**Sample ASC programs**

**Example 1: Source file: ex1.asc**

**main ex1**

**int parallel aa[$], bb[$], cc[$];**

**index parallel xx[$], bi[$];**

**real parallel dd[$];**

**int scalar summ;**

**read aa[$], bb[$], cc[$] in bi[$];**

**msg "Dump iput: ";**

**print aa[$], bb[$], cc[$] in bi[$];**

**summ = 0;**

**while xx in aa[$] .eq. 2**

**summ = summ + bb[xx];**

**if cc[xx] .eq. 1 .and.**

**aa[$] .eq. 2 then**

**aa[$] = 5;**

**else**

**aa[xx] = summ;**

**endif;**

**endwhile xx;**

**msg "summ=" summ;**

**print aa[$], bb[$], cc[$] in bi[$];**

**end;**

**Input file: ex1.dat, with a blank line at the end**

**1 17 0**

**2 13 0**

**2 8 1**

**3 11 1**

**2 9 0**

**4 67 0**

**Output file: ex1.out**

**Pass 2 (020505) e option --**

**INPUT VALUES FOR BI:**

**A blank line terminates input**

**AA,BB,CC,**

**Dump iput:**

**DUMP OF ASSOCIATION BI FOLLOWS:**

**AA,BB,CC,**

**1 17 0**

**2 13 0**

**2 8 1**

**3 11 1**

**2 9 0**

**4 67 0**

**summ= 21**

**DUMP OF ASSOCIATION BI FOLLOWS:**

**AA,BB,CC,**

**1 17 0**

**13 13 0**

**21 8 1**

**3 11 1**

**5 9 0**

**4 67 0**

**STOP quadruple detected.**

**==========================================**

**Example 2:**

**Source file: ex2.asc**

**main ex2**

**int parallel aa[$], bb[$], cc[$];**

**index parallel xx[$], bi[$], bbi[$];**

**real parallel dd[$];**

**int scalar summ;**

**logical parallel yy[$];**

**read aa[$], bb[$] bi[$] in bbi[$];**

**msg "Input data: ";**

**print aa[$], bb[$] bbi[$] in bbi[$];**

**yy[$] = bb[$] .eq. 13;**

**release yy[$] from bbi[$];**

**allocate xx in bbi[$]**

**aa[xx] = 5;**

**bb[$] = 6;**

**endallocate xx;**

**print aa[$], bb[$] bbi[$] xx[$] yy[$] in bbi[$];**

**msg "bi[$]=" bi[$];**

**msg "bbi[$]=" bbi[$];**

**msg "xx[$]=" xx[$];**

**end;**

**Input file: ex2.dat with a blank line at the end**

**1 17 1**

**2 13 0**

**2 8 1**

**3 11 1**

**2 9 0**

**4 67 0**

**Output file: ex2.out**

**-- Pass 2 (020505) e option --**

**INPUT VALUES FOR BBI:**

**A blank line terminates input**

**AA,BB,BI,**

**Input data:**

**DUMP OF ASSOCIATION BBI FOLLOWS:**

**AA,BB,BBI,**

**1 17 1**

**2 13 1**

**2 8 1**

**3 11 1**

**2 9 1**

**4 67 1**

**DUMP OF ASSOCIATION BBI FOLLOWS:**

**AA,BB,BBI,XX,YY,**

**1 6 1 0 0**

**5 6 1 1 1**

**2 6 1 0 0**

**3 6 1 0 0**

**2 6 1 0 0**

**4 6 1 0 0**

**bi[$]=**

**PE 0: 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0**

**PE 16: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 32: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 48: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 64: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 80: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 96: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE112: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE128: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE144: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE160: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE176: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE192: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE208: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE224: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE240: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE256: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE272: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE288: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE304: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**bbi[$]=**

**PE 0: 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0**

**PE 16: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 32: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 48: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 64: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 80: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 96: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE112: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE128: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE144: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE160: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE176: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE192: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE208: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE224: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE240: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE256: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE272: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE288: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE304: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**xx[$]=**

**PE 0: 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 16: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 32: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 48: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 64: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 80: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE 96: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE112: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE128: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE144: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE160: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE176: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE192: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE208: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE224: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE240: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE256: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE272: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE288: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**PE304: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

