Problema 0.1 – Fibonacci

int fibonacciRic(int n)
- **se** $n \leqslant 1$ **allora**
  - **ritorna** 1
- **allora**
  - **ritorna** fibonacciRic$(n-1)$ + fibonacciRic$(n-2)$

int fibonacciIter(int n)
- $DP \leftarrow$ new **int**$[0 \dots n]$
- $DP[0] \leftarrow DP[1] \leftarrow 1$
- **da** $i \leftarrow 2$ **fino a** n **fai**
  - $DP[i] \leftarrow DP[i-1] + DP[i-2]$
- **ritorna** $DP[n]$

int fibonacci(int n)
- **int** $DP_0 = 1$
- **int** $DP_1 = 1$
- **int** $DP_2 = 1$
- **da** $i = 2$ **fino a** n **fai**
  - $DP_0 = DP_1$
  - $DP_1 = DP_2$
  - $DP_2 = DP_0 + DP_1$
- **ritorna** DP_2

Fibonacci Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. lorem

Problema 0.2 – Codice di Huffman

TREE huffman(int[ ] c, int[ ] f, int n)
- // c$[1 \dots n]$: caratteri dell'alfabeto
- // $f[1 \dots n]$: frequenze dei caratteri
- // n: dimensione dell'alfabeto
- PRIORITYQUEUE $Q \leftarrow$ MinPriorityQueue
- **da** $i \leftarrow 1$ **fino a** n **fai** // $\mathcal{O}(n)$
  - Q.inserisci($f[i]$, Tree($f[i]$, c$[i]$)) // $\mathcal{O}(\log n)$
- **da** $i \leftarrow 1$ **fino a** $n-1$ **fai** // n: radice $\mathcal{O}(n)$
  - // estraggo i 2 caratteri meno frequenti
  - $z_1 \leftarrow Q$.deleteMin
  - $z_2 \leftarrow Q$.deleteMin
  - // Creo un nuovo nodo
  - $z \leftarrow$ Tree($z_1.f + z_2.f$, **nil**)
  - $z.left \leftarrow z_1$
  - $z.right \leftarrow z_2$
  - // Lo inserisco nella coda
  - Q.inserisci($z.f$, z)
- **ritorna** Q.deleteMin

Codice di Huffman Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Problema 0.3 – Resto

restoDP(int[ ] t, int n, int R)
- // t: tagli disponibili,n: numero di monete, R il resto da dare
- $DP \leftarrow$ new **int**$[0 \dots R]$
- $S \leftarrow$ new **int**$[0 \dots R]$
- $DP[0] \leftarrow 0$ // caso base
- // Riempire la tabella DP
- **da** $i \leftarrow 1$ **fino a** R **fai**
  - $DP[i] \leftarrow +\infty$
  - **da** $j \leftarrow 1$ **fino a** n **fai** // Riempio la tabella
    - **se** $i > t[j]$ **and** $DP[i - t[j]] + 1 < DP[i]$ **allora**
      - // aggiorno il valore
      - $DP[i] \leftarrow DP[i - t[j]] + 1$ // registro il valore
      - $S[i] \leftarrow j$ // la moneta da utilizzare per risolvere il problema quando il taglio è i
- **finché** $R > 0$ **fai** // ho resto da dare
  - **stampa** $t[S[R]]$ // stampo la moneta
  - $R \leftarrow R - t[S[R]]$ // decremento il resto

restoGreedy(int[ ] t, int n, int R, int[ ] x)
- { Ordina le monete in modo *decrescente* }
- // $\mathcal{O}(n)$ se già ordinato, $\mathcal{O}(n \log n)$ altrimenti
- **da** $i \leftarrow 1$ **fino a** n **fai** // $\mathcal{O}(n)$
  - // il numero di monete di taglio massimo
  - $x[i] \leftarrow \left\lfloor \frac{R}{t[i]} \right\rfloor$
  - // calcolo il resto rimanente
  - $R \leftarrow R - x[i] \cdot t[i]$
- **ritorna** R

Problema del resto Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Problema 0.4 – Zaino

zaino(float[ ] p, float[ ] v, float C, int n, float[ ] x)
- { ordina p e v in modo che $\frac{p[1]}{w[1]} \geqslant \frac{p[2]}{w[2]} \geqslant \dots \geqslant \frac{p[n]}{w[n]}$ }
- // $\mathcal{O}(n)$ se già ordinato, $\mathcal{O}(n \log n)$ altrimenti
- **da** $i \leftarrow 1$ **fino a** n **fai**
  - $x[i] \leftarrow \min\left(\frac{C}{w[i]}, 1\right)$ // ne prendo solo una frazione?
  - $C \leftarrow C - x[i] \cdot w[i]$ // aggiorno la capacità residua

