**Implement a Multi-threading application for echo server using socket programming in JAVA**.

**Mathematical Model:**

Let M is set of messages that to be sent by client and receive by server. The server again send same message as response to client.

M={m1,m2,……mn} where n is number of messages.

Let S be server program which is going to run on H machine and P port number.

Let φ be function that binds the server to a port P on machine H.

φ (S, H, P) = bind S to a port P on machine H.

Let C be set of clients.

C={c1,c2,c3, ……. ck} where k is number of clients.

Let fc and gc be function used for send and receiver message .

 i fc(ci, mi, S) = (ci X mi )  S

 i gc(ci, mi, S) = (S X mi )  ci

Let fs and gs be function used for send and receiver message .

 i fs(S, mi, ci) = (S X mi )  ci

 i gs(S, mi, ci) = (ci X mi )  S

**Implement n-ary search algo using OpenMP**

**Mathematical Model:**

A = { e1,e2,e3,..................en}

n : no of elements in the arrray

m: no of threads that run concurrently

T={t1,t2,t3,...........tm}

key: the element to search

The function is to allocate elements to the each thread

offset=chunk=g(n,m)= (n+m-1) /m

The function returns the id of the each thread

getid(ti)=id; It is in the range of o to m-1

Index of the array elements i={0,1,2,3,......n-1}

Retrieving the element of the array

φ(id,chunk,i)= a[id\*chunk+i]

The function for the comparing the element to be search and array element.

f(key, φ(id,chunk, i))= 1 if thread ti found key in the allocated area

0 otherwise not found

**Implement Concurrent Prim’s Algorithm using OPENMP**

**Mathematical Model:**

Let G = (V,E) be a weighted, connected graph.

**Lets V be set of vertices.** *V* = {1, 2,..., *n*} and E be set of edges.

Let T be the edge set that is grown in Prim's algorithm **. Assume that initially T is** empty set.

**T = Ф;**

Assume that the vertex 1 is source/starting vertex of Prims Algorithm

**U = { 1 };**

When the algorithm stops, U includes all vertices of the graph and hence T is a spanning tree

Use two arrays, **closest and lowcost.**

* For *i* Є*V* - *U*, closest[*i*] gives the vertex in *U* that is closest to *i*
* For *i* Є *V* - *U*, lowcost[*i*] gives the cost of the edge (*i*, closest(*i*))

Example:

