Comments on the exam and questions.

General comment – the exam was not well done at all. The usual faults were to blame

1. NOT READING THE QUESTION CAREFULLY TO START WITH - examples
   a. 1(a) - a recursive **definition** – **not function**!!!
   b. 1(d) – **tree** not binary tree!
   c. Principles – not what the algorithm does but the general ideas behind HOW it works
   d. 7 (a) **undirected graph** – this is stated in the question
2. Not knowing the subject
3. Lack of detail in the answer
4. Nor giving each step in a calculation
5. Prim – not using the algorithm – "by inspection" answers – no steps in the calculation
6. Time management during the exam – running out of time

Question 1: definitions/descriptions – short answers
Problems – not knowing the subject / not reading the question carefully

Question 2:
Problems – 2(a) - principles – not what the algorithm does but the general ideas behind HOW it works. 2(e) – you MUST be able to give these definitions – use the formal notation
S ::= H T | ¤; H ::= element; T ::= S; **¤ - empty**
BT ::= LC N RC | ¤; N ::= element; LC ::= BT; RC ::= BT; **¤ - empty**

Question 3:
Problems – lack of detail – see the facit answer below – collision handling H(key) + f(i) where f(i) should be defined for each case – linear probing, quadratic probing & double hashing

Question 4:
Problems – inability to construct an argument and a discussion question – there is an example on the web http://www.cs.kau.se/~donald/index.php?DSAexamdiss=1

Question 5:
Problems – this was the hardest question! The removal of a value from a BST

The given code (7 steps) is a series of tests to exclude certain conditions – you also need to read between the lines as to what these tests imply

1. The empty tree – you cannot delete a value which does not exist by definition!
2. The value is in the LC – reconstruct the tree from the LC with value removed, node and RC
3. The value is in the RC – reconstruct the tree from the LC, node and RC with value removed
4. The value to be removed is at the (local) root node – there are 4 possibilities here for the tree (¤ = empty) – N is the root node to be removed
   (i) (¤, N, ¤), (ii) (LC, N, ¤), (iii) (¤, N, RC) (iv) (LC, N, RC)
   Case 4 covers (i) and (iii) - i.e. LC is empty – return RC – either case (i) empty or case (iii) non-empty
5. Covers case (ii) – empty RC hence return the non-empty LC
6. Is case (iv) above – i.e. (LC, N, RC) - and decides on the tree balance whether to take the maximum value of the LC to replace the root node (N) to be removed ➔ LCMasAsRoot
7. Use the minimum value of the RC as the value to replace the root node N

In cases (6) & (7) LCMaxAsRoot/RCMinAsRoot both functions will create a new tree with maxLC / minRC removed from the appropriate tree (hence using b_rem(…) recursively) – i.e.

See the facit answer below.

Question 6:
Problems – 6(a) – not using an example; the simplest explanation uses the adjacency list – see the facit answer below.

Question 7:
Problems – Prim/Kruskal – undirected graph! It would have been better to do 7(c) – Kurskal – first since it takes less time and gives the answer to 7(a) which means that you can cross-check your calculations as you go.
Too many “by inspection” answers – no detailed, stepwise calculations for Prim!!!

FACIT TILL
OMTENTAMEN I
DATASTRUKTURER OCH ALGORITMER DVG B03

160402 kl. 09:00 – 14:00

Ansvarig Lärare: Donald F. Ross

Hjälpmedel: Inga. Algoritmerna finns i de respektive uppgifterna eller i bilogarna.

***** OBS *****

Betygsgräns:	Kurs:	Max 60p, Med beröm godkänd 50p, Icke utan beröm godkänd 40p, Godkänd 30p (varav minimum 20p från tentan, 10p från labbarna)
	Tenta:	Max 40p, Med beröm godkänd 34p, Icke utan beröm godkänd 27p, Godkänd 20p
	Labbarna:	Max 20p, Med beröm godkänd 18p, Icke utan beröm godkänd 14p, Godkänd 10p

SKRIV TYDLIGT – LÄS UPPGIFTERNA NOGGRANT

Ange alla antaganden.

---

(1) Ge ett kortfattat svar till följande uppgifter ((a)-(j)).

(a) Vad skulle ett alternativ för Floyds algoritm vara?

Apply Dijkstra's algorithm to each node in the graph.

