Quesito 1. La seguente soluzione del problema dei lettori-scrittori (ove un'area dati condivisa può essere acceduta in modo simultaneo e concorrente da più processi con diritti di lettura ed in modo individuale ed esclusivo da processi con diritti di scrittura) contiene un errore importante. Si individui l'errore, spiegando quale situazione potrebbe verificarsi nell'esecuzione concorrente di processi così congegnati, e proponendone una versione corretta, realizzata apportando il minor numero possibile di modifiche all'originale.

```
void Lettore (void) {
  while (true) {
    numeroLettori++;
    if (numeroLettori == 1)
      P(dati);
    // leggi il dato
    P(mutex);
    numeroLettori--;
    if (numeroLettori == 0)
      V(dati);
    V(mutex);
    // usa il dato letto
  }
}
```

```
void Scrittore (void) {
  while (true) {
    // prepara il dato da scrivere
    P(dati);
    // scrivi il dato
    V(dati);
  }
}
```

Figura 1: Soluzione errata del problema lettori-scrittori.

Quesito 2. Si consideri il seguente programma concorrente codificato in Java:

```
public class Figlio implements Runnable {
  int indice;
  Thread t;
  private volatile boolean esegui = true;
  public Figlio(int prio, String nome, int val) {
    t = new Thread(this, nome);
    t.setPriority(prio); // in ordine crescente
    indice = val;
  }
  public void run() {
    while (esegui) {
    /* blocco F1:  esecuzione 10 unita' di tempo */
      indice++;
    }
  }
  public void start() { t.start(); }
  public void stop() { esegui = false; }
}
```

```
class Padre {
  public static void main(String args[]) {
  /* blocco P1:  esecuzione istantanea */
    final long sonnoPadre = 30; // in unita' di tempo
    final int val = 1;
    Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
    Figlio A = new Figlio(1, ``A'', val);
    Figlio B = new Figlio(2, ``B'', val);
    Figlio C = new Figlio(3, ``C'', val);
    A.start(); B.start(); C.start();
    for (int i=0; i<2; i++) {
    /* blocco P2:  esecuzione 10 unita' di tempo */
      System.out.print(A.indice + B.indice + C.indice);
      try { Thread.sleep(sonnoPadre); }
      catch (InterruptedException e) {}
    }
  /* blocco P3:  esecuzione 10 unita' di tempo */
    A.stop(); B.stop(); C.stop();
    try { A.t.join(); B.t.join(); C.t.join(); }
    catch (InterruptedException e) {}
  }
}
```

Figura 2: Un programma concorrente codificato in Java.

Assumendo che:

- i tempi d'esecuzione di ciascun blocco significativo dei metodi principali delle classi `Padre` e `Figlio` (identificati come P1-3, F1) siano quelli indicati in figura 2
- la durata della sospensione del processo `Padre`, invocata tramite `Thread.sleep()`, sia espressa in unità di tempo
- la terminazione dei processi figli conseguente alla negazione della condizione `esegui` sia istantanea

si specifichi: (1) il valore assunto da ciascuna istanza della variabile `indice` alla fine dell'esecuzione del programma e (2) il tempo di attesa cumulato dal processo `Padre` al suo completamento (espresso in quanti od ampiezze equivalenti), ove la politica di ordinamento del sistema operativo sottostante fosse rispettivamente di tipo:

1. *Round-Robin* con quanto ampio 10 unità di tempo, nei sottocasi:
   (a) senza priorità e con prerilascio
   (b) con priorità e con prerilascio
2. PD (*priority dispatching*), strettamente su base prioritaria e con prerilascio

Si assuma inoltre che, in caso di arrivo simultaneo in coda dei pronti di un processo risvegliato e del processo che abbia appena completato il proprio quanto, la politica 1.(a) attribuisca precedenza al primo sul secondo.

