Baseball Scorekeeper

```
int score(vector<string> &blocks) {
   int sum = 0;
   stack<int> scores;
   for (auto b: blocks) {
      if (b == "+") {
         int x = scores.top(); // last score
         scores.pop();
         int y = scores.top(); // second to last score
         scores.push(x);
         scores.push(x+y);
         sum += (x+y);
      } else if (b == "X") {
         int cur = scores.top()*2;
         scores.push(cur);
         sum += cur;
      } else if (b == "Z") {
         int r = scores.top();
         scores.pop();
         sum -= r;
      } else {
         int s = stoi(b);
         scores.push(s);
         sum += s;
      }
   }
   return sum;
}
```

Break a Palindrome

```
string solution(string p) {
   int n = p.length();
   if (n <= 1) {
      return "";
   }
   for (int i = 0; i < n/2; i++) {
      if (p[i] != 'a') {
         p[i] = 'a';
         return p;
      }
   }
   p[n-1] = 'b'; // if p = 'aaa'
   return p;
}
```

Copy List from Random Pointer

```
string solution(string p) {
    int n = p.length();
    if (n <= 1) {
        return "";
    }

    for (int i = 0; i < n/2; i++) {
        if (p[i] != 'a') {
            p[i] = 'a';
            return p;
        }
    }

    p[n-1] = 'b'; // if p = 'aaa'
    return p;
}
```

Count Cluster / Number of Islands

```
void dfs(vector<vector<char>>& grid, int i, int j) {
    if (i < 0 || i >= grid.size() || j < 0 || j >= grid[0].size() || grid[i][j] == '0' || grid[i][j] == '2') {
        return;
    }
    grid[i][j] == '2';
    dfs(grid, i-1, j);
    dfs(grid, i-1,j);
    dfs(grid, i, j-1);
    dfs(grid, i, j+1);
}
int numIslands(vector<vector<char>>& grid) {
    int c = 0;
    for (int i = 0; i < grid.size(); i++) {
        for (int j = 0; j < grid[0].size(); j++) {
            if (grid[i][j] == '1') {
                dfs(grid, i, j);
                c++;
            }
        }
    }
    return c;
}
```

Count Teams

```
int fac(int x) {
  if (x == 0) {
    return 1;
  }
  int res = 1;
  for (int i = 1; i <= x; i++) {
    res *= i;
  }
  return res;
}

int C(int m, int n) { // 组合 function
  int nu = fac(n); // fenzi
  int de = fac(m) * fac(n-m); // fenmu
  return nu/de;
}
int countTeams(int num, vector<int> skills, int minAsso, int minLevel, int maxLevel) {
  int n = 0; // count of associates meet requirement
  int ans = 0;

  for (auto skill: skills) { // store assos meet requirement
    if (skill <= maxLevel && skill >= minLevel) {
      n++;
    }
  }

  if (n < minAsso) {
    return 0;
  } else {
    for (int m = minAsso; m <= n; m++) {
      ans += C(m,n);
    }
  }
  return ans;
}
```

Critical Connections

```
void dfs(int u,int& t,vector<vector<int>>& gp,vector<int>& par,vector<int>& disc,vector<int>& low,vector<pair<int,int>>& criticalConnections){
        disc[u]=low[u]=t++;
        for(int v:gp[u]){
                if(v==par[u])continue;
                if(disc[v]==-1){
```

```
            par[v]=u;
            dfs(v,t,gp,par,disc,low,criticalConnections);
            low[u]=min(low[u],low[v]);
            if(disc[u]<low[v])criticalConnections.push_back({u,v});
        }else{
                low[u]=min(low[u],disc[v]);
        }
    }
}
vector<pair<int,int>> findConnections(int numOfServers,int numOfConnections,vector<pair<int,int>>
connections){
    vector<vector<int>> gp(numOfServers+1);
    for(int i=0;i<numOfConnections;i++){
        int u=connections[i].first,v=connections[i].second;
        gp[u].push_back(v);
        gp[v].push_back(u);
    }
    vector<int> par(numOfServers+1,-1);
    vector<int> disc(numOfServers+1,-1);
    vector<int> low(numOfServers+1,-1);
    int t=1;
    vector<pair<int,int>> criticalConnections;
    for(int i=1;i<=numOfServers;i++){
        if(disc[i]==-1) dfs(i,t,gp,par,disc,low,criticalConnections);
    }
    //if(criticalConnections.size()==0)criticalConnections.push_back({});
    return criticalConnections;
}
```

