

Subset SumInput:  $S = \{s_i\}_{i=1}^n$ ,  $k$ ,  $B$ Output: σύνολο  $A \subseteq S$  :  $\sum s_i = B$  και  $|A| \leq k$  $\leadsto dp[i][b][m] =$  σύνολο  $S' \subseteq \{s_1, \dots, s_i\}$  με  $\sum s_i = b$  και  $|S'| = m$ 1.  $dp[i][0][0] = 1$ ,  $dp[i][0][m] = 0, m > 0$ ,  $dp[i][b][0] = 0, b > 0$ 2.  $dp[i][b][m] = \max \{ dp[i-1][b][m] \}$ 3.  $\sum_{m=1}^k dp[i][B][m]$  if  $(b \geq s_i \text{ and } m > 0)$   $dp[i-1][b-s_i][m-1]$  $O(nBk)$   
ψευδοκώδικαςSubsequence SumInput:  $a = \{a_1, \dots, a_n\}$ ,  $k$ Output: μέγιστο άθροισμα  $k$ -almost incr subsequence $dp[i][m]$ : μέγιστο άθροισμα  $\{a_1, \dots, a_i\}$  με  $m$  παραβ να περιλαμβάνει  $a_i$ 1.  $dp[0][m] = 0$ 2.  $dp[i][m] = \max_{0 \leq j < i} \begin{cases} \text{if } (a_i > a_j) & a_i + dp[j][m] \\ \text{else if } (m > 0) & a_i + dp[j][m-1] \end{cases}$ 3.  $\max_{i, m \in (n) \times (k)} dp[i][m] \rightarrow O(n^2k)$ Max Length Interval SchedulingInput:  $\{(s_i, f_i)\}_{i=1}^n$ 

Output: μέγιστο μήκος από μη επικαλ διαστήματα

 $\leadsto dp[i] =$  άθροισμα για  $\{(s_1, f_1), \dots, (s_i, f_i)\}$ 1. sort:  $\{(s_i, f_i)\}_{i=1}^n$   $f_1 \leq \dots \leq f_n \rightarrow O(n \log n)$ unoblog prev(1..n):  $prev(i) =$  μέγιστο  $j$  από μη επικαλ διάστημα  $\rightarrow O(n \log n)$  $dp[0] = 0$ 2.  $dp[i] = \max \{ dp[i-1] \}$ 3.  $dp[n] = f_i - s_i + dp[prev(i)]$  $O(n)$ 

binary search

Parting ProblemInput:  $\{G_i = (v_i, d_i, s_i)\}_{i=1}^n$ ,  $L$ Output: μέγιστο value με παρκενγκ  $G_i$  σε θέση  $\leq d_i$  και μέγεθος  $s_i$  $dp[i][j] =$  άθροισμα για  $\{G_1, \dots, G_i\}$  και παρκενγκ μεγέθους  $j$ 1. sort  $\{G_i = (v_i, d_i, s_i)\}_{i=1}^n$   $i < j \Rightarrow (v_i > v_j) \parallel (v_i = v_j \text{ AND } d_i < d_j) \rightarrow O(n \log n)$  $dp[i][0] = 0$ ,  $dp[0][j] = 0$ 2.  $dp[i][j] = \max \{ dp[i-1][j] \}$ 3.  $dp[n][L] = \max \{ dp[i-1][j] \mid \text{if } (j \geq s_i \text{ AND } j \leq d_i) v_i + dp[i-1][j-s_i] \}$  $O(nL)$

### Independent Set

→  $G(V, E), k$

← NAI, αν  $\exists V' \subseteq V$  να  $|V'| \geq k$  να δν ειναι  
ανά δύο μεταξύ τους

### Set Cover (NAE)

→ σφηναν  $U$ , υποσυν  $S = \{S_1, \dots, S_m\}, k$

← NAI, αν  $\exists$  το πολύ  $k$  υποσύνολα με  $\bigcup S_i = U$

### Clique $\leq$ Independent Set

Clique input: γραφος  $G(V, E), k$

M/Z: δημιουργούμε σφηναν  $\bar{G}$

Independent Set Input:  $\bar{G}, k$

$\left\{ \begin{array}{l} G \text{ έχει κλικά μεγέθους } k \\ \bar{G} \text{ έχει independent set μεγέθους } k \end{array} \right.$

### Vertex Cover $\leq$ Set Cover

Vertex Cover input:  $H, m$

M/Z:  $U = \{e_1, \dots, e_m\}$ , κάθε κορυφή  $v$ , έχει δικό  
της υποσύνολο με ακμές,  $k = m$

Set Cover input:  $U, S = \{S_1, \dots, S_n\}, k$

### Subset Sum

→  $S = \{s_1, \dots, s_n\}$ , target sum  $t$

← NAI, αν  $\exists S' \subseteq S: \sum s_i = t$

### Partition $\leq$ Subset Sum

Partition input:  $A$

M/Z:  $S = A, t = \sum a_i / 2$

Subset Sum input:  $A, t$

$\left\{ \begin{array}{l} \text{δίνεται partition στο } A \\ \exists \text{ subset sum στο } S=A \text{ με } t = \sum a_i / 2 \end{array} \right.$

### Clique

→  $G(V, E), k$

← NAI, αν  $\exists$  σύνολο κορυφ  $\geq k$ , τέτοιον ανα 2

### Vertex Cover

→  $H(V, E), m$

← NAI, αν  $\exists V' \subseteq V, |V'| \leq m$ , ώστε  $\forall e \in E: x, y \in V' \Rightarrow x \in V' \wedge y \in V'$

### Independent Set $\leq$ Vertex Cover

IS input: γραφος  $G(V, E), k$

M/Z:  $m = n - k$  OLS

Independent Set Input:  $G, m$

$\left\{ \begin{array}{l} G \text{ έχει independent set μεγέθους } k \\ G \text{ έχει vertex cover μεγέθους } m = n - k \end{array} \right.$

### Set Cover $\leq$ Vertex Cover

Set Cover input:  $U, S = \{S_1, \dots, S_m\}, k$

M/Z: - κορυφή  $\forall$  ακμης  $U$   $\left\{ \begin{array}{l} \text{αν } u_j \in S_i \rightarrow \\ \text{- κορυφή } \forall S_i \end{array} \right.$  ακμης  $(u_j, S_i)$

Vertex Cover input:  $G, m (=k)$

### Partition

→ σύνολο  $A$

← NAI, αν  $\exists$  σύνολα  $B, C: B \cap C = \emptyset, B \cup C = A: \sum b_i = \sum c_i$

### Subset Sum $\leq$ Partition

Subset Sum input:  $S, t$

M/Z:  $A = S \cup \{2t - \sum s_i\}$

Partition input:  $A$

$\left\{ \begin{array}{l} \exists \text{ subset sum για } S, t \\ \text{δίνεται partition στο } A = S \cup \{2t - \sum s_i\} \end{array} \right.$

