Tomasz Idziaszek
Treść zadania, Opracowanie

Paweł Parys
Program

Dostępna pamięć: 32 MB. *OI, Etap III, dzień drugi, 07.04.2011*

Konkurs programistyczny

Bartuś i jego koledzy startują w Drużynowym Konkursie Programistycznym. Każda drużyna składa się z n zawodników i ma do dyspozycji n komputerów. Zawody trwają t minut i w tym czasie zawodnicy mają do rozwiązania m zadań programistycznych. Dodatkowo, drużynom przyznawane są punkty karne — za rozwiązanie zadania po upływie s minut od początku konkursu drużyna otrzymuje s punktów karnych. Wygrywa ta drużyna, która rozwiąże największą liczbę zadań, a w przypadku remisu ta, która ma najmniej punktów karnych.

W dniu zawodów Bartuś szybko przegląda treści zadań i rozdziela je między kolegów. Zna on dobrze swoją drużynę i potrafi bezbłędnie ocenić, kto jest w stanie zrobić które zadanie. Każdemu zawodnikowi rozwiązanie zadania, które umie on zrobić, zajmuje dokładnie r minut pracy przy komputerze.

Drużynie Bartusia nie poszło zbyt dobrze na tegorocznej edycji Konkursu. Zastanawia się on, jaki wpływ na ten przykry fakt miały jego decyzje dotyczące przydziału zadań. Poprosił Cię, abyś napisał program, który na podstawie danych, jakie posiadał Bartuś na początku konkursu, obliczy maksymalny możliwy wynik drużyny Bartusia, a także przydział zadań członkom drużyny, który umożliwi osiągnięcie tego wyniku.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się pięć liczb całkowitych n, m, r, t oraz k ($1 \leqslant n, m \leqslant 500$, $1 \leqslant r, t \leqslant 1\,000\,000$), pooddzielanych pojedynczymi odstępami i oznaczających odpowiednio: liczbę zawodników, liczbę zadań, czas rozwiązywania zadania przez zawodnika, czas trwania zawodów i liczbę par zawodnik–zadanie (podanych dalej na wejściu). Każdy z kolejnych k wierszy zawiera dwie liczby całkowite a i b ($1 \leqslant a \leqslant n$, $1 \leqslant b \leqslant m$), oddzielone pojedynczym odstępem, oznaczające, że zawodnik a jest w stanie rozwiązać zadanie b. Każda taka para może pojawić się na wejściu co najwyżej raz.

W testach wartych łącznie 30% punktów zachodzi dodatkowy warunek $n, m \leqslant 100$.

Wyjście

W pierwszym wierszu standardowego wyjścia należy wypisać najlepszy możliwy wynik drużyny Bartusia w postaci dwóch liczb całkowitych oddzielonych pojedynczym odstępem: liczby rozwiązanych zadań z oraz liczby punktów karnych. W kolejnych z wierszach należy wypisać przykładowy przydział zadań: w każdym wierszu mają znaleźć się trzy liczby całkowite a, b i c ($1 \leqslant a \leqslant n$, $1 \leqslant b \leqslant m$, $0 \leqslant c \leqslant t - r$), pooddzielane pojedynczymi odstępami i oznaczające, że zawodnik a powinien zacząć rozwiązywać zadanie b w chwili c (konkurs rozpoczyna się w chwili 0). Nie należy nikomu przydzielać zadań, których dana osoba nie potrafi rozwiązać. Jeśli istnieje więcej niż jedna poprawna odpowiedź, Twój program powinien wypisać dowolną z nich.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`2 4 3 15 4`	`3 12`
`1 1`	`1 4 0`
`2 3`	`2 3 0`
`1 4`	`1 1 3`
`1 3`	

Rozwiązanie

Zacznijmy od kilku prostych obserwacji. Załóżmy, że Bartuś dokonał już przydziału zadań pomiędzy zawodników swojej drużyny, i zastanówmy się, jak powinni oni postępować, by zminimalizować liczbę zdobytych punktów karnych. Jasne jest, że każdy zawodnik powinien rozpocząć rozwiązywanie nowego zadania tak szybko, jak to możliwe, zatem opłaca się zaczynać rozwiązywanie zadań w chwilach, które są wielokrotnościami r. Wynika z tego, że każdy zawodnik może rozwiązać co najwyżej $\lfloor t/r \rfloor$ zadań. Nie ma też znaczenia, w jakiej kolejności zawodnik będzie rozwiązywał przydzielone mu zadania — liczba punktów karnych naliczonych temu zawodnikowi zależy tylko od liczby zadań, które on rozwiązał. Jeśli zawodnik rozwiąże d zadań, to do wyniku dołoży $r(1 + 2 + \ldots + d)$ punktów karnych.