**STOP quadruple detected.**

**--**

**=========================================**

**Example 3: Source file: ex3.asc**

**main ex3**

**/\* for nested in a loop \*/**

**deflog (TRUE, 1);**

**deflog (FALSE, 0);**

**int parallel firsti[$], second[$], third[$];**

**int scalar node,i,j;**

**index parallel xx[$];**

**logical parallel test[$],result[$];**

**associate second[$],firsti[$],third[$] with test[$];**

**read firsti[$] second[$] third[$] in test[$];**

**print firsti[$] second[$] third[$] in test[$];**

**first**

**j=1;**

**loop**

**until j .gt. 2**

**for xx in firsti[$] .eq. j**

**if second[xx] .eq. third[$] then result[$] = TRUE;**

**else result[$] = FALSE; endif;**

**print firsti[$] second[$] third[$] in result[$];**

**endfor xx;**

**j = j + 1;**

**endloop;**

**end;**

**Input file: ex3.dat with blank at end**

**1 1 2**

**2 3 2**

**1 2 3**

**2 1 3**

**1 3 1**

**2 2 1**

**Output file: ex3.out**

**\-- Pass 2 (020505) e option --**

**INPUT VALUES FOR TEST:**

**A blank line terminates input**

**FIRSTI,SECOND,THIRD,**

**DUMP OF ASSOCIATION TEST FOLLOWS:**

**FIRSTI,SECOND,THIRD,**

**1 1 2**

**2 3 2**

**1 2 3**

**2 1 3**

**1 3 1**

**2 2 1**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**FIRSTI,SECOND,THIRD,**

**1 3 1**

**2 2 1**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**FIRSTI,SECOND,THIRD,**

**1 1 2**

**2 3 2**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**FIRSTI,SECOND,THIRD,**

**1 2 3**

**2 1 3**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**FIRSTI,SECOND,THIRD,**

**1 2 3**

**2 1 3**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**FIRSTI,SECOND,THIRD,**

**1 3 1**

**2 2 1**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**FIRSTI,SECOND,THIRD,**

**1 1 2**

**2 3 2**

**STOP quadruple detected.**

**======================================**

**Example 4: Source file: ex4.asc**

**main ex4**

**/\* Arithmetic \*/**

**deflog (TRUE, 1);**

**deflog (FALSE, 0);**

**int parallel firsti[$], second[$], third[$];**

**int scalar node,i,j;**

**index parallel xx[$];**

**logical parallel test[$],result[$];**

**associate second[$],firsti[$],third[$] with test[$];**

**read firsti[$] second[$] third[$] in test[$];**

**firsti[$] = second[$]\*third[$]\*firsti[$]/5;**

**second[$] = second[$]+third[$]+firsti[$]-5;**

**third[$] = second[$]+third[$]\*firsti[$]--5;**

**print firsti[$] second[$] third[$] in test[$];**

**end;**

**Input file: ex4.dat, with blank line at end**

**10 1 2**

**2 31 2**

**1 29 3**

**2 1 3**

**1 3 62**

**2 2 39**

**Output file: ex4.out**

**-- Pass 2 (020505) e option --**

**INPUT VALUES FOR TEST:**

**A blank line terminates input**

**FIRSTI,SECOND,THIRD,**

**DUMP OF ASSOCIATION TEST FOLLOWS:**

**FIRSTI,SECOND,THIRD,**

**4 2 15**

**24 52 105**

**17 44 100**

**1 0 8**

**37 97 2396**

**31 67 1281**

**STOP quadruple detected.**

**==========================================**

**Example 5: Source file: msti.asc**

**/\* The ASC Minimum Spanning Tree - with slight modifications from original ASC PRIMER \*/**

**main msti**

**/\* Note: Vertices were encoded as integers, not chars \*/**

**deflog (TRUE, 1);**

**deflog (FALSE, 0);**

**int parallel tail[$], head[$];**

**int parallel weight[$], state[$];**

**int scalar nodehead, nodetail;**

**index parallel xx[$];**

**logical parallel nxtnod[$], graph[$], result[$];**

**associate head[$], tail[$], weight[$], state[$] with graph[$];**

**/\* On a directed graph, the arrow goes from tail to head \*/**

**read tail[$], head[$], weight[$] in graph[$];**

**msg "Input data is: ";**

**print tail[$], head[$], weight[$] in graph[$];**

**/\* Picks first node and edge. \*/**

**setscope graph[$]**

**nodetail = tail[mindex(weight[$])];**

**nodehead = head[mindex(weight[$])];**

**endsetscope;**

**msg "nodetail and nodehead " nodetail, nodehead;**

**/\* Select all edges incident with node and put them in V2; else**

**put them in V3 \*/**

**if (nodetail == tail[$]) then state[$] = 2; else state[$] = 3; endif;**

**state[nxtnod[$]] = 1; /\* Before- failed to put start node in V1 \*/**

**/\* Throw out