### SAT

→ CNF  $\varphi = G_1 \wedge \dots \wedge G_m$ ,  $G_i = (l_{i1} \vee \dots \vee l_{in_i})$ ,  $l_{ij} \in \{x, \neg x\}$   
 ← NAI, αν  $\exists$  ανάθεση:  $\varphi \rightarrow \text{true}$

### SAT $\leq$ 3SAT

SAT input:  $\varphi = G_1 \wedge \dots \wedge G_m$

M1:  $(x_1) \rightarrow (x_1 \vee a \vee a)$

$(x_1 \vee x_2) \rightarrow (x_1 \vee x_2 \vee a) \wedge (x_1 \vee x_2 \vee \neg a)$

$(x_1 \vee x_2 \vee x_3) \rightarrow \text{idio}$

$(x_1 \vee x_2 \vee x_3 \vee x_4) \rightarrow (x_1 \vee x_2 \vee a) \wedge (x_3 \vee x_4 \vee \neg a)$

3SAT input:  $n \varphi' \rightarrow$

### Graph Isomorphism

→  $G_1, G_2$

→  $\forall \epsilon$ , αν  $\exists$  subgraph  $G_1 \cong G_2$

### Clique $\leq$ Graph isomorphism

clique input:  $G, k$

M1:  $\delta\eta\mu K_k$ ,  $\gamma\alpha\alpha\omega\sigma K$  κορυφές + όλες ακμές

Graph Isom input:  $G_1 = K_k$ ,  $G_2 = G$

### B-R Vertex Cover

→  $G(V, E)$  blue/red vert.,  $k$

← NAI,  $\exists$  Vertex Cover  $|V'| \leq k$  και blr κορυφ

### Vertex Cover $\leq$ BR Vert Cover

VC input:  $G(V, E), k$

M1: όλες κορ μπλε + 1 r αλφ με όλες

BR VC:  $G', k' = k+1$

### Independent Set

→  $G(V, E), k$

← NAI, αν  $\exists |V'| \geq k$  που δεν εμψονται ανά δύο μεταξύ τους

### 3SAT $\leq$ Independent Set

3SAT input: CNF  $\varphi = G_1 \wedge \dots \wedge G_m$ ,  $G_i = (l_{i1} \vee l_{i2} \vee l_{i3})$

M1:  $\gamma\alpha\alpha\omega\sigma: \forall l_{i1}, l_{i2}, l_{i3} \rightarrow \text{τριγωνο (ακμές ανά 2)}$   
 εμψονουμε κάθε  $x_i$  με κάθε  $\neg x_i$

Independent Set input:  $G, m$

### Clique

→  $G(V, E), k$

← NAI, αν  $\exists$  συνολο

κορυφ  $\geq k$ , όλων ανά 2

### Blue-Red Clique

→  $G(V, E)$ , blue/red vert.,  $k$

← NAI, αν  $\exists$  clique με  $k$

με κορυφ και μπλε κορ

### Clique $\leq$ blue red clique

clique input:  $G(V, E), k$

M1: όλα  $V \rightarrow \text{μπλε} \oplus 1 \text{ κορυφή κοκκιν}$   
 που συνδέεται με όλα

BR Clique input:  $G', k' = k+1$

### Vertex Cover

→  $H(V, E), m$

← NAI, αν  $\exists |V'| \leq m$ , ώστε  $\forall e \in E: y \in V' \wedge x \in V'$

## Θέμα 2: Άθροισμα στοιχείων υποσυνόλων και υποκαταδιωγίων

παράλλαξη subset sum

(α) Θεωρούμε σύνολο  $S = \{s_1, s_2, \dots, s_n\}$  με  $n \geq 3$  θετικούς ακέραιους. Για δεδομένους ακέραιους  $B \geq 1$  και  $k \geq 1$ , θέλουμε να υπολογίσουμε το πλήθος των υποσυνόλων  $A \subseteq S$  με  $|A| \leq k$  στοιχεία και άθροισμα στοιχείων ίσο με  $B$ , δηλαδή  $\sum_{s_i \in A} s_i = B$ .

1. Να διατυπώσετε αποδοτικό αλγόριθμο για αυτό το πρόβλημα, και να αιτιολογήσετε την ορθότητα και την υπολογιστική του πολυπλοκότητα. Είναι ο αλγόριθμος που προτείνετε πολυωνυμικού χρόνου (εξηγήστε);

1. Input:  $S = \{s_1, \dots, s_n\}$ ,  $B, k$

Αναδρομική σχέση:

Output:  $|S' \subseteq S: \sum_{s_i \in S'} s_i = B, |S'| \leq k|$

$OPT(i, b, m) = |S' \subseteq \{s_1, \dots, s_i\}: \sum s_i = b, |S'| = m|$

Απικονομισμό:

$OPT(i, 0, 0) = 1$

$OPT(i, 0, m) = 0, m > 0$

$OPT(i, b, 0) = 0, b > 0$

Binary choice:

Case 1:  $exclude\_s_i = OPT(i-1, b, m)$

Case 2: if  $(b > s_i, m > 0)$   $include\_s_i = OPT(i-1, b-s_i, m-1)$

$OPT(i, b, m) = exclude\_s_i + include\_s_i$

Result:  $\sum_{m=1}^k OPT(n, B, m)$

→ ψευδοπολυωνυμικό

Complexity:  $O(n \cdot B \cdot k)$

2. Ανεξάρτητα από την υπολογιστική πολυπλοκότητα του αλγορίθμου που προτείνετε, πιστεύετε ότι υπάρχει αλγόριθμος πολυωνυμικού χρόνου για αυτό το πρόβλημα (εξηγήστε);

Για να αποδείξουμε ότι ανήκει σε πολυωνυμικό χρόνο  
πρέπει να το ανάγουμε σε γνωστό P πρόβλημα

(β) Θεωρούμε ακολουθία  $\alpha = (a_1, \dots, a_n)$  αποτελούμενη από  $n \geq 3$  θετικούς ακέραιους αριθμούς.

1. Θέλουμε να υπολογίσουμε μια γνησίως αύξουσα υποακολουθία της  $\alpha$ , τα στοιχεία της οποίας επιτυγχάνουν μέγιστο συνολικό άθροισμα (μεταξύ όλων των γνησίως αύξουσων υποακολουθιών της  $\alpha$ ). Να διατυπώσετε αποδοτικό αλγόριθμο για αυτό το πρόβλημα. Να αιτιολογήσετε την ορθότητα και την πολυπλοκότητα του αλγορίθμου σας.