To pozwala nam na sformułowanie problemu w języku teorii grafów: rozważmy graf dwudzielny $G = (O \cup Z, E)$, $|O| = n$, $|Z| = m$, $|E| = k$. Wierzchołki O reprezentują zawodników (osoby), wierzchołki Z — zadania, a krawędź między $o \in O$ a $z \in Z$ istnieje, jeśli zawodnik o potrafi rozwiązać zadanie z.

Przydział zadań możemy przedstawić jako podzbiór zbioru krawędzi $M \subseteq E$. Niech $d_M(o)$ oznacza stopień wierzchołka $o \in O$ w podgrafie $(O \cup Z, M)$; stopień ten odpowiada liczbie zadań przydzielonych zawodnikowi o. Wprowadźmy też oznaczenie $c_M(o) = 1 + 2 + \ldots + d_M(o)$, które nazwiemy *kosztem* zawodnika o; koszt pomnożony przez r jest liczbą punktów karnych naliczonych zawodnikowi o.

Zadanie polega na wyznaczeniu takiego podzbioru $M \subseteq E$ (oznaczającego przydział zadań), który spełnia następujące warunki:

1. każdy wierzchołek z Z jest incydentny z co najwyżej jedną krawędzią z M;
2. dla każdego $o \in O$ mamy $d_M(o) \leqslant t/r$;
3. liczba krawędzi z M jest jak największa;
4. przy założeniu z poprzedniego punktu, sumaryczny koszt $c(M) = \sum_{o \in O} c_M(o)$ jest jak najmniejszy.

Pierwszy warunek oznacza, że każde zadanie jest rozwiązywane przez co najwyżej jednego zawodnika. Drugi warunek gwarantuje, że każdy zawodnik zdąży rozwiązać przydzielone mu zadania. Ostatnie dwa warunki stwierdzają, że przydział zadań M maksymalizuje wynik drużyny.

Rozwiązanie pierwsze: ścieżki polepszające koszt

Na początku rozwiążemy prostsze zadanie, bez zakładania drugiego warunku. Później zobaczymy, że z rozwiązania dla tej wersji łatwo wynika rozwiązanie oryginalnego zadania. Wykorzystamy metodę ścieżek powiększających, znaną przede wszystkim z algorytmu wyznaczania najliczniejszego skojarzenia w grafie.

Zauważmy, że jeśli istnieją zadania, których żaden zawodnik nie umie rozwiązać, to możemy je usunąć. Załóżmy zatem bez straty ogólności, że w zbiorze Z nie ma wierzchołków izolowanych. Dla każdego zadania $z \in Z$ wybierzmy do przydziału M dowolną krawędź wychodzącą z z. Dostaliśmy $|M| = m$, zatem w przydziale M wszystkie zadania zostały rozwiązane. Więcej oczywiście się nie da, postarajmy się teraz uzyskać optymalny koszt.

Ścieżkę w grafie G nazwiemy *naprzemienną*, jeśli zawiera parzystą liczbę krawędzi, zaczyna się w wierzchołku ze zbioru O krawędzią z przydziału M i z każdych dwóch kolejnych krawędzi na ścieżce dokładnie jedna należy do M. Naprzemienną ścieżkę $P = (o_1, z_1, o_2, z_2, \ldots, o_l)$, $o_i \in O$, $z_i \in Z$, $(o_i, z_i) \in M$, $(z_i, o_{i+1}) \in E \setminus M$ nazwiemy *polepszającą koszt*, jeżeli

