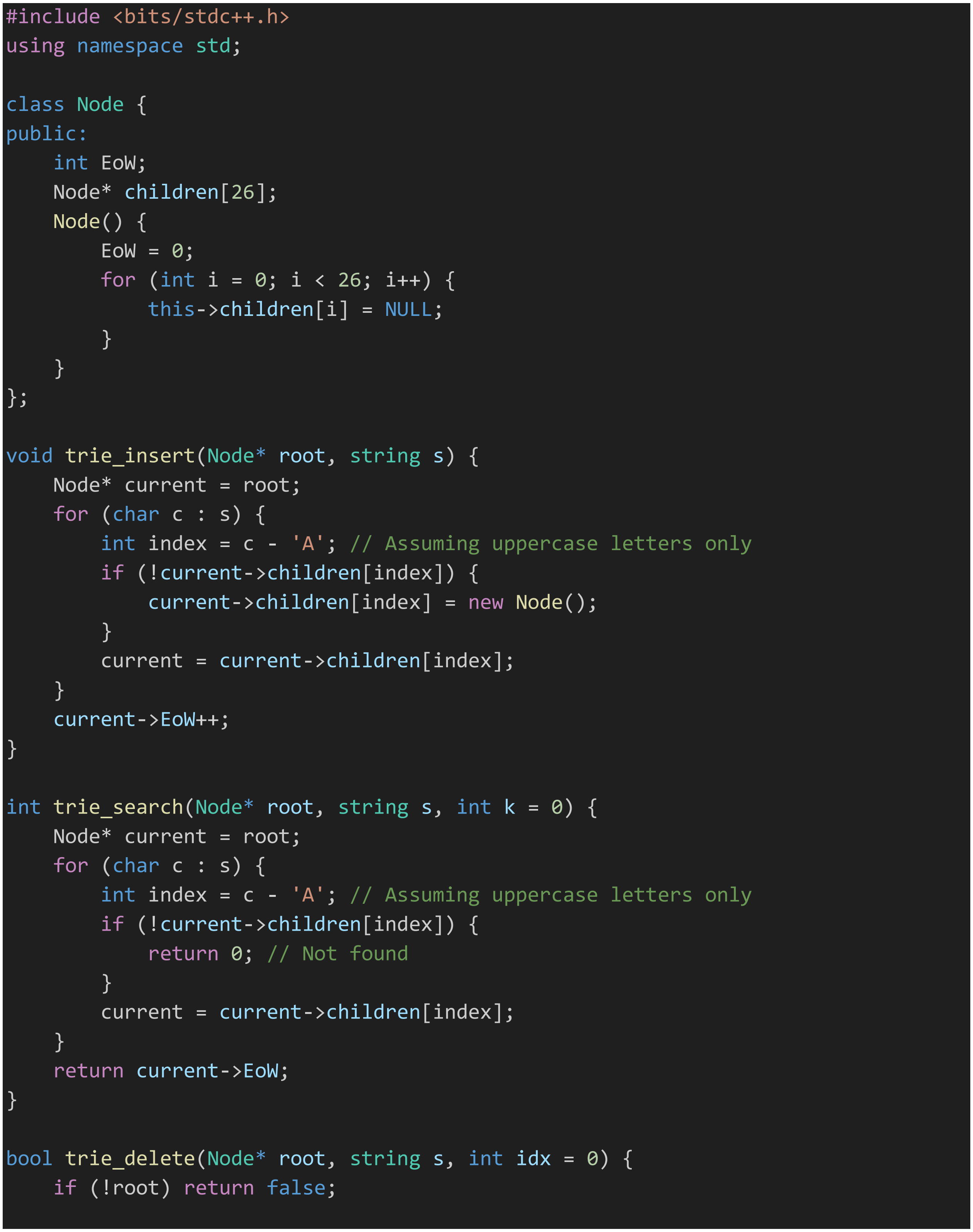
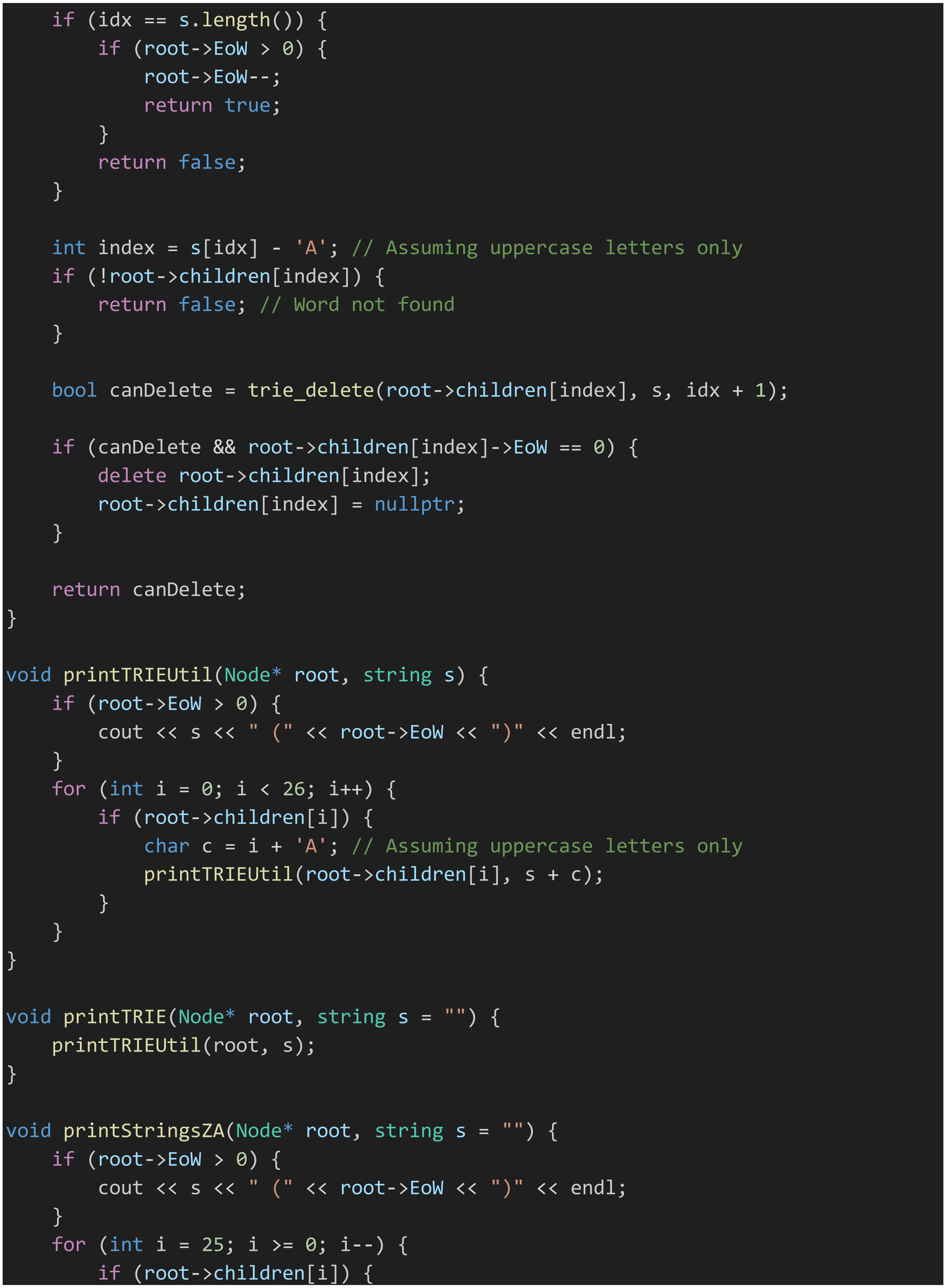
Eval 2

