FACIT TILL
OMTENTAMEN I
DATASTRUKTURER OCH ALGORITMER DAV B03

130611 kl. 08:15 – 13:15

Ansvarig Lärare: Donald F. Ross

Hjälpmedel: Inga. Algoritmerna finns i de respektive uppgifterna.

***** OBS *****

Ni som har läst från och med HT 2006

Betygsgräns:

Kurs: Max 60p, Med beröm godkänd 50p, Icke utan beröm godkänd 40p, Godkänd 30p
(varav minimum 15p från tentamen, 15p från labbarna)
Tentamen: Max 30p, betyg 5: 26p-30p, betyg 4: 21p-25p, betyg 3: 15p-20p
Labbarna: Max 30p, betyg 5: 26p-30p, betyg 4: 21p-25p, betyg 3: 15p-20p

Ni som har läst tidigare än HT 2006

Betygsgräns:

Kurs: Max 60p, Med beröm godkänd 50p, Icke utan beröm godkänd 40p, Godkänd 30p
(varav minimum 20p från tentamen, 10p från labbarna)
Tentamen: Max 40p, betyg 5: 34p-40p, betyg 4: 27p-33p, betyg 3: 20p-26p
Labbarna: Max 20p, betyg 5: 18p-20p, betyg 4: 14p-17p, betyg 3: 10p-13p

SKRIV TYDLIGT – LÄS UPPGIFTERNA NOGGRANT

***** OBS *** Denna tentamen är kopierad på båda sidor *** OBS *****

(1) Ge ett kortfattat svar till följande uppgifter.

(a) Vad är ”big-O” för en funktion som skriver ut en adjacency matrix? Varför?
$O(n^2)$ – matrix is 2D which implies 2 nested for loops to display the content.

(b) Vad gör Dijkstras algorithm?
Calculates the shortest PATH between a given node (the start node) and the remaining nodes in the graph.

(c) Vad gör Floyds algorithm?
All pairs shortest path algorithm. Calculates the shortest PATH between each pair of nodes ((a, b) a != b)) in the graph.

(d) Vad gör Warshalls algorithm?
Calculates the transitive closure of the graph, i.e. if there is a PATH between any pair of nodes (a, b).

(e) Vad gör Topologisk sortering?
Given a DAG as input, produces a sequence which represents a partial ordering of the nodes in the DAG (Directed Acyclic Graph).

(f) Vad är en heap?
A data structure, which may be represented as an array or as a (binary) tree with the property that the parent node has a value which is greater than (or less than) its children. Is used to implement a priority queue (PQ)

(g) Vad är fördelen med hashning?
The add and find operations are O(1).

(h) Vad är en rekursiv funktion?
A function which calls itself – usually in a conditional call otherwise the function will ”disappear” in an endless sequence of recursive calls.

(i) Vad är ett AVL-träd?
A BST, Binary Search Tree, with an added constraint that the height of the left and right sub-trees may not differ by more than 1.

(j) Vad är dubbel hashning?
A conflict resolution technique where the f(i) function is a second hash function. Give an example.

Totalt 5p

(2) Sekvens

Diskutera ingående ADT:en sekvens, dess egenskaper och varför sekvensen är så viktigt inom datavetenskap. Vilket förhållande har sekvensen till ADT:erna ”träd” och ”graf”?

5p

- **Discuss the definition of a sequence (recursive or non recursive)**
- **Discuss operations on the sequence**
- **Importance: used throughout computer science**
  - **Sequential files**
  - **DB systems a sequential collection of tuples**
  - **Text editors – a sequence of characters**
  - **Basic program flow of control in programs – sequence**
  - **Excluding parallel machines, most von Neumann architectures are based on sequential execution and sequential memory (data + code)**
  - **Models state machines and state changes in time $S_0 \rightarrow S_1 \rightarrow \ldots \rightarrow S_n$**
- **Relationship to trees – a binary tree can be represented as an array (cf heap)**
- **Relationship to graphs – a graph is typically implemented as and adjacency list which is a sequence of sequences or an adjacency matrix which is a sequence of arrays**

Marks for good points and discussion.