παράλλαξη longest increasing subsequence

Input:  $\alpha = \{a_1, \dots, a_n\}$

Output:  $\max \{ \sum_{i \in S} a_i, S \subseteq \{1, \dots, n\} : \forall i < j \in S \Rightarrow a_i < a_j \}$

Αναδρομική σχέση:  $OPT(i)$  μέγιστο άθροισμα ↑ υποακολουθίας που τελειώνει στο  $a_i$

Αρχικοποίηση:  $OPT(0) = 0$

Αναδρομή:

$cur\_max = 0$

Result:  $\max \{ OPT(i), i \in \{1, \dots, n\} \}$

for ( $j = 0$  to  $i-1$ )

Complexity:  $O(n^2)$

if ( $a_j < a_i$ )  $cur\_max = \max \{ cur\_max, OPT(j) \}$

$OPT(i) = a_i + cur\_max$

2. Για δεδομένο ακέραιο  $k, 1 \leq k \leq n - 1$ , θέλουμε να υπολογίσουμε μια  $k$ -σχεδόν γνησίως αύξουσα υποακολουθία της  $\alpha$ , τα στοιχεία της οποίας επιτυγχάνουν μέγιστο συνολικό άθροισμα (μεταξύ όλων των  $k$ -σχεδόν γνησίως αύξουσων υποακολουθιών της  $\alpha$ ). Μια (υπακ)ακολουθία  $(\beta_1, \beta_2, \dots, \beta_n)$  είναι  $k$ -σχεδόν γνησίως αύξουσα αν η ιδιότητα ότι η ακολουθία "διακόπτεται" σε το πολύ  $k$  σημεία, δηλαδή αν υπάρχουν  $k$  δείκτες  $\{i_1, \dots, i_k\} \subseteq \{1, \dots, n-1\}$  ώστε να ισχύει ότι για κάθε  $i \in \{1, \dots, n-1\} \setminus \{i_1, \dots, i_k\}, \beta_i < \beta_{i+1}$ . Να διατυπώσετε αποδοτικό αλγόριθμο για αυτό το πρόβλημα. Να αιτιολογήσετε την ορθότητα και την υπολογιστική πολυπλοκότητα του αλγορίθμου σας.

Input:  $\alpha = \{a_1, \dots, a_n\}, k$

Output:  $\max \{ \sum_{i \in S} a_i, S \subseteq \{1, \dots, n\} : |\{i < j : a_i > a_j\}| \leq k \}$

Αναδρομική σχέση:  $OPT(i, m)$  μέγιστο άθροισμα ↑ υποακολουθίας που τελειώνει στο  $a_i$  με  $m$  παραβιάσεις μονοτονίας

Αρχικοποίηση:  $OPT(0, j) = 0$

Αναδρομή:

$cur\_max = 0$

Result:  $\max_{(i, m) \in (0, n) \times (0, k)} \{ OPT(i, m) \}$

for ( $j = 0$  to  $i-1$ )

Complexity:  $O(n^2 k)$

if ( $a_j < a_i$ )  $cur\_max = \max \{ cur\_max, OPT(j, m) \}$

else if ( $m > 0$ )  $cur\_max = \max \{ cur\_max, OPT(j, m-1) \}$

$OPT(i) = a_i + cur\_max$

### Θέμα 3: Συντομότερα Μονοπάτια

Θεωρούμε ισχυρά συνεκτικό κατευθυνόμενο γράφημα  $G(V, E, w, c)$  με  $n$  κορυφές και  $m$  ακμές, στο οποίο κάθε ακμή  $e \in E$  έχει μήκος  $w(e) \geq 0$ . Στο γράφημα  $G$ , μπορούμε να χρησιμοποιήσουμε οποιαδήποτε κατευθυνόμενη ακμή  $e = (u, v)$  στην αντίθετη φορά, δηλαδή να χρησιμοποιήσουμε την ακμή  $e$  στη φορά  $(v, u)$ , αντί της προκαθορισμένης φοράς  $(u, v)$ , αλλά αυτό επιφέρει αύξηση του μήκους της ακμής  $e$  κατά  $c(e) \geq 0$ , από  $w(e)$  σε  $w(e) + c(e)$ . Στα ερωτήματα (1) και (2) παρακάτω, να αιτιολογήσετε την ορθότητα και την υπολογιστική πολυπλοκότητα των αλγορίθμων σας.

$$\begin{array}{c} u \xrightarrow{w(e)} v \\ u \xleftarrow{w(e)+c(e)} v \end{array}$$

1. Να διατυπώσετε αποδοτικό αλγόριθμο που υπολογίζει το συντομότερο μονοπάτι μεταξύ κορυφών  $s$  και  $t$  του  $G$ , αν επιτρέπεται να χρησιμοποιήσουμε οποιεσδήποτε ακμές στην αντίθετη φορά (κάθε φορά που χρησιμοποιούμε μια ακμή στην αντίθετη φορά, θα πρέπει να λάβουμε υπόψη το αυξημένο μήκος της ακμής αυτής στο συνολικό μήκος του μονοπατιού).

1. Αρχικοποίηση:  $O((2E+V)\log V) = O((E+V)\log V)$

-πίνακα αποστάσεων  $d[] = \infty$ ,  $d[s] = 0$

-χρησιμοποιούμε ουρά προτεραιότητας (min-heap),  $pq \leftarrow (d[s], s)$

2. Εξερεύνηση κορυφών:

-σε κάθε επανάληψη, επιλέγουμε την κορυφή  $u$  με την ελάχ απόσταση:  $(\text{curr\_dist}, u)$

-if  $(\text{curr\_dist} > d[u])$  continue;

- $\forall e(u, v)$ :  $d_1 = d[u] + w(u, v)$  if  $(d_1 < d[v])$   $d[v] = d_1$ ,  $pq \leftarrow (d[v], v)$

- $\forall e(v, u)$ :  $d_2 = d[u] + w(u, v) + c(v, u)$  if  $(d_2 < d[v])$   $d[v] = d_2$ ,  $pq \leftarrow (d[v], v)$

3. Επιστροφή αποστέλλεσματος:  $d[t]$

2. Να διατυπώσετε αποδοτικό αλγόριθμο που υπολογίζει το συντομότερο μονοπάτι μεταξύ κορυφών  $s$  και  $t$  του  $G$ , αν επιτρέπεται να χρησιμοποιήσουμε το πολύ  $k$ ,  $1 \leq k \leq n - 1$ , ακμές στην αντίθετη φορά (κάθε φορά που χρησιμοποιούμε μια ακμή στην αντίθετη φορά, θα πρέπει να λάβουμε υπόψη το αυξημένο μήκος της ακμής αυτής στο συνολικό μήκος του μονοπατιού).