reverse edges \*/**

**if (head[$] == nodetail && tail[$] == nodehead) then state[$] = 0; endif;**

**while xx in (state[$] == 2)**

**/\* Loop until there are no more nodes in V2. Select lowest order PE holding minimum weight of those nodes in V2 \*/**

**if (state[$] == 2) then nxtnod[$] = mindex(weight[$]); endif;**

**/\* Select the head node in the PE chosen above\*/**

**nodehead = head[nxtnod[$]];**

**/\* Put new node in V1 \*/**

**state[nxtnod[$]] = 1;**

**/\* If selected XY for V1, throw out YX the double edge \*/**

**if (head[$] == nodetail && tail[$] == nodehead)then state[$] = 0; endif;**

**/\* Throw out edges with head the same as one picked \*/**

**if (head[$] == nodehead && state[$] != 1) then**

**state[$] = 0; endif;**

**/\* Get new candidates \*/**

**if (state[$] == 3 && nodehead == tail[$]) then state[$] = 2; endif;**

**nxtnod[$] = FALSE; /\* must clear when done for next iteration \*/**

**endwhile xx;**

**/\* All state 1 edges are in the final minimum spanning tree \*/**

**if (state[$] == 1) then result[$] = TRUE; endif;**

**print tail[$] head[$] weight[$] in result[$];**

**end;**

**Input file: mst.dat**

**1 2 3**

**2 1 3**

**2 3 1**

**3 2 1**

**3 4 4**

**4 3 4**

**4 1 8**

**1 4 8**

**Output file: mst.out**

**-- Pass 2 (020505) e option --**

**INPUT VALUES FOR GRAPH:**

**A blank line terminates input**

**TAIL,HEAD,WEIGHT,**

**Input data is:**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**1 2 3**

**2 1 3**

**2 3 1**

**3 2 1**

**3 4 4**

**4 3 4**

**4 1 8**

**1 4 8**

**nodetail and nodehead 2 3**

**34 WARNING - first called with null mask, command=1d02**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**2 1 3**

**2 3 1**

**3 4 4**

**STOP quadruple detected.**

**======================================**

**Source file: msti.asc above**

**Input file: mst1.dat:**

**1 2 3**

**2 1 3**

**2 3 1**

**3 2 1**

**3 4 4**

**4 3 4**

**4 1 8**

**1 4 2**

**5 1 9**

**6 2 2**

**2 6 2**

**7 3 1**

**3 7 1**

**8 1 4**

**1 8 4**

**8 9 6**

**9 8 6**

**9 10 2**

**10 9 2**

**6 3 4**

**3 6 4**

**10 11 4**

**11 10 4**

**10 12 3**

**12 10 3**

**9 12 2**

**12 9 2**

**Onput file: mst1.out**

**-- Pass 2 (020505) e option --**

**INPUT VALUES FOR GRAPH:**

**A blank line terminates input**

**TAIL,HEAD,WEIGHT,**

**Input data is:**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**1 2 3**

**2 1 3**

**2 3 1**

**3 2 1**

**3 4 4**

**4 3 4**

**4 1 8**

**1 4 2**

**5 1 9**

**6 2 2**

**2 6 2**

**7 3 1**

**3 7 1**

**8 1 4**

**1 8 4**

**8 9 6**

**9 8 6**

**9 10 2**

**10 9 2**

**6 3 4**

**3 6 4**

**10 11 4**

**11 10 4**

**10 12 3**

**12 10 3**

**9 12 2**

**12 9 2**

**nodetail and nodehead 2 3**

**34 WARNING - first called with null mask, command=1d02**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**2 1 3**

**2 3 1**

**1 4 2**

**2 6 2**

**3 7 1**

**1 8 4**

**8 9 6**

**9 10 2**

**10 11 4**

**9 12 2**

**STOP quadruple detected.**

**==========================================**

**Example 6: Source file: mstc.asc**

**/\* The ASC Minimum Spanning Tree - with slight modifications from ASC PRIMER \*/**

**main mst**

**/\* Note: Vertices were encoded as chars. See slides for a trace of**

**algoritm \*/**

**deflog (TRUE, 1);**

**deflog (FALSE, 0);**

**char parallel tail[$], head[$];**

**int parallel weight[$], state[$];**

**char scalar nodehead, nodetail;**

**index parallel xx[$];**

**logical parallel nxtnod[$], graph[$], result[$];**

**associate head[$], tail[$], weight[$], state[$] with graph[$];**

**/\* On a directed graph, the arrow goes from tail to head \*/**

**read tail[$], head[$], weight[$] in graph[$];**

**msg "Input data is: ";**

**print tail[$], head[$], weight[$] in graph[$];**

**/\* Picks first node and edge. \*/**

**setscope graph[$]**

**nodetail = tail[mindex(weight[$])];**

**nodehead = head[mindex(weight[$])];**

**endsetscope;**

**msg "nodetail and nodehead " nodetail, nodehead;**

**/\* Select all edges incident with node and put them in V2; else**

**put them in V3 \*/**

**if (nodetail == tail[$]) then state[$] = 2; else state[$] = 