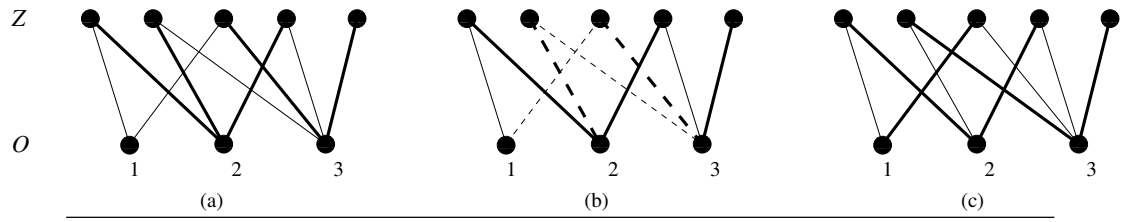
$$d_M(o_1) > d_M(o_l) + 1, \qquad (\star)$$

czyli stopień pierwszego wierzchołka ścieżki jest o co najmniej 2 większy niż stopień ostatniego wierzchołka. Pokażemy, że odwracając przydział krawędzi na ścieżce P, uzyskamy lepszy koszt.

Oznaczmy przez $M' = (M \setminus P) \cup (P \setminus M)$ przydział odwrócony wzdłuż ścieżki P — taką operację nazwiemy *dodaniem* ścieżki P do przydziału M. Jasne jest, że $|M'| = |M|$ oraz że zmieniły się stopnie krańcowych wierzchołków ścieżki: $d_{M'}(o_1) = d_M(o_1) - 1$, $d_{M'}(o_l) = d_M(o_l) + 1$, natomiast stopnie reszty wierzchołków pozostały bez zmian. Koszt nowego przydziału M' jest w istocie mniejszy niż koszt M (w ostatniej nierówności korzystamy z $(\star)$):

$$\begin{aligned} c(M') &= c_{M'}(o_1) + c_{M'}(o_l) + \textstyle\sum_{o \in O \setminus \{o_1, o_l\}} c_{M'}(o) = \\ &= (c_M(o_1) - d_M(o_1)) + (c_M(o_l) + d_M(o_l) + 1) + \textstyle\sum_{o \in O \setminus \{o_1, o_l\}} c_M(o) = \\ &= c(M) - d_M(o_1) + d_M(o_l) + 1 < c(M). \end{aligned}$$

Przychodzi więc na myśl algorytm, który znajduje kolejne ścieżki proste (czyli ścieżki bez cykli) polepszające koszt, aż już żadnej nie będzie. Pojedynczy krok algorytmu — znalezienie ścieżki polepszającej koszt lub stwierdzenie, że taka ścieżka nie



Rys. 1: Dla przydziału zadań M (pogrubione krawędzie w (a)) znajdujemy ścieżkę polepszającą koszt z wierzchołka o_2 do wierzchołka o_1 (przerywane krawędzie w (b)) i odwracamy przydział wzdłuż tej ścieżki, uzyskując M' (c).

istnieje — możemy wykonać w czasie $O(k)$. W tym celu rozważamy wierzchołki ze zbioru O w kolejności nierosnących stopni i z każdego z nich przeszukujemy graf w głąb (oczywiście z wierzchołków z O wychodzimy krawędziami należącymi do przydziału M, a z wierzchołków z Z wychodzimy krawędziami z $E \setminus M$). Ponadto zaznaczamy odwiedzane wierzchołki i tak zaznaczonych wierzchołków nie odwiedzamy ponownie. Dlaczego możemy tak zrobić? Otóż powiedzmy, że przenumerowaliśmy wierzchołki w kolejności nierosnących stopni ($d_M(o_1) \geqslant d_M(o_2) \geqslant \ldots \geqslant d_M(o_n)$) i ostatnim rozważonym wierzchołkiem był wierzchołek o_i. Jeśli nie znaleźliśmy ścieżki polepszającej koszt zaczynającej się w tym wierzchołku, to znaczy, że nie ma ścieżki naprzemiennej zaczynającej się w o_i i kończącej się w wierzchołku z O o stopniu mniejszym niż $d_M(o_i) - 1$. Tak więc na pewno wśród dotychczas odwiedzonych wierzchołków nie znajdziemy też wierzchołka o stopniu mniejszym niż $d_M(o_j) - 1$ dla $j > i$.

Początkowy koszt może wynosić co najwyżej $m(m+1)/2$ (jest tak w przypadku, gdy wszystkie zadania przydzielimy jednemu zawodnikowi), a w każdym kroku algorytmu koszt zmniejsza się co najmniej o 1, zatem uzyskujemy czas $O(m^2k) = O(m^3n)$.

