Задача за разделянето

Научен екип трескаво търси информация по даден проблем. За целта трябва да се претърси дълъг библиотечен рафт, съдържащ n наредени една след друга научни книги, всяка от които съдържа s_i ($1 \le i \le n$) страници. С претърсването се е заела група от k на научни сътрудника. Наредбата на книгите в библиотеката е изключително важна и не бива да се нарушава. За да се избегнат евентуални проблемите в това отношение, сътрудниците решават да си поделят рафта на области, без да разместват книгите. Тъй като времето за приключване на проекта е изключително критично, групата решава да си подели рафта по такъв начин, че претърсването да приключи за минимално време. Всички членове на групата са опитни научни сътрудници и преглеждането на една страница им отнема едно и също фиксирано време. за целта екипът преглежда последователно книгите и записва на лист хартия броя на страниците за всяка от тях и получават редица от числа $s_1,s_2,...,s_n$. Така задачата им се свежда до подходящо разбиване на тази редица на k подпоследователности, така че максимумът на сумите от броя на страниците във всяка от групите да бъде минимален.

Веднага може да бъде посочен един възможен евристичен похват: намираме средната стойност $\frac{1}{k}\sum_{i=1}^{n} s_i$ на сумата за всяка група, след което се опитваме да поставим разделителите така, че максимално да се приближим до нея. За съжаление пруи някои входни данни този подход няма да даде оптимален резултат, тъй като не изследва систематично всички възможности.

Да видим как можем да решим задачата с помощта на динамично оптимиране. Ще разсъждаваме по следния начин: Да разделим изходната последователност $s_1,s_2,...,s_n$ на две подпоследователности на позиция i ($1 \le i \le n$): получаваме $s_1,s_2,...,s_i$ и $s_{i+1},s_{i+2},...,s_n$. Втората последователност ще считаме за окончателно разделена и ще я зачислим на един от научните сътрудници, а първата ще разделим на k–1 части. За целта в нея трябва да бъдат поставени k–2 разделителя. За лявата част можем да разсъждаваме по същия начин: поставяме разделител на някоя позиция j ($1 \le j \le i$). Едната част ($s_{j+1},s_{j+2},...,s_i$) считаме за окончателно разделена, докато в другата ($s_1,s_2,...,s_j$) следва да се поставят още k–3 разделителя. Така успяхме да разделим задачата на две подзадачи, първата от които се решава директно, а втората е задача с по-малка размерност. Оттук директно получаваме съответно рекурсивно решение.

Ще се опитаме да обобщим и формализираме горните разсъждения. Да въведем двуаргументна целева функция $f(n,k)$, която ще ни дава минималната цена по всевъзможните разделяния на книгите в k групи. Цената ще дефинираме като максимум по групите на сумата от елементите в групата. Или формално:

$$f(i,j) = \min_{l=1}^{i} \max\left(f(l, j-1), \sum_{r=l+1}^{i} s_r \right),\ 1 \le i \le n,\ 1 \le j \le k$$

Остава да се наложат подходящи гранични условия по двата аргумента, а именно:

$$f(1,j) = s_1,\ 1 \le j \le k$$
$$f(i,1) = s_1 + s_2 + ... + s_i,\ 1 \le i \le n.$$

Сега вече сме готови да реализираме съответно итеративна програма. Преди да пристъпим към това, нека се опитаме да оценим времевата сложност

на алгоритъма. Основна характеристика на динамичното програмиране е, че $f(i,j)$ се пресмята точно веднъж за всяка възможна стойност на аргументите i ($1 \le i \le n$) и j ($1 \le j \le k$). Броят на различните комбинации от тези стойности е nm. Остава да оценим времето за пресмятане стойността на функцията за една фиксирана двойка стойности (i,j), при предположението, че всички стойности на функцията, за по-малки стойности на аргументите вече са били пресметнати. Тогава от формулата по-горе се вижда, че времето за пресмятане се определя от времето за намиране на сумата $s_{l+1} + s_{l+2} + ... + s_i$. На пръв поглед изглежда, че за това ще бъде необходимо време от порядъка на $O(n^2)$. В действителност при използване на кумулативен масив, съдържащ всевъзможните префиксни суми на елементите на s_i, можем да пресметнем тази сума директно. Така максимумът се пресмята за време $O(1)$. Вземайки предвид минимума от формулата, получаваме, че пресмятането на една конкретна стойност отнема време $O(n)$. Окончателно за решаването на задачата получаваме $O(kn^2)$.

Целевата функция $f(i,j)$ ще представим като двумерен масив. Ще въведем и втори масив b[][], който ще съдържа индексите, за които се е получила съответната оптимална стойност на функцията. Това ще ни позволи да възстановим едно възможно оптимално решение по добре познатия ни вече начин. Подробностите по реализацията се виждат от кода на програмата.

```
#include <stdio.h>
#include <values.h>
#define MAX 100
#define max(a,b) ((a > b) ? a : b)  // Връща по-големия аргумент

int s[MAX] = { 0,23,15,89,170,25,1,86,80,2,27 }; // Редица (първият не се ползва)
unsigned n = 10;  // Брой елементи в редицата
unsigned k = 4;   // Брой групи
long p[MAX];      // Префиксни суми
long f[MAX][MAX]; // Целева функция
long b[MAX][MAX]; // За възстановяване на решението

long DoPartition(unsigned k) { // Извършва оптимално разделяне на k групи
int i,j,l;
  // Пресмятаме префиксните суми
  p[0] = 0;
  for (i=1; i<=n; i++)
    p[i] = p[i-1] + s[i];

  // Установяване на граничните условия
  for (i=1; i<=n; i++) f[i][1] = p[i];
  for (i=1; i<=k; i++) f[1][j] = s[1];

  // Основен цикъл
  for (i=2; i<=n; i++)
    for (j=2; j<=k; j++) {
      f[i][j] = MAXINT;
      for (l=1; l<=i-1; l++) {
        int m = max(f[l][j-1],p[i]-p[l]);
        if (f[i][j] > m) {
          f[i][j] = m;
          b[i][j] = l;
        }
      }
    }
  return f[n][k];
}

void Print(unsigned from, unsigned to) {
  printf("\n");
```

```
  for (int i=from; i<=to; i++)
    printf("%d ",s[i]);
}

void PrintPartition(unsigned n, unsigned k) {
  if (k == 1)
    Print(1,n);
  else {
    PrintPartition(b[n][k],k-1);
    Print(b[n][k]+1,n);
  }
}

int main() {
  printf("\nМаксимална сума в някоя от групите: %ld",DoPartition(k));
  PrintPartition(n,k);
  return 0;
}
```

partitio.cpp

Резултат от изпълнението на горната програма:

```
Максимална сума в някоя от групите: 170
23 15 89
170
25 1 86
80 2 27
```