國立中央大學108學年度碩士班考試入學試題

所別： 資工類

科目： 資料結構與演算法

本科考試禁用計算器 ＊請在答案卷(卡)內作答

單選題 25% (每題5分，答錯倒扣一分) ，倒扣到該大題0分為止。

1. Suppose that the following list is the result of the first partition step of quick sort.

   12 9 1 13 19 24 22 20

   Which of the following statements is correct about the partition step?

   (A) The pivot could have been either 13 or 19.

   (B) The pivot must be 13.

   (C) Both 13 and 19 are pivots in the first partition.

   (D) Neither 13 nor 19 could have been the pivot.

   (E) None of the above

2. Build a *binary search tree* for the input sequence 9, 4, 8, 7, 20, 15, 14, 3, 10. It is assumed that the tree root is on level 1.

   (A) There are five levels in the tree.

   (B) 20 is on level 3.

   (C) 7 and 10 have the same father.

   (D) The left subtree of 20 has 1 node.

   (E) None of the above

3. Which of the following tree is a legal max-heap?

   (A) A tree with level-order traversal sequence {67, 45, 19, 22, 43, 16}

   (B) A tree with level-order traversal sequence {67, 43, 19, 22, 45, 16}

   (C) A tree with level-order traversal sequence {14, 18, 27, 19, 63, 48}

   (D) A tree with level-order traversal sequence {45, 22, 67, 8, 34, 52}

   (E) None of the above.

The following procedure recursively generates all the permutations of list[*i*] to list[*n*].

```
void perm(char *list, int i, int n)
{
 int j, temp;
 if (i==n) { /* print the newly generated permutation */
   for (j=0; j<=n; j++)
       printf("%c", list[j]);
   printf("     ");
}
else { /* generate permutations recursively */
     for (j=i; j<=n; j++) {
         ____(B1)____
         ____(B2)____
         swap(list[i], list[j], temp);}}}
```


4. Blank (B1) in the algorithm above should be

   (A) swap(*list*[*i*+1], *list*[*j*], *temp*)

   (B) swap(list[*i*], list[*j*], *temp*)

   (C) swap(*list*[*i*], *list*[*j*+1], *temp*)

   (D) swap(*list*[*i*+1], *list*[*j*+1], *temp*)

   (E) None of the above

注意：背面有試題

5. Blank (B2) in the algorithm above should be
   (A) perm(*list*, *i*+1, *n*)
   (B) perm(*list*, *i*, *n*)
   (C) perm(*list*, *j*, *n*)
   (D) perm(*list*, *j*+1, *n*)
   (E) None of the above

複選題 35% (每題5分，答錯每題倒扣1分) ，倒扣到該大題0分為止。

6. Consider an empty hash table with 10 buckets and each bucket has 2 slots. Suppose that the hash function is h(n) = n % 10, and the following numbers, 6 16 22 45 54 25 7 75 5 108, are sequentially inserted into the hash table. Which of the following statements are true?
   (A) if linear probing is adopted to handle overflow, the summation of the numbers in the full buckets of the hash table is 287.
   (B) if linear probing is adopted to handle overflow, the summation of the numbers in the full buckets of the hash table is 172.
   (C) if quadratic probing is adopted to handle overflow, the summation of the numbers in the full buckets of the hash table is 221.
   (D) Compared with quadratic probing, linear probing requires fewer bucket accesses (in average).
   (E) None of the above.
7. A hash function maps a key into a bucket in the hash table. Which of the following statements is true?
   (A) A division hash function with divisor $D = 7^r$, where r is an integer, may result in serious collision.
   (B) Compared with hash chaining, open addressing requires fewer bucket accesses (in average)
   (C) When hash chaining is used to resolve overflows, the search for a key involves comparison with keys that have different hash values.
   (D) In dynamic hashing, data buckets grows or shrinks (added or removed dynamically) as the records increases or decreases.
   (E) None of the above.
8. Consider the problem of solving all-pairs shortest-paths on a weighted directed graph $G = (V, E)$. A famous dynamic programming algorithm gives a recursive formula to compute $d^{(k)}[i, j]$, the length of a shortest path from i to j using only vertices with indices not greater than k. Here it is assumed that the vertex set $V = \{1, 2, \ldots, n\}$ if the given graph contains n vertices. Now if the given graph has 5 vertices and edges: (1,4,–5), (1,5,2), (2,1,6), (3,1,1), (3,2,7), (4,3,4), (5,2,–4), (5,3,3), (5,4,8), where each triple (i, j, t) represents there is an edge directed from i to j with weight t. Then, after the execution of the algorithm, each term $d^{(k)}[i, j]$ will be computed correctly. For the following items, choose the correct one(s):
   (A) $d^{(2)}[5,1] = 2$
   (B) $d^{(2)}[5,4] = -3$
   (C) $d^{(3)}[4,1] = 4$
   (D) $d^{(3)}[4,2] = 11$
   (E) $d^{(3)}[4,5] = 6$
9. Follow the previous question. Choose the correct item(s):
   (A) $d^{(4)}[1,2] = 7$
   (B) $d^{(4)}[1,3] = -1$
   (C) $d^{(5)}[1,2] = -2$
   (D) $d^{(5)}[3,2] = -2$
   (E) $d^{(5)}[4,2] = 3$

