一、Basic Concepts

1． Algorithmic mathematics provides a language for talking about program behavior.

2． The complexity of an algorithm is reflected in the time and memory resources needed to run the algorithm, so the complexity of the algorithm can be divided into (The Time Complexity) and (The Space Complexity).

2. Drop low-order terms; ignore leading constants. $3n^3 + 90n^2 – 5n + 6046 = (\Theta(n^3))$

3. An algorithm is a set of finite rules, in which rules specify a series of operations to solve a particular type of problem. In addition, the algorithm should have the following five important characteristics:

4. Prim algorithm uses (Greedy) strategy to solve. Many problems that can be solved by Greedy algorithms generally have two important properties: (optimal substructure properties) and (Greedy selection) properties。

5. A **greedy algorithm** is an algorithmic paradigm that follows the problem solving heuristic of （making the locally optimal choice） at each stage with the intent of finding a global optimum.

6. The two basic elements of dynamic programming algorithm are: (optimal substructure) and (overlapping subproblem)

7. Optimal substructure means that the solution to a given optimization problem can be obtained by the combination of optimal solutions to （its sub-problems）.

8. Overlapping sub-problems means that the space of sub-problems must be small, that is, any recursive algorithm solving the problem should solve （the same sub-problems） over and over, rather than generating （new sub-problems）.

9. An algorithm that calls itself directly or indirectly is called (a recursive algorithm)

10. The single source shortest path problem can be solved by(a greedy **algorithm**) and(branch and bound strategy).

11. Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, and the backtracking searches the solution space tree from the root node according to (the depth-first way)

12. Branch-and-bound algorithm often searches the solution space tree of the problem(in the way of breadth-first) or(minimum cost-first).

13. The branch-and-bound method includes two forms (FIFO branch-and-bound way) and (pioneer branch-and-bound way).

14. The main difference between greedy algorithm and dynamic programming algorithm is (greedy selectivity).

15. Algorithmic features: An algorithm should have the following five important features: (1) Finiteness: An algorithm must ensure that it finishes after a finite step is executed; (2) Accuracy: Each step of the algorithm must be clearly defined; (3) Input: An algorithm has 0 or more inputs to describe the initial condition of the operation object. The so-called

zero input refers to the definition of the algorithm itself, except the initial condition. (4) Output: An algorithm has one or more outputs to reflect the results of processing input data. Algorithms without output are meaningless; (5) Feasibility: The algorithm can run accurately in principle, and people can do finite operations with pen and paper.

16．*A **flow network** is a directed graph (V,E,c,s,t) where (V,E,c) forms a weighted graph with positive edge weights, s ∈ V is a source node and t ∈ V is a sink node.*

17．A flow must satisfy the restriction that the amount of flow into a node （ equals ） the amount of flow out of it, unless it is a **source**, which has only outgoing flow, or **sink**, which has only incoming flow.

18． Max-flow min-cut theorem. The maximum value of an s-t flow is equal to(the minimum capacity over all s-t cuts).

19. (The divide-and-conquer) paradigm is often used to find the optimal solution of a problem. Its basic idea is to decompose a given problem into two or more similar, but simpler, subproblems, to solve them in turn, and to compose their solutions to solve the given problem. Merge sort is a (divide and conquer) algorithm.

20. The class P problems means the existence of an algorithm solving the task that runs in (polynomial time), such that the time to complete the task varies as a polynomial function on the size of the input to the algorithm. For NP problems, they could not be solved in （polynomial time）, but the answer could be verified in polynomial time.(that is, there is no known way to find an answer quickly, but if one is provided with information showing what the answer is, it is possible to verify the answer quickly. The class of questions for which an answer can be (verified in polynomial time) is called NP, which stands for "(nondeterministic polynomial time)".

