# 6-11 Shortest Path [1] (25 分)

Write a program to find the unweighted shortest distances from any vertex to a given source vertex in a digraph.

### Format of functions:

void ShortestDist( LGraph Graph, int dist[], Vertex S );

where LGraph is defined as the following:

typedef struct AdjVNode \*PtrToAdjVNode;

struct AdjVNode{

Vertex AdjV;

PtrToAdjVNode Next;

};

typedef struct Vnode{

PtrToAdjVNode FirstEdge;

} AdjList[MaxVertexNum];

typedef struct GNode \*PtrToGNode;

struct GNode{

int Nv;

int Ne;

AdjList G;

};

typedef PtrToGNode LGraph;

The shortest distance from V to the source S is supposed to be stored in dist[V]. If V cannot be reached from S, store -1 instead.

### Sample program of judge:

#include <stdio.h>

#include <stdlib.h>

typedef enum {false, true} bool;

#define MaxVertexNum 10 /\* maximum number of vertices \*/

typedef int Vertex; /\* vertices are numbered from 0 to MaxVertexNum-1 \*/

typedef struct AdjVNode \*PtrToAdjVNode;

struct AdjVNode{

Vertex AdjV;

PtrToAdjVNode Next;

};

typedef struct Vnode{

PtrToAdjVNode FirstEdge;

} AdjList[MaxVertexNum];

typedef struct GNode \*PtrToGNode;

struct GNode{

int Nv;

int Ne;

AdjList G;

};

typedef PtrToGNode LGraph;

LGraph ReadG(); /\* details omitted \*/

void ShortestDist( LGraph Graph, int dist[], Vertex S );

int main()

{

int dist[MaxVertexNum];

Vertex S, V;

LGraph G = ReadG();

scanf("%d", &S);

ShortestDist( G, dist, S );

for ( V=0; V<G->Nv; V++ )

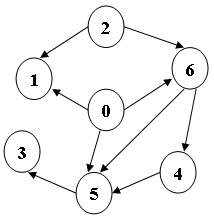
printf("%d ", dist[V]);

return 0;

}

/\* Your function will be put here \*/

### Sample Input (for the graph shown in the figure):



7 9

0 1

0 5

0 6

5 3

2 1

2 6

6 4

4 5

6 5

2

### Sample Output:

-1 1 0 3 2 2 1

# 6-12 Shortest Path [2] (25 分)

Write a program to find the weighted shortest distances from any vertex to a given source vertex in a digraph. It is guaranteed that all the weights are positive.

### Format of functions:

void ShortestDist( MGraph Graph, int dist[], Vertex S );

where MGraph is defined as the following:

typedef struct GNode \*PtrToGNode;

struct GNode{

int Nv;

int Ne;

WeightType G[MaxVertexNum][MaxVertexNum];

};

typedef PtrToGNode MGraph;

The shortest distance from V to the source S is supposed to be stored in dist[V]. If V cannot be reached from S, store -1 instead.

### Sample program of judge:

#include <stdio.h>

#include <stdlib.h>

typedef enum {false, true} bool;

#define INFINITY 1000000

#define MaxVertexNum 10 /\* maximum number of vertices \*/

typedef int Vertex; /\* vertices are numbered from 0 to MaxVertexNum-1 \*/

typedef int WeightType;

typedef struct GNode \*PtrToGNode;

struct GNode{

int Nv;

int Ne;

WeightType G[MaxVertexNum][MaxVertexNum];

};

typedef PtrToGNode MGraph;

MGraph ReadG(); /\* details omitted \*/

void ShortestDist( MGraph Graph, int dist[], Vertex S );

int main()

{

int dist[MaxVertexNum];

Vertex S, V;

MGraph G = ReadG();

scanf("%d", &S);

ShortestDist( G, dist, S );

for ( V=0; V<G->Nv; V++ )

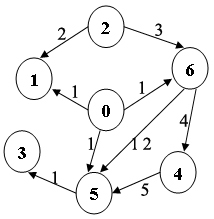
printf("%d ", dist[V]);

return 0;

}

/\* Your function will be put here \*/

### Sample Input (for the graph shown in the figure):



7 9

0 1 1

0 5 1

0 6 1

5 3 1

2 1 2

2 6 3

6 4 4

4 5 5

6 5 12

2

### Sample Output:

-1 2 0 13 7 12 3

# 7-1 Maximum Subsequence Sum（25 分）

Given a sequence of *K* integers { *N*​1​​, *N*​2​​, ..., *N*​*K*​​ }. A continuous subsequence is defined to be { *N*​*i*​​, *N*​*i*+1​​, ..., *N*​*j*​​ } where 1≤*i*≤*j*≤*K*. The Maximum Subsequence is the continuous subsequence which has the largest sum of its elements. For example, given sequence { -2, 11, -4, 13, -5, -2 }, its maximum subsequence is { 11, -4, 13 } with the largest sum being 20.

