

**if((G.arcs[v][w]==1)&& (visited[w]==0))**

**DFS(G, w);**

**}**

**void DFSTraverse(AMGraph G) {**

**int v;**

**for(v = 0; v < G.vexnum; ++v)**

**visited[v] = 0;**

**for(v = 0; v < G.vexnum; ++v)**

**if(visited[v]==0)**

**DFS(G,v);**

**}**

**int main() {**

**AMGraph G;**

**CreateUDN(G);**

**DFSTraverse(G);**

**return 0;**

**}**

**void CreateUDN(AMGraph &G) {**

**scanf("%d %d",&G.vexnum,&G.arcnum);**

**for(int i=0; i<G.vexnum; i++)**

**//scanf("%c",&G.vexs[i]);**

**G.vexs[i]=i;**

**for(int i=0; i<G.vexnum; i++)**

**for(int j=0; j<G.vexnum; j++) {**

**G.arcs[i][j]=0;**

**}**

**for(int i=0; i<G.arcnum; i++) {**

**int u,v;**

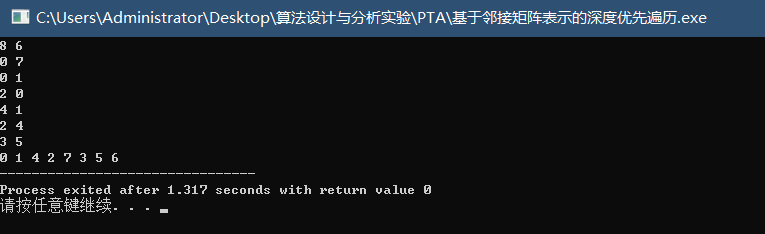
**scanf("%d %d",&u,&v);**

**G.arcs[u][v]=1;**

**G.arcs[v][u]=1;**

**}**

**}**



b. 图的顶点，按照被BFS遍历访问到的先后次序。

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MVNum 100**

**int visited[MVNum];**

**typedef struct {**

**char vexs[MVNum]; //顶点阵**

**int arcs[MVNum][MVNum]; //边阵**

**int vexnum,arcnum; //顶点数 边数**

**} Graph;**

**void CreateUDN(Graph &G);//实现细节隐藏**

**void BFS(Graph G, int v) {**

**char Q[MVNum];**

**int f=0,r=0;**

**int u,w;**

**printf("%d ",G.vexs[v]);**

**visited[v] = 1;**

**Q[r++]=v;**

**while(f<r) {**

**u=Q[f++];**

**for(w = 0; w < G.vexnum; w++) {**

**if(!visited[w]&&G.arcs[u][w]==1) {**

**printf("%d ",G.vexs[w]);**

**visited[w] = 1;**

**Q[r++]=w;**

**}**

**}**

**}**

**}**

**void CreateUDN(Graph &G) {**

**scanf("%d %d",&G.vexnum,&G.arcnum);**

**for(int i=0; i<G.vexnum; i++)**

**//scanf("%c",&G.vexs[i]);**

**G.vexs[i]=i;**

**for(int i=0; i<G.vexnum; i++)**

**for(int j=0; j<G.vexnum; j++) {**

**G.arcs[i][j]=0;**

**}**

**for(int i=0; i<G.arcnum; i++) {**

**int u,v;**

**scanf("%d %d",&u,&v);**

**G.arcs[u][v]=1;**

**G.arcs[v][u]=1;**

**}**

**}**

**void BFSTraverse(Graph G) {**

**int v;**

**for(v = 0; v < G.vexnum; ++v)**

**visited[v] = 0;**

**for(v = 0; v < G.vexnum; ++v)**

**if(!visited[v])**

**BFS(G, v);**

**}**

**int main() {**

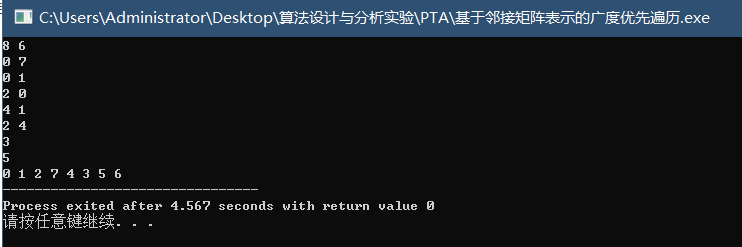
**Graph G;**

**CreateUDN(G);**

**BFSTraverse(G);**

**return** 0;

}



1. **注意事项**
2. 注意要保证枚举了每一种情况，不会有遗漏的情形发生，这是枚举算法正确性的保证。
3. 注意在枚举时要保证每种情形只枚举一次，尽量避免重复的计算，这样才能保证时间复杂度在可以接受的范围内。