Problema dello zaino Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Problema 0.5 – Parentesizzazione

int recPar(int[ ] c, int i, int j)
- **se** i $\leftarrow$ j **allora** // se gli indici corrispondono
  - **ritorna** 0 // non devo fare operazioni
- **allora**
  - $min \leftarrow +\infty$
  - // Tutta la logica dell'algoritmo
  - **da int** $k \leftarrow i$ **fino a** $j-1$ **fai**
    - **int** $q \leftarrow$ recPar(c, i, k) + recPar(c, $k+1$, j) + **c**$[i-1] \cdot$ **c**$[k] \cdot$ **c**$[j]$
    - **se** $q < min$ **allora** // se q è più piccolo del minimo
      - $min \leftarrow q$ // aggiorniamo il minimo
  - **ritorna** min

int[ ][ ] multiply(int[ ][ ] A, int[ ][ ] S, int i, int j)
- **se** $i == j$ **allora**
  - **ritorna A**$[i]$
- **allora**
  - **int**[ ][ ] X = multiply($last, i, last[i][j]$)
  - **int**[ ][ ] Y = multiply($last, last[i][j] + 1, j$)
  - **ritorna** multiplyMatrices(X, Y)

Moltiplicazione fra matrici Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

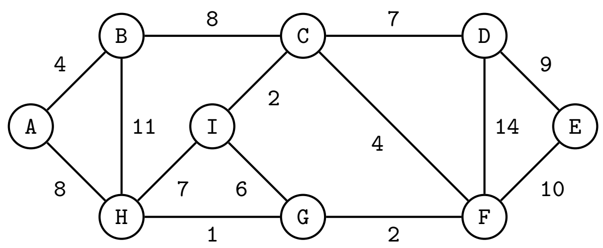
Problema 0.6 – Insieme disgiunto di intervalli pesati

SET maxinterval(**int** a, **int** b, **int** w, **int** n)
- { ordina gli intervalli per estremi di fine crescenti }
- **int**[ ][ ] $pred$ ← computePredecessor(a, b, n)
- **int**[ ][ ] DP ← new **int**$[0 \dots n]$
- `// riempio la tabella`
- $DP[0] \leftarrow 0$
- **da** $i \leftarrow 1$ **fino a** n **fai**
  - $DP[i] \leftarrow \max(DP[i-1],\ w[i] + DP[pred[i]])$
- `// costruisco l'insieme dei predecessori`
- $i \leftarrow n$
- SET S ← Set
- **finché** $i > 0$ **fai** `// fintanto che ci sono intervalli disponibili`
  - **se** $DP[i-1] > w(i) + DP[pred[i]]$ **allora** `// commento`
    - i — `// non considerarlo`
  - **allora**
    - S.insert(i) `// inseriscilo nell'insieme`
    - $i \leftarrow pred[i]$ `// scorri gli interalli`
- **ritorna** S `// ritorna l'insieme di intervalli ordinati`

`// Pre-computa i predecessori`

int[ ] computePredecessor(**int**[ ] a, **int**[ ] a, **int** n)
- **int**[ ][ ] $pred$ ← new **int**$[0 \dots n]$
- $pred[0] \leftarrow 0$
- **da** $i \leftarrow 1$ **fino a** n **fai**
  - $j \leftarrow i - 1$
  - **finché** $j > 0$ **and** $b[j] > a[i]$ **fai**
    - $j \leftarrow j - 1$
  - $pred[i] \leftarrow j$
- **ritorna** $pred$

SET independentSet(**int**[ ] a, **int**[ ] b)
- { ordina a e b in modo che $b[1] \leqslant b[2] \leqslant \cdots \leq b[n]$ }
- `// ` $\mathcal{O}(n)$ `se già ordinati,` $\mathcal{O}(n \log n)$ `altrimenti`
- SET S ← Set
- S.insert (1) `// inserisco il primo intervallo`
- **int** $ultimo \leftarrow 1$ `// ultimo intervallo inserito`
- **da** $i \leftarrow 2$ **fino a** n **fai**
  - **se** $a[i] \geqslant b[ultimo]$ **allora**
    - `// gli intervalli sono indipendenti`
    - S.insert (i) `// lo inserisco`
    - $ultimo \leftarrow i$ `// lo rendo l'ultimo inserito`
- **ritorna** S