3; endif;**

**state[nxtnod[$]] = 1; /\* Error in original primer- failed to put start node in V1 \*/**

**/\* Throw out reverse edge \*/**

**if (head[$] == nodetail && tail[$] == nodehead) then state[$] = 0; endif;**

**while xx in (state[$] == 2) /\* Loop until there are no more nodes in V2 \*/**

**/\* Select lowest order PE holding minimum weight of those**

**nodes in V2 \*/**

**if (state[$] == 2) then nxtnod[$] = mindex(weight[$]); endif;**

**/\* Select the head node in the PE chosen above\*/**

**nodehead = head[nxtnod[$]];**

**/\* Put new node in V1 \*/**

**state[nxtnod[$]] = 1;**

**/\* If selected XY for V1, throw out YX the double edge \*/**

**if (head[$] == nodetail && tail[$] == nodehead)then state[$] = 0; endif;**

**/\* Throw out edges with head the same as one picked \*/**

**if (head[$] == nodehead && state[$] != 1) then**

**state[$] = 0; endif;**

**/\* Get new candidates \*/**

**if (state[$] == 3 && nodehead == tail[$]) then state[$] = 2; endif;**

**nxtnod[$] = FALSE; /\* must clear when done for next iteration \*/**

**endwhile xx;**

**/\* All state 1 edges are in the final minimum spanning tree \*/**

**if (state[$] == 1) then result[$] = TRUE; endif;**

**print tail[$] head[$] weight[$] in result[$];**

**end;**

**Input file: mstc.dat**

**Encodes the diagram in the slides**

**A B 2**

**B A 2**

**A G 3**

**G A 3**

**A F 7**

**F A 7**

**B C 4**

**C B 4**

**B G 6**

**G B 6**

**F E 6**

**E F 6**

**F I 5**

**I F 5**

**I E 2**

**E I 2**

**I H 4**

**H I 4**

**E D 1**

**D E 1**

**H D 8**

**D H 8**

**G I 1**

**I G 1**

**G H 3**

**H G 3**

**C H 2**

**H C 2**

**C D 2**

**D C 2**

**Output file: mstc.dat**

**See final slide in example in slides**

**-- Pass 2 (020505) e option --**

**INPUT VALUES FOR GRAPH:**

**A blank line terminates input**

**TAIL,HEAD,WEIGHT,**

**Input data is:**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**A B 2**

**B A 2**

**A G 3**

**G A 3**

**A F 7**

**F A 7**

**B C 4**

**C B 4**

**B G 6**

**G B 6**

**F E 6**

**E F 6**

**F I 5**

**I F 5**

**I E 2**

**E I 2**

**I H 4**

**H I 4**

**E D 1**

**D E 1**

**H D 8**

**D H 8**

**G I 1**

**I G 1**

**G H 3**

**H G 3**

**C H 2**

**H C 2**

**C D 2**

**D C 2**

**nodetail and nodehead E D**

**34 WARNING - first called with null mask, command=1d02**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**A B 2**

**G A 3**

**I F 5**

**E I 2**

**E D 1**

**I G 1**

**C H 2**

**D C 2**

**STOP quadruple detected.**

**======================================**

**Example 7: Source file: SP2.asc**

**/\*NOTE: A program for Dijkstra's**

**'s Shortest Path submitted to Dr. Slotterbeck at Hiram College for an ASC homework assignment.\*/**

**/\* Program to find the Shortest Path in a graph \*/**

**/\* Written by: Matt Boggus \*/**

**/\*\* For the algorithm given, the following variables mean: \*\*/**

**/\*\* State values:**

**1 - in shortest path \*\*/**

**/\*\* 2 - in S \*\*/**

**/\*\* 3 - consideration edge \*\*/**

**/\*\* 4 - not explored yet \*\*/**

**/\*\* 0 - explored and out \*\*/**

**/\*\* Distance stores d(v) where v is the node the edge ends at \*\*/**

**main SPdebug**

**deflog (TRUE, 1);**

**deflog (FALSE, 0);**

**/\* As the emulator doesn't support scalar input, START and ENDING**

**must be set and the program recompiled.**

**\*/**

**define (START, 1);**

**define (ENDING, 4);**

**int parallel tail[$], head[$], weight[$], state[$], distance[$];**

**int scalar node;**

**index parallel xx[$];**

**logical parallel nxtnod[$], graph[$], result[$], used[$];**

**associate head[$], tail[$], weight[$], state[$], distance[$] with graph[$];**

**/\*\* input \*\*/**

**read tail[$], head[$], weight[$] in graph[$];**

**print tail[$] head[$] weight[$] in graph[$];**

**PERFORM = 1;**

**/\*\* Create initial state of the graph and path \*\*/**

**distance[$] = 0; /\*\* set empty distance values \*\*/**

**if (tail[$] == START) then state[$] = 3; else state[$] = 4; endif;/\* set consideration edges \*/**

**if (tail[$] == START) then distance[$] = weight[$]; endif; /\* set consideration edges distance \*/**

