Fondamenti di Programmazione 2

9 Dicembre 2022

Exercise 1

Let $x_1, ..., x_n$ be a set of integers. Write a backtracking function to compute all the possible ways it is possible to split the set into two distinct subsets S', S'' such that.

- $S' \cup S'' = S$
- $S' \cap S'' = \{\}$
- $\sum_{x \in S'} x = \sum_{x \in S''} x$

Exercise 2

Let G be a non oriented graph, V its node set, E its edge set. A set $W \subseteq V$ is called *independent set* if for all edges $(a, b) \in E$ it holds that $a \notin W$ or $b \notin W$.

1. Write a backtracking function that computes an independent set for an input graph.
2. Write a backtracking function that computes an independent set of maximum size for an input graph

Exercise 3

Write a program that takes in input a string S and a list of strings W and prints all permutations of S that do not contain strings of W as substrings. You can suppose S is made of distinct characters.

Esempio Consider the string `rane`, and `W = {na, re}`. This is the list of valid and non-valid permutations. (The program might aso print only the valid ones)

```
VALID
rane
raen
```

```
rnea
rean
aren
aner
aenr
aern
naer
nrae
nrea
nera
near
eanr
earn
enar
enra
eran

NOT VALID:
rnae
rena
arne
anre
nare
erna
```

Suggerimento: Suppose we know how to generate permutations for strings of length $k-1$. Given a string of length k, we can fix its first character, obtain a string of length $k-1$ and generate all its permutations. In this way, we obtain all the permutations for a string of length k. Furthermore, if $k \leq 1$, the string itself is its unique permutation. Let our string be `abc`, denote by `[]` fixed characters.

```
abc
    => [a]bc
        => [ab]c
            => [abc]
        => [ac]b
            => [acb]
    => [b]ac
        => [ba]c
            => [bac]
        => [bc]a
            => [bca]
    => [c]ba
        => [cb]a
            => [cba]
        => [ca]b
            => [cab]
```

The permutations of `abc` are `abc`, `acb`, `bac`, `bca`, `cba`, `cab`.

Esercizio 4

Let $s_1, ..., s_n$ be a set of n students. Each students must attend some courses $c_1, ..., c_m$. Each course can be planned into exactly one of three time slots - `MORNING`, `AFTERNOON`, `EVENING`. **A student can't attend two courses that are scheduled in the same time slot.**

Write a C++ that takes in input a list of students and the courses they have to attend, a computes a course-scheduling that is compatible with all of them. **The program should also report if such scheduling does not exist.**

Suggerimento: Suppose the student s_k must attend the courses $c_{k,1}, c_{k,2}, ..., c_{k,m}$. This means that no pair of courses $(c_{k,i}, c_{k,j}), i \neq j$ can be in the same time slot. Let $G(s)$ be the complete graph (with all possible edges) that has $c_{k,1}, ..., c_{k,m}$ as nodes. The graph G we obtain as union of all $G(s_i)$ for all students s_i, seen as a single graph, represents "time slot compatibility'' between courses. To compute an admissible scheduling, it is sufficient to k-color G, where k is the number of available time slots, in our case $k = 3$.

Esercizio 5

Let G be a non-oriented graph. A *vertex cover* for G is a subset of nodes $W \subseteq V$ such that each edge in G has at least one endpoint in W, that is it is not possible $(a, b) \in E$ but $a \notin W, b \notin W$.

1. Write a C++ program that computes, using backtracking, a vertex cover for an input graph.
2. Write a C++ program that computes, using backtracking, a vertex cover *of minimum cost* for an input graph. The cost of a cover is the sum of the costs of its nodes, that are also part of the input.