Δυναμικός Προγραμματισμός - Ασκήσεις

Αναστάσης Κολιόπουλος

A - Frog 1

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(i:)$, όπου $1 \leq i \leq n$, το υποπρόβλημα εύρεσης του ελάχιστου κόστους για να μεταβούμε από την πέτρα i στην πέτρα n.
- (Καθορισμός επιλογών). Έχουμε τις εξής δύο επιλογές. Είτε πηγαίνουμε στην πέτρα $i+1$, είτε στην πέτρα $i+2$. Είναι προφανές ότι αν βρισκόμαστε στην πέτρα $n-1$ η δεύτερη επιλογή δεν είναι εφικτή, καθώς και ότι αν βρισκόμαστε στην πέτρα n έχουμε φτάσει στον προορισμό μας.
- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει από τις παραπάνω επιλογές.

$$S(i) = \begin{cases} 0, & i = n, \\ |h_{n-1} - h_n|, & i = n-1, \\ \min\{S(i+1) + |h_i - h_{i+1}|, S(i+2) + |h_i - h_{i+2}|\}, & 1 \leq i \leq n-2. \end{cases}$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα θα επιλυθούν κατά φθίνουσα σειρά του δείκτη i.
- (Επίλυση αρχικού προβλήματος). Η λύση είναι το $S(1:)$.

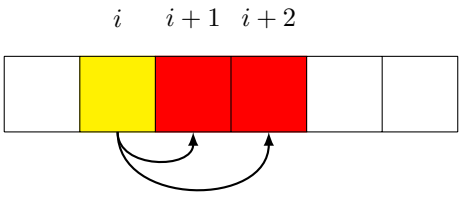
Θα εκτελέσουμε τον αλγόριθμο ΔΠ για το στιγμιότυπο $H = [10, 30, 40, 20]$, στο οποίο $n = 4$. Για $i = 4, 3, 2, 1$ έχουμε ότι

$$\begin{aligned} &S_4 = 0, \\ &S_3 = |h_3 - h_4| = 20, \\ &S_2 = \min\{S_3 + |h_2 - h_3|, S_4 + |h_2 - h_4|\} = \min\{20 + 10, 0 + 10\} = 10, \\ &S_1 = \min\{S_2 + |h_1 - h_2|, S_3 + |h_1 - h_3|\} = \min\{10 + 20, 20 + 30\} = 30. \end{aligned}$$

Επομένως, το ελάχιστο κόστος για να μεταβούμε από την πέτρα 1 στην πέτρα 4 είναι ίσο με 30. Η συνάρτηση FrogOne κωδικοποιεί την ιδέα σε C++.

```
int FrogOne(vector<int>& H, int N) {
    dp[N] = 0;
    dp[N - 1] = abs(H[N - 1] - H[N]);
    for (int i = N - 2; i >= 1; --i)
        dp[i] = min(dp[i + 1] + abs(H[i] - H[i + 1]), dp[i + 2] + abs(H[i]
            - H[i + 2]));
    return dp[1];
}
```

Είναι προφανές ότι η χρονική και η χωρική πολυπλοκότητα είναι τάξης $\Theta(n)$. Παρ'όλα αυτά, παρατηρούμε ότι για ένα δεδομένο i χρειάζεται να αποθηκεύουμε στη μνήμη μόνο τις λύσεις των υποπροβλημάτων $i+1, i+2$.



Σχήμα 1. Το υποπρόβλημα i εξετάζει τα υποπροβλήματα $i+1, i+2$.

Αυτή η παρατήρηση μας οδηγεί στο να κρατάμε έναν πίνακα μόνο με τρεις θέσεις μνήμης, μειώντας τη χωρική πολυπλοκότητα σε $\Theta(1)$.

```
int FrogOne(vector<int>& H, int N) {
    dp[N % 3] = 0;
    dp[(N - 1) % 3] = abs(H[N - 1] - H[N]);
    for (int i = N - 2; i >= 1; --i)
        dp[i % 3] = min(dp[(i + 1) % 3] + abs(H[i] - H[i + 1]), dp[(i + 2)
            % 3] + abs(H[i] - H[i + 2]));
    return dp[1];
}
```

Αυτή η ιδέα γενικεύεται σε οποιοδήποτε πρόβλημα ΔΠ, στο οποίο για να λύσουμε το τρέχον υποπρόβλημα χρειαζόμαστε τη λύση των c προηγούμενων υποπροβλημάτων, όπου c μία μικρή σταθερά μεγαλύτερη-ίση του 1. Ο πίνακας που θα δεσμεύσουμε θα έχει $c+1$ θέσεις, μία για το πάρον υποπρόβλημα και άλλες c για τα προηγούμενα υποπροβλήματα.

B - Frog 2

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(i:)$, όπου $1 \leq i \leq n$, το υποπρόβλημα εύρεσης του ελάχιστου κόστους για να μεταβούμε από την πέτρα i στην πέτρα n.
- (Καθορισμός επιλογών). Οι επιλογές που έχουμε όταν στεκόμαστε στην πέτρα i, είναι να πάμε σε οποιαδήποτε πέτρα του συνόλου $\{i+1, \ldots, \min\{i+k, n\}\}$. Είναι προφανές ότι αν βρισκόμαστε στην πέτρα n έχουμε φτάσει στον προορισμό μας.
- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει από τις παραπάνω επιλογές.

$$S(i) = \begin{cases} 0, & i = n, \\ \min_{i+1 \leq i^* \leq \min\{i+k,n\}} \{S(i^*) + |h_i - h_{i^*}|\}, & 1 \leq i \leq n-1. \end{cases}$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα θα επιλυθούν κατά φθίνουσα σειρά του δείκτη i.