Now you are supposed to find the largest sum, together with the first and the last numbers of the maximum subsequence.

Input Specification:

Each input file contains one test case. Each case occupies two lines. The first line contains a positive integer *K* (≤10000). The second line contains *K* numbers, separated by a space.

Output Specification:

For each test case, output in one line the largest sum, together with the first and the last numbers of the maximum subsequence. The numbers must be separated by one space, but there must be no extra space at the end of a line. In case that the maximum subsequence is not unique, output the one with the smallest indices *i* and *j* (as shown by the sample case). If all the *K* numbers are negative, then its maximum sum is defined to be 0, and you are supposed to output the first and the last numbers of the whole sequence.

Sample Input:

10

-10 1 2 3 4 -5 -23 3 7 -21

Sample Output:

10 1 4

# 7-2 Reversing Linked List（25 分）

Given a constant *K* and a singly linked list *L*, you are supposed to reverse the links of every *K* elements on *L*. For example, given *L* being 1→2→3→4→5→6, if *K*=3, then you must output 3→2→1→6→5→4; if *K*=4, you must output 4→3→2→1→5→6.

### Input Specification:

Each input file contains one test case. For each case, the first line contains the address of the first node, a positive *N* (≤10​5​​) which is the total number of nodes, and a positive *K* (≤*N*) which is the length of the sublist to be reversed. The address of a node is a 5-digit nonnegative integer, and NULL is represented by -1.

Then *N* lines follow, each describes a node in the format:

Address Data Next

where Address is the position of the node, Data is an integer, and Next is the position of the next node.

### Output Specification:

For each case, output the resulting ordered linked list. Each node occupies a line, and is printed in the same format as in the input.

### Sample Input:

00100 6 4

00000 4 99999

00100 1 12309

68237 6 -1

33218 3 00000

99999 5 68237

12309 2 33218

### Sample Output:

00000 4 33218

33218 3 12309

12309 2 00100

00100 1 99999

99999 5 68237

68237 6 -1

# 7-3 Pop Sequence（25 分）

Given a stack which can keep *M* numbers at most. Push *N* numbers in the order of 1, 2, 3, ..., *N* and pop randomly. You are supposed to tell if a given sequence of numbers is a possible pop sequence of the stack. For example, if *M* is 5 and *N* is 7, we can obtain 1, 2, 3, 4, 5, 6, 7 from the stack, but not 3, 2, 1, 7, 5, 6, 4.

### Input Specification:

Each input file contains one test case. For each case, the first line contains 3 numbers (all no more than 1000): *M* (the maximum capacity of the stack), *N* (the length of push sequence), and *K* (the number of pop sequences to be checked). Then *K* lines follow, each contains a pop sequence of *N* numbers. All the numbers in a line are separated by a space.

### Output Specification:

For each pop sequence, print in one line "YES" if it is indeed a possible pop sequence of the stack, or "NO" if not.

### Sample Input:

5 7 5

1 2 3 4 5 6 7

3 2 1 7 5 6 4

7 6 5 4 3 2 1

5 6 4 3 7 2 1

1 7 6 5 4 3 2

### Sample Output:

YES

NO

NO

YES

NO

# 7-4 List Leaves（25 分）

Given a tree, you are supposed to list all the leaves in the order of top down, and left to right.

### Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer *N* (≤10) which is the total number of nodes in the tree -- and hence the nodes are numbered from 0 to *N*−1. Then *N* lines follow, each corresponds to a node, and gives the indices of the left and right children of the node. If the child does not exist, a "-" will be put at the position. Any pair of children are separated by a space.

### Output Specification:

For each test case, print in one line all the leaves' indices in the order of top down, and left to right. There must be exactly one space between any adjacent numbers, and no extra space at the end of the line.

### Sample Input:

8

1 -

- -

0 -

2 7

- -

- -

5 -

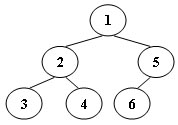
4 6

### Sample Output:

4 1 5

# 7-5 Tree Traversals Again（25 分）

An inorder binary tree traversal can be implemented in a non-recursive way with a stack. For example, suppose that when a 6-node binary tree (with the keys numbered from 1 to 6) is traversed, the stack operations are: push(1); push(2); push(3); pop(); pop(); push(4); pop(); pop(); push(5); push(6); pop(); pop(). Then a unique binary tree (shown in Figure 1) can be generated from this sequence of operations. Your task is to give the postorder traversal sequence of this tree.

