**Shortest path using parallel processing and MPI**

**Problem**

**Parallel shortest path algorithm (graph algorithms) using matrix representation of the graph and matrix multiplication.**

**Report Content:**

**- Using matrix graph modelling.**

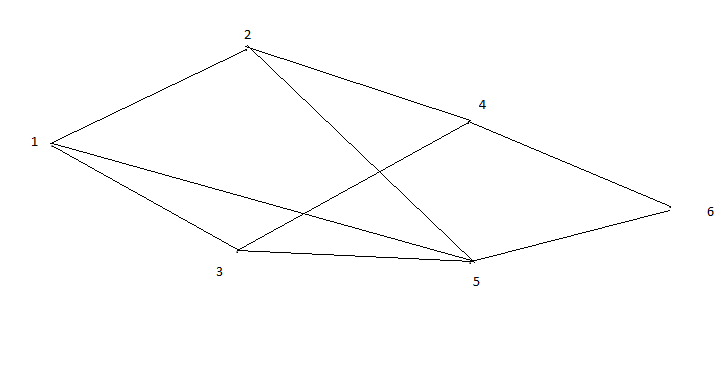
**- Pseudocode programming**

**- implement parallel processing in function**

**- Testing**

**- Conclusion**

**1.Using graph model**



I used this graph model, and I converted this to 6\*6 matrix below. And I read from file, and file name is input.txt.

**6**

**0 3 5 0 7 0**

**3 0 0 2 4 0**

**5 0 0 7 4 0**

**0 2 7 0 0 5**

**7 4 4 0 0 8**

**0 0 0 5 8 0**

**2. pseudocode programming code**

pseudocode function gets shortest path using parallel processing and pseudocode method.

this is pseudocode algorithm implement using recursion sub function. using graph matrix, implement pseudocode.

in first, using matrix, find possible path step by step, and increase step 1.

and then solve path number and possible direction.

this function has 3 subs.

- if key is over node, function will exit.

- if key is the same code, compare minimum cost and output possible paths.

- if key is less than node, implement pseudocode.

the code can be found below

int path\_find(int k,int length, string path) {

if(k>node-1) return 0;

if (k == node-1) {

cout<<path<<" weight: "<<length<<endl;

if (min > length) {

min=length;

short\_path=path;

}

}

if (k < node-1) {

#pragma omp parallel for

for (int i = 0;i < node;i++) {

if (mat[k][i] != 0) {

if (flag[i] == false) {

flag[i]=true;

char no[3];

\_itoa\_s(i+1,no,10);

string sno(no);

path\_find(i,length+mat[k][i],path+sno+" ");

flag[i]=false;

}

}

}

}

}

also this function can implement in another way pseudocode

int max\_sum=0,s=0;

for(int i=0;i<node;i++){

for(int j=0;j<node;j++){

if(i!=j && matrix[i][j]!=0) s+=matrix[i][j];

break;

}

if(max\_sum<s){

max\_sum=s;

matrix[i][j]=s+matrix[i][j];

}

else{

break;

goto1;

}

}

**3. implement parallel processing function call**

parallel processing can be implemented 2 ways:

1) mpi

2) omp

I used both methods, using <mpi.h> and <omp.h>.

the code can be found below:

int main(int argc, char \*argv[])

{

int comm\_sz;

int my\_rank;

int comm\_sz;

MPI\_Init(&argc,&argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &comm\_sz);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank);

fstream fp;

fp.open("input.txt",ios::in);

fp>>node;

mat=(int\*\*)malloc(sizeof(int\*)\*node);

flag=(bool\*)malloc(sizeof(bool)\*node);

for (int i = 0;i < node;i++) {

mat[i]=(int\*)malloc(sizeof(int)\*node);

}

for (int i = 0;i < node;i++) {

flag[i]=false;

for (int j = 0;j < node;j++) {

fp>>mat[i][j];

}

}

flag[0]=true;

path\_find(0,0,"1 ");

cout<<"----------------------------"<<endl;

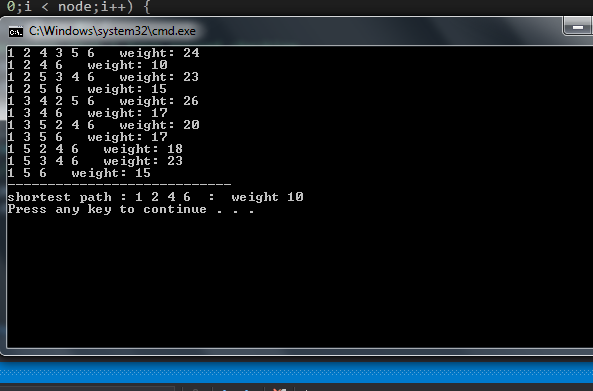
cout<<"shortest path : "<<short\_path<<" : weight "<<min<<endl;

return 0;

}

**4.Testing**

Using upper graph, and converted matrix, shortest path weight is 10 and possible paths are 11 ways using visual studio 2015, made program.



**5. conclusion**

I would like to say the mpi and omp are good parallel processing method and know well about graph data structure and pseudocode algorithm.