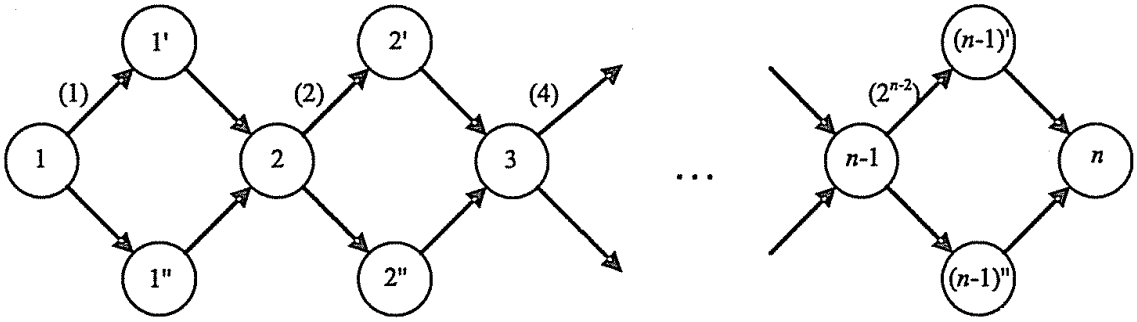
ДИЈКСТРИН АЛГОРИТАМ ЗА НАЛАЖЕЊЕ НАЈКРАЋЕГ ПУТА

***Дино Сејдиновић,* Бристол, УК**

Овде ћемо размотрити један познати алгоритам за налажење најкраћег пута на тежинском усмереном графу, Дијкстрин алгоритам (Edsger Wybe Dijkstra, 1930-2002, славни холандски информатичар). Овај алгоритам спада, можемо слободно рећи, у фолклорне алгоритме. Он је био познат и много пре Дијкстре, но Дијкстра је први дао његову детаљну анализу. С обзиром на то да је број путева у графу у експоненцијалној зависности од броја врхова (чворова) и броја ивица (грана), врло је значајно имати добар (полиномијалан) алгоритам за налажење најкраћег пута. На пример, на графу представљеном на слици 1 имамо 2^{n-1} путева између врхова означених са 1 и n. Даље, уколико су цене (тежине) грана као на слици (гране које су неозначене имају цену 0), сваки од ових путева је различите дужине. Дакле, за велике графове, груба претрага за најкраћим путем не долази у обзир. Занимљиво је да се добар алгоритам добија када заправо пооштримо критеријуме и решавамо наизглед већи и тежи проблем. Наиме, уместо да тражимо најкраћи пут на графу G од врха s до врха t, ми ћемо тражити најкраће путеве од врха s до свих осталих врхова графа G. Иако на први поглед ова идеја делује бизарно, она доводи до елегантног решења и овакав приступ се неретко сусреће у конструкцији алгоритама. У графу G, скуп свих врхова ћемо обележити са V, скуп свих грана са E и скуп свих цена грана са c.



Слика 1. Граф са експоненцијалним бројем путева при чему су сви путеви различите дужине

Алгоритам. Сада ћемо навести псеудокод Дијкстриног алгоритма који у полиномијалном времену рачуна најкраће путеве у графу G од једног фиксираног врха s до свих осталих врхова у графу. У алгоритму ћемо позивати помоћну функцију *update* која као аргумент прима ознаку врха.

Најкраћи пут (Дијкстра):

Input: Graf G, vrh s
Output: – vektor $d(j)$ najkraćih udaljenosti od vrha s do vrha j, $j \in V$
– vektor $prethodnik(j)$ oznaka vrhova koji su u najkraćem putu od vrha s do vrha j prethodnici od j, $j \in V$
Inicijalizacija: $S = \{s\}$; $T = V \setminus S$; $d(s) = 0$; $prethodnik(s) = 0$;
$(\forall j \in T) d(j) = \infty$;

$update(s)$;

while $S \neq V$
    neka je $i \in T$ vrh za koji je $d(i) = \min\{d(j) \mid j \in T\}$
    $S = S \cup \{i\}$; $T = T \setminus \{i\}$;
    $update(i)$;
end while
return d, $prethodnik$
end

Наведимо сада и псеудокод функције $update$ која мења вредности у векторима d и $prethodnik$ у зависности од одабраног врха v који улази у „тражени“ скуп S.

