Разбор задачи «Объединение двусвязных списков»

Для решения данной задачи заведём массив $used$, где в $used[i]$ будем хранить информацию о том, выписана ли i-я ячейка памяти.

Будем выполнять следующий алгоритм. Найдем любую ячейку, у элемента в которой нет левого соседа и ячейка ещё не выписана. Если такой ячейки нет, то нужно перейти к выводу ответа. В противном случае, пусть эта ячейка имеет номер pos. Тогда добавим pos в ответный список, выполним $used[pos] = true$ и присвоим $pos = r[pos]$, где r — массив, в котором хранятся правые (следующие) ячейки памяти. Будем выполнять описанную процедуру перехода до тех пор, пока у элемента pos есть правый (следующий) сосед.

После описанных операций, мы получаем ответный список элементов, для которого осталось запомнить информацию о предыдущих и следующих элементах. Это можно сделать за они проход с помощью двух вспомогательных векторов $ansL$ и $ansR$, в которых мы будем хранить предыдущих и следующих соседей для ответного списка. Пример построения ответных векторов показан ниже; ans — это вектор, в котором хранится ответный список.

```
vector<int> ansL(n, 0), ansR(n, 0);
for (int i = 0; i < int(ans.size()); i++) {
   if (i != 0)
      ansL[ans[i]] = ans[i - 1] + 1;
   if (i != n - 1)
      ansR[ans[i]] = ans[i + 1] + 1;
}
```

Разбор задачи «Подготовка к сортировке слиянием»

Для решения данной задачи будем делать sqrt-декомпозицию. Разобьём данный массив на блоки длины $sqrt(n)$. Для каждого блока будем хранить максимальный элемент, который в нем встречается.

Проитеруремся переменной i от 1 до n. Если очередное число уже было выписано — пропустим его. В противном случае, пусть текущее число равно cur. Выпишем его в новый список. Проитерируемся до конца текущего блока и найдём число, большее cur. Пусть это число в позиции pos. Если такого нет, проитерируемся по блокам, правее текущего, и найдём первый, максимум для которого больше cur. Если такого блока нет, то просто перейдем к следующему значению i. В противном случае, проитерируемся по самому блоку и найдем первое число, которое больше cur. Пусть оно находится в позиции pos.

Таким образом, в pos будет храниться ближайшее справа число большее cur. Поэтому нужно присвоить $cur = a[pos]$, выписать $a[pos]$, запомнить что мы выписывали это число (например, с помощью массива $used$ типа $bool$), и вновь искать для cur ближайшее справа число, большее чем cur.

Основной фрагмент решения представлен ниже.

```
szbl = (int)sqrt(n + 0.0);
for (int i = 0; i < n; i++)
   mabl[i / szbl] = max(mabl[i / szbl], a[i]);
for (int i = 0; i < n; i++) {
   if (a[i] == -1)
      continue;
   int last = a[i];
   cout << last << ' ';
   for (int j = i + 1; j < min(n, (i / szbl + 1) * szbl); j++) {
      if (a[j] > last) {
         cout << a[j] << ' ';
         last = a[j];
         a[j] = -1;
      }
   }
```

```
    for (int j = i / szbl + 1; j < (n / szbl) + 1; j++) {
       if (mabl[j] < last)
          continue;
       for (int k = j * szbl; k < min(j * szbl + szbl, n); k++)
       {
          if (a[k] > last) {
             cout << a[k] << ' ';
             last = a[k];
             a[k] = -1;
          }
       }
       mabl[j] = -1;
       for (int k = j * szbl; k < min(j * szbl + szbl, n); k++)
          mabl[j] = max(mabl[j], a[k]);
       }
       cout << endl;
    }
```

Разбор задачи «Суммарная вложенность»

Будем хранить в переменной *op* текущее количество открывающих скобок, для которых нет парных закрывающих. Изначально *op* равно нулю. Проитерируемся по i от 1 до n. Пока *op* больше k будем добавлять в ответ закрывающую скобку (так как иначе итоговая суммарная вложенность будет гарантированно больше k) и уменьшать *op* на единицу. После этого вычтем из k значение *op* и добавим новую открывающую скобку в ответ, при этом увеличим *op* на единицу.

После этого нужно добавить *op* закрывающих скобок в ответ, чтобы у всех открывающих была парная закрывающая.

