UNIVERSIDAD
DE GRANADA

Algoritmica
Grado en Ingeniería Informática
Curso 2018-2019

Ejercicio Propuesto.
Suma subconjuntos

Félix Ramírez García
felixramirezgarcia@correo.ugr.es

6 de mayo de 2019

Índice

Índice de figuras

Índice de tablas

1. Introducción

El problema de la suma de subconjuntos es un problema importante en la teoría de la complejidad y en la criptografía. El problema es este: dado un conjunto de enteros, ¿existe algún subconjunto cuya suma sea exactamente un determinado numero?. La suma de subconjuntos también puede verse como un caso especial del problema de la mochila.

El problema de la suma de subconjuntos es una buena introducción a los problemas NP-completos por dos razones:

Es un problema de decisión, no uno de optimización y su definición es simple. La mayoría de los problemas físicos pueden ser resueltos con un índice de error del 1 por ciento. Resolver un problema de suma de subconjuntos para 100 enteros con tanta precisión puede parecer irrelevante, pero no lo es por dos razones.

Primero, el problema de la suma de subconjuntos tiene una declaración precisa de la complejidad lógica de una clase de problemas (los NP-completos). Resolverlo exactamente significaría resolver todos los problemas en esta clase. Resolverlo con un margen de error de +/- 1 por ciento volvería inútil al algoritmo para algunos otros problemas. Segundo, en cuando menos un contexto, es de hecho importante resolver el problema de la suma de subconjuntos de manera exacta. En criptografía, este problema surge cuando un asaltacódigos trata de deducir la llave secreta a partir de un mensaje y su versión cifrada. Una llave a una distancia de +/- 1 por ciento de la real es inservible.

2. Solución fuerza bruta

Hay varias maneras de resolver la suma de subconjuntos en tiempo exponencial sobre N. El algoritmo más simplista verificaría todos los posibles subconjuntos de N y, para cada uno de ellos, compararía la suma al total buscado. El tiempo de ejecución es de orden O(2^N*N), dado que hay 2^N subconjuntos y, para verificar cada subconjunto, tenemos que sumar N elementos.

El código en c++ usado para solventar este enfoque del problema es el que se muestra a continuación:

```
#include <bits/stdc++.h>
#include <vector>
#include <iostream>
#include <map>
#include <stdlib.h>
#include <time.h>
#include <cmath>

using namespace std;
```

```
map<int, vector<int>> allPossibleSubset(vector<int> arr,int n)
{
        map<int, vector<int>> result;
        vector<int> v;
        int k = 0;
        double count = pow(2,n);

        for (int i = 0; i < count; i++)
        {
                for (int j = 0; j < n; j++)
                {
                        if ((i & (1 << j)) > 0){
                        v.push_back(arr[j]);
                }
        }
                result.insert(pair<int,vector<int>>(k,v));
                v.clear();
                k++;
        }

        return result;
}

int main(int argc, char* argv[])
{
        if(argc != 2){
                cout << "Faltan parametros" << endl;
                return -1;
        }

        int n = atoi(argv[1]);
        vector<int> arr;

        for(int i=1;i<=n;i++){
                arr.push_back(i);
        }

        map<int, vector<int>> result = allPossibleSubset(arr,n);

        vector<int> aux;
        int ale = rand()%(n);
        cout << "El tamanio es " << n << " y la suma a buscar es: " <<
            ale << endl;
        cout << "Los subconjuntos solucion (si existen) son: " << endl;

        for( const auto& pair : result )
        {
                int suma = 0;
                for( size_t i = 0 ; i < pair.second.size() ; ++i ){
                        suma += pair.second[i];
                        aux.push_back(pair.second[i]);
                }
                if(suma == ale){
```

```
                        for(int l = 0; l < aux.size(); l++)
                        {
                                cout << aux[l] << " ";
                        }
                        cout << endl;
                        aux.clear();
                }else{
                        aux.clear();
                }
        }

        return 0;
}
```

