**Report**

Вовед

**Програмот ги содржи алгоритмите за Max Flow:**

1. Edmonds-Karp Algorithm: Специфична имплементација на Ford-Fulkerson, кој користи BFS, за да најде аугментирачки патишта.
2. Ford-Fulkerson Algorithm: Користи DFS за да најде аугментирачки патишта

**Програмата содржи граф, прикажан на два формати:**

* Adjacency List
* Adjacency Matrix

Целта на оваа прогрма е да се најде максималниот можен проток од даден source до sink во граф со дадените алгоритми.

1. **Граф класата**

Класата Graph служи како основна класа за претставите на графикот. Тој е дизајниран да се прошири за две подкласи:

GraphAdjList: Граф претставен со помош на листа со соседства.

GraphAdjMatrix: Граф претставен со помош на матрица за соседство.

**Атрибути на класа на графи:**

n: Бројот на јазли во графот.

Капацитет: 2D низа (од вектори) што го складира капацитетот на секој раб.

капацитет: Матрица за преостанат капацитет, која се ажурира кога протокот се пренесува низ графикот.

adj: Структурата на соседството, која може да биде листа или матрица во зависност од приказот на графикот.

**Методи за класи на графи:**

Конструктор и разурнувач: Тие распределуваат и распоредуваат динамичка меморија за соседните структури и матриците на капацитет.

build\_graph: Овој метод му овозможува на корисникот да ги внесе рабовите и нивните капацитети за да го изгради графикот.

solveEdmondKarp: Користи BFS за да најде патеки за зголемување и да го пресмета максималниот проток користејќи го алгоритмот Edmonds-Karp.

solveFordFulkerson: Користи DFS за да најде патеки за зголемување и да го пресмета максималниот проток користејќи го класичниот алгоритам Форд-Фулкерсон.

реши: Овој метод ги извршува двата алгоритми (Едмондс-Карп и Форд-Фулкерсон) и ги печати пресметаните вредности на максималниот проток.

**Апстрактни методи:**

Основната класа Graph содржи виртуелни функции за:

bfs: За пронаоѓање на патеки за зголемување користејќи BFS.

dfs: За наоѓање патеки за зголемување со помош на DFS.

add\_edge: За додавање рабови и дефинирање на капацитетите помеѓу јазлите.

1. **Класи за прикажување на графи**

Постојат две изведени класи, GraphAdjList и GraphAdjMatrix, кои ја имплементираат функционалноста на графикот користејќи различни претстави.

**GraphAdjList:**

Го претставува графикот користејќи списоци со соседство.

add\_edge: Додава раб на списокот со соседство и за напред и за назад.

bfs: Имплементира BFS за да најде патеки за зголемување за Едмондс-Карп. Таа користи редица за истражување на јазлите слој по слој, следејќи го протокот по секоја патека.

dfs: Имплементира DFS за наоѓање патеки за зголемување за Ford-Fulkerson, каде што пребарувањето продолжува со истражување колку што е можно пред да се врати назад.

GraphAdjMatrix:

Го претставува графикот користејќи матрица за соседство.

add\_edge: Додава рабови во матрицата. Матрицата складира дали постои раб помеѓу јазлите и неговиот капацитет.

bfs и dfs: логиката е слична на верзијата на списокот со соседство, но матричната репрезентација го прави пристапот до соседите поедноставен (со користење на вгнездени јамки).

1. **Алгоритми**

**Алгоритам Едмондс-Карп:**

Користи BFS за наоѓање патеки за зголемување.

Започнува од изворниот јазол и го истражува секој сосед користејќи редица.

За секоја пронајдена патека, протокот се пресметува како минимален капацитет по таа патека.

Откако ќе се најде патека за зголемување, протокот се зголемува, преостанатите капацитети се ажурираат и алгоритмот продолжува додека не постојат повеќе патеки за зголемување.

Временска сложеност: O(VE^2), каде што V е бројот на темиња и E е бројот на рабовите.