Quesito 3. Un *router* utilizza l'algoritmo detto del "*Token Bucket*" per adattare il profilo del proprio flusso di uscita alle possibili irregolarità del flusso in ingresso. L'algoritmo in questione produce un gettone (*token*) ad ogni intervallo temporale di ampiezza ΔT prefissata: ogni segmento in ingresso al *router* deve impossessarsi di un gettone per poter guadagnare l'uscita, e lo distrugge con l'uso; in caso di mancanza temporanea di gettoni, il segmento dovrà attendere in coda fino alla loro prossima generazione. La coda naturalmente ha capacità finita prefissata, ciò comportando la perdita di segmenti in caso di congestione.
Assumendo un intervallo di generazione di gettoni di ampiezza $\Delta T = \frac{1}{3}$ unità di tempo, si descrivano lo stato della coda (inizialmente vuota e di ampiezza pari a 12 segmenti) ed il profilo di uscita di tale *router* in presenza del seguente flusso di ingresso:

flusso in ingresso (i)	istante di arrivo (T_i)	ampiezza in segmenti (S_i)
1	5	4
2	10	2
3	15	6
4	20	9

Si discuta poi il comportamento dell'algoritmo, spiegando il motivo per il quale esso non aderisce pienamente alle attese e delineandone una correzione dei parametri che risolva il problema.

Quesito 4. Si consideri l'interconnessione di *subnet* mostrata in figura 3, ove ciascun nodo è un *router*, punto d'ingresso verso una specifica rete locale, mentre gli archi sono i collegamenti punto a punto tra i *router*.

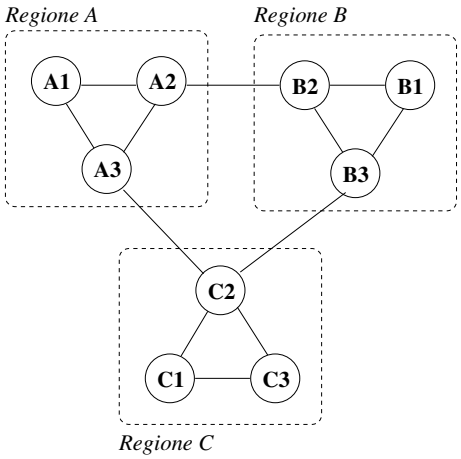

Figura 3: Topologia di interconnessione dei *router*.

I progettisti di tale rete di interconnessione desiderano scegliere la modalità di instradamento che riduca gli oneri di gestione delle relative tabelle da mantenere presso i *router*. Vogliamo aiutare questi progettisti raffrontando l'indirizzamento lineare (in cui ogni nodo della rete conosce il primo passo del percorso minimo verso qualunque altro nodo, e conosce anche il numero di passi complessivi che tale cammino richiede) con l'indirizzamento gerarchico (in cui ogni nodo conosce la topologia della propria regione di appartenenza — pertanto adottando in essa l'indirizzamento linerare — e conosce anche il cammino fino al punto d'ingresso verso le altre regioni, delle quali però non conosce la topologia).
A tal fine, si costruiscano, raffrontandole per ampiezza e complessità gestionale, le tabelle di instradamento necessarie al nodo **A1** per supportare ciascuna delle due forme di indirizzamento.

Quesito 5. Lo schema logico riportato in figura 4 rappresenta la rete dati di una piccola Azienda composta da 2 reparti operativi ed 1 stanza per i gestori della rete. Il reparto amministrazione è composto da 10 postazioni di lavoro, mentre il reparto produzione è suddiviso in 2 aree, ciascuna ospitante 8 postazioni di lavoro.

