```
// Author : Dhananjay Kajla
// File Created : 29.08.2019 04:12:00
#include
#include // for time function
#include // for time manipulation
#include // for associative arrays
#include // to convert double to string
#include // to convert char to int to get mc fast
#include // to take input from files
#include
using namespace std;

/Variable Declaration**/

std::string infile,outfile;
time_t start,counter; //Counts execution time
double timer = 0.0; //Allowed time
int v_size = 0; //Vocabulary size
std::vector v; //Vocabulary
size_t K = 0; //Number of strings
std::vector X; //Set of initial strings
std::vector S; //Final solution strings
double cc = 0; //Conversion cost constant
std::vector > MC; //Mactching cost matrix

//

/Extra Container*/

std::map mp;

//
```

/Search Space/ class Node { public: std::vector coordinates; std::vector solution; double best_cost; Node(std::vector v, std::vector s, double bc); Node(std::vector v); bool Is_Goal(); bool Is_Valid(); std::vector,Node> > Expansion_Function(); }; /*/*

/Search Space Functions/

```
Node::Node(std::vector v, std::vector s, double bc)
{
coordinates = v;
```

```
solution = s;
best_cost = bc;
}

Node::Node(std::vector v)
{
coordinates = v;
solution.resize(K);
best_cost = -1;
}

bool Node::Is_Valid()
{
for (size_t i = 0; i < coordinates.size(); i++) { if(coordinates[i] > (int)X[i].size())
{
return false;
}
}
return true;
}

bool Node::Is_Goal()
{
for (size_t i = 0; i < coordinates.size(); i++)
{
if(coordinates[i] == (int)X[i].size())
{
return false;
}
}
best_cost = 0;
solution.resize(K);
return true;
}

std::vector,Node> > Node::Expansion_Function()
{
std::vector,Node> > v;
int z = 1 << (K+1); int i = 0; while(i < z) { i++; int z1 = i; std::vector va = coordinates;
std::vector vc(K,'-');
int counter = -1;
while(z1 > 0)
{
```

```
counter++;
if(z1 & 1)
{
va[counter]++;
vc[counter] = X[counter][coordinates[counter]];
}
z1 /= 2;
}
Node r(va);
if(r.Is_Valid())
{
v.push_back(std::make_pair(vc,r));
}
}
return v;
}
//
```

/*Search manipulation Functions**/

```
double Step_Cost(Node a, Node b)
{
std::vector v;
int dash_counter = 0;
for(size_t i=0; i< K ; i++) { int q = b.coordinates[i] - a.coordinates[i]; //std::cout << q<< std::endl; if(q
> 1)
{
return -1;
}
if(q == 1)
{
v.push_back(true);
}
else
{
v.push_back(false);
dash_counter++;
}
}
std::vector temp;
double cost = dash_counter * cc;
for (size_t i = 0; i < K ; i++)
```

```
{
if(v[i])
{
temp.push_back(X[i][a.coordinates[i]]);
}
else
{
temp.push_back('-');
}
}
for (size_t i = 0; i < temp.size(); i++)
{
for (size_t j = i+1; j < temp.size(); j++)
{
cost += MC[mp[temp[i]]][mp[temp[j]]];
}
}
return cost;
}

//

/*
Input Function
*/
void input()
{
ifstream inobj; //declaring inputstreaming object
inobj.open(infile.c_str()); //opening the given file
inobj >> timer; //input timer
timer *= 60; //converts minutes into seconds
inobj >> v_size; //input size of Vocabulary
v.resize(v_size); //resize the Vocabulary vector to the given size
std::string str; //temporary variable used to get Vocabulary
for (size_t i = 0; i < v.size(); i++) //getting Vocabulary { inobj >> str; //Each string has the required
character and a ','
v[i] = str[0]; //Removing the ','
}
for (size_t i = 0; i < v.size(); i++) { mp[v[i]] = i; //Mapping char to int to make access to MC eaesier }
mp['-'] = v.size(); //last column of the matrix is for '-' inobj >> K; //input the size of the initial strings
X.resize(K); //resizing the initial string vector to the given number of strings
S.resize(K); //resizing the final string vector to the given number of strings
```

```
for (size_t i = 0; i < X.size(); i++) //getting the initial strings { inobj >> X[i]; //taking in the strings
}
inobj >> cc; //input the conversion cost
for (size_t i = 0; i <= (size_t)v_size; i++) taking in the mc matrix { std::vector temp(v_size+1);
//temporary vector to get input
for(size_t j = 0; j <= (size_t)v_size; j++) getting the cost vector for ith character { inobj>> temp[j];
//getting the individual cost
}
MC.push_back(temp); //Adding a new row to MC
}
inobj >> str; //Taking the last input '#'
inobj.close(); //Closing the stream
return;
}

/*
Input_Test Function
/ /
void input_test()
{
ofstream onobj; //declaring outputstreaming object
onobj.open("input__t.txt"); //opening the test file
onobj << timer << std::endl; //output timer
onobj << v_size << std::endl; //output size of Vocabulary
for (size_t i = 0; i < v.size()-1; i++) //outputting Vocabulary
{
onobj << v[i] << ", "; //Each string has the required character and a ','
}
onobj << v[v.size()-1] << std::endl; //Formatting
onobj << K << std::endl; //output the size of the initial strings
for (size_t i = 0; i < X.size(); i++) //getting the initial strings
{
onobj << X[i] << std::endl; //outputting the strings
}
onobj << cc << std::endl; //output the conversion cost
for (size_t i = 0; i <= (size_t)v_size; i++) //outputting in the MC matrix
{
for(size_t j = 0; j <= (size_t)v_size; j++)
{
onobj << MC[i][j] << ' '; //outputting the cost for i, j
}
onobj << std::endl;
}
onobj << "#" << std::endl; //Taking out the last output '#'
onobj.close(); //Closing the stream
return;
}
*/

/*
Output Function
*/
void output()
{
ofstream outobj; //declaring inputstreaming object
outobj.open(outfile.c_str()); //opening the given file
for (size_t i = 0; i < K; i++)
{
outobj << S[i] << std::endl; //Outputting the solution
}
outobj.close(); //Closing the stream
return;
}

/*
Dynamic Programming
/ std::map, Node > mpp; void dp(Node q) { if(mpp[q.coordinates].best_cost != -1) {
return; } mpp[q.coordinates] = q; if(q.Is_Goal()) { return; } std::vector, Node > > v =
q.Expansion_Function(); double min_cost = -1; size_t id = -1; for (size_t i = 0; i < v.size();
i++) { dp(v[i].second); if(min_cost == -1) { min_cost = v[i].second.best_cost; id = i; } else {
if(v[i].second.best_cost < min_cost) { min_cost = v[i].second.best_cost; id = i; } } }
q.best_cost = min_cost; for (size_t i = 0; i < K; i++) { q.solution[i] = v[id].first[i] +
v[id].second.solution[i]; } return; } /
Algorithm Function
/ void algorithm() { /
std::vector a(2), b(2);
a[0] = 0;
a[1] = 0;
b[0] = 0;
b[1] = 1;
Node A(a);
Node B(b);
std::cout << Step_Cost(A,B) << std::endl;
*/
```

```
/Dynamic Programming/
std::vector v(K,0); //Temporary vector to initiate the start state
Node start(v); //Start state
dp(start);
/***/

}

/*
Main Function
/ int main(int argc, char argv[])
{

// infile = argv[1]; outfile = argv[2]; //

// input(); algorithm(); output(); //
}
```