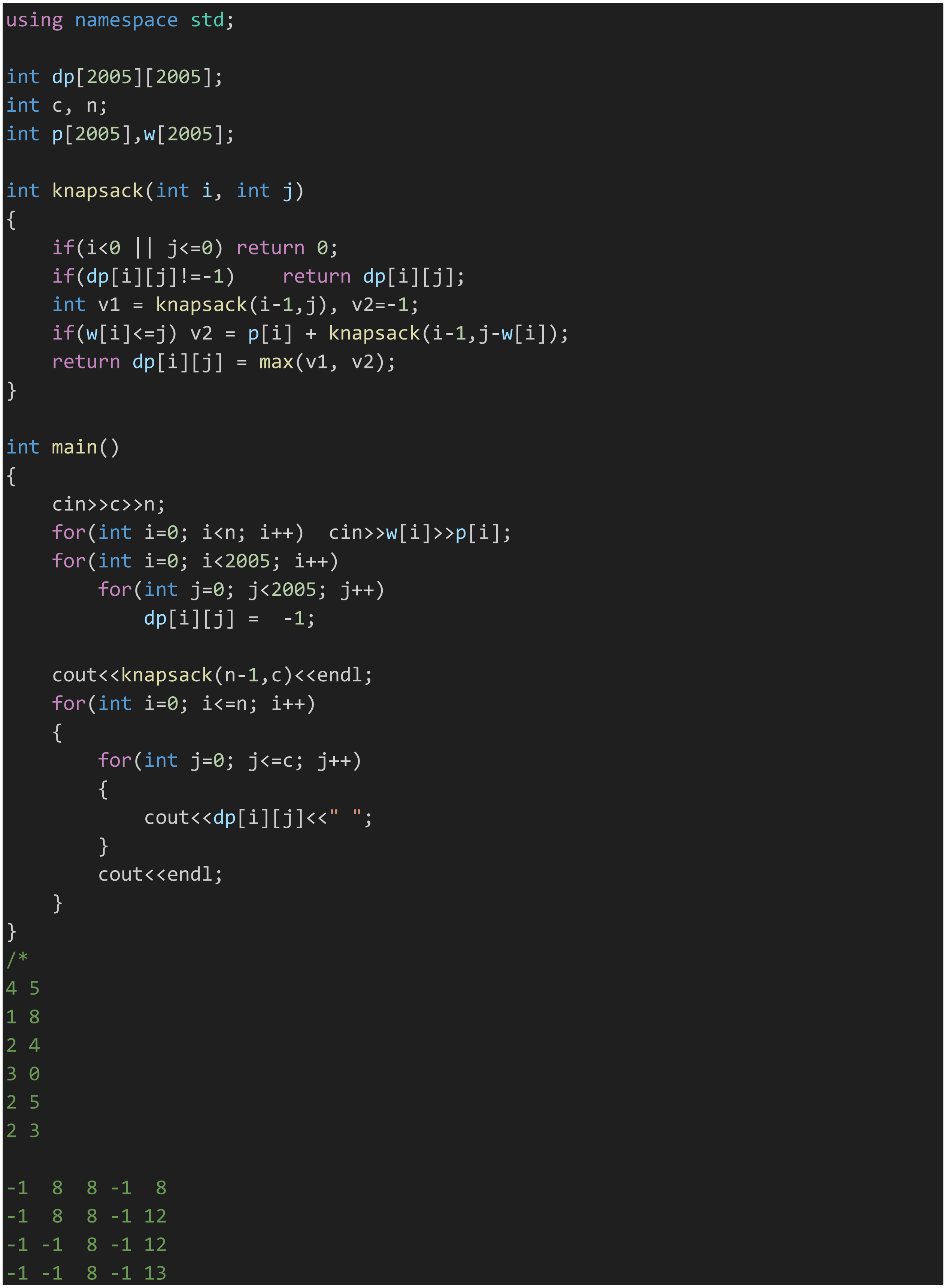


```cpp
#include <bits/stdc++.h>
using namespace std;

class Node {
public:
    int EoW;
    Node* children[26];
    Node() {
        EoW = 0;
        for (int i = 0; i < 26; i++) {
            this->children[i] = NULL;
        }
    }
};

void trie_insert(Node* root, string s) {
    Node* current = root;
    for (char c : s) {
        int index = c - 'A'; // Assuming uppercase letters only
        if (!current->children[index]) {
            current->children[index] = new Node();
        }
        current = current->children[index];
    }
    current->EoW++;
}

int trie_search(Node* root, string s, int k = 0) {
    Node* current = root;
    for (char c : s) {
        int index = c - 'A'; // Assuming uppercase letters only
        if (!current->children[index]) {
            return 0; // Not found
        }
        current = current->children[index];
    }
    return current->EoW;
}

bool trie_delete(Node* root, string s, int idx = 0) {
    if (!root) return false;
```



```
    if (idx == s.length()) {
        if (root->EoW > 0) {
            root->EoW--;
            return true;
        }
        return false;
    }

    int index = s[idx] - 'A'; // Assuming uppercase letters only
    if (!root->children[index]) {
        return false; // Word not found
    }

    bool canDelete = trie_delete(root->children[index], s, idx + 1);

    if (canDelete && root->children[index]->EoW == 0) {
        delete root->children[index];
        root->children[index] = nullptr;
    }

    return canDelete;
}

void printTRIEUtil(Node* root, string s) {
    if (root->EoW > 0) {
        cout << s << " (" << root->EoW << ")" << endl;
    }
    for (int i = 0; i < 26; i++) {
        if (root->children[i]) {
            char c = i + 'A'; // Assuming uppercase letters only
            printTRIEUtil(root->children[i], s + c);
        }
    }
}

void printTRIE(Node* root, string s = "") {
    printTRIEUtil(root, s);
}

void printStringsZA(Node* root, string s = "") {
    if (root->EoW > 0) {
        cout << s << " (" << root->EoW << ")" << endl;
    }
    for (int i = 25; i >= 0; i--) {
        if (root->children[i]) {
            char c = i + 'A'; // Assuming uppercase letters only
            printStringsZA(root->children[i], s + c);
        }
    }
}

void printPrefixStrings(Node* root, string prefix, string s = "") {
    if (prefix.length() > 0 && s != prefix) return;

    if (root->EoW > 0) {
        cout << s << " (" << root->EoW << ")" << endl;
    }

    for (int i = 0; i < 26; i++) {
        if (root->children[i]) {
            char c = i + 'A'; // Assuming uppercase letters only
            printPrefixStrings(root->children[i], prefix, s + c);
        }
    }
}