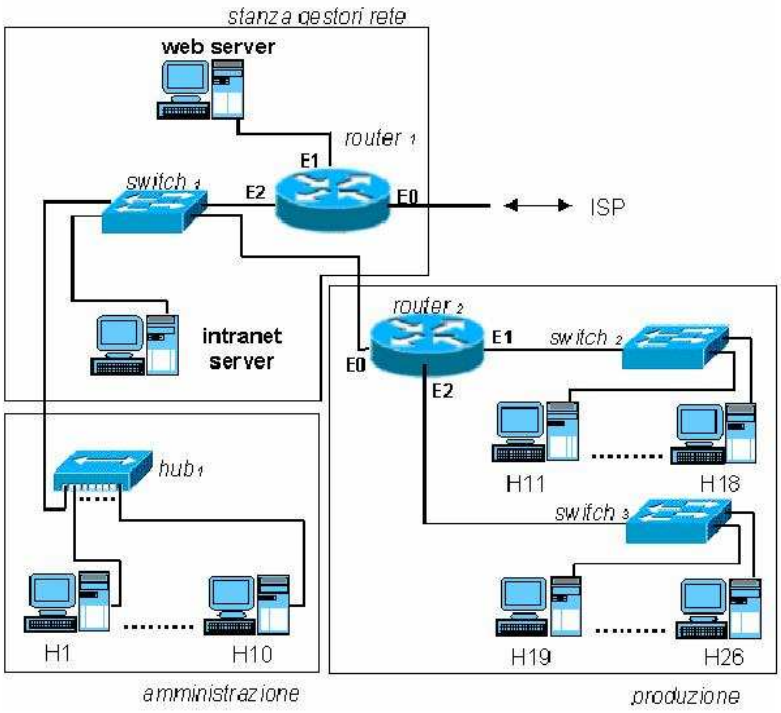

Figura 4: Schema logico della rete dati dell'Azienda.

Nella rete aziendale sono inoltre presenti 2 serventi posizionati nella stanza gestori rete: uno dei quali (detto web server) offre servizi Web destinati prevalentemente ad utenti esterni all'Azienda, mentre l'altro (denominato intranet server) offre servizi di *file server*, *application server* ed accreditamento a tutta l'Azienda.
L'Azienda accede ad *Internet* mediante un unico indirizzo IP statico fornito direttamente dal proprio ISP. Al suo interno, invece, decide di condividere gli indirizzi privati di una intera classe C (192.168.1.0), sfruttando la funzione di traduzione degli indirizzi (NAT) realizzata all'interno del router 1.
Si proponga una suddivisione degli indirizzi utili in sottoreti di uguale ampiezza (ossia dotate della medesima *subnet mask*) e della massima dimensione possibile, compilando una tabella che contenga, per tutti i dispositivi di rete, l'indirizzo IP associato, la *subnet mask* ed il *default gateway*.

Soluzione 1 (punti 5). La soluzione proposta è quella classica, comprendente:

- un semaforo binario `mutex` che regolamenta l'accesso alla variabile condivisa `numeroLettori`, tramite la quale i lettori determinano chi tra essi sia il primo ad accedere all'area dati e chi l'ultimo a rilasciarla, con ciò consentendo più letture simultanee (per esercizio si valuti se tale funzionamento potrebbe essere reso tramite l'uso di un semaforo contatore)
- un semaforo binario `dati`, che disciplina l'accesso all'area dati, preservando la mutua esclusione tra i lettori e lo scrittore.

La codifica proposta per la funzione `Scrittore()` è appropriata, in quanto la sua sezione critica è correttamente racchiusa dalla chiamata alle funzioni `P()` e `V()` sul semaforo `dati`.
La funzione `Lettore()` accede invece due volte in successione alla variabile condivisa `numeroLettori`: la prima volta per incrementarla, prima dell'accesso all'area dati, e, successivamente, per decrementarla al termine della lettura dei dati. Tuttavia, mentre per il decremento la corrispondente sezione critica è regolamentata correttamente da una coppia di `P()` e `V()` sull'apposito semaforo `mutex`, per l'incremento tale incapsulazione non è stata prevista, così che questa azione risulta essere priva di garanzie di mutua esclusione.
In tal modo si potrebbe incorrere in *race condition* sull'incremento di `numeroLettori`, rendendo inconsistente la determinazione del primo lettore cui spetta l'invocazione di `P(dati)`, vanificando pertanto l'intero algoritmo.
È quindi ovviamente necessario regolamentare l'accesso mutuamente esclusivo anche nella fase di incremento, come riportato nella seguente versione corretta del codice:

```
void Lettore (void) {
  while (true) {
    P(mutex);
    numeroLettori++;
    if (numeroLettori == 1)
      P(dati);
    V(mutex);
    // leggi il dato
    P(mutex);
    numeroLettori--;
    if (numeroLettori==0)
      V(dati);
    V(mutex);
    // usa il dato letto
  }
}
```

```
void Scrittore (void) {
  while (true) {
    // prepara il dato da scrivere
    P(dati);
    // scrivi il dato
    V(dati);
  }
}
```

Figura 5: Soluzione corretta del problema lettori-scrittori.

Soluzione 2 (punti 8).

