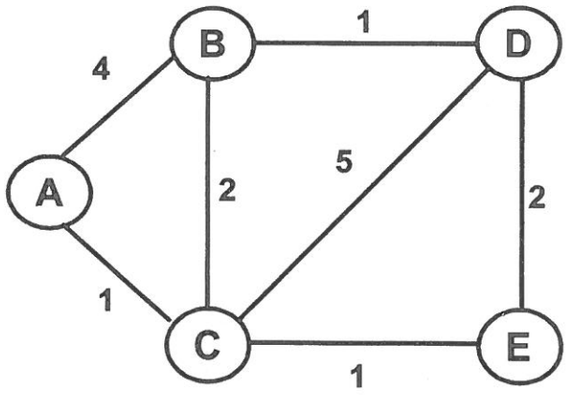
1. The shortest path algorithm is as follows, where s means the source node, LC(s,v) denotes the link cost from the node s to the node v, and V denotes the nodes in the graph. C(v) and P(v) are arrays which store the minimum cost and the upstream node identifier in the shortest path for the node v, respectively.

```
for each v in V-{s} {
  if v is adjacent to s {
     C(v) = LC(s,v)
     P(v) = s
  }
  else
     C(v) = infinity
}
T = {s}
while (T != V) {
  find w in (V-T),s.t. C(w) is the minimum for all possible w
  T = T ∪ {w}
  for each v in (V-T)
    if (C(w) + LC(w,v) < C(v)) {
      C(v) = C(w) + LC(w, v)
      P(v) = w
    }
  }
}
```

In the following case, assume the node A is the source node.

(a) Fill in the iteration table in your answer sheet. (20 points)

(b) Use big-O notation to express the time complexity of this algorithm and explain your answer. (5 points)



Iteration	T	C(B),P(B)	C(C),P(C)	C(D),P(D)	C(E),P(E)
0	A	4,A	1,A	∞	∞
1	AC				
2					
3					
4					

2. The IP checksum is computed in the following C code. What's its output? (25 points)

```
#include <stdio.h>
unsigned short
checksum(unsigned short *addr, int count) {
  /* Compute Internet Checksum for "count" bytes
      beginning at location "addr".  */
  register long sum = 0;
  while( count > 1 )  {
    sum += * (unsigned short *) addr++;
    count -= 2;
    printf("%04x\n", (unsigned short)sum);
  }
  if( count > 0 )
    sum += * (unsigned char *) addr;
  while (sum >> 16)
    sum = (sum & 0xffff) + (sum >> 16);
  return  ~sum;
}
int main() {
  unsigned short A[] = {0x36f7,0xf670,0x2148,
                        0x8912,0x2345,0x7863,0x0076};
  printf("Checksum is %04x\n", checksum(A, sizeof A));
  return 0;
}
```

3. If every nonleaf node in a binary tree has nonempty left and right subtrees, the tree is termed a ***strictly binary tree***. Prove that a strictly binary tree with n leaves contains $2n - 1$ nodes. (15 points)

4. Consider the following code:

```
if (c1)
 if (c2)
  if (c3)
   ...
    if (cn)
       { statement }
```

where c_i is a condition that is either true or false. Note that rearranging the conditions in a different order results in an equivalent program, since the {statement} is only executed if all the c_i are true. Assume that time(i) is the time needed to evaluate condition c_i and that prob(i) is the probability that condition c_i is true. In what order should the conditions be arranged to make the program most efficient? Prove your answer. (15 points)

5. (a) Write a program to store capital letters alphabetically in a two-dimensional array whose width and height are given by users, and display them as follows. (5 points)

```
5 3  ← user's input
ABCDE
FGHIJ
KLMNO
```

```
8 2  ← user's input
ABCDEFGH
IJKLMNOP
```

(b) Modify the above program, and let it print this two-dimensional array in counterclockwise spiral order from the top-left corner as the following example. (15 points)

```
5 3  ← user's input
ABCDE
FGHIJ
KLMNO
AFKLMNOJEDCBGHI  ← output in counterclockwise spiral order
```