Soluciones OIFem 2020

Apoyo Entrenamiento 5

Zeros and Ones

Teoría

- Backtracking: subconjuntos

Solución

Para este problema, alteraremos la función de backtracking para encontrar subconjuntos de una lista de longitud N de forma que, con cada paso que demos, guardemos 0 para los "valores no cogidos" y 1 para los "valores cogidos" en una lista. Una vez *ind* sea igual a N, paramos e imprimimos la lista de ceros y unos.

Código

C++

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

void backtrack(int N, vector<int> & nums, int ind) {
        if (ind == N) {
                cout << nums[0];
                for (int i = 1; i < N; i++)
                        cout << ' ' << nums[i];
                cout << '\n';
                return;
        }
        // Probamos con 0 en la posición ind
        nums[ind] = 0;
        backtrack(N, nums, ind+1);
        // Probamos con 1 en la posición ind
        nums[ind] = 1;
        backtrack(N, nums, ind+1);
}
```

```cpp
int main() {
        int N;
        cin >> N;
        vector<int> nums(N, 0);
        backtrack(N, nums, 0);
        return 0;
}
```

Subsets (1)

Teoría

- Backtracking: subconjuntos

Solución

Usamos el código visto en clase para encontrar subconjuntos.

Código

C++

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

void subconjuntos(vector<int> & subconjunto, vector<string> & lista, int ind) {
        if (ind == (int) lista.size()) {
                // imprimimos el subconjunto
                cout << '{';
                for (int i = 0; i < (int) subconjunto.size() - 1; i++)
                        cout << lista[subconjunto[i]] << ',';
                if (subconjunto.size())
                        cout << lista[*subconjunto.rbegin()]; // forma rápida de
                        ↪  obtener el último elemento de subconjunto
                cout << "}\n";
                return;
        }

        // subconjuntos sin el elemento
        subconjuntos(subconjunto, lista, ind+1);

        // subconjuntos con el elemento
        subconjunto.push_back(ind);
        subconjuntos(subconjunto, lista, ind+1);
        subconjunto.pop_back();
}

int main() {
        int N;
        cin >> N;
        vector<string> palabras(N);
        for (int i = 0; i < N; i++)
                cin >> palabras[i];
        vector<int> subconj;
        subconjuntos(subconj, palabras, 0);
        return 0;
}
```

Multisets (1)

Teoría

- Backtracking: subconjuntos

Solución

Este problema es una variación de backtracking con subconjuntos. Iteramos sobre los números de 1 a N, representados por la variable num. Para cada número, probamos a añadirlo entre 0 y X veces, encontrando así todos los multiconjuntos.

Código

C++

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

void buscaMulticonjuntos(int N, int X, int num, vector<int> & multiconjunto) {
        if (num == N+1) {
                // hemos llegado al final -> imprimimos el subconjunto
                cout << '{';
                if ((int) multiconjunto.size() >= 1) {
                        cout << multiconjunto[0];
                        for (int i = 1; i < (int) multiconjunto.size(); i++) cout
                        ↪  << "," << multiconjunto[i];
                }
                cout << "}\n";
                return;
        }
        buscaMulticonjuntos(N, X, num+1, multiconjunto);
        for (int x = 1; x <= X; x++) {
                // añadimos num al multiconjunto x veces
                multiconjunto.push_back(num);
                buscaMulticonjuntos(N, X, num+1, multiconjunto);
        }
        for (int x = 1; x <= X; x++) multiconjunto.pop_back();
}
```

```cpp
int main() {
        int N, X;
        cin >> N >> X;
        if (X == 0)
                cout << "{}\n";
        else {
                vector<int> multiconjunto = {};
                buscaMulticonjuntos(N, X, 1, multiconjunto);
        }
        return 0;
}
```

From one to en

Teoría

- Backtracking: permutaciones

Solución

Usamos la función de backtracking para permutaciones aprendida en clase. No nos hace falta guardar los números en una lista, ya que con saber N sabemos qué números usar en la permutación (los números de 1 a N).

Código

C++

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

void permutaciones(vector<int> & permutacion, vector<bool> & seleccionado, int N) {
        if ((int) permutacion.size() == N) {
                // ya están todos los números en la permutación
                cout << '(' << permutacion[0];
                for (int i = 1; i < (int) permutacion.size(); i++)
                        cout << ',' << permutacion[i];
                cout << ")\n";
                return;
        }

        for (int i = 1; i <= N; i++) {
                if (!seleccionado[i-1]) {
                        seleccionado[i-1] = true;
                        permutacion.push_back(i);
                        permutaciones(permutacion, seleccionado, N);
                        permutacion.pop_back();
                        seleccionado[i-1] = false;
                }
        }
}


int main() {
        int N;
        cin >> N;
        vector<int> permutacion = {};
        vector<bool> seleccionado = vector<bool>(N, false);
        permutaciones(permutacion, seleccionado, N);
        return 0;
}
```

Permutations of words

Teoría

- Backtracking: permutaciones

Solución

Usamos la función aprendida en clase, esta vez guardando una lista de palabras en vez de una de números. Para tardar un poco menos y ocupar menos espacio, en la permutación guardaremos el índice de cada palabra en lugar de la palabra.

Código

C++

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

void permutaciones(vector<int> & permutacion, vector<bool> & seleccionado,
↪  vector<string> & lista) {
        if (permutacion.size() == lista.size()) {
                // ya están todos los índices en la permutación
                cout << '(' << lista[permutacion[0]];
                for (int i = 1; i < (int) permutacion.size(); i++)
                        cout << ',' << lista[permutacion[i]];
                cout << ")\n";
                return;
        }
        for (int i = 0; i < (int) lista.size(); i++) {
                if (!seleccionado[i]) {
                        seleccionado[i] = true;
                        permutacion.push_back(i);
                        permutaciones(permutacion, seleccionado, lista);
                        permutacion.pop_back();
                        seleccionado[i] = false;
                }
        }
}

int main() {
        int N;
        cin >> N;
        vector<int> permutacion = {};
        vector<bool> seleccionado = vector<bool>(N, false);
        vector<string> palabras(N);
        for (int i = 0; i < N; i++)
                cin >> palabras[i];
        permutaciones(permutacion, seleccionado, palabras);
        return 0;
}
```