Critical Routers

```
void dfs(unordered_map<int, unordered_set<int>> gp, int source, unordered_set<int> &visited) {
  if (visited.find(source) != visited.end()) {
    return;
  }

  visited.insert(source);
  for(auto it = gp[source].begin(); it != gp[source].end(); it++) {
    dfs(gp, *it, visited);
  }
}

int findnodes(unordered_map<int, unordered_set<int>> gp, unordered_set<int> &visited, int
numNodes, int Rnodes) {
  int nodes = 0;
  for (int i = 0; i < numNodes; i++) {
        if (i == Rnodes) {
            continue;
        }
        if (visited.find(i) == visited.end()) {
            nodes++;
            dfs(gp, i, visited);
        }
    }

    return nodes;
}

vector<int> getNodes (vector<vector<int>> edges, int numNodes, int numEdges) {
    unordered_map<int, unordered_set<int>> gp;
    for (auto edge: edges) {
        int u = edge[0];
        int v = edge[1];
        gp[u].insert(v);
        gp[v].insert(u);
    }

    vector<int> res;
    unordered_set<int> ininodes;
    int iniNodes = findnodes(gp, ininodes, numNodes, -1);
    for (int nodeR = 0; nodeR < numNodes; nodeR++) {
        unordered_set<int> nodeEdges = gp[nodeR];
        int source = 0;

        for (auto it = nodeEdges.begin(); it != nodeEdges.end(); it++) {
            gp[*it].erase(nodeR);
            source = *it;
        }

        unordered_set<int> visited;
        int nodes = findnodes(gp, visited, numNodes, nodeR);
        if (nodes != iniNodes) {
            res.push_back(nodeR);
        }
        for (auto it = nodeEdges.begin(); it != nodeEdges.end(); it++) {
            gp[*it].insert(nodeR);
        }
    }
    return res;
}
```

Cutoff Ranks

```
int cutOffrank(int k, int num, vector<int> scores) {
    sort(scores.begin(), scores.end(), greater<int>());
    int r = 1;
    int res = 0;

    for (int i = 0; i < scores.size(); i++) {
        if (i == 0) {
            r = 1;
        } else if (scores[i] != scores[i-1]){
            r = i+1;
        }

        if (r <= k && scores[i] > 0) {
            res++;
        } else {
            break;
        }
    }
    return res;
}
```

Disk Space Analysis

```
int diskAnalysis(int n, vector<int> space, int x) {
    vector<int> mins;
    int ans = INT_MIN;
    deque<int> dq;

    for (int i = 0; i < n; i++) {
        int cur = space[i];
        while (!dq.empty() && dq.front() <= (i - x)) {
            dq.pop_front();
        }
        while(!dq.empty() && cur < space[dq.back()]) {
            dq.pop_back();
        }
        dq.push_back(i);
        if (i >= x-1) {
            ans = max(ans, space[dq.front()]);
        }
    }
    return ans;
}
```