(b) Vad är en rekursiv definition?

A definition which is PARTLY defined in terms of itself – e.g. sequence, BT

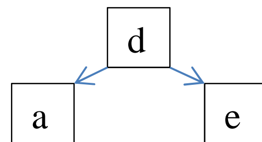
(c) Vad betyder "collection abstraction"?

Treating the set, sequence, tree and graph as a collection of entities and defining common operations as operations on a collection e.g. add, remove, find element, count the number of elements (cardinality), is_empty operation.

(d) Ge en definition av ett träd?

A collection of entities (nodes) defined by a parent/child relationship (hierarchical), where each node (except the leaves) may have a number of children and each node (except the root) has exactly 1 parent. If the collection of children is considered a set then the tree is unordered; if considered as a sequence, the tree is ordered.

(e) Ge ett exempel av **en allmän djupet-först** traversering av ett BST?



A general depth-first traversal of the above BST would visit each node 3 times and produce the following sequence: d_1 a_1 a_2 a_3 d_2 e_1 e_2 e_3 d_3 where the x_1 represent a pre-order traversal, the x_2 an in-order traversal and the x_3 a post-order traversal.

(f) Vad är meningen med sortering?

To make searching more efficient.

(g) Vad är hanterliga (”tractable”) problem i komplexitetsteori?

These are problems whose Big-oh solutions may be expressed as polynomial time. e.g. n, n^2, and even logn, n logn.

(h) Nämn två metoder för att upptäcka cykler i grafer.

Apply Warshall’s algorithm and check the diagonal (top-left to bottom right) for 1s Construct the depth-first spanning forest and check for back edges.

(i) Vad är ett minimal spanningträd (”minimal spanning tree”) (MST)?

A Free Tree derived from an undirected graph where for n nodes there are (n-1) edges which represent the cheapest way of connecting the nodes in a single component.

(j) Nämn två exempel av algoritmer där djupet-först-sökning tillämpas?

Topological sort and creating a depth-first spanning forest.

Totalt 5p

(2) Ge ett kortfattat svar till följande uppgifter ((a)-(e)).

(a) Beskriv **principen** bakom **Dijkstras** algoritmen.

2p

Marks for a good answer.

(b) Beskriv en **abstrakt data typ (ADT)** som en **virtuell maskin**. Vilka fördelar finns med en sådan beskrivning?

2p

Marks for a good answer.

(c) Skriv **abstrakt rekursiv pseudokod** till en funktion för att ombalansera ett AVL-träd. Ange alla antagande och förklara alla hjälpfunktioner som behövs.

2p

(1) Test the tree (T) to see whether a left or right rotation is required.
(2) Test the RC/LC to decide whether a single or double rotation is required.

NOTE: the is not the answer required but since this is part of the labs for the course, I do not intend to provide "free code". The assumption is that you have done the labs!

(d) Skriv **abstrakt rekursiv pseudokod** till funktionen **T2Q** från labb 1. **T2Q** förvandlar ett binärt träd till en array.

2p

NOTE: the is not the answer required but since this is part of the labs for the course, I do not intend to provide "free code". The assumption is that you have done the labs!

(e) Skriv den **rekursiva definitionen** av en sekvens och ett binärt-träd.

2p

Seq	::= Head Tail | empty	
Head	::= element	non-recursive part
Tail :	::= Seq	recursive part
BT	::= LC N RC | empty	
N	::= element (node)	non-recursive part
LC	::= BT	recursive part
RC	::= BT	recursive part

Totalt 10p

(3) Hashning

Diskutera ingående hur hashning fungerar. Vilket är det största problemet? Hur löser man detta? Vilka aspekter skulle man ta hänsyn till? Ge exempel för varje fall Du beskriver.

5p

Points for discussion

(1) Hash function H(key) ➔ index in the hash space.
(2) Mention the possibilities for the type of hash function.
(3) The main problem is collision handling – we looked at 4 methods
(4) Collisions are handled by **H(key) + f(i)** where i is the ith collision
(5) Linear probing **f(i) = i;** quadratic probing **f(i) = i*i;** double hashing **f(i) = i*H2(key)**
(6) Method 1 – separate chaining – describe & discuss the advantages and disadvantages.
(7) Method 2 – linear probing – describe & discuss the advantages and disadvantages.
(8) Method 3 – quadratic probing – describe & discuss the advantages and disadvantages.
(9) What are the main problems with methods 1 to 3?
(10) What is the main problem with method 3?
(11) Method 4 - double hashing – describe & discuss the advantages and disadvantages.
(12) Describe re-hashing – when is this used?
(13) Mention load factors
(14) Mention physical implementation factors – give examples.