注意：背面有試題

參考用

10. Given a weighted undirected graph $G = (V, E)$, let $\delta[u, v]$ denote the distance (the length of a shortest path) from vertex u to vertex v. A path $[v_1, v_2, \ldots, v_k]$ is said to be a *progress path*, if $\delta[v_i, v_k] > \delta[v_{i+1}, v_k]$ for $1 \leq i < k$. For example, consider the graph with vertex set {1, 2, 3, 4} and edge set: (1,2,2), (2,4,1), (3,1,2), (3,4,2), (1,4,3), where each triple (i, j, t) represents there is an undirected edge between i and j with weight t. Then, in this graph, there are totally 2 shortest paths from 1 to 4 (i.e. [1,2,4] and [1,4]) and there are 3 progress paths from 1 to 4 (they are [1,2,4], [1,4] and [1,3,4]). Now consider another graph with vertex set {a, b, c, d, e, f, g, h} and edge set: (a,b,3), (a,c,1), (b,c,1), (b,e,5), (b,f,4), (c,d,9), (d,e,5), (d,g,3), (e,f,2), (e,g,2), (f,h,7), (g,h,4). According to this graph, choose the correct item(s):
(A) δ[a,h] = 13
(B) δ[b,h] = 12
(C) δ[d,h] = 6
(D) δ[e,h] = 5
(E) δ[f,h] = 7

11. Follow the previous question. Choose the correct item(s):
(A) [b,c,d,g,h] is a progress path.
(B) [b,e,d,g,h] is a progress path.
(C) [b,e,f,h] is a progress path.
(D) [a,b,f,h] is a progress path.
(E) ([a,b,e,g,h] is a progress path.

12. Follow the previous question. Choose the correct item(s):
(A) There is totally 1 progress path from e to h.
(B) There are totally 9 progress paths from a to h
(C) There is totally 1 progress path from f to h.
(D) There are totally 3 progress paths from b to h.
(E) There are totally 5 progress paths from c to h.

問答題 40%

1. A divide-and-conquer algorithm solves a problem directly if the problem is small. If the problem is large, it is first **divided** into two or more parts called subproblems. Each subproblem is then recursively solved (**conquered**) in the same manner. Afterwards, the solutions to the subproblems are combined (**merged**) into a solution to the original problem. The merge sort algorithm is a well-known example following the divide-and-conquer paradigm. The key operation of the merge sort algorithm is the merging of two sorted sequences. To perform the merging, we use an auxiliary procedure MERGE(A, p, q, r), where A is an array and p, q, and r are indices numbering elements of the array such that $p \leq q < r$. The procedure assumes that the subarrays $A[p .. q]$ and $A[q + 1 .. r]$ are in sorted order. It merges them to form a single sorted subarray that replaces the current subarray $A[p .. r]$. The pseudo code of the MERGE operation is shown below. Based on the operation, we can design the merge sort algorithm easily.
(a) Write a detailed divide-and-conquer merge sort algorithm using the MERGE operation. Note that the algorithm **MUST** include well-described input and output. (9%)
(b) Analyze the time complexity of the merge sort algorithm with the big-O notation. (8%)

MERGE(A, p, q, r)
Input:
A: an array of elements;
p, q, r: indices of A, where $p \leq q < r$, and the subarrays $A[p .. q]$ and $A[q + 1 .. r]$ are in sorted order (from small to large).