**if (head[$] == START) then state[$] = 0; endif; /\* remove edges going into START \*/**

**if (tail[$] == ENDING) then state[$] = 0; endif; /\* remove edges going out of ENDING \*/**

**/\*\* LOOP to find S \*\*/**

**first**

**node = START;**

**loop**

**/\* Find next node to add and set values for it \*/**

**if (state[$] == 3) then nxtnod[$] = mindex(distance[$]); endif;**

**node = head[nxtnod[$]];**

**/\* Eliminate all other edges to new by changing states \*/**

**if (head[$] == node && state[$] != 2) then state[$] = 0; endif;**

**/\* Add selected edge to graph \*/**

**state[nxtnod[$]] = 2;**

**/\* Find new possible nodes and set their distances and states \*/**

**if (state[$] == 4 && tail[$] == node) then distance[$] = distance[nxtnod[$]] + weight[$]; endif;**

**if (state[$] == 4 && tail[$] == node) then state[$] = 3; endif;**

**nxtnod[$] = FALSE;**

**print tail[$] head[$] weight[$] state[$] distance[$] in graph[$];**

**until node == ENDING**

**endloop;**

**/\*\* END LOOP to find S \*\*/**

**/\*\* LOOP to trace back shortest path \*\*/**

**first**

**node = ENDING;**

**loop**

**if (state[$] == 2) then nxtnod[$] = head[$] == node; endif;**

**node = tail[nxtnod[$]];**

**state[nxtnod[$]] = 1;**

**nxtnod[$] = FALSE;**

**print tail[$] head[$] weight[$] state[$] distance[$] in graph[$];**

**until node == START**

**endloop;**

**/\*\* END LOOP to trace back shortest path \*\*/**

**PERFORM = 0;**

**MSG "SCALAR OPERATIONS" SC\_PERFORM;**

**MSG "PARALLEL OPERATIONS" PA\_PERFORM;**

**/\*\* print shortest path \*\*/**

**if (state[$] == 1) then result[$] = TRUE; endif;**

**print tail[$] head[$] weight[$] in result[$];**

**end;**

**Input file: SP2.dat**

**1 2 3**

**2 1 3**

**2 3 1**

**3 2 1**

**3 4 4**

**4 3 4**

**4 1 9**

**1 4 9**

**Output file: SP2.OUT**

**-- Pass 2 (020505) e option --**

**INPUT VALUES FOR GRAPH:**

**A blank line terminates input**

**TAIL,HEAD,WEIGHT,**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**1 2 3**

**2 1 3**

**2 3 1**

**3 2 1**

**3 4 4**

**4 3 4**

**4 1 9**

**1 4 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 2 3**

**2 1 3 0 0**

**2 3 1 3 4**

**3 2 1 0 0**

**3 4 4 4 0**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 3 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 2 3**

**2 1 3 0 0**

**2 3 1 2 4**

**3 2 1 0 0**

**3 4 4 3 8**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 3 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 2 3**

**2 1 3 0 0**

**2 3 1 2 4**

**3 2 1 0 0**

**3 4 4 2 8**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 0 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 2 3**

**2 1 3 0 0**

**2 3 1 2 4**

**3 2 1 0 0**

**3 4 4 1 8**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 0 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 2 3**

**2 1 3 0 0**

**2 3 1 1 4**

**3 2 1 0 0**

**3 4 4 1 8**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 0 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 1 3**

**2 1 3 0 0**

**2 3 1 1 4**

**3 2 1 0 0**

**3 4 4 1 8**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 0 9**

**SCALAR OPERATIONS 63**

**PARALLEL OPERATIONS 1495**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**1 2 3**

**2 3 1**

**3 4 4**

**STOP quadruple detected**

**==========================================**

**Example 8: Source file: SP.asc**

**/\*NOTE: A program for Dijkstra's Shortest Path submitted to for an ASC homework assignment in Dr. Slotterbeck’s Hiram College class.**

**------------------------------------\*/**

**/\* Program to find the Shortest Path in a graph \*/**

**/\* Written by: Matt Boggus \*/**

**/\*\* For the algorithm given, the following variables mean: \*\*/**

**/\*\* State values:**

**1 - in shortest path 2 - in S 3 - consideration edge 4 - not explored yet 0 - explored and out \*\*/**

**/\*\* Distance stores d(v) where v is the node the edge ends at \*\*/**

**main SP**

**deflog (TRUE, 1);**

**deflog (FALSE, 0);**

**/\* IMPORTANT NOTE: As the emulator doesn't support scalar input, START and ENDING must be set manually and the program recompiled.\*/**