二、

1. write out the main steps of designing the dynamic programming algorithm.
   Solution:
   写出了动态规划算法设计的主要步骤。
   （1） Find out the nature of the optimal solution and characterize its structural characteristics. （ 找出最优解的性质并描述其结构。）
   （2） The optimal value is defined recursively. （递归地定义最优值）;
   （3） Calculate the optimal value from bottom to top.（以自底向上的方式计算最优值;）
   （4） Construct the optimal solution according to the information obtained when calculating the optimal value.（根据计算最优值时得到的信息，构造最优解。）

3．Greedy algorithm

Greedy algorithm always makes the best choice at present. That is to say, the greedy algorithm does not consider the overall optimum, and the choice it makes is only the local optimum choice in a sense.
贪婪算法总是做出当前最优的选择。也就是说，贪婪算法不考虑全局最优，它所做的选择在

某种意义上只是局部最优选择。

4． Describe the basic idea of divide and conquer.

Answer: The basic idea of divide-and-conquer is to decompose a problem of n_scale into k smaller sub-problems, which are independent of each other and the same as the original problem. Recursively solve these sub problems, and then merge the solutions of the sub problems to get the solution of the original problem.

答：分治的基本思想是将 n 尺度问题分解为 k 个相互独立、与原问题相同的小子问题。递归求解这些子问题，然后合并子问题的解，得到原问题的解。

5．Many problems that can be solved by greedy algorithms generally have two important properties: the optimal substructure property and greedy selection property.

贪心算法所能解决的许多问题通常具有两个重要性质：最优子结构性质和贪心选择性质。

6．What is the optimal substructure property? 最优子结构性质是什么？

Answer: When the optimal solution of a problem contains the optimal solution of its sub-problem, it is said that the problem has the optimal sub-structure property. The optimal substructure property of the problem is the key feature that the problem can be solved by dynamic programming or greedy algorithm.

7. Briefly describe the basic idea of backtracking.

Answer: Backtracking is a systematic and jumping search algorithm. When using it to solve problems, the solution space of the problem should be clearly defined. After determining the organizational structure of the solution space, the retrospective method searches the whole solution space in a depth-first manner starting from the starting node (root node). This starting node becomes a live node, and also becomes the current expansion node. At the current extension node, the search moves to a new node in depth. This new node becomes a new active node and becomes the current expansion node. If the current extended node can not move in the depth direction, the current extended node will become a dead node. At this point, the loopback should be moved back to the nearest loopback node and make the loopback node the current extended node. In this way, the backtracking method searches the solution space recursively until it finds the required solution or there are no active nodes in the solution space.

8．Briefly describe the basic idea of the Branch and bound algorithm.

Answer: Branch and bound is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as mathematical optimization. A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. The algorithm explores branches of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm. There are 3 types of nodes.

Alive node: a generated node whose sons have not all been generated.

E-node: the node whose son is generating currently.

Dead node: a node that all his sons have been generated or there is no need to generate his sons.

9．Briefly describe the maximum flow problems。

Answer： In optimization theory, maximum flow problems involve finding a feasible flow through a single-source, single-sink flow network that is maximum. The maximum flow problem can be seen as a special case of more complex network flow problems, such as the circulation problem. The maximum value of an s-t flow (i.e., flow from source s to sink t) is equal to the minimum capacity of an s-t cut (i.e., cut severing s from t) in the network, as stated in the max-flow min-cut theorem.

10．Compare greedy algorithm with dynamic programming.

Answer: The basic idea of greedy is to approach the given goal step by step from an initial solution of the problem in order to get a better solution as soon as possible. The algorithm stops when a step in an algorithm can't continue.
But greedy algorithms have some problems:
1. There is no guarantee that the final solution is the best.
2. Can not be used to find the maximum or minimum solution problem;
3. Only find the range of feasible solutions satisfying some constraints.

Greedy algorithm always makes the best choice at present, that is to say, greedy algorithm does not consider the overall optimum, the choice it makes is only the local optimum choice in a sense. Of course, we hope that the final result from the local optimal selection will be the global optimal solution of the problem.

The basic idea of dynamic programming: Dynamic programming algorithms are usually used to solve problems with some optimal properties. In such problems, there may be many feasible solutions. Each solution corresponds to a value, and we want to find the solution with the best value. The essence of dynamic programming is to divide and solve redundancy. Therefore, dynamic programming is an algorithmic strategy that decomposes problem instances into smaller and similar sub-problems, and stores the solutions of sub-problems to avoid duplication of computation in order to solve optimization problems.