(3) Träd

(a) Ange en rekursiv definition av ett träd.

(1p)

BT ::= LC N RC | empty
N ::= element
LC ::= BT
RC ::= BT

(b) Utifrån din definition i (a) ovan, skriv rekursiv pseudokod för att räkna antalet element i ett binärt träd.
Ange alla antaganden.
Ange ett exempel för att visa hur din kod fungerar.

(2p)

```
static int b_card(treeref T)
{
  return is_empty(T) ? 0 : 1 + b_card(LC(T)) + b_card(RC(T));
  }
```

(c) Under kursens lopp hade vi diskuterat 2 sätt att ta bort ett element från ett BST. Vilka var dessa?
Ange alla antaganden.
Ange exempel för att visa hur en ”ta bort” funktion skulle fungerar.

(2p)

Totalt 5p

Method 1:

```
BT remove(T, v) {
   if IsEmpty(T) then return T
   if v < value(T) then return cons(Remove(left(T), v), T,  right(T))
   if v > value(T) then return cons(left(T), T, Remove(right(T), v))
   return add(left(T), right(T));   // add right child to left child
   }
```

Method 2: Take the value to be removed, which is the root node of a sub-tree, swap this value with either (i) the lowest valued leaf node of the right sub-tree or (ii) the highest valued leaf node of the left sub-tree and then remove the lowest/highest valued node that was used in the swap.

```
e.g. remove 5 from      [5]        might give      [4]      or         [7]
                     [3]     [8]                [3]     [8]        [3]    [8]
                   ¤   [4] [7]   ¤            ¤    ¤  [7]    ¤    ¤   [4] ¤   ¤
```

(4) Övriga frågor – svar ingående med exempel

(a) En student har skapat ett AVL-träd från följande sekvens:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
Rita slutversionen av trädet. 1p

```
                    [8]
          [4]                       [12]
     [2]         [6]           [10]          [14]
[1]      [3]  [5]    [7]     [9]     [11]  [13]     [15]
```

(b) Hur gör man för att skriva ut ett binärträd i 2 dimensioner?
Vilka av trädets egenskaper använder man? 2p

```
     e.g.              [8]
          [4]                       [12]
     [2]         [6]              ¤           [14]
[1]      ¤     ¤      [7]                  [13]     [15]
```

Perform a breadth first search through the tree ans store the results in a sequence (Q) so the tree above becomes

[8] | [4] [12] | [2] [6] [¤] [14] | [1] [¤] [¤] [7] [¤] [¤] [13] [15]
Then work out the height h of the tree (4) and calculate the maximum number of leaf nodes (2^{h-1}) which gives 8 leaf nodes. This can be used to calculate the maximum display width for the tree 8 * width(element) and then the tree can be displayed using the height h (h rows) with level = 1..h which determines the number of nodes per level to be displayed ($2^{level -1}$) i.e. level 1 (from the root) 1 node; level 2, 2 nodes; level 3, 4 nodes; level 4, 8 nodes.
If you consider that the leaf nodes have height 0 and not 1 then adjust the figures accordingly.

(c) Förklara principerna bakom **Kruskals algorithm**. 2p

Totalt 5p

(1) Each node forms a component
(2) Add an edge with least cost (use a PQ) which connects 2 distinct components until there is only one component left
(3) This will be a free tree with n-1 edges where |V| = n

(5) Labbkod

(a) I graflabben har en student skrivit följande kod för att ta bort en kant (edge) från an adjacency lista. Förklara **ingående** hur koden fungerar. Använd gärna exempel. **Ange alla antagande.**
Vilka är förutsättningarna för att koden ska fungera?

```
void reme(char cs, char cd) {
  set_edges(b_findn(cs, G), b_reme(cd, get_edges(b_findn(cs, G))));
  }
```

2p

Assumptions: (i) G is a reference to the tree, (ii) the tree is represented as an adjacency list (AL) (iii) (cs, cd) define the edge. Working from the inside out (functional thinking) b_findn(cs, G) gives a reference to the node in the AL; get_edges(N) then gives a reference to the edge list for this node and b_reme(e, Elist) removes cd from this edge list and returns a (new) reference to the edge list which is "reconnected" to the edge list of the node cs by set_edges(N, Elist)