  
Figure 1

### Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer *N* (≤30) which is the total number of nodes in a tree (and hence the nodes are numbered from 1 to *N*). Then 2*N* lines follow, each describes a stack operation in the format: "Push X" where X is the index of the node being pushed onto the stack; or "Pop" meaning to pop one node from the stack.

### Output Specification:

For each test case, print the postorder traversal sequence of the corresponding tree in one line. A solution is guaranteed to exist. All the numbers must be separated by exactly one space, and there must be no extra space at the end of the line.

### Sample Input:

6

Push 1

Push 2

Push 3

Pop

Pop

Push 4

Pop

Pop

Push 5

Push 6

Pop

Pop

### Sample Output:

3 4 2 6 5 1

# 7-6 Root of AVL Tree（25 分）

An AVL tree is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Figures 1-4 illustrate the rotation rules.

Now given a sequence of insertions, you are supposed to tell the root of the resulting AVL tree.

### Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer *N* (≤20) which is the total number of keys to be inserted. Then *N* distinct integer keys are given in the next line. All the numbers in a line are separated by a space.

### Output Specification:

For each test case, print the root of the resulting AVL tree in one line.

### Sample Input 1:

5

88 70 61 96 120

### Sample Output 1:

70

### Sample Input 2:

7

88 70 61 96 120 90 65

### Sample Output 2:

88

# 7-7 Complete Binary Search Tree（30 分）

A Binary Search Tree (BST) is recursively defined as a binary tree which has the following properties:

 The left subtree of a node contains only nodes with keys less than the node's key.

 The right subtree of a node contains only nodes with keys greater than or equal to the node's key.

 Both the left and right subtrees must also be binary search trees.

A Complete Binary Tree (CBT) is a tree that is completely filled, with the possible exception of the bottom level, which is filled from left to right.

Now given a sequence of distinct non-negative integer keys, a unique BST can be constructed if it is required that the tree must also be a CBT. You are supposed to output the level order traversal sequence of this BST.

### Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer *N* (≤1000). Then *N* distinct non-negative integer keys are given in the next line. All the numbers in a line are separated by a space and are no greater than 2000.

### Output Specification:

For each test case, print in one line the level order traversal sequence of the corresponding complete binary search tree. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

### Sample Input:

10

1 2 3 4 5 6 7 8 9 0

### Sample Output:

6 3 8 1 5 7 9 0 2 4

# 7-8 File Transfer（25 分）

We have a network of computers and a list of bi-directional connections. Each of these connections allows a file transfer from one computer to another. Is it possible to send a file from any computer on the network to any other?

### Input Specification:

Each input file contains one test case. For each test case, the first line contains *N* (2≤*N*≤10​4​​), the total number of computers in a network. Each computer in the network is then represented by a positive integer between 1 and *N*. Then in the following lines, the input is given in the format:

I c1 c2

where I stands for inputting a connection between c1 and c2; or

C c1 c2

where C stands for checking if it is possible to transfer files between c1 and c2; or

S

where S stands for stopping this case.

### Output Specification:

For each C case, print in one line the word "yes" or "no" if it is possible or impossible to transfer files between c1 and c2, respectively. At the end of each case, print in one line "The network is connected." if there is a path between any pair of computers; or "There are k components." where k is the number of connected components in this network.

### Sample Input 1:

5

C 3 2

I 3 2

C 1 5

I 4 5

I 2 4

C 3 5

S

### Sample Output 1:

no

no

yes

There are 2 components.

### Sample Input 2:

5

C 3 2

I 3 2

C 1 5

I 4 5

I 2 4

C 3 5

I 1 3

C 1 5

S

### Sample Output 2:

no

no

yes

yes

The network is connected.

# 7-9 Huffman Codes（30 分）

In 1953, David A. Huffman published his paper "A Method for the Construction of Minimum-Redundancy Codes", and hence printed his name in the history of computer science. As a professor who gives the final exam problem on Huffman codes, I am encountering a big problem: the Huffman codes are NOT unique. For example, given a string "aaaxuaxz", we can observe that the frequencies of the characters 'a', 'x', 'u' and 'z' are 4, 2, 1 and 1, respectively. We may either encode the symbols as {'a'=0, 'x'=10, 'u'=110, 'z'=111}, or in another way as {'a'=1, 'x'=01, 'u'=001, 'z'=000}, both compress the string into 14 bits. Another set of code can be given as {'a'=0, 'x'=11, 'u'=100, 'z'=101}, but {'a'=0, 'x'=01, 'u'=011, 'z'=001} is NOT correct since "aaaxuaxz" and "aazuaxax" can both be decoded from the code 00001011001001. The students are submitting all kinds of codes, and I need a computer program to help me determine which ones are correct and which ones are not.