$update(v)$:
    Za sve sledbenike i vrha v uraditi:
    if $d(v) + c_{vi} < d(i)$
        $d(i) = d(v) + c_{vi}$;
        $prethodnik(i) = v$;
    end if
end

Дакле, сврха функције $update$ је да, уколико пронађе да је пут до врха i који води преко актуелног врха v (над којим је функција позвана) краћи од пута који је већ откривен, то решење (пут од s до i) сачува у векторима d и $prethodnik$ на месту које одговара чвору i, тј. да побољша решење које је већ постојало. На сликама 2 и 3 приказан је једноставан пример Дијкстриног алгоритма на графу са 7 чворова. Алгоритам проналази најкраћи пут од чвора A до свих осталих чворова у графу.

Доказ исправности. Дајемо и доказ исправности Дијкстриног алгоритма. У доказу се користи до сада подразумевана претпоставка да су цене на свакој од грана ненегативне. У генералном случају, уколико постоји грана са негативном ценом Дијкстрин алгоритам неће дати исправно решење. Тада је потребно тражити општији алгоритам од Дијкстриног. Интуитивно, исправност Дијкстриног алгоритма заснива се на чињеници да најкраћи пут од чвора s до неког чвора v, који се налази у скупу S испитаних чворова у одређеном тренутку времена, пролази само кроз чворове који су већ испитани, тј. смештени у скуп S. Наиме, ако би постојао неки пут који води од s до $v \in S$ преко чвора $e \notin S$, онда би тај пут морао бити дужи од већ пронађеног пута. Наиме, како је v ушао у S пре чвора u, то мора бити $d(u) \geqslant d(v)$, јер је v у тренутку уласка у S био такав чвор за који је $d(v) = \min\{d(j) \mid j \in T\}$, а како је $c_{uv} \geqslant 0$, то је $d(v) \leqslant d(u) + c_{uv}$. Ово управо значи да је ознака d на чвору v мања од било којег пута који води од s до $v \in S$ преко неког чвора $u \notin S$.

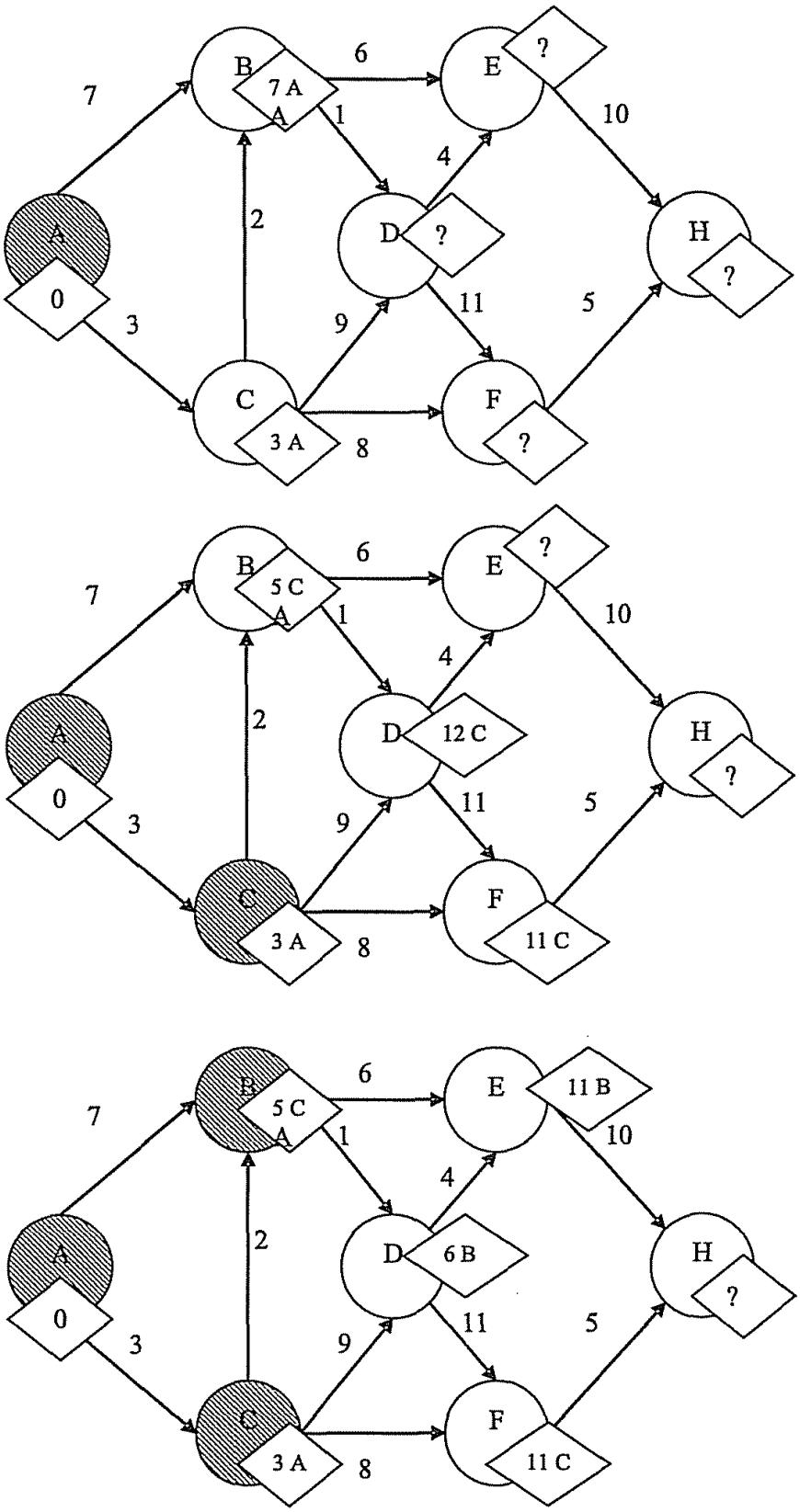
Подсетимо се који су сви кораци неопходни за доказивање исправности неког алгоритма. У „стандардном“ доказу исправности алгоритма треба:

(1) одредити особине алгоритма (инваријанте) које су тачне у свакој итерацији;

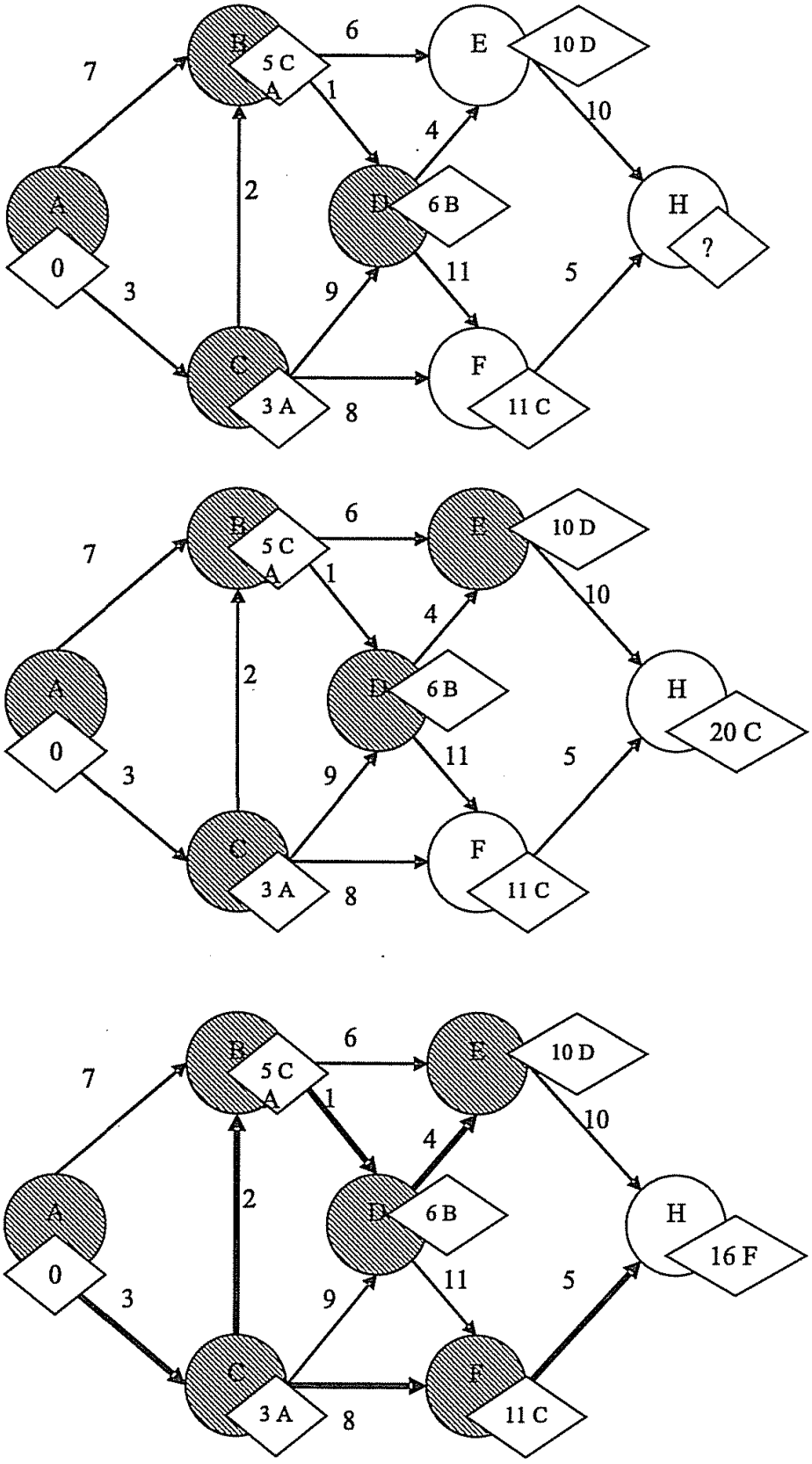
(2) доказати постојање (тачност) инваријанти користећи индукцију;