(b) I trädlabben har en student skrivit kod för att söka efter ett värde i ett BST (binärt sökträd). Sedan har studenten kommit på att denna kod kunde lätt anpassas för att söka efter ett värde i ett komplett träd. Dessa funktioner finns nedan. Vad har studenten skrivit för "xxx" och "yyy"? **Ange alla antagande.**

```
static int b_findb(treeref T, int v)
{
  return is_empty(T) ? 0
        : v <  get_value(node(T)) ? b_findb(LC(T), v)
        : v >  get_value(node(T)) ? b_findb(RC(T), v)
        : 1;
}

static int b_findc(treeref T, int v)
{
  return is_empty(T) ? 0
        : xxx ? 1            xxx ➔ v == get_value(node(T))
        : yyy;               yyy ➔ b_findc(LC(T), v) || b_findc(RC(T), v);
}
```

1p

(c) Skriv (pseudo)kod för att lägga till ett element i ett binärt träd.
Ange alla antagande. **2p**

```
T: Add(T,v) {
    if IsEmpty(T) then return v
    if IsEmpty(v) then return T
    if value(v) < value(T) then return cons(Add(left(T), v), T, right(T))
    if value(v) > value(T) then return cons(left(T), T,  Add(right(T), v))
    return T
    }
```

Totalt 5p

(6) Graf - Prims algorithm

Tillämpa **den givna Prims algoritm nedan** på **den oriktade grafen**,

(a-20-b, a-36-c, a-34-d, b-22-c, b-24-e, c-28-d, c-30-e, c-38-f, d-26-f, e-36-f).

Börja med nod "a".

Ange *alla* antaganden och visa *alla* beräkningar och mellanresultat

(3p)

Förklara **principerna** bakom Prims algoritm. Använd gärna exemplet ovan.

(2p)

```
Prim ( node v) -- v is the start node
{   U = {v}; for i in (V-U) { low-cost[i] = C[v,i]; closest[i] = v; }

    while (!is_empty (V-U) ) {
        i = first(V-U); min = low-cost[i]; k = i;
        for j in (V-U-k) if (low-cost[j] < min) {min = low-cost[j]; k = j; }
        display(k, closest[k]);
        U = U + k
        for j in (V-U) if ( C[k,j] < low-cost[j] ) )   {low-cost[j] = C[k,j]; closest[j] = k; }
        }
    }
```

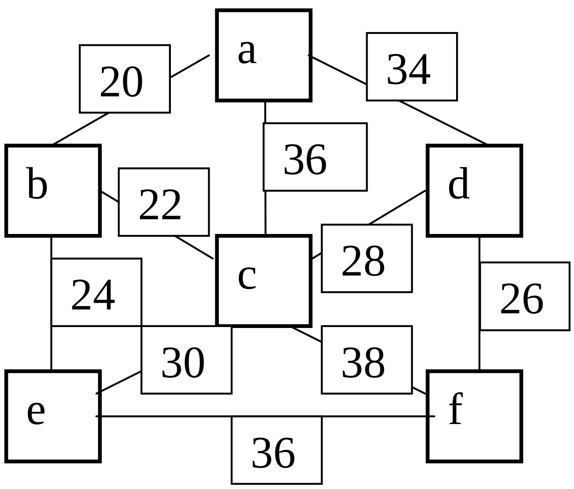
Totalt 5p

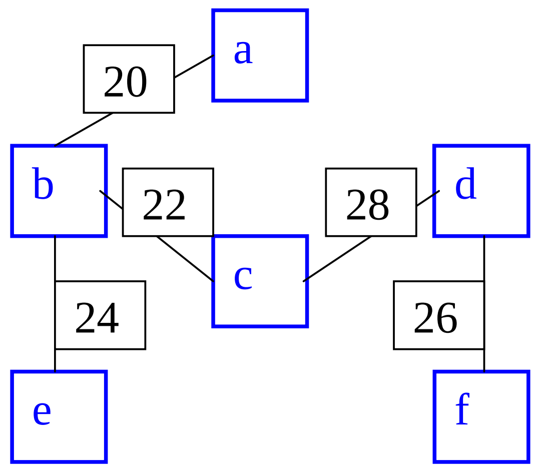
The principle is that the MST ” grows” from the **one component** (here ”a”) by connecting this component to any other component (a node) by the shortest edge and then this component “grows” to form the MST – this last proviso reveals that Prim’s is a GREEDY algorithm i.e. used a local best solution.