1. Αρχικοποίηση:  $O(k(V+E)\log(V+k))$

-πίνακα αποστάσεων  $d[] = \infty$ ,  $d[s][0] = 0$

-χρησιμοποιούμε ουρά προτεραιότητας (min-heap),  $pq \leftarrow (d[s][0], s, 0)$

2. Εξερεύνηση κορυφών:

-σε κάθε επανάληψη, επιλέγουμε την κορυφή  $u$  με την ελάχ απόσταση:  $(\text{curr\_dist}, u, m)$

- $\forall e(u, v)$ :  $d_1 = \text{curr\_dist} + w(u, v)$  if  $(d_1 < d[v](m))$   $d[v](m) = d_1$ ,  $pq \leftarrow (d[v](m), v, m)$

- $\forall e(v, u)$ : if  $(m < k)$   $d_2 = \text{curr\_dist} + w(u, v) + c(v, u)$  if  $(d_2 < d[v](m))$   $d[v](m+1) = d_2$

$pq \leftarrow (d[v](m+1), v, m+1)$

3. Επιστροφή αποστέλλεσματος:  $\min\{d[t]\}$

## Θέμα 4: NP-λήρηματα, Αναγωγές

1. Θεωρήστε το πρόβλημα **Exactly-3-SAT**, που ορίζεται όπως το **3-SAT** με τη διαφορά ότι απαιτούμε κάθε clause να περιέχει ακριβώς 3 literals. Ορίζουμε αντίστοιχα το πρόβλημα **Exactly-4-SAT**, όπου κάθε clause περιέχει ακριβώς 4 literals.

- (α) Αποδείξτε ότι το πρόβλημα **Exactly-3-SAT** είναι NP-πλήρες (θεωρώντας γνωστό ότι το **3-SAT** είναι).
- (β) Αποδείξτε ότι το πρόβλημα **Exactly-4-SAT** είναι επίσης NP-πλήρες.  
Σημείωση: Αν σας διευκολύνει, μπορείτε για το (β) να θεωρήσετε γνωστό ότι το **Exactly-3-SAT** είναι NP-πλήρες.

### 1α. ① Θδο **Exactly-3-SAT** $\in$ NP

Για μία δεδομένη ακολουθία μεταβλητών, μπορούμε να επαληθεύσουμε σε πολυωνυμικό χρόνο αν κάθε πρόταση περιέχει ακριβώς 3 literals και αν ικανοποιείται από την ακολουθία

② Θδο **Exactly-3-SAT** είναι NP-complete  $\leadsto$  μείωση από **3-SAT** σε **Exactly-3-SAT**

Οι προτάσεις του κανονικού **3-SAT** περιέχουν το πολύ 3 literals. Θα τις μετατρέψουμε σε προτάσεις του **Exactly-3-SAT** με ακριβώς 3 literals:

- clause με 1 literal:  $(x_1) \rightarrow (x_1 \vee y \vee \neg y)$

- clause με 2 literals:  $(x_1 \vee x_2) \rightarrow (x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y)$

- clause με 3 literals:  $(x_1 \vee x_2 \vee x_3) \rightarrow$  μένει ίδιο

Έτσι, κάθε πρόταση του **SAT** μετατρέφεται σε μία ή δύο προτάσεις του **Exactly-3-SAT**, διατηρώντας την ικανοποισιμότητα.

Το **3-SAT** είναι NP-λήρες } το **Exactly-3-SAT** είναι NP-λήρες  
} πολυωνυμική μείωση στο **Exactly-3-SAT**

### 1β. ④ Θδο **Exactly-4-SAT** $\in$ NP

Η επαλήθευση δοσμένης ανάθεσης μπορεί να γίνει σε πολυωνυμικό χρόνο

② Θδο **Exactly-4-SAT** είναι NP-complete  $\leadsto$  μείωση από **Exactly-3-SAT** σε **Exactly-4-SAT**

Θα μετατρέψουμε κάθε πρόταση του **Exactly-3-SAT** σε πρόταση του **Exactly-4-SAT** ως εξής:

-  $(x_1 \vee x_2 \vee x_3) \rightarrow (x_1 \vee x_2 \vee x_3 \vee y) \wedge (x_1 \vee x_2 \vee x_3 \vee \neg y)$

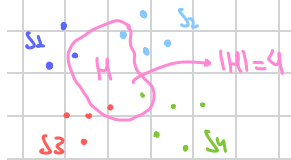
Η μετατροπή διπλασιάζει τα clauses και ο χρόνος είναι πολυωνυμικός

Το **Exactly-3-SAT** είναι NP-λήρες } το **Exactly-4-SAT** είναι NP-λήρες  
} πολυωνυμική μείωση στο **Exactly-4-SAT**

2. Το πρόβλημα **Hitting Set** ορίζεται ως εξής:

Δίνονται ένα σύνολο στοιχείων  $U$ , μια συλλογή υποσυνόλων του  $U$ , που συμβολίζονται με  $S_1, \dots, S_k$ , και ένας θετικός ακέραιος  $B$ . Ζητείται να απαντηθεί αν υπάρχει υποσύνολο  $H \subseteq U$ , με πληθικότητα  $|H| \leq B$ , που να "τέμνει" κάθε σύνολο εισόδου, δηλαδή για κάθε σύνολο εισόδου  $S_i$ , να ισχύει  $|S_i \cap H| \geq 1$ .

Αποδείξτε ότι το πρόβλημα **Hitting Set** είναι NP-πλήρες.



① Θδο **Hitting Set**  $\in$  NP

Θα δείξουμε ότι μια υποψήφια λύση  $H$  μπορεί να επαληθευτεί σε πολωνυμικό χρόνο

- έστω υποσύνολο  $H \subseteq U$ :  $|H| \leq B$
- επαληθεύουμε αν  $\forall S_i$  ισχύει  $S_i \cap H \neq \emptyset$
- ελέγχος κάθε στοιχείου του  $H$  ως προς τα στοιχεία των  $S_i \sim O(k \cdot |H|)$

② Θδο **Hitting Set** είναι NP-complete  $\sim$  μείωση από **Set Cover** στο **Hitting Set**

Πρόβλημα **Set Cover**:

Δοθέν ένα σύνολο στοιχείων  $U$  και μια συλλογή από υποσύνολα  $S_1, S_2, \dots, S_k$ , καθώς και ένας θετικός ακέραιος  $B$ , ζητείται να βρεθεί ένα υποσύνολο των συνόλων  $C \subseteq \{S_1, S_2, \dots, S_k\}$  με μέγεθος το πολύ  $B$ , ώστε η ένωσή τους να καλύπτει ολόκληρο το  $U$ :

$$\bigcup_{S_i \in C} S_i = U$$

Δεδομένο πρόβλημα **Set Cover**  $(U, \{S_1, \dots, S_k\}, B)$ :

- ορίζουμε το σύνολο  $X$  ως **Hitting Set** ως το αρχικό σύνολο  $U$
- κάθε υποσύνολο  $S_i$  του **Set Cover** μετατρέπεται σε  $S_i$  ως **Hitting Set**
- το  $B$  παραμένει ίδιο



## Θέμα 5: επιβεβαίωση μέγιστης ροής

Θεωρούμε s-t δίκτυο  $G(V, E, c)$  με  $n$  κορυφές,  $m$  ακμές, και ακέραια χωρητικότητα  $c(e) \geq 1$  σε κάθε ακμή  $e \in E$ . Στα (1) και (2) παρακάτω, να αιτιολογήσετε την ορθότητα και την υπολογιστική πολυπλοκότητα των αλγορίθμων σας.