Fetch Items to Display

```
vector<string> fetchItem(int num, map<string, pair<int, int>> &products, int sortKey, bool sortOrder, int
productsPerRow, int rowNum) {
  vector<string> res;
  if (sortOrder) { // 从小到大
    priority_queue<pair<int, string>, vector<pair<int, string>>, greater<pair<int, string>>> pq;
    for (auto p: products) {
      if (sortKey == 0) {
        pq.push(make_pair(-1, p.first));
      } else if (sortKey == 1) {
        pq.push(make_pair(p.second.first, p.first));
      } else {
        pq.push(make_pair(p.second.second, p.first));
      }
    }
    while(!pq.empty()) {
      auto temp = pq.top();
      res.push_back(temp.second);
      pq.pop();
    }
  } else {
    priority_queue<pair<int, string>> pq; // 从大到小
    for (auto p: products) {
      if (sortKey == 0) {
        pq.push(make_pair(-1, p.first));
      } else if (sortKey == 1) {
        pq.push(make_pair(p.second.first, p.first));
      } else {
        pq.push(make_pair(p.second.second, p.first));
      }
    }
    while(!pq.empty()) {
      auto temp = pq.top();
      res.push_back(temp.second);
      pq.pop();
    }
  }
  int start = productsPerRow * rowNum;
  int n = res.size();
  int end = min((start + productsPerRow), n);
  vector<string> ans(res.begin()+start, res.begin()+end);
  return ans;
}
```

Fill the Truck / Get Maximum Units

```
long getMaxUnit (int num, vector<int> boxes, int unitSize, vector<int> unitsPerBox, int truckSize) {
   long ans=0;
   priority_queue<int> pq;

   for (int i = 0; i < num; i++) {
      for (int j = 0; j < unitsPerBox[i]; j++) {
         pq.push(boxes[i]);
      }
   }

   while(truckSize > 0 && !pq.empty()) {
      ans += pq.top();
      cout << pq.top() << endl;
      pq.pop();
      truckSize--;
   }

   return ans;
};
```

Least Number of Integers after Removing k

```
int findUniqueInts(vector<int> &arr, int k) {
   unordered_map<char, int> mp;
   vector<int> v;
   int r = 0;
   for (auto a: arr) {
      mp[a]++;
   }
   for (auto m: mp) {
      v.push_back(m.second);
   }
   sort(v.begin(), v.end());
   for (auto i: v) {
      if (k >= i) {
         r++;
      }
      k = k-i;
      if (k <= 0) {
         break;
      }
   }
   return v.size() - r;
}
```

LRU Cache Misses

```
class LRUCache {
public:
    list<int> dp;
    int csize;
    unordered_map<int, int> mp;
    int miss;

    LRUCache(int capacity) {
        csize = capacity;
        miss = 0;
    }

    int put(int key, int value) {
        if (mp.size() < csize || mp.find(key) != mp.end()) {
            if (mp.find(key) != mp.end()) {
                mp[key] = value;
                dp.remove(key);
                dp.push_back(key);
            } else {
                mp[key] = value;
                dp.push_back(key);
                miss++;
            }
        } else {
            int last = dp.front();
            dp.pop_front();
            mp.erase(last);

            dp.push_back(key);
            mp[key] = value;
            miss++;
        }
        return key;
    }
    int get(int key) { // not used for count misses
        if (mp.find(key) != mp.end()) {
            dp.remove(key);
            dp.push_back(key);
            return mp[key];
        }
        return -1;
    }
};
```

Maximal Squares of 1

```
int maximalSquares(vector<vector<char>>& matrix) {
   int r = matrix.size();
   int c = matrix[0].size();

   vector<vecto<int>> dp(r, vector<int>(c,0));
   int res = 0;
   for (int i = 0; i < r; i++) {
      for (int j = 0; j < c; j++) {
         dp[i][j] = matrix[i][j] - '0';
         res = max(res, dp[i][j]);
      }
   }

   for (int i = 1; i < r; i++) {
      for (int j = 1; j < c; j++) {
         if (matrix[i][j] == '1') {
            res = max(res, 1);
            dp[i][j] = min(dp[i-1][j], dp[i][j-1]);
            dp[i][j] = min(dp[i][j], dp[i-1][j-1]);
            dp[i][j] += 1;
            res = max(res, dp[i][j]);
         }
      }
   }

   return res*res;
}
```

Max of Min Altitude

```
int maxScore(vector<int>& grid) {
   if (grid.empty()) {
      return 0;
   }
   for (int i = 1; i < grid.size(); i++) {
      grid[i][0] = min(grid[i][0], grid[i-1][0]); // find minimum score of first row
   }
   for (int i = 1; i < grid.size(); i++) {
      grid[0][i] = min(grid[0][i], grid[0][i-1]); // find minimum score of first col
   }

   for (int i = 1; i < grid.size(); i++) {
      for (int j = 1; j < grid[0].size(); j++) {
         grid[i][j] = max(min(grid[i][j], grid[i-1][j]), min(grid[i][j], grid[i][j-1]));
```

```
        }
    }

    return grid[grid.size()-1][grid[0].size()-1];

}
```