Если $k = 0$, то выведем ответ. В противном случае, ответа нет, и нужно вывести `Impossible`. Ниже представлен основной фрагмент решения.

```
int op = 0;
for (int i = 0; i < n; i++) {
   while (op > k) {
      ans.push_back(')');
      --op;
   }
   k -= op;
   ++op;
   ans.push_back('(');
}
for (int i = 0; i < op; i++)
   ans.push_back(')');
```

Разбор задачи «Дог-шоу»

Заметим, что вместо того чтобы ждать у каждой миски отдельно - собака может подождать x секунд в начале, и потом бежать вправо, съедая по пути все готовые миски.

Найдем для каждой миски с номером i отрезок которому должен принадлежать x, чтобы согбака эту миску съела. Чтобы успеть съесть миску до истечения времени, x должен строго меньше $T - i$. Чтобы еда успела отсыть, x должен быть не меньше $t_i - i + 1$. Также x не может быть меньше 0.

Получив отрезки для каждой миски, надо найти точку покрытую наибольшим количеством отрезков. Для этого создадим список событий открытия и закрытия отрезков. Отсортируем этот список по позиции события. Если у двух событий позиция одинаковая, событие закрытия отрезка должно идти раньше. После этого пройдем по получившемуся списку слева направа, поддерживая количество открытых отрезков, прибавляя к нему 1 в случае события открытия, вычитая 1 в случае закрытия. Максимальное значение количества открытых отрезков во время такого прохода и будет ответом на задачу.

Разбор задачи «Пакманы»

Изначально сохраним в векторе $posP$ позиции всех Пакманов, а в $posA$ — всех звёздочек. Нужно быть уверенными, что в обоих векторах все элементы отсортированы по возрастанию.

Будем делать бинарный поиск по ответу. Пусть текущий ответ равен t. Тогда проитерируемся по Пакманам слева направо. Пусть текущий Пакман имеет номер i, а самая левая звёздочка, которую еще не съели, имеет номер j (то есть находится в позиции $posA[j]$). Тогда найдем сколько звёздочек успеет съесть Пакман i, если обязательно нужно съесть звездочку j за t секунд. Если он не успеет съесть j-ю звёздочку, то перейдем к следующему Пакману. Если же Пакман i успеет съесть j-ю звёздочку, то найдем максимальное количество здёздочек d, которые он может съесть за t секунд, начиная со звёздочки j. После этого присвоим $j = j + d$ и перейдем к следующему Пакману.

Ниже представлена функция, вычисляющая минимальное время, за которое Пакман в позиции i съесть cnt звёздочек подряд, начиная со звёздочки j.

```
int getT(int i, int j, int cnt) {
   if (cnt == 0)
      return 0;
   if (posP[i] < posA[j])
      return posA[j + cnt - 1] - posP[i];
   if (posP[i] > posA[j + cnt - 1])
      return posP[i] - posA[j];
   return min(posP[i] - posA[j] + posA[j + cnt - 1] - posA[j], posA[j + cnt - 1] -
posP[i] + posA[j + cnt - 1] - posA[j]);
}
```

Если после рассмотрения всех Пакманов все звёздочки будут съедены, то нужно сдвинуть правую границу бинарного поиска в t, так как мы минимизируем время. В противном случае, нужно сдвинуть левую границу в t.

Разбор задачи «Выборы в Берляндии»

Для решения данной задачи для каждого кандидата будем хранить количество отданных за него голосов, номер жителя, который проголосовал за него последним, а также номер самого кандидата. Затем отсортируем всех канидидатов по убыванию голосов, отданных за него, а при равенсте голосов — по времени последнего голосовавшего за кандидата жителя.

Затем проитерируемся по лидирующим k кандидатам. Если за очередного кандидата не отдано ни одного голоса и $n > 1$, то прекратим итерацию. Для кандидата i проверим сколько голосов должны получить следующие за ним кандидаты так, чтобы он занял в результате $(k + 1)$-е место. Если это количество не превышает $(m - a)$, то текущий кандидат i может быть как выбран, так и не выбран в парламент. В противном случае, он гарантировано будет выбран в парламент. Так как мы говорили вначале, что будем хранить номера кандидатов, то просто запишем в ответный вектор число 1 или 2, в зависимости от рассмотренных выше условий.