3. Solución 1 con backtraking

Para esta solución se ha implementado un código recursivo en el que se almacena un vector con los diferentes estados posibles, siendo el estado 2(nulo), estado 1 correcto y estado -1(fin). Esta solución tiene una eficiencia de O(N)) y los dos ficheros usados son los siguientes:

```
#include <iostream>
#include <vector>
#include <cstdlib>

using namespace std;

class solucion {     //estado 2(nulo),1,0,-1(fin)

        private:
        vector<int> x;
        vector<int> orig;
        int m;//suma

        public:
        solucion(int tama, int suma) { //constructor
                m=suma;
                for(int i=1; i<=tama; i++) {
                        orig.push_back(i);
                        x.push_back(2);//2=NULO
                }
        }

        void mostrar() {
                cout << "tamanio es " << orig.size() << " y la suma a
                    buscar es: "<< m << endl;
        }

        int size() const {
                x.size();
```

```
        }

        void iniciacomp(int k) {
                x[k]=2;//2=NULO
        }

        void sigvalcomp(int k) {
                x[k]--;
        }

        bool todosgenerados(int k) {
                return x[k]==-1;//devuelve true si estamos ya en el
                    ultimo.
        }

        int decision(int k) {
                return x[k];
        }

        void procesasolucion() {
                for(int i=0; i<size(); i++) {
                        if(x[i]==1) { //elemento selecionado
                                cout<<orig[i]<<"  ";
                        }
                }
                cout<<endl;
        }

        bool factible(int k) {
                int s=0,r=0,aux=0;
                for(int i=0; i<=k; i++) {
                        if(x[i]==1) { //elemento selecionado
                                s+=orig[i];
                        }
                }
                for(int i=k+1; i<orig.size(); i++) {
                        r+=orig[i];
                }
                if(k+1<orig.size()) aux=orig[k+1];

                return (((s+r>=m) and s+aux<=m) or (s==m));
        }

};
```

```
using namespace std;
#include "solucion.h"

void back_recursivo(solucion &sol,int k) {
        if(k==sol.size())   sol.procesasolucion();
        else {
                sol.iniciacomp(k);
```

```
                sol.sigvalcomp(k);
                while(!sol.todosgenerados(k)) {
                        if(sol.factible(k)) back_recursivo(sol,k+1);
                        sol.sigvalcomp(k);
                }
        }
}

int main(int argc, char *argv[]) {

        if(argc != 2){
                cout << "Faltan parametros" << endl;
                return -1;
        }

        int n = atoi(argv[1]);
        int suma = rand()%(n);
        solucion s(n,suma);
        int aux=0;
        s.mostrar();
        back_recursivo(s,aux);
        return 0;

}
```

4. Solución 2 con backtraking

Esta solución del problema es muy similar a la anterior , la diferencia es que ahora almacenamos ademas de los estados en un vector , la suma de lo que llevamos y la suma de lo que nos queda. Esta implementacion tiene mejor eficiencia que la anterior , ya que si k==size() ya no vamos a calcular k+1. Los dos ficheros usados son los siguientes:

```
#include <iostream>
#include <vector>
#include <cstdlib>

//estados 2(nulo),1,0,-1(end)
using namespace std;

class solucion {
private:
        vector<int> x;
        vector<int> orig;
        int m,s,r;
        //m es la suma
        //s es la suma de lo que llevamos
        //r es la suma de los que nos qeda.
public:
        solucion(int tama) { //constructor
                s=r=0;
```

```
        m=tama+tama/2;
        for(int i=1; i<=tama; i++) {
                orig.push_back(i);
                x.push_back(2);//2=NULO
                r+=i;
        }
}

void mostrar() {
        cout<<"tamanio es "<<size()<<" la suma es: "<<m<<" s="<<s
            <<" r="<<r<<endl;
}

int size() const {
        x.size();
}

void iniciacomp(int k) {
        x[k]=2;//2=NULO
}

void sigvalcomp(int k) {
        x[k]--;
        if(x[k]==1) {//si lo cogemos el valor en k, se lo sumamos
             a s y se lo restamos a r.
                s+=orig[k];
                r-=orig[k];
        }
        if(x[k]==0) {//si no cogemos le valor en k, se lo
            restamos a s, ( ya que lo habiamos introducido antes
            en la x[k]==1).
                //No es necesario sumarselo a r ya que no esta en
                     los aun no seleccioandos.
                s-=orig[k];
        }
}

bool todosgenerados(int k) {
        return x[k]==-1;//devuelve true si estamos ya en el
            ultimo.
}

int decision(int k) {
        return x[k];
}

void procesasolucion() {
        cout<<"solucion"<<endl;
        for(int i=0; i<size(); i++) {
                if(x[i]==1) { //elemento selecionado
                        cout<<orig[i]<<"  ";
                }
        }
        cout<<endl;
```

```
        }

        bool factible(int k) {
                return (((s+x[k]<=m) and (s+r>=m)) or (s==m));
        }

        void VueltaAtras(int k) {
                if(k!=size()) {//si k==size() ya no vamos a calcular k+1
                        r+=orig[k];//mantiene el en r el siguiente valor
                            ya que en el back_recursivo(k+1) se ha restado
                        x[k] = 2;//y pone k a nulo ya que en le
                            back_recursivo(k+1) se ha modificado.
                }
        }

};
```

```
using namespace std;
#include "solucion.h"

void back_recursivo(solucion &sol,int k) {
        if(k==sol.size())    sol.procesasolucion();
        else {
                sol.iniciacomp(k);
                sol.sigvalcomp(k);
                while(!sol.todosgenerados(k)) {
                        if(sol.factible(k)) {
                                back_recursivo(sol,k+1);
                                sol.VueltaAtras(k+1);//para que en las
                                    siguientes iteraciones no se tnega en
                                    cuenta k+1
                        }
                        sol.sigvalcomp(k);
                }
        }
}

int main(int argc, char *argv[]) {

        if(argc != 2){
                cout << "Faltan parametros" << endl;
                return -1;
        }

        int n = atoi(argv[1]);
        solucion s(n);
        int aux=0;
        s.mostrar();
        back_recursivo(s,aux);
        return 0;
}
```