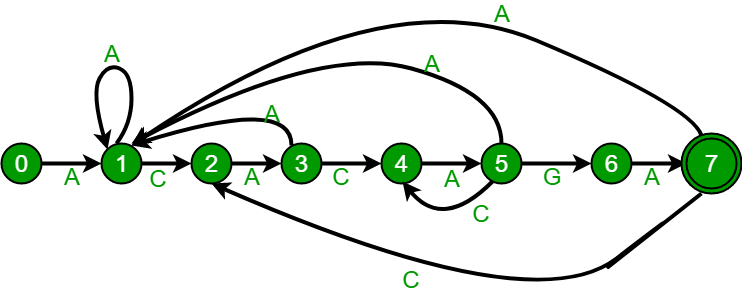
**Deterministic Finite Automata**



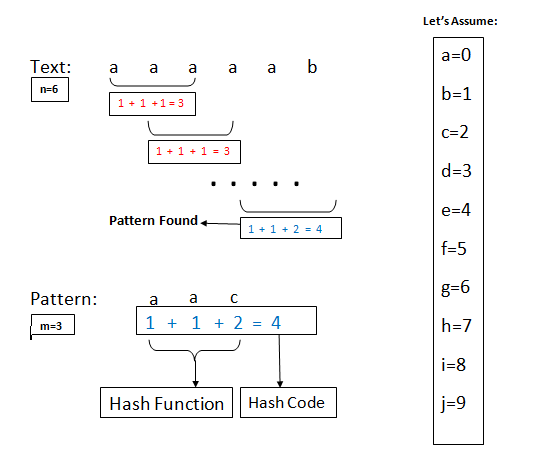
|  |
| --- |
| #include<stdio.h>  #include<string.h>  char pattern[20];  char text[200];  int sufpref(int s,char sc);  int DFA(char patttern[],char text[]);  int step=0,count=0;  int main()  {  int pa,te;  printf("\nEnter the pattern(Without space): ");  scanf("%s",&pattern);  printf("%s\n",pattern);  printf("\nEnter the Text(Without space): ");  scanf("%s",&text);  printf("%s\n",text);  int x = DFA(pattern,text);  printf("%d\n",x);  }  int DFA(char patttern[],char text[])  {  int n=strlen(pattern)-1,i,m = strlen(text)-1;  char c;  for(i = 0;i<m;i++){  c = text[i];  if(pattern[step+1]==c)  {  step++;  if(step == n)  {  count++;  step=0;  }  }  else  {  step = sufpref(step,c);  if(step == n)  {  count++;  step=0;  }  }  }  return count;  }  //creating highest number of suffuxes and prefixes which are same and returning the result.  int sufpref(int s,char sc)  {  char temp[s+1];  int i,x=s+1,flag=0,step;  for(i=0;i<s;i++)  {  temp[i]=pattern[i+1];  }  temp[s]=sc;  for(int j=1;j<x;j++)  {  char s[x-1],p[x-1];  for(i=0;i<x-j;i++)  {  s[i]=temp[i];  p[i]=temp[i+j];  }  if(!strcmp(s,p)){  flag++;  step=x-j;  break;  }  }  if(flag==0){  return 0;  }  else  {  return step;  }  } |