+ marks for good examples and explanations.

(4) Diskussionsuppgift

Man kan påstå att sekvensen är den viktigaste abstrakt datastrukturen (ADT). Skriv en utförlig beskrivning av en sekvens (2p) och ett detaljerad diskussion som stödjar detta påstående (3p).
Svara ingående. Ange alla antagande.

Total 5p

Marks for a good discussion

Points to note

(1) A sequence is an ordered collection of non-unique entities i.e. each element has a position and value element.
(2) A sequence may be sorted but this is an extra property and not part of the definition.
(3) A sequence may be defined either iteratively or recursively – describe both.
(4) Describe the operations on a sequence – is_empty, cardinality, add, find & remove.
(5) A sequence may be used to implement a set and a graph as well as a set.

(5) Rekursion

Titta på koden för att ta bort ett värde från ett BST nedan.
Denna kod ska tillämpas på ett binärt sök träd som skapats genom att lägga till följande värden i denna ordning:-

10, 5, 30, 20, 15, 25, 13, 17, 16

Värdet 15 ska sedan tas bort.
Ange alla antagande.

1. Skriv (pseudo)koden till funktionen HDiff(treeref T)

   (1p)

2. Skriv (pseudo)koden till funktionerna LCmaxAsRoot(treeref T) samt RCminAsRoot(treeref T)

   (1p)

   Använd rekursion och samma programmeringsstil för Dina versioner av HDiff, LCmaxAsRoot och RCminAsRoot.

3. **Förklara stegvis och utförligt** hur koden, inklusiv Dina versioner av (i) HDiff (ii) LCmaxAsRoot och (iii) RCminAsRoot fungerar när den tilllämpas på detta träd.
   Visa varje rekursivt anrop till b_rem. Det första anropet är b_rem([10], 15) där [10] står för trädet med rotvärde 10. Dvs [x] står för trädet med rotvärde x.

   Rita trädet som returneras efter varje anrop i rader 2-7.

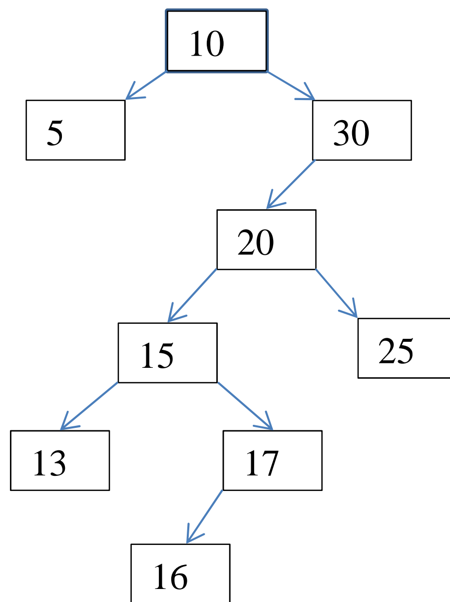
   (3p)
   Total 5p

```
static treeref b_rem(treeref T, int v)
{
    1.  return is_empty(T)          ? T
    2.  : v < get_value(node(T))    ? cons(b_rem(LC(T), v), node(T), RC(T))
    3.  : v > get_value(node(T))    ? cons(LC(T), node(T), b_rem(RC(T), v))
    4.  : is_empty(LC(T))           ? RC(T)
    5.  : is_empty(RC(T))           ? LC(T)
    6.  : HDiff(T) > 0              ? LCmaxAsRoot(T)
    7.  :                             RCminAsRoot(T);
}
```

```
1. static int HDiff(treeref T) { return b_height(LC(T)) - b_height(RC(T)); }

2. static treeref find_maxLC(treeref T) {
      return  is_empty(RC(T)) ? T : find_maxLC(RC(T));
      }

3. static treeref find_minRC(treeref T) {
      return  is_empty(LC(T)) ? T : find_minRC(LC(T));
     }

4. static treeref LCmaxAsRoot(treeref T) {
      treeref maxnode = find_maxLC(LC(T));
      return cons(b_rem(LC(T), get_value(maxnode)), maxnode, RC(T));
      }

5. static treeref RCminAsRoot(treeref T) {
      treeref minnode = find_minRC(RC(T));
     return cons(LC(T), minnode, b_rem(RC(T), get_value(minnode)));
     }
```

```
static treeref b_rem(treeref T, int v)
{
   1.  return is_empty(T)          ? T
   2.  : v < get_value(node(T))  ? cons(b_rem(LC(T), v), node(T), RC(T))
   3.  : v > get_value(node(T))  ? cons(LC(T), node(T), b_rem(RC(T), v))
   4.  : is_empty(LC(T))          ? RC(T)
   5.  : is_empty(RC(T))          ? LC(T)
   6.  : HDiff(T) > 0             ? LCmaxAsRoot(T)
   7.  :                            RCminAsRoot(T);
}
```

