* **Implementation of Single linkage algorithm**

#include <bits/stdc++.h>

**using** **namespace** std;

**int** main()

{

**int** n, i,j,flagi,flagj,k,run;

**double** solution[100][100],x[100],y[100],t1,t2,min,temp[100][100];

**while**(cin>>n)

{

min=9999;

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

cin>>x[i]>>y[i];

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

{

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

{

t1=sqrt(pow((x[j]-x[i]),2)+pow((y[j]-y[i]),2));

solution[i][j]=t1;

temp[i][j]=t1;

**if**(t1<min && t1!=0)

{

min=t1;

flagi=i;

flagj=j;

}

}

}

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

{

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

printf("%.1lf ",solution[i][j]);

cout<<"\n";

}

run=n-2;

**while**(run)

{

min=9999999.0;

n--;

cout<<endl<<flagi<<" "<<flagj<<" will merge.\n";

**for**(i=flagi; i<=flagi; i++)

{

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

{

**if**(i==j)

solution[i][j]=0;

**else**

{

**if**(j>flagi)

{

*///the value of right side will merge*

**if**(temp[i][j+1]<temp[i+1][j+1])

solution[i][j]=temp[i][j+1];

**else**

solution[i][j]=temp[i+1][j+1];

}

**else** **if**(j<flagi)

{

*///the value of left side will merge*

**if**(temp[i][j]<temp[i+1][j])

solution[i][j]=temp[i][j];

**else**

solution[i][j]=temp[i+1][j];

}

}

}

} *///End of merge*

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

{

**if**(i==flagi)*///because this row is calculated during merging*

**continue**;

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

{

**if**(i==j)

solution[i][j]=0;

**else** **if**(j==flagi) *///to retrive the data from merged row*

solution[i][j]=solution[j][i];

**else** **if**(j>=flagj) *///when merging occurs at the up from current position*

solution[i][j]=temp[i+1][j+1];

**else** **if**(j<flagj) *///when merging occurs at the bellow from current position*

solution[i][j]=temp[i][j];

}

}

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

{

cout<<"\n";

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

{

temp[i][j]=solution[i][j];

**if**(solution[i][j]<min && solution[i][j]!=0)

{

min=solution[i][j];

flagi=i;

flagj=j;

}

printf("%.1lf ",solution[i][j]);

}

}

cout<<"\n\n";

run--;

}

}

}

* **Implementation of K-means clustering algorithm**

#include<bits/stdc++.h>

**using** **namespace** std;

**int** main()

{

**int** n,i,x,j,k,numberOfelement[20];

**double** data[200],cluster[200][200],mean[20],distance,tempmean[20];

**while**(cin>>n)

{

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

cin>>data[i];

cin>>k;

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

{

mean[i]=data[i];

tempmean[i]=data[i];*///keeping means in temporary array*

}

x=0;

*// sort(data,data+n);*

**while**(**true**)

{

**int** fcluster;

**double** min;

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

{

numberOfelement[i]=0;

}

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

{

min=999.9;

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

{

distance=fabs(mean[j]-data[i]);

**if**(distance<min)

{

min=distance;

fcluster=j;

}

}

cluster[fcluster][numberOfelement[fcluster]]=data[i];

numberOfelement[fcluster]++;

**double** sum=0;

**if**(numberOfelement[fcluster]>1)

{

**for**(**int** p=0; p<=numberOfelement[fcluster]; p++)

{

sum=sum+cluster[fcluster][p];

}

**int** temp=numberOfelement[fcluster];

mean[fcluster]=**double**(sum/(temp+1));

}

}

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

{

cout<<"Cluster "<<i+1<<endl;

**for**(j=0; j<numberOfelement[i]; j++)

{

cout<<cluster[i][j]<<" ";

}

cout<<endl<<endl;

}

cout<<"\n\n";

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

cout<<mean[i]<<" ";

cout<<"\n\n";

x++;

**int** run=0;

*///if no change in mean then break;*

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

{

**if**(tempmean[i]!=mean[i])*///Checking for changes*

run=1;

}

**if**(run==1)

{

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

tempmean[i]=mean[i];

}

**else**

**break**;

}

cout<<"\n\nit takes "<<x<<" passes\n";

}

**return** 0;

}

* **Finding complete linkage and single linkage distance between two clusters.**

#include <bits/stdc++.h>

**using** **namespace** std;

**int** main()

{

**int** i,j,k,n,fi1,fj1,fi2,fj2;

**double** x1[20],y1[20],x2[20],y2[20],solution[20][20],min,max;

cout<<"Enter the number of value: ";

cin>>n;

cout<<"Enter the value of cluster1:\n";

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

cin>>x1[i]>>y1[i];

cout<<"Enter the value of cluster2:\n";

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

cin>>x2[i]>>y2[i];

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

{

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

{

solution[i][j]=sqrt(pow(x2[j]-x1[i],2)+pow(y2[j]-y1[i],2));

**if**(i==0 && j==0)

min=solution[i][j],max=solution[i][j];

**if**(solution[i][j]<min)

{

min=solution[i][j];

fi1=i,fj1=j;

}

**else** **if**(solution[i][j]>max)

{

max=solution[i][j];

fi2=i,fj2=j;

}

}

}

cout<<"\nSolution :\n";

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

{

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

cout<<solution[i][j]<<" ";

cout<<endl;

}

cout<<"\nSingle linkage distance is between ( "<<x1[fi1]<<", "<<y1[fi1]<<" ) and ( "<<x2[fj1]<<", "<<y2[fj1]<<" ).\n";

cout<<"\ncomplete linkage distane is between ( "<<x1[fi2]<<", "<<y1[fi2]<<" ) and ( "<<x2[fj2]<<", "<<y2[fj2]<<" ).\n";

}