**Алгоритам Форд-Фулкерсон:**

Користи DFS за наоѓање патеки за зголемување.

Како и BFS, протокот се зголемува по наоѓање на патека за зголемување, а преостанатите капацитети се ажурираат.

За разлика од Едмондс-Карп, овој метод прво ја истражува длабочината на графикот, кој може да потрае различни патеки.

Временска сложеност: O(max\_flow \* E), каде што max\_flow е максималниот проток во мрежата, а E е бројот на рабовите. Во пракса, може да работи побавно од Едмондс-Карп во густи графи.

1. **Клучни операции**

Зголемување на пребарувањето на патеките:

И BFS и DFS се користат за пронаоѓање на патеки за зголемување, т.е. патеки од изворот до мијалникот со достапен преостанат капацитет.

Пребарувањето BFS гарантира наоѓање на најкратката патека за зголемување во однос на бројот на рабовите, додека DFS не.

**Преостанати ажурирања на графот:**

Откако ќе се најде патека, капацитетите долж патеката се прилагодуваат:

Напредните рабови (од u до v) имаат капацитет намален со протокот.

На задните рабови (од v до u) нивниот капацитет се зголемува со протокот.

Ова му овозможува на алгоритмот да се врати назад и да го турка протокот обратно доколку е потребно во следните повторувања.

**Претставување на рабовите:**

Во двете претстави на графот, рабовите се двонасочни, што значи дека обратниот раб автоматски се додава кога раб се вметнува во графикот. Ова обезбедува правилно управување со преостанатиот капацитет.

1. **Кориснички внес и извршување**

Главната функција го придвижува извршувањето на програмата:

Го поттикнува корисникот да го внесе бројот на графови.

За секој граф, од корисникот се бара да го наведе бројот на јазли и рабови, како и приказот на графот (список со соседство или матрица).

Графот е конструиран и се додаваат рабови со капацитети.

По изградбата на графот, корисникот ги специфицира изворот и јазлите за пресметување на протокот.

Програмата го пресметува и прикажува максималниот проток користејќи ги и Edmonds-Karp и Ford-Fulkerson.

Конечно, динамички распределената меморија за секој графот се ослободува.

1. **Заклучок**

Оваа имплементација обезбедува разновидна рамка за решавање на проблемот со максималниот проток користејќи два различни алгоритми. Тоа му овозможува на корисникот да експериментира со две различни претстави на граф и да ги спореди перформансите на пристапите базирани на BFS и DFS за проблемот со максимален проток.

И двата алгоритма се применуваат на иста структура на графикот, а резултатите покажуваат дека тие даваат ист максимален проток, но нивните внатрешни механизми се разликуваат во начинот на кој ги наоѓаат патеките за зголемување и ракуваат со преостанатиот граф.

**1. Преглед на алгоритам на Диниц**

Алгоритмот на Диниц ги комбинира BFS и DFS на структуриран начин:

Создавање график на ниво (BFS):

BFS се користи за да се конструира график на ниво, каде што на секој јазол му е доделено ниво врз основа на најкратката патека од изворниот јазол. Само рабовите што одат од пониско на повисоко ниво се земаат во предвид за проток.

Пресметка на проток на блокирање (DFS):

Откако ќе се изгради графикот на нивоа, DFS се користи за пронаоѓање на блокирачки текови - зголемувајќи ги патеките во кои барем еден раб станува заситен.

Алгоритмот се повторува помеѓу конструирање график на нивоа и испраќање проток по блокирачките патеки додека не може да се најдат повеќе патеки за зголемување.

Употребата на блокирачки текови осигурува дека, во секое повторување, алгоритмот заситува значителен број рабови, што го прави поефикасен од едноставните пристапи базирани на DFS или BFS.

