CORE EXAMINATION
Department of Computer Science
New York University
January 24, 1997

Programming Languages and Compilers

Question 1

Consider the following two versions of multiple assignments, for some hypothetical new languages:

```
L1 := L2 := L3 ... := Ln := E;         (maybe in Pascal2000)
L1, L2, ...  Ln := E1, E2, .. En;      (maybe in  Ada2000)
```

In the first case, all the L-values of L-expressions L1, L2.. and the R value of E are evaluated, and then all the L-values are updated.
In the second case, all the L-values of L-expressions L1, L2 .. Ln and then all the R-values of E1, E2 .. En are evaluated, in left-to-right order. Then all the L-values are updated in parallel with the corresponding R-values.

(a) Give BNF productions for these two assignment forms.

(b) Under the semantics described above, are the following statements equivalent:

```
a := b := c;
a, b := b, c;
```

Give some example to justify your answer.

Solution

(a) Let Lhs and Expr be the nonterminals for left-hand-side and expression respectively. The proposed multiple assignments can be described by the following productions:

```
Lhs_Chain ::=  Lhs   | Lsh ´:=´ Lhs_Chain
Lhs_List  ::=  Lhs   | Lhs  ´,´  Lhs_List
Expr_List ::=  Expr  | Expr ´,´  Expr_List

Multi1   ::= Lhs_Chain := Expr ´;´
Multi2   ::= Lhs_List  := Expr_List ´;´
```

(b) The semantics of the two forms are not equivalent. Consider the multiple assignment that exchanges two values, or assignments that involve indexed components where the index is itself modified:

```
    a := b := a;            vs.            a, b := b, a;
or
    a[b] := b := b + 1      vs.            a[b], b := b,  b + 1;
```

Question 2

The following Scheme function reduces the contents of a list of integers by applying a given function to successive members of the list:

```
(define (fold fun lis default)
   (cond ((null? lis) default)
         ((null? (cdr lis)) (fun default (car lis)))
         (ELSE  (fold fun (cdr lis) (fun default (car lis) )))))
```

(a) In Ada95 or C++ write a function that has the same purpose. How do you declare the parameter that corresponds to fun above?

(b) Write a call to the function of part a) that returns the product of the elements of a list.

Solution

(a) We can see from the code that *fun* takes two arguments. The problem discusses lists of integers, so we know that the arguments of *fun*,the result, and default, are all integers. In C++, we declare the functional parameter as a pointer to a function that returns an integer:

```
typedef int (*FPTR) (int, int);

int fold (FPTR fun, List lis, int default);
```

Assuming that List has the usual representation, the body of the function is almost identical to the Scheme version:

```
{
if (!lis) return default;
if (!lis-> Next)
   return fun (default, lis->value);
else
   return fold (fun, lis-> Next, fun (default, lis->value));
}
```

It is easy to rewrite this in iterative form.

```
{
result = default;
while (lis) {
   result = fun (result, lis->value);
   lis = lis -> Next;
};
return result;
}
```

In Ada95, we declare *fun* as an access to a subprogram:
type Fptr is access function (X, Y : Integer) return Integer;
and the body of the function is almost identical. An alternate approach is to define *fold* as a generic procedure with a formal subprogram parameter:

```
generic
   with function fun (X, Y : Integer) return Integer;
function fold (lis : List; Default : Integer);
```

In that case we supply an explicit *fun* for each instance of *fold.*

(b)
```
int times (int x, int y) { return x * y};
 result = fold (&times, lis, 1);
```

Question 3

(a) Briefly explain the difference between overloading and dynamic dispatching.

(b) In the object-oriented language of your choice, define a class Par, a class derived from Par, and write an example of an overloaded call and of a dispatching call.

(c) The keyword **virtual** does not exist in Java, while it is central to dynamic dispatching in C++. Briefly explain why.

Solution

(a) Overloading is a syntactic facility, available in most programming languages, that allows the declaration of multiple subprograms with the same name, so that several of them are visible at the same program point. These subprograms must have different signatures (different parameter types, or different return type). A call that uses an overloaded name is legal if it is possible to determine from the types of the actuals which declaration is meant. Arithmetic operators are overloaded in all programming languages. Ada, C++, and Java allow the user to overload subprograms. Ada and C++ (but not Java) allow the user to overload operators as well. Overloading is resolved at compile-time.

Dynamic dispatching is the run-time mechanism that invokes the proper implementation of an operation that is applied to a polymorphic object. Such an object designates at run-time a value that can belong to several classes. These clases are related by inheritance. The name of the subprogram in a dispatching operation also has multiple meanings, but it is not possible to determine statically which one is meant, because one of the parameters has no single compile-time meaning. Therefore the choice of operation is made dynamically, by means of the dispatch tables of the various classes involved.