Merge Two Sorted List

```
struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next): val(x), next(next) {}
};

ListNode* mergeTwoList (ListNode* l1, ListNode* l2) {
    if (!l1 && !l2) {
        return nullptr;
    }
    if (!l1) {
        return l2;
    }
    if (!l2) {
        return l1;
    }
    ListNode* res = NULL;

    if (l1->val <= l2->val) {
        res = l1;
        res->next = mergeTwoList(l1->next, l2);
    } else {
        res = l2;
        res->next = mergeTwoList(l1, l2->next);
    }
    return res;
}
```

Min Difficulty of Job Scheduling

```
int minDifficulty(vector<int> &jobDifficulty, int d) {
    if (jobDifficulty.size() < d) {
        return -1;
    }
    int n = jobDifficulty.size();
    vector<vector<int>> dp(n+1, vector<int>(n+1, INT_MAX/2));
    dp[0][0] = 0;
    for(int i = 1; i <= n; i++) {
        for(int k = 1; k <= d; k++) {
            int temp = 0;
            for (int j = i-1; j >= k-1; j--) {
                temp = max(temp, jobDifficulty[j]);
                dp[i][k] = min(dp[i][k], dp[j][k-1] + temp);
            }
        }
    }
    return dp[n][d];
}
```

Min Distance of Robot / Minimum Squared Distance

```
int dist(pair<int, int> p1, pair<int, int> p2) {
    return (p1.first - p2.first) * (p1.first - p2.first) + (p1.second - p2.second) * (p1.second - p2.second);
}

int divide(int left, int right, vector<pair<int, int>> &pos) {
    int curMin = INT_MAX;
    if (left == right) { // if only one pos
        return curMin;
    }
    if (left+1 == right) { // if only two pos
        return dist(pos[left], pos[right]);
    }

    int mid = (left + right) / 2;
    int leftMin = divide(left, mid, pos);
    int rightMin = divide(mid, right, pos);
    curMin = min(leftMin, rightMin);

    vector<int> posInd;
    for (int i = left; i <= right; i++) {
        if (abs(pos[mid].first - pos[i].first) <= curMin) {
            posInd.push_back(i);
        }
    }

    for (int i = 0; i < posInd.size(); i++) {
        for (int j = i+1; j < posInd.size(); j++) {
            if (abs(pos[posInd[i]].second - pos[posInd[j]].second) > curMin) {
                continue;
            }
            int tempdis = dist(pos[posInd[i]], pos[posInd[j]]);
            curMin = min(tempdis, curMin);
        }
    }

    return curMin;
}

int minDistRobot(int numRobots, vector<int> positionX, vector<int> positionY) {
    vector<pair<int, int>> pos; // <xpos, ypos>
    for (int i = 0; i < numRobots; i++) {
        auto temp = make_pair(positionX[i], positionY[i]);
        if (find(pos.begin(), pos.end(), temp) == pos.end()) {
            pos.push_back(make_pair(positionX[i], positionY[i]));
        }
    }
    sort(pos.begin(), pos.end());

    return divide(0, pos.size()-1, pos);
}
```

Most Frequent Word

```
string mostCommonWord(string p, vector<string> &b) {
    replace(p.begin(), p.end(), ',', ' ');
    p.erase(remove_if(p.begin(), p.end(), [](char A){return ispunct(A);}), p.end());
    transform(p.begin(), p.end(), p.begin(), ::tolower);
    vector<string> words = {istream_iterator<string>{istringstream()=istringstream(p)},
istream_iterator<string>{}};

    unordered_map<string, int> mp;

    for (int i = 0; i < words.size(); i++) {
        mp[words[i]]++;
    }
    for (int i = 0; i < b.size(); i++) {
        mp[b[i]] = 0;
    }

    string res = "";
    int m = 0;
    for (auto word: mp) {
        if (word.second > m) {
            m = word.second;
            res = word.first;
        }
    }
    return res;
}
```

