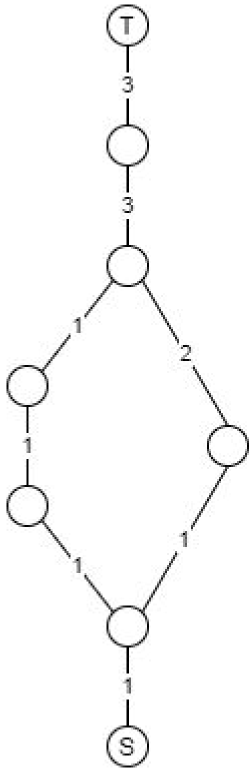
Zadania przed kolokwium II e02 - rozwiązania

Zadanie 1

Dlaczego prosty algorytm Dijkstry (w domyślnej wersji) nie zadziała:



Dikstra na tym grafie na głębokość k=5 nie zadziała. Zaczynamy od źródła S. W przedostatnim wierzchołku (tym tuż poniżej T) będziemy mieli wagę 6, bo poszliśmy ścieżką po kolei S-1-1-1-1-3 (lewa gałąź). Wyczerpaliśmy tym samym nasz zapas krawędzi. Algorytm Dijkstry jest “ślepy” na prawą gałąź, bo to algorytm zachłanny - “nie widzi” tego, że tam jest mniej krawędzi, a większa waga. W związku z tym nigdy nie dojdziemy do wierzchołka T.

Rozwiązanie 1:

Robimy coś w stylu dynamicznego algorytmu Bellmana-Forda. Tworzymy tablicę Vxk, gdzie t[i][j] oznacza wagę najkrótszej ścieżki ze źródła do i, mającą i krawędzi. Obliczamy to próbując zrelaksować po każdym sąsiedzie danego wierzchołka, używając wag z t[sąsiad][j-1]. Innymi słowy jest to k iteracji bellmana forda, z tym, że i i-tej iteracji używamy wag zrelaksowanych w i-1 tej iteracji.

Rozwiązanie 2:

Dijkstra na tablicach przecież działa!

W każdym wierzchołku grafu tworzymy k-elementową tablicę, wypełnioną inf. Index w tablicy oznacza liczbę krawędzi, którą do tej pory przeszliśmy w danej ścieżce. I jeśli chcesz relaksowac z i-tego miejsca w tablicy, to relaksujesz na i+1 miejsce i tylko na nie. A rzucając na kolejkę pamiętasz która krawędź i który index. Czyli w przykładzie po prawej wrzucasz najpierw (S,0), potem wyzszy wierzchołek (...,1), potem w lewo (...,2) i prawo (...,2), potem lewa scieżka kolejno (...,3), (...,4), a potem prawa (...,3) (... to jakaś nazwa wierzchołka, ale tu nie są opisane). No i w ten sposób nie tracisz informacji. Tylko trzeba pamiętać, że a) od razu robisz k*n operacji (przygotowanie tablic) b) tracisz zysk z dijkstry, to znaczy i tak przeglądasz wszystko, zatem dijkstra też sam w sobie ma k * n.

Zadanie 2

Sortujemy topologicznie i w takiej kolejności relaksujemy (ze zmienionym zwrotem nierówności oczywiście).

Zadanie 3

Tworzymy node Find&Union z każdej wiśni. Iterujemy po wszystkich czarnych krawędziach i robimy union, które zwraca nam informację, czy udało się unionowanie (może się nie udać, jak już są w jednym zbiorze), zawsze jak się uda inkrementujemy zawartość cukru o 100. Utworzyliśmy w ten sposób zbiór spójnych składowych, które musimy połączyć czerwonymi krawędziami. Jeżeli wyobrazimy sobie, że każda spójna składowa, to duży wierzchołek, to potrzebujemy n-1 czerwonych krawędzi, gdzie n to liczba spójnych składowych. n wyznaczamy iterując po wierzchołkach i zliczając liczbę unikalnych reprezentantów.

Zadanie 4

Rozwiązanie 1:
Usuwamy krawędzie o roadCost większym niż libraryCost (bo po co ją naprawiać, skoro można walnąć bibliotekę). Teraz robimy Kruskala i dostajemy serię spójnych składowych. W każdej robimy bibliotekę.

Rozwiązanie 2:
Można też użyć find&union, bo jeśli usunie się roadCost'y większe od libraryCost'a, to ma się pewność, że decyzja zachłanna o wykorzystaniu drogi jest poprawna. Tak więc sortujemy roadCost'y i lecimy po nich od najmniejszego, robiąc union, jeśli się da. Potem dla każdego zbioru robimy biblioteczkę.

Zadanie 5

Mamy własność, że 2^n > 1 + 2 + .. + 2^n-1. Z tej własności wynika, że możemy odrzucić wszystkie krawędzie, które nie należą do MST. Teraz DFS z każdego wierzchołka.

Zadanie 6

Funkcja dynamiczna f(i,j) - maksymalna piękność uzyskiwalna do i-tego stosu, przy założeniu, że biorę dokładnie j talerzy. Jak obliczamy:

```
ll solve(vector<vector<ll>>& plates, ll** dp, int N, int K, int P)
{
   for(int i = 0; i<N; i++)
      dp[i][0] = 0;

   for(int i = 1; i<=K && i<=P; i++)
      dp[0][i] = dp[0][i-1] + plates[0][K-i];

   for(int i = 1; i<N; i++)
      for(int j = 1; j<=(i+1)*K && j<= P; j++)
      {
         ll sum = 0;
         dp[i][j] = -1;
         for(int plat = 0; plat<= K; plat++, sum += plates[i][K-plat])
                  if((j-plat)<=i*K && (j-plat)>=0)
                     dp[i][j] = max(dp[i][j], dp[i-1][j-plat]+sum);
      }
   return dp[N-1][P];
}
```

[Widziałem kiedyś podobne zadanko z modyfikacją, że możesz przełożyć talerz ze stosu i na stos j, pod warunkiem, że stos j ma mniejszą wysokość niż k. Ale to już wtedy było niezbyt przyjemne. I to już może być NP-trudne]

Zadanie 7

Startuję DFS z jednego wierzchołka z listy miast wyznaczam najdalszy do niego. Dla tego znalezionego powtarzam to samo. Mam średnicę grafu. Można teraz sobie wyobrazić, że ta średnica to długa ścieżka, a do pozostałych wierzchołków z listy dochodzimy takimi ‘drzewkami’ do niej doczepionymi. Teraz szukany koszt to raz waga tej średnicy + 2*waga każdego drzewka. Chodzi, o to, żeby znaleźć jak najdłuższy fragment, po którym możemy poruszać się bez zawracania.

https://www.hackerrank.com/challenges/jeanies-route/editorial