**Knuth Moris Pratt Algorithm(KMP)**

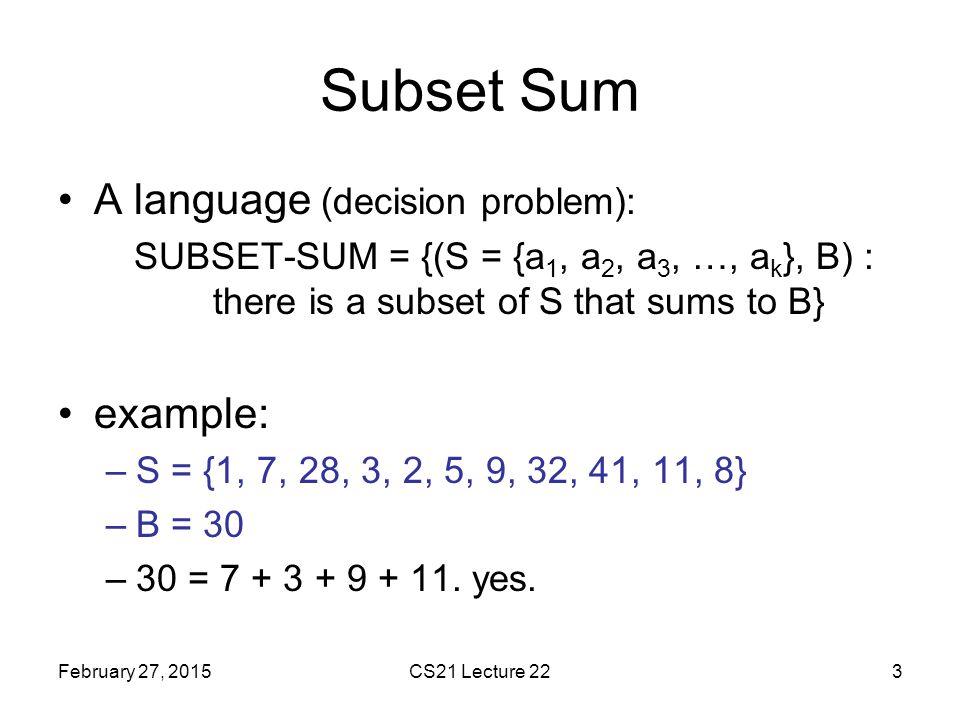
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** |
| **pattern** | **A** | **C** | **A** | **C** | **A** | **B** | **A** | **C** | **A** | **C** | **A** | **B** | **A** | **C** | **A** | **C** | **A** | **C** |
| **Overlap** | **0** | **0** | **1** | **2** | **3** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **4** |

|  |
| --- |
| #include<stdio.h>  #include<string.h>  void sufpTable(char pat[]);  int KMP();  int ind,flag=0,textln,s=0,t=0;  char pat[200];  int sufP[200];  char text[500];  int main()  {  printf("Please Enter the pattern:" );  scanf("%s",&pat);  //creating the array of substring  sufpTable(pat);  printf("Please Enter your text:\n");  scanf("%s",&text);  //maching pattern with text using kmp  int match = KMP();  printf("Pattern found %d time(s)\n",match);  }  int KMP()  {  int count =0;  int patlen = strlen(pat);  textln=strlen(text);  while(t<textln)  {  if(s<patlen-1)  {  if(text[t]==pat[s])  {  t++;  s++;  }  else if(s == 0 && text[t]!=pat[s])  {  t++;  }  else if(s != 0 && text[t]!=pat[s])  {  s = sufP[s-1];  }  }  else if( s == patlen-1)  {  if(pat[s]==text[t])  {  count++;  s = sufP[s];  t++;  }  else  {  s = sufP[s-1];  }  }  }  return count;  }  void sufpTable(char pat[])  {  ind = strlen(pat);  int i=1,j=0;  sufP[0]=0;  for(int i=0;i<ind;i++)  {  printf("%c ",pat[i] );  }  while(i!=ind)  {  if(pat[j]==pat[i])  {  sufP[i]=j+1;  j++;  i++;  }  else  {  if(j!=0)  {  j=sufP[j-1];  }  else  {  sufP[i]=0;  i++;  }  }  }  printf("\n");  return;  } |