It can be seen that the dynamic programming algorithm is similar to the greedy algorithm. Both of them generalize the problem instances into smaller and similar sub-problems, and generate a global optimal solution by solving sub-problems.

Dynamic programming is different from the greedy algorithm.

Compared with the greedy strategy, the current choice of the greedy strategy may depend on all the choices already made, but not on the choices and sub-problems to be made. Therefore, the greedy method makes greedy choices step by step from top to bottom, so once the solution of each sub-problem is found recursively, the solution of the sub-problem can be merged into the solution of the problem from bottom to top. However, if the current choice may depend on the solution of the sub-problem, it is difficult to achieve the global optimal solution through local greedy strategy.

However, unlike the greedy approach, dynamic programming allows these sub-problems to be independent (i.e., each sub-problem may contain a common sub-problem). It is also allowed to make a choice through the solution of its own sub-problems. This method solves each sub-problem only once, and saves the results so as to avoid repeated calculation every time it encounters.

Therefore, the dynamic programming method has a remarkable feature, that is, the sub-problems in the corresponding sub-problem tree present a lot of repetition. The key of dynamic programming is to solve the recurring sub-problems only at the first encounter and save the answers so that they can be directly quoted in future encounters without resolving them again.

11. To explain the Recursive algorithm

Answer: The recursion is to transform a "big problem" that can not be solved directly into one or several "small problems" similar to the original problem to solve, and then further decompose these "small problems" into smaller "small problems" to solve, so as to decompose until each "small problem" can be solved directly (at this time decomposed into recursive exports). After solving the "small problem" step by step, we can return to the solution of the "big problem". Recursion is a basic method of describing and solving problems, which means that subroutines (or functions) call themselves directly or indirectly through a series of call statements. Recursion has two basic elements:

(1) Boundary conditions: determine when recursion ends;

(2) Recursive model: how to decompose big problems into small ones.

Recursive algorithm design usually has the following two steps:

(1) simple base case (or cases)—a terminating scenario that does not use recursion to produce an answer

(2) A set of rules that reduce all other cases toward the base case

三、

1. 1. Given the n elements a [0:n-1] in ascending order, it is now necessary to find a specific element X in the n elements and return its position in the array if Return-1 is not found.

The algorithm of binary search is written and its time complexity is analyzed.

```
    1. template<class Type>
    int BinarySearch(Type a[], const Type& x, int n)
    {// Search for X in a [0:n] and return to its position in the array when x is found, otherwise
return to -1
            Int left=0;   int   right=n-1;
            While (left<=right){
                int middle=(left+right)/2;
                if (x==a[middle]) return middle;
                if (x>a[middle]) left=middle+1;
                else right=middle-1;
             }
             Return -1;
    }
```

Time complexity is O (logn)

2． Greedy algorithm for knapsack problem

```
void Knapsack(int n,float M,float v[],float w[],float x[])

{
        Sort(n,v,w);
        int i;
        for (i=1;i<=n;i++) x[i]=0;
        float c=M;
        for (i=1;i<=n;i++) {
            if (w[i]>c) break;
            x[i]=1;
            c - =w[i];
            }
        if (i<=n) x[i]=c/w[i];
}
```

3． Greedy algorithm for activity selection.

```
    template<class Type>
    void GreedySelector(int n, Type s[], Type f[], bool A[])
    {
            A[1]=true;
            int j=1;
            for (int i=2;i<=n;i++) {
                if (s[i]>=f[j])
                {   A[i]=true;
                    j=i;
                  }
                else A[i]=false;
                }
    }
```

Time complexity is O (logn)