**define (START, 1);**

**define (ENDING, 11);**

**int parallel tail[$], head[$], weight[$], state[$], distance[$];**

**int scalar node;**

**index parallel xx[$];**

**logical parallel nxtnod[$], graph[$], result[$], used[$];**

**associate head[$], tail[$], weight[$], state[$], distance[$] with graph[$];**

**/\*\* input \*\*/**

**read tail[$], head[$], weight[$] in graph[$];**

**PERFORM = 1;**

**/\*\* Create initial state of the graph and path \*\*/**

**distance[$] = 0; /\*\* set empty distance values \*\*/**

**if (tail[$] == START) then state[$] = 3; else state[$] = 4; endif;/\* set consideration edges \*/**

**if (tail[$] == START) then distance[$] = weight[$]; endif; /\* set consideration edges distance \*/**

**if (head[$] == START) then state[$] = 0; endif; /\* remove edges going into START \*/**

**if (tail[$] == ENDING) then state[$] = 0; endif; /\* remove edges going out of ENDING \*/**

**/\*\* LOOP to find S \*\*/**

**first**

**node = START;**

**loop**

**/\* Find next node to add and set values for it \*/**

**if (state[$] == 3) then nxtnod[$] = mindex(distance[$]); endif;**

**node = head[nxtnod[$]];**

**/\* Eliminate all other edges to new node by changing states \*/**

**if (head[$] == node && state[$] != 2) then state[$] = 0; endif;**

**/\* Add selected edge to graph \*/**

**state[nxtnod[$]] = 2;**

**/\* Find new possible nodes and set their distances and states \*/**

**if (state[$] == 4 && tail[$] == node) then distance[$] = distance[nxtnod[$]] + weight[$]; endif;**

**if (state[$] == 4 && tail[$] == node) then state[$] = 3; endif;**

**nxtnod[$] = FALSE;**

**until node == ENDING**

**endloop;**

**/\*\* END LOOP to find S \*\*/**

**/\*\* LOOP to trace back shortest path \*\*/**

**first**

**node = ENDING;**

**loop**

**if (state[$] == 2) then nxtnod[$] = head[$] == node; endif;**

**node = tail[nxtnod[$]];**

**state[nxtnod[$]] = 1;**

**nxtnod[$] = FALSE;**

**until node == START**

**endloop;**

**/\*\* END LOOP to trace back shortest path \*\*/**

**PERFORM = 0;**

**MSG "SCALAR OPERATIONS" SC\_PERFORM;**

**MSG "PARALLEL OPERATIONS" PA\_PERFORM;**

**/\*\* print shortest path \*\*/**

**if (state[$] == 1) then result[$] = TRUE; endif;**

**print tail[$] head[$] weight[$] in result[$];**

**end;**

**Input file: SP.dat**

**5 4 2**

**4 5 2**

**12 4 1**

**4 12 1**

**1 2 3**

**2 1 3**

**2 3 1**

**3 2 1**

**3 4 4**

**4 3 4**

**4 1 8**

**1 4 8**

**5 1 9**

**1 5 9**

**6 2 2**

**2 6 2**

**7 3 1**

**3 7 1**

**8 1 4**

**1 8 4**

**8 9 6**

**9 8 6**

**9 10 2**

**10 9 2**

**6 3 4**

**3 6 4**

**10 11 4**

**11 10 4**

**10 12 3**

**12 10 3**

**9 12 2**

**12 9 2**

**Output file: SP.out**

**-- Pass 2 (020505) e option --**

**INPUT VALUES FOR GRAPH:**

**A blank line terminates input**

**TAIL,HEAD,WEIGHT,**

**SCALAR OPERATIONS 171**

**PARALLEL OPERATIONS 3771**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**1 8 4**

**8 9 6**

**9 10 2**

**10 11 4**

**STOP quadruple detected.**

**==========================================**

**Example 9: Source file: SP2.asc**

**/\*NOTE: Dijkstra's Shortest Path with debug dumps.\*/**

**/\* Program to find the Shortest Path in a graph \*/**

**/\* Written by: Matt Boggus (with additional debug print statements to see the execution.) \*/**

**/\*\* For the algorithm given, the following variables mean: \*\*/**

**/\*\* State values:**

**1 - in shortest path \*\*/**

**/\*\* 2 - in S \*\*/**

**/\*\* 3 - consideration edge \*\*/**

**/\*\* 4 - not explored yet \*\*/**

**/\*\* 0 - explored and out \*\*/**

**/\*\* Distance stores d(v) where v is the node the edge ends at \*\*/**

**main SPdebug**

**deflog (TRUE, 1);**

**deflog (FALSE, 0);**

**/\* As the emulator doesn't support scalar input, START and ENDING**

**must be set and the program recompiled.