(b) a typical use of overloading in C++ or Java is in the definition of constructors.

```
class Par {
   Par ();                   //  default constructor.
   Par (int p1 , int p2);    //  make a Par with two integers.
   virtual void f(int x);    //  a dispatching operation.
};

class Child : public Par {
    Child ();
    virtual void f (int x); //  override inherited operation.
 };

 Par a =  Par(10, 10);      //  call second constructor.
                            //  more commonly written a(10, 10)

 Par *ptr;                  //  can be pointer to Par or Child.
 ptr = ...
 ptr -> f(10);              //  dispatching call.
```

(c) For efficiency reasons, C++ makes a distinction between operations that are dispatching and those that are resolved statically. Only functions labelled *virtual* can be dispatching, and only operations whose receiving object is a polymorphic pointer are dispatched. Java does not make any of those distinctions: all class operations are dispatching, and there are no pointers. Instead, every object has reference semantics, and is polymorphic (in other words, a variable of class A can denote a value of any class derived from A). If all operations dispatch, they don't need a special keyword!

Question 1

Consider the following resource-allocation policy. Requests and releases for resources are allowed at any time. If a request for resources cannot be satisfied because the resources are not available, then we check all processes that are blocked waiting for resources. If they currently hold some of the desired resources, then these resources are taken away from them and given to the requesting process. The vector of resources for which the process is waiting is increased to include the resources that were taken away.

For example, consider a system with three resource types and the vector *Available* initialized to (4, 2, 2). If process P_0 asks for (2, 2, 1) it gets them, If P_1 asks for (1, 0, 1) it gets them, Then if P_0 asks for (0, 0, 1) it is blocked (resource not available). If P_1 now asks for (2, 0, 0) it gets the available one (1, 0, 0) and one that was allocated to P_0, since P_0 is blocked. P_0's allocation vector becomes (1, 2, 1) and its *Need* vector becomes (1, 0, 1).

(a) Can deadlock occur in such a system? Is so, give an example. If not, which condition necessary for deadlock cannot occur?

(b) Can indefinite blocking occur? If so, give an example.

(c) Briefly, can you suggest an allocation strategy to avoid indefinite blocking?

Solution

(a) Deadlock cannot occur because preemption exists.

(b) Yes, a process may never acquire all the resources it needs if they are continuously preempted by a series of requests by other processes. In the example below, P_0 will be blocked as long as P_1 and P_2 continue to hold the resources necessary to satify's P_0's request.

Process	Allocation	Need	Available	Result
...				
P_0	2,2,1	0,0,1	1,0,0	Blocked
P_2	0,0,0	2,0,0	1,0,0	Satisfied
P_0	1,2,1	1,0,1	1,0,0	Blocked
P_1	1,0,1	1,0,0	0,0,0	Satisfied
P_0	0,2,1	2,0,1	0,0,0	Blocked
P_2	2,0,0	0,1,0	0,0,0	Satisfied
P_0	0,1,1	2,1,1	0,0,0	Blocked
...				

Question 2

Suppose that the head of a moving-head disk with 200 tracks, numbered 0 to 199, is currently serving a request at track 143 and just finished a request at track 125. The queue of requests is kept in FIFO order:

$$86, 147, 91, 177, 94, 150, 102, 175, 130.$$

What is the total number of head movements (i,e, tracks traversed) needed to satisfy all the pending requests, for each of the following disk scheduling algorithms? Show the path that the head follows in each case:

(a) FCFS

(b) SSTF

(c) SCAN

(d) LOOK

Solution

(a) FCFS: 565.
$(143 \rightarrow 86 \rightarrow 147 \rightarrow 91 \rightarrow 177 \rightarrow 94 \rightarrow 150 \rightarrow 102 \rightarrow 175 \rightarrow 130)$

(b) SSTF: 162.
$143 \rightarrow 147 \rightarrow 150 \rightarrow 130 irighrarrow 102 \rightarrow 94 \rightarrow 91 \rightarrow 86 \rightarrow 175 \rightarrow 177)$

(c) SCAN: 169.
$143 \rightarrow 147 \rightarrow 150 \rightarrow 175 \rightarrow 177 \rightarrow 199 \rightarrow 130 \rightarrow 102 \rightarrow 94 \rightarrow 91 \rightarrow 86)$

(d) LOOK: 125.
$143 \rightarrow 147 \rightarrow 150 \rightarrow 175 \rightarrow 177 \rightarrow 130 \rightarrow 102 \rightarrow 94 \rightarrow 91 \rightarrow 86)$

Algorithms

Question 1

Consider 3 sequences, each containing n integers,

$$a_1, a_2, ..., a_n$$

$$b_1, b_2, ..., b_n$$

$$c_1, c_2, ..., c_n$$

and assume they are all sorted in increasing order. Next consider the sequences of n^3 integers of the form

$$d_{ijk} = a_i \times b_j + c_k.$$

Consider the two questions:

(i) Are there both positive and negative integers among the d_{ijk}?