### Input Specification:

Each input file contains one test case. For each case, the first line gives an integer *N* (2≤*N*≤63), then followed by a line that contains all the *N* distinct characters and their frequencies in the following format:

c[1] f[1] c[2] f[2] ... c[N] f[N]

where c[i] is a character chosen from {'0' - '9', 'a' - 'z', 'A' - 'Z', '\_'}, and f[i] is the frequency of c[i] and is an integer no more than 1000. The next line gives a positive integer *M* (≤1000), then followed by *M* student submissions. Each student submission consists of *N* lines, each in the format:

c[i] code[i]

where c[i] is the i-th character and code[i] is an non-empty string of no more than 63 '0's and '1's.

### Output Specification:

For each test case, print in each line either "Yes" if the student's submission is correct, or "No" if not.

Note: The optimal solution is not necessarily generated by Huffman algorithm. Any prefix code with code length being optimal is considered correct.

### Sample Input:

7

A 1 B 1 C 1 D 3 E 3 F 6 G 6

4

A 00000

B 00001

C 0001

D 001

E 01

F 10

G 11

A 01010

B 01011

C 0100

D 011

E 10

F 11

G 00

A 000

B 001

C 010

D 011

E 100

F 101

G 110

A 00000

B 00001

C 0001

D 001

E 00

F 10

G 11

### Sample Output:

Yes

Yes

No

No

# 7-10 Saving James Bond - Easy Version（25 分）

This time let us consider the situation in the movie "Live and Let Die" in which James Bond, the world's most famous spy, was captured by a group of drug dealers. He was sent to a small piece of land at the center of a lake filled with crocodiles. There he performed the most daring action to escape -- he jumped onto the head of the nearest crocodile! Before the animal realized what was happening, James jumped again onto the next big head... Finally he reached the bank before the last crocodile could bite him (actually the stunt man was caught by the big mouth and barely escaped with his extra thick boot).

Assume that the lake is a 100 by 100 square one. Assume that the center of the lake is at (0,0) and the northeast corner at (50,50). The central island is a disk centered at (0,0) with the diameter of 15. A number of crocodiles are in the lake at various positions. Given the coordinates of each crocodile and the distance that James could jump, you must tell him whether or not he can escape.

### Input Specification:

Each input file contains one test case. Each case starts with a line containing two positive integers *N* (≤100), the number of crocodiles, and *D*, the maximum distance that James could jump. Then *N* lines follow, each containing the (*x*,*y*) location of a crocodile. Note that no two crocodiles are staying at the same position.

### Output Specification:

For each test case, print in a line "Yes" if James can escape, or "No" if not.

### Sample Input 1:

14 20

25 -15

-25 28

8 49

29 15

-35 -2

5 28

27 -29

-8 -28

-20 -35

-25 -20

-13 29

-30 15

-35 40

12 12

### Sample Output 1:

Yes

### Sample Input 2:

4 13

-12 12

12 12

-12 -12

12 -12

### Sample Output 2:

No

# 7-11 Saving James Bond - Hard Version（30 分）

This time let us consider the situation in the movie "Live and Let Die" in which James Bond, the world's most famous spy, was captured by a group of drug dealers. He was sent to a small piece of land at the center of a lake filled with crocodiles. There he performed the most daring action to escape -- he jumped onto the head of the nearest crocodile! Before the animal realized what was happening, James jumped again onto the next big head... Finally he reached the bank before the last crocodile could bite him (actually the stunt man was caught by the big mouth and barely escaped with his extra thick boot).

Assume that the lake is a 100 by 100 square one. Assume that the center of the lake is at (0,0) and the northeast corner at (50,50). The central island is a disk centered at (0,0) with the diameter of 15. A number of crocodiles are in the lake at various positions. Given the coordinates of each crocodile and the distance that James could jump, you must tell him a shortest path to reach one of the banks. The length of a path is the number of jumps that James has to make.

### Input Specification:

Each input file contains one test case. Each case starts with a line containing two positive integers *N* (≤100), the number of crocodiles, and *D*, the maximum distance that James could jump. Then *N* lines follow, each containing the (*x*,*y*) location of a crocodile. Note that no two crocodiles are staying at the same position.