- (Επίλυση αρχικού προβλήματος). Η λύση είναι το $S(1:)$.

Την ιδέα υλοποιεί η συνάρτηση FrogTwo.

```
int FrogTwo(vector<int>& H, int N, int K) {
    dp[N] = 0;
    for (int i = N - 1; i >= 1; --i) {
        dp[i] = INF;
        for (int j = i + 1; j <= min(i + K, N); ++j)
            dp[i] = min(dp[i], dp[j] + abs(H[i] - H[j]));
    }
    return dp[1];
}
```

Η χρονική πολυπλοκότητα της παραπάνω συνάρτησης είναι τάξης $\Theta(nk) = O(n^2)$ και η χωρική τάξης $\Theta(n)$.

Bonus. $K \leq 10^5$.

Είναι προφανές ότι στην περίπτωση αυτή, ένας αλγόριθμος ΔΠ τάξης $O(n^2)$ δεν είναι εφικτός. Παρατηρούμε ότι μπορούμε να «σπάσουμε» την απόλυτη τιμή στην αναδρομική σχέση παίρνοντας δύο περιπτώσεις. Έχουμε ότι

$$S(i) = \min \begin{cases} \min_{i+1 \leq i^* \leq \min\{i+k,n\}} \{S(i^*) - h_{i^*}\} + h_i, & \text{για } h_i \geq h_{i^*}, \\ \min_{i+1 \leq i^* \leq \min\{i+k,n\}} \{S(i^*) + h_{i^*}\} - h_i, & \text{για } h_i < h_{i^*}, \end{cases}$$

με την οριακή συνθήκη να παραμένει ίδια.

Ο υπολογισμός του πίνακα ΔΠ μπορεί πλέον να γίνει σε χρόνο $\Theta(n \lg k) = O(n \lg n)$, χρησιμοποιώντας *δέντρα διαστημάτων* με τη χωρική πολυπλοκότητα να διατηρείται ίδια. Η κωδικοποίηση της λύσης αυτής αφήνεται ως άσκηση στον αναγνώστη.

Ανέβα τη σκάλα

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(i:)$, όπου $0 \leq i \leq n+1$, το υποπρόβλημα εύρεσης του μέγιστου αθροίσματος που μπορούμε να πετύχουμε εκκινώντας από το σκαλοπάτι i και καταλήγοντας στο σκαλοπάτι $n+1$.
- (Καθορισμός επιλογών). Οι επιλογές που έχουμε όταν στεκόμαστε στο σκαλοπάτι i, είναι να ανέβουμε στο σκαλοπάτι $i+1$ ή στο σκαλοπάτι $i+2$. Είναι προφανές ότι αν βρισκόμαστε στο σκαλοπάτι n η δεύτερη επιλογή δεν είναι εφικτή, καθώς και ότι αν είμαστε στο σκαλοπάτι $n+1$ έχουμε ανέβει ολόκληρη τη σκάλα.
- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει

από τις παραπάνω επιλογές.

$$S(i) = \begin{cases} 0, & i = n+1, \\ a_n, & i = n, \\ \max\{S(i+1), S(i+2)\} + a_i, & 0 \le i \le n-1. \end{cases}$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα θα επιλυθούν κατά φθίνουσα σειρά του δείκτη i.
- (Επίλυση αρχικού προβλήματος). Η λύση είναι το $S(0:)$.

Η κωδικοποίηση του ΔΠ ακολουθεί.

```
int Stairstep(vector<int> A, int N) {
    dp[N + 1] = 0;
    dp[N] = A[N];
    for (int i = N - 1; i >= 0; --i)
        dp[i] = max(dp[i + 1], dp[i + 2]) + A[i];
    return dp[0];
}
```

Η χρονική και η χωρική πολυπλοκότητα είναι τάξης $\Theta(n)$, με τη χωρική να επιδέχεται βελτίωση σε $\Theta(1)$, εφαρμόζοντας τη βελτιστοποίηση που περιγράφηκε παραπάνω.

Bonus. Σε κάθε βήμα μπορούμε να ανέβουμε μέχρι K σκαλιά, όπου $K \le 10^6$.

Η αναδρομική σχέση πλέον γράφεται ως

$$S(i) = \begin{cases} 0, & i = n+1, \\ \max_{i+1 \le i^* \le \min\{i+k, n+1\}}\{S(i^*)\} + a_i, & 0 \le i \le n. \end{cases}$$

Δεδομένου ότι για ένα συγκεκριμένο i μας ενδιαφέρουν οι τιμές του παραθύρου $\{i+1, \ldots, \min\{i+k, n+1\}\}$, το πρόβλημα ανάγεται στην εύρεση του *μεγίστου κάθε κυλιόμενου παραθύρου μεγέθους* k, το οποίο με τη χρήση *μονότονης ουράς*, επιλύεται σε χρόνο $O(n)$ και χώρο $\Theta(n)$.

Τρίγωνο

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

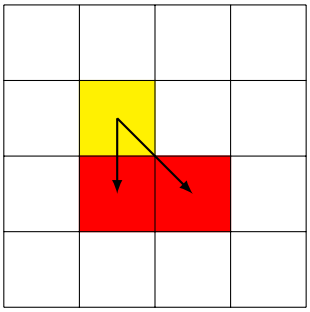
- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(i:, j)$, όπου $1 \le i \le n$ και $1 \le j \le i$, το υποπρόβλημα εύρεσης του μεγαλύτερου αθροίσματος που μπορούμε να πετύχουμε, ξεκινώντας από το στοιχείο της i-οστής γραμμής και j-οστής στήλης και καταλήγοντας σε κάποιο από τα στοιχεία της n-οστής γραμμής.
- (Καθορισμός επιλογών). Μπορούμε είτε να μεταβούμε στο στοιχείο $(i+1, j)$ ή στο $(i+1, j+1)$. Αν ισχύει ότι $i = n$ έχουμε φτάσει στο τελευταίο επίπεδο.

- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει από τις παραπάνω επιλογές.

$$S(i,j) = \begin{cases} a_{ij}, & i = n, \\ \max\{S(i+1,j), S(i+1,j+1)\} + a_{ij}, & 1 \leq i \leq n-1. \end{cases}$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα θα επιλυθούν κατά φθίνουσα σειρά του δείκτη i. Παρατηρήστε ότι όταν ορίσαμε το υποπρόβλημα $S(i:,j)$, δεν ορίσαμε τον δείκτη j ούτε ως προθεματικό, αλλά ούτε ως επιθεματικό. Ο λόγος για αυτό είναι ότι η λύση του $S(i:,j)$ δεν εξαρτάται από κανένα υποπρόβλημα της γραμμής i. Έτσι έχουμε τη δυνατότητα να επιλέξουμε αν στον υπολογισμό, ο δείκτης j θα αυξάνεται ή θα μειώνεται.

- (Επίλυση αρχικού προβλήματος). Η λύση είναι το $S(1:,1)$.



Σχήμα 2. Η λύση του υποπροβλήματος $S(i,j)$ εξαρτάται από τα δύο υποπροβλήματα της γραμμής $i+1$.

Η συνάρτηση Triangle κωδικοποιεί τον αλγόριθμο ΔΠ.

```
int Triangle(vector<vector<int>>& A, int N) {
    for (int i = 1; i <= N; ++i)
        dp[N][i] = A[N][i];
    for (int i = N - 1; i >= 1; --i)
        for (int j = 1; j <= i; ++j)
            dp[i][j] = max(dp[i + 1][j], dp[i + 1][j + 1]) + A[i][j];
    return dp[1][1];
}
```

Η χρονική πολυπλοκότητα της συνάρτησης Triangle είναι τάξης $\Theta(n^2)$ και η χωρική από $\Theta(n^2)$ μπορεί να βελτιωθεί σε $\Theta(n)$.

C - Vacation

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(i:,j)$, όπου $1 \leq i \leq n+1$ και $1 \leq j \leq 3$, τους μέγιστους πόντους ευτυχίας που μπορούμε να αποκομίσουμε από τις δραστηριότητες του συνόλου $\{i,\ldots,n\}$, δεδομένου ότι προηγουμένως έχουμε κάνει τη δραστηριότητα j.

- (Καθορισμός επιλογών). Δεδομένου ότι την προηγούμενη ημέρα κάναμε τη δραστηριότητα j, την τρέχουσα ημέρα μπορούμε να κάνουμε μία από τις δραστηριότητες του συνόλου $\{1, 2, 3\} - \{j\}$. Αν η τρέχουσα ημέρα είναι $n+1$, οι καλοκαιρινές διακοπές μας έχουν ολοκληρωθεί.
- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει από τις παραπάνω επιλογές.

$$S(i,1) = \max\{S(i+1,2) + b_i, S(i+1,3) + c_i\},$$
$$S(i,2) = \max\{S(i+1,1) + a_i, S(i+1,3) + c_i\},$$
$$S(i,3) = \max\{S(i+1,1) + a_i, S(i+1,2) + b_i\}.$$

  Επίσης ισχύει ότι

$$S(n+1,1) = S(n+1,2) = S(n+1,3) = 0.$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα θα επιλυθούν κατά φθίνουσα σειρά του δείκτη i.
- (Επίλυση αρχικού προβλήματος). Η λύση είναι το

$$\max\{S(2:,1) + a_1, S(2:,2) + b_1, S(2:,3) + c_1\}.$$

Η συνάρτηση Vacation κωδικοποιεί την ιδέα σε χρόνο $\Theta(n)$ και χώρο επίσης $\Theta(n)$, ο οποίος βελτιώνεται σε $\Theta(1)$.

```
int Vacation(vector<int>& A, vector<int>& B, vector<int>& C, int N) {
    dp[N + 1][1] = dp[N + 1][2] = dp[N + 1][3] = 0;
    for (int i = N; i >= 2; --i) {
        dp[i][1] = max(dp[i + 1][2] + B[i], dp[i + 1][3] + C[i]);
        dp[i][2] = max(dp[i + 1][1] + A[i], dp[i + 1][3] + C[i]);
        dp[i][3] = max(dp[i + 1][1] + A[i], dp[i + 1][2] + B[i]);
    }
    return max({dp[2][1] + A[1], dp[2][2] + A[1], dp[2][3] + C[1]});
}
```

Minimizing Coins

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(:x)$, όπου $0 \leq x \leq X$, το υποπρόβλημα εύρεσης του ελάχιστου αριθμού νομισμάτων για να σχηματίσουμε το ποσό x.
- (Καθορισμός επιλογών). Για ένα δεδομένο ποσό x, οποιοδήποτε νόμισμα, του οποίου η αξία δεν ξεπερνάει το x, αποτελεί μία επιλογή. Είναι προφανές ότι αν $x = 0$, δε απαιτείται κανένα νόμισμα για τον σχηματισμό αυτού του ποσού.

- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει από τις παραπάνω επιλογές.

$$S(x) = \begin{cases} 0, & x = 0, \\ \min_{1 \le i \le n}\{S(x - C[i]) : x \ge C[i]\} + 1, & 1 \le x \le X. \end{cases}$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα θα επιλυθούν κατά αύξουσα σειρά του x.
- (Επίλυση αρχικού προβλήματος). Η λύση είναι το $S(: X)$.

Ο αλγόριθμος ΔΠ κωδικοποιείται στη συνέχεια.

```
int MinimizingCoins(vector<int>& C, int N, int X) {
    dp[0] = 0;
    for (int x = 1; x <= X; ++x) {
        dp[x] = INF;
        for (int i = 1; i <= N; ++i)
            if (x >= C[i])
                dp[x] = min(dp[x], dp[x - C[i]] + 1);
    }
    return dp[X];
}
```

Η χρονική πολυπλοκότητα του αλγορίθμου είναι τάξης $\Theta(Xn)$ και η χωρική τάξης $\Theta(X)$.

Penguin

Θεωρούμε ότι τα ταξίδια δεικτοδοτούνται σε αύξουσα σειρά με βάση τη χρονική στιγμή που έχουν προγραμματιστεί. Έχουμε δηλαδή ότι

$$t_1 \le t_2 \le \cdots \le t_n. \tag{1}$$

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(i :)$, όπου $1 \le i \le n + 1$, το ελάχιστο κόστος για να πραγματοποιήσουμε τα ταξίδια του συνόλου $\{i, \ldots, n\}$.
- (Καθορισμός επιλογών). Οι δυνατές επιλογές είναι οι εξής. Είτε αγοράζουμε το εισιτήριο μίας χρήσης και προχωράμε στο ταξίδι $i + 1$, είτε αγοράζουμε το εισιτήριο πολλαπλών χρήσεων και μεταβαίνουμε στο **πρώτο** ταξίδι της συντομότερης ημέρας που **δεν** καλύπτει το εισιτήριο αυτό.
- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει από τις παραπάνω επιλογές.

$$S(i) = \begin{cases} 0, & i = n + 1, \\ \min\{S(i + 1) + s, S(i^* + 1) + p : i^* = \max_{i \le j \le n}\{j : t_i + m - 1 \ge t_j\}\}, & 1 \le i \le n. \end{cases}$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα προβλήματα θα επιλυθούν κατά φθίνουσα σειρά του δείκτη i.
- (Επίλυση αρχικού προβλήματος). Η λύση είναι το $S(1:)$.

Δεδομένου ότι έχουμε $\Theta(n)$ υποπροβλήματα ο αλγόριθμος που θα υλοποιήσουμε θα είναι τάξης $\Omega(n)$. Το σημείο από το οποίο εξαρτάται το πολυώνυμο που θα φράζει από τα πάνω τον αλγόριθμό μας, εξαρτάται από τον υπολογισμό του i^*, για κάθε $1 \leq i \leq n$.

Ο υπολογισμός αυτός μπορεί να υλοποιηθεί τετριμμένα σε χρόνο $O(n)$, αναζητώντας γραμμικά τη θέση i^*, το οποίο οδηγεί σε χρονική πολυπλοκότητα τάξης $O(n^2)$. Για να καταργήσουμε τη γραμμική αναζήτηση, εκμεταλλευόμαστε τη σχέση (1) και κάνουμε δυαδική αναζήτηση για να εντοπίσουμε τη θέση i^*. Η ιδέα υλοποιείται παρακάτω.

```
int FindLastTrip(vector<int>& T, int N, int M, int i) {
    int lo = i;
    int hi = N;
    while (lo < hi) {
        int j = (lo + hi + 1) / 2;
        if (T[i] + M - 1 >= T[j])
            lo = j;
        else
            hi = j - 1;
    }
    return lo;
}

int Penguin(vector<int>& T, int N, int S, int P, int M) {
    dp[N + 1] = 0;
    for (int i = N; i >= 1; --i) {
        int j = FindLastTrip(i);
        dp[i] = min(dp[i + 1] + S, dp[j + 1] + P);
    }
    return dp[1];
}
```

Η πολυπλοκότητα του παραπάνω αλγορίθμου είναι τάξης $\Theta(n \lg n)$.

Με μία προσεκτικότερη ματιά, βλέπουμε ότι

$$\max_{i \leq j \leq n} \{j : t_i + m - 1 \geq t_j\}\} \leq \max_{i' \leq j \leq n} \{j : t_{i'} + m - 1 \geq t_j\}\}, \text{ για } i \leq i'.$$

Με την παρατήρηση αυτή, έχουμε τη δυνατότητα να υπολογίσουμε την τιμή i^* σε αποσβετικό χρόνο $O(1)$ χρησιμοποιώντας την τεχνική των δύο δεικτών και πετυχαίνοντας έτσι χρονική πολυπλοκότητα τάξης $O(n)$. Η χωρική πολυπλοκότητα σε όλες τις περιπτώσεις είναι τάξης $\Theta(n)$.

```
int Penguin(vector<int>& T, int N, int S, int P, int M) {
```

```
    dp[N + 1] = 0;
    int j = N;
    for (int i = N; i >= 1; --i) {
        while (A[i] + M - 1 < A[j])
            --j;
        dp[i] = min(dp[i + 1] + S, dp[j + 1] + P);
    }
    return dp[1];
}
```

Time is Mooney

Αρχικά, πρέπει να βρούμε τον μεγαλύτερο δυνατό αριθμό μετακινήσεων (μεταξύ δύο πόλεων) έτσι ώστε το ταξίδι να είναι κερδοφόρο (ή καλύτερα ένα άνω φράγμα αυτού). Επομένως, ψάχνουμε τον μεγαλύτερο φυσικό T, για τον οποίο ισχύει ότι

$$\max_{v \in V}\{m_v\}T > CT^2 \Rightarrow \max_{v \in V}\{m_v\} > CT \Rightarrow T < \frac{\max_{v \in V}\{m_v\}}{C},$$

δηλαδή $T_{\max} = \left\lceil \frac{\max_{v \in V}\{m_v\}}{C} \right\rceil - 1$.