Осталось рассмотреть оставшихся кандидатов, для которых еще не был определен ответ. Если оставшихся $(m - a)$ голосов достаточно, чтобы кандидат набрал столько же голосов, сколько и кандидат, находящийся на k-месте, то текущий кандидат еще может быть избран и для него нужно записать в ответный вектор 2. В противном случае, для него нужно записать 3, так как он точно не будет избран.

Ниже представлено решения данной задачи. Ответ хранится в векторе r.

```
vector<pair<pair<int,int>,int>> c(n);
for (int i = 0; i < n; i++)
   c[i].second = i;
for (int i = 0; i < a; i++) {
   int x;
   cin >> x;
   x--;
   c[x] = make_pair(make_pair(c[x].first.first - 1, i), x);
}
```

```
sort(c.begin(), c.end());
vector<int> r(n);
int f = 0;
for (int i = 0; i < k; i++) {
   if (c[i].first.first == 0 && n > 1)
      break;
   f++;
   int t = 0, min_add = 0;
   for (int j = i + 1; t < k - i && j < min(n, k + 1); j++)
      t++, min_add += c[j].first.first - c[i].first.first + 1;
   if (t == k - i && min_add <= m - a)
      r[c[i].second] = 2;
   else
      r[c[i].second] = 1;
}
for (int i = f; i < n; i++) {
   if (c[i].first.first - (m - a) < c[k - 1].first.first)
      r[c[i].second] = 2;
   else
      r[c[i].second] = 3;
}
```

Разбор задачи «Пары в университете»

Это была самая простая задача соревнования. Для её решения нужно было найти максимальное количество единиц в столбце. Для этого переберем столбцы от 1 до 7. Затем за один проход по всем строкам от 1 до n посчитаем количество единиц в текущем столбце и обновим ответ получившимся числом. Ниже представлено решение данной задачи.

```
int ans = 0;
for (int j = 0; j < 7; j++) {
   int cnt = 0;
   for (int i = 0; i < n; i++) {
      if (a[i][j] == '1') cnt++;
   }
   ans = max(ans, cnt);
}
```

Разбор задачи «Нагрузочное тестирование»

Насчитаем четыре массива l, r, $numL$ и $numR$. В $l[i]$ будем хранить количество запросов, которые нужно сделать, чтобы префикс до позиции i, включительно, строго возрастал, а в $numL[i]$ — минимальное число, которое окажется после выполнения запросов в позиции i. В $r[i]$ будем хранить количество запросов, которые нужно сделать, чтобы суффикс с позиции i строго убывал, а в $numR[i]$ — минимальное число, которое окажется после выполнения запросов в i.

Каждый из массивов l и r можно насчитать за один проход. Массив l строится следующим образом — переберем i от 1 до n. Если текущее число $a[i]$ больше $numL[i-1]$, то никаких запросов больше делать не нужно и переходим к следующему числу. В противном случае, нужно сделать $numL[i-1] - a[i] + 1$ дополнительных запросов. Массив r строится аналогично, но итерироваться нужно справа налево.

Ниже представлен пример построения массивов l и $numL$.

```
l[0] = 0;
numL[0] = a[0];
for (int i = 1; i < n; i++) {
   if (a[i] > numL[i - 1]) {
      l[i] = l[i - 1];
      numL[i] = a[i];
```

```
    }
    else {
       l[i] = l[i - 1] + numL[i - 1] - a[i] + 1;
       numL[i] = numL[i - 1] + 1;
    }
}
```

После этого переберем все позиции i, в которых должен достигаться пик возрастания и начало убывания, и обновим ответ следующей величиной: если $numL[i] > numR[i]$, то ответ обновляем величиной $l[i] + r[i + 1]$; если $numL[i] < numR[i]$, то ответ обновляем величиной $l[i - 1] + t[i]$; в противном случае, нужно обновить ответ обеими вышеперечисленными величинами.

Разбор задачи «Уровень шума»

Данная задача на реализацию. В ней нужно было запускать обход в ширину из каждого шумного квартала. Будем хранить в двумерном массиве суммарный уровень шума в каждом в квартале. В обходе в ширину для каждого квартала нужно использовать массив *used*, где мы будем хранить — были ли мы в клетке для текущего обхода в ширину, чтобы не прибавить несколько раз шум от текущего квартала к одной и той же клетке. Также нужно помнить, что есть кварталы-препятствия для шума, в них шум проникать не может.