- *Round-Robin* senza priorità e con prerilascio

  (ciascuna esecuzione dei blocchi P2-3 ed F1 impiega esattamente un quanto di tempo)

```
processo P     PssspppPssspppP.          LEGENDA DEI SIMBOLI
processo A    -aAaaAaaaAaaAa.            - non ancora arrivato
processo B    -bbBbbBbbbBbbb.            x (minuscolo) attesa
processo C    -cccCcccCccCcc.            X (maiuscolo) esecuzione
                                         s sospeso
                                         . coda vuota
CPU           .PABCABPCABCAP.
coda          pabcabpcabcapb.            ogni spazio corrisponde
              .bcabpcabcapbc.            ad un quanto
              .c...cab...bca.
```

processo	# iterazioni	valore finale di `indice`	tempo di attesa (in quanti)
P			4
A	4	5	
B	3	4	
C	3	4	

- *Round-Robin* con priorità e con prerilascio

  (ciascuna esecuzione dei blocchi P2-3 ed F1 impiega esattamente un quanto di tempo)

```
processo P     PsssPsssP.                LEGENDA DEI SIMBOLI
processo A    -aaaaaaaaa.                - non ancora arrivato
processo B    -bbbbbbbbb.                x (minuscolo) attesa
processo C    -cCCCcCCCc.                X (maiuscolo) esecuzione
                                         s sospeso
                                         . coda vuota
CPU           .PCCCPCCCP
coda          pcbbbcbbb.                 ogni spazio corrisponde
              .baaabaaa.                 ad un quanto
              .a...a....
```

processo	# iterazioni	valore finale di `indice`	tempo di attesa (in quanti)
P			0
A	6	7	
B	0	1	
C	0	1	

- PD (*priority dispatching*), strettamente su base prioritaria e con prerilascio

  (nel programma in esame si ha prerilascio solamente quando il processo `Padre`, a priorità maggiore, si risveglia dopo la sospensione volontaria e scalza il processo `Figlio` C, quello a priorità maggiore tra i figli, dall'esecuzione)

```
processo P     PsssPsssP.                LEGENDA DEI SIMBOLI
processo A    -aaaaaaaaa.                - non ancora arrivato
processo B    -bbbbbbbbb.                x (minuscolo) attesa
processo C    -cCCCcCCCc.                X (maiuscolo) esecuzione
                                         s sospeso
                                         . coda vuota
CPU           .PCCCPCCCP
coda          pcbbbcbbb.                 ogni spazio corrisponde
              .baaabaaa.                 ad un blocco di esecuzione
              .a...a....                 (singola iterazione o terminazione)
```

processo	# iterazioni	valore finale di `indice`	tempo di attesa (in unità di tempo)
P			0
A	6	7	
B	0	1	
C	0	1	

Si noti come l'equivalenza tra la durata di ogni blocco di esecuzione considerato con l'ampiezza del quanto di tempo utilizzando dalla politica *Round-Robin* renda questo caso del tutto analogo a quello in cui la politica *Round-Robin* con prerilascio sia affiancata da selezione a priorità.

Soluzione 3 (punti 6). La figura 6 illustra la relazione temporale che si verifica tra la generazione di gettoni, il flusso di ingresso ed uscita, ed il conseguente stato della coda presso il *router* in esame, sottoposto al traffico ipotizzato nel quesito. Come vediamo, il flusso di ingresso (misurato come segmenti in ingresso in un intervallo fissato t, $B_{in}(t) = \sum_i S_i \times \lceil \frac{t-T_i}{t} \rceil$) al tempo immediatamente successivo all'istante $t = 20$, è pari a $B_{in}(20) = \sum_i S_i = 21$, largamente inferiore alla banda di uscita consentita dalla frequenza fissata per la generazione dei gettoni, pari a $B_{out}(t = 20) = \lfloor \frac{t=20}{\Delta T = \frac{1}{3}} \rfloor = 60$.

Un tale sbilanciamento comporta l'immediata emissione di tutti i messaggi in ingresso, indipendentemente dalla loro ampiezza, con conseguente rischio di congestione delle linee di uscita.

Questo problema può essere risolto riducendo la frequenza di generazione di gettoni così da porre un limite superiore ai picchi istantanei di uscita. Per esempio, come illustrato in figura 7, fissando $\Delta T = 1$ e limitando l'orizzonte di osservazioni agli arrivi indicati dal quesito, si ridurrebbe il picco massimo di uscita ad 8 segmenti, con ritardo di $R = 1$ unità di tempo nel completamento dell'emissione del 4^o messaggio. Per $\Delta T = \frac{4}{5}$, invece, il picco massimo di uscita si ridurrebbe a 6 segmenti, senza eccedere la capacità data della coda di attesa, e portando a $\frac{25}{4}$ il ritardo R. E così via.

