Soluciones OIFem 2020

Concurso de Práctica 1

Moda-GCF

Teoría

- Vectores
- Ordenamiento
- Bucles

Solución

Ordenamos el vector, de forma que las casillas con el mismo número sean contiguas. Así, iteramos posición a posición, viendo cuántas copias hay de cada número y guardándonos el récord hasta ese momento. Una vez hayamos recorrido todo el vector, habremos encontrado la moda.

Código

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

int main() {
        ios::sync_with_stdio(false);
        cin.tie(NULL);
        int n;
        cin >> n;
        vector<int> nums(n);
        for (int i = 0; i < n; i++)
                cin >> nums[i];
        sort(nums.begin(), nums.end());
        int record = 1, moda = nums[0], cur = 1;
        for (int i = 1; i < n; i++) {
                if (nums[i] == nums[i-1]) {
                        cur++;
                        if (cur > record) {
                                record = cur;
                                moda = nums[i];
                        }
                } else
```

```
                        cur = 1;
        }
        cout << moda;
        return 0;
}
```

Secuencias saltarinas

Teoría

- Conjuntos desordenados

Solución

Iteramos sobre el vector, guardando el valor absoluto de cada diferencia.

Una vez hayamos terminado este primer bucle, iteramos por todos números en orden ascendente de 1 a n hasta encontrar uno que falte y lo imprimimos si lo encontramos. En caso contrario, diremos que la secuencia es saltarina.

Código

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

int main() {
        ios::sync_with_stdio(false);
        cin.tie(NULL);
        int N;
        cin >> N;
        int nums[N];
        for (int i = 0; i < N; i++)
                cin >> nums[i];
        unordered_set<int> diffs;
        for (int i = 0; i < N-1; i++)
                diffs.insert(abs(nums[i]-nums[i+1]));
        bool saltarina = true;
        for (int i = 1; i < N; i++) {
                if (diffs.find(i) == diffs.end()) {
                        saltarina = false;
                        cout << i;
                        break;
                }
        }
        if (saltarina)
                cout << "0";
        return 0;
}
```

Promedios en intervalos

Teoría

- Programación dinámica
- Búsqueda binaria

Solución

Primero, ordenamos los alumnos por orden de lista y rellenamos un vector con las *prefix sums* de las notas. Para cada intervalo que nos pregunten, usaremos búsqueda binaria para encontrar el índice inferior y el índice superior en nuestra lista, usando esos índices para obtener la suma rápidamente de la DP. Sabiendo la suma, basta con imprimirla dividiendo por la longitud del intervalo.

Código

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
typedef long long int ll;

int primerEstudiante(int I, vector<pair<int, ll>> & alumnos) {
        int lo = 0, hi = (int) alumnos.size()-1, ret = -1, mid;
        while(lo <= hi) {
                mid = lo + (hi-lo)/2;
                if (alumnos[mid].first >= I) {
                        ret = mid;
                        hi = mid-1;
                } else lo = mid+1;
        }
        return ret;
}

int ultimoEstudiante(int U, vector<pair<int, ll>> & alumnos) {
        int lo = 0, hi = (int) alumnos.size()-1, ret = -1, mid;
        while(lo <= hi) {
                mid = lo + (hi-lo)/2;
                if (alumnos[mid].first <= U) {
                        ret = mid;
                        lo = mid+1;
                } else hi = mid-1;
        }
        return ret;
}

int main() {
        ios::sync_with_stdio(false);
        cin.tie(NULL);
        int N, M, I, U, startPos, endPos;
        cin >> N;
        vector<pair<int, ll>> alumnos(N);
        for (int i = 0; i < N; i++)
```