void printDuplicateStrings(Node* root, string s = "") {
    if (root->EoW > 1) {
        cout << s << " (" << root->EoW << ")" << endl;
    }

    for (int i = 0; i < 26; i++) {
        if (root->children[i]) {
            char c = i + 'A'; // Assuming uppercase letters only
            printDuplicateStrings(root->children[i], s + c);
        }
    }
}

int main() {
    Node* root = new Node();

    while (1) {
        cout << "1. Insert    2. Search    3. Delete   4. Lexicographical Sorting  5. Display Strings (Z to A)"
             << "  6. Print Strings with Prefix  7. Print Duplicate Strings  8. End"
             << endl
             << endl;
        int choice;
        string x;
        cin >> choice;
        if (choice == 1) {
            cout << "Insert String: ";
            cin >> x;
            trie_insert(root, x);
            cout << x << " is inserted in the trie" << endl;
        } else if (choice == 2) {
            cout << "Enter string to search: ";
            cin >> x;
            if (trie_search(root, x) > 0)
                cout << x << " FOUND " << endl;
            else
                cout << x << " NOT FOUND " << endl;
        } else if (choice == 3) {
            cout << "Enter string to delete: ";
            cin >> x;
            if (trie_delete(root, x))
                cout << x << " DELETED " << endl;
            else
                cout << x << " NOT FOUND " << endl;
        } else if (choice == 4) {
            printTRIE(root);
        } else if (choice == 5) {
            printStringsZA(root);
        } else if (choice == 6) {
            cout << "Enter prefix: ";
            cin >> x;
            printPrefixStrings(root, x);
        } else if (choice == 7) {
            printDuplicateStrings(root);
        } else if (choice == 8) {
            break;
        } else {
            cout << "Invalid Choice" << endl;
            break;
        }
        cout << endl;
    }

    return 0;
}
```

```
#include<bits/stdc++.h>
```



```
using namespace std;

int dp[2005][2005];
int c, n;
int p[2005],w[2005];

int knapsack(int i, int j)
{
    if(i<0 || j<=0) return 0;
    if(dp[i][j]!=-1)    return dp[i][j];
    int v1 = knapsack(i-1,j), v2=-1;
    if(w[i]<=j) v2 = p[i] + knapsack(i-1,j-w[i]);
    return dp[i][j] = max(v1, v2);
}

int main()
{
    cin>>c>>n;
    for(int i=0; i<n; i++)  cin>>w[i]>>p[i];
    for(int i=0; i<2005; i++)
        for(int j=0; j<2005; j++)
            dp[i][j] =  -1;

    cout<<knapsack(n-1,c)<<endl;
    for(int i=0; i<=n; i++)
    {
        for(int j=0; j<=c; j++)
        {
            cout<<dp[i][j]<<" ";
        }
        cout<<endl;
    }
}
/*
4 5
1 8
2 4
3 0
2 5
2 3

-1  8  8 -1  8
-1  8  8 -1 12
-1 -1  8 -1 12
-1 -1  8 -1 13
```

```
-1 -1 -1 -1 13
-1 -1 -1 -1 -1
*/
```