(ii) How many of the $d_{ijk} = 0$?

Clearly these questions can be answered using $O(n^3)$ work.

(a) Design an $O(1)$ algorithm to answer question (i).

(b) Design an $O(n^2 \log(n))$ algorithm or better to answer Question (ii).

(c) How well can you do if the initial sequences are not sorted?

Solution

(a) The crucial observation is that the largest and smallest of the d_{ijk} can occur only for $i = 1$ or n, $j = 1$ or n, and $k = 1$ or n. We can therefore decide by computing d_{111}, d_{11n}, d_{1n1}, d_{1nn},d_{n11},d_{n1n},d_{nn1} and d_{nnn} and look at their signs. This is clearly $O(1)$.

(b) Since $c_1, c_2, ..., c_n$ is sorted, we can decide if a number α is equal to $-c_i$ for any i using $O(\log n)$ work. Form all the n^2 products $a_i \times b_j$ and test one after the other. This requires only $n^2 \log(n)$ work.

(c) (i) In order to use the method described in the solution to part (a), we need only to find the largest and smallest elements in the three sequences. This can be done by inspecting the $3n$ elements once. Therefore $O(n)$ work is enough. (ii) The method described in the solution of (b) works if the third sequence has been sorted. Therefore, only $O(n \log n)$ extra work is needed.

Question 2

Let G be a connected undirected graph represented by an adjacency list.

(a) Find an algorithm that determines if G is a tree. State clearly what properties of a tree your algorithm is based on.

(b) What is the running time of your algorithm?

Solution

If G is a connected undirected graph, then if it has no cycles G is a tree. Recall that in an adjacency list representation of a graph, each edge (u, v) appears twice (v is on the adjacency list for u, and vice versa).

A depth first search algorithm can be used to detect cycles. Starting at any node, perform a depth first search, marking the nodes as they are visited. If during the search, an edge is checked that leads to a node that is already visited and that is NOT your parent, you have a cycle. The cost of the depth first search is $O(V + E)$, where V is the number of vertices and E is number of edges in the graph.

Question 3

We consider minimum spanning trees (MSTs) in an undirected graph $G = (V, E)$ whose edges have weights that are assigned in a special way. Specifically, we first give each vertex $v \in V$ a numerical value $W(v) \geq 0$. The **weight** $W(u, v)$ of an edge $(u, v) \in E$ is then defined to be $W(u) + W(v)$.

(a) Let G be the graph G_8 in figure 1:

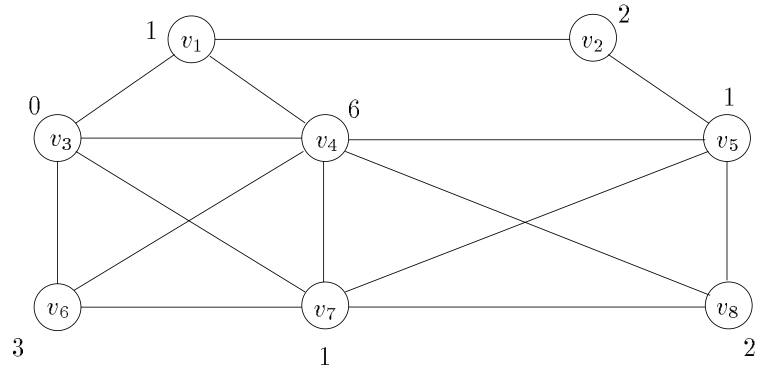

Figure 1: The graph G_8.

In this figure, the value $W(v)$ is written next to node v. For instance, $W(v_4) = 6$ and $W(v_1, v_4) = 1 + 6 = 7$. Compute an MST of G_8. Organize your computation systematically, so that we can verify your intermediate results. You must state the cost of the computed MST.

(b) Suppose G is the complete bipartite graph $G_{m,n}$ whose edges are given weights by the same scheme as above. Recall that in $G_{m,n}$, the vertices V are partitioned into two subsets V_0 and V_1 where $|V_0| = m$ and $|V_1| = n$ and $E = V_0 \times V_1$. See figure 2 for the case $m = 3, n = 2$. Give a simple description of an MST of $G_{m,n}$. State the weight of the MST. Argue that your description is indeed an MST.
(HINT: Try assigning distinct weights to vertices in $G_{3,2}$ and compute the MST. Does this suggest a pattern?)