The start tree



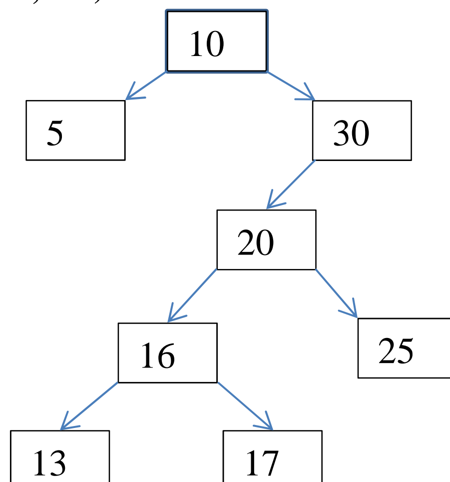
1. **Initial call b_rem([10], 15)**
2. **Line 3: recursive call 1 cons([5], [10], b_rem([30], 15))**
3. **Line 2: recursive call 2 cons(b_rem([20], 15), [30], [¤])**
4. **Line 2: recursive call 3 cons(b_rem([15], 15), [20], [25])**
5. **Line 6: HDiff([15] ➔ 1 – 2 ➔ -1 i.e. the RC is deeper**
6. **Line 8; RCminAtRoot([15])**
   a. **Calls find_minRC([15]) which returns [16]**
   b. **Recursive call 4 cons([13], [16], b_rem([17], 16);**

Turning point in the recursion

---

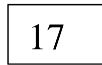
   c. **Line 2: recursive call 5 cons(b_rem([16], 16), [17], [¤])**
   d. **Line 4: returns [¤] to the cons above in (c)**
   e. **The cons in (c) returns [17] to the call in (b)**
   f. **The cons in (b) then returns the tree ([13], [16], [17]) to (6) above**
   g. **This cons returns (([13], [16], [17]), [20], [25]) from (4) to (3)**
   h. **(3) returns ((([13], [16], [17]), [20], [25])), [30], [¤]) to (2)**
   i. **(2) returns ([5], [10], ([13], [16], [17]), [20], [25])), [30], [¤])) to (1)**