Aby wykazać poprawność naszego algorytmu, wystarczy udowodnić następujące twierdzenie:

Twierdzenie 1. *Koszt przydziału o rozmiarze m jest optymalny wtedy i tylko wtedy, gdy nie istnieje ścieżka prosta polepszająca koszt.*

Dowód: Implikacja w prawo jest oczywista: gdyby istniała ścieżka polepszająca koszt, to dodanie jej do przydziału zmniejszyłoby jego koszt. Udowodnimy implikację w lewo. Niech M będzie przydziałem, którego koszt nie jest optymalny; pokażemy, że istnieje ścieżka prosta polepszająca koszt tego przydziału. Rozważmy przydział N taki, że $|N| = m$ oraz koszt $c(N)$ jest optymalny. Jeśli jest więcej niż jeden kandydat na N, to weźmy tego, dla którego zbiór $D = (M \setminus N) \cup (N \setminus M)$ ma jak najmniej krawędzi. Ścieżką *naprzemienną* względem D nazwiemy ścieżkę $P = (o_1, z_1, \ldots, o_l)$, taką, że $(o_i, z_i) \in M \setminus N$, $(z_i, o_{i+1}) \in N \setminus M$. Z minimalności D wynika, że nie istnieją cykle (ścieżki spełniające $o_1 = o_l$) naprzemienne względem D, gdyż inaczej dodalibyśmy taki cykl do przydziału N i dostalibyśmy również optymalny przydział N, tyle że z mniejszą liczbą krawędzi w zbiorze D. Ponadto, każda ścieżka naprzemienna względem D, prowadząca z o_1 do o_l, spełnia $d_N(o_1) \geqslant d_N(o_l)$. Gdyby bowiem było $d_N(o_1)+1 < d_N(o_l)$, to odwracając kolejność krawędzi na tej ścieżce (tzn. rozważając tę ścieżkę od o_l do o_1), uzyskalibyśmy ścieżkę polepszającą koszt w N. Gdyby zaś zachodziło $d_N(o_1) + 1 = d_N(o_l)$, to dodanie tej ścieżki do N nie zmieniłoby kosztu, ale zmniejszyłoby rozmiar D, co znowu przeczy minimalności zbioru D.

Ścieżkę prostą polepszającą koszt w M znajdujemy następująco: wybieramy wierzchołek $o_1 \in O$ taki, że $d_M(o_1) > d_N(o_1)$ (skoro $c(M) > c(N)$, to taki wierzchołek musi istnieć). Konstruujemy teraz ścieżkę naprzemienną względem D do wierzchołka o_l takiego, że $d_{M \setminus N}(o_l) = 0$. Konkretnie, z wierzchołka o_i wychodzimy dowolną krawędzią z $M \setminus N$ (jeśli $d_{M \setminus N}(o_i) \neq 0$, to taka krawędź istnieje), natomiast z wierzchołka z_i wychodzimy jedyną krawędzią z $N \setminus M$ (z każdego wierzchołka ze zbioru Z wychodzi jedna krawędź z N i jedna z M, a ponieważ do z_i weszliśmy krawędzią z $M \setminus N$, to w przypadku tego wierzchołka są to dwie różne krawędzie). Nie może się zdarzyć, że w którymś momencie wrócimy do wierzchołka, z którego wyszliśmy, bo wtedy

otrzymalibyśmy cykl naprzemienny względem D. Zauważmy również, że ścieżka jest niepusta ($l > 1$), bo $d_{M\setminus N}(o_1) > 0$. Widzimy wreszcie, że do ostatniego wierzchołka o_l weszliśmy krawędzią z $N \setminus M$, a nie wychodzą z niego krawędzie z $M \setminus N$, a więc $d_M(o_l) \leqslant d_N(o_l) - 1$. Dostajemy zatem

$$d_M(o_l) \leqslant d_N(o_l) - 1 \leqslant d_N(o_1) - 1 < d_M(o_1) - 1,$$

przy czym środkową nierówność uzasadniliśmy w poprzednim akapicie. To kończy dowód — skonstruowana ścieżka jest ścieżką polepszającą koszt w przydziale M, gdyż spełnia warunek ($\star$). ■

Co z czasem trwania zawodów?