Problema 0.7 – Cammini Minimi

SET kruskal(EDGE[] A, **int** n, **int** m)
- `// EDGE[]: vettore di archi`
- (1) SET T ← Set `// insieme (inizialmente vuoto) che conterrà gli archi dell'albero minimo`
- MFSET M ← Mfset(n) `// insieme disgiunto grande`
- `// ordino per peso crescente`
- (2) { ordina $A[1 \dots m]$ in modo che $A[1].peso \leqslant \cdots \leqslant A[m].peso$ }
- **int** $c \leftarrow 0$ `// quanti archi ho aggiunto`
- **int** $i \leftarrow 1$ `// quale arco sto guardando`
- (3) **finché** $c < n - 1$ **and** $i \leqslant m$ **fai** `// Termina quando l'albero è costruito`
  - `//` $c < n-1$`: ho raggiunto tutti gli archi necessari per fare un albero`
  - `//` $i \leqslant m$`: ho esaurito tutti gli archi da guardare (per controllo)`
  - **se** M.find($A[i].u$) $\neq$ M.find($A[i].v$) **allora** `// non fanno parte dello stesso albero`
    - M.merge($A[i].u$, $A[i].v$) `// unisco gli insiemi disgiunti`
    - T.insert($A[i]$) `// inserisco l'arco all'albero`
    - $c \leftarrow c + 1$ `// ho aggiunto un altro arco`
  - $i \leftarrow i + 1$ `// guardo il prossimo arco`
- **ritorna** T `// Ritorna l'albero di copertura minimo`

prim(GRAPH G, NODE r, **int**[ ] p)
- `//` r`: nodo dalla quale parto`
- `//` p`: vettore dei padri`
- PRIORITYQUEUE Q ← MinPriorityQueue
- PRIORITYITEM[] pos ← new PRIORITYITEM[1...G.n]
- `// inserisco i nodi nella coda, memorizzando la loro posizione`
- (4) **per ciascun** $u \in G.V() - \{r\}$ **fai**
  - $pos[u] \leftarrow Q$.inserisci($u, +\infty$)
- `// Inserisco il "nodo di partenza"`
- $pos[r] \leftarrow Q$.inserisci(r, 0)
- $p[r] \leftarrow 0$ `// convenzione per indicare che non ha padre`
- (5) **finché not** Q.isEmpty **fai** `// non ci sono più nodi`
  - NODE $u \leftarrow Q$.deleteMin `// cancello e restituisco il nodo`
  - $pos[u] \leftarrow$ **nil** `// non considero più quel nodo`
  - `// per ciascun nodo adiacente a quello considerato`
  - (6) **per ciascun** $v \in G$.adj(u) **fai**
    - **se** $pos[v] \neq$ **nil and** w$(u,v) < pos[v].priority$ **allora**
      - `//` $pos[v] \neq$ **nil**`: è già stato visitato`
      - `//` w$(u,v) < pos[v].priority$`:`
      - Q.decrease($Pos[v]$, w(u,v)) `// commento`
      - $p[v] \leftarrow u$ `// commento`

int[ ], **int**[ ] CamminiMinimi(GRAPH G, NODE s)
- (7) PRIORITYQUEUE S ← PriorityQueue `//` $\mathcal{O}(n) \cdot 1$
- S.inserisci(s, 0)
- **finché not** S.isEmpty() **fai** `//` $\mathcal{O}(n)$
  - (8) `//` $\mathcal{O}(n)$ `vettore ordinato /` $\mathcal{O}(\log n)$ `heap binario`
  - **int** $u \leftarrow S$.deleteMin
  - $b[u] \leftarrow$ **falso**
  - **per ciascun** $v \in G$.adj(u) **fai**
    - **se** $d[u] + G$.w$(u,v) < d[v]$ **allora**
      - **se not** $b[v]$ **allora**
        - (9) `//` $\mathcal{O}(1) \cdot n$ `vettore ordinato /` $\mathcal{O}(\log n) \cdot n$ `heap binario`
        - $b[v] \leftarrow$ **vero**
      - **altrimenti**
        - (10) `//` $\mathcal{O}(1) \cdot m$ `vettore ordinato /` $\mathcal{O}(\log n) \cdot m$ `heap binario`
      - $T[v] \leftarrow u$
      - $d[v] \leftarrow d[u] + G$.w(u,v)
- **ritorna** (T, d)