**

**\*/**

**define (START, 1);**

**define (ENDING, 11);**

**int parallel tail[$], head[$], weight[$], state[$], distance[$];**

**int scalar node;**

**index parallel xx[$];**

**logical parallel nxtnod[$], graph[$], result[$], used[$];**

**associate head[$], tail[$], weight[$], state[$], distance[$] with graph[$];**

**/\*\* input \*\*/**

**read tail[$], head[$], weight[$] in graph[$];**

**print tail[$] head[$] weight[$] in graph[$];**

**PERFORM = 1;**

**/\*\* Create initial state of the graph and path \*\*/**

**distance[$] = 0; /\*\* set empty distance values \*\*/**

**if (tail[$] == START) then state[$] = 3; else state[$] = 4; endif;/\* set consideration edges \*/**

**if (tail[$] == START) then distance[$] = weight[$]; endif; /\* set consideration edges distance \*/**

**if (head[$] == START) then state[$] = 0; endif; /\* remove edges going into START \*/**

**if (tail[$] == ENDING) then state[$] = 0; endif; /\* remove edges going out of ENDING \*/**

**/\*\* LOOP to find S \*\*/**

**first**

**node = START;**

**loop**

**/\* Find next node to add and set values for it \*/**

**if (state[$] == 3) then nxtnod[$] = mindex(distance[$]); endif;**

**node = head[nxtnod[$]];**

**/\* Eliminate all other edges to new by changing states \*/**

**if (head[$] == node && state[$] != 2) then state[$] = 0; endif;**

**/\* Add selected edge to graph \*/**

**state[nxtnod[$]] = 2;**

**/\* Find new possible nodes and set their distances and states \*/**

**if (state[$] == 4 && tail[$] == node) then distance[$] = distance[nxtnod[$]] + weight[$]; endif;**

**if (state[$] == 4 && tail[$] == node) then state[$] = 3; endif;**

**nxtnod[$] = FALSE;**

**print tail[$] head[$] weight[$] state[$] distance[$] in graph[$];**

**until node == ENDING**

**endloop;**

**/\*\* END LOOP to find S \*\*/**

**/\*\* LOOP to trace back shortest path \*\*/**

**first**

**node = ENDING;**

**loop**

**if (state[$] == 2) then nxtnod[$] = head[$] == node; endif;**

**node = tail[nxtnod[$]];**

**state[nxtnod[$]] = 1;**

**nxtnod[$] = FALSE;**

**print tail[$] head[$] weight[$] state[$] distance[$] in graph[$];**

**until node == START**

**endloop;**

**/\*\* END LOOP to trace back shortest path \*\*/**

**PERFORM = 0;**

**MSG "SCALAR OPERATIONS" SC\_PERFORM;**

**MSG "PARALLEL OPERATIONS" PA\_PERFORM;**

**/\*\* print shortest path \*\*/**

**if (state[$] == 1) then result[$] = TRUE; endif;**

**print tail[$] head[$] weight[$] in result[$];**

**end;**

**Input file: SP2.dat**

**-- Pass 2 (020505) e option --**

**INPUT VALUES FOR GRAPH:**

**A blank line terminates input**

**TAIL,HEAD,WEIGHT,**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**1 2 3**

**2 1 3**

**2 3 1**

**3 2 1**

**3 4 4**

**4 3 4**

**4 1 9**

**1 4 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 2 3**

**2 1 3 0 0**

**2 3 1 3 4**

**3 2 1 0 0**

**3 4 4 4 0**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 3 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 2 3**

**2 1 3 0 0**

**2 3 1 2 4**

**3 2 1 0 0**

**3 4 4 3 8**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 3 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 2 3**

**2 1 3 0 0**

**2 3 1 2 4**

**3 2 1 0 0**

**3 4 4 2 8**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 0 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 2 3**

**2 1 3 0 0**

**2 3 1 2 4**

**3 2 1 0 0**

**3 4 4 1 8**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 0 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 2 3**