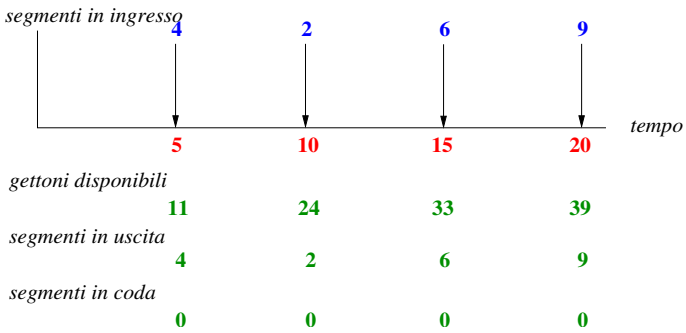

Figura 6: Generazione di gettoni, flusso di ingresso, stato della coda e profilo di uscita per il *router* in esame. ($\Delta T = \frac{1}{3}$)

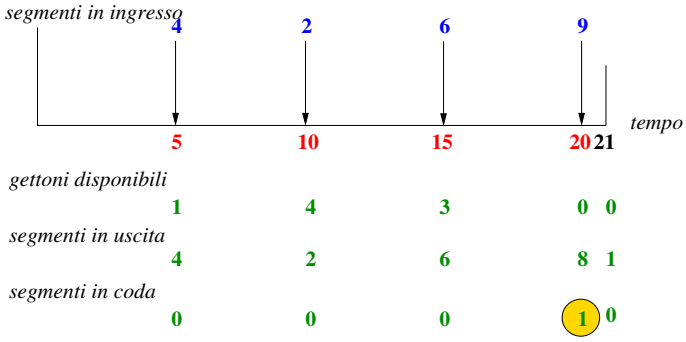

Figura 7: Stato della coda e profilo di uscita per il *router* in esame dopo la correzione della frequenza di generazione dei gettoni a $\Delta T = 1$.

Al contrario, ove il flusso di ingresso fosse superiore alla banda di uscita consentita dalla frequenza di generazione dei gettoni, il comportamento dell'algoritmo degraderebbe fino a diventare indistinguibile dall'algoritmo detto *Leaky Bucket*, il cui flusso di uscita è costante ed indipendente da eventuali picchi nel flusso di ingresso, con conseguente incremento del tempo medio richiesto per lo smaltimento della coda e rischio di intasamento di coda e perdita di segmenti.

Soluzione 4 (punti 6). Poiché la figura 3 mostrata nel quesito non assegnava pesi agli archi, dobbiamo assumere che tutte le connessioni abbiano la medesima banda. La figura 8 mostra le tabelle di instradamento di cui il nodo **A1** dovrebbe dotarsi per supportare le due forme di indirizzamento descritte dal quesito.

indirizzamento lineare

destinazione	arco di uscita	# passi
A1	—	—
A2	A2	1
A3	A3	1
B1	A2	3
B2	A2	2
B3	A2	3
C1	A3	3
C2	A3	2
C3	A3	3

indirizzamento gerarchico

destinazione	arco di uscita	# passi
A1	—	—
A2	A2	1
A3	A3	1
Regione B	A2	2
Regione C	A3	2

Figura 8: Tabelle di instradamento del nodo **A1**.

Dalla figura è facile riconoscere il vantaggio principale dell'instradamento gerarchico, che riduce l'ampiezza delle tabelle di instradamento al crescere del numero di nodi per regione. Tale importante vantaggio è però attenuato dal fatto che, ove le regioni non fossero completamente connesse (come invece è nel caso in esempio) la lunghezza di taluni percorsi potrebbe risultare indesiderabilmente maggiore. Si effettui per esercizio la verifica di questa osservazione.

Soluzione 5 (punti 5). L'indirizzo acquistato dall'Azienda è un dato di fatto che non influisce sulla pianificazione degli indirizzi IP della rete Aziendale, fatta salva la configurazione della porta E0 del *router* 1. Per gli altri dispositivi abbiamo a disposizione un'intera classe C di indirizzi riservati 192.168.1.0.
Per prima cosa individuiamo tutte le sottoreti cui dobbiamo assegnare gli indirizzi della classe C data, opportunamente suddivisi in sottoreti.