Rozwiązanie uzyskane powyższą metodą nie uwzględnia warunku, że żaden zawodnik nie może rozwiązać więcej niż t/r zadań. Może się zatem zdarzyć, że w optymalnym przydziale M dla niektórych wierzchołków $o \in O$ będzie $d_M(o) > t/r$. Okazuje się jednak, że aby uzyskać rozwiązanie pierwotnego zadania, wystarczy w optymalnym przydziale M usunąć dowolne nadmiarowe krawędzie wychodzące z tych wierzchołków. Poniżej udowodnimy ten fakt.

Przez $M_{|d}$ oznaczmy (pewien) przydział uzyskany z M poprzez ograniczenie z góry stopnia każdego z wierzchołków ze zbioru O przez d. Wprowadźmy też oznaczenie na jakość rozwiązania obciętego. Mianowicie *d-kosztem* przydziału zadań M nazwiemy parę

$$c_d(M) = (|M_{|d}|, c(M_{|d})),$$

w której pierwsza współrzędna oznacza rozmiar obciętego przydziału, a druga jego koszt. Ponieważ

$$\begin{aligned} |M_{|d}| &= \textstyle\sum_{o\in O} \min(d_M(o), d), \\ c(M_{|d}) &= \textstyle\sum_{o\in O} 1 + 2 + \ldots + \min(d_M(o), d), \end{aligned}$$

więc definicja $c_d(M)$ nie zależy od wyboru konkretnego przydziału $M_{|d}$.

Powiemy, że przydział M ma d-koszt większy niż przydział N, jeśli albo $|M_{|d}| < |N_{|d}|$, albo $|M_{|d}| = |N_{|d}|$ i $c(M_{|d}) > c(N_{|d})$. Jasne jest, że w zadaniu chodzi o zminimalizowanie $\lfloor t/r \rfloor$-kosztu przydziału zadań.

Twierdzenie 2. *Jeśli nie istnieje ścieżka prosta polepszająca koszt, to d-koszt przydziału o rozmiarze m jest optymalny.*

Dowód: Dowód przebiega w analogiczny sposób jak dowód implikacji w lewo twierdzenia 1. Niech M będzie przydziałem, którego d-koszt nie jest optymalny — wskażemy ścieżkę prostą polepszającą koszt tego przydziału. Analogicznie rozważamy przydział N o optymalnym d-koszcie i minimalnym zbiorze $D = (M \setminus N) \cup (N \setminus M)$.

Jedynym miejscem w dowodzie, w którym korzystaliśmy z nieoptymalności kosztu przydziału M, było pokazanie istnienia wierzchołka początkowego ścieżki, tzn. wierzchołka $o_1 \in O$ takiego, że $d_M(o_1) > d_N(o_1)$. Pokażemy, że do znalezienia go wystarczy nieoptymalność d-kosztu M.

Istotnie, jeśli $|M_{|d}| < |N_{|d}|$, to mamy $\sum_{o \in O} \min(d_M(o), d) < \sum_{o \in O} \min(d_N(o), d)$, zatem musi istnieć wierzchołek $o'_1 \in O$ taki, że $d_M(o'_1) < d_N(o'_1)$. Ale jednocześnie mamy $\sum_{o \in O} d_M(o) = m = \sum_{o \in O} d_N(o)$, co wymusza istnienie wierzchołka o_1.

Jeśli z kolei $|M_{|d}| = |N_{|d}|$, to musi być $c(M_{|d}) > c(N_{|d})$. W takim wypadku również łatwo widać, że wierzchołek o_1 musi istnieć. ■

Powyższe rozwiązanie zostało zaimplementowane w pliku `pro3.cpp`. Można je przyspieszyć, stosując pewne proste optymalizacje, które nie zmniejszają teoretycznej złożoności algorytmu, ale w praktyce sprawdzają się całkiem dobrze. Przykładowo, dużo daje, jeśli nie zaczynamy od dowolnego rozwiązania, lecz od już dość dobrego. Może to być, na przykład, wynik zachłannego poprawiania poprzez zmianę jednej krawędzi. Wtedy ścieżki polepszające koszt trzeba znajdować już stosunkowo mało razy. Dodatkowo, można też nie zaczynać szukania ścieżki polepszającej koszt z wierzchołków o najmniejszym lub o jeden większym stopniu, bo wiadomo, że i tak żadnej nie znajdziemy.