**Rabin Karp**



|  |
| --- |
| #include<stdio.h>  #include<string.h>  #define d 256  void Rabin(char pat[], char txt[], int q)  {  int patternLen = strlen(pat);  int txtLen = strlen(txt);  int i, j;  int pHash = 0;  int tHash = 0;  int h = 1;  for (i = 0; i < patternLen-1; i++)  {  h = (h\*d)%q;  }  // Calculate the hash value of pattern and first window of text  for (i = 0; i < patternLen; i++)  {  pHash = (d\*pHash + pat[i])%q;  tHash = (d\*tHash + txt[i])%q;  }  // Sliding the pattern over text one by one  for (i = 0; i <= txtLen - patternLen; 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 ( pHash == tHash )  {  /\* Check for characters one by one \*/  for (j = 0; j < patternLen; 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 == patternLen)  {  printf("Pattern found at index %d \n", i);  }  }  /\*Calculate hash value for next window of text\*/  if ( i < txtLen- patternLen )  {  tHash = (d\*(tHash - txt[i]\*h) + txt[i+patternLen])%q;  if (tHash < 0)  {  tHash = (tHash + q);  }  }  }  }  int main()  { char txt[80],pat[80];  int q;  printf("Enter text \n");  scanf("%s",txt);  printf("Enter a pattern \n");  scanf("%s",&pat);  printf("Enter a prime number \n");  scanf("%d",&q);  Rabin(pat, txt, q);  return 0;  } |

**Subset sum**

**Subset sum using dynamic programming:**

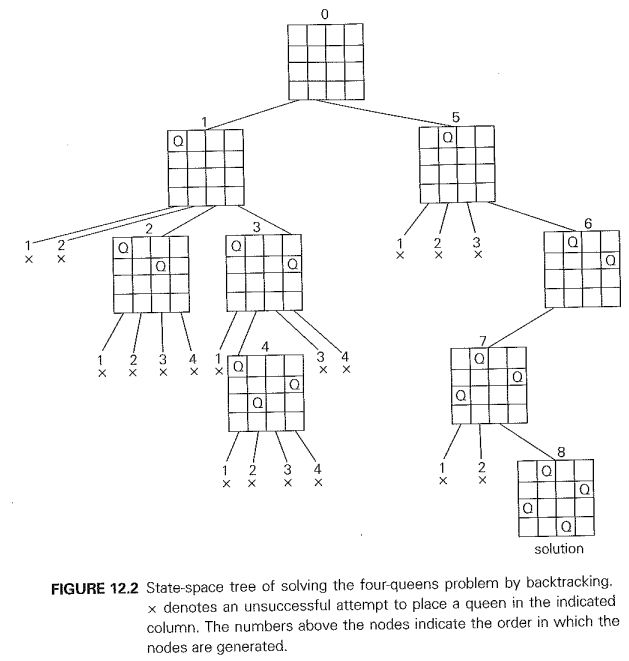
|  |
| --- |
| //subset sum using dynamic programming  #include<stdio.h>  int main()  {  int set[20],i,j,x,value,n,sum,t,z,subset[20];  scanf("%d",&t);  for(z=0;z<t;z++)  {  printf("Enter number of elements in set: ");  scanf("%d",&n);//number of element in set  printf("Enter the elements of the set:\n");  for(i=0;i<n;i++)  {  scanf("%d",&set[i]);  }  printf("Enter the sum to search: ");  scanf("%d",&sum);//total sum  int table[n+5][sum+5];  //creating the table.  //making the first column true  for(int s = 0;s < n;s++)  {  table[s][0]=1;  }  for(i=0;i < n;i++)  {    value = set[i];  for(j=1;j<=sum;j++)  {  if(i==0)  {  if(j == value)  {  table[i][j] = 1;  }  else  {  table[i][j] = 0;  }  }  else  {  if(value > j)  {  table[i][j] = table[i-1][j];  }  else  {  if(table[i-1][j]==1)  {  table[i][j] = 1;  }  else  {  table[i][j] = table[i-1][j-value];  }  }  }  }  }  j=sum;  i=n-1;  x = 0;  while( i!=0 || j!=0)  {  if(table[i][j] & table[i-1][j])  {  i=i-1;  }  else  {  j=j-set[i];  subset[x]=set[i];  x++;  }  }  if(sum == 0)  {  printf("Yes(0)\n");  }  else  { printf("Yes (");  for(i=0;i<x;i++)  {  printf("%d ",subset[i]);  }  printf(")\n");  }  }  } |

**Meet in the middle**

Meet in the middle is a search technique which is used when the input is small but not as small that brute force can be used. Like divide and conquer it splits the problem into two, solves them individually and then merge them.