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(T:, v)$, όπου $0 \leq T \leq T_{\max}$ και $v \in V$, το υποπρόβλημα εύρεσης των μέγιστων χρημάτων που μπορούμε να καταλήξουμε, εκκινώντας από την κορυφή v και έχοντας διαθέσιμες $T_{\max} - T$ μετακινήσεις.
- (Καθορισμός επιλογών). Οι επιλογές που έχουμε είναι οι εξής. Αν βρισκόμαστε στη κορυφή 1 είτε σταματάμε το ταξίδι μας, είτε συνεχίζουμε και επισκεπτόμαστε κάποιες από τις γειτονικές κορυφές. Σε περίπτωση που δεν βρισκόμαστε στην κορυφή 1 πρέπει να συνεχίσουμε υποχρεωτικά το ταξίδι μας, πηγαίνοντας σε κάποιους από τις γείτονές μας. Εάν έχουμε συμπληρώσει τον μέγιστο αριθμό επιτρεπόμενων μετακινήσεων και βρισκόμαστε στην κορυφή 1, ολοκληρώνουμε υποχρεωτικά το ταξίδι. Εάν βρισκόμαστε σε κάποια από τις κορυφές του συνόλου $V - \{1\}$, το ταξίδι είναι αδύνατο, διότι έχουμε ολοκληρώσει όλες τις δυνατές μετακινήσεις, χωρίς να έχουμε επιστρέψει στην αφετηρία.
- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει από τις παραπάνω επιλογές.

$$S(T, v) = \begin{cases} -CT_{\max}^2, & T = T_{\max} \text{ και } v = 1, \\ -\infty, & T = T_{\max} \text{ και } 2 \leq v \leq n, \\ \max\{-CT^2, \max_{u \in N^+(1)}\{S(T+1, u)\}\}, & 0 \leq T \leq T_{\max} - 1 \text{ και } v = 1, \\ \max_{u \in N^+(v)}\{S(T+1, u)\} + p_v, & 0 \leq T \leq T_{\max} - 1 \text{ και } 2 \leq v \leq n. \end{cases}$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα επιλύονται κατά φθίνουσα σειρά του T.

- (Επίλυση αρχικού προβλήματος). Η λύση είναι το $S(0:,1)$.

```
int TimeIsMooney(vector<int>& P, vector<vector<int>>& Adj, int N, int C) {
    int T_max = ceil((double)*max_element(P.begin() + 1, P.begin() + N + 1)
         / C) - 1;
    dp[T_max % 2][1] = -C * T_max * T_max;
    for (int v = 2; v <= N; ++v)
        dp[T_max % 2][v] = -INF;
    for (int T = T_max - 1; T >= 0; --T) {
        dp[T % 2][1] = -C * T * T;
        for (int u : Adj[1])
               dp[T % 2][1] = max(dp[T % 2][1], dp[(T + 1) % 2][u]);
        for (int v = 2; v <= N; ++v) {
                dp[T % 2][v] = -INF;
                for (int u : Adj[v])
                      dp[T % 2][v] = max(dp[T % 2][v], dp[(T + 1) % 2][u] +
                           P[v]);
        }
    }
    return dp[0][1];
}
```

Ο παραπάνω αλγόριθμος έχει χρονική πολυπλοκότητα $\Theta(T_{\max}(n+m))$ και χωρική $\Theta(n)$, εφαρμόζοντας τη βελτιστοποίηση που περιγράφηκε σε προηγούμενο πρόβλημα.

The Cow Run

Έστω $l = \{x_i < 0 : i \in \{1, \ldots, n\}\}$ και $r = \{x_i > 0 : i \in \{1, \ldots, n\}\}$. Υποθέτουμε ότι $|l| = n_1$ και $|r| = n_2$. Θεωρούμε επίσης ότι $l_1 < l_2 < \cdots < l_{n_1}$ και $r_1 < r_2 < \cdots < r_{n_2}$.

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(: i, j :, p)$ το υποπρόβλημα εύρεσης του ελάχιστου συνολικού κόστους, για να επιστρέψουμε τις πρώτες i αγελάδες, όπου $0 \leq i \leq n_1$, με αρνητικές συντεταγμένες και τις τελευταίες $n_2 - j + 1$, όπου $1 \leq j \leq n_2 + 1$, με θετικές συντεταγμένες, έχοντας επισκεφθεί προηγουμένως αγελάδα με αρνητική ή θετική συντεταγμένη, αντίστοιχα. Δηλαδή έχουμε ότι $p \in \{0, 1\}$.

- (Καθορισμός επιλογών). Μπορούμε είτε να επιστρέψουμε την αγελάδα i, είτε την αγελάδα j. Αν $i = 0$ ή $j = n_2 + 1$, η μία από τις δύο επιλογές δεν είναι εφικτή. Αν ισχύουν αμφότερες οι ισότητες, έχουμε επιστρέψει όλες τις αγελάδες στον στάβλο.

- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει

από τις παραπάνω επιλογές.

$$S(i,j,0) = \begin{cases} 0, & i = 0 \text{ και } j = n_2 + 1, \\ S(0, j+1, 1) + (n_2 - j + 1) \cdot (r_j - l_1), & i = 0 \text{ και } 1 \le j \le n_2, \\ S(i-1, n_2+1, 0) + i \cdot (l_{i+1} - l_i), & 1 \le i < n_1 \text{ και } j = n_2 + 1, \\ \min \begin{cases} S(i-1, j, 0) + (i + n_2 - j + 1) \cdot (l_{i+1} - l_i) \\ S(i, j+1, 1) + (i + n_2 - j + 1) \cdot (r_j - l_{i+1}) \end{cases} & 1 \le i < n_1 \text{ και } 1 \le j \le n_2. \end{cases}$$

$$S(i,j,1) = \begin{cases} 0, & i = 0 \text{ και } j = n_2 + 1, \\ S(0, j+1, 1) + (n_2 - j + 1) \cdot (r_j - r_{j-1}), & i = 0 \text{ και } 2 \le j \le n_2, \\ S(i-1, n_2+1, 0) + i \cdot (r_{n_2} - l_i), & 1 \le i \le n_1 \text{ και } j = n_2 + 1, \\ \min \begin{cases} S(i-1, j, 0) + (i + n_2 - j + 1) \cdot (r_{j-1} - l_i) \\ S(i, j+1, 1) + (i + n_2 - j + 1) \cdot (r_j - r_{j-1}) \end{cases} & 1 \le i \le n_1 \text{ και } 2 \le j \le n_2. \end{cases}$$

Είναι προφανές ότι η πρώτη αναδρομική συνάρτηση ισχύει αν $n_1 \ge 1$ και η δεύτερη αν $n_2 \ge 1$.

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα επιλύονται κατά αύξουσα σειρά του δείκτη i και κατά φθίνουσα σειρά του δείκτη j.
- (Επίλυση αρχικού προβλήματος). Η λύση είναι το

$$\begin{cases} S(0,2,1) + n \cdot r_1, & n_1 = 0, \\ S(n_1 - 1, 1, 0) - n \cdot l_{n_1}, & n_2 = 0, \\ \min\{S(n_1 - 1, 1, 0) - n \cdot l_{n_1}, S(n_1, 2, 1) + n \cdot r_1\}, & \text{διαφορετικά.} \end{cases}$$

Η χρονική πολυπλοκότητα του αλγορίθμου ΔΠ είναι τάξης $\Theta(n^2)$ και η βέλτιστη χωρική είναι τάξης $\Theta(n)$. Η κωδικοποίησή του αφήνεται ως άσκηση στον αναγνώστη.

Mountain Range

Θεωρώντας ότι $h_0 = h_{n+1} = \infty$, ορίζουμε ότι

$$l_i = \max_{0 \le i^* \le i-1} \{i^* : h_{i^*} \ge h_i\} + 1,$$
$$r_i = \min_{i < i^* \le n+1} \{i^* : h_{i^*} \ge h_i\} - 1.$$

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

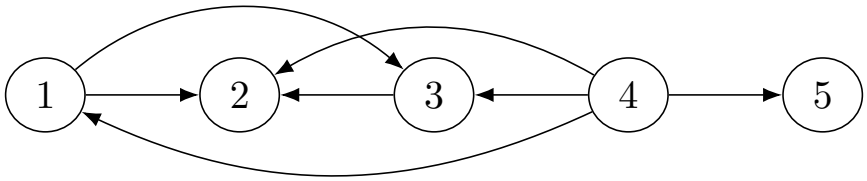
- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(i)$, όπου $1 \le i \le n$, το υποπρόβλημα εύρεσης του μέγιστου αριθμού βουνών που μπορούμε να επισκεφθούμε εκκινώντας από το βουνό i.

- (Καθορισμός επιλογών). Μπορούμε να μεταβούμε σε οποιοδήποτε βουνό του συνόλου $\{l_i, l_i + 1, \ldots, i - 1\} \cup \{i + 1, i + 2, \ldots, r_i\}$. Προσέξτε ότι αυτό το σύνολο ενδέχεται να είναι κενό.
- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει από τις παραπάνω επιλογές.

$$S(i) = 1 + \max \begin{cases} \max_{l_i \le i^* \le i-1}\{S(i^*)\} \\ \max_{i+1 \le i^* \le r_i}\{S(i^*)\} \\ 0 \end{cases}$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα επιλύονται κατά αύξουσα σειρά των τιμών των στοιχείων. Ο λόγος για αυτό φαίνεται στο γράφημα υποπροβλημάτων.
- (Επίλυση αρχικού προβλήματος). Η λύση είναι το $\max_{1 \le i \le n}\{S(i)\}$.

Η εύρεση των τιμών l_i, r_i, για κάθε $i \in \{1, 2, \ldots, n\}$, υλοποιείται τετριμμένα σε χρόνο $\Theta(n^2)$ και με *μονότονη στοίβα* σε χρόνο $O(n)$. Ο τετριμμένος υπολογισμός της παραπάνω αναδρομικής σχέσης ΔΠ απαιτεί χρόνο τάξης $O(n^2)$ και χώρο $\Theta(n)$. Ο χρόνος με τη χρήση δέντρων διαστημάτων βελτιώνεται σε $O(n \lg n)$.



Σχήμα 3. Το γράφημα υποπροβλημάτων για $H = [20, 15, 17, 35, 25]$.