(3) доказати да је алгоритам коначан;

(4) изабрати инваријанте тако да коначност алгоритма и тачност тих инваријанти имплицирају исправност алгоритма.



Слика 2. Прве три итерације Дијкстриног алгоритма - врхови који су затамњени су ушли у скуп S, а d и претходник за сваки врх се бележе.



Слика 3. Алгоритам се окончава када су сви врхови истражени. Болдиране су гране које чине најкраће путеве.

У стварности, ово је прилично тежак посао, а поготово ставке (1) и (4) захтевају велику досетљивост и углавном почивају на интуицији. Пређимо коначно на извођење доказа исправности Дијкстриног алгоритма.

Теорема. *Ако је дат граф $G = (V, E, c)$, при чему су све компоненте вектора c (цена) ненегативне, и један чвор $s \in V$, који ћемо прогласити за почетни, онда Дијкстрин алгоритам проналази најкраћи пут од чвора s до сваког чвора из $V \setminus \{s\}$.*

Доказ. Користимо следеће инваријанте Дијкстриног алгоритма:

1. $j \in S \Rightarrow d(j)$ је дужина најкраћег пута од чвора s до чвора j;
2. $i \in S$, $j \in T \Rightarrow d(i) \leqslant d(j)$;
3. $j \in T \Rightarrow d(j)$ је дужина најкраћег пута од чвора s до чвора j у графу индукцираном скупом чворова $S \cup \{j\}$.

Сада ћемо инваријанте доказати користећиу принцип математичке индукције.

Основни корак. $S = \{s\}$.

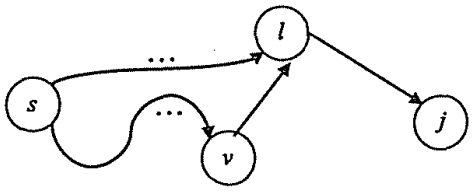
Инваријанта 1. $d(s) = 0$ јесте дужина најкраћег пута од s до s.

Инваријанта 2. d-ознаке на свим осталим чворовима су ненегативне: сваки сусед v од s има d-ознаку једнаку цени c_{sv}, док сви остали чворови имају бесконачну d-ознаку. Зато сигурно важи $(\forall i \in T) d(s) \leqslant d(i)$.

Инваријанта 3. Сваки од графова индуциран скупом $S \cup \{j\}$, где је $j \in T$, има само два чвора, то може бити или једна грана или два изолована чвора. Ако је $j \in T$ сусед од s онда је $d(j) = c_{sj}$, што јесте дужина најкраћег пута у графу $[\{s, j\}]$, а у супротном је $d(j) = \infty$, што можемо схватити (и схватамо) као дужину најкраћег пута између чворова међу којима не постоји пут.

