Correzione della Seconda Provetta
ASD1 2002-2003

Exercise 1 *Dimostrare le seguenti proprietà relative ai numeri di Fibonacci.*

1. $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$;
2. $F_{k+2} \geq \phi^k$, *dove* $\phi := \frac{1+\sqrt{5}}{2}$ *indica la sezione aurea.*

Svolgimento Esercizio 1

1. $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$;
   base: $(k = 0)$: $F_2 = 1 = 1 + F_0 = 1 + \sum_{i=0}^{0} F_i$.
   passo: $F_{k+2} = F_{k+1} + F_k = 1 + \sum_{i=0}^{k-1} F_i + F_k = 1 + \sum_{i=0}^{k} F_i$.

2. $F_{k+2} \geq \phi^k$, dove $\phi := \frac{1+\sqrt{5}}{2}$ indica la sezione aurea.
   base: $(k = 0)$: $F_2 = 1 = \phi^0$.
   base: $(k = 1)$: $F_3 = 2 > \phi$.
   passo: $F_{k+2} = F_{k+1} + F_k \geq \phi^{k-1} + \phi^{k-2} = \phi^k$. Dove l'ultima equazione disegue dal fatto che ϕ è una radice del polinomio $x^2 - x - 1$.
   Nota: Se uno si fosse limitato a considerare il caso $k = 0$ come base, sarebbe potuto giungere all'erronea conclusione che $F_{k+2} = \phi^k$.

Exercise 2 *Si consideri una foresta di alberi gestita da una famiglia di guardiaboschi. All'inizio la foresta non contiene alcun albero. I guardiaboschi eseguono, in sequenze arbitrarie, operazioni delle seguenti 3 tipologie.*

1. **impianto:** *impiantare un nuovo albero fatto di un solo nodo senza discendenza;*
2. **diversificazione:** *individuare due alberi T_1 e T_2 della stessa specie (ossia con radici aventi lo stesso numero di figli), sradicare T_1 ed innestarne la radice r_1 come figlio diretto della radice di T_2;*
3. **potatura:** *la potatura è un'operazione ricorsiva. Dato un nodo v in un albero, indichiamo con $\pi(v)$ il papà di v, ossia il nodo in cui v risulta innestato come figlio.* Potar via *un nodo v significa staccare da $\pi(v)$ il sottoalbero radicato in v ed innestarlo direttamente nel terreno, ed inoltre, se $\pi(v)$ si era già visto potar via un figlio antecedentemente a v, da quando si trova innestato come figlio del suo papà $\pi(\pi(v))$, allora la potatura si applica ricorsivamente a $\pi(v)$.*

Con riferimento al precedente esercizio, sapresti dimostrare che in un qualsiasi momento, ed ove v sia un qualsiasi nodo di albero della nostra foresta con, diciamo, k figli, allora la cardinalità della discendenza di v, ossia il numero di nodi del sottoalbero radicato in v, è almeno ϕ^k.

Svolgimento Esercizio 2

Si fissi l'attenzione su un qualsiasi istante della felice storia della nostra famigliola di guardiaboschi. Si indichi con $d(v)$ la cardinalità della discendenza di v. Per induzione su k, mostreremo che, $d(v) \geq F_{k+2}$ per un qualsiasi nodo v con k figli. A quel punto il risultato desiderato seguirà come conseguenza della proprietà di cui al Punto 2 dell'Esercizio 1.

base: Chiaramente, se v ha 0 figli, allora la sua discendenza si compone solamente di v stesso, ossia $d(v) = 1 = F_2$. Inoltre, se v ha un figlio, allora $d(v) \geq 2 = F_3$.

passo: Siano $v_1, v_2, \ldots, v_k$ i figli di v nell'ordine in cui sono stati innestati in v. Chiaramente, $d(v_1), d(v_2) \geq 1$. Mostreremo innanzittutto che $d(v_i) \geq F_i$ per ogni $i > 2$. Infatti, quando v_i è stato innestato in v, il nodo v aveva almeno $i - 1$ figli: $v_1, v_2, \ldots, v_{i-1}$. Siccome una radice può venire innestata come figlia di un'altra radice solamente quando esse hanno lo stesso numero di figli, allora v_i aveva almeno $i - 1$ figli quando v_i è stato innestato in v. Da quel momento v_i può aver perso al più

un solo figlio, per come è stata definita l'operazione di potatura e visto che v_i non è stato potato via da v posteriormente all'innesto discusso. Ne consegue che v_i ha almeno $i-2$ figli nell'istante su cui abbiamo deciso di fissare la nostra attenzione, e quindi, per ipotesi induttiva, $d(v_i) \geq F_i$ come sopra affermato.