Knapsack problems

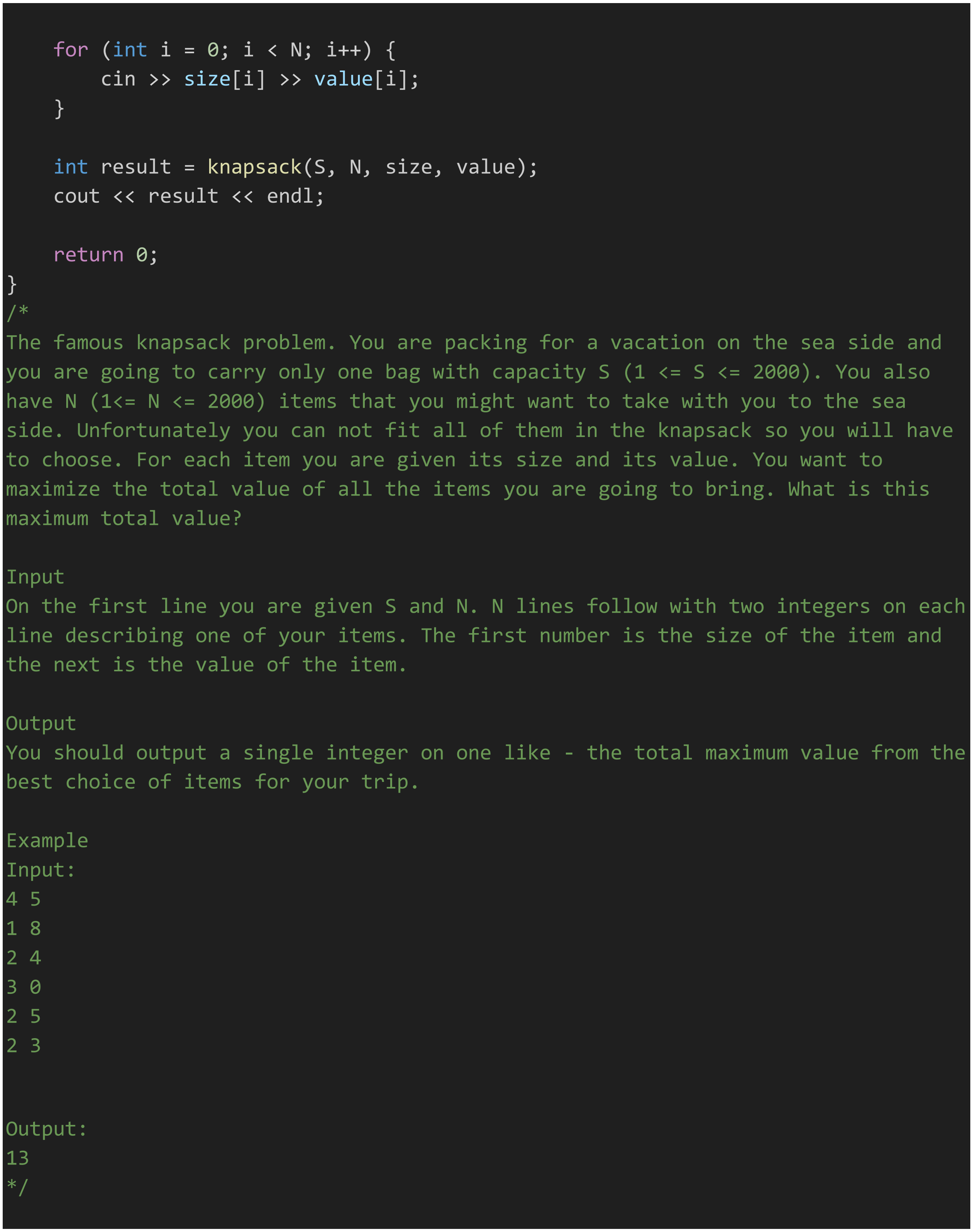
A

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int n,W;
    cin>>W>>n;
    int cost[n];
    for(int i=0; i<n; ++i){
        cin>>cost[i];
    }
    sort(cost,cost+n);
    int res=0;
    for(int i=n-1; i>=max(n-W,0); --i){
        res+=cost[i];
    }
    cout<<res;
    return 0;
}
/*
Vasya is going to hike with fellow programmers and decided to take a responsible
approach to the choice of what he will take with him. Vasya has n things that he
could take with him in his knapsack. Every thing weighs 1 kilogram. Things have
different "usefulness" for Vasya.

The hiking is going to be very long, so Vasya would like to carry a knapsack of
weight no more than w kilo.

Help him to determine the total "usefulness" of things in his knapsack if the
weight of backpack can be no more than w kilo.

