1. Research **queue**which is implemented in C library at <http://www.cplusplus.com/reference/queue/queue/>. You can use library **queue**in c++ for this question.

Using **queue**, complete function **bool isBipartite(vector<vector<int>> graph)** to determine if a graph is bipartite or not (the graph can be disconnected). In caat <https://en.wikipedia.org/wiki/Bipartite_graph>.

You can use below liberaries in this question.

#include <iostream>

#include <vector>

#include <queue>

**For example:**

| **Test** | **Result** |
| --- | --- |
| int G[6][6] = { {0, 1, 0, 0, 0, 1},  {1, 0, 1, 0, 0, 0},  {0, 1, 0, 1, 0, 0},  {0, 0, 1, 0, 1, 0},  {0, 0, 0, 1, 0, 1},  {1, 0, 0, 0, 1, 0} };  int n = 6;  vector<vector<int>> graph(n, vector<int>());  for (int i = 0; i < n; ++i) {  for (int j = 0; j < n; ++j) {  if (G[i][j]) graph[i].push\_back(j);  }  }  isBipartite(graph) ? cout << "Yes" : cout << "No"; | Yes |

|  |
| --- |
| bool isBipartite(vector<vector<int>> graph) {  int n = graph.size();  vector <int> color (n,0);  for (int i = 0; i < n; i ++) {  if (color [i] != 0) continue;  color [i] = 1;  queue <int> q;  q.push (i);  while (!q.empty()) {  int t = q.front();  q.pop();  for (auto &x:graph[t]) {  if (color [x] == 0) {  color [x] =- color [t];  q.push(x);  }  else if (color [x] == color [t]) return false;  }  }  }  return true;  } |

|  | **Test** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | | | | |

Passed all tests!

2. Research **queue**which is implemented in C library at: <http://www.cplusplus.com/reference/queue/queue/>. You can use library **queue**in c++ for this question.

Using **queue**, complete function **void bfs(vector<vector<int>> graph, int start)** to traverse all the nodes of the graph from given start node using Breadth First Search algorithm and data structure **queue**, and print the order of visited nodes.

You can use below liberaries in this question.

#include <iostream>

#include <vector>

#include <queue>

**For example:**

| **Test** | **Result** |
| --- | --- |
| int init\_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},  {0, 0, 1, 1, 0, 0, 0, 1, 0, 0},  {0, 1, 0, 0, 0, 1, 1, 0, 1, 1},  {1, 0, 0, 0, 0, 0, 0, 1, 0, 0},  {0, 1, 0, 0, 0, 0, 0, 1, 0, 0},  {1, 0, 1, 0, 1, 0, 0, 0, 1, 0},  {0, 0, 1, 1, 0, 1, 0, 0, 0, 0},  {1, 0, 0, 0, 0, 1, 1, 0, 1, 0},  {0, 0, 0, 0, 0, 1, 0, 1, 0, 1},  {1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };  int n = 10;  vector<vector<int>> graph(n, vector<int>());  for (int i = 0; i < n; ++i) {  for (int j = 0; j < n; ++j) {  if (init\_graph[i][j]) graph[i].push\_back(j);  }  }  bfs(graph, 0); | 0 1 2 4 6 8 3 7 5 9 |

|  |
| --- |
| void ctrl (int u) {  cout << u << " ";  }  void bfs (vector<vector<int>> graph, int start) {  queue <int> q;  q.push (start);  bool visit [100];  visit [start] = true;  while (!q.empty()) {  int u = q.front();  ctrl (u);  q.pop();  for (auto i = graph[u].begin(); i != graph[u].end(); i++) {  if (!visit[\*i]) {  q.push (\*i);  visit [\*i] = true;  }  }  }  } |

|  | **Test** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | | | | |

Passed all tests!

3. You are keeping score for a basketball game with some new rules. The game consists of several rounds, where the scores of past rounds may affect future rounds' scores.

At the beginning of the game, you start with an empty record. You are given a list of strings **ops**, where **ops[i]** is the operation you must apply to the record, with the following rules:

* A non-negative integer **x (from 0 to 9)** - record a new score of **x**
* **'+'** - Record a new score that is the sum of the previous two scores. It is guaranteed there will always be two previous scores.
* **'D'**- Record a new score that is double the previous score. It is guaranteed there will always be a previous score.
* **'C'** - Invalidate the previous score, removing it from the record. It is guaranteed there will always be a previous score.

Finally, return the sum of all scores in the record.