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  typedef long long int LongNum;//Defined a custum type  LongNum FirstHalf[1000000],SecondHalf[1000000];  void subSetGenerate(LongNum set[],LongNum subSet[], int end,int start)  {  for(int i=0;i<(1<<end);i++)  {  LongNum temp = 0;  for(int j=0;j<end;j++)  {  if(i & (1<<j))  {  temp+= set[j+start];  }  subSet[i] = temp;  }  }  }  LongNum mergeAndSolve(LongNum set[],int size,LongNum value)  {  subSetGenerate(set,FirstHalf,size/2,0);  subSetGenerate(set,SecondHalf,size-size/3,size/2);  int FirstHalf\_size = 1<<(size/2);  int SecondHalf\_size = 1<<(size-size/2);  sort(SecondHalf,SecondHalf+SecondHalf\_size);  LongNum max = 0;  for(int i = 0;i<FirstHalf\_size;i++)  {  if(FirstHalf[i]<=value)  {  int pointer = lower\_bound(SecondHalf,SecondHalf+SecondHalf\_size,(value-FirstHalf[i]))-SecondHalf;  if( pointer == SecondHalf\_size || SecondHalf[pointer]!=(value-FirstHalf[i]))  {  pointer--;  }  if((SecondHalf[pointer]+FirstHalf[i])>max)  {  max = SecondHalf[pointer]+FirstHalf[i];  }  }  }  return max;  }  int main()  {  LongNum set[]={2,4,3,6,7,8};  int n = sizeof(set)/sizeof(set[0]);  LongNum value = 29;  LongNum sum = mergeAndSolve(set,n,value);  printf("Largest value smaller than (or equal) %lld is %lld.\n",value,sum);  } |

**N-Queens problem**

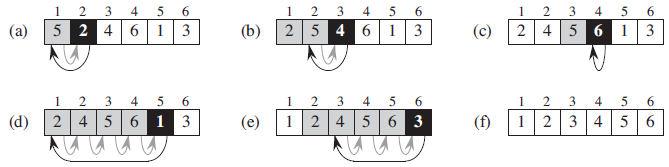


|  |
| --- |
| #include <iostream>  using namespace std;  void printTheGrid(int x);  bool validaionCheck(int column, int row, int n) ;  bool solve (int n, int row);  int matrix[10][10];  int main()  {  int queen;  cout<<"Enter the number of queen"<<endl;  cin >> queen;  for (int i = 0;i < queen;i++)  {  for (int j = 0;j < queen;j++)  {  matrix[i][j] = 0;  }  }    bool solution = solve(queen, 0);  if(solution == false)  {  cout << "no solutions" << endl;  }  else  {  cout << endl;  }  return 0;  }  bool solve (int n, int row)  {  if (n == row)  {  printTheGrid(n);  return true;  }  bool res = false;  for (int i = 0;i <=n-1;i++)  {  if (validaionCheck(i, row, n))  {  matrix[row][i] = 1;  res = solve(n, row+1) || res;  matrix[row][i] = 0;  }  }  return res;  }  bool validaionCheck(int column, int row, int n)  {  //checking the column.  for (int i = 0; i < row; i++)  {  if (matrix[i][column])  {  return false;  }  }  //checking the diagonas.(have to check the upper diagonals for validation).  for (int i = row,j = column;i >= 0 && j >= 0; i--,j--)  {  if (matrix[i][j])  {  return false;  }  }  for (int i = row, j = column; i >= 0 && j < n; j++, i--)  {  if (matrix[i][j])  {  return false;  }  }  return true;  }  void printTheGrid(int x)  {  for (int i = 0;i <= x-1; i++)  {  for (int j = 0;j <= x-1; j++)  {  cout <<matrix[i][j]<< " ";    }  cout<<endl;  }  cout<<endl;  cout<<endl;  } |

