Cubi

Siano dati n cubi. Su ogni faccia del cubo è presente una lettera del'alfabeto. Ogni cubo può essere descritto da una stringa di 6 caratteri (e.g. "ABCDEF"), che rappresenta le 6 facce del cubo.
Sia data una parola costituita da $t \leq n$ caratteri. Descrivere un algoritmo che ritorni **true** se è possibile comporre la parola utilizzando t degli n cubi (scegliendo *una* faccia per ognuno di essi), **false** altrimenti. Discutere correttezza e complessità dell'algoritmo risultante.

Ad esempio, supponete di avere i seguenti cubi: "ABCDEF", "GHIJKL", "AABBCC", "ISTUVW" e di voler comporre la parola IDEA.

Il gioco delle coppie (2-Partition)

Scrivere un algoritmo che, dato un vettore A di n interi distinti (n pari), ritorna **true** se è possibile partizionare A in coppie di elementi che hanno tutte la stessa somma (intesa come la somma degli elementi della coppia), **false** altrimenti. Ad esempio:

$$7, 4, 5, 2, 3, 6$$

può essere partizionato in $7 + 2 = 4 + 5 = 3 + 6$.

Discutere la complessità e la correttezza – per questo esercizio, la dimostrazione di correttezza è importante e va scritta bene.

Confrontatelo con il problema 3-partition discusso a lezione.

Stringhe primitive

Scrivere un algoritmo che, dati un insieme S contenente m stringhe dette *primitive* ed una stringa $X[1 \ldots n]$, conti in quanti modi diversi X è ottenibile dalla concatenazione di stringhe primitive.

Ad esempio, dato $S = \{\texttt{01}, \texttt{10}, \texttt{011}, \texttt{101}\}$:

- $X = \texttt{0111010101}$: ci sono 3 modi
  ($\texttt{011} - \texttt{10} - \texttt{10} - \texttt{101}$, $\texttt{011} - \texttt{10} - \texttt{101} - \texttt{01}$ e $\texttt{011} - \texttt{101} - \texttt{01} - \texttt{01}$)
- $X = \texttt{0110001}$: non c'è modo di ottenere tre 0 consecutivi

Per comodità, supponete che la lunghezza di una stringa s sia $|s|$ e di avere a disposizione una primitiva $\mathsf{check}(X, s, i)$ che ritorna vero se la stringa s è contenuta nella stringa X a partire dalla posizione i. Il costo della chiamata a $\mathsf{check}()$ è $O(|s|)$. Ad esempio, se $X = \texttt{1001}$ e $s = \texttt{00}$, $\mathsf{check}(X, s, 2)$ ritorna vero, per tutti gli altri indici i ritorna falso.

MC Donald

Gestire un McDonald non è semplice.

- La giornata lavorativa è suddivisa in 3 turni da 4 ore, dalle 11 alle 23.
- Durante il turno $t_i \in T$ (dove T l'insieme di 21 turni), è necessario che siano presenti p_i unità di personale
- Avete a disposizione un insieme D di dipendenti; ogni dipendente $d_j \in D$ dichiara un insieme di turni $n_j \subseteq T$ in cui non può lavorare.
- Ad esempio, d_j non può lavorare nei turni $\{t_1, t_7, t_{21}\}$.
- Per contratto aziendale, ogni lavoratore non può lavorare per più di 5 turni.
- Ogni giorno, un dipendendente non può lavorare per più di due turni (qualsiasi, anche non consecutivi).

Progettare un algoritmo che produca uno scheduling che illustri, per ogni turno, il personale associato e discuterne la complessità.

