طراحی الگوریتم ها

نیم‌سال دوم ۹۸ـ۹۹

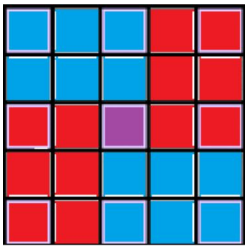
دانشکده‌ی مهندسی کامپیوتر گردآورندگان: کیارش گل زاده ، محمد مهدوی ، مجید گروسی ، آراد محمدی

پاسخ نامه تمرین عملی سری دوم برنامه نویسی پویا

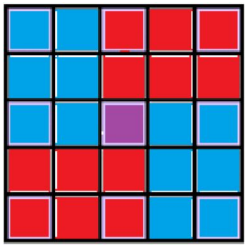
مسئله‌ی ۱. ادن و ویلیان

روی خانهای که دو نفر به صورت مشترک در آن قدم م یگذارند، حالتبندی میکنیم. این خانه را در نظر بگیرید؛ دو بازیکن نمیتوانند از خانهی سمت چپِ این خانه واردش شوند)در این صورت، مسیرهای آن دو در حداقل دو خانه اشتراک دارد(و همچنین، حالتی که یک نفر از خانهی بالایی و نفر دیگر از خانهی پایینی وارد آن شود نیز قابل قبول نیست)زیرا در حرکت بعدی، هر دو از خانهی سمت راست این خانه خارج میشوند و دوباره حداقل دو خانهی مشترک در مسیر های آنها وجود دارد(. پس تنها حالت های مورد قبول، حالت های زیر است :

۱ . ادن هازارد از خانه ی بالایی وارد و از خانهی پایینی خارج، و ویلیان از خانهی سمت چپ وارد و از خانهی سمت راست خارج شود.



۲ . ادن هازارد از خانهی سمت چپ وارد و از خانهی سمت راست خارج، و ویلیان از خانهی پایینی وارد و از خانه ی بالایی خارج شود.



حال با حالتبندی روی خانه ی مشترک و محاسبهی بیشترین پولی که میتوان با گذر از آن خانهی مشترک به دست آورد، میتوان پاسخ مسئله را پیدا کرد. این کار را به کمک ۴ آرایهی dp انجام میدهیم.

۱ . dp۱۱[i][j] : بیشترین پولی که میتوان با رفتن از خانه ی (۱ ، ۱) به خانه ی (i , j) با استفاده از حرکت های راست و پایین به دست آورد .

۲ . dpnm[i][j] : بیشترین پولی که میتوان با رفتن از خانه ی (i , j) به خانه ی (n , m) با استفاده از حرکت های راست و پایین به دست آورد.

۳. dpn۱[i][j] : بیشترین پولی که میتوان با رفتن از خانه ی (n , 1) به خانه ی (i , j) با استفاده از حرکت های راست و بالا به دست آورد .

۴ . dp۱m[i][j] : بیشترین پولی که میتوان با رفتن از خانه ی (i , j) به خانه ی (1 , m) با استفاده از حرکت های راست و بالا به دست آورد.

با پر کردن این ۴ آرایه، میتوانیم به ازای یک خانه ی مشترکِ مشخص، برای دو حالتِ عبور از آن، بیشترین پول به دست آمده را محاسبه کرده، و بین آن دو حالت ماکسیمم بگیریم. (به ازای تمام خانه های جدول به جز خانه های حاشیه ای که نمیتوانند خانه ی مشترک باشند(چرا؟)) این مقدار را محاسبه کرده و میان تمام آنها ماکسیمم میگیریم تا جواب نهایی مشخص شود.

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  constexpr int maxn = 2000;
4  long long ar[maxn + 1][maxn + 1], dp11[maxn + 1][maxn + 1], dpnm[maxn + 1
5
6  int n, m;
7
8  void init()
9  {
10     dp11[1][1] = ar[1][1];
11     dpnm[n][m] = ar[n][m];
12     dpn1[n][1] = ar[n][1];
13     dp1m[1][m] = ar[1][m];
14     int i, j;
15     for (int i = n; i >= 1; i--)
16         for (int j = m; j >= 1; j--)
17             if (i != n || j != m) dpnm[i][j] =
18             max(dpnm[i + 1][j], dpnm[i][j + 1]) + ar[i][j];
19
20     for (int i = 1; i <= n; i++)
21         for (int j = 1; j <= m; j++)
22             if (i != 1 || j != 1) dp11[i][j] =
23             max(dp11[i - 1][j], dp11[i][j - 1]) + ar[i][j];
24
25     for (int i = 1; i <= n; i++)
26         for (int j = m; j >= 1; j--)
27             if (i != 1 || j != m) dp1m[i][j] =
28             max(dp1m[i - 1][j], dp1m[i][j + 1]) + ar[i][j];
29
30     for (int i = n; i >= 1; i--)
31         for (int j = 1; j <= m; j++)
```