Flight Discount

Ορίζουμε ως $S(v, p)$ το υποπρόβλημα εύρεσης του συντομότερου μονοπατιού από την κορυφή v, όπου $v \in V$ και $p \in \{0, 1\}$, προς την κορυφή n, χωρίς να έχουμε χρησιμοποιήσει το κουπόνι ή έχοντας χρησιμοποιήσει το κουπόνι, αντίστοιχα.

Δεδομένου ότι βρισκόμαστε στην κορυφή v, έχουμε τη δυνατότητα να επισκεφθούμε οποιαδήποτε κορυφή $u \in N^+(v)$. Σε περίπτωση που δεν έχουμε χρησιμοποιήσει το κουπόνι, είτε θα το χρησιμοποιήσουμε και θα πληρώσουμε το μειωμένο βάρος, είτε δεν θα το χρησιμοποιήσουμε και θα χρεωθούμε το κανονικό βάρος. Αν το έχουμε χρησιμοποιήσει ήδη, η πρώτη από τις προηγούμενες επιλογές δεν είναι εφικτή. Αν $v = n$ έχουμε φτάσει στον προορισμό μας.

Με βάση τις παραπάνω επιλογές έχουμε ότι

$$S(v,0) = \min_{u \in N^+(v)} \begin{cases} S(u,0) + w(v,u) \\ S(u,1) + \left\lfloor \frac{w(v,u)}{2} \right\rfloor \end{cases}$$

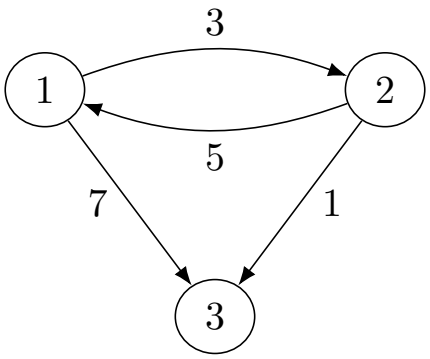
$$S(v,1) = \min_{u \in N^+(v)} \{S(u,1) + w(v,u)\}$$

για $v \neq n$ και

$$S(v,0) = S(v,1) = 0$$

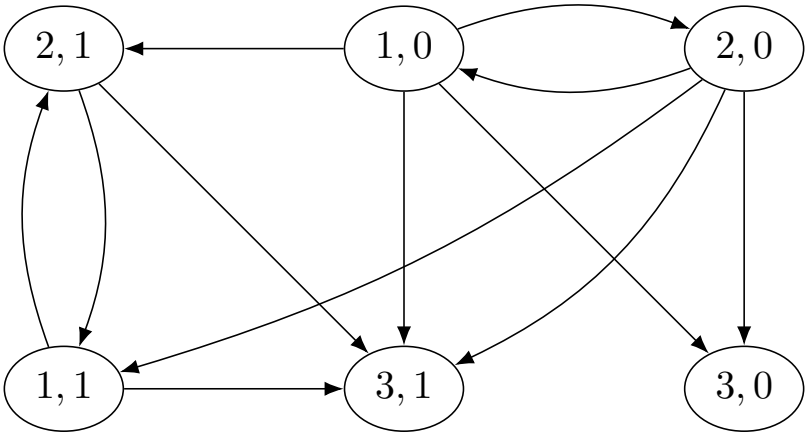
για $v = n$.

Ας θεωρήσουμε ως είσοδο το γράφημα του Σχήματος 4.



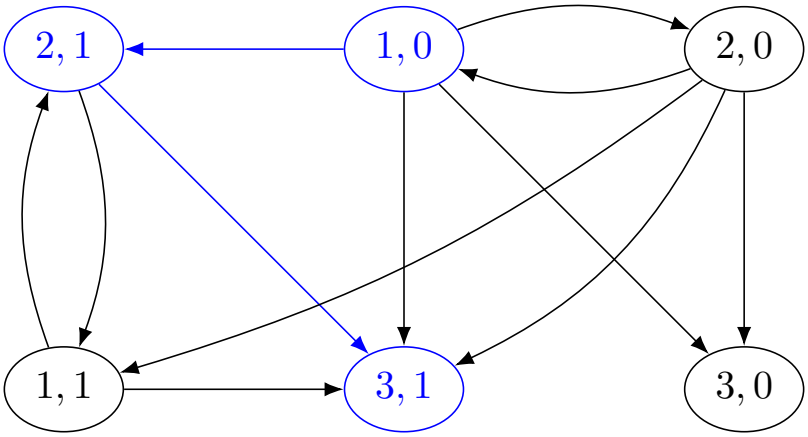
Σχήμα 4. Κατευθυνόμενο γράφημα.

Παρατηρούμε ότι το γράφημα των υποπροβλημάτων, που παρουσιάζεται στο σχήμα που ακολουθεί, περιέχει κύκλους. Αυτό καθιστά αδύνατη την εφαρμογή ενός αλγορίθμου ΔΠ, διότι δεν γνωρίζουμε τη σειρά με την οποία πρέπει να επιλυθούν τα υποπροβλήματα.



Σχήμα 5. Το γράφημα υποπροβλημάτων που προκύπτει από την αναδρομική σχέση ΔΠ, με είσοδο το γράφημα του Σχήματος 4.

Ας προσεγγίσουμε, λοιπόν, το πρόβλημα λίγο διαφορετικά. Δεδομένου του γραφήματος των υποπροβλημάτων του Σχήματος 5, καλούμαστε να βρούμε ένα μονοπάτι ελάχιστου μήκους, το οποίο εκκινεί από την κορυφή $(1, 0)$ και καταλήγει στην κορυφή $(3, 1)$. Δεδομένου ότι τα βάρη των ακμών του γραφήματος είναι μη αρνητικά, μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο του Dijkstra, τον οποίο δεν επηρεάζουν οι κύκλοι.