In virtù della proprietà di cui al Punto 1 dell'Esercizio 1 possiamo ora concludere il nostro passo induttivo come segue:

$$d(v) = 1 + \sum_{i=1}^{k} d(v_1) \geq 3 + \sum_{i=3}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2}.$$

Exercise 3 *Sapresti dare uno pseudocodice od una descrizione comunque efficace e formale dell'operazione di* EstraiMinimo *(e* Consolida*) relativa ad uno* Heap di Fibonacci*? Sapresti indicare il costo reale nel caso peggiore e sapresti tracciare l'analisi ammortizzata?*

Svolgimento Esercizio 3
Siccome in uno heap di Fibonacci teniamo un puntatore $min[H]$ ad una radice che ospita un elemento a chiave minima tra le radici (e quindi tra tutti gli elementi ospitati nello heap, a causa della proprietà dello heap), non sarà certo difficile restituire un elemento a chiave minima in $O(1)$.

Tuttavia vorremmo non solo individuare ma anche estrarre dallo heap l'elemento indicatoci da $min[H]$. L'estrazione di per se non è difficoltosa in quanto la lista delle radici è una lista circolare e doubled linked e quindi ho a disposizione tutte le informazioni necessarie per aprire il "braccialetto" nel punto indicatomi da $min[H]$, estrarre la perla indirizzata da $min[H]$, e ricomporre le estremità del "braccialetto", il tutto in $O(1)$.

Dove si annida il problema? Il punto è che non sò più a quale radice dovrà ora puntare $min[H]$, e di fatto per scoprirlo non posso evitare di scandirmi l'intera lista delle radici, di lunghezza $t(H)$, impiegando quindi un tempo effettivo di calcolo $O(t(H))$.

Visto allora che questo $O(t(H))$ lo debbo comunque spendere, ne approfitto per giocare una *Consolidate*, ossia man mano che, durante la mia scansione della lista redici, individuo due radici dello stesso grado, effettuo allora un innesto di una radice nell'altra come i nostri amici guardiaboschi di cui all'esercizio precedente. Ovviamente sceglierò quale radice và innestata nell'altra con l'avvertenza di mantenere la proprità dell'ordinamento dello heap, cosa che mi è sempre possibile fare in $O(1)$. Il costo effettivo della *Consolidate* non sforerà pertanto sul costo asintotico di $O(t(H))$ richiesto per la sola scansione dell'intera lista delle radici, purchè io abbia l'accortezza di avvalermi di un vettore di appoggio che mi ricordi, per ogni k, se ho lasciato dietro di me nella scansione una radice con k figli, ed eventualmente il puntatore alla stessa.

In base a quanto dimostrato nell'Esercizio 2, siamo garantiti che comunque $k \leq \log_\phi n$, e siccome a seguito della *Consolidate* nel mio giardino non saranno presenti alberi della stessa varietà (con radici aventi lo stesso numero di figli), allora il numero di radici $t(H')$ a seguito della *Consolidate* sarà limitato superiormente da $\log_\phi n$ (ossia $t(H') \leq \log_\phi n$). Durante la consolidate sarà inoltre mia cura di smarcare quelle radici che vanno ad innestarsi in altre radici, in modo da mantenere l'invariante che un nodo sia marcato se e solo se ha subito la perdita di un figlio per potatura da quando è andato a scegliersi il padre corrente tramite suo ultimo innesto. Pertanto, ove si sia indicato con $m(H)$ ed $m(H')$ il numero di nodi marcati prima e dopo l'esecuzione della *Consolidate*, allora $m(H') \leq m(H)$. Ricordiamo che la funzione potenziale scelta per condurre l'analisi ammortizzata per gli heaps di Fibonacci ha la seguente forma: $\Phi(H) = t(H) + 2m(H)$. Il costo ammortizzato per l'operazione di *EstraiMinimo* sarà pertanto limitato da costo effettivo meno differenza di potenziale, ossia da

$$t(H) - (t(H) + 2m(H)) + (t(H') + 2m(H')) = t(H') + 2m(H') - 2m(H) \leq t(H') \leq \log_\phi n$$

L'unità di misura scelta per l'espressione della funzione potenziale è infatti assunta essere tale da dominare ogni altro fattore costante che appare nel costo effettivo di qualsivoglia operazione prevista per gli heaps di Fibonacci. Solitamente, su una qualsiasi struttura dati, viene definito (ed implementato) solamente un numero finito di operazioni, pertanto questa assunzione è sempre legittima e fa parte della prassi nel progetto di una struttura dati nell'ottica dell'analisi ammortizzata.