```
32                 if (i != n || j != 1) dpn1[i][j] =
33                 max(dpn1[i + 1][j], dpn1[i][j - 1]) + ar[i][j];
34 }
35
36 int main() {
37     ios_base::sync_with_stdio(false);
38     cin.tie(NULL);
39     cin >> n >> m;
40
41     for (int i = 1; i <= n; i++)
42         for (int j = 1; j <= m; j++)
43             cin >> ar[i][j];
44     init();
45
46     long long ans = 0, f = 0;
47     for(int i = 2; i < n; i++)
48         for(int j = 2; j < m; j++)
49         {
50             ans = max(ans,dp11[i - 1][j] + dpnm[i + 1][j] +
51                                 dpn1[i][j - 1]+dp1m[i][j + 1] + ar[i][j])
52             ans = max(ans,dp1m[i - 1][j] + dpn1[i + 1][j] +
53                                 dp11[i][j - 1]+dpnm[i][j + 1] + ar[i][j])
54         }
55     cout << ans << endl;
56     return 0;
57 }
```

مسئله‌ی ۲. انتون و پشمک

می‌توان دید که اگر جعبه‌هایی که می‌خریم یکی از شرایط زیر را داشته باشند حتما k پشمک از یک طعم خواهیم داشت:

$$\sum max(a_i - ۱, ۰) \geqslant k$$

$$\sum max(b_i - ۱, ۰) \geqslant k$$

$$\sum a_i + b_i \geqslant ۲k - ۱$$

در دو حالت اول از یک طعم به وضوح k تا داریم و در حالت بعدی نیز حداقل سقف نصف پشمک‌ها از یکی از دو طعم است پس k پشمک از یک طعم خواهیم داشت. همچنین اگر هیچ یک از شرط‌ها برقرار نباشند حالتی هست که از هر دو طعم کمتر از k عدد داشته باشیم. پس می‌توانیم برای هر شرط به صورت جداگانه چک کنیم کم‌هزینه‌ترین حالتی که این شرط برقرار شود چقدر است. با این فرض مسئله به سه مسئله به این صورت تبدیل می‌شود: n شی داریم که

شی iم هزینهc_i و ارزش v_i دارد و می‌خواهیم کم‌هزینه‌ترین حالتی را پیدا کنیم که جمع ارزش‌های آن حداقل x شود که این مسئله برای هر شرط ارزش‌ها در سمت چپ نامساوی اند و حداقل ارزشی که می‌خواهیم در راست آن. این مسئله، مسئله کلاسیک کوله‌پشتی است که با رابطه

$$dp[i][j] = min(dp[i-۱][j], dp[i-۱][max(۰, j-v_i)] + c_i)$$

و شرط اولیه

$$dp[۰][۰] = ۰, dp[۰][j۰] = \infty$$

به دست می‌آید که$dp[i][j]$ کمترین هزینه برای یافتن ارزش حداقل j با i عنصر اول را نشان می‌دهد. همچنین برای استفاده از حافظه اول می‌توان با توجه به اینکه تنها از ردیف قبلی استفادهdp می‌شود و تنها جواب$dp[n][x]$فقط دو سطر آخر dp را ذخیره کنیم و حتی با توجه به اینکه از های j کمتر استفاده می‌کنیم با پیمایش از j بزرگ به کوچک تنها به اندازه یک سطر حافظه بگیریم و n بار از x به ۰

$$dp[j] = min(dp[j], dp[max(۰, j-v_i)] + c_i)$$

را اجرا کنیم.