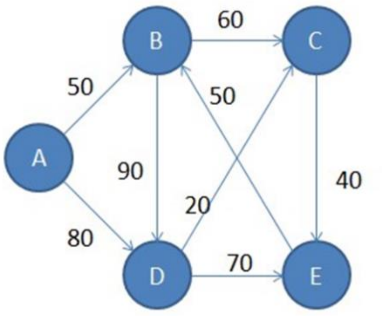
4. Using divide-and-conquer algorithm to write merge sorting algorithm and analyze its time complexity

```
void MergeSort(Type a[], int left, int right)
  {
     if (left<right) {// There are at least two elements
     int i=(left+right)/2;    // Take the middle poin
     mergeSort(a, left, i);
     mergeSort(a, i+1, right);
     merge(a, b, left, i, right);    // Merge into array B
     copy(a, b, left, right);        // Copy to array A
     }
  }
```

In the worst case, the time complexity of the algorithm is O(nlogn)

8. In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

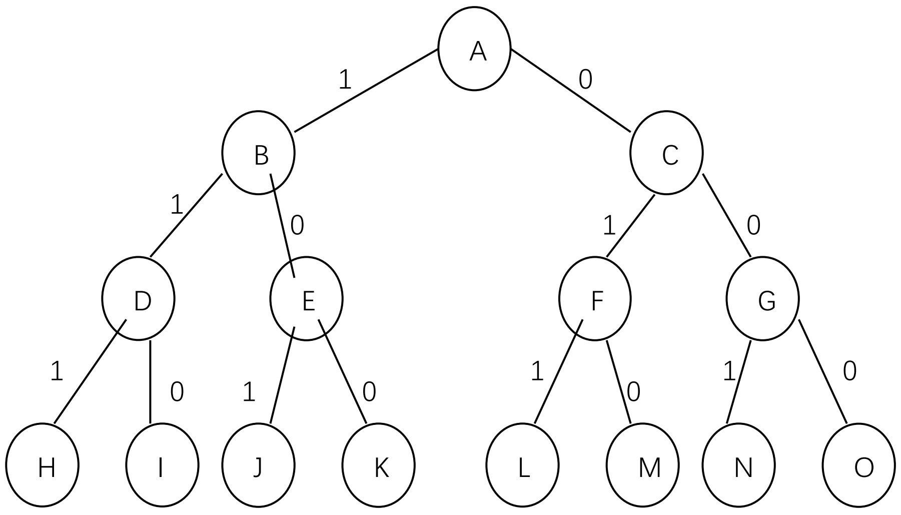
For example: The figure represents the road network between cities, and the number on the line segment represents the cost. The problem of finding the shortest cost between two cities.



9. **0/1 knapsack problem**: n=3, C=9, V={6,10,3}, W={3,4,4}. The optimal value and solution are calculated.

Solution: The solution space is composed of 0-1 vectors with length of 3. A complete binary tree is used to represent the solution space (starting from the root, left 1 and right 0).

The solution space tree is drawn and its optimal value and solution are calculated. The solution space is {(0,0,0), (0,1,0), (0,0,1), (1,0,0), (0,1,1), (1,0,1), (1,0,1), (1,1,0), (1,1,1)}. The solution space tree is following:



The optimal value of the problem is: 16. The optimal solution is: (1, 1, 0).

10. 23*789=? (calculated by dividing and conquering algorithm)。

Solution: Let X = 123 Y = 789 divide X and Y into two parts.

X=1*102+23 Y=7*102+89, A=1 B=23 C=7 ,D=89

The formula is derived from the multiplication of large integers.:

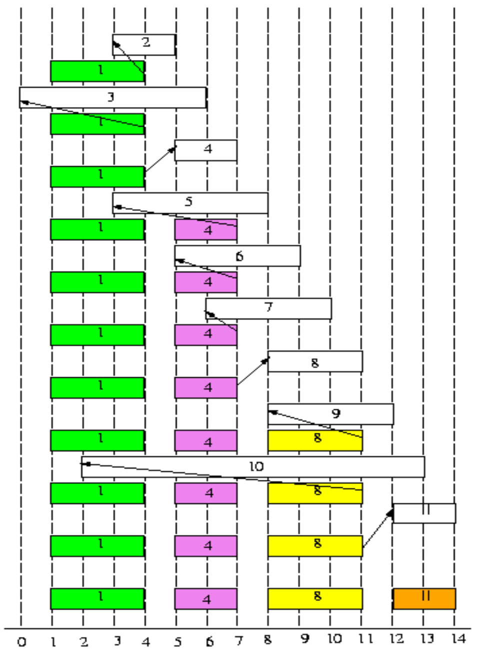
X*Y=AC104+((A-B)(D-C) +AC+BD) 102+BD

=1*23*104+(-22*82+1*23+23*89)*102+23*89

=97047

11. **Activity-Selection Problem:** Several competing activities that require exclusive use of common resources (Same place at a fixed period of time), with the goal is of selecting a maximum-size set of mutually compatible activities that wish to use a resource (a period of time in the place)

i	s_i	f_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14



1. Sort f_i

2. Select the first activity.

3. Pick the first activity i such that $s_i >= f_j$ where activity j is the most recently selected activity.

12. How to determine the calculation order of matrix continuous product, so that the order of calculation matrix continuous product requires the least number of multiplications.

Given a sequence of matrices **$A_1,A_2,\ldots,A_n$**, find an optimal order of multiplication. A1(30x35),A2(35x15),A3(15x5),A4(5x10),A5(10x20),A6(20x25) .

Solution: The optimal solution is $((A_1(A_2A_3))((A_4A_5)A_6)$

13 . A flow network, with source s and sink t. The source is *s*, and the sink *t*. The numbers denote flow and capacity.

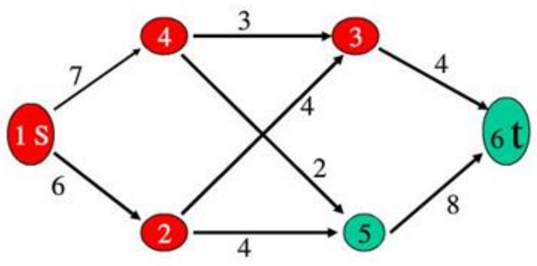


Fig.1　A flow network, with source s and sink t. The numbers next to the edge are the capacities.

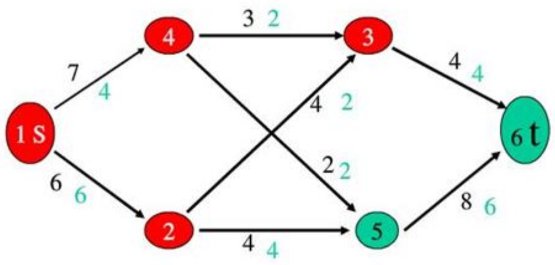


Fig.2 A network with an example of maximum flow. The source is *s*, and the sink *t*. The numbers denote flow and capacity. (the capacity of　maximum flow is 10)

13. Maximum sum of sub-arrays

Problem: Input an array of integers, positive and negative numbers in the array.

One or more integers in an array form a subarray, and each subarray has a sum.

Seek the maximum of the sum of all subarrays. The required time complexity is O (n).

For example, the input array is 1, -2, 3, 10, -4, 7, 2, -5, and the largest subarray is 3, 10, -4, 7, 2, so the output is the subarray and 18.

15. Input a positive integer n (n number of bits ≤ 200), after removing any s digits, the remaining digits will form a new positive integer in the original left-right order. Programming for a given n and s, to find a solution, so that the remaining number of new numbers constitute the smallest.
[Sample input]　149532
S = 4
[Sample output]　1432
Solution: In order to approach the target as much as possible, the greedy strategy is chose: each step always selects a number that minimizes the remaining number, that is, searches in the order from high to low. If the digits are incremented, the last digit is deleted. Otherwise, delete the first character of the first decrement interval. Then go back to the beginning of the string and delete the next number according to the above rules. Repeat the above process s times, the remaining number string is the solution to the problem.
The algorithm is as follows:

```
Input   s, n;
while ( s > 0 )
  {   i=1;   // seardh from the beginning of the string
```

```
      while (i < length(n)) && (n[i]<n[i+1])
        {i++;}
         delete(n,i,1); // Delete the ith character of the string n
        s--;
    }
while (length(n)>1)&& (n[1]='0')
    delete(n,1,1); // Delete the useless zeros that may be generated at the beginning of the string
  Ouput n;
```

13.The police arrested four suspects a, b, c and d, and only one of them was a thief.