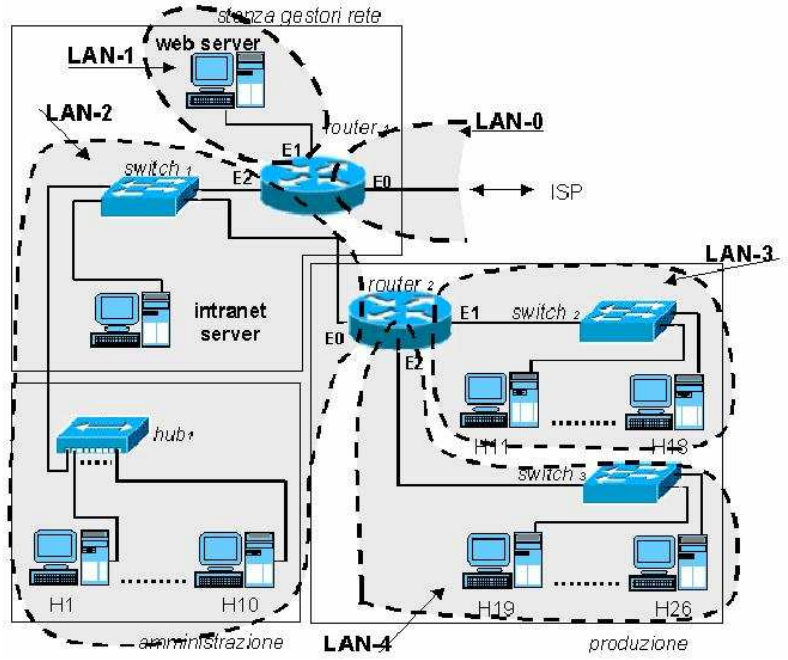

Figura 9: Individuazione delle sottoreti presenti nella rete Aziendale.

Come mostrato in figura 9, individuiamo 5 sottoreti:

LAN-0	sottorete di collegamento con l'ISP, non di nostra pertinenza come assegnazione indirizzi IP
LAN-1	sottorete demilitarizzata (DMZ), accessibile 'liberamente' agli utenti *Internet*, composta solo dal web server
LAN-2	sottorete composta dall'intranet server e dagli utenti del reparto amministrativo
LAN-3	sottorete composta dagli utenti della produzione appartenenti alla prima zona
LAN-4	sottorete composta dagli utenti della produzione appartenenti alla seconda zona

Il dato di progetto da cui partire è che dobbiamo utilizzare sottoreti tutte fissate alla stessa, massima, dimensione e con la medesima *subnet mask*. Dovendo creare 4 sottoreti utili (LAN-1 — LAN-4), e sapendo che il numero di sottoreti utili S è determinato dalla condizione $S \leq 2^N - 2$, dove N, il minimo intero che soddisfa tale condizione, è il numero di *bit* sottratti al campo nodo dell'indirizzo IP per rappresentare le sottoreti (operazione che, come sappiamo, viene denominata *subnetting*), otteniamo: $S = 4 < 2^{N=3} - 2 = 6$. Pertanto, oltre al campo rete, ampio 24 *bit* (come prescritto per gli indirizzi di classe C), dobbiamo isolare un campo sottorete ampio 3 *bit*, lasciando i rimanenti $8 - 3 = 5$ *bit* per il campo nodo. L'ampiezza di quest'ultimo campo ci consente di esprimere $2^5 - 2 = 30$ indirizzi IP utili per sottorete, un numero più che sufficiente per risolvere il problema proposto.

La tabella 1 riporta le caratteristiche degli indirizzi IP delle prime quattro sottoreti utili.
Da questa ripartizione possiamo determinare una possibile attribuzione di indirizzi IP alle varie postazioni della rete aziendale, riportata in tabella 2.