Индуктивни корак. Нека су сада све инваријанте тачне у k-том итеративном кораку ($k \geqslant 1$), при којем у скуп S улази чвор v, пре позива функције *update*. Након нове итерације видимо да инваријанта 2 важи на тривијалан начин: $d(v) = \min\{d(j) \mid j \in T\}$, а по индуктивној претпоставци је $i \in S \setminus \{v\}, j \in T \Rightarrow d(i) \leqslant d(j)$. За инваријанту 1 тачност поново за све $i \in S \setminus \{v\}$ следи на основу индуктивне претпоставке. За новоиспитани чвор v, с друге стране, имамо да је на основу инваријенте 3 индуктивне претпоставке, ознака $d(v)$ најкраћи пут у графу индуцираном чворовима из S и самим чвором v (пре његовог премештања). Зато је довољно показати да најкраћи пут у графу G од s до v не пролази ни преко једног чвора из T (након премештања). А ово (на основу већ поменутог интуитивног разматрања) следи из одабира чвора v. Наиме, $d(v) \leqslant d(i)$ за све $i \in T$, па је тим пре $d(v) \leqslant d(i) + c_{iv}$ (уколико ивица iv постоји - у сваком случају најмања удаљеност од s до v по путу преко чвора i биће мања од најмање удаљености од s до i, јер путује преко барем још једне гране која, још једном напомињемо, има ненегативну цену). То значи да смо нашли најкраћи пут од s до v чим v уђе у S. Управо зато скуп S називамо скупом *испитаних* чворова. Још је потребно само доказати тачност инваријанте 3. Нека је $j \in T$. Означимо са $d(j)$ d-ознаку чвора j пре, а са $d'(j)$ после итерације. Разликоваћемо два случаја. Најкраћи пут у графу $[S]$ (након премештања v) може садржавати или не садржавати чвор v. Уколико најкраћи пут не садржи v, онда је $d(j) = d'(j)$ и како је $d(j)$ била дужина најкраћег пута у $[S \setminus \{v\}]$, биће то и у $[S]$. Но, ако најкраћи пут пролази кроз v, онда у том путу v мора бити

непосредни претходник чвора j. Заиста, ако би неки други чвор l био непосредни претходник, онда би он морао бити у скупу S и пре премештања. Самим тим је $d(l) \leqslant d(v)$, а јасно је и $c_{lj} \leqslant c_{vl} + c_{lj}$. Зато је $d(l) + c_{lj} \leqslant d(v) + c_{vl} + c_{lj}$, па би краћи пут од претпостављеног био онај који води директно преко l у j не пролазећи кроз v. Ова контрадикција нам показује да v мора бити непосредни претходник чвора j. Но, тада функција *update* позвана над чвором v управо маркира ивицу vj и у $d'(j)(< d(j))$ смешта дужину најкраћег пута у $[S]$. Тиме је доказ инваријанти комплетиран.



Слика 4. Случај у коме v није директни претходник чвора j води у контрадикцију.

Сада је довољно приметити да се при свакој итерацији скуп S увећава за један елемент (чвор). Пошто је V коначан скуп чворова на крају мора бити $S = V$. Дакле, алгоритам се завршава након коначног броја итерација - њих тачно $n - 1$, где је $n = |V|$ број чворова у графу. Сада на основу инваријанте 1 следи да ће по окончању алгоритма у вектору d бити садржане све најкраће удаљености од чвора s до свих осталих чворова графа. Тиме је доказ теореме завршен.