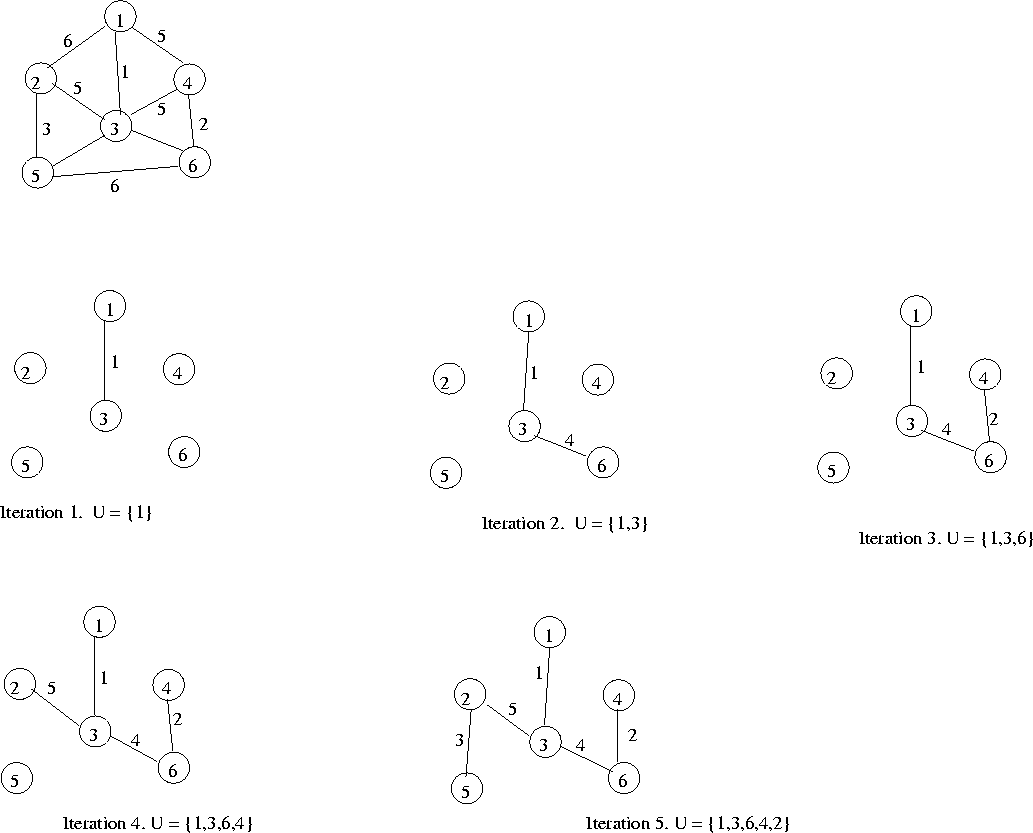
\fbox{Step 1}

|  |  |  |
| --- | --- | --- |
| *U* = {1} | | *V* - *U* = {2, 3, 4, 5, 6} |
| closest | | lowcost |
| *V* - *U* | *U* |  |
| 2 | 1 | 6 |
| 3 | 1 | 1 |
| 4 | 1 | 5 |
| 5 | 1 | $ \infty$ |
| 6 | 1 | $ \infty$ |

Select vertex 3 to include in ***U***

|  |  |  |
| --- | --- | --- |
| \fbox{Step 2} | | |
|  |  |  |
| *U* = {1, 3} | | *V* - *U* = {2, 4, 5, 6} |
| closest | | lowcost |
|  |  |  |
| *V* - *U* | *U* |  |
| 2 | 3 | 5 |
| 4 | 1 | 5 |
| 5 | 3 | 6 |
| 6 | 3 | 4 |
| Now select vertex 6 | | |

|  |  |  |
| --- | --- | --- |
| \fbox{Step3} | | |
| *U* = {1, 3, 6} | | *V* - *U* = {2, 4, 5, 6} |
| closest | | lowcost |
|  |  |  |
| *V* - *U* | *U* |  |
| 2 | 3 | 5 |
| 4 | 6 | 2 |
| 5 | 3 | 6 |
| Now select vertex 4, and so on | | |



**Reader Writer using OPENMP**

**Mathematical Model:**

Let R and W be set of read and writer process/thread.

R={r1,r2,…….rn}

W={w1,w2,….wm}

Where n and m is number of reader and writer process.

Let readcount, is shared variable between the readers.

Let r\_lock and w\_lock are shared locks between the readers and writers.

Let φ be the function to set the mutual exclusive lock for reader ri.

Φ\_lock(ri, r\_lock) = true if r\_lock is 1

= false if r\_lock is 0

Φ\_unlock(ri, r\_lock) = true if r\_lock is 0

= false if r\_lock is 1

If the read count is 1 then writer need to be blocked. The blocking function is g.

g(w\_lock)= true if readcount=1

= false otherwise

Let β be the function to set the mutual exclusive lock for writer wi.

β\_lock(wi, w\_lock) =true if w\_lock is 1

= false if w\_lock is 0

β\_unlock(wi, w\_lock) =true if w\_lock is 0

= false if w\_lock is 1

**Implement a Calculator application using concurrent lisp**

**Mathematical Model:**

Let T be the set of threads.

T={t1,t2,t3,...........tm}

Where m is number of threads.

Let C be the set of task to be done concurrently.

C={c1,c2,c3…..cn}

Where n is number of task.

Here m=n, indicate that one thread per task.

The task c1=addition, c2=subtraction, c3=multiplication…..

Let f be the function that assigns the task to threads.

f(ti, cj) = produce output of cj