```
38                  cin >> alumnos[i].first >> alumnos[i].second;
39          sort(alumnos.begin(), alumnos.end());
40          ll sumas[N+1];
41          sumas[0] = 0;
42          for (int i = 1; i <= N; i++)
43                  sumas[i] = sumas[i-1] + alumnos[i-1].second;
44          cin >> M;
45          while(M--) {
46                  cin >> I >> U;
47                  startPos = primerEstudiante(I, alumnos);
48                  endPos = ultimoEstudiante(U, alumnos);
49                  if (startPos == -1 || endPos == -1 || endPos < startPos) cout <<
                    ↪  "-1\n";
50                  else cout << (sumas[endPos+1] -
                    ↪  sumas[startPos])/(endPos-startPos+1) << '\n';
51          }
52          return 0;
53  }
```

Perfectamente balanceado

Teoría

- Programación dinámica

Solución

La diferencia entre todas las posiciones contiguas de la lista debe ser la misma que la diferencia entre los dos primeros elementos. Por lo tanto, probaremos las 9 posibilidades (-1, 0 y +1 para ambos elementos), iterando elemento a elemento por la lista viendo si es posible alcanzar esa diferencia en el elemento actual, habiendo ya fijado todos los anteriores. Nos quedaremos con la mejor de las 9 posibilidades e imprimiremos -1 si ninguna es válida.

Código

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

int main() {
        ios::sync_with_stdio(false);
        cin.tie(NULL);
        vector<int> numeros, nuevos;
        int n;
        cin >> n;
        numeros = vector<int>(n);
        nuevos = vector<int>(n);
        for (int i = 0; i < n; i++)
                cin >> numeros[i];
        int record = 1e9, actual = 0, diferencia;
        // probamos las 9 opciones para la primera diferencia
        for (int dif1 = -1; dif1 <= 1; dif1++) {
                nuevos[0] = numeros[0] + dif1;
                for (int dif2 = -1; dif2 <= 1; dif2++) {
                        actual = abs(dif1) + abs(dif2);
                        nuevos[1] = numeros[1]+dif2;
                        diferencia = nuevos[1]-nuevos[0]; // guardamos la
                        ↪  diferencia buscada
                        bool posible = true;
                        for (int i = 2; i < n; i++) {
                                if (abs(numeros[i] - nuevos[i-1] - diferencia) <=
                                ↪  1) {
                                        // es posible mantener la diferencia en
                                        ↪  esta nueva posición
                                        nuevos[i] = nuevos[i-1] + diferencia;
                                        actual += abs(numeros[i] - nuevos[i-1] -
                                        ↪  diferencia);
                                } else {
                                        // no es posible: cortamos
                                        posible = false;
                                        break;
                                }
                        }
```

```
                        if (posible)
                                record = min(record, actual);
                }
        }
        if (record == 1e9)
                record = -1;
        cout << record << '\n';
        return 0;
}
```

Eligiendo el horario

Teoría

- Algoritmo de Dijkstra

Solución

La solución consiste en hacer Dijkstra partiendo desde el lugar de destino, usando una función para ver el último bus antes de una hora en concreto. Podemos hacer esto, ya que buscar la forma más rápida de llegar desde la hora final y destino implica llegar *tarde* a la casilla de inicio.

Código

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

typedef long long int ll;

struct bus {
        int u;
        ll tiempo, primero, periodicidad;
};

typedef vector<bus> vb;
typedef vector<vb> vvb;

int N;
ll L;
vvb G;

ll siguienteBus(ll curT, ll primerBus, ll diff, ll dur) {
        if (curT < primerBus + dur) return -1;
        ll val = (curT - (primerBus+dur))/diff;
        return primerBus + val*diff;
}

ll shortest() {
        vector<ll> dist;
        dist.assign(N, -1);
        dist[N-1] = L;
        queue<pair<ll, int>> Q; // {tiempo, nodo}
        Q.push(make_pair(L, N-1));
        pair<ll, int> rutaPosible;
        int u;
        ll distU, siguiente;
        bus busActual;
        while(!Q.empty()) {
                rutaPosible = Q.front();
                Q.pop();
                u = rutaPosible.second;
```

```
                distU = rutaPosible.first;
                if (distU < dist[u]) continue;
                for (auto busActual: G[u]) {
                        siguiente = siguienteBus(distU, busActual.primero,
                        ↪  busActual.periodicidad, busActual.tiempo);
                        if (siguiente == -1) continue;
                        if (siguiente > dist[busActual.u]) {
                                dist[busActual.u] = siguiente;
                                Q.push(make_pair(dist[busActual.u], busActual.u));
                        }
                }
        }
        return dist[0];
}