See below for the calculations.

Draw the graph (and possibly sketch the answer – use Kruskalls for a quick check!):

Cost 120





Draw the cost matrix C and array D

	a	b	c	d	e	f
a		**20**	**36**	**34**		
b	**20**		**22**		**24**	
c	**36**	**22**		**28**	**30**	**38**
d	**34**		**28**			**26**
e		**24**	**30**			**36**
f			**38**	**26**	**36**	

a	b	c	d	e	f
lowcost	**20**	**36**	**34**	**§**	**§**
closest	**a**	**a**	**a**	**a**	**a**

Minedge**: lowcost: 20 36 34 § § closest: a a a a a** U = {a,b} V-U = {c,d,e,f} min = **20**; k = **b**
Readjust costs: if C[k,j] < lowcost[j] then { lowcost[j] = C[k,j]; closest[j] = k }
j = c; if C[b,c] < lowcost[c] then { lowcost[c] = C[b,c]; closest[b] = b } → 22<36 ➔ **c-22-b**
j = d; if C[b,d] < lowcost[d] then { lowcost[d] = C[b,d]; closest[d] = b } → §<34 ➔ no change
j = e; if C[b,e] < lowcost[e] then { lowcost[e] = C[b,e]; closest[e] = b } → 24<§ ➔ **b-24-e**
j = f; if C[b,f] < lowcost[f] then { lowcost[f] = C[b,f]; closest[f] = b } → §<§ ➔ no change

Minedge**: lowcost: 20 22 34 24 § closest: a b a b a** U = {a,b,c} V-U = {d,e,f} min = **22**; k = **c**
Readjust costs: if C[k,j] < lowcost[j] then { lowcost[j] = C[k,j]; closest[j] = k }
j = d; if C[c,d] < lowcost[d] then { lowcost[d] = C[c,d]; closest[d] = c } → 28<34 → **c-28-d**
j = e; if C[c,e] < lowcost[e] then { lowcost[e] = C[c,e]; closest[e] = c} → 30<24 → no change
j = f; if C[c,f] < lowcost[f] then { lowcost[f] = C[c,f]; closest[f] = c } → 38<§ → **c-38-f**

Minedge**: lowcost: 20 22 28 24 38 closest: a b c b c** U = {a,b,c,e} V-U = {d,f} min = **24**; k = **e**
j = d; if C[e,d] < lowcost[d] then { lowcost[d] = C[e,d]; closest[d] = e } → §<28 → no change
j = f; if C[e,f] < lowcost[f] then { lowcost[f] = C[e,f]; closest[f] = e } → 36<38 ➔ **e-36-f**

Minedge**: lowcost: 20 22 28 24 36 closest: a b c b e** U = {a,b,c,e,d} V-U = {f} min = **28**; k = **d**
j = f; if C[d,f] < lowcost[f] then { lowcost[f] = C[d,f]; closest[f] = d} → 26<36 → **d-26-f**

Min edge**: lowcost: 20 22 28 24 26 --- closest: a b c b d ---** U = {a,c,b,d,f e} V-U = {¤}