**Sub-Grid problem**

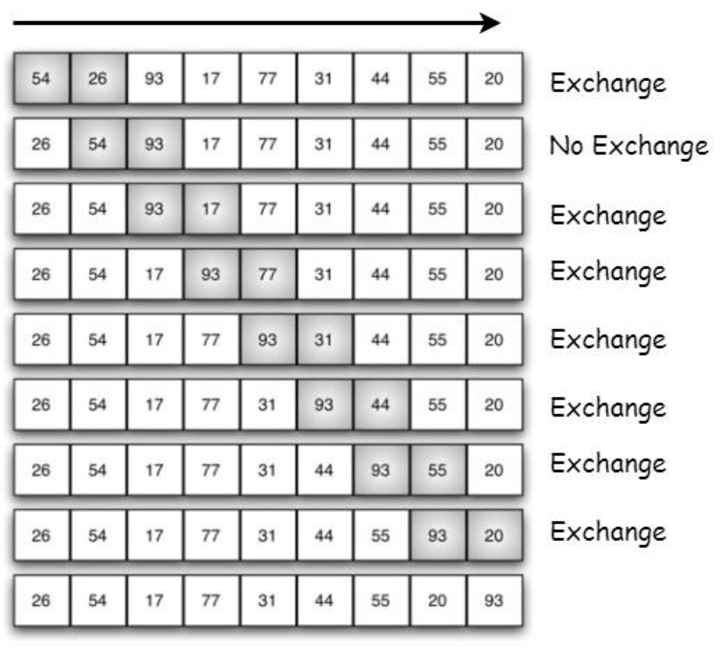
|  |  |  |  |
| --- | --- | --- | --- |
|  | **X** | **X** |  |
| **X** | **X** | **X** |  |
|  | **X** | **X** | **X** |
|  |  |  | **X** |

|  |
| --- |
| #include <bits/stdc++.h>  #include <cmath>  #include <string.h>  using namespace std;  int main()  {  int m, n, ans=0;  cin >> n >> m;  string s[n];  int c=0;  for(int i=0; i<n; i++)  cin >> s[i];  for(int k=0; k<n-1; k++){  for(int i=k+1; i<n; i++){  for(int j=0; j<m; j++){  if((s[k][j]=='x') && (s[i][j]=='x'))  c++;  }  ans+=(c\*(c-1))/2;  c=0;  }  }  cout << ans << endl;  return 0;  } |

**Insertion sort**

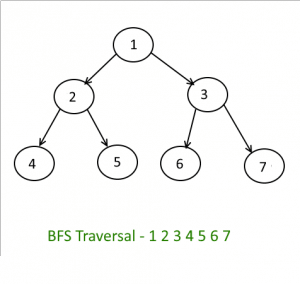
|  |
| --- |
| #include<stdio.h>  void insertion(int ar[],int n);  int main()  {  int ar[20],i,j,n;  printf("Enter the number of element for sorting: ");  scanf("%d",&n);  printf("\n Enter the Numbers: \n");  for(i=0;i<n;i++)  {  scanf("%d",&ar[i]);  }  //sorting:  insertion(ar,n);  printf("\n");  for(i=0;i<n;i++)  {  printf("%d ",ar[i]);  }  }  void insertion(int ar[],int n)  {  int i,j,k,temp,temp2;  for(i=1;i<n;i++)  {  temp=ar[i];  j=i;  while(temp<=ar[j-1])  {  ar[j]=ar[j-1];  j--;  }  ar[j]=temp;  }  } |