Exercise 4 *Agli archi di un grafo $G = (V, E)$ sono associate delle lunghezze (valori numerici non negativi). Abbiamo cioè una funzione $l : E \mapsto R_+$. Inoltre viene indicato un nodo sorgente s. Vorremo valutare la distanza $d(s, v)$ per ogni nodo in V, ossia la minima lunghezza di un cammino da s a v. Poggiando su uno heap di servizio (preso come black-box), si dia uno pseudocodice per l'algoritmo di Dijkstra che computa le distanze $d(s, v)$ da un dato nodo sorgente s a tutti gli altri nodi, ma anche i cammini ottimi (si utilizzi un array di predecessori $\pi : V \mapsto V \cup \{-1\}$), qualora le lunghezze degli archi siano tutte non negative. Nello pseudocodice, i nodi siano colorati bianco, grigio e nero, secondo il seguente schema:*

bianco *se non si è ancora individuato un cammino da s al nodo in questione;*

grigio *se si è già individuato qualche cammino da s al nodo in questione v, ma v non è stato ancora estratto dallo heap di servizio;*

nero *il nodo in questione è già stato estratto dallo heap di servizio.*

Dimostrare poi le seguenti invarianti:

1. *ad ogni iterazione, ad uno specifico passo della tua procedura da te indicato, puoi affermare che i nodi grigi sono tutti e soli i nodi nello heap di servizio;*
2. *sempre, $\lambda(v) \geq d(v)$;*
3. *quando v è nero, allora $\lambda(v) = d(v)$.*

Svolgimento Esercizio 3

Faremo riferimento (come black-box) ad uno heap H ed assumeremo implicitamente che la chiave dell'elemento v in seno ad H sia il valore di λ_v.

1. per ogni nodo v: $\pi_v := -1$, $\lambda_v := \infty$, $c_v :=$WHITE;
2. $\pi_s := s$, $\lambda_s := 0$, $c_s :=$GRAY, heap $H := \{s\}$;
3. finchè $H \neq \emptyset$ esegui:
   - 3.1 $v :=$ *EstraiMinimo*(H); $c_v :=$BLACK;
   - 3.2 per ogni arco vu esegui:
     - 3.2.1 se $c_u =$WHITE: $\pi_u := v$, $\lambda_u := \lambda_v + w(vu)$, $c_u :=$GRAY, inserisci u in H;
     - 3.2.2 se $c_u =$GRAY e $\lambda_u > \lambda_v + w(vu)$: $\pi_u := v$, $\lambda_u := \lambda_v + w(vu)$, (ossia decrementa la chiave di u in H);

Dimostriamo ora le invarianti proposte:

1. Si assuma la simultaneità delle operazioni espresse su una stessa linea (separate da una semplice virgola). Allora possiamo dimostrare qualcosa di più forte di quanto non richiesto dall'esercizio: ad ogni istante, i nodi grigi sono tutti e soli i nodi nello heap di servizio.
   Infatti, da principio i nodi sono tutti settati come bianchi; la pila viene allocata solamente alla Linea 2, inserendovi il nodo s che in contemporanea riceve una spennellata di grigio. Inoltre, ogni qual volta un nodo viene estratto dalla pila viene colorato di nero. Infine, ogni qual volta un nodo viene inserito nella pila viene colorato di grigio.

2. sempre, $\lambda(v) \geq d(v)$;
   infatti, se $\lambda(v) < \infty$, allora seguendo il vettore dei predecessori (come da seguente esercizio) ricostruisco un cammino da s a v di peso λ_v. (È facile argomentare tale conclusiva affermazione procedendo per induzione sull'istante t in cui λ_v è settato.

3. quando v è nero, allora $\lambda(v) = d(v)$. Per induzione sull'istante t in cui v diviene nero. Ossia assumiamo che le distanze λ computate siano ottime per tutti i nodi estratti da H e colorati di nero antecedentemente a v. Si assuma, per assurdo, l'esistenza di un cammino P da s a v tale che $w(P) < \lambda_v$. Percorrendo P da s verso v, sia x l'ultimo nodo nero e y il primo nodo grigio. Siccome x è nero, ossia è stato estratto da H, allora $\lambda_y \leq \lambda_x + w(xy) = d(x) + w(xy) \leq w(P) < \lambda_v$. Ma allora, a be vedere, non sarebbe forse y il nodo da estrarre ora dallo heap, invece che v?