**For example**:

ops = "52CD+"

* '5' - add to the record. Record now is [5]
* '2' - add to the record. Record now is [5,2]
* 'C' - invalid the previous score (2). Record now is [5]
* 'D' - Record new score that is double of previous score (5\*2). Record now is [5,10]
* '+' - Record a new score that is the sum of the previous two scores. Record now is [5,10,15]

Return the sum: 5+10+15 = 30

**For example:**

| **Test** | **Result** |
| --- | --- |
| cout << baseballScore("52CD+"); | 30 |
| cout << baseballScore("524CD9++"); | 55 |

|  |
| --- |
| int baseballScore (string ops) {  int a [1000];  int count = 0;  int len = ops.length();  for (int i = 0; i < len; i++) {  if (ops [i] >= '0' && ops [i] <= '9') {  count++;  a [count-1] = ops [i] - 48;  }  if (ops [i] == 'C') {  count--;  }  if (ops [i] == 'D') {  count++;  a [count-1] = a [count-2] \* 2;  }  if (ops [i] == '+') {  count++;  a [count-1] = a [count-2] + a [count-3];  }  }  int sum = 0;  for (int i = 0; i < count; i++){  sum += a [i];  }  return sum;  } |

|  | **Test** | **Expected** | **Got** |
| --- | --- | --- | --- |
|  | cout << baseballScore("52CD+"); | 30 | 30 |  |
|  | cout << baseballScore("524CD9++"); | 55 | 55 |  |
|  | cout << baseballScore("5C4C2C11+D3"); | 11 | 11 |  |

Passed all tests!

4. Given a string **S** of characters, a *duplicate removal* consists of choosing two adjacent and equal letters, and removing them.

We repeatedly make duplicate removals on **S** until we no longer can.

Return the final string after all such duplicate removals have been made.

**For example:**

| **Test** | **Result** |
| --- | --- |
| cout << removeDuplicates("abbaca"); | ca |
| cout << removeDuplicates("aab"); | b |

|  |
| --- |
| string removeDuplicates (string S) {  int n = S.length();  int count = 0;  int i = 0;  do {  count=0;  while(i<n) {  if (S [i] == S [i+1]){  count++;  S.erase (i,2);  if (i>0) {  i--;  }  break;  }  else {  i++;  }  }  }  while(count>0);  return S;  } |

|  | **Test** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | cout << removeDuplicates("abbaca"); | ca | ca |  |
|  | cout << removeDuplicates("aab"); | b | b |  |

Passed all tests!

5. Given a string **s**containing just the characters **'(', ')', '[', ']', '{', and '}'**. Check if the input string is valid based on following rules:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

For example:

* String **"[]()"** is a valid string, also **"[()]".**
* String "**[])**" is **not** a valid string.

Your task is to implement the function

bool isValidParentheses (string s){  
    /\*TODO\*/  
}

**For example:**

| **Test** | **Result** |
| --- | --- |
| cout << isValidParentheses("[]"); | 1 |
| cout << isValidParentheses("[]()"); | 1 |
| cout << isValidParentheses("[)"); | 0 |

|  |
| --- |
| bool isValidParentheses (string s) {  vector <char> stk;  for (char c : s) {  switch (c) {  case '(':  case '{':  case '[':  stk.push\_back(c);  break;  case ')':  if (stk.empty() || stk.back() != '(')  return false;  stk.pop\_back();  break;  case '}':  if (stk.empty() || stk.back() != '{')  return false;  stk.pop\_back();  break;  case ']':  if (stk.empty() || stk.back() != '[')  return false;  stk.pop\_back();  break;  }  }  return stk.empty();  } |

|  | **Test** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | cout << isValidParentheses("[]()"); | 1 | 1 |  |
|  | cout << isValidParentheses("[)"); | 0 | 0 |  |

Passed all tests!

6. Implement static methods **sortSegment**and **ShellSort**in class **Sorting**to sort an array in ascending order.

#ifndef SORTING\_H

#define SORTING\_H

#include <sstream>

#include <iostream>

#include <type\_traits>

using namespace std;

template <class T>

class Sorting {

private:

static void printArray(T\* start, T\* end)

{

int size = end - start;

for (int i = 0; i < size; i++)

cout << start[i] << " ";

cout << endl;

}

static void sortSegment(T\* start, T\* end, int segment\_idx, int cur\_segment\_total) ;

public:

static void ShellSort(T\* start, T\* end, int\* num\_segment\_list, int num\_phases) ;

};