1. Δίνεται ροή  $f$  που (υποτίθεται ότι) αποτελεί μέγιστη s-t ροή στο  $G$ . Να διατυπώσετε αλγόριθμο γραμμικού χρόνου που ελέγχει αν η  $f$  αποτελεί πράγματι μια μέγιστη s-t ροή στο  $G$ .

Για να ελέγχουμε αν μια δόση  $f$  είναι η μέγιστη s-t ροή στο  $G$  θα χρησιμοποιήσουμε το θεώρημα Max-Flow, Min-Cut

① Κατασκευή Residual Graph  $G_f \sim O(m)$

$\forall$  ακμή  $(u, v)$ : υπολ χωρητικότητα  $c_f(u, v) = c(u, v) - f(u, v)$

- if  $(c_f > 0)$  προσθήκη  $e(u, v)$  με  $c_f(u, v)$

- if  $(f(u, v) > 0)$  προσθήκη  $e(v, u)$  με  $f(u, v)$

② Ψήρεση προσβάσιμων κορυφών από  $s \sim O(n+m)$

εκτελούμε BFS ή DFS για να βρούμε τις προσβάσιμες κορυφές από το  $s$  στο  $G_f$

- κομβος  $t$  ΜΜ προσβάσιμος  $\rightarrow f$  είναι μέγιστη

- κομβος  $t$  προσβάσιμος  $\rightarrow f$  δεν είναι μέγιστη

2. Έστω ότι διαπιστώνουμε ότι η  $f$  αποτελεί πράγματι μια μέγιστη s-t ροή στο  $G$ , αλλά στη συνέχεια ανακαλύπτουμε ότι η πραγματική χωρητικότητα  $c'(e)$  μιας ακμής  $e$  είναι μεγαλύτερη κατά  $k$  ( $k \geq 1$ ) μονάδες από τη χωρητικότητα  $c(e)$  που είχαμε θεωρήσει αρχικά. Να διατυπώσετε αποδοτικό αλγόριθμο που (αν χρειάζεται) τροποποιεί την  $f$  σε μία μέγιστη s-t ροή  $f'$  για το δίκτυο  $G'$ , το οποίο προκύπτει από το  $G$  θέτοντας  $c'(e) = c(e) + k$ . Ο χρόνος εκτέλεσης του αλγορίθμου σας πρέπει να είναι σημαντικά μικρότερος από τον χρόνο υπολογισμού της  $f'$  εξ' αρχής.

① Update residual graph

νέο capacity:  $c'(e) = c(e) + k$

νέο residual capacity:  $c'_f(e) = c_f(e) + k$

② Ψήρεση αυθηκτικών μονοπατιών + επαναπροσαρμογή ροής  $\rightarrow$  το πολύ  $k$  loops  $O(k \cdot (nm))$

- χρήση BFS για έναρξη μονοπατιού s-t

- εάν υπάρχει, εισαγωγή μονοπατιού + bottleneck (ελάχιστη υπολ χωρητικότητα)

αύξηση ροής στο μονοπάτι κατά bottleneck

ανανέωση residual graph

αλλιώς, εισαγωγή  $f$

Άσκηση: Τηλεφωνικό ~ Greedy

καρτά:  $\{(v_i, p_i)\}_{i=1}^n$   $v_i$ : value καρτών,  $p_i$ : δύναμη για να σπάσει

σφυριά:  $\{f_i\}_{i=1}^n$   $f_i$ : δύναμη κάθε σφυριού, κάθε σφυρί 1 φορά

στόχος: max value

① Sort καρτά  $\{(v_i, p_i)\}_{i=1}^n$  :  $\forall i < j \Rightarrow (v_i > v_j) \parallel (v_i = v_j \ \&\& \ p_i < p_j)$  }  $O(n \log n)$

Sort σφυριά  $\{f_i\}_{i=1}^n$  :  $\forall i < j \Rightarrow p_i < p_j$

inivakas  $used[1..n] = 0$ ,  $opt\_value = 0$

②  $\forall$  καρτά  $(v_i, p_i)$ : for  $(j=1 \text{ to } n)$  if  $(used[j] == 0 \ \&\& \ f_j \geq p_i)$   $\left\{ \begin{array}{l} \rightarrow opt\_value += v_i \\ \rightarrow used[j] = i \end{array} \right.$  }  $O(n^2)$

③ Return  $opt\_value$

Απόδειξη με ενδεικτικό αυταπαθούς ++

Coin Change

coin system:  $c_1, c_2, \dots, c_k$  (κατηγορίες κερμάτων)

νόσο  $X$

Στόχος: να βρω minimum #κερμάτων να κάνουν  $X$

Greedy: λειτουργεί μόνο για ευνοϊκά coin systems ( $c_i$  είναι παλιότερα μεταξύ τους)

Dynamic Programming:  $dp[0..X]$ :  $dp[j] = \min(\#κερμάτων)$  να κάνουν  $j$

Αρχικοποίηση:  $dp[1..X] = \infty$ ,  $dp[0] = 0$

Αναδρομική σχέση:  $dp[j] = 1 + \min_{i \in A} \{dp[j - c_i]\}$

curr.min =  $\infty$

$\forall c_i$ : if  $(j - c_i \geq 0)$  curr.min =  $\min\{\text{curr.min}, dp[j - c_i]\}$

$dp[j] = 1 + \text{curr.min}$

Τελικό αποτέλεσμα:  $dp(x)$

## Interval Scheduling

① Μέγιστο πλήθος μη-επικαλυπτόμενων διαστημάτων → Greedy

Input:  $\{(s_i, f_i)\}_{i=1}^n$

Greedy κριτήριο: επιλέγουμε πάντα το διάστημα που τελειώνει νωρίτερα

1. sort intervals:  $\{(s_i, f_i)\}_{i=1}^n, i < j \Rightarrow f_i < f_j$  →  $O(n \log n)$
2. επιλογή πρώτου διαστήματος που άρχει νωρίτερα }  $O(n)$
3. παραίτηση διαστημάτων που επικαλύπτονται

② Weighted Interval Scheduling (μεγιστοποίηση βάρους) → dp

Input:  $\{(s_i, f_i), w_i\}_{i=1}^n$

Dynamic Programming:  $dp[i]$ : μέγιστο κέρδος για διαστήματα  $\{1, \dots, i\}$

1. sort intervals:  $\{(s_i, f_i), w_i\}_{i=1}^n, i < j \Rightarrow (f_i < f_j) \vee (f_i = f_j \text{ AND } w_i > w_j)$
2. compute  $prev[1..n]$ :  $prev[j]$  = το index του αμέσως προηγ. μη επικαλυπτόμενου interval
3. initialization:  $dp[0] = 0$
4. for  $(i=1 \text{ to } n)$ :  $dp[i] = \max \begin{cases} dp[i-1] & \text{// exclude} \\ w_i + dp[prev[i]] & \text{// include} \end{cases}$
5. Return  $dp[n]$