Nearest City

```
vector<string> NC(int num, vector<string> points, vector<int> xCoor, vector<int> yCoor, int
numOfQueriedPoints, vector<string> queries) {
    unordered_map<int, vector<string>> xTocity;
    unordered_map<int, vector<string>> yTocity;
    unordered_map<string, int> pointsInd;
    vector<string> res{numOfQueriedPoints, ""};

    for (int i = 0; i < num; i++) {
        xTocity[xCoor[i]].push_back(points[i]);
        yTocity[yCoor[i]].push_back(points[i]);
        pointsInd[points[i]] = i;
    }

    for (int i = 0; i < num; i++) {
        string city = queries[i];
        int ind = pointsInd[city]; // city's index
        int x = xCoor[ind]; // city's x
        int y = yCoor[ind]; // city's y

        vector<string> xN = xTocity[x]; // neighbors of city including city
        vector<string> yN = yTocity[y]; // neighbors of city including city

        if (xN.size() == 1 && yN.size() == 1) {
            res[i] = "None";
            continue;
        }

        int mindist = INT_MAX;
        string mincity = "";

        for (string xn: xN) {
            if (xn == city) {
                continue;
            }
            int ind_n = pointsInd[xn];
            int nei_xcoor = xCoor[ind_n];
            int dist = abs(nei_xcoor - x);

            if (dist < mindist) {
                mindist = dist;
                mincity = xn;
            }

        }

        for (string yn: yN) {
            if (yn == city) {
                continue;
            }
            int ind_n = pointsInd[yn];
            int nei_ycoor = yCoor[ind_n];
            int dist = abs(nei_ycoor - y);

            if (dist < mindist) {
                mindist = dist;
                mincity = yn;
            }
        }
        res[i] = mincity;
    }
    return res;
}
```

Power Grid

```
int id[27];

 void init(vector<pair<pair<char, char>, int>> &connections) {
     for (auto c: connections) {
         int n1 = (c.first.first - 'A');
         int n2 = c.first.second - 'A';
         id[n1] = n1;
         id[n2] = n2;
     }
 }

 bool customsort(pair<pair<char, char>, int> &a, pair<pair<char, char>, int> &b) {
     return a.second < b.second;
 }

 bool anssort(pair<pair<char, char>, int> &a, pair<pair<char, char>, int> &b) {
     return a.first.first < b.first.first;
 }

 int root(int node) {
     while(id[node] != node) {
         id[node] = id[id[node]];
```

```
        node = id[node];
    }
    return node;
 }

 void un(int r1, int r2) {
    int p = root(r1);
    int q = root(r2);
    id[p] = id[q];
 }

 vector<pair<pair<char, char>, int>> solve(vector<pair<pair<char, char>, int>> &connections) {
    init(connections);
    vector<pair<pair<char, char>, int>> ans;
    sort(connections.begin(), connections.end(), customsort); // sort connections by weights in increasing fashion
    for (auto c: connections) {
        int n1 = c.first.first - 'A';
        int n2 = c.first.second - 'A';
        int cur_cost = c.second;
        if (root(n1) != root(n2)) {
           ans.push_back({{n1+'A', n2+'A'}, cur_cost});
           un(n1, n2);
        }
    }
    sort(ans.begin(), ans.end(), anssort);
    return ans;
 }
```

Products Suggestions

```
vector<string> help(vector<string> &products, string s) {
    auto it1 = lower_bound(products.begin(), products.end(), s); // time: O(logn)
    s[s.size()-1]++;
    auto it2 = lower_bound(products.begin(), products.end(), s);
    if (it2 - it1 > 3) {
        it2 = it1+3;
    }
    return vector<string> (it1, it2);
}

vector<vector<string>> suggestedProd(vector<string>& products, string searchword) {
    string s = "";
    vector<vector<string>> res;
    sort(products.begin(), products.end());
    for(char ch: searchword) {
        s += ch;
        res.push_back(help(products, s));
    }

    return res;
}
```