Rozwiązanie drugie: ścieżki powiększające

Przedstawimy teraz szybszy algorytm. Podobnie jak poprzednio, zakładamy, że zbiór Z nie zawiera wierzchołków izolowanych, oraz nie bierzemy pod uwagę ograniczenia na czas trwania konkursu. Zaczynamy od pustego przydziału M. W każdym kroku koszt $c(M)$ będzie optymalny dla mniejszego grafu, zawierającego tylko zadania incydentne z którąś krawędzią z M. W kroku i (dla $i = 1, 2, \ldots, m$) będziemy chcieli dodać krawędź wychodzącą z zadania z_i. Ścieżkę w grafie nazwiemy *powiększającą*, jeśli zawiera nieparzystą liczbę krawędzi, zaczyna się w wierzchołku ze zbioru Z krawędzią nienależącą do M i z każdych dwóch kolejnych krawędzi na ścieżce dokładnie jedna należy do M. Szukamy ścieżki powiększającej z wierzchołka z_i, która kończy się w wierzchołku $o \in O$ o najmniejszym stopniu $d_M(o)$. Dodajemy tę ścieżkę do M, uzyskując M', takie że $|M'| = |M| + 1$ i koszt $c(M')$ jest optymalny. Pojedynczy krok wykonujemy przeszukiwaniem grafu w głąb w czasie $O(k)$, zatem cały algorytm działa w czasie $O(mk) = O(m^2 n)$.

Uzasadnimy teraz poprawność tego algorytmu. Oznaczmy przez R ścieżkę powiększającą, którą dodaliśmy do przydziału M, tworząc przydział M', oraz niech $o \in O$ i $z \in Z$ będą końcami ścieżki R. Pokażemy, że jeśli przydział M miał optymalny koszt, to M' także ma optymalny koszt, a konkretnie, że dla M' nie istnieje ścieżka polepszająca koszt.

Załóżmy nie wprost, że ścieżka taka istnieje. Możemy przyjąć, że jest ona prosta, oznaczmy ją przez $P = (o_1, z_1, \ldots, o_l)$. Załóżmy na początek, że $o_1 \neq o$.

Ponieważ P jest ścieżką polepszającą koszt dla M', więc $d_{M'}(o_1) > d_{M'}(o_l) + 1$, a ponieważ przez dodanie R nie zmieniliśmy stopnia wierzchołkowi o_1, a wierzchołkowi o_l mogliśmy go co najwyżej podnieść (gdy $o_l = o$), zatem $d_M(o_1) > d_M(o_l) + 1$. Ponadto, ścieżka P musi przecinać ścieżkę R, w przeciwnym bowiem wypadku byłaby ścieżką polepszającą koszt dla M. Niech v' i v'' będą, odpowiednio, pierwszym i ostatnim wierzchołkiem ścieżki P spośród tych, które należą również do R. Można zauważyć, że $v' \in O$. Ponieważ ścieżka naprzemienna $o_1 \overset{P}{\rightsquigarrow} v' \overset{R}{\rightsquigarrow} o$ nie może być

ścieżką polepszającą koszt dla M, zatem $d_M(o_1) \leqslant d_M(o) + 1$. Z tych dwóch nierówności wynika, że $d_M(o_l) + 1 < d_M(o_1) \leqslant d_M(o) + 1$.

Jeśli $v'' \in O \setminus \{o_l\}$, to ścieżka P opuszcza wierzchołek v'' krawędzią z M, a jeśli $v'' \in Z$, to krawędzią spoza M. Wynika z tego, że ścieżka $z \overset{R}{\rightsquigarrow} v'' \overset{P}{\rightsquigarrow} o_l$ jest ścieżką powiększającą dla M. Co więcej, z faktu $d_M(o_l) < d_M(o)$ wynika, że wierzchołek o_l jest lepszym kandydatem na koniec ścieżki R niż wierzchołek o. Sprzeczność — zatem ścieżka P nie istnieje, co dowodzi optymalności kosztu $c(M')$.

Pozostał przypadek, gdy $o_1 = o$. Wtedy jednak mamy $d_M(o) + 1 = d_{M'}(o)$ oraz $d_M(o_l) = d_{M'}(o_l)$, a z własności P jest $d_{M'}(o) > d_{M'}(o_l) + 1$, zatem ostatecznie $d_M(o) > d_M(o_l)$ i dalej działa ten sam argument.