③ Interval Partitioning (ελάχιστος αριθμός νόρων) → Greedy

Input:  $\{(s_i, f_i)\}_{i=1}^n$

Στόχος: ελάχιστος αριθμός νόρων  $\equiv$  πόσα διαδ. επικαλύπτ. ταυτόχρονα

Greedy κριτήριο: επιλέγουμε το διάστημα που ξεκινάει νωρίτερα, αν υπάρχει διαθέσιμος νόρος, τον επαναχρησιμοποιούμε, αλλιώς δεσμεύουμε νέο

1. sort intervals:  $\{(s_i, f_i)\}_{i=1}^n, i < j \Rightarrow s_i < s_j$  →  $O(n \log n)$
2. min heap για αρχές ευερχών διαστημάτων
3.  $\forall$  interval  $(s_i, f_i)$ : - αν το μικρότερο στοιχείο του σωρού είναι μικρότερο ή ίσο από το νέο διάστημα, αφαιρούμε το παλαιότερο διάστημα από τον σωρό  
- εισάγουμε το νέο χρόνο λήξης στο σωρό
4. Return μέγιστο μέγεθος του σωρού

## Fractional Knapsack Problem

Input:  $\{(s_i, v_i)\}_{i=1}^n$   $s_i$ : μέγεθος αντικ  $v_i$ : αξία αντικ  
μέγεθος σακιδίου  $B$

Output:  $\forall (s_i, v_i)$  να βρω  $f_i \in [0, 1]$ :  $\sum f_i \cdot s_i \leq B$  και  $\sum f_i \cdot v_i \rightarrow \max$

Greedy κριτήριο: επιλέγουμε αντικείμενα με υψηλότερο λόγο μέγεθος / βάρος μέχρι να γεμίσει το σακίδιο

1. sort  $\{(w_i, p_i)\}_{i=1}^n$   $i < j \Rightarrow v_i/s_i > v_j/s_j$

2. total\_val = 0

3.  $\forall$  αντικ  $(s_i, v_i)$ : if  $(B \geq s_i)$   $B = B - s_i$ ,  $f_i = 1$ , total\_val +=  $v_i$   
else  $f_i = B/s_i$ , total\_value +=  $f_i \cdot v_i$ , break

3. Τελικό αποτέλεσμα:  $f[1..n]$ , total\_value

## 0/1 Knapsack Problem

Input:  $\{(s_i, v_i)\}_{i=1}^n$

Output:  $S \subseteq \{1, \dots, n\}$ :  $\sum s_i \leq B$ ,  $\sum v_i \rightarrow \max$

Dynamic programming:  $dp[i][b] \rightarrow \max$  value για αντ  $\{1..i\}$ ,  $b$  μέγεθος σακιδίου

1. Αρχικοποίηση:  $dp[i][0] = 0$ ,  $dp[0][b] = 0$

2. Binary choice:  $dp[i][b] = \max \begin{cases} dp[i-1][b] & \text{// exclude} \\ \text{if } (b \geq s_i) v_i + dp[i-1][b-s_i] & \text{// include, if possible} \end{cases}$

3. Τελικό αποτέλεσμα:  $dp[n][B]$

## Partition Subset Sum

Input:  $A = \{s_i\}_{i=1}^n$  και target sum  $B$

Output:  $S \subseteq \{1..n\}$ :  $\sum s_i = B$

Dynamic Programming:  $dp[i][b]$  αριθμός συνόλων με  $\{s_1..s_i\}$  με άθροισμα  $b$

1. Αρχικοποίηση:  $dp[i][0] = 1$ ,  $dp[0][b] = 0$ ,  $b > 0$

2. Binary choice:  $dp[i][b] = \text{sum} \begin{cases} dp[i-1][b] \\ \text{if } (b \geq s_i) s_i + dp[i-1][b-s_i] \end{cases}$

3. Final Result:  $dp[n][B]$

## Longest Increasing Subsequence (LIS)

Input:  $a = \{a_1, \dots, a_n\}$

Output: μέγιστος μήκος  $\nearrow$  υπο-ακολουθία

Dynamic Programming:  $dp[i]$ : μέγιστος μήκος αύξουσας υποακολουθίας με  $\{a_1, \dots, a_i\}$  να περιέχει το  $a_i$

1. Απικονόμηση:  $dp[i] = 1$  (κάθε στοιχείο μόνο του)
2. Choice:  $\forall i$  for  $(j=1 \text{ to } i-1)$  if  $(a_j < a_i)$   $dp[i] = \max\{dp[i], dp[j] + 1\}$
3. Final result:  $\max\{dp[i]\}$

## Longest k-Almost Increasing Subsequence

Input:  $a = \{a_1, \dots, a_n\}, k$

Output: μέγιστος μήκος  $\nearrow$  υπο-ακολουθία με  $k$  το πολύ παραβιάσεις

Dynamic Programming:  $dp[i][m]$ : μέγιστος μήκος αύξουσας υποακολουθίας με  $\{a_1, \dots, a_i\}$  να περιέχει το  $a_i$  με  $m$  παραβιάσεις

1. Απικονόμηση:  $dp[i][0] = 1$
2. Choice:  $\forall i$  for  $(j=1 \text{ to } i-1)$  if  $(a_j < a_i)$   $dp[i][m] = \max\{dp[i][m], dp[j][m] + 1\}$   
else if  $(m > 0)$   $dp[i][m] = \max\{dp[i][m], dp[j][m-1] + 1\}$
3. Final result:  $\max\{dp[i][m]\}$

## String Matching

Input: κείμενο  $T[1..n]$  πρότυπο  $P[1..m]$

Output: εύρεση  $T_s = T[s..s+m-1] = P$

① Brute force: να πάρω 1<sup>ο</sup> γράμμα, μετά το 2<sup>ο</sup> να πάρω επόμενο κτλ. Αν σε τζόκο, mismatch  
 $\hookrightarrow O(n \cdot m)$  συνεχίζω από εκεί να βρω το πρώτο match

② Almost brute force: ίδιο με πριν, αλλά σταματώ το loop σε 1<sup>ο</sup> mismatch  
 $\hookrightarrow O(n \cdot m)$

③ Karp - Rabin: - υπολογισμός hash προτύπου  
- υπολογισμός hash πρώτης υποσυμβολοσειράς κειμ. μεγέθους  $\delta$   
- rolling hash για επόμενα τμήματα  
- σε περίπτωση match: έλεγχος και με το πρότυπο

## Longest Common Subsequence (LCS)

Input:  $X = \{x_1 \dots x_n\}$   $Y = \{y_1 \dots y_m\}$

Dynamic Programming:  $dp[i][j]$ : μέγιστο μήκος κοινής ακολουθίας τμή  $X(1..i)$  με  $Y(1..j)$

1. Απικονόηση:  $dp[0][j] = 0$   $dp[i][0] = 0$

2. Αναδρομή:  $dp[i][j] \rightarrow$  if  $(x_i == y_j)$   $dp[i][j] = 1 + dp[i-1][j-1]$   
else  $dp[i][j] = \max \begin{cases} dp[i-1][j] \\ dp[i][j-1] \end{cases}$