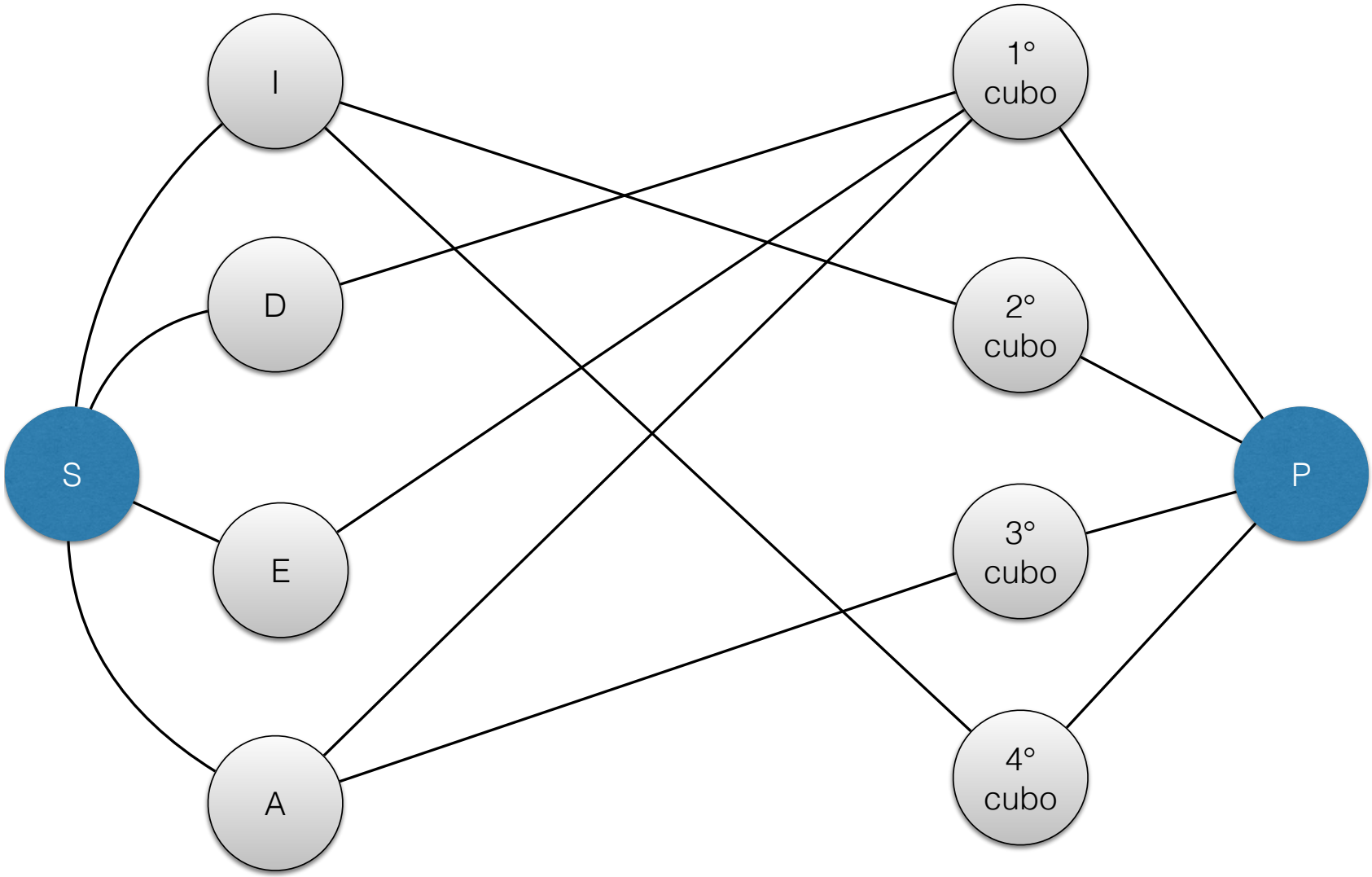
Anagrammi

Un'anagramma è una parola o frase ottenuta riarrangiando le lettere di un'altra parola o frase. Per esempio, "notremors" è un anagramma di "montresor".
Si supponga di avere in input un vettore di n stringhe di lunghezza massima k; si scriva un algoritmo che stampi in output tutti i gruppi di anagrammi contenuti in queste n stringhe. Se ne discuta correttezza e complessità.
Esempio di input: `rosa, pippo, poppi, raso, orsa, giappone`
Esempio di output:

```
rosa, raso, orsa
pippo, poppi
giappone
```

Spoiler alert!

Cubi



Il gioco delle coppie – 2012/05/03 ($O(n \log n)$)

boolean checkPairs(**int**[ ] A, **int** n)

```
sort(A, n)
int pairSum = A[1] + A[n]
for i = 2 to n/2 do
    if A[i] + A[n − i + 1] ≠ pairSum then
        return false
return true
```

Dimostrazione: supponiamo per assurdo che esista un insieme di coppie che rispetti le condizioni per restituire **true**, in cui l'elemento maggiore M sia associato ad un elemento M' diverso dal minore m ($m < M'$). Quindi il minore m è associato ad un elemento m' diverso dal massimo M ($m' < M$). Allora $m + m' < M + M'$, il che contraddice l'ipotesi che tale insieme di coppie rispetti le condizioni per restituire **true**.

Il gioco delle coppie - $O(n)$, hash set

```
boolean checkPairs(int[] A, int n)
int tot = sum(A, n)                    % Sum all elements, O(n)
real pairSum = tot/(n/2)
if pairSum ≠ ⌊pairSum⌋ then
    return false
SET set = Set()                        % Based on a hash table
for i = 1 to n do
    set.insert(A[i])
for i = 1 to n do
    if not set.contains(pairSum − A[i]) then
        return false
return true
```

Stringhe primitive (Compito 09/07/2012)

$$DP[i] = \begin{cases} \sum_{s \in S \wedge \mathsf{check}(X,s,i)} DP[i+|s|] & 1 \leq i \leq n \\ 1 & i > n \end{cases}$$