**2 1 3 0 0**

**2 3 1 1 4**

**3 2 1 0 0**

**3 4 4 1 8**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 0 9**

**DUMP OF ASSOCIATION GRAPH FOLLOWS:**

**TAIL,HEAD,WEIGHT,STATE,DISTANCE,**

**1 2 3 1 3**

**2 1 3 0 0**

**2 3 1 1 4**

**3 2 1 0 0**

**3 4 4 1 8**

**4 3 4 0 0**

**4 1 9 0 0**

**1 4 9 0 9**

**SCALAR OPERATIONS 63**

**PARALLEL OPERATIONS 1495**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**1 2 3**

**2 3 1**

**3 4 4**

**STOP quadruple detected.**

**Example 9: Source file: SPcountsA.asc**

**/\* Program to find the Shortest Path in a graph \*/**

**/\* Written by: Matt Boggus \*/**

**/\* Modified to show counts of pa\_perform and sc\_perform during the run \*/**

**/\*\* For the algorithm given, the following variables mean:**

**State values:**

**1 - in shortest path**

**2 - in S**

**3 - consideration edge**

**4 - not explored yet**

**0 - explored and out \*\*/**

**/\*\*Distance stores d(v) where v is the node the edge ends at \*\*/**

**main SPcountsA**

**deflog (TRUE, 1);**

**deflog (FALSE, 0);**

**/\* As the emulator doesn't support scalar input, START and ENDING**

**must be set and the program recompiled.**

**\*/**

**define (START, 1); /\* NOTE hand set \*/**

**define (ENDING, 5);**

**int parallel tail[$], head[$], weight[$], state[$], distance[$];**

**int scalar node;**

**index parallel xx[$];**

**logical parallel nxtnod[$], graph[$], result[$], used[$];**

**associate head[$], tail[$], weight[$], state[$], distance[$] with graph[$];**

**/\*\* input \*\*/**

**read tail[$], head[$], weight[$] in graph[$];**

**PERFORM = 1; /\* monitor on \*/**

**/\*\* Create initial state of the graph and path \*\*/**

**distance[$] = 0; /\* set empty distance values \*/**

**if (tail[$] == START) then state[$] = 3; else state[$] = 4; endif; /\* set consideration edges \*/**

**if (tail[$] == START) then distance[$] = weight[$]; endif;**

**/\* set consideration edges distance \*/**

**if (head[$] == START) then state[$] = 0; endif;**

**/\* remove edges going into START \*/**

**if (tail[$] == ENDING) then state[$] = 0; endif;**

**/\* remove edges going out of ENDING \*/**

**MSG " ";**

**MSG " ";**

**MSG "Created initial state: ";**

**MSG "SCALAR OPERATIONS" SC\_PERFORM;**

**MSG "PARALLEL OPERATIONS" PA\_PERFORM;**

**MSG " ";**

**MSG " ";**

**/\*\* LOOP to find S \*\*/**

**first**

**node = START;**

**loop**

**/\* Find next node to add and set values for it \*/**

**if (state[$] == 3) then nxtnod[$] = mindex(distance[$]); endif;**

**node = head[nxtnod[$]];**

**/\* Eliminate all other edges to new node by changing states \*/**

**if (head[$] == node && state[$] != 2) then state[$] = 0; endif;**

**/\* Add selected edge to graph \*/**

**state[nxtnod[$]] = 2;**

**/\* Find new possible nodes and set their distances and states \*/**

**if (state[$] == 4 && tail[$] == node) then distance[$] = distance[nxtnod[$]] + weight[$]; endif;**

**if (state[$] == 4 && tail[$] == node) then state[$] = 3; endif;**

**nxtnod[$] = FALSE;**

**MSG "In loop to find S ";**

**MSG "SCALAR OPERATIONS" SC\_PERFORM;**

**MSG "PARALLEL OPERATIONS" PA\_PERFORM;**

**MSG " ";**

**MSG " ";**

**until node == ENDING**

**endloop;**

**/\*\* END LOOP to find S \*\*/**

**/\*\* LOOP to trace back shortest path \*\*/**

**first**

**node = ENDING;**

**loop**

**if (state[$] == 2) then nxtnod[$] = head[$] == node; endif;**

**node = tail[nxtnod[$]];**

**state[nxtnod[$]] = 1;**

**nxtnod[$] = FALSE;**

**MSG "In loop to find S ";**

**MSG "SCALAR OPERATIONS" SC\_PERFORM;**

**MSG "PARALLEL OPERATIONS" PA\_PERFORM;**

**MSG " ";**

**MSG " ";**

**until node == START**

**endloop;**

**/\*\* END LOOP to trace back shortest path \*\*/**

**/\*\* print shortest path \*\*/**

**if (state[$] == 1) then result[$] = TRUE; endif;**

**print tail[$] head[$] weight[$] in result[$];**

**PERFORM = 0;**

**MSG "Fnal counts: ";**

**MSG "SCALAR OPERATIONS" SC\_PERFORM;**

**MSG "PARALLEL OPERATIONS" PA\_PERFORM;**

**end;**

**Input file: SPcounts.dat with blank line at end**

**1 2 3**

**2 1 3**

**2 3 1**

**3 2 1**

**3 4 4**

**4 3 4**

**4 1 9**

**1 4 9**

**2 4 5**

**4 2 5**

**4 5 1**

**5 4 1**

**3 6 1**

**6 3 1**

**5 6 2**

**6 5 2**

**Output file A.out. Shows how the performance monitor executes**

**-- Pass 2 (020505) e option --**

**INPUT VALUES FOR GRAPH:**

**A blank line terminates input**

**TAIL,HEAD,WEIGHT,**

**Created initial state:**

**SCALAR OPERATIONS 9**

**PARALLEL OPERATIONS 175**

**In loop to find S**

**SCALAR OPERATIONS 23**

**PARALLEL OPERATIONS 404**

**In loop to find S**

**SCALAR OPERATIONS 37**

**PARALLEL OPERATIONS 722**

**In loop to find S**

**SCALAR OPERATIONS 51**

**PARALLEL OPERATIONS 1038**

**In loop to find S**

**SCALAR OPERATIONS 62**

**PARALLEL OPERATIONS 1210**

**In loop to find S**

**SCALAR OPERATIONS 67**

**PARALLEL OPERATIONS 1380**

**In loop to find S**

**SCALAR OPERATIONS 72**

**PARALLEL OPERATIONS 1550**

**In loop to find S**

**SCALAR OPERATIONS 77**

**PARALLEL OPERATIONS 1720**

**In loop to find S**

**SCALAR OPERATIONS 82**

**PARALLEL OPERATIONS 1890**

**DUMP OF ASSOCIATION RESULT FOLLOWS:**

**TAIL,HEAD,WEIGHT,**

**1 2 3**

**2 3 1**

**3 6 1**

**6 5 2**

**Fnal counts:**

**SCALAR OPERATIONS 84**

**PARALLEL OPERATIONS 1971**

**STOP quadruple detected.**