(LC, N, RC) is used to denote trees with nested elements for LC and RC

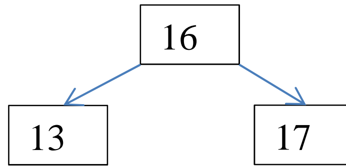


Pictures for the above:-

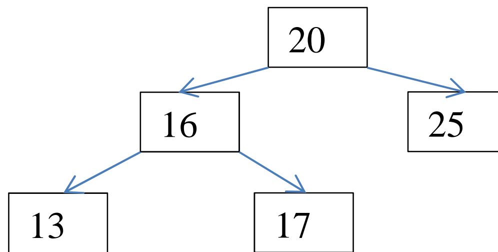
(e) returns



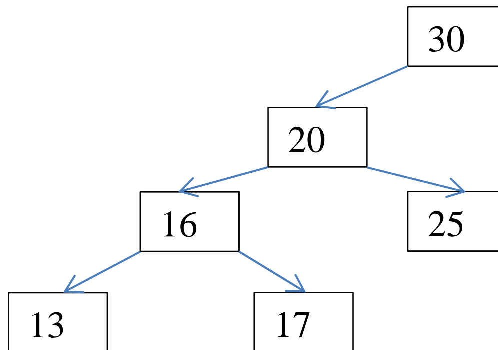
(f) returns



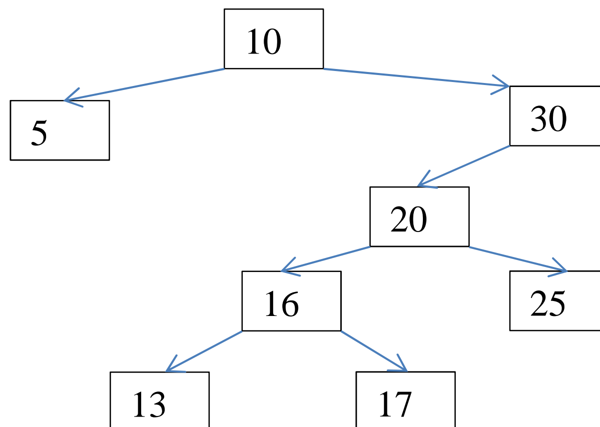
(g) returns



(h) returns



(i) returns



(6) 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

(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
        : yyy;
}
```

1p

(c) Skriv abstrakt rekursiv (pseudo)kod för att lägga till ett element i ett binärt träd.
Ange alla antagande.

2p

Totalt 5p

(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 graph, (ii) the graph 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)

Draw an example of a graph as well as the corresponding adjacency matrix
e.g. undirected graph **nodes: a, b, c edges (a,b), (a,c), (b,c) – this is G**

adjacency matrix	a: b, c
	b: a, c
	c: a, b
remove edge (a,b)	
result	a: c
	b: c
	c: a, b

now analyse the code

b_findn(cs, G)	find cs (source node) in the **node list**
get_edges(N)	get the **edge list** for node N
b_reme(e, L)	remove e from the **edge list** L
set_edges(N, L)	set the **edge list** of **node** N to **edge list** L
reme(cs, cd)	cs is the “source” **node** in the **node list** cd is the “destination” **node** in the **edge list**

for an undirected graph, the front end must issue 2 calls reme(a,b) and reme(b,a) AFTER having checked that nodes a and b exist and that edge (a,b) exists.

(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**

```
TreeRef: Add(TreeRef T, integer v) {
    if IsEmpty(T) then return create_el(v);
    if v < value(T) then return cons(Add(left(T), v), T, right(T));
    if v > value(T) then return cons(left(T), T,  Add(right(T), v));
    return T;    // no duplicates.
    }
```

(7) Prims

a) Tillämpa **Prims algoritm** (nedan) på den **oriktade** grafen:

(a-7-b, a-1-c, a-10-d, b-3-c, b-2-e, c-15-d, c-4-e, c-9-f, d-4-f, e-8-f).

Börja med nod a. Ange alla antaganden och visa alla beräkningar och mellanresultat. Vad representerar resultatet? **Ange varje steg i din beräkning.**

Rita delresultatet efter varje iteration.

(2p)

b) **Förklara principerna bakom Prims** algoritm.

(2p)

c) Visa hur Du skulle använda **principerna** bakom **Kruskals algoritm** för att bekräfta resultatet från Prims algoritm.

(1p)

Totalt 5p

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