**Bubble Sort**

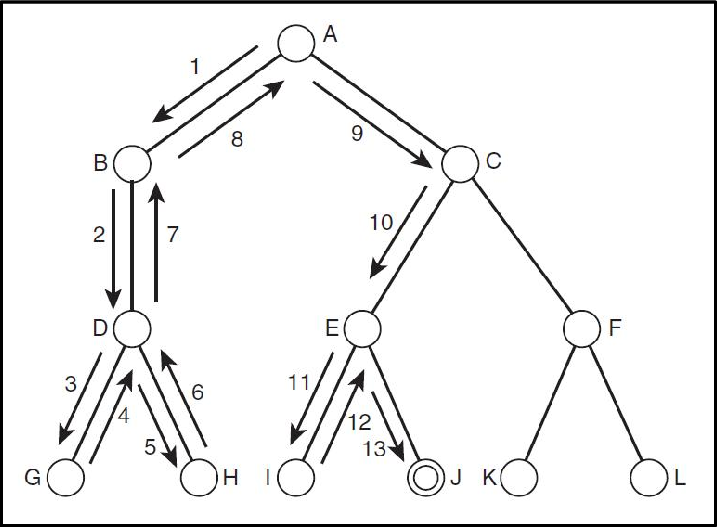


|  |
| --- |
| #include <stdio.h>  void bubbleSort(int array[], int size);  void printArray(int array[], int size);  int main()  {  int ar[20],i,j,n;  printf("Enter the number of element for sorting: ");  scanf("%d",&n);  printf("\n Enter the Numbers: \n");  for(i=0;i<n;i++)  {  scanf("%d",&ar[i]);  }  //sorting:  bubbleSort(ar,n);  printArray(ar, n);  }  void bubbleSort(int array[], int size)  {  for (int step = 0; step < size - 1; ++step)  {  for (int i = 0; i < size - step - 1; ++i)  {  if (array[i] > array[i + 1])  {  int temp = array[i];  array[i] = array[i + 1];  array[i + 1] = temp;  }  }  }  }  void printArray(int array[], int size)  {  for (int i = 0; i < size; ++i)  {  printf("%d ", array[i]);  }  printf("\n");  } |

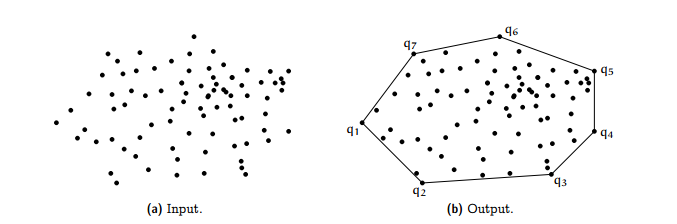
**BFS**



|  |
| --- |
| #include<bits/stdc++.h>  #include <vector>  using namespace std;  void BFS(int start);  void process(int node);  std::vector<int> graph[1000];  bool visit[1000];  queue<int> q;  int node;  int main()  {  int ver,x,y,starting;//x and y are adjecent nodes connected by a vertecis.  cin>>node;//entering number of node;  if(node == 0)  {  cout<<"There is no graph";  return 0;  }  cin >> ver;//entering number of vertecis  for(int i =0 ;i <ver;i++)  {  cin >> x >> y;  graph[x].push\_back(y);  graph[y].push\_back(x);  }  cin >> starting;//entering the starting node for dfs;  BFS(starting);  }  void BFS(int s)  {  visit[s]=true;  q.push(s);  while(!q.empty())  {  int node = q.front();  q.pop();  process(node);  for(auto u:graph[node])  {  if(visit[u])  {  continue;  }  visit[u]=true;  q.push(u);  }  }  }  void process(int node)  {  cout<< node << endl;  } |

