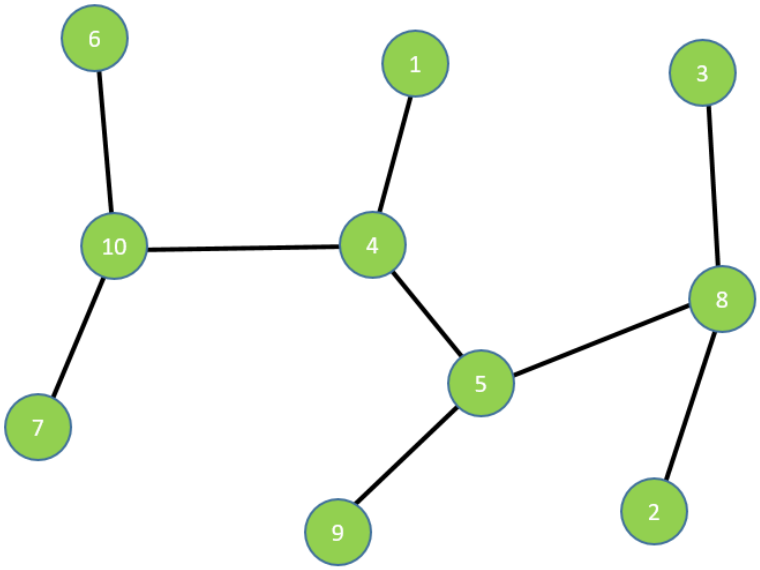
Średnica drzewa

#drzewo_nieukorzenione

Nie wszystkie drzewa mają korzenie. Amatorzy westernów na pewno pamiętają takie pustynne krzaki, które nie są ukorzenione, ale toczą się z wiatrem po powierzchni ziemi.* No, może nie są to drzewa, ale ich sposób na egzystencję jest dość specyficzny.

W informatyce mamy do czynienia z rozmaitymi rodzajami drzew: w szczególności tym mianem określa się dowolny nieskierowany, acykliczny i spójny graf. Czyli żeby na jego krawędziach nie było zwrotów (strzałek), żeby nie dało się biegać w kółko (po zamkniętym cyklu krawędzi) i żeby był w jednym kawałku – kumacie, *c'nie*?

Taki graf ma dokładnie $V - 1$ krawędzi (dlaczego?), o ile oczywiście V oznacza ilość wierzchołków. Oto przykład takiego grafu:



Drzewo nie jest ukorzenione, zatem nie posiada żadnego wyróżnionego wierzchołka (w przeciwieństwie na przykład do kopca binarnego).

Zanim się zacznie, przygotujmy dane wejściowe opisujące ten graf (według standardowego wzorca):

```
10 9
6 10
10 7
9 5
4 5
```

*Takie rośliny to *biegacze*.

```
4 10
1 4
3 8
8 2
8 5
```

Dla powyższych danych funkcja `print_graph()` wypisze takie oto listy sąsiedztwa:

```
1: 4
2: 8
3: 8
4: 5 10 1
5: 9 4 8
6: 10
7: 10
8: 3 2 5
9: 5
10: 6 7 4
```

OK, ale czego my właściwie poszukujemy?[†] Wymieniona w tytule podrozdziału *średnica drzewa* to po prostu długość najdłuższej ścieżki w tym drzewie.[‡] Oczywiście ścieżkę traktujemy jako drogę jednokierunkową, nie biegamy po niej tam i z powrotem.

Posłużymy się tutaj algorytmem BFS, który co prawda służy do wyszukiwania najkrótszych ścieżek, ale tym razem – odpowiednio użyty – pozwoli na znalezienie najdłuższej ścieżki, czyli średnicy grafu.

Ogólna strategia będzie taka: najpierw puszczamy BFS-a z byle którego wierzchołka grafu, powiedzmy z wierzchołka 1 (jest to naprawdę głęboko obojętne). W rezultacie dla każdego z wierzchołków zostanie wyznaczona odległość od wierzchołka startowego (1). Następnie wybieramy wierzchołek o największej odległości i z niego puszczamy drugiego BFS-a. Największa odległość znaleziona przez ten drugi przebieg algorytmu to poszukiwana średnica grafu (dlaczego?).

OK, może się okazać, że po pierwszym przebiegu mamy kilka wierzchołków o takiej samej maksymalnej odległości. Wtedy wybieramy dowolny z nich jako wierzchołek startowy do drugiego przebiegu. Wyszukanie właściwego wierzchołka możemy przeprowadzić po zakończeniu pierwszego przebiegu, aby nie modyfikować dobrze działającej funkcji `BFS()`.

Należy wszelako pamiętać, aby po pierwszym przebiegu wyczyścić wektor *visited*, bo inaczej drugi przebieg w ogóle nie wystartuje.

[†]Poza straconym czasem...

[‡]Ważne, że graf jest drzewem, wtedy długość takiej maksymalnej ścieżki jest jednoznacznie określona. Niech Czytelnik znajdzie kontrprzykłady dla grafu zawierającego choć jeden cykl.

Tak może wyglądać przykładowa funkcja `main()` rozwiązująca ten problem (pomijamy nagłówek programu oraz definicje funkcji `read_graph()` i `BFS()`) – najpierw pierwszy przebieg:

```
. . .

int main()
{
  int V, E;
  vector<vector<int>> G;
  read_graph(G, V, E);
  vector<bool> visited;
  vector<int> parent, dist;
  BFS(G, V, 1, visited, parent, dist);
```

Teraz szukamy najodleglejszego wierzchołka (jego numer to *v_max*):

```
int v_max = 1;
for(int v = 2; v <= V; v++)
  if(dist[v] > dist[v_max])
    v_max = v;
```

Czyścimy wektor *visited* i robimy drugi przebieg:

```
fill(visited.begin(), visited.end(), false);
BFS(G, V, v_max, visited, parent, dist);
```

Znów szukamy najodleglejszego wierzchołka i wypisujemy średnicę drzewa:

```
  v_max = 1;
  for(int v = 2; v <= V; v++)
    if(dist[v] > dist[v_max])
      v_max = v;
  cout << dist[v_max] << endl;
}
```

Dla naszego przykładowego grafu po pierwszym przebiegu maksymalną odległością będzie 4 – taki wynik zanotują wierzchołki 2 oraz 3. Po drugim przebiegu największy rezultat (średnica grafu) to 5 – dla wierzchołków 6 i 7.