Search 2D Matrix 2

```
bool searchMatrix(vector<vector<int>> &matrix, int target) {
    int m = matrix.size();
    int n = matrix[0].size();
    if (m == 0 || n == 0) {
        return false;
    }
    for (int i = 0, j = n-1; i < m && j >= 0;) {
        if (target == matrix[i][j]) {
            return true;
        } else if (target > matrix[i][j]) {
            i++;
        } else {
            j--;
        }
    }

    return false;
}
```

Secret Fruit List

```
int winPrize(vector<vector<string>> &codeList, vector<string> &shoppingCart) {
    if (codeList.size() == 0) {
        return 1;
    }
    if (shoppingCart.size() == 0) {
        return 0;
    }

    int i = 0, j = 0;
    while(i < codeList.size() && j+codeList[i].size() <= shoppingCart.size()) {
        bool match = true;
        for (int k = 0; k < codeList[i].size(); k++) {
            if ((codeList[i][k] != "anything") && (shoppingCart[j+k] != codeList[i][k])) {
                match = false;
                break;
            }
        }
        if (match) {
            j += codeList[i].size();
            i++;
        } else {
            j++;
        }
    }
    if (i == codeList.size()) {
        return 1;
    } else {
        return 0;
    }
}
```

Smallest Negative Balance / debt Record

```
class debtRecord {
public:
    string borrower;
    string lender;
    int amount;
    debtRecord() {}
    debtRecord(string borrower, string lender, int amount) {
        borrower = borrower;
        lender = lender;
        amount = amount;
    }
};

vector<string> negativeBalance(int numRows, int numCols, vector<debtRecord> records) {
    if (numRows == 0 || records.size() == 0) {
        return {};
    }

    unordered_map<string, int> mp;
    priority_queue<pair<int, string>, vector<pair<int, string>>, greater<pair<int, string>>> pq;
    vector<string> res;

    for (auto r: records) {
        string b = r.borrower;
        string l = r.lender;
        int a = r.amount;
        cout << b << endl;

        if (mp.find(b) == mp.end()) {
            mp.emplace(b, 0-a);
        } else {
            mp[b] -= a;
        }

        if (mp.find(l) == mp.end()) {
            mp.emplace(l, a);
        } else {
            mp[l] += a;
        }
    }

    for (auto& it: mp ) {
        if (it.second < 0) {
            pq.push(make_pair(it.second, it.first));
        }
    }

    if(!pq.empty()) {
        int minNeg = pq.top().first;
        while(!pq.empty() && pq.top().first == minNeg) {
            res.push_back(pq.top().second);
            pq.pop();
        }
    }
    return res;
}
```

Favorite Genres

```
unordered_map<string, vector<string>> solution(unordered_map<string, vector<string>> users,
unordered_map<string, vector<string>> genres) {
    unordered_map<string, string> songTogenre;
    unordered_map<string, unordered_map<string, int>> userTogenre;
    unordered_map<string, vector<string>> res;
    unordered_map<string, int> maxi;

    for (auto genre: genres) {
        for (auto song: genre.second) {
            songTogenre[song] = genre.first; // song -> genre
        }
    }

    for (auto user: users) {
        for (auto item: user.second) {
            userTogenre[user.first][songTogenre[item]]++; // user -> genre -> count
            auto tempMax = userTogenre.at(user.first).at(songTogenre[item]);
            maxi[user.first] = max(tempMax, maxi[user.first]); // user -> max count
        }
    }

    for (auto user: userTogenre) {
        for(auto genre: user.second) {
            if (genre.second == maxi.at(user.first)) {
                res[user.first] .push_back(genre.first);
            }
        }
    }

    return res;
}
```