```
#include<bits/stdc++.h>
using namespace std;
const int MK = 1e4 + 100,MN = 50 + 10;
int dp[2*MK];
int n;
int a[MN],b[MN],c[MN],v[MN];
int answer(int k){
    memset(dp,-1,sizeof dp);
    dp[0]=0;
    for(int i=0;i<n;i++){
        for(int j=k;j>=0;j--){
            int j1 = max(0,j-v[i]);
            if (dp[j1]!=-1 && (dp[j] == -1 || dp[j1] + c[i] <= dp[j])){
                dp[j] = dp[j1] + c[i];
            }
        }
    }
    return dp[k];
}
int main(){
    int k;
    cin >> n >> k;
    for(int i=0;i<n;i++){
        cin>>a[i];
    }
    for(int i=0;i<n;i++){
        cin>>b[i];
    }
    for(int i=0;i<n;i++){
```

```
30          cin>>c[i];
31      }
32      for (int i=0;i<n;i++){
33          v[i] = max(0,a[i]-1);
34      }
35      int ans = -1;
36      int tmp = answer(k);
37      if (ans == -1 || (ans >= tmp && tmp!=-1)){
38          ans = tmp;
39      }
40      for (int i=0;i<n;i++){
41          v[i] = max(0,b[i]-1);
42      }
43      tmp = answer(k);
44      if (ans == -1 || (ans >= tmp && tmp!=-1) ){
45          ans = tmp;
46      }
47      for (int i=0;i<n;i++){
48          v[i] = max(0,a[i]+b[i]);
49      }
50      tmp = answer(2*k-1);
51      if (ans == -1 || (ans >= tmp && tmp != -1)){
52          ans = tmp;
53      }
54      cout << ans;
55      return 0;
56  }
```

مسئله‌ی ۳. پرسه با کیانوش

فکت: اگر از برج s شروع کنیم و به برج t بریم(می‌شه فرض کرد s > t) به ازای هر دیواری بین s و t اون دیوار رو به اندازه‌ی بزرگترین عدد فرد کوچک‌تر مساوی ظرفیتش می‌تونیم طی کنیم. همچنین در مورد پل‌های قبل و بعد از s و t هم می‌تونیم همین حرف رو بزنیم. (تا جایی که ظرفیت دیوار ۱ نباشه و به اندازه‌ی بزرگترین عدد زوج کم‌تر مساوی ظرفیت هر دیوار)
حالا مقدار زیر را در نظر می گیریم dp[i] یعنی از برج i شروع کنیم از بین دیوارای قبلش حداکثر چه مسافتی می‌تونیم بریم پس رابطه زیر برای dp برقرار است:

$$dp[۰] = ۰$$

$$a[i-۱]! = ۱- > dp[i] = dp[i-۱] + maxeven(a[i-۱])$$

$$a[i-۱] == ۱- > dp[i] = ۰$$

pd[i] را هم به طور مشابه تعریف می کنیم. تو راس i اگر قرار باشه تموم کنیم، حداکثر چه مسافتی از بین دیوارای بعدش می‌تونیم طی کنیم و با رابطه مشابه ، آن را تعریف می کنیم.

until[i] یعنی اگر از برجی کوچکتر مساوی i شروع کرده باشیم حداکثر چه مسافتی می‌تونستیم طی کنیم از بین پل‌های قبل از i که برسیم به i که با رابطه زیر تعریف میشه

$$until[i] = max(dp[i], until[i - ۱] + maxeven(a[i - ۱]))$$

در نهایت مقدار زیر برابر جواب نهایی است.

$$max(until[i] + pd[i])$$

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long int LL;

#define smax(x, y) (x) = max((x), (y))

const int maxn = 1e5 + 10;
LL a[maxn],
   dp[maxn],
   pd[maxn],
   until[maxn],
   n;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    cin >> n;
    for (int i = 1; i < n; i++)
        cin >> a[i];
    dp[0] = 0;
    for (int i = 1; i < n; i++)
        dp[i] = (a[i] == 1) ? 0 : dp[i - 1] + a[i] - (a[i] & 1LL);
    pd[n - 1] = 0;
    for (int i = n - 1; i > 0; i--)
        pd[i - 1] = (a[i] == 1) ? 0 : pd[i] + a[i] - (a[i] & 1LL);
    LL ans = 0;
    for (int i = 0; i < n; i++) {
        if (i == 0)
            until[i] = dp[i];
        else
            until[i] = max(until[i - 1] + ((a[i] & 1LL) ?
                                a[i] : a[i - 1]), dp[i]);
        smax(ans, until[i] + pd[i]);
    }
    cout << ans << endl;
```

```
38      return 0;
39  }
```

موفق باشید :)