Podobnie jak poprzednio, możemy na koniec usuwać nadmiarowe krawędzie wychodzące od osób, które rozwiązują więcej niż t/r zadań. Równoważnie, możemy jednak od razu nie dodawać ścieżki do M, jeśli jej dodanie powoduje zwiększenie stopnia jakiegoś wierzchołka ponad t/r (dowód podobny do poprzednich nie jest trudny).

To rozwiązanie zostało zaimplementowane w pliku `pro4.cpp`. Także w tym rozwiązaniu możemy poczynić pewne optymalizacje. Często zdarza się tak, że ścieżka powiększająca jest bardzo krótka, nawet jednokrawędziowa. Mimo to nasze rozwiązanie przejdzie cały graf w jej poszukiwaniu. Program dużo lepiej zachowuje się, jeśli najpierw sprawdzimy, czy może któryś z sąsiednich wierzchołków z O nie należy jeszcze w ogóle do żadnej krawędzi z M. Wtedy możemy ich po prostu skojarzyć i nie przeszukiwać całego grafu. Optymalizacja ta spisuje się najsłabiej, gdy zadań jest istotnie więcej niż osób. Wtedy (choć graf musi być mniejszy) dla wielu zadań nie będzie sąsiada, który nie rozwiązywałby jeszcze żadnego zadania, i będzie trzeba dla nich przejść cały graf.

Najsłabszym punktem naszego rozwiązania $O(m^2n)$ jest to, że w zasadzie dla znalezienia najlepszej ścieżki powiększającej trzeba przejrzeć cały graf (bo ma to być najlepsza ścieżka, więc może kończyć się wszędzie). Możemy jednak spojrzeć na to od drugiej strony: zamiast od zadania, będziemy szukać najlepszej ścieżki powiększającej od osoby. Wtedy musimy zacząć od osoby o najmniejszym możliwym stopniu (pomijając osoby, o których już wiemy, że żadna ścieżka z nich nie istnieje). Przy tym założeniu możemy dotrzeć do dowolnego zadania i ścieżka będzie najlepsza. Możemy zatem szukać najkrótszej takiej ścieżki, przeszukując graf wszerz. Jeśli jest ona krótka (co zdarzy się w większości przypadków), to przeszukiwanie zakończy się szybko i nie będzie przechodziło całego grafu. Takie rozwiązanie jest zaimplementowane w plikach `pro.cpp` i `pro2.pas` oraz, z drobnymi usprawnieniami, w pliku `pro1.cpp`.

Rozwiązanie trzecie: przepływ w sieci

Zadanie można sprowadzić do problemu znajdowania maksymalnego przepływu o minimalnym koszcie w sieci (ang. *min-cost max-flow problem*). Oznaczmy przez $d = \min(m, \lfloor t/r \rfloor)$ maksymalną liczbę zadań, którą może rozwiązać pojedynczy zawodnik. Tworzymy sieć H o następujących wierzchołkach:

- źródło s i ujście t;
- wierzchołki $z_1, \ldots, z_m$ odpowiadające zadaniom;

- wierzchołki $o_1, \ldots, o_n$ odpowiadające zawodnikom.

Ponadto w sieci (dla każdego $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$):

- istnieje krawędź o koszcie 0 ze źródła s do wierzchołka z_i;
- jeśli i-te zadanie może być rozwiązane przez j-tego zawodnika, to istnieje krawędź o koszcie 0 z wierzchołka z_i do wierzchołka o_j;
- istnieje d krawędzi o rosnących kosztach $1, 2, \ldots, d$ z wierzchołka o_j do ujścia t.

Wszystkie krawędzie mają przepustowość jednostkową. Każdy przepływ w sieci H odpowiada pewnemu przydziałowi zadań (i na odwrót). Wartość przepływu jest równa liczbie rozwiązanych zadań, natomiast koszt przepływu pomnożony przez r jest równy liczbie punktów karnych, zatem maksymalny przepływ o minimalnym koszcie odpowiada optymalnemu rozwiązaniu.

Liczba wierzchołków w sieci wynosi $n' = 2 + m + n = O(m + n)$, natomiast liczba krawędzi $k' = m + k + nd = O(mn)$. W pliku `pros5.cpp` zaimplementowano to rozwiązanie, korzystając z algorytmu znajdowania przepływu działającego w czasie $O(fn'k') = O(m^2n(m + n))$, gdzie $f = m$ jest maksymalną wartością przepływu.