int main() {
        ios::sync_with_stdio(false);
        cin.tie(NULL);
        int M, u, v;
        ll tiempo, primero, periodicidad;
        cin >> N >> M >> L;
        G.assign(N, vb());
        bus BB;
        while(M--) {
                cin >> u >> v >> tiempo >> primero >> periodicidad;
                BB.u = u;
                BB.tiempo = tiempo;
                BB.primero = primero;
                BB.periodicidad = periodicidad;
                G[v].push_back(BB);
        }
        cout << max((ll) -1, shortest()) << "\n";
        return 0;
}
```

Plantas extrañas

Teoría

- Árboles de segmentos

Solución

Para resolver este problema, crearemos un árbol de segmentos, donde la función sea la suma y los valores en los nodos, las diferencias. Cuando cambie la altura de un nodo, cambiaremos las diferencias entre esa planta y su izquierda y su derecha, si es que existen.

Código

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

vector<int> alturas, seg, diferencias;
vector<pair<int, pair<int, int>>> cambios, preguntas;

int hijoIzquierdo(int p) {
        return p << 1;
}

int hijoDerecho(int p) {
        return (p << 1)+1;
}

int funcion(int valorIzq, int valorDer) {
        return valorIzq + valorDer;
}

void construirArbol(int indiceNodo, int L, int R) {
        if (L < R) {
                int mid = L + (R-L)/2;
                construirArbol(hijoIzquierdo(indiceNodo), L, mid);
                construirArbol(hijoDerecho(indiceNodo), mid + 1, R);
                seg[indiceNodo] = funcion(seg[hijoIzquierdo(indiceNodo)],
                ↪  seg[hijoDerecho(indiceNodo)]);
        } else
                seg[indiceNodo] = diferencias[L];
}

int conseguirValor(int indiceNodo, int L, int R, int indicePrincipio, int
↪  indiceFinal) {
        if (L >= indicePrincipio && R <= indiceFinal) {
                return seg[indiceNodo];
        } else if (indicePrincipio > R || indiceFinal < L) {
                return -1;
        } else {
                int mid = L + (R-L)/2;
```

```
38                  int valorHijoIzquierdo = conseguirValor(hijoIzquierdo(indiceNodo),
                    ↪  L, mid, indicePrincipio, indiceFinal);
39                  int valorHijoDerecho = conseguirValor(hijoDerecho(indiceNodo), mid
                    ↪  + 1, R, indicePrincipio, indiceFinal);
40                  if (valorHijoIzquierdo == -1)
41                          return valorHijoDerecho;
42                  else if (valorHijoDerecho == -1)
43                          return valorHijoIzquierdo;
44                  else
45                          return funcion(valorHijoIzquierdo, valorHijoDerecho);
46          }
47  }
48
49  void cambio(int indiceNodo, int L, int R, int indiceCambiado, int nuevoValor) {
50      if (indiceCambiado < L || indiceCambiado > R)
51          return;
52      if (L == R) {
53          seg[indiceNodo] = diferencias[indiceCambiado] = nuevoValor;
54      } else {
55          int mid = L + (R-L)/2;
56          cambio(hijoIzquierdo(indiceNodo), L, mid, indiceCambiado, nuevoValor);
57          cambio(hijoDerecho(indiceNodo), mid + 1, R, indiceCambiado, nuevoValor);
58          seg[indiceNodo] = funcion(seg[hijoIzquierdo(indiceNodo)],
            ↪  seg[hijoDerecho(indiceNodo)]);
59      }
60  }
61
62  int main() {
63          int n, u, q, j;
64          cin >> n >> u >> q;
65          alturas = vector<int>(n);
66          for (int i = 0; i < n; i++)
67                  cin >> alturas[i];
68          diferencias = vector<int>(n-1);
69          for (int i = 0; i < n-1; i++)
70                  diferencias[i] = abs(alturas[i+1]-alturas[i]);
71          seg.assign(4*n, 0);
72          construirArbol(1, 0, n-2);
73          cambios = vector<pair<int, pair<int, int>>>(u);
74          for (int i = 0; i < u; i++)
75                  cin >> cambios[i].second.first >> cambios[i].second.second >>
                    ↪  cambios[i].first;
76          preguntas = vector<pair<int, pair<int, int>>>(q);
77          for (int i = 0; i < q; i++)
78                  cin >> preguntas[i].second.first >> preguntas[i].second.second >>
                    ↪  preguntas[i].first;
79          sort(cambios.begin(), cambios.end());
80          sort(preguntas.begin(), preguntas.end());
81          j = 0;
82          for (int i = 0; i < q; i++) {
83                  while(j < u && cambios[j].first <= preguntas[i].first) {
84                          int ind = cambios[j].second.first-1;
85                          alturas[ind] += cambios[j].second.second;
86                          if (ind < n-1)
```

```
87                                  cambio(1, 0, n-2, ind, abs(alturas[ind] -
                                    ↪  alturas[ind+1]));
88                          if (ind > 0)
89                                  cambio(1, 0, n-2, ind-1, abs(alturas[ind-1] -
                                    ↪  alturas[ind]));
90                          j++;
91                  }
92                  cout << conseguirValor(1, 0, n-2, preguntas[i].second.first-1,
                    ↪  preguntas[i].second.second-2) << '\n';
93          }
94          return 0;
95  }
```