**DFS**

|  |
| --- |
| #include<bits/stdc++.h>  #include <vector>  using namespace std;  void DFS(int start);  void process(int node);  std::vector<int> graph[1000];  bool visit[1000];  int node;  int main()  {  int ver,x,y,starting;//x and y are adjecent nodes connected by a vertecis.  cin>>node;//entering number of node;  if(node == 0)  {  cout<<"There is no graph";  return 0;  }  cin >> ver;//entering number of vertecis  for(int i =0 ;i <ver;i++)  {  cin >> x >> y;  graph[x].push\_back(y);  graph[y].push\_back(x);  }  cin >> starting;//entering the starting node for dfs;  DFS(starting);  }  void DFS(int s)  {    //cout << visit[s];  if(visit[s])  {  return;  }  visit[s] = true;  process(s);  for(auto x: graph[s])  {  DFS(x);  }  }  void process(int node)  {  cout<< node << endl;  }{  cout<< node << endl;  } |

**Convex hull**

|  |
| --- |
| //Light oj 1203 using convex hull jarvis march algorithm  #include <bits/stdc++.h>  #include <cmath>  using namespace std;  struct Point  {  long long int x, y;  };  long double angle(long long int a1,long long int b1,long long int a2,long long int b2,long long int a3,long long int b3)  {  long double A\_dot\_B,modA,modB,temp1,temp2,ang;  A\_dot\_B = ((a1-a2)\*(a3-a2))+((b1-b2)\*(b3-b2));  modA = sqrt((pow((a1-a2),2)+pow((b1-b2),2)));  modB = sqrt((pow((a3-a2),2)+pow((b3-b2),2)));  temp1 = (A\_dot\_B\*1.0) / (modA \* modB);  if(modA == 0 || modB == 0)  {  ang = 0;  }  else  {  temp2 = acos(temp1);  ang = ((180/3.141592653589793 )\*temp2);  }  return ang;  }  int orientation(Point a, Point b, Point c)  {  long long int slope = (b.y - a.y) \* (c.x - b.x) - (b.x - a.x) \* (c.y - b.y);  if (slope == 0)  {  return 0; // colinear  }  else if(slope > 0)  {  return 1;//clockwise  }  else  {  return 2;//Counterclockwise  }  }  long double convexHull(Point points[], long long int n)  {  long double ag;  long long int flag = -1;  if(n<3)  {  return 0;  }  else{  vector<Point> cv;  //find left most;  long long int left = 0;  for(int i = 0;i< n;i++)  {  if (points[i].x < points[left].x)  {  left = i;  }  }  long long int p = left,q;//starting from left.  do  {  cv.push\_back(points[p]);  q = (p+1)%n; //taking a random point at first  //Finding orientation  for (int i = 0; i < n; i++)  {  if (orientation(points[p], points[i], points[q]) == 2)  q = i;  }  p = q;  }while (p != left);  for (int i = 0; i < cv.size(); i++)  cout << "(" << cv[i].x << ", "  << cv[i].y << ")\n";  long double temp = 360.00;  for (int i = 0; i < cv.size(); i++)  {  if(i == (cv.size()-2) )  {  ag = angle(cv[i].x,cv[i].y,cv[i+1].x,cv[i+1].y,cv[0].x,cv[0].y);  }  else if(i == (cv.size()-1))  {  ag = angle(cv[i].x,cv[i].y,cv[0].x,cv[0].y,cv[1].x,cv[1].y);  }  else  {  ag = angle(cv[i].x,cv[i].y,cv[i+1].x,cv[i+1].y,cv[i+2].x,cv[i+2].y);  }  if(ag <= temp)  {  temp = ag;  flag = i;  }  }  return temp;  }  }  int main()  {  ios\_base::sync\_with\_stdio(false);  cin.tie(NULL);  int T;  cin >> T;  for(int s = 0;s<T;s++)  {  int np;  cin >> np;  if(np<3)  {  cout<<"Case "<< s+1<<": "<< "0"<<endl;  }  else  {  Point points[np];  for(int i = 0;i<np;i++)  {  cin>>points[i].x >>points[i].y;  }  long long int n = sizeof(points)/sizeof(points[0]);  cout <<"Case "<<s+1<<": "<< fixed << setprecision(6) << convexHull(points, n) <<endl;  }  }  return 0;  } |