Напомена. Из самог вектора d најкраће путеве можемо реконструисати - вратимо се уназад и видимо за сваки врх j за који то врх i важи $d(j) = d(i) + c_{ij}$. Но, ово избегавамо коришћењем вектора $prethodnik$. Кад год се d-ознака на неком врху j промени, промени се и информација о његовом претходнику: на j-ту позицију вектора $prethodnik$ смештамо управо онај непосредни претходник од j преко којег се долази до j тек откривеним побољшаним путем. На тај начин чувамо све гране по којима се треба кретати да би се добили најкраћи путеви у графу G. Ради се заправо о имплицитном маркирању грана. Није тешко показати да све маркиране гране чине једно дрво, које се назива *дрво најкраћих пӯтева.*

Комплексност. Размотримо сада комплексност Дијкстриног алгоритма. Комплексност ће јасно зависити од структуре података у којој се чувају чворови и нарочито је додела „нека је $i \in T$ врх за који је $d(i) = \min\{d(j) \mid j \in T\}$“ отворена за побољшавања уз помоћ згодних структура података. У сваком случају, видимо да нам не треба више времена од $O(n^2)$ за извршавање алгоритма. Наиме, алгоритам захтева $n - 1$ итерација и уколико користимо обичну секвенцијалну претрагу за налажење чвора из T са минималном d-ознаком претпостављамо такође линеарно време претраге. Даље, функција *update* ће укупно (у свим позивима заједно) потрошити време пропорционално броју ивица $m = |E|$, јер ће мењајући аргументе проћи кроз све ивице графа. Дакле, ако су све структуре података вектори, онда је комплексност $O(n^2 + m)$, што је, с обзиром да је m у најгорем случају пропорционално са n^2, управо $O(n^2)$. Чини се да је нешто побољшана она верзија која чворове графа чува у heapu, тј. реду са приоритетом где је поредак на чворовима управо заснован на поретку по њиховим d-ознакама. Тада сва налажења чвора са најмањом ознаком узимају време $O(n \log n)$, али heap додатно компликује функцију *update*. Поправке треба вршити највише m пута као и

раније, али сада свака поправка кошта $O(\log n)$ времена (поправљање heap услова), па је потребно извршити највише $O(m \log n)$ упоређивања у heapu. То даје укупну комплексност од $O((n+m)\log n)$. У случају ретких графова, оваква имплементација јесте боља, али на густим графовима где је $m \sim n^2$ комплексност постаје $O(n^2 \log n)$, па је питање да ли се увођење heapa уопште исплати.

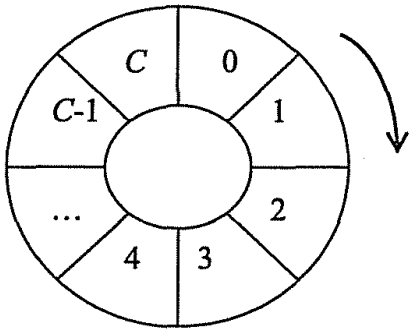
Дијкстрин алгоритам са корпама. Једна друга имплементација Дијкстриног алгоритма такође доноси побољшања, али под одређеним претпоставкама. То је тзв. *Дијкстрин алгоритам са корпама (bucket).* Чворови се уместо у некој стандардној структури података, чувају у наменској структури коју зовемо bucket, тј. корпа. Наиме, ако са C означимо највећу цену на некој грани, тј. $C = \max_{uv \in E} c_{uv}$, тада је највећа дужина најкраћег пута $(n-1)C$. Сада формирамо $(n-1)C+1$ корпи са ознакама од 0 до $(n-1)C$ и додатну корпу са ознаком ∞. Корпе имплементирамо помоћу двоструко повезаних листи. Операција укључења и искључења је у том случају за корпе константна, а ако корпе сместимо у један низ и операција премештања из корпе у корпу биће константна. Ознаке корпи управо ће означавати d-ознаке чворова који су у њима. Ако у корпама у одређеном тренутку времена стоје само чворови из T, онда је јасно да операција налажења чвора са најмањом ознаком заправо представља налажење прве непразне корпе. С обзиром на то да увек радимо са неопадајућим d-ознакама, није тешко видети да је довољан један пролазак кроз низ корпи за извршавање целог алгоритма. Као и раније, за извршавање свих позива функције *update* треба нам време $O(m)$. То нас наводи на закључак да оваква имплементација Дијкстриног алгоритма има комплексност $O(m+nC)$. Поново, оваква комплексност је боља у случају ретких графова, јер за густе графове деградирамо на почетну комплексност $O(n^2)$. Но, то није све. Овакав алгоритам заправо и није полиномијалан алгоритам! Наиме, он зависи од бројног параметра C проблема, који такође може да зависи од величине проблема, тј. од броја чворова графа G и ми немамо начин да ову зависност ограничимо. Није тешко формирати класу проблема у којима је $C \sim n^2$ (видети граф на слици 1) и тада Дијкстрин алгоритам са корпама постаје експоненцијалан, самим тим и потпуно неприхватљив. Овакви алгоритми који су наизглед полиномијални, али зависе од неког бројног параметра проблема, називају се *псеудополиномијални* алгоритми. У томе је смисао исказа да оваква имплементација алгоритма доноси побољшања под одређеним претпоставкама. Наиме, ако је C константна величина за неку класу проблема најкраћег пута на ретким графовима ($m = O(n)$), Дијкстрин алгоритам са корпама је најзгоднији пут решавања ових проблема, јер му је под тим претпоставкама време извршавања линеарно.