### Output Specification:

For each test case, if James can escape, output in one line the minimum number of jumps he must make. Then starting from the next line, output the position (*x*,*y*) of each crocodile on the path, each pair in one line, from the island to the bank. If it is impossible for James to escape that way, simply give him 0 as the number of jumps. If there are many shortest paths, just output the one with the minimum first jump, which is guaranteed to be unique.

### Sample Input 1:

17 15

10 -21

10 21

-40 10

30 -50

20 40

35 10

0 -10

-25 22

40 -40

-30 30

-10 22

0 11

25 21

25 10

10 10

10 35

-30 10

### Sample Output 1:

4

0 11

10 21

10 35

### Sample Input 2:

4 13

-12 12

12 12

-12 -12

12 -12

### Sample Output 2:

0

# 7-12 How Long Does It Take（25 分）

Given the relations of all the activities of a project, you are supposed to find the earliest completion time of the project.

### Input Specification:

Each input file contains one test case. Each case starts with a line containing two positive integers *N* (≤100), the number of activity check points (hence it is assumed that the check points are numbered from 0 to *N*−1), and *M*, the number of activities. Then *M* lines follow, each gives the description of an activity. For the i-th activity, three non-negative numbers are given: S[i], E[i], and L[i], where S[i] is the index of the starting check point, E[i] of the ending check point, and L[i] the lasting time of the activity. The numbers in a line are separated by a space.

### Output Specification:

For each test case, if the scheduling is possible, print in a line its earliest completion time; or simply output "Impossible".

### Sample Input 1:

9 12

0 1 6

0 2 4

0 3 5

1 4 1

2 4 1

3 5 2

5 4 0

4 6 9

4 7 7

5 7 4

6 8 2

7 8 4

### Sample Output 1:

18

### Sample Input 2:

4 5

0 1 1

0 2 2

2 1 3

1 3 4

3 2 5

### Sample Output 2:

Impossible

# 7-13 Insert or Merge（25 分）

According to Wikipedia:

**Insertion sort** iterates, consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

**Merge sort** works as follows: Divide the unsorted list into N sublists, each containing 1 element (a list of 1 element is considered sorted). Then repeatedly merge two adjacent sublists to produce new sorted sublists until there is only 1 sublist remaining.

Now given the initial sequence of integers, together with a sequence which is a result of several iterations of some sorting method, can you tell which sorting method we are using?

### Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer *N* (≤100). Then in the next line, *N* integers are given as the initial sequence. The last line contains the partially sorted sequence of the *N* numbers. It is assumed that the target sequence is always ascending. All the numbers in a line are separated by a space.

### Output Specification:

For each test case, print in the first line either "Insertion Sort" or "Merge Sort" to indicate the method used to obtain the partial result. Then run this method for one more iteration and output in the second line the resuling sequence. It is guaranteed that the answer is unique for each test case. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

### Sample Input 1:

10

3 1 2 8 7 5 9 4 6 0

1 2 3 7 8 5 9 4 6 0

### Sample Output 1:

Insertion Sort

1 2 3 5 7 8 9 4 6 0

### Sample Input 2:

10

3 1 2 8 7 5 9 4 0 6

1 3 2 8 5 7 4 9 0 6

### Sample Output 2:

Merge Sort

1 2 3 8 4 5 7 9 0 6

# 7-14 Insertion or Heap Sort（25 分）

According to Wikipedia:

**Insertion sort** iterates, consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

**Heap sort** divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. it involves the use of a heap data structure rather than a linear-time search to find the maximum.

Now given the initial sequence of integers, together with a sequence which is a result of several iterations of some sorting method, can you tell which sorting method we are using?

### Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer *N* (≤100). Then in the next line, *N* integers are given as the initial sequence. The last line contains the partially sorted sequence of the *N* numbers. It is assumed that the target sequence is always ascending. All the numbers in a line are separated by a space.

### Output Specification:

For each test case, print in the first line either "Insertion Sort" or "Heap Sort" to indicate the method used to obtain the partial result. Then run this method for one more iteration and output in the second line the resuling sequence. It is guaranteed that the answer is unique for each test case. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

### Sample Input 1:

10

3 1 2 8 7 5 9 4 6 0

1 2 3 7 8 5 9 4 6 0

### Sample Output 1:

Insertion Sort

1 2 3 5 7 8 9 4 6 0

### Sample Input 2:

10

3 1 2 8 7 5 9 4 6 0

6 4 5 1 0 3 2 7 8 9

### Sample Output 2:

Heap Sort

5 4 3 1 0 2 6 7 8 9