#endif /\* SORTING\_H \*/

**For example:**

| **Test** | **Result** |
| --- | --- |
| int num\_segment\_list[] = {1, 3, 5};  int num\_phases = 3;  int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 };  Sorting<int>::ShellSort(&array[0], &array[10], &num\_segment\_list[0], num\_phases); | 5 segments: 5 4 3 2 1 10 9 8 7 6  3 segments: 2 1 3 5 4 7 6 8 10 9  1 segments: 1 2 3 4 5 6 7 8 9 10 |

|  |
| --- |
| static void sortSegment (T\* start, T\* end, int segment\_idx, int cur\_segment\_total) {  int current = segment\_idx + cur\_segment\_total;  int size = end - start;  while (current < size) {  int temp = start[current];  int walker = current - cur\_segment\_total;  while (walker >= 0 && temp < start [walker]) {  start [walker + cur\_segment\_total] = start [walker];  walker -= cur\_segment\_total;  }  start [walker + cur\_segment\_total] = temp;  current += cur\_segment\_total;  }  return;  }  static void ShellSort (T\* start, T\* end, int\* num\_segment\_list, int num\_phases) {  // Note: You must print out the array after sorting segments to check whether your algorithm is true.  int i = 1;  int segment = 0;  while (num\_phases > 0) {  while (segment < num\_segment\_list [num\_phases - i]) {  sortSegment (start, end, segment, num\_segment\_list [num\_phases - i]);  segment++;  }  int size = end - start;  for (int j = 0; j < size; j++) {  if (j == 0) cout << num\_segment\_list [num\_phases - i] << " " << "segments: " << start [j] << " ";  else  cout << start [j] << " ";  }  cout << endl;  num\_phases--;  segment = 0;  }  } |

|  | **Test** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | | | | |

Passed all tests!

7. Two strings are called permutation of each other when they have exactly the same number of character, and the number of appearance of each character in each string must be the same.

For example:

String a = "abba" and String b = "baba" are said to be permutation of each other. While String a = "abbc" and String b = "baba" are not.

Your task in this exercise is to implement the **isPermutation** function. Note that, you can write one or more functions in order to achieve this exercise.

#ifndef SORTINGAPPLICATION\_H  
#define SORTINGAPPLICATION\_H  
#include <iostream>  
#include <string>

#include <algorithm>  
**using** **namespace** std;  
**bool** isPermutation (string a, string b) {}  
#endif /\* SORTINGAPPLICATION\_H \*/

**For example:**

| **Test** | **Result** |
| --- | --- |
| string a = "abba";  string b="baba";  cout << isPermutation(a, b); | 1 |
| string a = "abbac";  string b="baba";  cout << isPermutation(a, b); | 0 |

|  |
| --- |
| bool isPermutation (string a, string b) {  if (a.length() != b.length()) {  return false;  }  for (unsigned int i = 0; i < a.length(); i++) {  for (unsigned int j = 0; j < b.length(); j++) {  if(a [i] == b [j]) {  a.erase (a.begin() + i);  b.erase (b.begin() + j);  i--;  break;  }  }  }  if (a != b) {  return false;  }  else return true;  } |

|  | **Test** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | string a = "abba";  string b="baba";  cout << isPermutation(a, b); | 1 | 1 |  |
|  | string a = "abbac";  string b="baba";  cout << isPermutation(a, b); | 0 | 0 |  |

Passed all tests!

8. You are given a list of integers nums with n elements (3 ≤ n ≤ 100000), the value of each element is between -1000 to 1000.

**Request:** Implement function:

int maximumProduct(vector<int>& nums);

Where nums is the given list of integers. This function returns the maximum product of 3 elements which can be found from the given list nums.

**Example:**

The list of integers is {5, 4, 1, 3, -2, -2}. Therefore, the maximum product is 60 by choosing 3 elements of the given list are 5, 4, 3.

**Note:**

In this exercise, the libraries iostream, string, cstring, climits, utility, vector, list, stack, queue, map, unordered\_map, set, unordered\_set, functional, algorithm has been included and namespace std are used. You can write helper functions and classes. Importing other libraries is allowed, but not encouraged, and may result in unexpected errors.

**For example:**

| **Test** | **Result** |
| --- | --- |
| vector<int>nums{5, 4, 1, 3, -2, -2};  cout << maximumProduct(nums); | 60 |
| vector<int>nums{4,4,1,2,-5,3,-2,-5,0,-2};  cout << maximumProduct(nums); | 100 |

|  |
| --- |
| int maximumProduct (vector<int>& nums) {  priority\_queue<int> pq {nums.begin(), nums.end()};  priority\_queue<int, vector<int>, greater<int>> pqm (nums.begin(), nums.end());  int s1, s2, s3, m1, m2;  int ans = 0, ans1 = 0;  if (pq.size() > 2) {  s1 = pq.top();  pq.pop();  s2 = pq.top();  pq.pop();  s3 = pq.top();  pq.pop();  m1 = pqm.top();  pqm.pop();  m2 = pqm.top();  pqm.pop();  }  ans = (s1\*s2\*s3);  ans1 = (s1\*m1\*m2);  return max (ans, ans1);  } |

|  | **Test** | **Expected** | **Got** |  |
| --- | --- | --- | --- | --- |
|  | vector<int>nums{5, 4, 1, 3, -2, -2};  cout << maximumProduct(nums); | 60 | 60 |  |

Passed all tests!