Spiral Matrix 2

```
vector<vector<int>> generateM(int n) {
     vector<vector<int>> res(n, vector<int>(n,0));
     vector<vector<int>> direction{{0,1}, {1,0}, {0, -1}, {-1,0}};
     int count = 1;
     int row = 0, col = 0, d = 0;
     while(count <= n*n) {
         res[row][col] = count++;
         int r = floor((row+direction[d][0]) % n);
         int c = floor((col+direction[d][1]) % n);

         if (res[r][c] != 0) {
             d = (d+1) % 4;
         }
         row += direction[d][0];
         col += direction[d][1];
     }
     return res;
}
```

Split String Into Unique Primes

```
map <int, int> sieve(int n) {
    map<int, int> primes;
    vector<int> p(1001,1);
    p[0] = 0; p[1] = 0;
    for(int i = 2; i <= 1000; i++) {
        if (p[i] == 0) {
           continue;
        }
        for (int j = i*i; j <= 1000; j += i) {
            p[j] = 0;
        }
    }
    for (int i = 2; i < 1000; i++) {
        if (p[i]) {
            primes[i] = 1;
        }
    }
    return primes;
}

int solve(string &s, map<int, int> &prime) {
    int n = s.length();
    int dp[n+1] = {0}; // dp[i] - number of ways to split till ith digit into primes
    dp[0] = 1;

    const int mod = 1e9 + 7;

    for (int i = 1; i <= n; i++) {
        if (s[i-1] != '0' && prime[stoi(s.substr(i-1,1))]) {
            dp[i] = dp[i-1];
        }
        if (i-2 >= 0 && s[i-2] != '0' && prime[stoi(s.substr(i-2,2))]) {
            dp[i] = (dp[i] + dp[i-2]) % mod;
        }
        if (i-3 >= 0 && s[i-3] != '0' && prime[stoi(s.substr(i-3,3))]) {
            dp[i] = (dp[i] + dp[i-3]) % mod;
        }
    }
    return dp[n];
}
```

Subtree of Another Tree

```
bool isIdentical(TreeNode* s, TreeNode* t) {
    if (s==NULL && t==NULL) {
        return true;
    }
    if ((s != NULL && t == NULL) || (s == NULL && t != NULL)) {
        return false;
    }
    if (s->val == t->val && isIdentical(s->left, t->left) && isIdentical(s->right, t->right)) {
        return true;
    }
    return false;
}

bool isSubtree(TreeNode* s, TreeNode* t) {
    if ((s != NULL && t == NULL) || (s == NULL && t != NULL)) {
        return false;
    }
    bool x;
    if (s->val == t->vall) {
        x = isIdentical(s, t);
    }
    bool l = isSubtree(s->left, t);
    bool r = isSubtree(s->right, t);

    return x||l||r;
}
```

Subtree with Max Average / Highest Tenure

```
struct TreeNode {
    int val;
    vector<shared_ptr<TreeNode>> child;
    TreeNode(int v): val(v) {}
};
Class Solution {
    double maxAns;
    shared_ptr<TreeNode> ans;
    pair<int, int> dfs(shared_ptr<TreeNode> root) {
        if (root) {
            if (root->child.size() == 0) {
                return {1, root->val};
            }
            int cost = root->val;
            int nodes = 1;
            for (int i = 0; i < root->child.size(); i ++) {
                auto p = dfs(root->child[i]);
                cost += p.second;
                nodes += p.first;
            }
            if (maxAns < double(cost*1.0f / nodes*1.0f)) {
                maxAns = double(cost*1.0f / nodes*1.0f);
                ans = root;
            }
            return {nodes, cost};
        }
        return {0,0};
    }
```

```
  shared_ptr<TreeNode> maxAverageSubtree(shared_ptr<TreeNode> root) {
    if (!root) {
      return root;
    }
    dfs(root);
    return ans;
  }
}
```

Supplier Inventory / Find the Highest Profit

```
long supplierInventory(int numSupplier, vector<long> inventory, long order) {
  unordered_map<long, long> mp;
  long highest = 0;
  long profit = 0;

  for (int i = 0; i < inventory.size(); i++) {
    mp[inventory[i]]++;
    if (highest < inventory[i]) {
      highest = inventory[i];
    }
  }

  while(order > 0 && !mp.empty()) {
    long highestFreq = mp[highest];
    if (order > highestFreq) {
      profit += highest * highestFreq;
      order -= highestFreq;
      mp.erase(highest);
      if (mp.find(highest-1) != mp.end()) {
        mp[highest-1] += highestFreq;
      } else {
        mp[highest-1] = highestFreq;
      }
      highest--;
    } else {
      profit += highest * order;
      order = 0;
    }
  }

  return profit;
}
```