Σχήμα 6. Το μονοπάτι που υπολογίζει ο αλγόριθμος του Dijkstra, συνολικού βάρους 2, χρωματισμένο με μπλε χρώμα.

Distinct Values Splits

Αρχικά, για κάθε $i \in \{1, 2, \ldots, n\}$, ορίζουμε

$$r_i = \max_{i \le j \le n} \{j : x_a \neq x_b,\ i \le a < b \le j\}.$$

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(i :)$, όπου $1 \le i \le n + 1$, το υποπρόβλημα εύρεσης των τρόπων με τους οποίους μπορούμε να χωρίσουμε τα στοιχεία του συνόλου $\{i, i + 1, \ldots, n\}$ σε συνεχόμενα διαστήματα διακριτών στοιχείων.
- (Καθορισμός επιλογών). Ξεκινώντας από τη θέση i μπορούμε να σταματήσουμε σε οποιαδήποτε θέση $i^* \in \{i, i + 1, \ldots, r_i\}$ και να επιλύσουμε το υποπρόβλημα $S(i^* + 1)$. Αν βρισκόμαστε στη θέση $n + 1$ η απάντηση είναι τετριμμένη.

- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει από τις παραπάνω επιλογές.

$$S(i) = \begin{cases} 1, & i = n+1, \\ \sum_{i^*=i}^{r_i} S(i^*+1), & 1 \le i \le n. \end{cases}$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα επιλύονται κατά φθίνουσα σειρά του δείκτη i.

- (Επίλυση αρχικού προβλήματος). Η λύση είναι το $S(1:)$.

Παρατηρώντας ότι

$$\sum_{i^*=i}^{r_i} S(i^*+1) = \sum_{i^*=i+1}^{r_i+1} S(i^*),$$

μπορούμε να υπολογίσουμε τη σχέση ΔΠ σε χρόνο και χώρο $\Theta(n)$ χρησιμοποιώντας μερικά αθροίσματα. Ο υπολογισμός της τιμής r_i επιτυγχάνεται επίσης σε χρόνο γραμμικό στο n με την τεχνική των δύο δεικτών.

Gamers

Θεωρούμε ότι τα επίπεδα ικανότητας των παικτών είναι διατεταγμένα σε αύξουσα σειρά. Έχουμε δηλαδή ότι

$$p_1 \le p_2 \le \cdots \le p_n. \tag{2}$$

Για τον λόγο του ότι μας ζητείται να **ελαχιστοποιήσουμε τη μέγιστη διαφορά** θα χρησιμοποιήσουμε δυαδική αναζήτηση. Πιο συγκεκριμένα, θα επιλέξουμε μία συγκεκριμένη μέγιστη διαφορά D και θα ελέγξουμε αν είναι δυνατό να πετύχουμε αυτή τη μέγιστη διαφορά, χρησιμοποιώντας ΔΠ.

Ορίζουμε για κάθε $i \in \{1, 2, \ldots, n\}$ ότι

$$r_i = \min\{\max_{i \le j \le n} \{j : p_j - p_i \le D\},\ i + y - 1\}.$$

Τα πέντε στάδια του ΔΠ εξειδικεύονται ως εξής.

- (Ορισμός υποπροβλημάτων). Ορίζουμε ως $S(i:)$, όπου $1 \le i \le n+1$, το υποπρόβλημα εύρεσης δυνατότητας να χωρίσουμε τα στοιχεία του συνόλου $\{i, i+1, \ldots, n\}$ σε συνεχόμενα διαστήματα, έτσι ώστε σε καθένα από αυτά τα διαστήματα η μέγιστη διαφορά μεταξύ δύο στοιχείων να είναι D.

- (Καθορισμός επιλογών). Εκκινώντας από τη θέση i, έχουμε τη δυνατότητα να «κόψουμε» τον υποπίνακα σε οποιαδήποτε θέση $i^* \in \{i+x-1, i+x, \ldots, r_i\}$ και να επιλύσουμε το υποπρόβλημα $S(i^*+1)$. Προσέξτε ότι το σύνολο αυτό ενδέχεται να είναι κενό, το οποίο σημαίνει ότι $S(i) =$ False. Αν $i = n+1$, η απάντηση είναι τετριμμένη.

- (Ορισμός αναδρομικής σχέσης). Ακολουθεί η αναδρομική σχέση που προκύπτει από τις παραπάνω επιλογές.

$$S(i) = \begin{cases} \text{True}, & i = n+1, \\ \bigvee_{i^*=i+x-1}^{r_i} S(i^*+1), & 1 \leq i \leq n. \end{cases}$$

- (Σειρά επίλυσης υποπροβλημάτων). Τα υποπροβλήματα επιλύονται κατά φθίνουσα σειρά των τιμών του δείκτη i.

- (Επίλυση αρχικού προβλήματος). Η λύση είναι το $S(1:)$.

Η τιμή $S(1:)$ υπολογίζεται με τρόπο αντίστοιχο του προηγούμενου προβλήματος σε χρόνο και χώρο $\Theta(n)$. Η τιμή r_i, δεδομένου ότι ισχύει η (2), υπολογίζεται για κάθε i σε αποσβεστικό χρόνο $O(1)$. Για τον λόγο του ότι ο αλγόριθμος ΔΠ είναι ενσωματωμένος στο εσωτερικό της δυαδικής αναζήτησης, ο συνολικός χρόνος είναι ίσος με $O(n \lg D_{\max})$, όπου $D_{\max} = p_n - p_1$.