Input data
The first line contains integers w и n (1 ≤ w, n ≤ 20). The second line contains
n integers c[i] (1 ≤ c[i] ≤ 1000) - the "usefulness" for each thing.

Output data
Print the total "usefulness" of things that Vasya can take with him.
```

```
Examples
Input example #1
2 3
1 5 3
Output example #1
8
Input example #2
3 2
3 2
Output example #2
5

Generate the code in c++, make sure to get it accepted.
*/
```

B

```cpp
#include <iostream>
#include <vector>
using namespace std;

int knapsack(int S, int N, vector<int>& size, vector<int>& value) {
    vector<vector<int> > dp(N + 1, vector<int>(S + 1, 0));

    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= S; j++) {
            if (size[i - 1] <= j) {
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - size[i - 1]] + value[i - 1]);
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }

    return dp[N][S];
}

int main() {
    int S, N;
    cin >> S >> N;

    vector<int> size(N);
    vector<int> value(N);
```



```cpp
    for (int i = 0; i < N; i++) {
        cin >> size[i] >> value[i];
    }

    int result = knapsack(S, N, size, value);
    cout << result << endl;

    return 0;
}
/*
The famous knapsack problem. You are packing for a vacation on the sea side and
you are going to carry only one bag with capacity S (1 <= S <= 2000). You also
have N (1<= N <= 2000) items that you might want to take with you to the sea
side. Unfortunately you can not fit all of them in the knapsack so you will have
to choose. For each item you are given its size and its value. You want to
maximize the total value of all the items you are going to bring. What is this
maximum total value?

Input
On the first line you are given S and N. N lines follow with two integers on each
line describing one of your items. The first number is the size of the item and
the next is the value of the item.

Output
You should output a single integer on one like - the total maximum value from the
best choice of items for your trip.

Example
Input:
4 5
1 8
2 4
3 0
2 5
2 3


Output:
13
*/
```

C

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int s, n;
    cin >> s >> n;

    vector<int> weight(n);
    vector<int> value(n);

    for (int i = 0; i < n; i++) {
        cin >> weight[i] >> value[i];
    }

    vector<vector<int> > dp(n + 1, vector<int>(s + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int j = 0; j <= s; j++) {
            dp[i][j] = dp[i - 1][j];
            if (j >= weight[i - 1]) {
                dp[i][j] = max(dp[i][j], dp[i - 1][j - weight[i - 1]] + value[i -
1]);
            }
        }
    }

    cout << dp[n][s] << endl;

    return 0;
}
/*
Entering into the cave with treasures, our Aladdin did not take an old blackened
lamp. He rushed to collect the gold coins and precious stones into his knapsack.
He would, of course, take everything, but miracles do not happen - too much
weight the knapsack can not hold.

Many times he laid out one thing and put others in their place, trying to raise
the value of the jewels as high as possible.

Determine the maximum value of weight that Aladdin can put in his knapsack.

We will assume that in the cave there are objects of
```

```
�
n different types, the number of objects of each type is not limited. The maximum
weight that a knapsack can hold is
�
s. Each item of type
�
i has the weight
�
�
w
i

  and cost
�
�
(
�
=
1
,
2
,
.
.
.
,
�
)
v
i

 (i=1,2,...,n).

Input data
First line contains two integers
�
s and
�
(
1
≤
�
≤
250
,
1
≤
�
≤
35
)
n(1≤s≤250,1≤n≤35) — the maximum possible weight of items in the knapsack and the
number of types of items. Each of the next
�
n lines contains two numbers
�
�
w
i

  and
�
�
(
1
≤
�
�
≤
250
,
1
≤
�
�
≤
250
)
v
i

 (1≤w
i

 ≤250,1≤v
i

 ≤250) — the weight of item of type
�
i and its cost.
```

```
Output data
Print the maximum value of the loading, which weight does not exceed
�
s.

Sample 1
Inputcopy   Outputcopy
10 2
5 10
6 19
20
*/
```

D

```
#include <iostream>
#include <vector>
using namespace std;

int superSale(int N, vector<int>& price, vector<int>& weight, int G, vector<int>&
maxWeight) {
    vector<vector<int> > dp(N + 1, vector<int>(31, 0));

    for (int i = 1; i <= N; i++) {
        for (int w = 1; w <= 30; w++) {
            if (weight[i - 1] <= w) {
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weight[i - 1]] +
price[i - 1]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    int result = 0;
    for (int i = 0; i < G; i++) {
        result += dp[N][maxWeight[i]];
    }

    return result;
}

int main() {
    int T;
    cin >> T;

    while (T--) {
        int N;
        cin >> N;

        vector<int> price(N);
        vector<int> weight(N);

        for (int i = 0; i < N; i++) {
            cin >> price[i] >> weight[i];
        }

        int G;
        cin >> G;

        vector<int> maxWeight(G);

        for (int i = 0; i < G; i++) {
            cin >> maxWeight[i];
        }

        int result = superSale(N, price, weight, G, maxWeight);
        cout << result << endl;
    }

    return 0;
}
/*
There is a SuperSale in a SuperHiperMarket. Every person can take only one object
of each kind, i.e.
one TV, one carrot, but for extra low price. We are going with a whole family to
that SuperHiperMarket.
Every person can take as many objects, as he/she can carry out from the
SuperSale. We have given
list of objects with prices and their weight. We also know, what is the maximum
weight that every
person can stand. What is the maximal value of objects we can buy at SuperSale?
Input
The input consists of T test cases. The number of them (1 ≤ T ≤ 1000) is given on
the first line of
the input file. Each test case begins with a line containing a single integer
number N that indicates
```

```
the number of objects (1 ≤ N ≤ 1000). Then follows N lines, each containing two
integers: P and W.
The first integer (1 ≤ P ≤ 100) corresponds to the price of object. The second
integer (1 ≤ W ≤ 30)
corresponds to the weight of object. Next line contains one integer (1 ≤ G ≤ 100)
its the number of
people in our group. Next G lines contains maximal weight (1 ≤ MW ≤ 30) that can
stand this i-th
person from our family (1 ≤ i ≤ G).
Output
For every test case your program has to determine one integer. Print out the
maximal value of goods
which we can buy with that family.
Sample Input
2
3
72 17
44 23
31 24
1
26
6
64 26
85 22
52 4
99 18
39 13
54 9
4
23
20
20
26
Sample Output
72
514
*/
```

