**Instituto**

**Politécnico**

**Nacional**

**Escuela Superior de Cómputo**

**Ejercicios teóricos**

**Materia:**

Análisis de algoritmos

**Grupo:**

3CM3

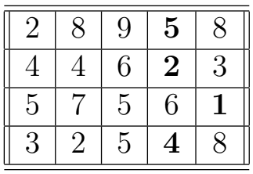
**Integrantes:**

Castro Cruces Jorge Eduardo

**Fecha:**

Lunes, julio 6, 2020

**Ejercicio 1**

Imagina a un escalador que trepa por una pared. La pared está construida con bloques cuadrados del mismo tamaño, cada uno de los cuales tiene un agarre. Algunos de los agarres instalados son más peligrosos o complicados que otros. Estando en un bloque, el escalador solo puede moverse hacia una de las siguientes opciones: arriba a la derecha, arriba a la izquierda o justo arriba del bloque, en el que se está (a menos que el escalador se encuentre en las orillas de la pared). La meta es encontrar la ruta menos peligrosa de abajo hacia arriba de la pared. El nivel de peligro de una ruta está dado por la suma del nivel de peligro de cada uno de los agarres usado en la ruta. Por ejemplo, en la siguiente figura, si el escalador está parado hasta abajo en el bloque marcado con un 2, para subir, puede elegir, en bloque inmediato superior, uno de los bloques marcados con 5, 7 o 5. Para este muro, la ruta menos peligrosa es la que aparece marcada en negritas.

¿Cuál sería la función de optimización para este problema?

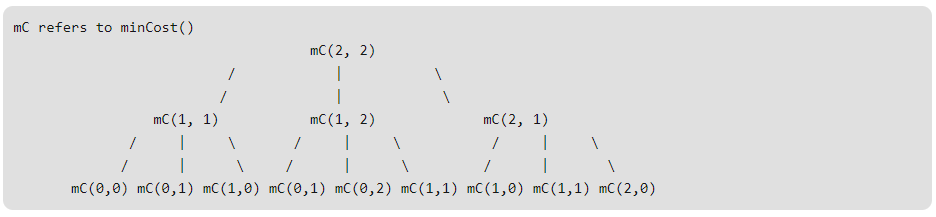
1. #include <bits/stdc++.h>
2. #define M 100
3. #define N 100
4. **using** **namespace** std;
6. **int** find\_min\_odd\_cost(**int** given[M][N], **int** m, **int** n) {
7. **int** floor[M][N] = {{ 0 }, { 0 }};
8. **int** min\_odd\_cost = 0;
9. **int** i, j, temp;
11. **for** (j = 0; j < n; j++) floor[0][j] = given[0][j];
13. **for** (i = 1; i < m; i++)
14. **for** (j = 0; j < n; j++){
15. **if** (j == 0){
16. floor[i][j] = given[i][j];
17. floor[i][j] += min(floor[i - 1][j], floor[i - 1][j + 1]);
18. }
19. **else** **if** (j == n - 1) {
20. floor[i][j] = given[i][j];
21. floor[i][j] += min(floor[i - 1][j], floor[i - 1][j - 1]);
22. }
23. **else** {
24. temp = min(floor[i - 1][j], floor[i - 1][j - 1]);
25. temp = min(temp, floor[i - 1][j + 1]);
26. floor[i][j] = given[i][j] + temp;
27. }
28. }
29. min\_odd\_cost = INT\_MAX;
30. **for** (j = 0; j < n; j++){
31. **if** (floor[n - 1][j] % 2 == 1){
32. **if** (min\_odd\_cost > floor[n - 1][j]) min\_odd\_cost = floor[n - 1][j];
33. }
34. }
35. **if** (min\_odd\_cost == INT\_MIN) **return** -1;
36. **return** min\_odd\_cost;
37. }
39. **int** main(){
40. **int** m = 5, n = 5;
41. **int** given[M][N] =  {{ 1, 2, 3, 4, 6 },
42. { 1, 2, 3, 4, 5 },
43. { 1, 2, 3, 4, 5 },
44. { 1, 2, 3, 4, 5 },
45. { 100, 2, 3, 4, 5 }};
47. cout << "Minimum odd cost is " << find\_min\_odd\_cost(given, m, n);
48. **return** 0;
49. }

Escribe el pseudocódigo de una solución recursiva que utilice programación dinámica para este problema.