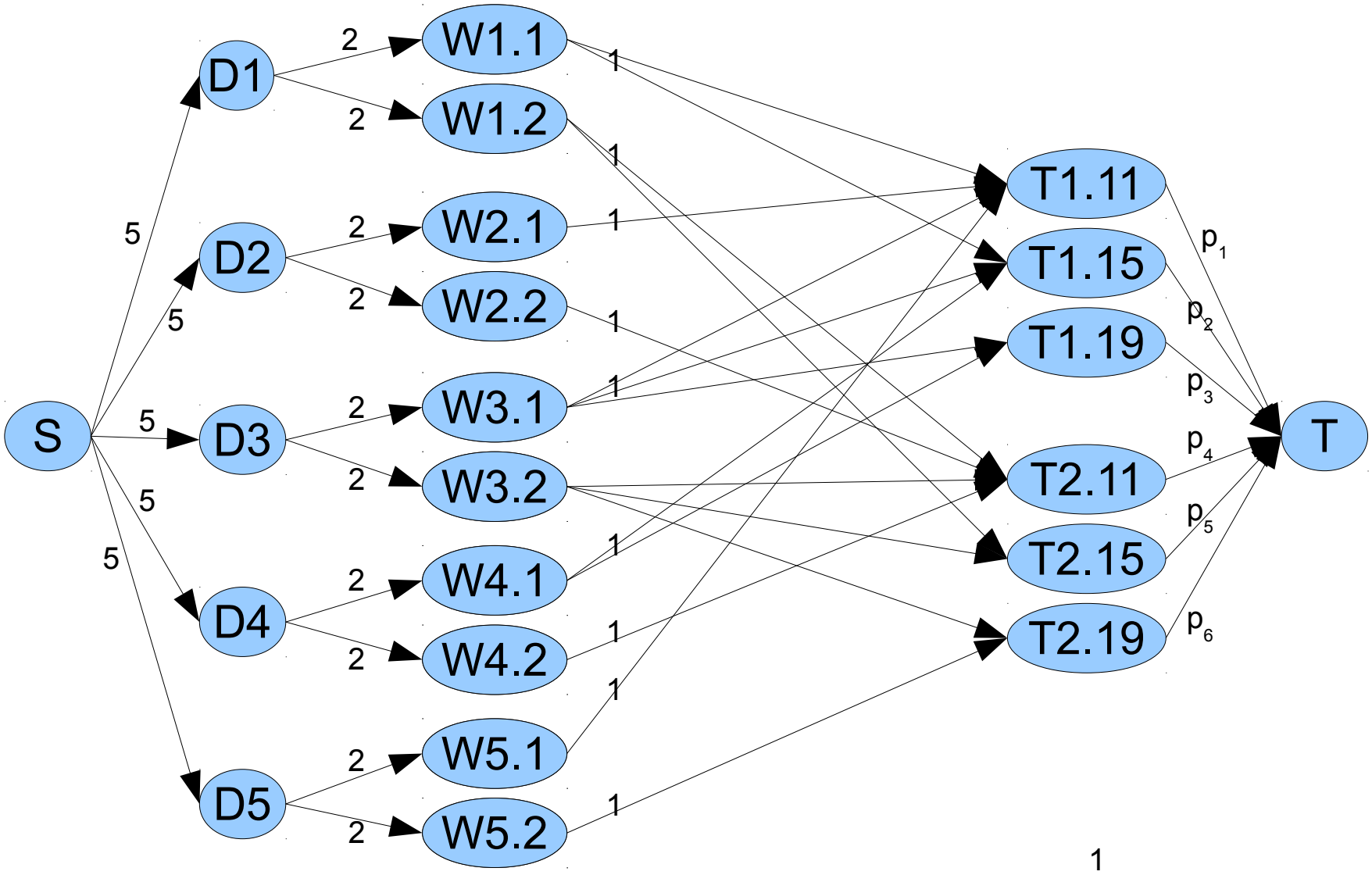
Stringhe primitive

```
int count(int[] X, int n, Set S, int[] DP, int i)
if i > n then
    return 1
else
    if DP[i] < 0 then
        DP[i] = 0
        foreach s ∈ S do
            if check(X, s, i) then
                DP[i] = DP[i] + count(X, n, S, DP, i + |s|)
return DP[i]
```

La chiamata iniziale è $\mathsf{count}(X, n, S, DP, 1)$. Detto $m = \sum_{s \in S} |s|$, la complessità è $O(mn)$.

La soluzione potrebbe essere migliorata con una struttura dati chiamata Trie - si ottiene complessità $O(mn)$, con $m = \max\{|s| : s \in S\}$.

MCDonald


S
D1
D2
D3
D4
D5
W1.1
W1.2
W2.1
W2.2
W3.1
W3.2
W4.1
W4.2
W5.1
W5.2
T1.11
T1.15
T1.19
T2.11
T2.15
T2.19
T
5
5
5
5
5
2
2
2
2
2
2
2
2
2
2
1
1
1
1
1
1
1
1
1
1
p1
p2
p3
p4
p5
p6
1

Anagrammi – 2011/09/07

```
anagrams(Item [][] S, int n)
Hash H = Hash()
for i = 1 to n do
    sorted = sort(S[i])
    Set S = H.lookup(sorted)
    if S == nil then
        S = Set()
    S.insert(S[i])
    H.insert(sorted, S)
foreach k ∈ H do
    Set S = H.lookup(k)
    print S
```

Il costo di questo algoritmo è $O(nk \log k + n)$.