**N-Queen’s Problem-**

1. #include<stdio.h>
2. #include<conio.h>
3. #include<math.h>
4. int a[30],count=0;
5. int place(int pos) {
6. int i;
7. for (i=1;i<pos;i++) {
8. if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
9. return 0;
10. }
11. return 1;
12. }
13. void print\_sol(int n) {
14. int i,j;
15. count++;
16. printf("\n\nSolution #%d:\n",count);
17. for (i=1;i<=n;i++) {
18. for (j=1;j<=n;j++) {
19. if(a[i]==j)
20. printf("Q\t"); else
21. printf("\*\t");
22. }
23. printf("\n");
24. }
25. }
26. void queen(int n) {
27. int k=1;
28. a[k]=0;
29. while(k!=0) {
30. a[k]=a[k]+1;
31. while((a[k]<=n)&&!place(k))
32. a[k]++;
33. if(a[k]<=n) {
34. if(k==n)
35. print\_sol(n); else {
36. k++;
37. a[k]=0;
38. }
39. } else
40. k--;
41. }
42. }
43. void main() {
44. int i,n;
45. clrscr();
46. printf("Enter the number of Queens\n");
47. scanf("%d",&n);
48. queen(n);
49. printf("\nTotal solutions=%d",count);
50. getch();
51. }

**LCS Problem-**

#include<stdio.h>

#include<string.h>

int i,j,m,n,c[20][20];

char x[20],y[20],b[20][20];

void print(int i,int j)

{

                if(i==0 || j==0)

                                return;

                if(b[i][j]=='c')

                {

                                print(i-1,j-1);

                                printf("%c",x[i-1]);

                }

                else if(b[i][j]=='u')

                                print(i-1,j);

                else

                                print(i,j-1);

}

void lcs()

{

                m=strlen(x);

                n=strlen(y);

                for(i=0;i<=m;i++)

                                c[i][0]=0;

                for(i=0;i<=n;i++)

                                c[0][i]=0;

                //c, u and l denotes cross, upward and downward directions respectively

                for(i=1;i<=m;i++)

                                for(j=1;j<=n;j++)

                                {

                                                if(x[i-1]==y[j-1])

                                                {

                                                                c[i][j]=c[i-1][j-1]+1;

                                                                b[i][j]='c';

                                                }

                                                else if(c[i-1][j]>=c[i][j-1])

                                                {

                                                                c[i][j]=c[i-1][j];

                                                                b[i][j]='u';

                                                }

                                                else

                                                {

                                                                c[i][j]=c[i][j-1];

                                                                b[i][j]='l';

                                                }

                                }

}

int main()

{

                printf("Enter 1st sequence:");

                scanf("%s",x);

                printf("Enter 2nd sequence:");

                scanf("%s",y);

                printf("\nThe Longest Common Subsequence is ");

                lcs();

                print(m,n);

return 0;

}

**MCM Problem**

|  |
| --- |
| #include<stdio.h>  #include<limits.h>    // Matrix Ai has dimension p[i-1] x p[i] for i = 1..n    int MatrixChainMultiplication(int p[], int n)  {      int m[n][n];      int i, j, k, L, q;        for (i=1; i<n; i++)          m[i][i] = 0;    //number of multiplications are 0(zero) when there is only one matrix        //Here L is chain length. It varies from length 2 to length n.      for (L=2; L<n; L++)      {          for (i=1; i<n-L+1; i++)          {              j = i+L-1;              m[i][j] = INT\_MAX;  //assigning to maximum value                for (k=i; k<=j-1; k++)              {                  q = m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j];                  if (q < m[i][j])                  {                      m[i][j] = q;    //if number of multiplications found less that number will be updated.                  }              }          }      }        return m[1][n-1];   //returning the final answer which is M[1][n]    }    int main()  {      int n,i;      printf("Enter number of matrices\n");      scanf("%d",&n);        n++;        int arr[n];        printf("Enter dimensions \n");        for(i=0;i<n;i++)      {          printf("Enter d%d :: ",i);          scanf("%d",&arr[i]);      }        int size = sizeof(arr)/sizeof(arr[0]);        printf("Minimum number of multiplications is %d ", MatrixChainMultiplication(arr, size));        return 0;  } |

**Output**

*Enter number of matrices*  
*4*  
*Enter dimensions*  
*Enter d0 :: 10*  
*Enter d1 :: 100*  
*Enter d2 :: 20*  
*Enter d3 :: 5*  
*Enter d4 :: 80*  
*Minimum number of multiplications is 19000*

**Naïve String-Matching Algorithm-**

**#include<stdio.h>**

**#include<conio.h>**

**#include<string.h>**

**void main()**

**{**

**char str1[20],str2[20];**

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

**clrscr();**

**printf("\nEnter text string : ");**

**scanf("%s",&str1);**

**printf("Enter pattern to search : ");**

**scanf("%s",&str2);**

**m = strlen(str1);**

**n = strlen(str2);**

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

**{**

**for(j=0;j<n;j++)**

**{**

**if(str1[i+j] != str2[j])**

**break;**

**}**

**if(j == n)**

**{**

**printf("\n\n Pattern found at %d. ",i);**

**}**

**}**

**getch();**

**}**

**Rabin-Karp Algorithm**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| /\* Following program is a C implementation of Rabin Karp  Algorithm given in the CLRS book \*/  #include<stdio.h>  #include<string.h>    // d is the number of characters in the input alphabet  #define d 256    /\* pat -> pattern      txt -> text      q -> A prime number  \*/  void search(char pat[], char txt[], int q)  {      int M = strlen(pat);      int N = strlen(txt);      int i, j;      int p = 0; // hash value for pattern      int t = 0; // hash value for txt      int h = 1;        // The value of h would be "pow(d, M-1)%q"      for (i = 0; i < M-1; i++)          h = (h\*d)%q;        // Calculate the hash value of pattern and first      // window of text      for (i = 0; i < M; i++)      {          p = (d\*p + pat[i])%q;          t = (d\*t + txt[i])%q;      }        // Slide the pattern over text one by one      for (i = 0; i <= N - M; i++)      {            // Check the hash values of current window of text          // and pattern. If the hash values match then only          // check for characters on by one          if ( p == t )          {              /\* Check for characters one by one \*/              for (j = 0; j < M; j++)              {                  if (txt[i+j] != pat[j])                      break;              }                // if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]              if (j == M)                  printf("Pattern found at index %d \n", i);          }            // Calculate hash value for next window of text: Remove          // leading digit, add trailing digit          if ( i < N-M )          {              t = (d\*(t - txt[i]\*h) + txt[i+M])%q;                // We might get negative value of t, converting it              // to positive              if (t < 0)              t = (t + q);          }      }  }    /\* Driver program to test above function \*/  int main()  {      char txt[] = "GEEKS FOR GEEKS";      char pat[] = "GEEK";      int q = 101; // A prime number      search(pat, txt, q);      return 0;  } |

**Output:**

Pattern found at index 0

Pattern found at index 10