Soluţie borg

Numărul total de moduri în care se pot plasa K motoare în paralelipiped, fără restricţia ca fiecare plan să conţină cel puţin un motor, este C^K_{N*M*H}. Din acestea, trebuie să le scădem pe cele care nu sunt bune.
Dacă notăm cu X_i ($1 \le i \le N$) numărul de moduri în care se pot plasa cele K motoare astfel încât planul `i` (de tip 1) să fie gol, cu Y_i ($1 \le i \le M$) numărul de moduri astfel încât planul `i` (de tip 2) să fie gol şi cu Z_i ($1 \le i \le H$) numărul de moduri astfel încât planul `i` (de tip 3) să fie gol, atunci numărul căutat va fi $C^K_{N*M*H} - \left| \bigcup_{1 \le i \le N} X_i \cup \bigcup_{1 \le i \le M} Y_i \cup \bigcup_{1 \le i \le H} Z_i \right|$. Cardinalul reuniunii se poate dezvolta cu principiul includerii şi excluderii, fiecare mulţime rezultată conţinând o intersecţie de `a` mulţimi X, `b` mulţimi Y şi `c` mulţimi Z. Cardinalul unei astfel de mulţimi ar reprezenta numărul de moduri în care se pot pune K motoare, astfel încât `a` dintre planele de tip 1 să fie goale, `b` dintre planele de tip 2 să fie goale şi `c` dintre planele de tip 3 să fie goale, adică ar fi $C^K_{(N-a)*(M-b)*(H-c)}$. În dezvoltarea produsă de principiul includerii şi excluderii vor fi $C^a_N * C^b_M * C^c_H$ astfel de mulţimi.
Aşadar numărul total va fi $\sum_{a=1}^{N} \sum_{b=1}^{M} \sum_{c=1}^{H} (-1)^{a+b+c} * C^a_N * C^b_M * C^c_H * C^K_{(N-a)*(M-b)*(H-c)}$.
Se observă că este nevoie de toate combinările până la maximul dintre N, M şi H, dar numai de cele care au K sus pentru restul. Acestea se pot calcula în complexitatea O(N*M*H*K) folosind regula generală $C^k_n = C^{k-1}_{n-1} + C^k_{n-1}$ ($k \le K$ şi $n \le N*M*H$). Pentru a reduce complexitatea la O(N*M*H), putem calcula C^K_n direct din C^K_{n-1}, folosindu-ne de faptul ca numărul 30103 este prim şi deci orice număr are un invers modular (care înmulţit cu el dă restul 1 la împărţirea la 30103).

Soluţie diamant

Soluţia se bazează pe faptul ca valoarea maximă în modul a lui **X** este 1*1+1*2+….+1*19+1*20+2*1+2*2+….2*20+3*1+……+19*20+20*20 în cazul în care valorile lui **N** şi **M** sunt maxime adică 20 şi avem +1 (sau -1) în toată matricea. Suma are valoarea 44100. Odată observat acest fapt se poate deduce că o soluţie asemănătoare cu cea a problemei rucsacului se încadrează în limita de timp. Capacitatea rucsacului ar fi **X** iar obiectele ar fi pătrăţelele, greutatea fiind `i*j` pentru pătrăţelul de pe linia `i` coloana `j`.

Soluţie vitale

Există mai multe soluţii de rezolvare în complexitate **O(m log m)**. Dam una dintre ele. Ideea porneşte de la algoritmul lui Kruskal pentru arbore de cost minim care adaugă muchii în graf în ordinea crescătoare a costurilor. Pentru toate muchiile de cost fixat **x** ne uităm la componentele conexe ale grafului ce conţine muchiile de cost mai mic decat **x,** acest graf va conţine nişte componente conexe. Putem sa le contractam acele componente conexe în noduri pentru şi să ignorăm muchiile de cost mai mare sau egal cu **x** care evident nu sunt muchii vitale. Acum observăm că muchiile vitale de cost **x** sunt muchiile critice în graful care conţine nodurile menţionate anterior şi muchiile de cost **x**. Contractarea nodurilor în componente conexe se realizează folosind structuri de mulţimi disjuncte, iar determinarea muchiilor critice se face cu binecunoscutul algoritm pentru aflarea componentelor biconexe de complexitate **O(m)**. Acest algoritm are complexitate **O(m log m)** în faza de sortare, fazele de contracţie ale grafului vor avea complexitate totala maxim **O(m log* n)** folosind structuri de mulţimi disjuncte, iar fazele de determinare ale muchiilor critice durează **O(m)** în total, pentru că la pasul de cost **x** determinarea componentelor biconexe are complexitate **O(mx)** unde **mx** este numărul de muchii de cost **x**, cum în total numărul de muchii este **m,** obţinem complexitatea **O(m)** . Astfel complexitatea finală este **O(m log m)**.

Algoritmi neoptimi la care ne-am gândit au fost generarea unui arbore partial de cost minim, iar apoi eliminarea câte unei muchii din graf după care determinăm arborele de cost minim pe graful obţinut. Dacă acest arbore are cost diferit de cel original, evident muchia eliminata a fost una critică. Acest algoritm are complexitatea **O(n * m)** pentru că avem n-1 muchii într-un arbore parţial, iar dupa ce muchiile grafului sunt sortate algoritmul Kruskal are complexitare **O(m)**. Acesta ar fi luat în jur de 30-40 de puncte.

Alt algoritm ar fi determinarea randomizată a mai multor arbori parţiali şi returnarea muchiilor comune tuturor arborilor. Acesta ar fi luat în jur de 40 - 50 de puncte.