A said, "I am not a thief."

B said, "c is a thief."

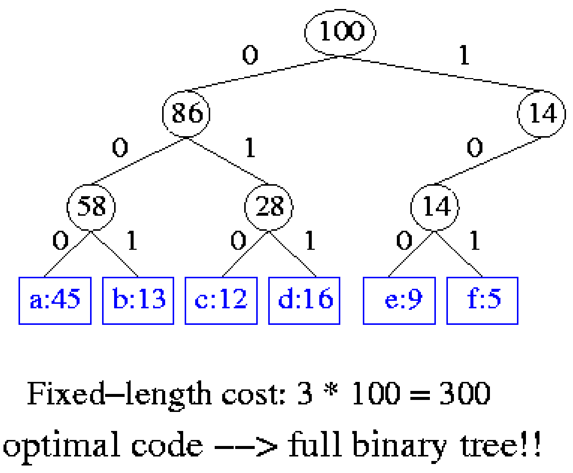
C said "The thieves must be d,"

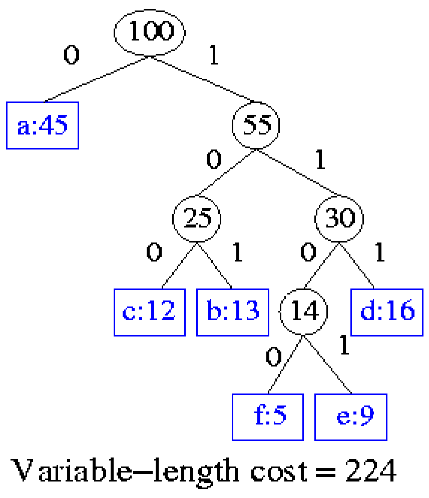
D said, "c is wronging people."

Now we knew that three of the four people told the truth and one person told a lie. Who is the thief? Write an algorithm to find the thief.

14. **Huffman codes Problem:** Suppose that you have a file of 100,000-character. To keep the example simple, suppose that each character is one of the 6 letters from *a* through *f*. Since we have just 6 characters, we need just 3-bit to represent a character, so the file requires 300,000 bits to store. Can we do better?

Solution: The idea is that we will use a variable length code instead of a fixed length code (3-bit for each character), with fewer bits to store the common characters, and more bits to store the rare characters.

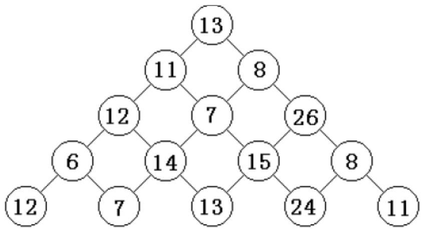

100
0
1
86
14
0
1
0
58
28
14
0
1
0
1
0
1
a:45
b:13
c:12
d:16
e:9
f:5
Fixed–length cost: 3 * 100 = 300
optimal code ––> full binary tree!!



15. Enter a string ,and ouput all strings that can be arranged all the characters in the string .
For example, the input string "a b c," then the ouput all strings that can be arranged by the characters a, b, and C. outputs strings :" abc, acb, bac,bca,cab, and cba ."

16. Starting from the top, you can choose to go left or right at each node and go to the bottom together. You need to find a path to maximize the value of the path.

```
for(r=n-2;r>=0;r--) //Bottom-up Recursive Computing
  for(c=0;      c<=r    ;c++)
    if( t[r+1][c]>t[r+1][c+1] )
    t[r][c]+=t[r+1][c]__;
       else       t[r][c]+=t[r+1][c+1]  ;
```



16. Find paths

Find all paths for a certain value in a binary tree.

Topic: enter an integer number and a binary tree.

From the root node of the tree, we have to go down until all the nodes passed through the leaf node form a path. Print all the paths that are equal to the input number.

For example: input “22”

Output two paths : “10, 12”;“10, 5, 7