3. Final result:  $dp[n][m]$

## Edit Distance Problem

Input:  $X(1..n)$ ,  $Y(1..m)$   $\rightarrow \delta$   $\rightarrow \delta$   $\rightarrow \text{cost}(x_i, y_j)$

Output: νόσες ενέργειες {προσθήκη/διαγραφή/αλλαγή} χαρακτήρα πρέπει να γίνουν ώστε  $X=Y$

Dynamic programming:  $dp[i][j]$ : άσυν θα  $X(1..i)$ ,  $Y(1..j)$

1. Απικονόηση:  $dp[i][0] = i \cdot \delta$   $dp[0][j] = j \cdot \delta$

2. Αναδρομή:  $dp[i][j] = \min \begin{cases} dp[i-1][j] + \delta & \text{// διαγράψω από } X \\ dp[i][j-1] + \delta & \text{// εισαγωγή στο } Y \end{cases}$   
if  $(x_i == y_j)$   $dp[i][j] = \min \{ dp[i][j], dp[i-1][j-1] \}$   
else  $dp[i][j] = \min \{ dp[i][j], dp[i-1][j-1] + \text{cost}(x_i, y_j) \}$

3. Final Result:  $dp[n][m]$

## Max Weight Independent Set (linear)

Input:  $\{p_i\}_{i=1}^n$

Output:  $SS \subseteq \{1..n\} \cdot \sum p_i \rightarrow \max, i \in S \Rightarrow \{i+1, i-1\} \notin S$

Dynamic Programming:  $dp[i]$  βέλτιστη άσυν θα  $\{p_1, \dots, p_i\}$

1. Απικονόηση:  $dp[0] = 0$ ,  $dp[1] = p_1$

2. Binary Choice:  $dp[i] = \max \begin{cases} dp[i-1] & \text{// exclude } i \\ p_i + dp[i-2] & \text{// include } i \end{cases}$

## Max Weight Independent Set (tree)

Input: tree  $\{(u, w_u, \text{children})\}$

Output:  $\sum p_i \rightarrow \max$ , αν επιλεγεί κόμβος  $u$ , δεν επιλέγονται τα παιδιά του

Dynamic Programming:  $dp[u](yes/no)$ : βέλτιστη άσυν με ή χωρίς κόμβο  $u$

1. Απικονόηση:  $\forall$  φυτό  $(u, w_u, \emptyset)$ :  $dp[u](yes) = w_u$ ,  $dp[u](no) = 0$

2. Αναδρομή:  $dp[u](yes) = w_u + \sum_{v \in \text{children}} dp[v](no)$   
 $dp[u](no) = \sum_{v \in \text{children}} \max \{ dp[v](no), dp[v](yes) \}$

3. Final Result:  $\max \{ dp[\text{root}](yes), dp[\text{root}](no) \}$

## Partitioning Sequence Problem

Input:  $a = \{a_1, \dots, a_n\}$ ,  $k$

Output: διαμερισμός σε  $k$  ομάδες ώστε να ελαχιστοποιηθεί το μέγιστο υπο-άθροισμα

Dynamic Programming  $dp[i][m]$ : μέγιστο άθροισμα για στοιχεία  $\{a_1 \dots a_i\}$  και  $m$  διαμερίσεις

1. αρχικοποίηση:  $dp[i][m] = \infty$ ,  $dp[i][1] = \sum_{j=1}^i a_j$ ,  $dp[0][m] = 0$

2. αναδρομή:  $\text{for } (p=0 \text{ to } i-1) \left. \begin{array}{l} \text{left-part} = dp[p][m-1] \\ \text{right-part} = \sum_{j=p+1}^i a_j \end{array} \right\} dp[i][m] = \min \left\{ \begin{array}{l} dp[i][m] \\ \max\{\text{left}, \text{right}\} \end{array} \right\}$

3. final result:  $dp[n][k]$

↳ Παράδειγμα

Input:  $a = \{(a_i, f_i)\}_{i=1}^n$ ,  $k$

Output: βάρος κουμπίου με  $\{a_1, \dots, b\}$  κουμπία  $\rightarrow \sum_{i=a}^b i \cdot f_i$

διαμερισμός ώστε να ελαχιστοποιείται το συνολικό βάρος κουμπιών

Dynamic Programming  $dp[i][m]$ : ελάχιστο συνολ. βάρος για  $\{a_1 \dots a_i\}$  και  $m$  διαμερίσεις

1. αρχικοποίηση:  $dp[i][m] = \infty$ ,  $dp[i][1] = \sum_{j=i}^i f_j$ ,  $dp[0][m] = 0$

2. αναδρομή:  $\text{for } (p=0 \text{ to } i-1) \quad dp[i][m] = \min \{ dp[i][m], dp[p][m-1] + \sum_{j=p+1}^i (j-p) f_j \}$

3. final result:  $dp[n][k]$

## Chem

Input:  $N$  αυσίες,  $K$  φιάλες,  $A[i][j]$  κόστος ζονοθ  $i, j$  αμ ίδια φιάλη

Output: ζονοθέτηση με  $\min$  συνολικό κόστος

Dynamic Programming:  $dp[i][k]$ : ζονοθ αυσια  $\{1 \dots i\}$  σε  $k$  φιάλες

1. αρχικοποίηση:  $dp[0][0] = 0$ ,  $dp[0][k] = \infty$

2.  $\text{for } (j=1 \text{ to } i) \quad dp[i][m] = \min \{ dp[i][m], dp[j][k-1] + \text{cost}[j+1][i] \}$

3. final result:  $dp[n][k]$

Υπολογισμός cost:  $\text{for } (i=1 \text{ to } N) \text{ for } (j=i \text{ to } N) \quad \text{cost}[i][j] = \text{cost}[i][j-1] + \sum_{p=i}^{j-1} A[p][j]$

## Revellion

Input:  $\{h_i\}_{i=1}^N$ ,  $k$ : max αριθμός ατόμων ανά ομάδα

Output: μεγιστοποίηση συνολ. ευτυχ  $\rightarrow$  ευτυχία ομάδας  $\{i \dots j\} = \max \{h_i \dots h_j\}$

Dynamic Programming:  $dp[i]$  αὖτη για άτομα  $\{1 \dots i\}$

1. Αρχικοποίηση:  $dp[0] = 0$

2. Αναδρομή:  $\text{for } (j=1 \text{ to } K) \quad dp[i] = \max \{ dp[i], dp[i-j] + j \cdot \max \{h_{i-j+1} \dots h_i\} \}$

3. final result:  $dp[n]$

## Line Market

Input: αριθμός κατασκευμάτων  $N$ , διάστημα  $M$ ,  $\{(s_i, t_i)\}_{i=1}^N$

Output: μέγιστη ελάχιστη απόσταση μεταξύ δύο κατασκευμάτων

- αν γνωρίζουμε συγκεκριμένη  $D$ , μπορούμε να ελέγξουμε αν είναι δυνατό να τοποθετηθούν τα κατασκευάσματα σε αυτή την απόσταση  $\rightarrow$  πρόβλημα απόφασης