Exercise 5 *Con riferimento al precedente esercizio, sapresti fornire lo pseudocodice di una procedura che, dato un nodo v, e sulla base del vettore dei predecessori π come computato dall'algoritmo di Dijkstra, restituisca un cammino ottimo da s a v in tempo lineare nella lunghezza del cammino stesso? Sapresti condurre l'analisi del running time nel caso peggiore evidenziando inoltre le differenze qualora lo heap sia sato realizzato come Heap Binomiale o come Heap di Fibonacci?*

Svolgimento Esercizio 4

StampaCammino (v)

1. se $\pi_v = -1$ allora non si può raggiungere v da s e quindi esci e termini.

2. se $v = s$ allora stampi s ed esci.

3. chiami *StampaCammino* $(\pi(v))$;

4. stampi v;

Per l'analisi della coplessità dell'algoritmo di Dijkstra si noti che l'operazione di estrazione di un nodo a λ minimo è eseguita al più n volte (n se il grafo è connesso) e quindi possiamo considerare che il tempo speso in tale operazione sia in tutto $O(n \log n)$, poichè l'estrazione del minimo prende $O(\log n)$ sia con gli heaps Binomiali che con quelli di Fibonacci (ammortizzata). Parimenti, ogni nodo viene inserito al più una volta e quindi il tempo complessivo imputabile ad inserimenti è $O(n \log n)$. Si noti che ogni arco può causare al più un solo decremento di chiave, e precisamente nell'istante in cui il primo dei suoi estremi diventa nero. Pertanto, il tempo speso decrementando chiavi è dominato complessivamente da $O(nDec(n))$, dove $Dec(n) = O(\log n)$ nel caso degli heaps Binomiali e $O(1)$ nel caso degli heaps di Fibonacci, ove si faccia riferimento all'analisi ammortizzata. Il tempo complessivo di calcolo sarà pertanto limitato da $O(m \log n + n \log n) = O(m \log n)$ nel caso di implementazione tramite heaps Binomiali e da $O(m + n \log n)$ nel caso di implementazione tramite heaps di Fibonacci.

Exercise 6 *Sapresti dare uno pseudocodice od una descrizione comunque efficace e formale dell'algoritmo* RadixSort *che usi come algoritmo intermedio il* CountingSort *preso come black-box? Sapresti indicare il tempo di calcolo nel caso peggiore e migliore? Riesci ad individuare una invariante di ciclo? Se al posto del* CountingSort *utilizzi un altro algoritmo di ordinamento, l'output resta comunque corretto? (Fornire eventuale controesempio).*

Svolgimento Esercizio 5
Sia A un vettore di n interi rappresentati ciascuno da D cifre in base B.

RadiSort (A, D)

1. per ogni cifra d, con $1 \leq d \leq D$, e partendo dalla cifra meno significativa, esegui:

   1.2 utilizza un algoritmo di ordinamento stabile per ordinare il vettore A sulla cifra d.

Se l'algoritmo di ordinamento stabile chiamato ogni volta è il CountingSort, allora esso avrà complessità $\Theta(n + B)$. Pertanto la complessità di RadixSort sarà $\Theta(Dn + DB)$.

L'algoritmo CountingSort è stabile, ossia in caso di pareggio sulla chiave, viene piazzato prima in output l'elemento che risultava piazzato prima in input. Il RadixSort come descritto sopra risulta corretto se l'algoritmo cui fa riferimento è stabile, in quanto possiamo verificare la seguente invariante:

> se dei numeri presenti in A consideriamo solo le cifre meno significative, dalla cifra d appena considerata in poi, il vettore A è ordinato.

Controesempio richiesto: si consideri $A = [11, 12]$. Dopo averlo ordinato sulla cifra meno significativa otterremo necessariamente $A_1 = [11, 12]$. Ora però dobbiamo ordinarlo anche sulla cifra più signifivativa. Se nessuno ci garantisce che l'algoritmo impiegato per ordinare sulla cifra più significativa sia stabile, allora non possiamo escludere che l'output sia $A_2 = [12, 11]$. Ossia un output errato.