```
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; }
    }
}
```

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 cheapest edge SO FAR found – this last proviso reveals that Prim's is a GREEDY algorithm i.e. used a local best solution.

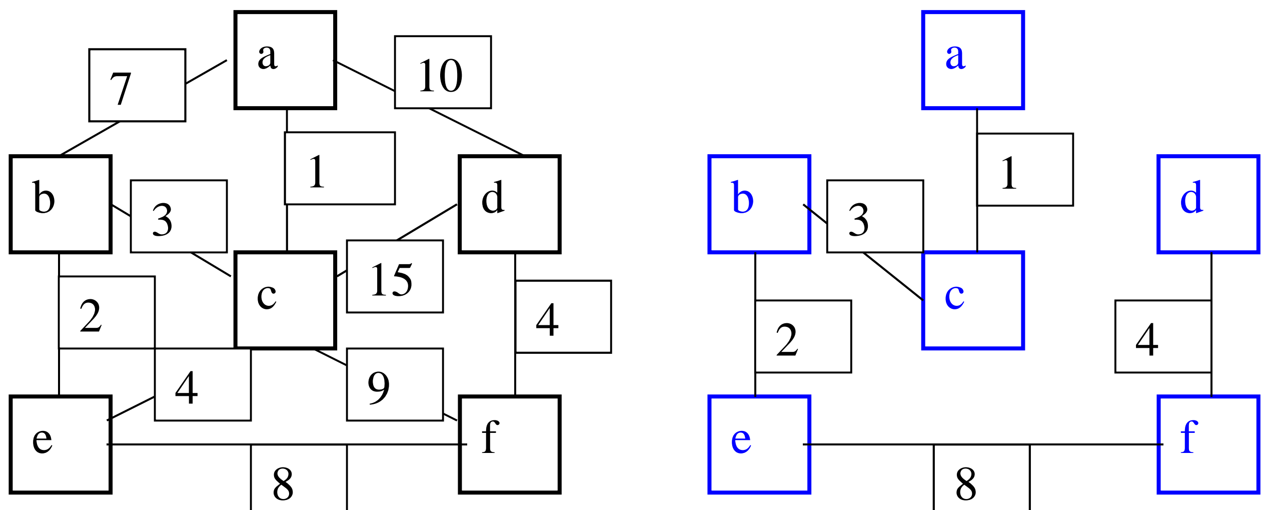
1. Choose a start node (usually a) and mark as visited U = {a} U is a component
2. Calculate the distances between a and the remaining nodes V-U and draw the MST calculated so far
3. Choose the closest node x to a and mark as visited U = {a,x}
4. If the distance x to any node y in V-U is shorter then readjust costs by removing the previous edge and adding the edge (x y) – now we have a new MST
5. U represents the visited nodes and V-U the unvisited nodes – choose the shortest edge from the visited nodes to a node in V-U and add this to the MST
6. Repeat 4 & 5 until the MST has been found and V-U = ¤ (empty)

Briefly, Prim's "grows" a component from a start node and the shortest edge from that start node. If there is a shorter edge from a node in the component (visited nodes) to a non-component node (non-visited nodes), replace the current edge with the new (shorter) edge.

See below for the calculations.

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

Cost 18



Draw the cost matrix C and array D

	a	b	c	d	e	f