1. #include<stdio.h>
2. #include<limits.h>
3. #define R 3
4. #define C 3
6. **int** min(**int** x, **int** y, **int** z);
8. **int** minCost(**int** cost[R][C], **int** m, **int** n){
9. **if** (n < 0 || m < 0) **return** INT\_MAX;
10. **else** **if** (m == 0 && n == 0) **return** cost[m][n];
11. **else** **return** cost[m][n] + min( minCost(cost, m-1, n-1),
12. minCost(cost, m-1, n),
13. minCost(cost, m, n-1) );
14. }
16. **int** min(**int** x, **int** y, **int** z){
17. **if** (x < y) **return** (x < z)? x : z;
18. **else** **return** (y < z)? y : z;
19. }
21. **int** main(){
22. **int** cost[R][C] =   {{1, 2, 3},
23. {4, 8, 2},
24. {1, 5, 3} };
25. printf(" %d ", minCost(cost, 2, 2));
26. **return** 0;
27. }

¿Cuál es la complejidad de esta solución?

Cabe señalar que la función anterior calcula los mismos subproblemas una y otra vez. Vea el siguiente árbol de recursión, hay muchos nodos que aparecen más de una vez. La complejidad temporal de esta ingenua solución recursiva es exponencial y es terriblemente lenta.



**Ejercicio 2**

Imagina que vas a iniciar un largo viaje por carretera. Supón que tu viaje comienza en el kilómetro 0, y que a lo largo del camino que vas a tomar hay n hoteles, con distancias d1 < d2 < · · · < dn, donde cada di se mide a partir del kilómetro 0. Los únicos lugares en donde está permitido detenerse es en los hoteles, pero es posible elegir en qué hoteles detenerse. El viaje termina en el hotel a la distancia dn. Tu plan es viajar 200 kilómetros durante el día, pero esto puede no ocurrir (dependiendo de qué tan lejos esté un hotel de otro). Si viajas x kilómetros durante el día, la penalización por día es (200 − x)^2. El objetivo es minimizar la penalización total, es decir, la suma de las penalizaciones de cada día. Diseña un algoritmo que utilice programación dinámica, para determinar la secuencia de hoteles con penalización total mínima. Escribe el pseudocódigo correspondiente.

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
4. **int** maxRevenue(**int** m, **int** x[], **int** revenue[], **int** n, **int** t){
5. **int** maxRev[m+1];
6. memset(maxRev, 0, **sizeof**(maxRev));
7. **int** nxtbb = 0;
8. **for** (**int** i = 1; i <= m; i++){
9. **if** (nxtbb < n){
10. **if** (x[nxtbb] != i) maxRev[i] = maxRev[i-1];
11. **else**{
12. **if** (i <= t) maxRev[i] = max(maxRev[i-1], revenue[nxtbb]);
13. **else** maxRev[i] = max(maxRev[i-t-1]+revenue[nxtbb], maxRev[i-1]);
14. nxtbb++;
15. }
16. }
17. **else** maxRev[i] = maxRev[i - 1];
18. }
19. **return** maxRev[m];
20. }
22. **int** main(){
23. **int** m = 20;
24. **int** x[] = {6, 7, 12, 13, 14};
25. **int** revenue[] = {5, 6, 5, 3, 1};
26. **int** n = **sizeof**(x)/**sizeof**(x[0]);
27. **int** t = 5;
28. cout << maxRevenue(m, x, revenue, n, t) << endl;
29. **return** 0;
30. }