E

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int main() {
    while (true) {
        int budget, n;
        cin >> budget >> n;

        if (budget == 0 && n == 0) {
            break;
        }

        vector<int> entranceFee(n);
        vector<int> funValue(n);

        for (int i = 0; i < n; i++) {
            cin >> entranceFee[i] >> funValue[i];
        }

        vector<vector<int> > dp(n + 1, vector<int>(budget + 1, 0));

        for (int i = 1; i <= n; i++) {
            for (int j = 0; j <= budget; j++) {
                if (entranceFee[i - 1] <= j) {
                    dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - entranceFee[i -
1]] + funValue[i - 1]);
                } else {
                    dp[i][j] = dp[i - 1][j];
                }
            }
        }

        int maxFun = dp[n][budget];
        int minCost = budget;

        while (dp[n][minCost - 1] == maxFun) {
            minCost--;
        }

        cout << minCost << " " << maxFun << endl;
    }

    return 0;
}
/*
You just received another bill which you cannot pay because you lack the money.
Unfortunately, this is not the first time to happen, and now you decide to
investigate the cause of your constant monetary shortness. The reason is quite
```

```
obvious: the lion's share of your money routinely disappears at the entrance of
party localities.

You make up your mind to solve the problem where it arises, namely at the parties
themselves. You introduce a limit for your party budget and try to have the most
possible fun with regard to this limit.

You inquire beforehand about the entrance fee to each party and estimate how much
fun you might have there. The list is readily compiled, but how do you actually
pick the parties that give you the most fun and do not exceed your budget?

Write a program which finds this optimal set of parties that offer the most fun.
Keep in mind that your budget need not necessarily be reached exactly. Achieve
the highest possible fun level, and do not spend more money than is absolutely
necessary.

Input
The first line of the input specifies your party budget and the number n of
parties.

The following n lines contain two numbers each. The first number indicates the
entrance fee of each party. Parties cost between 5 and 25 francs. The second
number indicates the amount of fun of each party, given as an integer number
ranging from 0 to 10.

The budget will not exceed 500 and there will be at most 100 parties. All numbers
are separated by a single space.

There are many test cases. Input ends with 0 0.

Output
For each test case your program must output the sum of the entrance fees and the
sum of all fun values of an optimal solution. Both numbers must be separated by a
single space.

Example
Sample input:
50 10
12 3
15 8
16 9
16 6
10 2
21 9
18 4
```

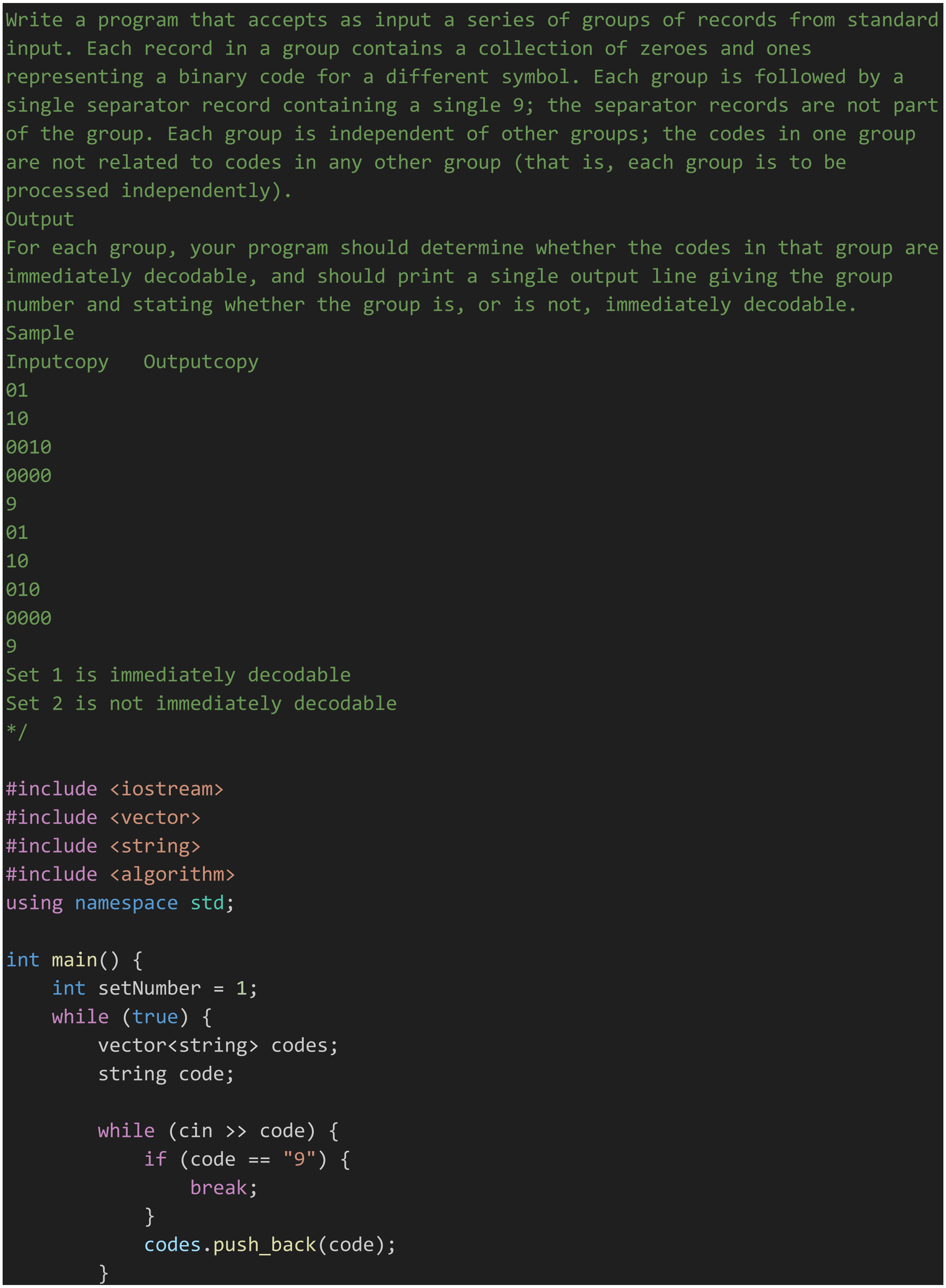
```
12 4
17 8
18 9