a		**7**	**1**	**10**		
b	**7**		**3**		**2**	
c	**1**	**3**		**15**	**4**	**9**
d	**10**		**15**			**4**
e		**2**	**4**			**8**
f			**9**	**4**	**8**	

a	b	c	d	e	f
lowcost	**7**	**1**	**10**	§	§
closest	**a**	**a**	**a**	**a**	**a**

Minedge: **lowcost: 7 1 10 § § closest: a a a a a** U = {a,c} V-U = {c,b,d,e,f} min = **1**; k = **c**
Readjust costs: if C[k,j] < lowcost[j] then { lowcost[j] = C[k,j]; closest[j] = k }
j = b; if C[c,b] < lowcost[b] then { lowcost[b] = C[c,b]; closest[b] = c } → 3<7 → **c-3-b**
j = d; if C[c,d] < lowcost[d] then { lowcost[d] = C[c,d]; closest[d] = c } → 15<10 → no change
j = e; if C[c,e] < lowcost[e] then { lowcost[e] = C[c,e]; closest[e] = c } → 4<§ → **c-4-e**
j = f; if C[c,f] < lowcost[f] then { lowcost[f] = C[c,f]; closest[f] = c } → 9<§ → **c-9-e**

Minedge: **lowcost: 3 1 10 4 9 closest: c a a c c** U = {a,c,b} V-U = {d,e,f} min = **3**; k = **b**
Readjust costs: if C[k,j] < lowcost[j] then { lowcost[j] = C[k,j]; closest[j] = k }
j = d; if C[b,d] < lowcost[d] then { lowcost[d] = C[b,d]; closest[d] = b } → §<10 → no change
j = e; if C[b,e] < lowcost[e] then { lowcost[e] = C[b,e]; closest[e] = b} → 2<4 → **b-2-e**
j = f; if C[b,f] < lowcost[f] then { lowcost[f] = C[b,f]; closest[f] = b } → §<9 → no change

Minedge: **lowcost: 3 1 10 2 9 closest: c a a b c** U = {a,b,c,e} V-U = {d,f} min = **2**; k = **e**