Output:

A: an array of elements, where the subarray $A[p .. r]$ is in sorted order (from small to large)

```
n1 ← q - p + 1
n2 ← r - q
create arrays L[1.. n1 + 1] and R[1.. n2 + 1]
for i ← 1 to n1
    do L[i] ← A[p + i - 1]
for j ← 1 to n2
    do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
    do if L[i] ≤ R[j]
          then A[k] ← L[i]
               i ← i + 1
          else A[k] ← R[j]
               j ← j + 1
```

2. Given a chain $<A_1,\ldots, A_n>$ of n matrices, where matrix A_i, $i=1,\ldots,n$, has dimension $p_{i-1}\times p_i$, the matrix-chain multiplication problem is to fully parenthesize the product $A_1\ldots A_n$ in a way that minimizes the number of scalar multiplications. We can use the dynamic programming strategy to solve the matrix-chain multiplication problem as follows. Define $m[i, j]$ to be the minimum number of scalar multiplications needed to compute the matrix product $A_i\ldots A_j$, $i\leq j$. Since we know the value of $m[i, j]$ for $i=j$, we can then calculate $m[1, n]$ in a bottom-up manner as the minimum number of scalar multiplications for the product $A_1\ldots A_n$ by a recursive form of $m[i, j]$.
Write down the recursive form of $m[i,j]$. (8%)

3. Below is the algorithm AllPairCost which computes the shortest distances between all pairs of vertices i, j, where $i \neq j$. Formally, given distances of edges in graph G, determine the shortest distances between all pairs of vertices in G. Note that the distance of an arbitrary edge in G is non-negative.
   (a) Blanks (B3) and (B4) in the algorithm below should be _______(4%)
   (b) Blank (B5) in the algorithm below should be _______(3%)

Algorithm AllPairCost
Input: a two dimensional array C, where $C[i][j]$ denotes the distance of directed edge (i, j) (i.e., the edge from vertex i to vertex j) .
Output: a two dimensional array D, where $D[i][j]$ denotes the shortest distance from vertex i to vertex j.

```
int i, j, k;
for ( i=0; i<n; i++ )
   for (j=0; j<n; j++ )
        ___(B5)___;
for ( k=0; k<n; k++ )
   for ( i=0; i<n; j++ )
        for (j=0; j<n; j++ )
            if(___(B3)___)
                 ___(B4)___;
```


注意：背面有試題

4. Suppose that array *A*[1:*n*] maintains a binary tree. For a binary tree node stored in *A*[*i*], its two children (if exist) are stored in *A*[2*i*] and *A*[2*i*+1], respectively. The program below aims to complete the following two tasks

   i. adjust array A to establish a max heap, and
   ii. apply heap sort on the max heap built in (1) in nondecreasing order.

(a) Blank (B6) in the program below should be_________ (4%)

(b) Blank (B7) in the program below should be ____________ (4%)

```
void adjust (int A[], int root, int n)
{
  int child, rootkey;
  int temp;
  temp = A[root];
  rootkey = A[root];
  child = 2* root;
  while ( child <=n ) {
     if (( child <=n ) && ( A[child] < A[child+1] ))
          child ++;
     if ( rootkey > A[child] )
          break;
     else {
          ____(B6)____;
          child *=2; }
  }
  ____(B7)____;
}

void heapsort (int A[], int n)
{ /* perform a heap sort on A[1:n] */
   int i, j;
   int temp;
   for (i=n/2; i>0; i++)       /* adjust the binary tree to establish the max heap */
     adjust (A, i, n);
   for (1=n-1; i>0; i--) {          /* heap sort */
     swap( A[1], A[i+1], temp);     /* exchange A[i] and A[i+1] */
     adjust(A, 1, i); }
}
```