50 10
13 8
19 10
16 8
12 9
10 2
12 8
13 5
15 5
11 7
16 2

0 0

Sample output:
49 26
48 32

*/
```

+++

TRIE

B

```
/*
An encoding of a set of symbols is said to be immediately decodable if no code
for one symbol is the prefix of a code for another symbol. We will assume for
this problem that all codes are in binary, that no two codes within a set of
codes are the same, that each code has at least one bit and no more than ten
bits, and that each set has at least two codes and no more than eight.

Examples: Assume an alphabet that has symbols {A, B, C, D}

The following code is immediately decodable:
A:01 B:10 C:0010 D:0000

but this one is not:
A:01 B:10 C:010 D:0000 (Note that A is a prefix of C)
Input
```



```
Write a program that accepts as input a series of groups of records from standard
input. Each record in a group contains a collection of zeroes and ones
representing a binary code for a different symbol. Each group is followed by a
single separator record containing a single 9; the separator records are not part
of the group. Each group is independent of other groups; the codes in one group
are not related to codes in any other group (that is, each group is to be
processed independently).
Output
For each group, your program should determine whether the codes in that group are
immediately decodable, and should print a single output line giving the group
number and stating whether the group is, or is not, immediately decodable.
Sample
Inputcopy   Outputcopy
01
10
0010
0000
9
01
10
010
0000
9
Set 1 is immediately decodable
Set 2 is not immediately decodable
*/

#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int main() {
    int setNumber = 1;
    while (true) {
        vector<string> codes;
        string code;

        while (cin >> code) {
            if (code == "9") {
                break;
            }
            codes.push_back(code);
        }
        if (codes.empty()) {
            break;
        }

        bool decodable = true;

        for (int i = 0; i < codes.size() - 1; i++) {
            for (int j = i + 1; j < codes.size(); j++) {
                if (codes[i] == codes[j].substr(0, codes[i].length()) || codes[j]
== codes[i].substr(0, codes[j].length())) {
                    decodable = false;
                    break;
                }
            }
            if (!decodable) {
                break;
            }
        }

        cout << "Set " << setNumber << " is " << (decodable ? "immediately
decodable" : "not immediately decodable") << endl;
        setNumber++;
    }