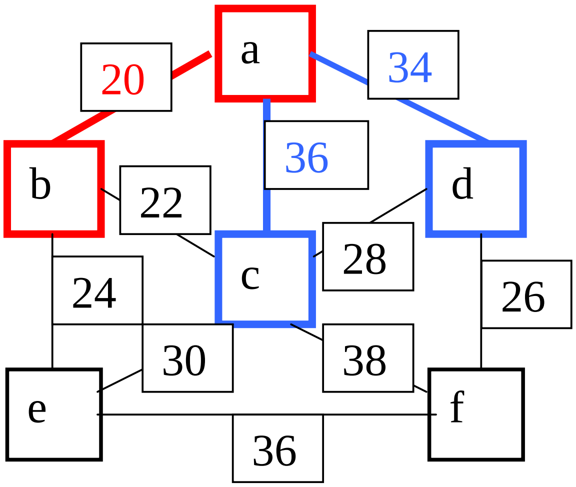
QED ☺ MST edges **a-20-b, b-22-c, b-24-e, c-28-d, d-26-f Total cost = 120**

(Confirm using Kruskal's)

The sequence of components as the MST graph develops:-

lowcost: 20 36 34 § § closest: a a a a a component a-20-b

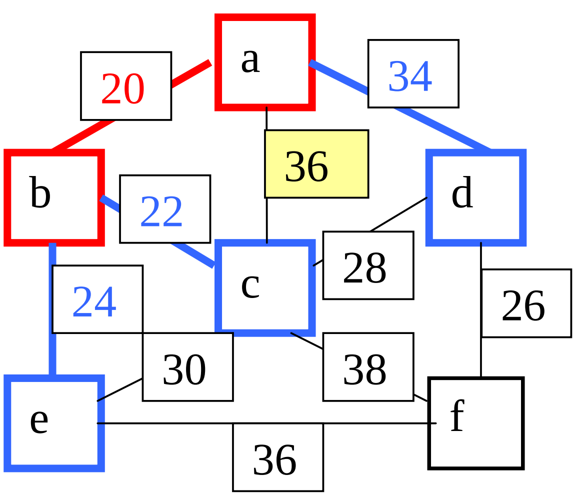
nodes c, d are reachable from a – a-36-c, a-34-d; nodes e, f are initially not reachable from a



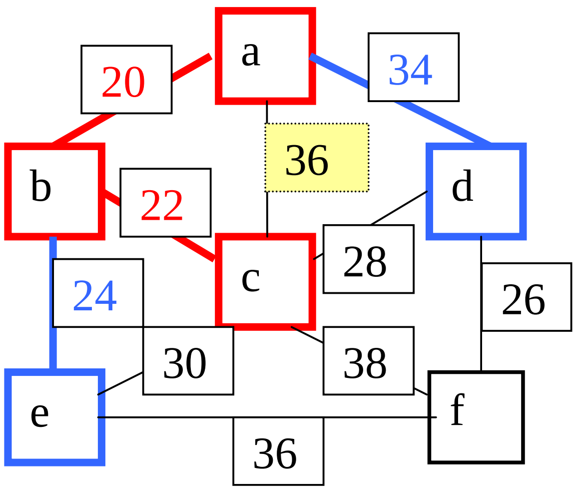
Now check to see if there are cheaper ways of reaching c, d, e, f (non component nodes) from the component a-20-b. We already know the costs from a so now look at costs from b

Minedge**: lowcost: 20 36 34 § § closest: a a a a a** U = {a,b} V-U = {c,d,e,f} min = **20**; k = **b**
Readjust costs: if C[k,j] < lowcost[j] then { lowcost[j] = C[k,j]; closest[j] = k }
j = c; if C[b,c] < lowcost[c] then { lowcost[c] = C[b,c]; closest[b] = b } → 22<36 ➔ **c-22-b**
j = d; if C[b,d] < lowcost[d] then { lowcost[d] = C[b,d]; closest[d] = b } → §<34 ➔ no change
j = e; if C[b,e] < lowcost[e] then { lowcost[e] = C[b,e]; closest[e] = b } → 24<§ ➔ **b-24-e**
j = f; if C[b,f] < lowcost[f] then { lowcost[f] = C[b,f]; closest[f] = b } → §<§ ➔ no change

b-22-c is **cheaper** than a-36 c
b-§-d is not cheaper than a-34-d
b-24-e is **cheaper** than a-§-e
b-§-f is not cheaper than a-§-f
and the picture becomes



lowcost: 20 22 34 24 § closest: a b a b a
b-22-c is the cheapest edge from the component a-20-b to the remaining nodes so make this part of the component [a-20-b, b-22-c]