Dial имплементација корпи. Још један од проблема са којим се суочавамо при предложеној имплементацији Дијкстриног алгоритма са корпама је и величина меморије која се користи при његовом извршавању. Наиме, видели смо да се као јединствена структура података, у којој се чувају чворови графа, погодна за манипулацију са њима, користи низ од $(n-1)C+1$ двоструко повезаних листи (корпи). Иако су све прилике да велика већина корпи уопште неће бити употребљена, захтеви за резервисање оволико меморијских ресурса и даље у много случајева могу бити превелики и тешко их је практично задовољити. Зато се дошло на идеју оптимизације меморијских ресурса која првенствено користи следеће две особине рада са низом корпи:

(a) у проласку кроз низ корпи увек се крећемо у истом смеру (од мањих ка већим ознакама), што је последица својства да у S улазе чворови са неопадајућим d-ознакама;

(б) ако је актуелан чвор v (управо је ушао у скуп S), онда се чворови који се евентуално премештају при позиву функције $update(v)$ могу сместити само у неку од корпи са ознакама $d(v), d(v)+1, \ldots, d(v)+C$.

Стога је довољно уместо низа од $(n-1)C+1$ корпи чувати само низ од $C+1$ корпи са ознакама $0, 1, \ldots, C$, уз додатну корпу са ознаком ∞, што је велика уштеда (нарочито ако C није јако велики број - већ смо рекли да ако C брзо расте у односу на n, Дијкстрин алгоритам са корпама и нема пуно смисла). Тада једноставно у раду са корпама, тј. при премештању чворова из једне корпе у другу, оперишемо по модулу $C+1$ и кроз низ се крећемо циклично - након проласка кроз корпу C, поново се враћамо на корпу 0. Заправо овакву структуру података можемо замислити као бројчанике на старијим телефонима (одатле назив dial), односно као корпе поређане на кружници. Тада се по корпама крећемо увек у смеру кретања казаљке на сату. На пример, ако је $C = 7$ и у S улази чвор i за који је $d(i) = 11$, онда чвор i вадимо из корпе 3, јер је $11 \equiv 3 \pmod 8$. Када применом функције $update$ над чвором i, откријемо побољшани пут до чвора j за који је било $d(j) = 15$, који води преко чвора i, јер је $c_{ij} = 2$, имамо $d(i) + c_{ij} = 11 + 2 = 13 < 15 = d(j)$, тада стављамо $d(j) = d(i) + c_{ij} = 13$. То значи да чвор j треба преместити из корпе 7 (јер је $15 \equiv 7 \pmod 8$) у корпу 5 (јер је $13 \equiv 5 \pmod 8$). Уколико нема других чворова чије d-ознаке можемо побољшати, настављамо кретање по „кругу“ корпи у смеру кретања казаљке на сату. Приметимо да је стварне d-ознаке чворова нешто теже конструисати из dial низа корпи, па је упутно чувати засебан низ d-ознака, као што смо то чинили у класичној имплементацији Дијкстриног алгоритма.



Слика 5. „Dial“ структура података

Литература

[1] М. Живковић: *Алгоритми*, Математички факултет, Београд, 2000.

[2] S. Skiena: *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica.* Reading, MA: Addison-Wesley, 1990.

[3] A. Lauschke, E.W. Weisstein: *Dijkstra's Algorithm.* From *MathWorld*, `http://mathworld.wolfram.com/DijkstrasAlgorithm.html`

Статијата прв пат е објавена во списанието ТАНГЕНТА на ДМ на Србија во 2009/10 година