1. **Имплементација на алгоритам на Диниц**

Во кодот, Диниц алгоритам е имплементиран како дел од класата Graph и користи BFS и DFS како помошни функции за да го изгради графикот на нивоа и да најде блокирачки текови. Специфичните чекори вклучени се:

Изградба на график на ниво (BFS)

Функцијата bfs\_Dinic (int source, int sink, int level[]) го гради графикот на нивоа:

BFS се иницира од изворниот јазол за да се доделат нивоа на секој јазол.

Нивоата се зачувани во низата ниво[], каде што ниво[i] го означува нивото на јазолот i.

Нивото на јазол го покажува минималниот број на рабови на патеката од изворот до тој јазол. Само рабовите од јазли од пониско ниво до јазли на повисоко ниво се дозволени за зголемување на патеките.

Ако не може да се стигне до јазолот на sink од изворот во BFS, алгоритмот запира, што покажува дека нема повеќе патеки за зголемување.

Наоѓање на патеки за зголемување (DFS)

Функцијата bfs\_Dinic(int source, int sink, int родител[], int level[]) наоѓа патеки за зголемување во графикот на нивоа:

DFS се иницира од изворот до sink долж важечките рабови на графикот на нивоа (рабови што одат од пониско ниво на повисоко ниво).

Протокот на патеката (тесно грло) се пресметува како минимален капацитет долж патеката за зголемување.

Ако се најде валидна патека, протокот се зголемува, а преостанатите капацитети се ажурираат.

Овој процес се повторува додека не се најдат повеќе патеки во графикот на тековното ниво.

**Главна функција: solveDinic**

Функцијата solveDinic (int source, int sink) ја координира пресметката на главниот тек:

Прво ја иницијализира матрицата за капацитет[] со оригиналниот Capacity[].

1. **Временска сложеност**

Временска сложеност:

Диниц алгоритам има временска сложеност од O(V²E) за општи графикони, каде V е бројот на темиња, а E е бројот на рабовите. За мрежи со капацитети на единицата, сложеноста може да се намали на O(VE).

Зошто е ефикасно:

Употребата на графот на нивоа и блокирачки текови му овозможуваат на алгоритмот на Диник да обработи голем број на патеки за зголемување во секоја итерација, наместо да наоѓа патеки една по една како во методот Форд-Фулкерсон. Ова драстично го намалува бројот на повторувања потребни за да се постигне максимален проток.

1. Заклучок

Диниц алгоритам е високо ефикасен метод за пресметување на максималниот проток во мрежата, особено за големи графици или графикони со високи капацитети. Имплементацијата во дадениот код го следи стандардниот пристап, користејќи BFS за градење графикони на нивоа и DFS за наоѓање блокирачки текови во тие графови. Со заситување на повеќе патеки во секоја итерација, алгоритмот на Диник постигнува подобри перформанси од едноставните пристапи засновани на BFS или DFS како Едмондс-Карп или Форд-Фулкерсон.

Имплементацијата овозможува флексибилна употреба и на репрезентациите на списокот на соседството и на матрицата на соседството, со што алгоритмот е применлив за различни типови и структури на графови.

**Min-Cut**

**Пресметка на max flow:**

Прво, алгоритмот Edmond Karp се користи за пресметување на максималниот проток од изворот до мијалникот.

За време на овој процес се ажурираат преостанатите капацитети на рабовите.

**BFS за достапност:**

По пресметувањето на максималниот проток, BFS (или DFS) се изведува од изворниот јазол во преостанатиот график за да се идентификуваат сите јазли што се достапни од изворот.

Јазлите до кои сè уште може да се дојде од изворот формираат едната страна од сечењето.

**Идентификување на минимално исечени рабови:**

Минималниот пресек е составен од рабови каде што почетниот јазол е достапен од изворот, но целниот јазол не е.

Овие рабови го претставуваат тесното грло што го ограничува протокот и мора да се отстранат за да се исклучи изворот од мијалникот.

**Печатење на работ:**

Функцијата print\_MinCut ги идентификува и печати овие рабови со проверка на капацитетот на рабовите помеѓу достапните и недостапните јазли во оригиналниот графот**.**