Turnstile

```
vector<int> solve(vector<int> &time, vector<int> &direction) {
    queue<pair<int, int>> exit;
    queue<pair<int, int>> enter; // pair<time, index>
    int n = time.size();
    vector<int> res(n);

    for (int i = 0; i < n; i ++) {
        if (direction[i]) {
            exit.push({time[i], i});
        } else {
            enter.push({time[i], i});
        }
    }

    int cur_time = 0;
    int pre_dir = -1;

    while(!exit.empty() || !enter.empty()) {
        // check for exit
        if (!exit.empty() && exit.front().first <= cur_time && (pre_dir == -1 || pre_dir == 1 || enter.empty()
|| (enter.front().first > cur_time))) {
            res[exit.front().second] = cur_time;
            pre_dir = 1;
            exit.pop();
        } else if (!enter.empty() && enter.front().first <= cur_time) { // check for enter
            res[enter.front().second] = cur_time;
            pre_dir = 0;
            enter.pop();
        } else {
            pre_dir = -1;
        }
        cur_time++;
    }
    return res;
}
```

Two Sum – Unique Pairs

```
int uniquePairs(vector<int> nums, int target) {
    vector<int> local;
    vector<int> seen;
    int count = 0;
    for (auto num: nums) {
        if (find(local.begin(), local.end(), target-num) != local.end() && find(seen.begin(), seen.end(), num)
== seen.end()) {
            seen.push_back(target - num);
            seen.push_back(num);
            count ++;
        } else if (find(local.begin(), local.end(), target-num) == local.end()) {
            local.push_back(num);
        }
    }
    return  count;
}
```

Find Related Products / Books

```
int dfs(string str, unordered_map<string, bool> &visited, unordered_map<string, vector<string>> &mp,
vector<string> &cur) {
    visited[str] = true;
    cur.push_back(str);
    for(auto nei: mp[str]) {
        if (visited[nei] != true) {
            dfs(nei, visited, mp, cur);
        }
    }
    return cur.size();
}

vector<string> findRelatedProducts(vector<vector<string>> graph) {
    unordered_map<string, bool> visited;
    unordered_map<string, vector<string>> mp;

    for (auto g: graph) {
        for (int i = 1; i < g.size(); i++) {
            mp[g[i]].push_back(g[i-1]);
            mp[g[i-1]].push_back(g[i]);
            visited[g[i]] = false;
            visited[g[i-1]] = false;
        }
    }

    vector<string> res;
    int max = 0;

    for (auto v: visited) {
        if (!v.second) {
            vector<string> cur;
            int size = dfs(v.first, visited, mp, cur);
            if (size > max) {
                max = size;
                res = cur;
            }
        }
    }
    return res;
}
```

Divisibility of String

```
int solve(string s1, string s2) {
    if ((s1.length() % s2.length()) != 0) {
        return -1;
    }

    int l2 = s2.length();
    for (int i = 0; i < s1.length(); i++) {
        if (s1[i] != s2[i%l2]) {
            return -1;
        }
    }

    for (int i = 0; i < s2.length(); i++) {
        int j = 0;
        for (; j < s2.length(); j++) {
            if (s2[j] != s2[j%(i+1)]){
                break;
            }
        }
        if (j == s2.length()) {
            return i+1;
        }
    }
    return -1;
}
```

Packaging Automation

```
int solve(int num, vector<int> arr) {
    sort(arr.begin(), arr.end());
    arr[0] = 1;
    for (int i = 1; i < num; i++) {
        if (arr[i] >= arr[i-1]+1) {
            arr[i] = arr[i-1]+1;
        } else {
            arr[i] = arr[i-1];
        }
    }
    return arr[num-1];
}
```