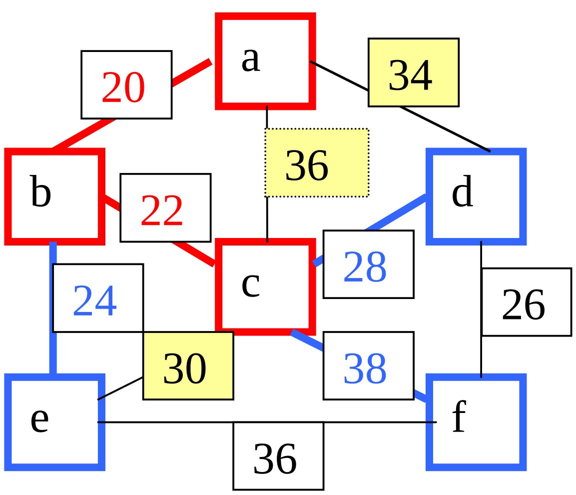


lowcost: 20 22 34 24 § closest: a b a b a

And repeat the process from c to d, e, f

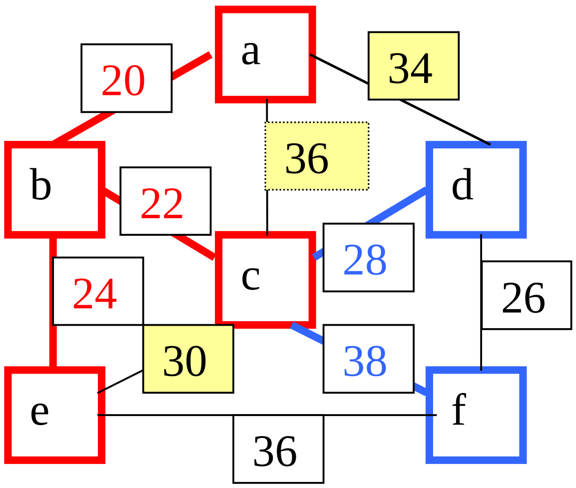
Minedge**: lowcost: 20 22 34 24 § closest: a b a b a** U = {a,b,c} V-U = {d,e,f} min = **22**; k = **c**
Readjust costs: if C[k,j] < lowcost[j] then { lowcost[j] = C[k,j]; closest[j] = k }
j = d; if C[c,d] < lowcost[d] then { lowcost[d] = C[c,d]; closest[d] = c } → 28<34 → **c-28-d**
j = e; if C[c,e] < lowcost[e] then { lowcost[e] = C[c,e]; closest[e] = c} → 30<24 → no change
j = f; if C[c,f] < lowcost[f] then { lowcost[f] = C[c,f]; closest[f] = c } → 38<§ → **c-38-f**

This gives **lowcost: 20 22 28 24 38 closest: a b c b c**



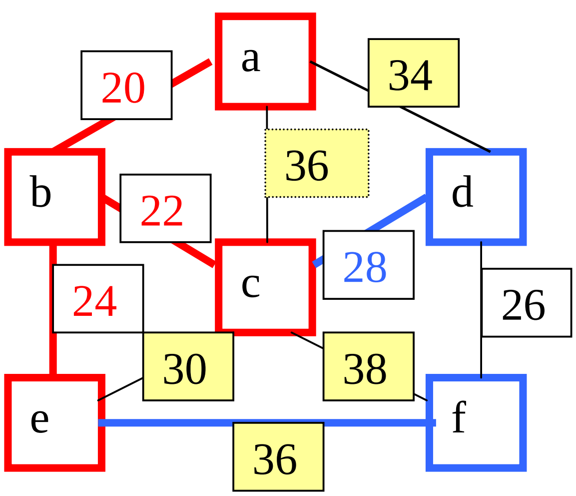
lowcost: 20 22 28 <u>24</u> 38 closest: a b c b c

The cheapest edge is **b-24-e** which is added to the component **[a-20-b, b-22-c, b-24-e]**