После этого нужно лишь посчитать количество клеток в массиве суммарного шума, для которых уровень шума выше допустимого, и вывести ответ.

Разбор задачи «Посвящение в студенты»

Сделаем бинарный поиск по ответу, внутри которого будем проверять, возможно ли сделать так, чтобы минимальное количество сувениров, подаренных самым невезучим первокурсником было не более mid.

Построим сеть состоящую из $n + m + 2$ вершин - вершина для каждого первокурсника, вершина для каждой пары знакомых первокурсников, две дополнительные вершины (исток и сток).

Добавим ребра из истока во все вершины соответствующие первокурсникам пропускной способности mid, ребра из каждой вершины соответствующей знакомству в сток пропускной способности 1. Также из каждой вершины соответствующей первокурснику добавим ребра во все вершины соответствующие знакомствам этого первокурсника пропускной способности 1.

Если максимальный поток равен m, то ответ меньше или равен mid, иначе ответ больше mid.

Разбор задачи «Проездные билеты»

Задача на реализацию. Для каждого маршрута автобуса, посчитаем сумму, которая нужна для проезда по нему, без учёта проездных. Это можно сделать с помощью *map*, где ключом будет упорядоченная пара строк (названия объектов), между которыми был совершен очередной переезд. Не забываем проверять, текущий переезд новый, или же это пересадка.

Сохраним все суммы для каждого маршрута в вектор *costs*. Отсортируем его по невозрастанию. После этого, проитеруемся по префиксу длины k вектора *costs*. Если очередное число больше f, то выгоднее купить проездной для текущего маршрута. Поэтому заменим текущее число в *costs* на f.

После этого осталось вывести ответ — это сумма всех элементов в векторе *costs*.

```
map<pair<string,string>, int> totals;
string py;
for (int i = 0; i < n; i++) {
   string x, y;
   cin >> x >> y;
   int c;
   if (x == py)
      c = b;
   else
      c = a;
   totals[make_pair(min(x, y), max(x, y))] += c;
```

```
    py = y;
}
int ans = 0;
vector<int> costs;
for (auto kv: totals)
    costs.push_back(kv.second), ans += kv.second;
sort(costs.begin(), costs.end(), std::greater<int>());
for (int i = 0; i < min(k, int(costs.size()); i++)
    if (costs[i] > f)
        ans -= costs[i] - f;
cout << ans << endl;
```

Разбор задачи «Компьютерная сеть Берляндского ГУ»

Выполним $n - 1$ итерацию и на каждой будем пытаться добавить в дерево ровно одно ребро и удалять из данных листов ровно одну вершину. Если на какой-то итерации этого сделать не получится, то ответ `-1`. Будем использовать вспомогательный массив del, где будем хранить какие из вершин удалены.

Рассмотрим что нужно делать на каждой итерации. Переберем все неудалённые из списков вершины. Пусть текущая вершина равна i. Тогда если хотя бы в одном из её списков есть ровно одна вершина, назовём её j, то ребро между i и j гарантировано есть в ответном дереве. Поэтому добавим его в ответ, удалим вершину j из списков (а точнее пометим информацию о том, что она удалена, в массиве del, и перейдем к следующей итерации.

Если на каждой из n итераций мы нашли ребро, то осталось проверить, получилось ли корректное дерево. Сначала проверим получившийся граф на связность. Если не связный — ответ `-1`. В противном случае, с помощью обхода в глубину, построим списки для каждой вершины, аналогично заданным в условии спискам. После этого сравним списки из входных данных и списки, получившиеся после построения ответного дерева. Если они не равны, то ответа не существует. В противном случае, нужно просто вывести все рёбра получившегося дерева.

Разбор задачи «Погода на завтра»

Если для каждого i от 3 до n верно, что $a[i] - a[i-1] = a[2] - a[1]$, то задана арифметическая прогрессия, и нужно вывести её $(n+1)$-й член равный $a[n] + a[2] - a[1]$. В противном случае, нужно вывести $a[n]$. Ниже представлено решение данной задачи.

```
for (int i = 2; i < n; i++)
    if (a[i] - a[i - 1] != a[1] - a[0]) {
        cout << a[n - 1] << endl;
        return 0;
    }
cout << a[n - 1] + a[1] - a[0] << endl;
```