parte di rete	*subnetting*	parte di nodo		destinazione indirizzo
11000000 . 10101000 . 00000001.	001	0 0000	192.168.1.32	riservato per la I sottorete
11000000 . 10101000 . 00000001.	001	0 0001	192.168.1.33	utilizzabile per 1^o nodo nella I sottorete
...				
11000000 . 10101000 . 00000001.	001	1 1110	192.168.1.62	utilizzabile per ultimo ($= 30^o$) nodo nella I sottorete
11000000 . 10101000 . 00000001.	001	1 1111	192.168.1.63	riservato al *broadcast* nella I sottorete
11000000 . 10101000 . 00000001.	010	0 0000	192.168.1.64	riservato per la II sottorete
11000000 . 10101000 . 00000001.	010	0 0001	192.168.1.65	utilizzabile per 1^o nodo nella II sottorete
...				
11000000 . 10101000 . 00000001.	010	1 1110	192.168.1.94	utilizzabile per ultimo ($= 30^o$) nodo nella II sottorete
11000000 . 10101000 . 00000001.	010	1 1111	192.168.1.95	riservato al *broadcast* nella II sottorete
11000000 . 10101000 . 00000001.	011	0 0000	192.168.1.96	riservato per la III sottorete
11000000 . 10101000 . 00000001.	011	0 0001	192.168.1.97	utilizzabile per 1^o nodo nella III sottorete
...				
11000000 . 10101000 . 00000001.	011	1 1110	192.168.1.126	utilizzabile per ultimo ($= 30^o$) nodo nella III sottorete
11000000 . 10101000 . 00000001.	011	1 1111	192.168.1.127	riservato al *broadcast* nella III sottorete
11000000 . 10101000 . 00000001.	100	0 0000	192.168.1.128	riservato per la IV sottorete
11000000 . 10101000 . 00000001.	100	0 0001	192.168.1.129	utilizzabile per 1^o nodo nella IV sottorete
...				
11000000 . 10101000 . 00000001.	100	1 1110	192.168.1.158	utilizzabile per ultimo ($= 30^o$) nodo nella IV sottorete
11000000 . 10101000 . 00000001.	100	1 1111	192.168.1.159	riservato al *broadcast* nella IV sottorete

Tabella 1: Ripartizione degli indirizzi nelle prime 4 sottoreti interne disponibili.

rete	**dispositivo**	*IP address*	*subnet mask*	*default gateway*
LAN-1	web server	192.168.1.61	255.255.255.224	192.168.1.62
	router-1 E1	192.168.1.62	$= /27$	—
LAN-2	H1	192.168.1.65	$/27$	192.168.1.94
	H2	192.168.1.66	$/27$	192.168.1.94
	H3	192.168.1.67	$/27$	192.168.1.94
	H4	192.168.1.68	$/27$	192.168.1.94
	H5	192.168.1.69	$/27$	192.168.1.94
	H6	192.168.1.70	$/27$	192.168.1.94
	H7	192.168.1.71	$/27$	192.168.1.94
	H8	192.168.1.72	$/27$	192.168.1.94
	H9	192.168.1.73	$/27$	192.168.1.94
	H10	192.168.1.74	$/27$	192.168.1.94
	intranet server	192.168.1.92	$/27$	192.168.1.94
	router-2 E0	192.168.1.93	$/27$	—
	router-1 E2	192.168.1.94	$/27$	—
LAN-3	H11	192.168.1.97	$/27$	192.168.1.126
	H12	192.168.1.98	$/27$	192.168.1.126
	H13	192.168.1.99	$/27$	192.168.1.126
	H14	192.168.1.100	$/27$	192.168.1.126
	H15	192.168.1.101	$/27$	192.168.1.126
	H16	192.168.1.102	$/27$	192.168.1.126
	H17	192.168.1.103	$/27$	192.168.1.126
	H18	192.168.1.104	$/27$	192.168.1.126
	router-2 E1	192.168.1.126	$/27$	—
LAN-4	H19	192.168.1.129	$/27$	192.168.1.158
	H20	192.168.1.130	$/27$	192.168.1.158
	H21	192.168.1.131	$/27$	192.168.1.158
	H22	192.168.1.132	$/27$	192.168.1.158
	H23	192.168.1.133	$/27$	192.168.1.158
	H24	192.168.1.134	$/27$	192.168.1.158
	H25	192.168.1.135	$/27$	192.168.1.158
	H26	192.168.1.136	$/27$	192.168.1.158
	router-2 E2	192.168.1.158	$/27$	—
LAN-0	router-1 E0	dati forniti dall'ISP		

Tabella 2: Una possibile attribuzione di indirizzi IP interni alle postazioni della rete Aziendale.