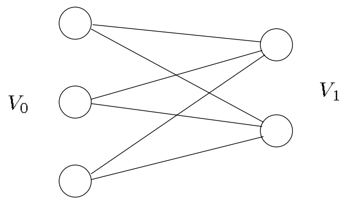

Figure 2: The graph $G_{3,2}$.

Solution

We use Kruskal's algorithm, consider the edges of G_{12} in order of non-decreasing weights. Each considered edges is accepted ($\surd$) or rejected ($\times$):

$$\begin{array}{l} v_1v_3(\surd), v_3v_7(\surd), v_5v_7(\surd), v_1v_2(\surd), v_3v_6(\surd), v_2v_5(\times), \\ v_5v_8(\surd), v_7v_8(\times), v_6v_7(\times), v_3v_4(\surd) \end{array}$$

We can stop after accepting 7 edges. The cost is of the MST is 19.

Let $v_i \in V_i$ $(i = 0, 1)$ have the smallest value $W(v_i)$ among the vertices in V_i. Let $W(v_0) = a$ and $W(v_1) = b$. Then an MST of $G_{m,n}$ is comprised of all the edges incident on v_0 and v_1. This MST has weight

$$\sum_{v \in V} W(v) + (m-1)b + (n-1)a. \qquad (1)$$

To prove that no other spanning tree has smaller weight, begin with any MST (call it T).
(i) We first transform T so that it contains the edge (v_0, v_1). Clearly, T contains a path from v_0 to v_1. Let this path be $(v_0, u_1, u_2, \ldots, u_k, v_1)$. If $k = 0$, we are done. If $k \geq 1$, we can replace the edge (u_k, v_1) with the edge (v_0, v_1). It is not hard to see (by the quotable fact) that the result is still an MST.
(ii) Next, take any vertex $v \in V_1$. If v is not directly connected to v_0, then the edge (v, v_0) added to T creates a unique cycle (quotable fact). This cycle contains another edge (v, u) where $u \in V_0$. Replacing (v, u) by (v, v_0) in T still gives an MST. Repeating this process, we finally transform T into the MST that we desire.

Algorithms

Question 1

Consider 3 sequences, each containing n integers,

$$a_1, a_2, ..., a_n$$

$$b_1, b_2, ..., b_n$$

$$c_1, c_2, ..., c_n$$

and assume they are all sorted in increasing order. Next consider the sequences of n^3 integers of the form

$$d_{ijk} = a_i \times b_j + c_k.$$

Consider the two questions:

(i) Are there both positive and negative integers among the d_{ijk}?

(ii) How many of the $d_{ijk} = 0$?

Clearly these questions can be answered using $O(n^3)$ work.

(a) Design an $O(1)$ algorithm to answer question (i).

(b) Design an $O(n^2 \log(n))$ algorithm or better to answer Question (ii).

(c) How well can you do if the initial sequences are not sorted?

Solution

(a) The crucial observation is that the largest and smallest of the d_{ijk} can occur only for $i = 1$ or n, $j = 1$ or n, and $k = 1$ or n. We can therefore decide by computing d_{111}, d_{11n}, d_{1n1}, d_{1nn},d_{n11},d_{n1n},d_{nn1} and d_{nnn} and look at their signs. This is clearly $O(1)$.

(b) Since $c_1, c_2, ..., c_n$ is sorted, we can decide if a number α is equal to $-c_i$ for any i using $O(\log n)$ work. Form all the n^2 products $a_i \times b_j$ and test one after the other. This requires only $n^2 \log(n)$ work.

(c) (i) In order to use the method described in the solution to part (a), we need only to find the largest and smallest elements in the three sequences. This can be done by inspecting the $3n$ elements once. Therefore $O(n)$ work is enough. (ii) The method described in the solution of (b) works if the third sequence has been sorted. Therefore, only $O(n \log n)$ extra work is needed.

Question 2

Let G be a connected undirected graph represented by an adjacency list.

(a) Find an algorithm that determines if G is a tree. State clearly what properties of a tree your algorithm is based on.

(b) What is the running time of your algorithm?

Solution

If G is a connected undirected graph, then if it has no cycles G is a tree. Recall that in an adjacency list representation of a graph, each edge (u, v) appears twice (v is on the adjacency list for u, and vice versa).