1 binary search στο  $[1, L]$ ,  $L = \sum (t_i - s_i) \rightarrow O(M \log L)$

mid =  $\frac{\text{left} + \text{right}}{2}$

greedy τοποθέτηση καταστ με απόσταση mid  $\left\{ \begin{array}{l} \text{γίνεται} \rightarrow \text{best-dist} = \text{mid}, \text{left} = \text{mid} + 1 \\ \text{δεν γίνεται} \rightarrow \text{right} = \text{mid} - 1 \end{array} \right.$

2. return best distance

## Bicriteria MST

Input:  $\{(u_i, v_i, p_i, w_i)\}_{i=1}^M$

Output: ελάχιστο δέντρο με  $\max \sum p_i$  και  $\min \sum w_i \sim \max \frac{\sum p_i}{\sum w_i} = \lambda$

Πρόβλημα βέλτιστοποίησης  $\rightarrow$  απόφασης

$\frac{\sum p_i}{\sum w_i} \geq \lambda \Leftrightarrow \sum p_i - \lambda \sum w_i \geq 0 \Leftrightarrow \sum (p_i - \lambda w_i) \geq 0 \sim \max \lambda$

1. mid =  $(\text{low} + \text{high}) / 2$

2.  $\forall$  edge  $(u, v, w, p)$ : adj-weight =  $p - \text{mid} \cdot w$

3. MST  $\leftarrow$  Kruskal με βάση adj-weight  $\rightarrow O(M \log M) \cdot O(\log \text{percutson})$

4. if (total adj-sum(MST)  $\geq 0$ ) low = mid

else high = mid

$\rightarrow$  συγκεκριμένο # loops για συγκεκριμένη ακρίβεια

5. ανακατασκευή δέντρου με low

## Bottleneck Shortest Path

Input:  $G(V, E)$  με βάση σε ακμές

Output: s-t μονοπάτι, όπου ελάχιστο βάρος ακμής είναι μέγιστο δυνατό

Binary Search σε  $[\text{left}, \text{right}] = [\min(w), \max(w)]$

while  $(\text{left} \leq \text{right})$   $\text{mid} = (\text{left} + \text{right}) / 2 \rightarrow$  BFS μόνο με ακμές  $\leq \text{mid}$

if  $(\exists \text{ s-t μονοπάτι})$   $\text{best} = \text{mid}$ ,  $\text{left} = \text{mid} + 1$

return best else  $\text{right} = \text{mid} - 1$

BFS:  $O(V+E)$

Σύνολο:  $O((V+E) \cdot \log w)$

## Max-flow

Input:  $G(V, E)$  με χωρητικότητα  $c(u, v)$  στις ακμές

Output: s-t max flow

Κατασκευάζω residual graph:  $\forall \text{ edge } (u, v) \text{ με flow } f(u, v)$

- υποή χωρη:  $\text{residual\_graph}(u)(v) = c(u, v) - f(u, v)$

- αντιστροφή ακμής:  $\text{residual\_graph}(v)(u) = f(u, v)$

Όσο  $\exists$  ενισχυτ μονοπάτι:

$\text{augmentingPath} \leftarrow \text{BFS}(G_f, s, t) \sim O(E+V)$  Σύνολο:  $O((E+V) \cdot \text{maxflow})$

$\text{min\_capacity} = \text{min capacity of edge } G_f \text{ in path}$

$\text{max\_flow} += \text{min\_capacity}$

Ενημέρωση  $G_f$ : (για κάθε  $e(u, v)$  στο path)

-  $\text{residual\_graph}(u)(v) -= \text{min\_capacity}$

-  $\text{residual\_graph}(v)(u) += \text{min\_capacity}$

Return max-flow

## Stable Marriage Problem

Input:  $M = \{m_1, \dots, m_n\} \forall m_i$ : λίστα προτιμήσεων γυναικών με φθίν σειρά

$W = \{w_1, \dots, w_n\} \forall w_i$ : λίστα προτιμήσεων ανδρών με φθίν σειρά



! 3-SAT

! 3Dimensional Matching

! Set Cover

πρόβλημα  $\in$  NP

δημιουργία άνω  $\left. \begin{array}{l} \text{δημιουργία άνω} \\ \text{έλεγχος άνω} \end{array} \right\}$  πολυων χρόνος

$\Pi_1 \in \text{NP} \Leftrightarrow \bar{\Pi}_1 \in \text{coNP}$

πρόβλημα  $\in$  NP-complete

$\Pi \in \text{NP}$

$\exists$  αναγωγή από γνωστο NP-complete με πολυων χρ

SAT:  $\rightarrow$  NP-complete

λογική πρόταση  $\phi$  σε CNF. Είναι η ικανοποίησιμη;

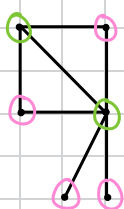
Κύκλος Hamilton  $\leq$  P TSP με αποσλ 1 και 2 - TSP(1,2)

| Κύκλος Hamilton                                                     | TSP(1,2)                                                                                                                                                              |
|---------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $G(V,E)$                                                            | $\langle x_1 \dots x_n \rangle, d(x_i, x_j) \in \{1, 2\}$ $\rightarrow$ βάζουμε $d(x_i, x_j) = \begin{cases} 1, & e(x_i, x_j) \\ 2, & \notin e(x_i, x_j) \end{cases}$ |
| έχει το G κύκλο Hamilton?                                           | $\exists$ TSP περίοδ με μήκος = <u>target-val</u>                                                                                                                     |
| $\hookrightarrow$ κύκλος που περνάει από κάθε κορυφή ακριβώς 1 φορά | $\hookrightarrow$ συντομότερη δυνατή διαδρομή που επιστρέφει 1 φορά κάθε πόλη και επιστρέφει στην αρχική                                                              |

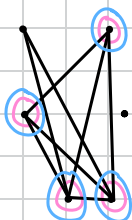
Vertex Cover: σύνολο κορυφών όπου κάθε ακμή έχει τουλάχιστον ένα από τις κορυφές

Independent Set: σύνολο κορυφών χωρίς καμία ακμή μεταξύ τους

Clique: σύνολο κορυφών, που είναι όλες γειτονικές μεταξύ τους ανα δύο



G



$\bar{G}$

G: vertex cover  $\leq k$

G: independent set  $\geq n-k$

$\bar{G}$ : clique  $\geq n-k$

k-SAT: λογική πρόταση  $\phi$  σε k-CNF:  $\phi = C_1 \wedge \dots \wedge C_m$ ,  $C_i = l_{i1} \vee \dots \vee l_{ik}$ ,  $l_{ij} \in \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\}$   
 $\exists$  αναθεση τιμών ώστε  $\phi \in \text{SAT} \Rightarrow$  κάθε  $C_i$  true  $\rightarrow$  τουλάχιστον 1  $l_{ij}$  true