**Binary indexed tree**

|  |
| --- |
| #include <iostream>  using namespace std;  void createTree(int Tree[],int n);  void printTree(int Tree[],int n);  void updateTree(int Tree[], int n, int value, int index);  int Sum(int Tree[], int index);  int array[1000];  int Tree[1000];  int main()  {  int n;  printf("Enter the size of array: ");  scanf("%d",&n);    printf("Enter the element of the array:\n");  Tree[0]=0;  for(int i=0;i<n;i++)  {  scanf("%d",&array[i]);  Tree[i+1]=0;//initializing the tree assigning 0 to every index;  }  createTree(array,n);  cout << "Sum of 0-5: "  << Sum(Tree, 5);  }  void createTree(int array[],int n)  {  for(int i = 0;i<n;i++)  {  updateTree(Tree,n,array[i],i);  }  printTree(Tree,n);  }  void printTree(int Tree[],int n)  {  for(int i=1;i<=n;i++)  {  cout << Tree[i]<<" ";  }  cout << "\n" << endl;  }  void updateTree(int Tree[], int n, int value, int index)  {  index = index + 1;  while (index <= n)  {  Tree[index] += value;  index += index & (-index);  }  }  int Sum(int Tree[], int index)  {  int sum = 0;  index = index + 1;  while (index>0)  {  sum += Tree[index];  index -= index & (-index);  }  return sum;  } |

**Segment tree**

|  |
| --- |
| #include <iostream>  #include<math.h>  using namespace std;  int min(int a, int b);  void printTree(int n);  void constructTree(int input[],int segmentT[],int low,int high, int pos);  int querry(int segmentT[],int qLow,int qHigh, int low, int high, int pos);  int input[]= {-1,0,3,6},segmentT[10000],infinity = 999999;  int main()  {  int n,lenSeg,a;  float b;  a = sqrt(sizeof(input)/sizeof(input[0]));  b = sqrt(sizeof(input)/sizeof(input[0]));  if(b-a == 0)  {  lenSeg = pow(2,a)\*2 -1;  }  else  {  lenSeg = pow(2,a+1)\*2 -1;  }  for(int i = 0; i < lenSeg;i++)  {  segmentT[i]= infinity;  }  constructTree(input,segmentT,0,3,0);  printTree(lenSeg);  cout << endl;  cout << "minimum of 1-3: " << querry(segmentT,1,3,0,3,0)<< endl;  }  void constructTree(int input[],int segmentT[],int low,int high, int pos)  {  if(low == high)  {  segmentT[pos] = input[low];  return;  }  int mid = (low + high) / 2 ;  constructTree(input,segmentT,low,mid, 2\*pos+1);  constructTree(input, segmentT, mid+1, high, 2\*pos+2);  segmentT[pos] = min(segmentT[2\*pos+1], segmentT[2\*pos+2]);  }  int querry(int segmentT[],int qLow,int qHigh, int low, int high, int pos)  {  if(qLow <= low && qHigh >= high)//total overlap  {  return segmentT[pos];  }  if(qLow > high || qHigh < low)//no overlap  {  return infinity;  }  int mid = (low+high)/2;  int temp = min(querry(segmentT,qLow,qHigh,low,mid,2\*pos+1),querry(segmentT,qLow,qHigh,mid+1,high,2\*pos+2));  return temp;  }  void printTree(int n)  {  for(int i = 0;i<n;i++)  {  cout << segmentT[i] << " ";  }  cout<<endl;  }  int min(int a, int b)  {  if(a>b)  return b;  else  return a;  } |