A depth first search algorithm can be used to detect cycles. Starting at any node, perform a depth first search, marking the nodes as they are visited. If during the search, an edge is checked that leads to a node that is already visited and that is NOT your parent, you have a cycle. The cost of the depth first search is $O(V + E)$, where V is the number of vertices and E is number of edges in the graph.

Question 3

We consider minimum spanning trees (MSTs) in an undirected graph $G = (V, E)$ whose edges have weights that are assigned in a special way. Specifically, we first give each vertex $v \in V$ a numerical value $W(v) \geq 0$. The **weight** $W(u, v)$ of an edge $(u, v) \in E$ is then defined to be $W(u) + W(v)$.

(a) Let G be the graph G_8 in figure 1:

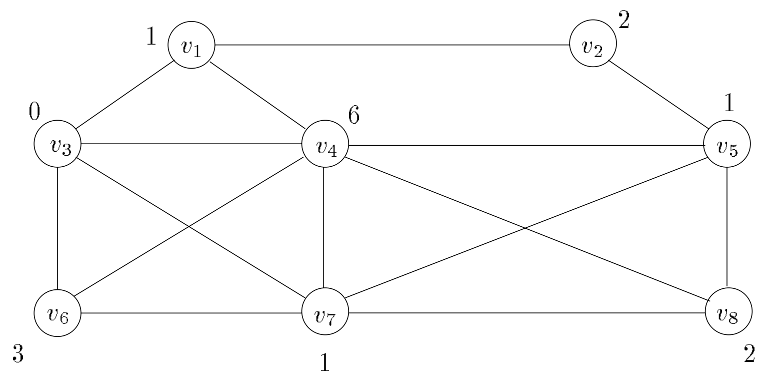

Figure 1: The graph G_8.

In this figure, the value $W(v)$ is written next to node v. For instance, $W(v_4) = 6$ and $W(v_1, v_4) = 1 + 6 = 7$. Compute an MST of G_8. Organize your computation systematically, so that we can verify your intermediate results. You must state the cost of the computed MST.

(b) Suppose G is the complete bipartite graph $G_{m,n}$ whose edges are given weights by the same scheme as above. Recall that in $G_{m,n}$, the vertices V are partitioned into two subsets V_0 and V_1 where $|V_0| = m$ and $|V_1| = n$ and $E = V_0 \times V_1$. See figure 2 for the case $m = 3, n = 2$. Give a simple description of an MST of $G_{m,n}$. State the weight of the MST. Argue that your description is indeed an MST.
(HINT: Try assigning distinct weights to vertices in $G_{3,2}$ and compute the MST. Does this suggest a pattern?)

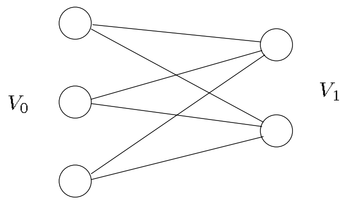

Figure 2: The graph $G_{3,2}$.

Solution

We use Kruskal's algorithm, consider the edges of G_{12} in order of non-decreasing weights. Each considered edges is accepted ($\surd$) or rejected ($\times$):

$$v_1v_3(\surd), v_3v_7(\surd), v_5v_7(\surd), v_1v_2(\surd), v_3v_6(\surd), v_2v_5(\times),$$
$$v_5v_8(\surd), v_7v_8(\times), v_6v_7(\times), v_3v_4(\surd)$$

We can stop after accepting 7 edges. The cost is of the MST is 19.

Let $v_i \in V_i$ $(i = 0, 1)$ have the smallest value $W(v_i)$ among the vertices in V_i. Let $W(v_0) = a$ and $W(v_1) = b$. Then an MST of $G_{m,n}$ is comprised of all the edges incident on v_0 and v_1. This MST has weight

$$\sum_{v \in V} W(v) + (m-1)b + (n-1)a. \tag{1}$$

To prove that no other spanning tree has smaller weight, begin with any MST (call it T).
(i) We first transform T so that it contains the edge (v_0, v_1). Clearly, T contains a path from v_0 to v_1. Let this path be $(v_0, u_1, u_2, \ldots, u_k, v_1)$. If $k = 0$, we are done. If $k \geq 1$, we can replace the edge (u_k, v_1) with the edge (v_0, v_1). It is not hard to see (by the quotable fact) that the result is still an MST.
(ii) Next, take any vertex $v \in V_1$. If v is not directly connected to v_0, then the edge (v, v_0) added to T creates a unique cycle (quotable fact). This cycle contains another edge (v, u) where $u \in V_0$. Replacing (v, u) by (v, v_0) in T still gives an MST. Repeating this process, we finally transform T into the MST that we desire.