Rozwiązania niepoprawne

Najbardziej narzucającym się rozwiązaniem niepoprawnym jest zachłanne poprawianie przydziału zadań: szukamy zadania, które możemy przydzielić komuś innemu, polepszając koszt. Takie rozwiązanie znajduje się w pliku `prob6.cpp` i przechodzi jedynie dwa testy: 7 i 8. Nie działa ono już na dwóch testach przykładowych.

Można też zaproponować następujące rozwiązanie zachłanne. Obliczamy w nim:

- trudność każdego zadania — im więcej zawodników umie zrobić dane zadanie, tym jest ono łatwiejsze,
- doświadczenie każdego zawodnika — im więcej zadań umie zrobić dany zawodnik, tym jest bardziej doświadczony.

Teraz próbujemy zachłannie przydzielać zadania w kolejności od najtrudniejszego, za każdym razem wybierając możliwie najmniej doświadczonego zawodnika. Rozwiązanie znajduje się w pliku `prob7.cpp` i przechodzi testy 7 i 8.

Kolejne rozwiązanie niepoprawne opiera się na spostrzeżeniu, że każdy przydział zadań M jest sumą pewnej liczby skojarzeń w grafie dwudzielnym G (dalej będziemy tę liczbę oznaczali przez Δ). Pomysł jest więc taki, by znajdować kolejne maksymalne skojarzenia w grafie, dokładać je do M i usuwać z G. Rozwiązanie to ma złożoność $O(m^2n)$ i w praktyce działa sprawnie. Zapisane zostało w pliku `prob8.cpp`. Rozwiązanie daje błędne odpowiedzi w przypadku grafów rzadkich lub takich, które wymagają dużego Δ. Przechodzi wszystkie testy przykładowe i tylko dwa testy punktowane (1 i 6). W pliku `prob9.cpp` znajduje się program, który 10 razy próbuje wykonać powyższy algorytm, za każdym razem permutując losowo listy sąsiedztwa grafu. Ten program nie wykazuje żadnej poprawy względem programu `prob8.cpp`.

Testy

Do zadania przygotowano 26 testów połączonych w 10 grup (testy 8b, 9b, 10b, 10c to testy proste pod względem złożoności, jednak sprawdzające różne skrajne przypadki). Część testów specjalnie odsiewa rozwiązanie `prob8.cpp`. Poniżej szczegółowe zestawienie testów.

Nazwa	n	m	k	$\lfloor t/r \rfloor$	wynik	Opis
pro1a.in	20	100	816	11	100	losowy
pro1b.in	20	100	817	11	100	losowy
pro2.in	50	100	269	2	95	losowy
pro3a.in	80	80	2 125	1	80	losowy
pro3b.in	8	80	33	7	27	losowy
pro4a.in	200	200	16 201	10	200	losowy
pro4b.in	200	200	15 971	10	200	losowy
pro4c.in	20	200	152	9	121	losowy
pro5a.in	252	499	97 624	2	494	losowy plus wierzchołki stopnia 0
pro5b.in	252	499	97 588	2	494	losowy plus wierzchołki stopnia 0
pro5c.in	7	494	103	400	91	losowy
pro6a.in	500	500	74 870	2	500	losowy
pro6b.in	497	486	30 381	419	478	rozłączne części 487 : 243 i 10 : 243 $(n : m)$
pro6c.in	6	495	471	469	469	jeden zawodnik rozwiązuje
pro7a.in	500	500	123 295	3	500	rozłączne części 249 : 498 i 251 : 2
pro7b.in	479	493	47 283	117	493	losowy
pro7c.in	3	500	439	120	324	losowy
pro8a.in	500	500	17 629	3	390	luźno połączone części 95 : 385 i 395 : 85 plus dodatki
pro8b.in	500	500	74 893	0	0	losowy, $r > t$
pro8c.in	430	500	107 652	497	500	losowy
pro9a.in	500	500	80 367	10^6	500	losowy
pro9b.in	500	500	0	5	0	bez krawędzi

Nazwa	n	m	k	⌊t/r⌋	wynik	Opis
pro9c.in	20	500	409	314	294	rozłączne części 2 : 100, 3 : 200, 2 : 200
pro10a.in	500	500	25634	3	452	luźno połączone części 100 : 200, 200 : 100, 192 : 191 plus dodatki
pro10b.in	500	500	250 000	1	500	każdy umie wszystko
pro10c.in	500	500	999	1	500	wąż