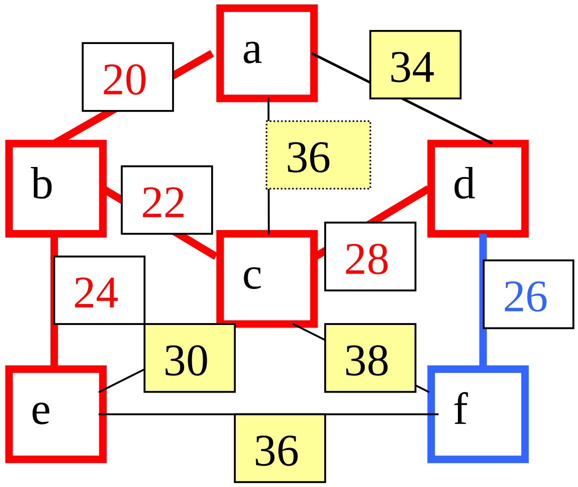
Now repeat the process from e to d, f

Minedge**: lowcost: 20 22 28 <u>24</u> 38 closest: a b c b c** U = {a,b,c,e} V-U = {d,f} min = **24**; k = **<u>e</u>**
j = d; if C[e,d] < lowcost[d] then { lowcost[d] = C[e,d]; closest[d] = e } → §<28 → no change
j = f; if C[e,f] < lowcost[f] then { lowcost[f] = C[e,f]; closest[f] = e } → 36<38 ➔ **e-36-f**



Which gives **20 22 <u>28</u> 24 36 closest: a b c b e**

The cheapest edge is **c-28-d** which is added to the component to give
[a-20-b, b-22-c, b-24-e, c-28-d]

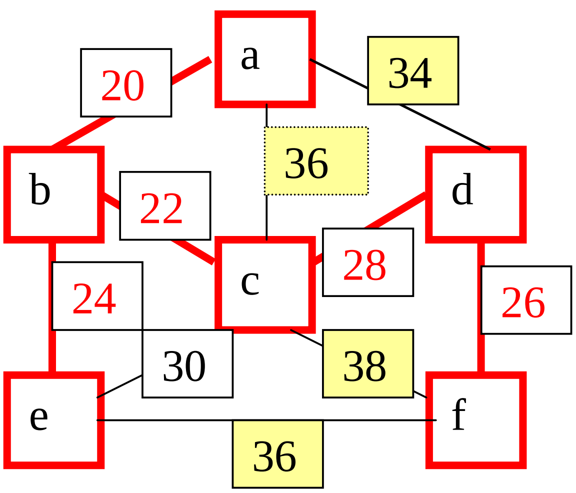


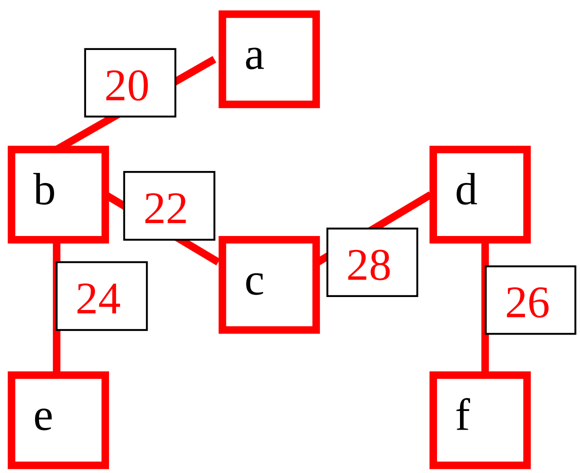
Now repeat the process from d to f

Minedge**: lowcost: 20 22 28 24 36 closest: a b c b e** U = {a,b,c,e,d} V-U = {f} min = **28**; k = **d**
j = f; if C[d,f] < lowcost[f] then { lowcost[f] = C[d,f]; closest[f] = d} → 26<36 → **d-26-f**

Now look for the closest edge from the component to the remaining node f which is **d-26-f.**
This gives [a-20-b, b-22-c, b-24-e, c-28-d, d-26-f]

Adding this to the component gives **lowcost: 20 22 28 24 26 --- closest: a b c b d - MST**





And now all nodes are part of the component. V –U ={ ¤ }