    return 0;
}
```

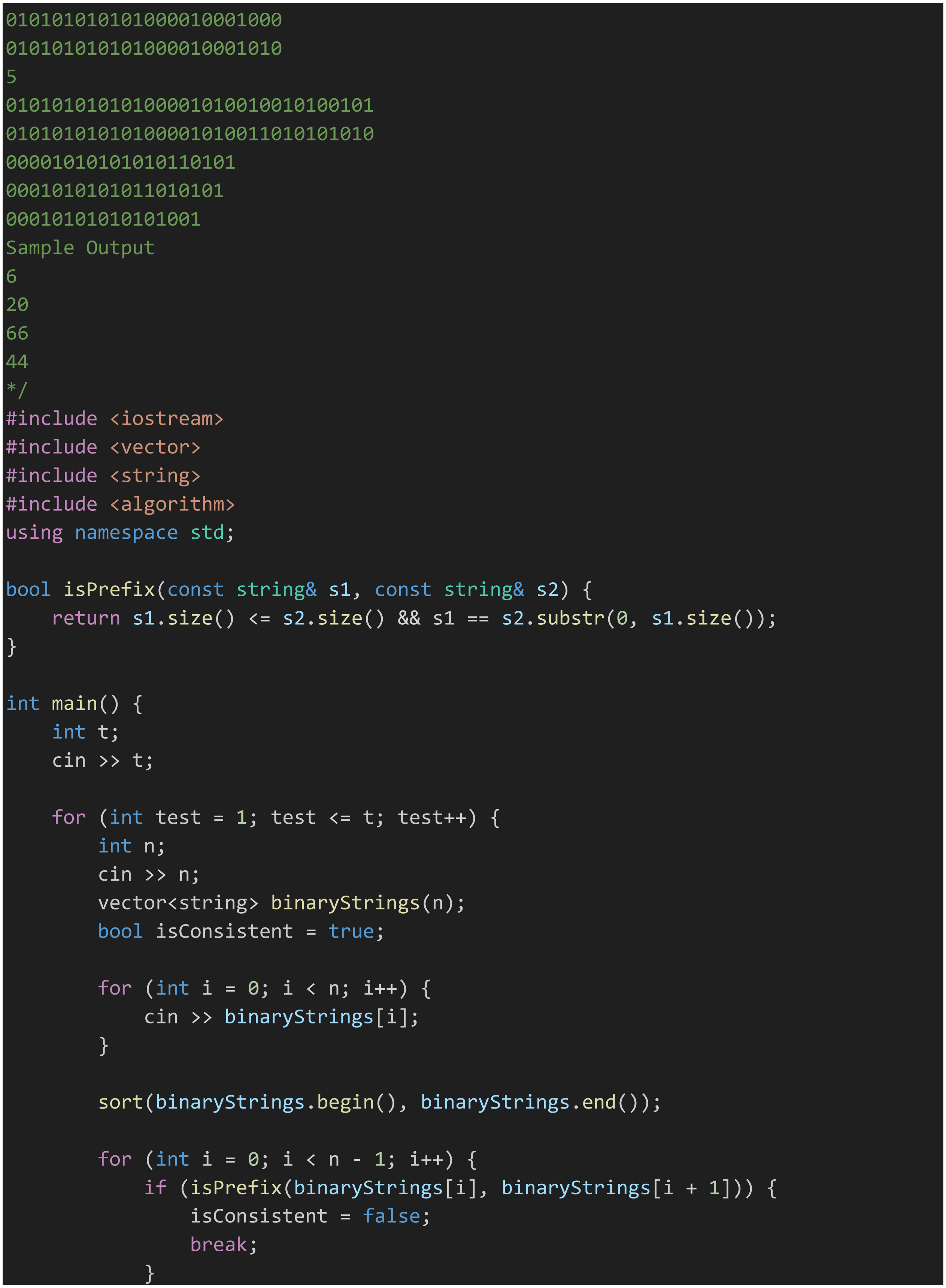
C

```
/*
Given a list of phone numbers, determine if it is consistent in the sense that
no number is the prefix of another. Let’s say the phone catalogue listed these
numbers:
• Emergency 911
• Alice 97 625 999
• Bob 91 12 54 26
In this case, it’s not possible to call Bob, because the central would direct
your call to the emergency line as soon as you had dialled the first three digits
of
Bob’s phone number. So this list would not be consistent.
Input
The first line of input gives a single integer, 1 ≤ t ≤ 40, the number of test
cases. Each test case starts
```

```
with n, the number of phone numbers, on a separate line, 1 ≤ n ≤ 10000. Then
follows n lines with
one unique phone number on each line. A phone number is a sequence of at most ten
digits.
Output
For each test case, output ‘YES’ if the list is consistent, or ‘NO’ otherwise.
Sample Input
2
3
911
97625999
91125426
5
113
12340
123440
12345
98346
Sample Output
NO
YES
*/
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

bool isConsistent(vector<string>& phoneNumbers) {
    sort(phoneNumbers.begin(), phoneNumbers.end());

    for (int i = 0; i < phoneNumbers.size() - 1; i++) {
        if (phoneNumbers[i + 1].find(phoneNumbers[i]) == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int t;
    cin >> t;

    while (t--) {
        int n;
        cin >> n;
        vector<string> phoneNumbers(n);

        for (int i = 0; i < n; i++) {
            cin >> phoneNumbers[i];
        }

        bool consistent = isConsistent(phoneNumbers);
        cout << (consistent ? "YES" : "NO") << endl;
    }

    return 0;
}
```

D

```
/*
Prefix goodness of a set string is length of longest common prefix*number of
strings in the set. For
example the prefix goodness of the set {000,001,0011} is 6.You are given a set of
binary strings. Find
the maximum prefix goodness among all possible subsets of these binary strings.
Input
First line of the input contains T (≤ 20) the number of test cases. Each of the
test cases start with n
(≤ 50000) the number of strings. Each of the next n lines contains a string
containing only ‘0’ and ‘1’.
Maximum length of each of these string is 200.
Output
For each test case output the maximum prefix goodness among all possible subsets
of n binary strings.
Sample Input
4
4
0000
0001
10101
010
2
01010010101010101010
11010010101010101010
3
010101010101000010001010
```



```
010101010101000010001000
010101010101000010001010
5
01010101010100001010010010100101
01010101010100001010011010101010
00001010101010110101
0001010101011010101
00010101010101001
Sample Output
6
20
66
44
*/
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

bool isPrefix(const string& s1, const string& s2) {
    return s1.size() <= s2.size() && s1 == s2.substr(0, s1.size());
}

int main() {
    int t;
    cin >> t;

    for (int test = 1; test <= t; test++) {
        int n;
        cin >> n;
        vector<string> binaryStrings(n);
        bool isConsistent = true;

        for (int i = 0; i < n; i++) {
            cin >> binaryStrings[i];
        }

        sort(binaryStrings.begin(), binaryStrings.end());

        for (int i = 0; i < n - 1; i++) {
            if (isPrefix(binaryStrings[i], binaryStrings[i + 1])) {
                isConsistent = false;
                break;
            }
        }

        cout << "Set " << test << " is " << (isConsistent ? "immediately
decodable" : "not immediately decodable") << endl;
    }

    return 0;
}
```