j = d; if C[e,d] < lowcost[d] then { lowcost[d] = C[e,d]; closest[d] = e } → §<10 → no change
j = f; if C[e,f] < lowcost[f] then { lowcost[f] = C[e,f]; closest[f] = e } → 8<9 → **e-8-f**

Minedge: **lowcost: 3 1 10 2 8 closest: c a a b e** U = {a,b,c,e,d} V-U = {f} min = **8**; k = **f**
j = f; if C[d,f] < lowcost[f] then { lowcost[f] = C[d,f]; closest[f] = d} → 4<10 → **d-4-f**

Min edge: **lowcost: 3 1 4 2 8 --- closest: c a f b e ---** U = {a,c,b,d,f e} V-U = {¤}

QED ☺ MST edges **a-1-c, c-3-b, b-2-e, e-8-f, f-4-d** **Total cost = 18**

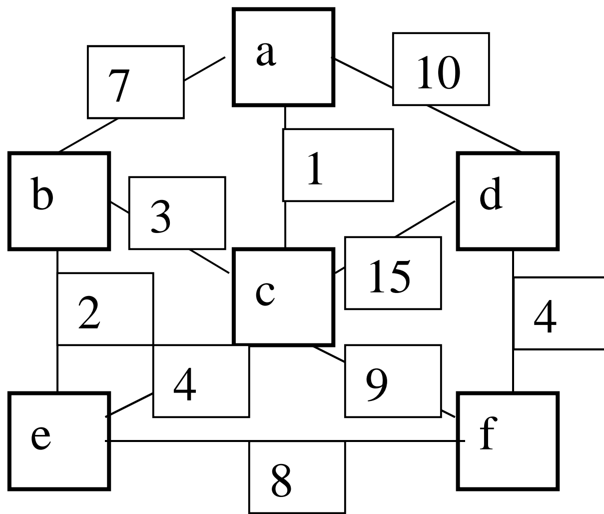
(Confirm using Kruskal's)

Förklara hur Prims algoritm fungerar genom att rita bilder som representerar varje mellanresultat under algoritmensexekvering. Använd exemplet ovan.
Vad är principen bakom Prims algoritm?

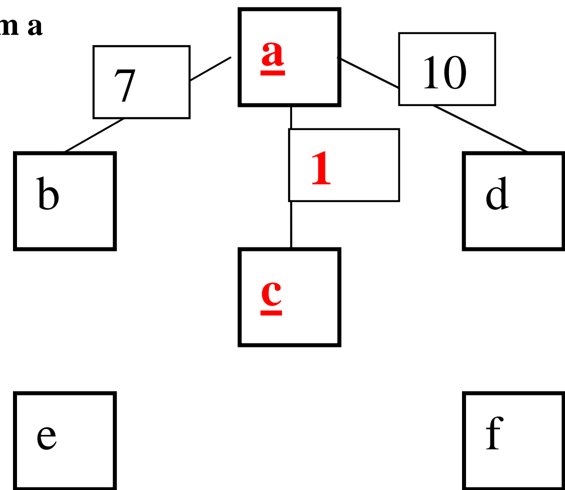
(2p)

See above for the principle.

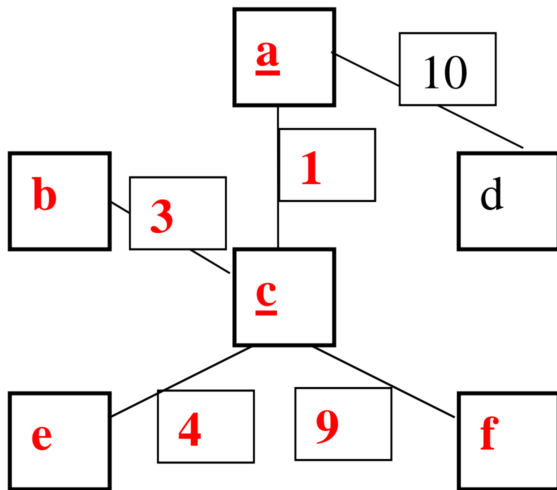
Diagrams:-



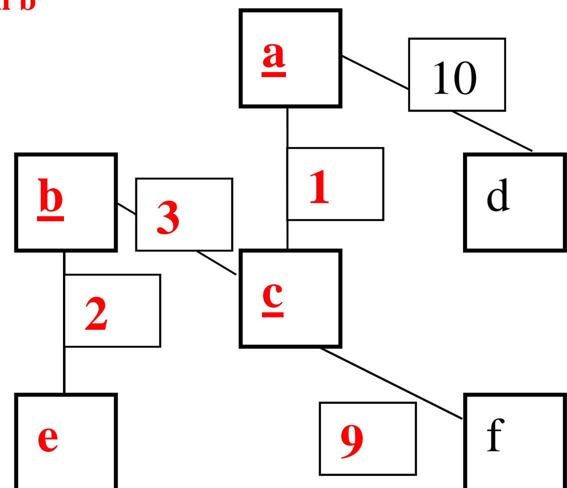
from a



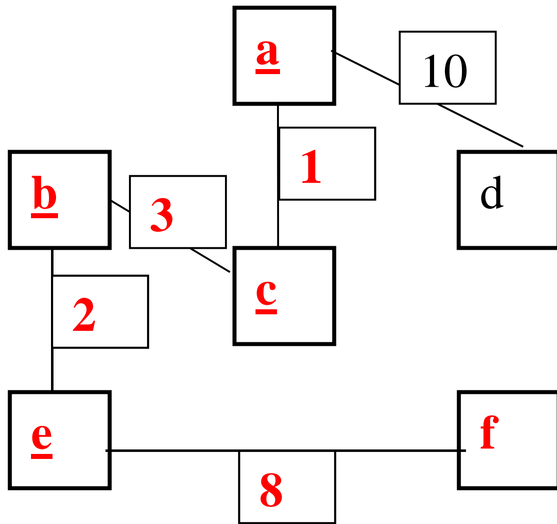
from c



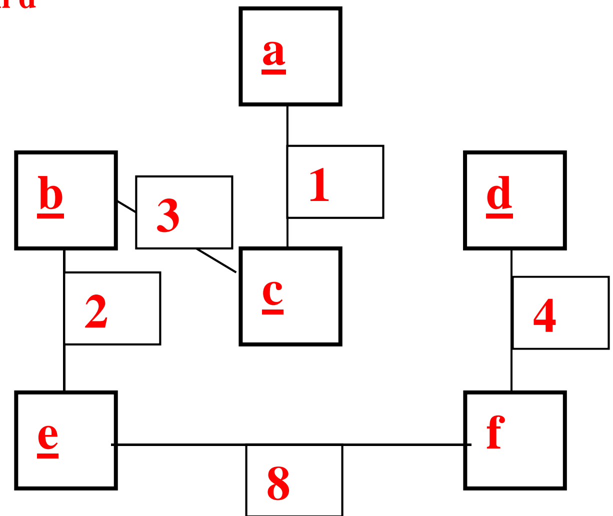
from b



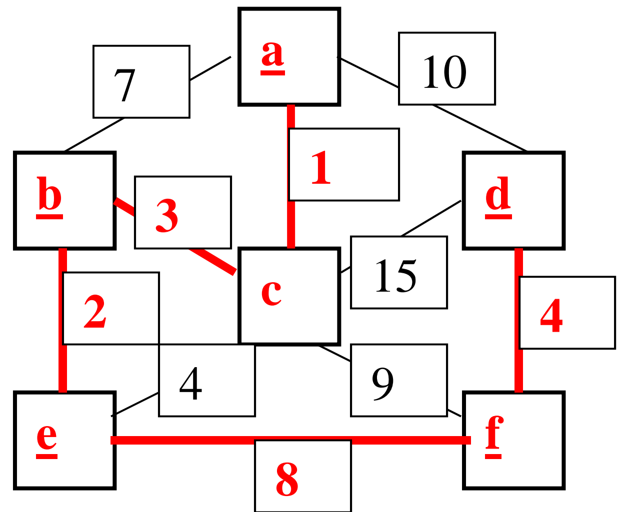
from e



from d



Check this result using Kruskall's



Priority Queue

a-1-c
b-2-e
b-3-c
c-4-e – would produce a cycle
d-4-f
a-7-b - would produce a cycle
e-8-f

the rest!